

## Part 1: Introduction to Java

1. What is Java? Explain its significance in modern software development.

Java is a high-level, object-oriented programming language known for its platform independence and security. It plays a vital role in modern software development by enabling the creation of secure, scalable, and portable applications across various domains like web, mobile, enterprise, and cloud computing.

2. List and explain the key features of Java.

Key features of Java include:

- Platform Independence: Write once, run anywhere (WORA) capability.
- Object-Oriented: Follows OOP principles like inheritance, encapsulation, and polymorphism.
- Robust and Secure: Strong memory management and security features.
- Multi-threading: Supports concurrent execution of tasks.
- High Performance: Uses Just-In-Time (JIT) compiler for optimization.

3. Difference between compiled and interpreted languages; Where does Java fit in?

Compiled languages translate code into machine language before execution, while interpreted languages execute code line by line. Java is both compiled and interpreted; it is first compiled into bytecode and then interpreted by the JVM.

4. Explain platform independence in Java.

Java programs run on any operating system without modification because they are compiled into bytecode, which the JVM interprets, making Java platform-independent.

5. Applications of Java in the real world.

Java is used in:

- Web applications (Spring, Struts, JSF)
- Mobile apps (Android development)
- Enterprise applications (Banking, E-commerce)
- Cloud computing and Big Data
- Embedded systems and IoT

## Part 2: History of Java

1. Who developed Java and when was it introduced?

Java was developed by James Gosling and his team at Sun Microsystems in 1995.

2. What was Java initially called? Why was its name changed?

It was initially called "Oak" but was later renamed to Java due to trademark issues and its resemblance to coffee (Java coffee).

3. Evolution of Java versions.

Java has evolved from JDK 1.0 to the latest versions, introducing features like generics (Java 5), lambda expressions (Java 8), modularization (Java 9), and modern enhancements like records and pattern matching.

#### 4. Major improvements in recent Java versions.

- Java 8: Streams, Lambda expressions
- Java 11: HTTP Client API, garbage collection improvements
- Java 17: Sealed classes, pattern matching
- Java 21: Virtual threads, record patterns

#### 5. Comparison of Java with C++ and Python.

- Java vs. C++: Java is simpler, has automatic garbage collection, and lacks pointers.
- Java vs. Python: Java is faster, statically typed, and better suited for enterprise applications, while Python is dynamically typed and preferred for rapid development.

### Part 3: Data Types in Java

#### 1. Importance of data types in Java.

Data types define the type of data a variable can hold, ensuring memory efficiency and type safety.

#### 2. Primitive vs. Non-primitive data types.

Primitive types (int, char, boolean) store values, whereas non-primitive types (String, arrays, classes) store references.

#### 3. Eight primitive data types.

- byte, short, int, long (Integer types)
- float, double (Floating-point types)
- char (Character type)
- boolean (Boolean type)

#### 4. Example of declaring and initializing data types.

```
int num = 10; double pi = 3.14; char letter = 'A'; boolean isJavaFun = true;
```

#### 5. Type casting with an example.

```
int x = 100; double y = x; // Implicit casting  
float f = (float) 10.5; // Explicit casting
```

#### 6. Wrapper classes.

Wrapper classes (Integer, Double, Boolean) allow primitive data to be used as objects.

#### 7. Static vs. Dynamic typing.

Java is statically typed, meaning variables are defined with a fixed type at compile time.

### Coding Questions on Data Types

#### 1. Java program to declare and initialize all primitive data types.

```
class PrimitiveDataTypes {
```

```

public static void main(String[] args) {
    byte b = 127; short s = 32000; int i = 100000; long l = 10000000000L;
    float f = 10.5f; double d = 99.99; char c = 'J'; boolean bool = true;
    System.out.println(b + ", " + s + ", " + i + ", " + l + ", " + f + ", " + d + ", " + c + ", " + bool);
}
}

```

#### Part 4: Java Development Kit (JDK)

##### 1. What is JDK? How does it differ from JRE and JVM?

JDK (Java Development Kit) includes JRE (Java Runtime Environment) and development tools.

JRE provides libraries and JVM, while JVM executes Java bytecode.

##### 2. Main components of JDK.

- Compiler (javac)
- Java Virtual Machine (JVM)
- Java Runtime Environment (JRE)
- Debugger (jdb)
- Java libraries

##### 3. Steps to install JDK and configure Java.

- Download JDK from Oracle/OpenJDK
- Install and set up environment variables (PATH and CLASSPATH)

##### 4. Simple Java program to print "Hello, World!"

```

class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

```

##### 5. Significance of PATH and CLASSPATH.

PATH helps locate Java commands, while CLASSPATH is used to load class files.

##### 6. OpenJDK vs. Oracle JDK.

OpenJDK is open-source, while Oracle JDK has commercial licensing for enterprise use.

##### 7. How Java programs are compiled and executed.

Java source code (.java) -> Compilation (javac) -> Bytecode (.class) -> Execution (JVM)

##### 8. Just-In-Time (JIT) Compilation.

JIT optimizes Java performance by converting bytecode into native code at runtime.

##### 9. Role of JVM in execution.

JVM loads, verifies, and executes Java bytecode, ensuring platform independence.