**Java Programming Basics**

**Name: Bhagyashri Haridas Date (JUHU)**

**Part 1: Introduction to Java**

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

**ANS:**

Java is a high-level, object-oriented, and platform-independent programming language developed by Sun Microsystems (now owned by Oracle) in 1995. It follows the "Write Once, Run Anywhere" (WORA) principle, meaning Java programs can run on any platform with a Java Virtual Machine (JVM).

Significance in Modern Software Development:

**Cross-Platform Compatibility:** Java runs on Windows, macOS, Linux, and mobile devices.

**Scalability:** Used in small applications to large-scale enterprise systems.

**Security:** Offers built-in security features like bytecode verification and access control.

**Multi-threading:** Supports concurrent programming, making it efficient for high-performance applications.

**Wide Industry Usage:** Powers web applications, mobile apps (Android), financial software, cloud computing, and AI/ML.

2. List and explain the key features of Java.

**ANS:**  Key Features of Java

**Platform Independence** – Java uses JVM, making it cross-platform.

**Object-Oriented** – Everything in Java is based on objects and classes.

**Simple & Readable** – Java has a clean syntax compared to C/C++.

**Secure** – Features like bytecode verification and a Security Manager protect against threats.

**Robust** – Java provides automatic memory management (Garbage Collection) and handles exceptions effectively.

**Multi-threaded** – Supports multiple threads for parallel execution.

**High Performance** – Uses Just-In-Time (JIT) compilation for faster execution.

**Distributed Computing** – Java supports distributed computing using RMI (Remote Method Invocation) and EJB (Enterprise JavaBeans).

**Rich API** – Provides predefined libraries for networking, database, GUI, and more.

**Community Support** – A large community ensures continuous improvement and innovation.

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

Java fit in?

**ANS:**

**Feature** **Compiled Languages** **Interpreted Languages** **Java**

**Execution** Translates the entire code Translates and executes Uses both

to machine code before line-by-line (compilation &

execution interpretation)

**Speed** Faster execution Slower execution Optimized by JIT

(since precompiled) (real-time translation) Compiler

**Portability** Less portable Highly portable Platform independent

(machine-dependent) (via JVM)

**Examples** C, C++ Python, JavaScript Java (Hybrid: Compiled + +Interpreted)

Java fits in both categories because:

Java code is compiled into bytecode (.class files) using the javac compiler.

The JVM interprets the bytecode or uses the JIT compiler for faster execution.

4. Explain the concept of platform independence in Java.

**ANS:**

Platform independence means that Java code can run on any operating system without modification.

How does Java achieve this?

**Compilation to Bytecode** – Java source code (.java) is compiled into bytecode (.class), which is not machine-specific.

**JVM Execution** – The Java Virtual Machine (JVM) on each OS interprets the bytecode and converts it into machine code.

**"Write Once, Run Anywhere" (WORA**) – Since the JVM is available for different operating systems, Java programs run unchanged on Windows, macOS, Linux, and mobile devices.

**Example:**

class file created on Windows can run on Linux or macOS without recompilation.

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

**ANS:**

Java is used in multiple industries for various applications:

**Enterprise Applications** – Used in banking, healthcare, and e-commerce systems (e.g., Spring, Hibernate).

**Mobile Development** – Android apps are built using Java (Android SDK).

**Web Development** – Java is used for server-side applications with frameworks like Spring, JSP, and Servlets.

**Cloud Computing** – Java is used in cloud platforms like AWS, Google Cloud, and Azure.

**Big Data & AI/ML** – Java is used in Hadoop, Apache Spark, and machine learning frameworks.

**Game Development** – Java is used in 2D/3D gaming engines like LibGDX and jMonkeyEngine.

**Embedded Systems & IoT** – Java powers smartcards, sensors, and IoT devices.

**Cybersecurity** – Java is used in secure banking applications and cryptographic systems.

Example Companies using Java:

Google (Android)

Netflix (Backend services)

Amazon (Cloud computing)

Banking Apps (HDFC, ICICI, SBI)

**Part 2: History of Java**

1. Who developed Java and when was it introduced?

**ANS:**

Java was **developed by James Gosling** and his team at **Sun Microsystems** in 1991. It was officially released in **1995.**

**Key Facts:**

Java was originally created for interactive television but was too advanced for cable TV at the time.

Sun Microsystems introduced Java as a platform-independent and object-oriented language.

In 2010, Oracle Corporation acquired Sun Microsystems, and Java is now maintained by Oracle.

