**Nishant Patil**

**PG-DAC**

**Java**

**Assignment 1**

**Part 1: Introduction to Java**

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

i) Java is a high-level, object-oriented programming language used to develop a wide variety of applications, from mobile to enterprise systems. It was created by James Gosling. It was then developed by Sun Microsystems in 1995.

ii) Java's main advantage is that it is platform-independent, meaning that programs written in Java can run on any device with a Java Virtual Machine (JVM).

iii) It is important today because it provides features like security, multi-threading, and portability, robustness, making it ideal for building large-scale, secure, and efficient software systems.

***2. List and explain the key Features of Java.***

* **Platform Independence**: Java code can run on any system with a JVM, making it platform-independent.
* **Object-Oriented**: Java focuses on using objects and classes, which makes the code more organized and reusable.
* **Simple**: Java has fewer complex features compared to languages like C++, making it easier to learn and use.
* **Secure**: Java has built-in security features like bytecode verification to help protect against malicious code.
* **Multithreading**: Java supports multi-threading, which allows for efficient execution of multiple tasks simultaneously.
* **Automatic Garbage Collection**: Java handles memory management automatically, freeing up memory that is no longer used.
* **Portability :** Java is portable across platforms because it's compiled into bytecode, which can run on any platform

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

* **Compiled Languages**: In compiled languages like C++, source code is translated into machine code by a compiler before it is run. This machine code is specific to each platform.
* **Interpreted Languages**: In interpreted languages like Python, the source code is executed line-by-line by an interpreter, which is slower but more flexible.

Java is a hybrid language because it is compiled into bytecode that is platform-independent. This bytecode is then executed by the JVM, which either interprets or compiles it into machine code specific to the platform it's running on.

***4. Explain the concept of platform independence in Java.***

Platform independence means that Java code, after being compiled, is converted into bytecode. This bytecode can be executed on any platform with a JVM, regardless of the underlying operating system or hardware. The phrase "write once, run anywhere" is often used to describe this feature of Java.

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

Java is used in various fields:

* **Web Development**: Backend development using frameworks like Spring and Hibernate.
* **Mobile Applications**: Android apps are built using Java.
* **Enterprise Applications**: Java is commonly used in business applications, like banking systems and e-commerce websites.
* **Scientific Applications**: Java is used for simulations and scientific computing.
* **Embedded Systems**: Java is used in embedded systems like smart devices.
* **Cloud Computing**: Java plays a major role in building cloud-based applications.

**Part 2: History of Java**

***1. Who developed Java and when was it introduced?***

Java was developed by **James Gosling** and **Mike Sheridan** at Sun Microsystems in 1991. It was released to the public in 1995.

***2. What was java initially called? Why was its name changed?***

Java was initially called Oak. The name was changed to Java because "Oak" was already trademarked by another company, and the developers wanted a unique name. "Java" was chosen after the developers enjoyed drinking coffee, and it reflected the language's dynamic and lively nature.

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

* Java 8 (1.8) introduced lambdas, Stream API, and the new Date/Time API.
* Java 11 (LTS) added the HTTP Client API, removed Java EE and CORBA, and improved garbage collection.
* Java 15 introduced Sealed Classes, Text Blocks, and hidden classes.
* Java 17 (LTS) finalized Sealed Classes, added Pattern Matching for switch, and improved JVM performance.
* Java 21 (LTS) introduced virtual threads (Project Loom), final Pattern Matching for switch, and Project Panama (native code interoperability).

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

* **Java 9**: Introduced the module system to organize large applications.
* **Java 10**: Added local variable type inference with the var keyword.
* **Java 11**: The HTTP Client API and Long-Term Support (LTS) features were added.
* **Java 12 - 16**: Features like text blocks, null pointer exceptions, and pattern matching were introduced.
* **Java 17**: Added sealed classes, which allow more control over class inheritance.

***5. How does Java compare with other languages like C++ and Python?***

* **C++**: C++ offers more control over system resources and is faster for performance-critical applications, but it’s harder to learn and manage. Java is easier to use, with better memory management, but is generally slower than C++.
* **Python**: Python is simpler and faster for scripting and prototyping, but Java is better suited for larger, more complex applications due to its performance and scalability.

**Part 3: Data Types in Java**

***1. Why are data types important in Java?***

Data types in Java define what kind of data a variable can hold (like numbers, text, or logical values). They help the program understand how much memory to allocate and which operations can be performed. Java is a statically-typed language, meaning data types must be defined when the program is written.

