Java Programming Basics

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 developed by Sun Microsystems (now owned by Oracle).
- It is widely used for enterprise applications, mobile apps (Android), web applications, and cloud computing.
- It is platform-independent due to its "Write Once, Run Anywhere" (WORA) feature.
- Java is known for its security, robustness, and scalability, making it ideal for large-scale applications.

2. List and explain the key features of Java.

Key Features of Java:

- Platform Independent – Runs on any OS via JVM.
- Object-Oriented – Supports OOP concepts (Encapsulation, Inheritance, Polymorphism).
- Robust – Strong memory management, exception handling, and garbage collection.
- Secure – Features like bytecode verification and runtime security.
- Multithreaded – Supports concurrent execution for better performance.
- High Performance – JIT compiler optimizes bytecode execution.
- Distributed Computing – Supports networking and RMI for distributed applications.

3. What is the difference between compiled and interpreted languages? Where does Java fit in?

Compiled Languages (e.g., C, C++) – Converted to machine code before execution.

Interpreted Languages (e.g., Python, JavaScript) – Executed line-by-line at runtime.

Java's Position – Java is both **compiled** (into bytecode) and **interpreted** (by JVM).

4. Explain the concept of platform independence in Java.

Platform Independence in Java

- Java code is compiled into bytecode (.class file).
- JVM (Java Virtual Machine) interprets the bytecode and runs it on any OS.
- This makes Java **platform-independent** across Windows, Linux, and macOS.

5. What are the various applications of Java in the real world?

Applications of Java

- **Web Development** – Spring, Hibernate frameworks.
- **Mobile Applications** – Android development.
- **Enterprise Applications** – Banking and e-commerce apps.
- **Cloud Computing** – Supports cloud-based applications.
- **Data Science & AI** – Used in ML frameworks like Weka.
- **Embedded Systems** – Java ME for IoT devices.

Part 2: History of Java

1. Who developed Java and when was it introduced?

- Java was developed by **James Gosling** at Sun Microsystems in **1995**.

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

- Initially called **"Oak"** (named after an oak tree outside Gosling's office).
- Renamed to **"Java"** because "Oak" was already a trademark.

3. Evolution of Java Versions

- **Java 1.0 (1995)** – Basic version.
- **Java 5 (2004)** – Introduced generics, enhanced for-loop.
- **Java 8 (2014)** – Introduced lambda expressions, streams.
- **Java 11 (2018)** – Long-term support (LTS) version, performance improvements.
- **Java 17 (2021)** – Latest LTS version, sealed classes, enhanced pattern matching.

4. Major Improvements in Recent Java Versions

- **Lambda Expressions** (Java 8) – Functional programming.
- **Modules (Java 9)** – Better modularity and dependency management.
- **Pattern Matching & Records (Java 14+)** – Improved syntax and performance.
- **Sealed Classes (Java 17)** – Better control over inheritance.

5. Java vs. C++ vs. Python

- **Java vs. C++** – Java has garbage collection, no pointers, platform-independent.
- **Java vs. Python** – Python is dynamically typed and easier for scripting, Java is faster and more scalable.

Part 3: Data Types in Java

1. Importance of Data Types in Java

- Data types define the **type of data** a variable can store.
- Helps in **memory management** and **error prevention**.

2. Primitive vs. Non-Primitive Data Types

- **Primitive (built-in types)** – int, char, double, etc. (stored in stack memory).
- **Non-Primitive (objects & classes)** – String, Arrays, Interfaces (stored in heap memory).

3. Eight Primitive Data Types in Java

1. **byte** – 1 byte, stores -128 to 127.
2. **short** – 2 bytes, stores -32,768 to 32,767.
3. **int** – 4 bytes, stores -2 billion to 2 billion.
4. **long** – 8 bytes, used for large integer values.
5. **float** – 4 bytes, stores decimal numbers (single precision).
6. **double** – 8 bytes, stores decimal numbers (double precision).
7. **char** – 2 bytes, stores a single character.
8. **boolean** – 1 bit, stores true or false.

4. Declaration and Initialization Examples

```
int age = 25;

double salary = 50000.75;

char grade = 'A';

 boolean isPass = true;
```

5. Type Casting in Java

- **Implicit Casting (Widening)** – Converting smaller type to larger type

```
int a = 10;

double b = a; // int to double
```

Explicit Casting (Narrowing) – Converting larger type to smaller type.

```
double x = 10.5;

int y = (int) x; // double to int
```

6. Wrapper Classes in Java

- Converts primitive types to objects (Autoboxing & Unboxing

```
Integer num = Integer.valueOf(10); // Autoboxing
int val = num; // Unboxing
```

7. Static vs. Dynamic Typing

- **Static Typing (Java, C)** – Type checking at compile time, safer and optimized.
- **Dynamic Typing (Python, JavaScript)** – Type checking at runtime, more flexible.

