

Multiple-Choice Quiz — Week 15

Modern Java · Marathon: cumulative review of Weeks 1–14

This is a marathon session: 25 questions instead of the usual 12, mixing this week's material with a cumulative review of Weeks 1–14.

Question 1. How many abstract methods can a functional interface have?

- ☐ A) 0
- ☐ B) 1
- ☐ C) 2
- ☐ D) Any number

Question 2. What does `new String("cat") == new String("cat")` evaluate to?

- ☐ A) true
- ☐ B) It fails to compile
- ☐ C) false
- ☐ D) It throws an exception

Question 3. Every element of the `args` array passed to `main` has this type.

- ☐ A) `int`
- ☐ B) `Object`
- ☐ C) `char`
- ☐ D) `String`

Question 4. Which functional interface represents a function from a `T` to a `boolean`?

- ☐ A) `Function<T, R>`
- ☐ B) `Predicate<T>`
- ☐ C) `Consumer<T>`
- ☐ D) `Supplier<T>`

Question 5. What happens when Java evaluates `false && (1 / 0 == 0)`?

- ☐ A) It evaluates to false
- ☐ B) It evaluates to true
- ☐ C) It throws an `ArithmeticException`
- ☐ D) It fails to compile

Question 6. True or false: a Java array's length can be changed after it is created.

- ☐ A) True
- ☐ B) False

Question 7. Which call turns `List<String> names` into a `Stream<String>`?

- ☐ A) `Stream.of(names)`
- ☐ B) `names.toStream()`
- ☐ C) `new Stream(names)`
- ☐ D) `names.stream()`

Question 8. Which method reads a single line of text typed by the user, via `StdIn`?

- ☐ A) `StdIn.readInt()`
- ☐ B) `StdIn.readChar()`
- ☐ C) `StdIn.readLine()`
- ☐ D) `StdIn.readAll()`

Question 9. What is `(int x, int y)` in a method declaration called?

- ☐ A) The method signature
- ☐ B) The return type
- ☐ C) The method body
- ☐ D) The formal parameter list

Question 10. What does the stream operation `filter(n -> n >= 0)` do?

- ☐ A) Computes the absolute value of each element
- ☐ B) Removes negative elements

- ☐ C) Counts the matching elements
- ☐ D) Sorts elements in ascending order

Question 11. Which method from the Math class returns the larger of two values?

- ☐ A) Math.max(a, b)
- ☐ B) Math.larger(a, b)
- ☐ C) Math.top(a, b)
- ☐ D) Math.compare(a, b)

Question 12. countdown(n) prints n and recurses on n - 1 until n == 0. How many numbers does countdown(3) print?

- ☐ A) 3
- ☐ B) 4
- ☐ C) 2
- ☐ D) 0

Question 13. What does List.of(1, 2, 3).stream().map(n -> n * 2).toList() evaluate to?

- ☐ A) [1, 2, 3]
- ☐ B) [1, 4, 9]
- ☐ C) [2, 4, 6]
- ☐ D) 6

Question 14. Every recursive solution can also be written using this control structure.

- ☐ A) A class
- ☐ B) An interface
- ☐ C) An array
- ☐ D) A loop

Question 15. Which visibility modifier enforces encapsulation for an object's fields?

- ☐ A) public
- ☐ B) static
- ☐ C) private
- ☐ D) final

Question 16. Which of these is a **terminal** stream operation, ending the pipeline?

- ☐ A) filter
- ☐ B) map
- ☐ C) count
- ☐ D) sorted

Question 17. Defining multiple constructors with different parameter lists is called this.

- ☐ A) Constructor overloading
- ☐ B) Constructor overriding
- ☐ C) Constructor inheritance
- ☐ D) Constructor aliasing

Question 18. True or false: a Java class can implement multiple interfaces at once.

- ☐ A) True
- ☐ B) False

Question 19. What problem does returning Optional<T> instead of a possibly-null T solve?

- ☐ A) It makes a value's absence explicit in the type
- ☐ B) It makes the code run faster
- ☐ C) It prevents all exceptions from being thrown
- ☐ D) It automatically supplies a default value

Question 20. Which keyword lets a class inherit from exactly one other class?

- ☐ A) implements
- ☐ B) interface
- ☐ C) super
- ☐ D) extends

Question 21. True or false: at runtime, a List<String> is distinguishable from a List<Integer>.

- ☐ A) True
- ☐ B) False

Question 22. Besides a constructor and getters, what else does a record automatically generate?

- ☐ A) Setters for every field
- ☐ B) A main method
- ☐ C) A clone() method
- ☐ D) equals(), hashCode(), and toString()

Question 23. What style of programming does JavaFX use?

- ☐ A) Event-driven programming
- ☐ B) Functional programming
- ☐ C) Recursive programming
- ☐ D) Declarative programming

Question 24. True or false: finally still runs even if an exception is never caught.

- ☐ A) True
- ☐ B) False

Question 25. Why does var x = null; fail to compile?

- ☐ A) var cannot be used inside main
- ☐ B) Java cannot infer a type from null alone
- ☐ C) null is not a valid Java value
- ☐ D) var only works with primitive types

Answer Key — Week 15 Quiz

Concept tested in *italics*.