***2. What’s the difference between primitive and non-primitive data types?***

* **Primitive Data Types**: These are basic types like integers, characters, and booleans. Examples are int, char, boolean, etc. They hold actual data and are not objects.
* **Non-Primitive Data Types**: These are more complex types like objects and arrays. Examples are String, Array, Class, etc. They are created using primitive types.

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

1. **byte**: 8-bit integer value.(1 byte)
2. **short**: 16-bit integer value.(2 bytes)
3. **int**: 32-bit integer value.(4 bytes)
4. **long**: 64-bit integer value.(8 bytes)
5. **float**: 32-bit floating-point number.(4 bytes)
6. **double**: 64-bit floating-point number.(8 bytes)
7. **char**: 16-bit Unicode character.(2 bytes)
8. **boolean**: Represents true or false.

***4. How to declare and initialize different data types?***

byte b = 10;

short s = 32000;

int i = 100000;

long l = 100000L;

float f = 3.14f;

double d = 3.14159;

char c = 'A';

boolean flag = true;

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

**Type casting** means converting one data type to another. There are two types:

* **Implicit Casting (Widening)**: Java automatically converts a smaller data type to a larger one. Example: converting int to long.

int i = 100;

long l = i;

* **Explicit Casting (Narrowing)**: The programmer must manually convert a larger data type to a smaller one. Example: converting long to int.

long l2 = 100L;

int i2 = (int) l2;

***6. What are wrapper classes and how are they used?***

Wrapper classes convert primitive types into objects. These are used when an object is required instead of a primitive. Examples include:

* **Integer** for int
* **Double** for double
* **Character** for char

Integer x = Integer.valueOf(10);

Double y = Double.valueOf(3.14);

***7. What’s the difference between static and dynamic typing?***

* **Static Typing**: The type of a variable is checked at compile-time (as in Java).
* **Dynamic Typing**: The type of a variable is checked at runtime (as in Python).

**Coding Questions on Data Types**

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

public class DataTypes {

public static void main(String[] args) {

boolean boolAssigned = true;

char charAssigned = 'A';

byte byteAssigned = 100;

short shortAssigned = 200;

int intAssigned = 300;

long longAssigned = 400L;

float floatAssigned = 5.5f;

double doubleAssigned = 10.5;

System.out.println("boolean: " + boolAssign);

System.out.println("char: " + charAssign);

System.out.println("byte: " + byteAssign);

System.out.println("short: " + shortAssign);

System.out.println("int: " + intAssign);

System.out.println("long: " + longAssign);

System.out.println("float: " + floatAssign);

System.out.println("double: " + doubleAssign); }

}

**2*. Write a Java program to take two integers as input and perform all arithmetic operations on them*.**

import java.util.Scanner;

public class ArithmeticOp {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int a = sc.nextInt();

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

int b = sc.nextInt();

System.out.println("Addition: " + (a + b));

System.out.println("Subtraction: " + (a - b));

System.out.println("Multiplication: " + (a \* b));

System.out.println("Division: " + (a / b));

System.out.println("Modulus: " + (a % b));

}

}

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

public class TypeCast {

public static void main(String[] args) {

int i = 100;

long l = i; // int to long

System.out.println("Implicit casting: " + l);

long l2 = 100L;

int i2 = (int) l2; // long to int

System.out.println("Explicit casting: " + i2);

}

}

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

public class WrapperClass {

public static void main(String[] args) {

int i = 100;

double d = 3.14;

Double doubleVal = Double.valueOf(i);

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

Integer intVal = Integer.valueOf((int) d);

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

}

}

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

public class SwapNum {

public static void main(String[] args) {

int a = 1, b = 2;

int temp = a;

a = b;

b = temp;

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

c = 10; d = 20;

c = c + d;

d = c - d;

c= c - d;

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

}

}

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

import java.util.Scanner;

public class VowelConsonant {

public static void main(String[] args) {

Scanner scr = new Scanner(System.in);

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

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

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

ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {

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

} else {

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

}

}

}

**7. *Create a Java program to check whether a given number is even or odd using command-line arguments.***

public class EvenOdd {

public static void main(String[] args) {

if (args.length == 0) {

System.out.println("Please provide a number.");

return;

}

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

if (num % 2 == 0) {

System.out.println(num + " is even.");

} else {

System.out.println(num + " is odd.");

}

}

}

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

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

* **JDK (Java Development Kit)**: It is a complete software development kit required for developing Java applications. It includes the **JRE (Java Runtime Environment)**, the **JVM (Java Virtual Machine)**, and development tools like the **Java compiler (javac)** and debuggers.
* **JRE (Java Runtime Environment)**: The JRE provides libraries and the JVM to run Java applications. It does not include development tools like the compiler, so it is intended for running, not developing, Java applications.
* JDK is used for development (includes JRE + tools), JRE is for running Java applications, and JVM is the engine that executes bytecode.

