15 Important Questions for Java Interview

1. What is Java and why is it platform independent?

Java is a high-level, class-based programming language that follows the object-oriented paradigm. Its platform independence comes from compiling source code into bytecode, which isn't tied to any specific machine. This bytecode runs on the Java Virtual Machine (JVM), which acts as an interpreter and executor. Because JVMs exist for most operating systems, you can write your code once and run it anywhere, making Java a top choice for cross-platform applications.

2. Explain the concept of object-oriented programming.

Object-Oriented Programming (OOP) is a method of structuring software using entities called objects. These objects represent real-world things and encapsulate both data (attributes) and behavior (methods). Java's OOP principles-Encapsulation, Inheritance, Polymorphism, and Abstraction-enable code to be more modular, easier to test, maintain, and scale across projects. OOP encourages reusability, reduces redundancy, and supports clear design thinking.

3. What are the main features of Java?

Java has several standout features: it's platform-independent, object-oriented, robust (handles exceptions and memory issues), secure (restricts low-level access), and supports multithreading for concurrent execution. Additionally, it includes features for portability, automatic garbage collection, and dynamic linking. All of these make Java suitable for developing everything from desktop software to large-scale enterprise applications.

4. What is the Java Virtual Machine (JVM)?

The Java Virtual Machine is the engine that runs Java bytecode on any platform. It converts bytecode into native machine instructions and manages system memory. The JVM handles critical tasks like garbage collection, exception handling, and security enforcement. It also creates an abstraction between compiled code and the underlying hardware, enabling Java's 'write once, run anywhere' capability.

5. What's the difference between JDK, JRE, and JVM?

JDK (Java Development Kit) is the full development package including the compiler, debugger, and tools to create Java applications. JRE (Java Runtime Environment) is a subset of JDK-it contains libraries and the JVM required to run Java applications but not to build them. JVM, on the other hand, is the core that executes the compiled bytecode and provides the runtime environment.

6. Explain Java's garbage collection process.

Java's garbage collection is an automatic memory management mechanism. It identifies objects that are no longer in use and reclaims their memory to avoid memory leaks. Developers don't need to manually delete objects, which reduces bugs and simplifies coding. The garbage collector works in the background and can use various algorithms (like generational GC) to optimize performance.

15 Important Questions for Java Interview

7. What is a class in Java?

A class in Java is a user-defined blueprint that outlines the structure of objects. It defines variables (fields) and functions (methods) that the created object will have. Classes serve as templates, and objects are instantiated based on them. They help organize code and enforce modular, reusable structures in object-oriented design.

8. What is an object in Java?

An object is a real-world entity created from a class. It holds its own data in attributes and executes behaviors through methods defined in the class. For instance, a Car class might have objects like myCar or friendCar, each with its own properties (like color or speed). Objects interact with one another and are the fundamental building blocks of Java applications.

9. What is inheritance in Java?

Inheritance enables a class (subclass) to inherit properties and behaviors from another class (superclass). This promotes code reuse and supports hierarchical classifications. For example, if Vehicle is a superclass, then Car and Bike can inherit its attributes and override its methods if needed. Java supports single inheritance through classes and multiple inheritance via interfaces.

10. What is polymorphism in Java?

Polymorphism allows the same method or object to behave differently based on context. In Java, it occurs in two main forms: method overloading (compile-time) and method overriding (runtime). This flexibility enables developers to write cleaner, scalable, and more intuitive code, especially in systems that require similar actions on different object types.

11. What is the difference between == and .equals() in Java?

The == operator compares object references, i.e., whether two variables point to the same object in memory. The .equals() method, however, compares the values within the objects. For example, two different String objects may contain the same text, but == would return false while .equals() would return true. Understanding this distinction is key to avoiding logical bugs.

12. What is the static keyword in Java?

The static keyword signifies that a variable or method belongs to the class itself, rather than to instances of the class. It means there's only one copy of the member, shared across all objects. You can access static methods and fields without creating an object. It's commonly used for utility functions and constants (e.g., Math.PI).

13. What is a constructor in Java and how is it different from a method?

A constructor is a special block used to initialize new objects. It has the same name as the class and no return type. Unlike methods, constructors run automatically when an object is created. Methods are designed

for behaviors and typically require an explicit call. Constructors can be overloaded but not overridden.

14. What is method overloading in Java?

Method overloading allows multiple methods in a class to share the same name as long as they differ in parameters (type, number, or order). It enhances readability and usability. For instance, you could have a print() method that works with strings, integers, or booleans. It's a form of compile-time polymorphism.

15. What is method overriding in Java?

Method overriding occurs when a subclass redefines a method from its superclass to provide specialized behavior. The method signature must be identical. It's a form of runtime polymorphism, allowing dynamic method dispatch. This is essential for implementing behaviors specific to derived classes while maintaining a consistent interface.