**1.Why is Java Platform Independent?**

**Correct Explanation:**

* Java source code (.java) is compiled by the javac **compiler** into **bytecode** (.class file), not directly into machine-specific executable code.
* This bytecode is **platform-independent** and can run on any system that has a **Java Virtual Machine (JVM)**.
* The JVM interprets or compiles the bytecode into native machine code **at runtime**, making Java programs portable across platforms.

**Key Points:**

* In contrast, languages like C/C++ compile directly to platform-specific machine code (.exe), which is not portable.
* Each operating system has its own JVM implementation, but the bytecode remains the same.
*  Java’s **Write Once, Run Anywhere (WORA)** principle is enabled by bytecode and JVM.
*  JVM is **platform-dependent**, but bytecode is **platform-independent**.
*  JIT improves performance by compiling frequently used bytecode to native code.
*  Java is **not 100% platform-independent** because execution depends on the JVM of the target platform.

**2.Java Compilers: javac and JIT**

**Corrected Statements:**

* ✅ javac is the **Java compiler** included in the **JDK (Java Development Kit)**. It compiles .java files into .class bytecode files.

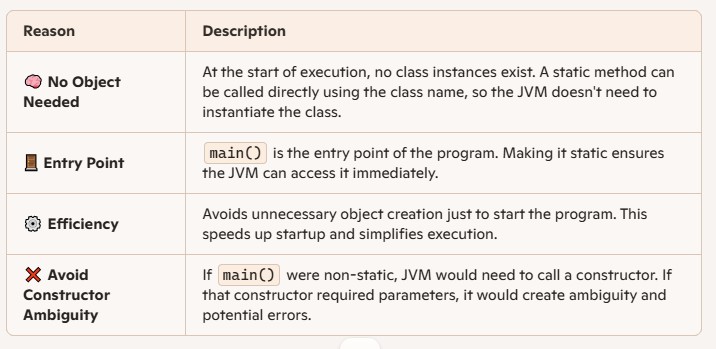
✅ **JJIT (Just-In-Time)** compiler is part of the **JVM**, not a separate compiler. It compiles frequently used bytecode into native machine code **during runtime** to improve performance.

**2. Why Must the Java File Name Match the Public Class Name?**

**Ans:** If a Java file contains a **public class**, the **filename must exactly match the name of that public class**, including capitalization.

### 🧠 Reason Behind This Rule:

* **Compilation Requirement**: The Java compiler (javac) enforces this rule to avoid ambiguity. If the public class is named MyApp, the file must be named MyApp.java.
*  **JVM Entry Point Identification**: When running a Java program, the JVM looks for the class containing the main() method. Matching the filename with the public class name helps the JVM locate the correct entry point.
*  **Code Organization & Readability**: It makes code easier to manage, especially in large projects. Developers and IDEs can quickly locate classes based on filenames.
*  **Multiple Classes in One File**: You can have multiple classes in a single .java file, but **only one can be public**, and that public class must match the filename. Other classes can be package-private (no modifier) and don’t affect the filename.



The main() method is static so that the JVM can invoke it without creating an object of the class. At the start of execution, no instances exist, so making main() static allows it to be called directly using the class name. This design simplifies program startup and avoids unnecessary object creation.

**Q: How is Java both compiled and interpreted?**

### Answer:

Java is considered both a compiled and interpreted language because it uses a two-step execution process. First, the Java source code (.java) is compiled by the javac compiler into an intermediate form called **bytecode** (.class file). This bytecode is platform-independent.

Then, during runtime, the **Java Virtual Machine (JVM)** either **interprets** the bytecode line-by-line or uses a **Just-In-Time (JIT) compiler** to convert frequently used bytecode into native machine code for faster execution.

This hybrid model allows Java to be portable across platforms and efficient in performance. The compilation ensures structure and syntax correctness, while interpretation allows dynamic execution and optimization.