1. ***Explain the main components of JDK.***

The **main components of the JDK** are:

* **JVM (Java Virtual Machine)**: Executes Java bytecode.
* **Java Compiler (javac)**: Compiles Java source code (.java files) into bytecode (.class files).
* **Java Debugger (jdb)**: Used for debugging Java programs.
* **Java Archiver (jar)**: Bundles Java classes into .jar (Java ARchive) files.
* **Java Doc (javadoc)**: Generates API documentation from Java source code comments.
* **Libraries and Tools**: Standard libraries (like java.util) and utilities for various tasks.
* **JRE**: It includes the libraries and JVM for running Java programs.

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

Here’s a general guide to installing and configuring Java:

1. **Download JDK**:
   * Visit the official Oracle or OpenJDK website and download the suitable version for your system (Windows, macOS, Linux).
2. **Install JDK**:
   * Follow the installation steps. On Windows, run the installer.
3. **Set JAVA\_HOME**:
   * Set the JAVA\_HOME environment variable to point to the installation directory of the JDK. Example:
     + On Windows: C:\Program Files\Java\jdk-XX
4. **Add Java to the PATH**:
   * Add the bin directory of JDK to the system PATH environment variable to make Java commands (javac, java) accessible globally.
     + On Windows: C:\Program Files\Java\jdk-XX\bin
5. **Verify Installation**:
   * Open a command prompt or terminal and type java -version and javac -version to check the installed version of Java and the compiler.

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

java

Copy

public class Simple {

public static void main(String[] args) {

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

}

}

**Explanation**:

* **public class Simple**: Defines a class named Simple. In Java, everything must be inside a class.
* **public static void main(String[] args)**: The main method is the entry point of any Java application. It’s where the program starts executing.
  + **String[] args**: An array of strings that can store command-line arguments passed to the program.
* **System.out.println("Hello, World!");**: Prints the message to the console.

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

* **PATH**: This environment variable tells the operating system where to find executable programs, like java and javac. By adding the bin directory of JDK to PATH, we can run Java from any directory in the terminal or command prompt.
* **CLASSPATH**: It defines the locations where Java classes are stored. The JVM uses CLASSPATH to find .class files or JAR files. It is essential for the system to locate user-defined classes and libraries.

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

* **OpenJDK**:
  + Open-source version of the JDK.
  + Fully compliant with the Java Standard Edition specifications.
  + Supported by the community, with regular updates.
* **Oracle JDK**:
  + Proprietary version of the JDK offered by Oracle.
  + Contains additional commercial features like Java Mission Control, flight recorder, and advanced monitoring tools.
  + Requires a commercial license for certain uses (after Oracle's licensing changes).

In most cases, OpenJDK is sufficient for development, but Oracle JDK provides additional tools for enterprise use.

***7. Explain how Java programs are compiled and executed.***

* **Compilation**: Java source code (.java file) is compiled using the javac compiler into bytecode (.class files).
* **Execution**: The bytecode is executed by the JVM. The JVM reads the bytecode and translates it into machine code, which is executed on the host machine.

This two-step process (compiling to bytecode and running on the JVM) ensures platform independence in Java.

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

* **JIT Compilation** is a feature of the JVM where bytecode is compiled into native machine code just before execution. It helps in optimizing performance by compiling only the frequently used code(Hotspot methods) , rather than compiling the entire program upfront.
* **Benefits**:
  + **Faster execution**: By compiling code at runtime, JIT allows for optimizations that improve the speed of execution.
  + **Adaptive optimization**: JIT can optimize code based on actual runtime behavior.

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

The **JVM** is responsible for:

* **Loading**: It loads compiled bytecode files into memory.
* **Execution**: It interprets or compiles the bytecode to native code for execution.
* **Memory Management**: It manages memory (heap and stack) and garbage collection.
* **Platform Independence**: The JVM allows Java programs to be platform-independent, as it abstracts away the underlying hardware.

Essentially, the JVM is what makes Java cross-platform, enabling Java programs to run on any system that has a JVM installed, without modification.