<u>Coding Questions on Data Types</u>

1. **Program to Declare & Initialize All Primitive Data Types**

[redacted]

2. Arithmetic Operations on Two Integers

```java
//2.    Arithmetic Operations on Two Integers
public class ArithmeticOperations {
    public static void main(String[] args) {
        int a = 10, b = 5;
        System.out.println("Addition: " + (a + b));
        System.out.println("Subtraction: " + (a - b));
        System.out.println("Multiplication: " + (a * b));
        System.out.println("Division: " + (a / b));
    }
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> javac PrimitiveTypes.java
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> java PrimitiveTypes
10 20 30 100000 5.75 19.99 A true
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> javac ArithmeticOperations.java
ArithmeticOperations.java:1: error: class, interface, or enum expected
2.      Arithmetic Operations on Two Integers
^
1 error
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> javac ArithmeticOperations.java
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> java ArithmeticOperations
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice>
```

3. Implicit and Explicit Type Casting

```java
public class TypeCasting{
    public static void main(String[] args) {
        int num = 100;
        double d = num; // Implicit (Widening)
        System.out.println(d);

        double x = 99.99;
        int y = (int) x; // Explicit (Narrowing)
        System.out.println(y);
    }
}
/*
Output:
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> javac TypeCasting.java
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> java TypeCasting
100.0
99

*/
```

4. Convert Integer to Double & Vice Versa using Wrapper Classes

```java
public class WrapperDemo {
    public static void main(String[] args) {
        Integer num = 100;
        Double d = num.doubleValue();
        System.out.println(d);

        Double d2 = 50.5;
        Integer num2 = d2.intValue();
        System.out.println(num2);
    }
}

/*
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> javac WrapperDemo.java
PS D:\CDAC\Core_Java_Kiran_Maam\Assignment\Code_Practice> java WrapperDemo
100.0
50
*/
```

Part 4: Java Development Kit (JDK)

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

- **JDK (Java Development Kit):** A complete package required for Java development, including JRE, compilers, and debugging tools.
- **JRE (Java Runtime Environment):** Contains only the runtime components necessary to execute Java applications (JVM + libraries).
- **JVM (Java Virtual Machine):** Responsible for running Java bytecode by converting it into machine code.
- **Difference:**
    - JDK = JRE + Development Tools
    - JRE = JVM + Libraries
    - JVM = Runtime Engine

---

2. Explain the main components of JDK.

1. **Java Compiler (javac):** Converts Java source code into bytecode.

2. **Java Runtime Environment (JRE):** Provides runtime support for Java applications.
3. **Java Virtual Machine (JVM):** Executes Java bytecode.
4. **Java Debugger (jdb):** Helps in debugging Java programs.
5. **Java Archive (JAR) Tool:** Bundles multiple Java files into a single JAR file.
6. **Java Documentation Generator (javadoc):** Generates API documentation.
7. **Additional Tools:** jshell, jconsole, javap (disassembler), etc.

---

3. Steps to Install JDK and Configure Java on Your System

1. **Download JDK** from the official Oracle website or OpenJDK.
2. **Install JDK** following on-screen instructions.
3. **Set Environment Variables:**
   - Add JDK's bin directory to the **PATH** variable.
   - Optionally, set the **CLASSPATH** for external libraries.
4. **Verify Installation:**
   - Open the terminal/command prompt and run:

```
java -version
javac -version
```

---

4. Write a Simple Java Program to Print "Hello, World!" and Explain Its Structure

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

```
}
```

Explanation:

- public class HelloWorld → Declares the class.
- public static void main(String[] args) → Main method (entry point).
- System.out.println("Hello, World!"); → Prints output.

Compile and Run:

```
javac HelloWorld.java
java HelloWorld
```

---

5. Significance of the PATH and CLASSPATH Environment Variables in Java

- **PATH:** Helps the system locate Java binaries like javac and java.
- **CLASSPATH:** Defines the location of Java class files and external libraries.
- **Example Configuration:**

```
set PATH=C:\Program Files\Java\jdk-XX.X.X\bin;%PATH%
set CLASSPATH=C:\myjava\lib\someLibrary.jar;%CLASSPATH%
```

---

6. Differences Between OpenJDK and Oracle JDK

Feature	OpenJDK	Oracle JDK
License	Open-source (GPL)	Commercial (Free for dev, paid for production)
Performance	Similar performance	Optimized for enterprise usage
Support	Community-driven updates	Official support from Oracle
Features	Includes core Java features	Includes additional tools, optimizations

---

7. How Java Programs are Compiled and Executed

1. **Write Code:** Create a .java file.
2. **Compile Code:** Convert .java to .class (bytecode) using javac.

javac MyProgram.java

3. **Execute Code:** Run .class using java command.

java MyProgram

4. **JVM Execution:** JVM loads, verifies, and executes the bytecode.

---

8. What is Just-In-Time (JIT) Compilation? How Does it Improve Java Performance?

- JIT compilation is a technique where JVM compiles bytecode into native machine code **at runtime** instead of interpretation.
- **Performance Improvement:**
  - Reduces execution time by avoiding repeated interpretation.
  - Optimizes frequently used code using **HotSpot compiler**.
  - Uses caching techniques for faster execution.

---

9. Role of Java Virtual Machine (JVM) in Program Execution

- **Loads Java Class Files** and checks bytecode.
- **Converts Bytecode into Machine Code** using the JIT compiler.
- **Manages Memory Allocation and Garbage Collection.**
- **Ensures Security** by running Java programs in a sandboxed environment.