- Q1 **B** *Functional interfaces*. A functional interface has **exactly one** abstract method, which is what lets a lambda implement it. 0 describes a marker interface; 2 or more makes Java unable to tell which method the lambda implements.
- Q2 **C** *String equality (Week 2)*. `==` compares references, not contents, and each new `String(...)` call allocates a fresh object — so the two sides are distinct objects even though they hold the same characters, making the comparison false. Comparing contents requires `.equals()`; the code compiles fine and throws nothing.
- Q3 **D** *Command-line arguments (Week 2)*. Every element of `args` is a `String`, regardless of what it looks like — parsing it into an `int` or other type requires an explicit conversion.
- Q4 **B** *Built-in functional interfaces*. `Predicate<T>` takes a `T` and returns `boolean`. `Function<T, R>` returns an `R`, `Consumer<T>` returns nothing, and `Supplier<T>` takes no input.
- Q5 **A** *Short-circuit evaluation (Week 3)*. `&&` only evaluates its right-hand side if the left-hand side is `true`; since it's `false` here, `1 / 0` is never reached and no exception is thrown.
- Q6 **B** *Fixed array length (Week 4)*. `false` — a Java array's length is fixed at creation and can never change. Growing a collection means creating a new, bigger array (or using a resizable type like `ArrayList`).
- Q7 **D** *Creating a stream*. `stream()` is the method every `Collection` provides to start a stream pipeline. `Stream.of(names)` would wrap the list itself as a single element, and the other two options don't exist.
- Q8 **C** *Reading input (Week 5)*. `StdIn.readLine()` reads one full line of text. `readInt()` and `readChar()` read a single value of their respective type, and `readAll()` reads everything remaining at once.
- Q9 **D** *Function terminology (Week 6)*. `(int x, int y)` is the **formal parameter list**. The **signature** is the broader combination of name and parameter list; the return type and body are separate parts of the declaration.
- Q10 **B** *The filter operation*. `filter` keeps only the elements for which the given predicate returns `true`, discarding the rest; it does not transform, count, or sort anything.
- Q11 **A** *The Math library (Week 6)*. `Math.max(a, b)` returns the larger of its two arguments. The other three are not real `Math` methods.
- Q12 **B** *Tracing recursion (Week 7)*. `countdown(3)` prints 3, then 2, then 1, then finally 0 when the base case (`n == 0`) is reached — four numbers in total, since the base case still prints before stopping. 3 forgets that the base case also prints; 2 and 0 undercount further.
- Q13 **C** *The map operation*. `map` applies the given function to every element, doubling each of 1, 2, 3 to produce [2, 4, 6]. `toList()` then collects the transformed stream into a list.
- Q14 **D** *Recursion vs. iteration (Week 7)*. Any recursive algorithm can be rewritten as an equivalent loop (and vice versa) — the two are interchangeable in what they can compute, just structured differently.

- Q15 C** *Encapsulation (Week 9)*. Fields are conventionally declared `private` so that outside code cannot read or modify them directly, forcing access through the class's own methods (like getters and setters).
- Q16 C** *Terminal vs. intermediate operations*. `count` produces a result and ends the pipeline, so it's terminal. `filter`, `map`, and `sorted` all return a new stream and are intermediate.
- Q17 A** *Constructor overloading (Week 9)*. Just like methods, a class can declare several constructors as long as their parameter lists differ — this is constructor overloading. Overriding, inheritance, and aliasing are unrelated mechanisms.
- Q18 A** *Multiple interfaces (Week 10)*. True — unlike class inheritance (`extends`, single only), a class can implement any number of interfaces at once, separated by commas.
- Q19 A** *Why Optional*. `Optional<T>`'s type itself signals that a value may be absent, forcing callers to handle that case explicitly — unlike a plain `T` that might silently be `null`. It has nothing to do with speed, and it doesn't prevent unrelated exceptions or invent default values on its own.
- Q20 D** *The extends keyword (Week 10)*. `extends` is used for subclassing exactly one other class. `implements` is for interfaces (any number), `interface` declares an interface type, and `super` refers to the superclass from inside a method, not a declaration keyword.
- Q21 B** *Type erasure (Week 12)*. False — Java erases generic type information at compile time, so at runtime a `List<String>` and a `List<Integer>` are indistinguishable; both are just a `List` of Objects underneath.
- Q22 D** *What records generate*. A record automatically provides `equals()`, `hashCode()`, and `toString()` alongside its constructor and getters. It has no setters (records are immutable), no `main` method, and no `clone()`.
- Q23 A** *Event-driven programming (Week 14)*. JavaFX programs wait for and respond to events like clicks or key presses, rather than executing a fixed sequence of statements top to bottom. Functional and declarative programming describe unrelated style distinctions covered elsewhere this week, and “recursive programming” is not a real paradigm name.
- Q24 A** *The finally block (Week 13)*. True — code in `finally` always runs after the `try` and any `catch` blocks, whether or not an exception was thrown, and whether or not it was caught.
- Q25 B** *Type inference limits*. `var` infers a variable's type from its initializer, but `null` has no type of its own to infer from, so an explicit type annotation like `String x = null;` is required instead.