Multiple-Choice Quiz — Week 9

Using and Creating Data Types · for reference: Weeks 5–7

Question 1. Which keyword must be used to create a new instance of a class in Java?

- ☐ **A)** `class`
- ☐ **B)** `new`
- ☐ **C)** `this`
- ☐ **D)** `static`

Question 2. What is the primary role of a constructor in a Java class?

- ☐ **A)** To return a value from a method
- ☐ **B)** To initialize a newly created object's fields
- ☐ **C)** To destroy an object once it is no longer needed
- ☐ **D)** To decide which methods are `private`

Question 3. A `Player` constructor takes a parameter named `score` and its body contains exactly the statement `score = score;`. What happens to the object's `score` field?

- ☐ **A)** It is set correctly to the parameter's value
- ☐ **B)** It stays at its default value
- ☐ **C)** The program fails to compile
- ☐ **D)** It throws a `NullPointerException` at runtime

Question 4. A class has a `private double balance;` field. Which line correctly reads its value from **outside** the class?

- ☐ **A)** `account.balance`
- ☐ **B)** `account.getBalance()`
- ☐ **C)** `Account.balance`
- ☐ **D)** `balance.get()`

Question 5. What distinguishes a `static` field from an instance field?

- ☐ **A)** A `static` field belongs to the class and is shared by all objects
- ☐ **B)** A `static` field can only store numbers; an instance field can store any type
- ☐ **C)** A `static` field is always `private`; an instance field is always `public`
- ☐ **D)** There is no difference — the terms are interchangeable

Question 6. If one object updates a static field's value, other objects of the same class:

- ☐ **A)** See the same, updated value
- ☐ **B)** Keep their own separate, unchanged value
- ☐ **C)** Get a compile error
- ☐ **D)** See `null` until they are recreated

Question 7. Why might you use an `enum` instead of plain `int` constants to represent a fixed set of values, such as days of the week?

- ☐ **A)** Enums run faster than integers at execution time
- ☐ **B)** Enums restrict a variable to only the valid, named values, improving type-safety and readability
- ☐ **C)** Enums allow unlimited new values to be added while the program is running
- ☐ **D)** Enums automatically sort their values alphabetically

Question 8. Two separate `Player` objects are created with identical stats. What does `player1 == player2` evaluate to?

- ☐ **A)** `true`
- ☐ **B)** `false`

□ **C)** It depends on which fields are `private`

□ **D)** It throws a compile error

Question 9. What typically happens if a recursive method is written without a base case?

□ **A)** It runs once and returns `null`

□ **B)** It compiles but throws a checked exception immediately

□ **C)** It keeps calling itself indefinitely, usually crashing with a `StackOverflowError`

□ **D)** Java automatically stops it after 100 calls

Question 10. What is it called when a class defines two methods with the same name but different parameter lists?

□ **A)** Recursion

□ **B)** Overloading

□ **C)** Overriding

□ **D)** Aliasing

Question 11. A method receives an `int` parameter and changes its value inside the method body. What happens to the variable back in the caller?

□ **A)** It also changes

□ **B)** It stays the same

□ **C)** It becomes `0`

□ **D)** The program fails to compile

Question 12. Which class from the course's standard library provides formatted printing methods like `printf`?

□ **A)** `StdIn`

□ **B)** `StdOut`

□ **C)** `StdDraw`

□ **D)** `StdRandom`

Answer Key — Week 9 Quiz

Concept tested in *italics*.

Q1 **B** *Object creation.* The `new` operator allocates a fresh object and invokes its constructor. `class` declares a type, `this` refers to the current object, and `static` marks a class-level member — none of them create instances.

Q2 **B** *Purpose of a constructor.* A constructor initializes the fields of a newly created object. Java has no explicit destructor (objects are garbage-collected), methods return values via `return`, and access modifiers are declared independently of constructors.

Q3 **B** *Why `this` matters.* Without `this`, `score = score;` assigns the parameter to itself and leaves the field at its default (`0`); it compiles fine and never touches `null`, so C and D are both wrong.

Q4 **B** *Encapsulation.* `balance` is `private`, so it can only be read from inside the class. External code must go through a public getter like `getBalance()`; the other options either access the field directly (compile error) or call a method that doesn't exist on the field.

Q5 **A** *Static vs. instance fields.* A `static` field is shared by the whole class — one copy for everyone — while each object gets its own independent copy of an instance field. Access level and type restrictions are unrelated to `static`.

Q6 **A** *Static fields are shared.* A static field belongs to the class itself, not to any one object, so there is only ever one copy. Updating it via one object updates that same copy, so every other object immediately sees the new value too.

Q7 **B** *Why enums.* Enums give a variable type-safety by restricting it to a fixed, named set of values, which also makes code more readable than raw integers. They are not about performance, unlimited growth, or ordering.

Q8 **B** *Reference equality.* For objects, == compares references (memory addresses), not field contents — even with identical stats, two separately created objects are not the same object. Access modifiers play no role here, and the expression compiles fine.

Q9 **C** *Recursion review (Week 7).* Without a base case, a recursive method keeps calling itself, and each call adds a frame to the call stack until it overflows, raising a `StackOverflowError`. There is no automatic call limit and no exception thrown up front.

Q10 **B** *Overloading review (Week 6).* Overloading is defining multiple methods with the same name but different parameter lists (arity or type). Overriding is a different, unrelated mechanism; recursion and aliasing describe unrelated concepts entirely.

Q11 **B** *Pass-by-value review (Week 6).* Java always passes primitives by value: the method receives a copy of the `int`, so changes inside it never propagate back to the caller's variable. Reference-passing for primitives is a common misconception this question targets directly.

Q12 **B** *Standard library review (Week 5).* `StdOut` is the course library's class for console output, including `printf`-style formatted printing. `StdIn` reads input, `StdDraw` draws graphics, and `StdRandom` generates random values.