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

**ANS:**

Java was initially called **"Oak"** after an **oak tree outside James Gosling’s office.**

Why was the name changed?

The name Oak was already trademarked by another company.

The team renamed it to Java, inspired by Java coffee from Indonesia, reflecting its speed and energy.

3. Describe the evolution of Java versions from its inception to the present.

**ANS:**

Java has evolved significantly since its first release. Below are the key versions and their major features:

Java Version Year Major Features

Java 1.0 1995 First official release, JVM introduced.

Java 2 (JDK 1.2 - 1.4) 1998-2002 Swing, Collections Framework, HotSpot JVM.

Java 5 (J2SE 5.0) 2004 Generics, Enhanced for-loop, Autoboxing.

Java 6 2006 Performance improvements, scripting support.

Java 7 2011 Try-with-resources, switch with strings.

Java 8 2014 Lambda expressions, Streams, Functional Programming.

Java 9 2017 Modularization (Project Jigsaw), JShell (REPL).

Java 10-11 2018 Local-variable var, LTS (Long Term Support).

Java 12-16 2019-2021 Pattern Matching, Records, Text Blocks.

Java 17 (LTS) 2021 Sealed classes, Strong encapsulation.

Java 18-20 2022-2023 Virtual threads, Scoped values.

Java 21 (LTS) 2023 New features in pattern matching, virtual threads, performance boosts.

4. What are some of the major improvements introduced in recent Java versions?

**ANS:**

Recent Java versions have introduced several new features and performance enhancements:

**Java 8 (2014)** - The Most Influential Version

Lambda Expressions → Supports functional programming.

Streams API → Efficient data processing.

Default Methods in Interfaces → Backward compatibility.

**Java 9 (2017)** - Modularization

Project Jigsaw → Introduced modules to improve scalability.

JShell (REPL) → Interactive Java shell for testing.

**Java 10-11 (2018)** - Performance & Stability

var Keyword → Local variable type inference.

Garbage Collection Enhancements → Improved memory management.

Java 11 - Long-Term Support (LTS)

**Java 17 (2021)** - LTS with Stability

Sealed Classes → Restrict class inheritance.

Enhanced Random Number API → Better randomness control.

**Java 21 (2023)** - Latest LTS Version

Virtual Threads → Lightweight threads for high-performance applications.

Pattern Matching Enhancements → More readable and concise code.

Performance Boosts → Faster garbage collection and startup times.

5. How does Java compare with other programming languages like C++ and Python in

terms of evolution and usability?

**ANS:**

Java has evolved significantly, and it compares with C++ and Python as follows:

**Feature** **Java** **C++** **Python**

**Paradigm** Object-Oriented Object-Oriented + Procedural Multi-Paradigm (OOP +

Functional)

**Platform Independence** Yes (JVM) No (Compiled for each OS) Yes (Interpreted)

**Memory Management** Automatic Manual (new & delete) Automatic

(Garbage Collector) (Garbage Collector)

**Performance** Moderate High (Compiled, Slower

(JIT Optimization) Direct Machine Code) (Interpreted)

**Ease of Learning** Moderate Difficult (Complex Syntax) Easy (Readable Syntax)

**Usage** Enterprise Apps, Game Dev, System Programming, AI/ML, Web

Android, Web, Cloud Embedded Systems Dev, Data Science

**Part 3: Data Types in Java**

1. Explain the importance of data types in Java.

**ANS:**

Data types are essential in Java because they define the type of values that variables can store. They ensure:

**Memory Efficiency** – Allocates the right amount of memory for variables.

**Type Safety** – Prevents errors by restricting incompatible operations.

**Performance Optimization** – Helps the compiler optimize code execution.

**Readability & Maintainability** – Makes the code easier to understand and debug.

**Example:**

int id = 25; // Ensures 'id' holds an integer value only.

Without data types, Java would be error-prone and inefficient.

