**1. What do you know about Java?**

Java is a **high-level, object-oriented, platform-independent programming language** developed by Sun Microsystems (now owned by Oracle). It is widely used for building web, desktop, and mobile applications, as well as backend services. Java follows the principle of "**write once, run anywhere (WORA)**," meaning the same code can run on any platform with a compatible Java Virtual Machine (JVM).

**2. Why is Java known as Architecture Neutral?**

Java is known as architecture neutral because its applications are compiled into **bytecode**, which is platform-independent. Bytecode is executed by the **Java Virtual Machine (JVM)**, which is available for various hardware and operating systems. This makes Java applications portable across different platforms without recompilation.

**3. What is SDK/JDK?**

* **SDK (Software Development Kit):** A collection of tools required to develop applications for a specific platform.
* **JDK (Java Development Kit):** A subset of the SDK that includes tools specifically for Java development. It contains:
  + **Compiler (javac):** To compile Java code into bytecode.
  + **JVM:** To run the Java application.
  + **Libraries and APIs:** Standard libraries required for development.
  + **Debugger and other tools.**

**4. What is JRE, and how is it different from JDK?**

* **JRE (Java Runtime Environment):** Provides the necessary runtime environment to execute Java applications. It includes the JVM and standard libraries but does not include development tools like a compiler.
* **JDK (Java Development Kit):** Includes the JRE, plus tools like the compiler, debugger, and libraries needed to develop and debug Java programs.

| **Aspect** | **JRE** | **JDK** |
| --- | --- | --- |
| **Purpose** | For running Java programs | For developing and running programs |
| **Includes** | JVM, libraries | JRE + Compiler, Debugger, Tools |

**5. What is bytecode, and how is it different from .exe?**

* **Bytecode:** Intermediate, platform-independent code generated by the Java compiler. It is executed by the JVM.
* **.exe:** Machine-specific binary code, which is directly executed by the operating system.

| **Aspect** | **Bytecode** | **.exe** |
| --- | --- | --- |
| **Platform** | Platform-independent | Platform-specific |
| **Execution** | Requires JVM to run | Directly executed by the OS |
| **Portability** | Highly portable | Not portable |

**6. OOP Principles with Real-time Examples**

The four OOP principles are **Encapsulation, Abstraction, Inheritance, and Polymorphism.**

**Example 1: Banking System**

* **Encapsulation:** Customer details (name, account number, balance) are private and accessible via public methods like getBalance() or deposit().
* **Abstraction:** The user interacts with an ATM without knowing the internal workings of how the withdrawal process works.
* **Inheritance:** A SavingsAccount class inherits common properties (balance, account number) from a BankAccount superclass.
* **Polymorphism:** A calculateInterest() method behaves differently for FixedDepositAccount and SavingsAccount.

**Example 2: Student System**

* **Encapsulation:** A student's grades and personal details are private, accessed only through getter and setter methods.
* **Abstraction:** A student uses a grading portal without understanding how the grades are calculated internally.
* **Inheritance:** A GraduateStudent inherits properties from a generic Student class.
* **Polymorphism:** A calculateScholarship() method calculates differently for Undergraduate and Graduate students.

**7. Why is main() in Java static?**

The main() method is static because:

1. It allows the JVM to invoke the method without creating an object of the class.
2. Static methods belong to the class rather than any instance, enabling the JVM to load the class and start execution.
3. This ensures that the application can be started without requiring an instance, which simplifies the program's entry point.

**8. Java vs C++**

| **Feature** | **Java** | **C++** |
| --- | --- | --- |
| **Platform** | Platform-independent (WORA) | Platform-dependent |
| **Memory Management** | Automatic (Garbage Collection) | Manual (Developers manage memory using pointers) |
| **Syntax Simplicity** | No explicit pointers | Supports pointers |
| **OOP Model** | Purely Object-Oriented | Hybrid (Procedural + Object-Oriented) |
| **Performance** | Slower due to JVM overhead | Faster (compiled to native code) |

**9. Java vs Python**

| **Feature** | **Java** | **Python** |
| --- | --- | --- |
| **Syntax** | Verbose and strict | Simple and concise |
| **Execution Speed** | Faster (compiled to bytecode) | Slower (interpreted language) |
| **Use Cases** | Enterprise apps, Android | AI, ML, Data Science, Scripting |
| **Typing** | Statically typed | Dynamically typed |
| **Community** | Large and mature | Growing rapidly |