Multiple-Choice Quiz — Week 6

Functions and Libraries · for reference: Weeks 3–5

Question 1. Which return type must a method use if it does not return any value?

- ☐ **A)** `void`
- ☐ **B)** `null`
- ☐ **C)** `none`
- ☐ **D)** `empty`

Question 2. A function that always produces the same output for the same input and has no side effects is called what?

- ☐ **A)** A recursive function
- ☐ **B)** A pure function
- ☐ **C)** An overloaded function
- ☐ **D)** A static function

Question 3. Which keyword exits a method immediately?

- ☐ **A)** `break`
- ☐ **B)** `exit`
- ☐ **C)** `return`
- ☐ **D)** `stop`

Question 4. What can Java use to tell two overloaded methods with the same name apart?

- ☐ **A)** Their return type
- ☐ **B)** The number and/or types of their parameters
- ☐ **C)** The names of their parameters
- ☐ **D)** The order in which they are declared

Question 5. When a Java array is passed as an argument to a method, what is actually passed?

- ☐ **A)** A full copy of the array
- ☐ **B)** A reference pointing to the same array in memory
- ☐ **C)** Only the first element of the array
- ☐ **D)** Nothing — arrays cannot be passed to methods

Question 6. A method reassigns its array parameter to a brand-new array. What does the caller's original array look like afterward?

- ☐ **A)** It becomes the new array
- ☐ **B)** It is unchanged
- ☐ **C)** It becomes `null`
- ☐ **D)** The code does not compile

Question 7. In a method declaration such as `static int add(int x, int y)`, what is `(int x, int y)` called?

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

Question 8. What is an API?

- ☐ **A)** The code that implements a library
- ☐ **B)** The rules for what a library provides and how to use it
- ☐ **C)** The program that uses a library
- ☐ **D)** The compiled bytecode of a library

Question 9. What does the > symbol do when running a program from the terminal?

□ **A)** Sends the program's output into a file, overwriting it
□ **B)** Reads input for the program from a file
□ **C)** Sends the program's output into another program
□ **D)** Compares two files

Question 10. Which Java Standard Library call sorts the elements of `int[] a` in ascending order?

□ **A)** `Arrays.sort(a)`
□ **B)** `a.sort()`
□ **C)** `Collections.sort(a)`
□ **D)** `StdArray.sort(a)`

Question 11. What is the main relationship between a `while`-loop and a `for`-loop in Java?

□ **A)** A `while`-loop can never be infinite, but a `for`-loop can
□ **B)** A `for`-loop cannot contain an `if`-statement inside it
□ **C)** They are interchangeable — a `for`-loop is convenient syntax for a common `while`-loop pattern
□ **D)** A `while`-loop must always count up, while a `for`-loop can count in any direction

Question 12. What is the effect of the statement `count--;`?

□ **A)** It is equivalent to `count = count - 1;`
□ **B)** It is equivalent to `count = -count;`
□ **C)** It resets `count` to `0`
□ **D)** It is equivalent to `count = count + 1;`

Answer Key — Week 6 Quiz

Concept tested in *italics*.

Q1 **A** *Void return type.* A method that returns no value must be declared `void`. `null` is a value reference types can hold, not a return type; `none` and `empty` are not Java keywords.

Q2 **B** *Pure functions.* A pure function is referentially transparent: same input, same output, no side effects. `static` only describes how the method is called, not its purity; overloading and recursion are unrelated properties.

Q3 **C** *Exiting a method.* `return` exits a method immediately. `break` only exits a loop or `switch`; `exit` and `stop` are not Java keywords.

Q4 **B** *Overloading resolution.* Java tells overloaded methods apart only by the number and/or types of their parameters — not by return type, parameter names, or the order they are declared in.

Q5 **B** *Argument passing for reference types.* Arrays are reference types, so passing one copies the reference (a pointer to the array), not the underlying data. This is why mutating the array inside the method is visible to the caller.

Q6 **B** *Reassignment vs. mutation.* Reassigning the local parameter variable only changes what that parameter points to inside the method; the caller's variable still points to the original array. Only mutating the array's contents (e.g. `arr[0] = ...`) is visible outside.

Q7 **C** *Function terminology.* `(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.

Q8 **B** *What an API is.* The API is the set of rules describing what a library provides and how to use it — not the code behind it (the implementation) nor the program using it (the client).

Q9 **A** *Redirection (Week 5).* `>` sends a program's standard output into a file, overwriting its contents. Reading input from a file is what `<` does; sending output to another program is what a pipe (`|`) does; comparing two files is unrelated to redirection.

Q10 **A** *The `Arrays` class (Week 4).* `Arrays.sort` is the Standard Library method for sorting primitive arrays. Arrays have no `sort()` instance method, `Collections.sort` works on Lists, and `StdArray` is not a real class.

Q11 **C** *For vs. while (Week 3).* Every `for`-loop can be rewritten as an equivalent `while`-loop; `for` is just convenient syntax for the common counting pattern. Both loop kinds can be infinite or finite, contain any statement, and count in any direction.

Q12 **A** *Decrement operator (Week 3).* `count--` is shorthand for `count = count - 1`. It does not reset `count` to zero, and it is the opposite of `count++` (`count = count + 1`), not the same thing.