2. Differentiate between primitive and non-primitive data types.

**ANS:**

Feature Primitive Data Types Non-Primitive Data Types

Definition Basic built-in data types User-defined complex types

Memory Usage Stores values directly Stores references to objects

Size Fixed memory size Dynamic memory allocation

Examples int, double, char, boolean String, Array, Class, Interface

Operations Supports arithmetic & Requires method-based operations

logical operations

**Example:**

**int num = 10;** // Primitive (stores the actual value)

**String text = "Hello";** // Non-primitive (stores reference)

3. List and briefly describe the eight primitive data types in Java.

**ANS:**

Java has 8 primitive data types, categorized into 4 groups:

Data Type Size Default Value Description

**Integer Types**

byte 1 byte 0 Smallest integer type (-128 to 127)

short 2 bytes 0 Medium integer (-32,768 to 32,767)

int 4 bytes 0 Standard integer (-2^31 to 2^31-1)

long 8 bytes 0L Large integer (-2^63 to 2^63-1)

**Floating-Point Types**

float 4 bytes 0.0f Single-precision floating point

double 8 bytes 0.0d Double-precision floating point

**Character Type**

char 2 bytes \u0000 Stores a single character (Unicode)

**Boolean Type**

boolean 1 bit false Stores true or false

**Example:**

1. byte smallNumber = 100;

2. int age = 25;

3. float percentage = 87.5f;

4. char grade = 'A';

5. boolean isJavaFun = true;

4. Provide examples of how to declare and initialize different data types.

**ANS:**

public class DataTypesExample {

public static void main(String[] args) {

// Integer types

byte smallNum = 127;

int num = 1000;

long bigNum = 100000L;

// Floating-point types

float pi = 3.14f;

double largeDecimal = 123.456;

// Character type

char letter = 'J';

// Boolean type

boolean isJavaFun = true;

// Printing values

System.out.println("Byte: " + smallNum);

System.out.println("Int: " + num);

System.out.println("Long: " + bigNum);

System.out.println("Float: " + pi);

System.out.println("Double: " + largeDecimal);

System.out.println("Char: " + letter);

System.out.println("Boolean: " + isJavaFun);

}

}

5. What is type casting in Java? Explain with an example.

**ANS:**

Type casting is converting one data type into another.

**Types of Type Casting:**

**Implicit Casting (Widening**) → Smaller to larger type (done automatically).

**Explicit Casting (Narrowing)** → Larger to smaller type (manual conversion).

Example:

public class TypeCastingExample {

public static void main(String[] args) {

// Implicit Casting (Widening)

int num = 100;

double decimalNum = num; // int to double (Automatic)

System.out.println("Widening: " + decimalNum);

// Explicit Casting (Narrowing)

double largeValue = 99.99;

int intValue = (int) largeValue; // double to int (Manual)

System.out.println("Narrowing: " + intValue);

}

}

**Output:**

Widening: 100.0

Narrowing: 99

6. Discuss the concept of wrapper classes and their usage in Java.

**ANS:**

Wrapper classes provide object representation of primitive data types.

**List of Wrapper Classes:**

**Primitive Type**  **Wrapper Class**

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

**Why Use Wrapper Classes?**

1. Used in Collections (e.g., ArrayList)

2. Provide useful utility methods

3. Allow null values (primitives cannot be null)

**Example:**

public class WrapperExample {

public static void main(String[] args) {

int num = 10;

Integer wrappedNum = Integer.valueOf(num); // Boxing (primitive to object)

int unwrappedNum = wrappedNum.intValue(); // Unboxing (object to primitive)

System.out.println("Boxed: " + wrappedNum);

System.out.println("Unboxed: " + unwrappedNum);

}

}

7. What is the difference between static and dynamic typing? Where does Java stand?

**ANS:**

**Feature** **Static Typing** **Dynamic Typing**

**Definition** Type checking happens Type checking happens

at compile-time at runtime

**Variable Declaration** Must specify data type No need to specify

explicitly (int x = 5;) type (x = 5 in Python)

**Error Detection**  Errors detected before Errors detected during

execution execution

**Examples** Java, C, C++ Python, JavaScript

**Java is a statically typed language because:**

1. Variables must have a fixed type

2. Type errors are caught during compilation

3. Ensures better performance and reliability

**Example:**

int num = "Hello"; // Compilation error (Type mismatch)

**Coding Questions on Data Types:**

1. Write a Java program to declare and initialize all eight primitive data types and print

their values.

ANS:

class PrimitiveDataTypes1 {

public static void main(String[] args) {

// Integer types

byte b = 127;

short s = 32000;

int i = 100000;

long l = 1000000000L;

// Floating-point types

float f = 10.5f;

double d = 99.99;

// Character type

char c = 'A';

// Boolean type

boolean bool = true;

// Printing values

System.out.println("Byte: " + b);

System.out.println("Short: " + s);

System.out.println("Int: " + i);

System.out.println("Long: " + l);

System.out.println("Float: " + f);

System.out.println("Double: " + d);

System.out.println("Char: " + c);

System.out.println("Boolean: " + bool);

}

}

**OUTPUT:**



2. Write a Java program that takes two integers as input and performs all arithmetic

operations on them.

