Object Oriented Programming Language - Short & Simple Answers

----

Q1.

a) Why should the main method always be public and static in Java?
- public: so the Java Virtual Machine (JVM) can call it from outside the class.
- static: JVM calls main without creating any object, so main must belong to the class, not an instance.
- signature: public static void main(String[] args) is the standard entry point.

b) Describe the compilation process of Java code (simple steps).
1. You write source code in a file with .java extension.
2. The Java compiler (javac) compiles .java to bytecode in .class files.
3. The class loader of the JVM loads the .class files.
4. The bytecode verifier checks code safety (basic correctness/security checks).
5. The JVM executes the bytecode (interprets or uses JIT to convert to native code).

(Short version: source -> compiler -> bytecode -> JVM runs it.)

c) What are the characteristics of Object Oriented Programming (OOP)?
- Encapsulation: bundle data and methods; hide internal details.
- Inheritance: create new classes from existing ones (reuse code).
- Polymorphism: same operation behaves differently for different objects (method overriding/overloading).
- Abstraction: expose only necessary features, hide complexity.
- Modularity & Reusability: build software from reusable parts.

Q2.

a) Explain method overloading with examples.
- Overloading: same method name, different parameter lists (different type/number/order).

Example:
```
void print(int x) { ... }
void print(String s) { ... }
void print(int x, int y) { ... }
```
The compiler picks the right method by matching arguments.

b) Differentiate between constructor and method.
- Constructor:
  * Same name as the class.
  * No return type (not even void).
  * Called automatically when an object is created (new ClassName()).
  * Used to initialize instance variables.
- Method:
  * Has its own name (can be anything) and a return type (void, int, String, etc.).
  * Called explicitly on an object or class (if static).
  * Performs actions/behaviour; not used for constructing the object.

c) Determine True/False. If false, give correct answer.

I. Java is platform dependent.
- False. Correct: Java is platform independent (because bytecode runs on any JVM).

II. In Java the size of char variable is 1 byte.
- False. Correct: char in Java is 2 bytes (16-bit, UTF-16).

III. We access a static variable from a non-static method.
- True. Non-static methods can access static variables directly.

IV. For local variables initialization is not mandatory before use.
- False. Correct: Local variables must be initialized before use (compiler enforces this).

V. Instance variable can be initialized using constructor.
- True. Constructors are commonly used to initialize instance variables.

Q3.
a) Create a class named Test. (Simple code and usage — explained in plain words)
- We have one instance variable A and one static variable B (no initial values in declaration).
- We'll initialize them in the constructor.
- Method m1(): adds A and B and stores result back into B, then prints A and B.
- Method m2(int x, String s): prints the two parameters.
- From main(): call m1() twice and then call m2() once.

Java code:

```
public class Test {
    int A;              // instance variable
    static int B;       // static variable

    // Constructor to initialize A and B
    public Test() {
        A = 1;  // you can change initialization as needed
        B = 2;
    }

    void m1() {
        B = A + B;
        System.out.println(A + " " + B);
    }

    void m2(int x, String s) {
        System.out.println(x + " " + s);
    }

    public static void main(String[] args) {
        Test t = new Test();
        t.m1();   // first call
        t.m1();   // second call
        t.m2(5, "Hi"); // one call to m2
    }
}
```

b) Output of the Test class (with A=1 and B=2 as initialized above):

```
1 3
1 4
5 Hi
```

(Explanation: First m1: B = 1+2 = 3 -> prints 1 3.
Second m1: B = 1+3 = 4 -> prints 1 4.
m2 prints the arguments passed.)

Q4.
a) What is a wrapper class?
- A wrapper class is an object representation of a primitive type. Examples:
  Integer for int, Double for double, Character for char, Boolean for boolean.
- Wrappers allow primitives to be used where objects are required and provide helper methods.
- Boxing: converting primitive to wrapper; Unboxing: wrapper back to primitive.

b) Differences between String and StringBuffer (short & clear):
- String is immutable: once created, its value cannot change. Any operation that seems to change it creates a new String.
- StringBuffer is mutable: it can be changed in-place (append, insert, delete). It is synchronize
d (thread-safe).

- StringBuilder is similar to StringBuffer but not synchronized (faster in single-threaded use).
- Use String for constant text, StringBuffer/StringBuilder for lots of modifications.

c) Output of the given program (class A):

Code summary:

```
String str1 = "NEUB ", str2 = "NEUB ";
String s1 = new String("Spring ");
String s2 = new String("Spring ");
System.out.println(str1);
str1.concat(str2);
System.out.println(str1);
if (s1 == s2) ...   // compares references
if (str2 == str1) ... // compares references (string pool)
StringBuffer sb1 = new StringBuffer("Fall ");
StringBuffer sb2 = new StringBuffer("Fall ");
System.out.println(sb1);
sb1.append(sb2);
System.out.println(sb1);
```

Expected output (line by line):

```
NEUB
NEUB
not equal
equal
Fall
Fall Fall
```

Short explanations:

- concat returns a new String, it does not change str1, so str1 stays "NEUB ".
- s1 and s2 are two different objects (new String(...)), so s1 == s2 is false.
- str1 and str2 are string literals interned in the pool, so they reference the same object; st
r1 == str2 is true.
- StringBuffer's append changes the buffer; after appending sb2, sb1 becomes "Fall Fall ".

End of answers.