**ANS:**

import java.util.Scanner;

class ArithmeticOperations {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter first number: ");

int num1 = sc.nextInt();

System.out.print("Enter second number: ");

int num2 = sc.nextInt();

sc.close();

System.out.println("Addition: " + (num1 + num2));

System.out.println("Subtraction: " + (num1 - num2));

System.out.println("Multiplication: " + (num1 \* num2));

System.out.println("Division: " + (num1 / num2));

System.out.println("Modulus: " + (num1 % num2));

}

}

**OUTPUT:**



3. Implement a Java program to demonstrate implicit and explicit type casting.

**ANS:**

class TypeCastingExample {

public static void main(String[] args) {

// Implicit (Widening) Casting: small type to larger type

int num = 50;

double d = num; // int to double (automatic conversion)

System.out.println("Implicit Casting (int to double): " + d);

// Explicit (Narrowing) Casting: larger type to smaller type

double largeValue = 99.99;

int intValue = (int) largeValue; // double to int (manual conversion)

System.out.println("Explicit Casting (double to int): " + intValue);

}

}

**OUTPUT:**



4. Create a Java program that converts a given integer to a double and vice versa using

wrapper classes.

**ANS:**

class WrapperConversion {

public static void main(String[] args) {

// Convert Integer to Double

Integer intValue = 50;

Double doubleValue = intValue.doubleValue();

System.out.println("Integer to Double: " + doubleValue);

// Convert Double to Integer

Double d = 99.99;

Integer intFromDouble = d.intValue();

System.out.println("Double to Integer: " + intFromDouble);

}

}

**OUTPUT:**



5. Write a Java program to swap two numbers using a temporary variable and without

using a temporary variable.

**ANS:**

import java.util.Scanner;

class SwapNumbers {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter first number: ");

int a = sc.nextInt();

System.out.print("Enter second number: ");

int b = sc.nextInt();

sc.close();

// Swap using a temporary variable

int temp = a;

a = b;

b = temp;

System.out.println("After swapping (Using temp variable): a = " + a + ", b = " + b);

// Swap without using a temporary variable

a = a + b;

b = a - b;

a = a - b;

System.out.println("After swapping (Without temp variable): a = " + a + ", b = " + b);

}

}

**OUTPUT:**



6. Develop a program that takes user input for a character and prints whether it is a

vowel or consonant.

**ANS:**

import java.util.Scanner;

class VowelOrConsonantExample {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a character: ");

char ch = sc.next().charAt(0);

sc.close();

// Convert to lowercase for uniformity

ch = Character.toLowerCase(ch);

// Check if vowel or consonant

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

System.out.println(ch + " is a Vowel.");

} else if ((ch >= 'a' && ch <= 'z')) {

System.out.println(ch + " is a Consonant.");

} else {

System.out.println("Invalid input! Please enter an alphabet.");

}

}

}

**OUTPUT:**



7. Create a Java program to check whether a given number is even or odd using

command-line arguments.

**ANS:**

public class EvenOddCLI {

public static void main(String[] args) {

// Ensure that a command-line argument is provided

if (args.length == 0) {

System.out.println("Please provide a number as a command-line argument.");

return;

}

// Convert string input to an integer

int num = Integer.parseInt(args[0]);

// Check even or odd

System.out.println(num + " is " + (num % 2 == 0 ? "Even" : "Odd"));

}

}

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

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

**ANS:**

The Java Development Kit (JDK) is a software development environment used to develop, compile, and run Java programs.

**Differences Between JDK, JRE, and JVM**

**JDK JRE JVM**

**(Java Development Kit)**  **(Java Runtime Environment)**  **(Java Virtual Machine)**

**Definition** A full development Provides runtime for Executes Java bytecode

package for Java Java applications

**Includes** JRE + Compiler (javac) JVM + Libraries Executes .class **files**

+ Debugger + Tools

**Usage** Needed for developing Needed only for running Part of both JDK & JRE

Java applications Java programs

**Who Needs It?** Developers End-users Internal component of Java

**In Simple Terms:**

**JVM** → Runs Java programs.

**JRE** → Needed to run Java applications.

**JDK** → Needed to develop, compile, and run Java applications.

2. Explain the main components of JDK.

**ANS:**

The JDK consists of several important tools and components:

**Component** **Description**

Java Compiler (javac) Converts Java source code (.java) into bytecode (.class).

Java Virtual Machine (JVM) Executes Java bytecode on different platforms.

Java Runtime Environment (JRE) Includes JVM + Java libraries for running applications.

Java API (Application Programming Predefined classes and libraries (e.g., java.lang, Interface) java.util).

Java Debugger (jdb) Helps in debugging Java programs.

Java Archive (jar) Packages multiple .class files into a .jar file.

Java Documentation (javadoc) Generates documentation from Java source code.

3. Describe the steps to install JDK and configure Java on your system.

**ANS:**

**Step 1:** Download JDK

Visit Oracle JDK or OpenJDK.

Download the appropriate version for Windows, macOS, or Linux.

**Step 2:** Install JDK

Run the installer and follow the setup instructions.

By default, JDK is installed in:

Windows: C:\Program Files\Java\jdk-XX

macOS/Linux: /usr/lib/jvm/java-XX-openjdk/

**Step 3:** Configure Environment Variables (Windows)

**1. Set PATH:**

Go to Control Panel → System → Advanced System Settings → Environment Variables.

Add C:\Program Files\Java\jdk-XX\bin to the PATH variable.

**2. Verify Installation:**

Open Command Prompt and type:

java -version

javac -version

**Expected Output:**

java version "XX"

javac version "XX"

4. Write a simple Java program to print "Hello, World!" and explain its structure.

**ANS:**

**Java Code**

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

**Explanation:**

Component Description

public class HelloWorld : Defines a Java class named HelloWorld.

public static void main(String[] args): The main method, where execution begins.

System.out.println("Hello, World!"); Prints "Hello, World!" to the console.

**To Compile and Run:**

javac HelloWorld.java // Compiles the program

java HelloWorld // Runs the compiled program

5. What is the significance of the PATH and CLASSPATH environment variables in Java?

**ANS:**

**PATH Variable:**

Specifies the location of Java binaries (javac, java).

Allows Java commands to be executed from any terminal location.

**CLASSPATH Variable:**

Specifies the location of Java classes and JAR files needed for execution.

Used for custom libraries and packages.

**Example:**

set PATH=C:\Program Files\Java\jdk-XX\bin

set CLASSPATH=C:\myJavaPrograms

6. What are the differences between OpenJDK and Oracle JDK?

**ANS:**

**Feature** **OpenJDK** **Oracle JDK**

**License** Open-source (GPL) Proprietary with free & paid versions

**Performance** Almost identical Optimized for enterprise use

**Support** Community-driven Commercial support from Oracle

**Usage** Suitable for most applications Preferred for enterprise-level applications

For most cases, OpenJDK and Oracle JDK work the same.

7. Explain how Java programs are compiled and executed.

**ANS:**

1. Write Java Code (.java file).

2. Compile using javac → Converts code into bytecode (.class).

javac MyProgram.java

3. Execute using java command → JVM runs the program.

java MyProgram

4. JVM interprets bytecode and executes it on the OS.

**Example Process:**

Source Code → Compiler (javac) → Bytecode (.class) → JVM Execution

8. What is Just-In-Time (JIT) compilation, and how does it improve Java performance?

**ANS:**

**JIT Compilation (Just-In-Time)**

A performance optimization technique used by JVM.

Instead of interpreting bytecode line-by-line, JIT compiles frequently used bytecode into native machine code for faster execution.

**How JIT Improves Performance?**

Reduces interpretation overhead.

Optimizes hot code paths (frequently executed code).

Results in faster execution, similar to compiled languages like C++.

**Example:**

public class JITExample {

public static void main(String[] args) {

for (int i = 0; i < 100000; i++) {

System.out.println("Iteration: " + i);

}

}

}

**The JIT compiler optimizes the loop, making execution faster.**

9. Discuss the role of the Java Virtual Machine (JVM) in program execution

**ANS:**

The Java Virtual Machine (JVM) is an engine that runs Java bytecode and makes Java platform-independent.

**Key Responsibilities of JVM:**

Loads Java bytecode (.class files).

Interprets or compiles bytecode into machine code.

Manages memory (Heap, Stack).

Performs Garbage Collection (GC) to free unused memory.

Ensures security by running code inside a sandbox environment.

**JVM Execution Process:**

Class Loader → Loads .class files.

Bytecode Verifier → Checks security & integrity.

Execution Engine → Uses Interpreter or JIT Compiler.

Garbage Collector → Frees memory.

**Diagram:**

Java Source Code → Compiler → Bytecode → JVM → Machine Code