Assignment no1

Java programming Basics

Name Akshay Janrao

PRN No :25024052006

Part 1: Introduction to Java

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

Answer Java is a programming language that is object oriented in nature, it is high level language with a key feature of "WORA"ie 'write once, runs anywhere' it is platform independent and secure language.

Java helped in developing modern software by introducing key features like

Platform independent nature, scalability of the software, versatility in nature i.e. it can be used to develop mobile apps, web applications, big data processing etc.

2. List and explain the key features of Java.

Answer Key features of JAVA are as follows

- Platform independence: As the JVM converts the compiled program into bytecode it can be read in any operating system like windows, Linux etc.
- Robust nature and secure nature: Due the garbage collector presented in JVM it clears the memory and hence removes the possibility if memory leak and manages the memory efficiently.
- Scalability of the platform: Due to its object-oriented nature JAVA is highly scalable in nature it can be used in developing in mobile apps to large scale website to various web applications.

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

Answer

COMPILED LANGUAGE	INTERPRETED LANGUAGE

Once the program is compiled it is expressed in the instructions of the target machine.	The instructions are not directly executed by the target machine.
There are at least two steps to get from source code to execution.	There is only one step to get from source code to execution.
Compilation errors prevent the code from compiling.	All the debugging occurs at run-time.

JAVA mainly lies between the both types of language as it compiled to bytecode and the it is interpreted.

4. Explain the concept of platform independence in Java.

Answer As the java code is written it is converted Into .java file .This .java file is then sent into compiler that converts it into .class file i.e. in bytecode this allows the file to be interpreted in any OS platform if it has JVM or JRE installed this allows java to achieve the platform independent nature.

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

Answer Java's versatility and robustness have led to its widespread adoption across numerous industries. Here are some key real-world applications of Java

Android Mobile Applications:

- Java is a foundational language for Android app development.

While Kotlin is now favored by Google, much of the Android operating system and many existing apps are built with Java.

Enterprise Applications:

- Java is extensively used for building large-scale enterprise applications. Its stability, security, and scalability make it ideal for complex systems

Big Data:

- Java plays a significant role in the big data ecosystem. Technologies like Hadoop and Spark, which are used for processing massive datasets, are often written in Java.

Part 2: History of Java

1. Who developed Java and when was it introduced?

Java was developed by James Gosling, Mike Sheridan, and Patrick Naughton at Sun Microsystems. It was officially introduced to the world on May 23, 1995, during the SunWorld conference.

2. What was Java initially called? Why was its name changed?
   Java was initially called as “oak” but was later it was changed ot Java to avoid trademark issues.
3. Describe the evolution of Java versions from its inception to the present.
   Java has gone under many changes they are as follows
   Java 1.0 (1996):

- The official release of Java, introducing the "write once, run anywhere" (WORA) concept through the Java Virtual Machine (JVM).
- Key features included applets, which brought interactive content to web browsers.

Java 1.1 (1997):

- Introduced inner classes, JavaBeans, Java Database Connectivity (JDBC), and Remote Method Invocation (RMI)

Java 2 (1998-2002):

- **J2SE 1.2 (1998):**
  - The platform was rebranded as Java 2, with Standard Edition (J2SE), Enterprise Edition (J2EE), and Micro Edition (J2ME).
  - Introduced the Collections Framework, Swing GUI components, and the Just-In-Time (JIT) compiler.
- Modern Java

Java SE 9 (2017):

Introduced the modular system (Project Jigsaw), enhancing platform scalability and maintainability.

Faster Release Cycle:

Oracle shifted to a six-month release cycle, delivering new features more frequently.

Java SE 11 (2018):

A Long-Term Support (LTS) release, providing stability for enterprise applications.

Java SE 17 (2021):

The next LTS release after java 11. Including features like sealed classes.

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

Some modern features are as followed

- Modular System

Project Jigsaw introduced modularity, allowing developers to break down applications into smaller, manageable modules.
This improves scalability and maintainability

- Lambda Expressions and Stream API

These features brought functional programming concepts to Java, enabling more concise and expressive code

- Records

Records provide a concise syntax for declaring data-only classes, reducing boilerplate code.

- Enhanced Garbage Collection

Improvements to garbage collectors like ZGC and Shenandoah have reduced pause times, leading to better application performance

5. How does Java compare with other programming languages like C++ and Python in terms of evolution and usability?

Answer

Java: Developed in 1995 with a focus on cross-platform compatibility, Java has consistently evolved through regular updates. Its adaptability has made it one of the most widely used languages for enterprise applications, Android development, and backend systems.

C++: Originating in the 1980s, C++ is an extension of C and focuses on high-performance, low-level memory control, and system programming. Though it evolves more slowly than Java, recent updates like C++11, C++17, and C++20 have introduced modern features like lambda expressions and better standard libraries.

Python: Created in 1991 with simplicity in mind, Python's evolution emphasizes developer productivity. Its frequent updates focus on enhancing usability, machine learning capabilities, and integrations with modern tools. Python has become a dominant force in AI, data science, and web development.

Usability

Java: Known for its balance between performance and readability, Java's strict object-oriented principles and static typing make it robust and reliable for large-scale applications. However, its verbosity can be a downside for smaller projects.

C++: Offers unparalleled control over system resources, making it ideal for gaming, real-time systems, and high-performance applications. However, its steep learning curve and potential for bugs (due to manual memory management) make it challenging for beginners.

Python: Renowned for its simplicity and readability, Python is beginner-friendly and highly versatile. Its dynamic typing and vast libraries make it perfect for prototyping and rapid application development. However, it may lag behind Java and C++ in performance for resource-intensive tasks.

Part 3: Data Types in Java

1. Explain the importance of data types in Java.

Memory Management: Data types determine how much memory is allocated for a variable. For example, an int requires 4 bytes, while a double requires 8 bytes. This helps optimize memory usage.

Type Safety: By specifying data types, Java ensures that only valid operations are performed on variables. For instance, you can't assign a string to an integer variable, which helps prevent errors.

Improved Performance: Since Java knows the data type at compile time, it can generate optimized machine code, leading to faster execution.

Code Clarity and Maintainability: Explicitly defining data types makes the code more readable and easier to understand, reducing ambiguity.

Support for Polymorphism: Data types, particularly with objects, enable polymorphism and method overloading, which are key principles of object-oriented programming

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

Primitive data	Non primitive data
These are data types pre-defined in java	These are user defined data they can represent complex data
There values can’t be changed	The content can be changed as per required

They can store the data or values directly	Stores a reference to the data location in memory
These are stored in stack memory	Generally stored in heap memory

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

The data types are as followed

byte:

- This is an 8-bit signed two's complement integer.
- It's used for saving memory in large arrays, where the memory savings actually matters.
- Range: -128 to 127.
- **short:**

- This is a 16-bit signed two's complement integer.
- It's also used for memory savings in large arrays, but for larger ranges than byte.
- Range: -32,768 to 32,767.
- **int:**

- This is a 32-bit signed two's complement integer.
- It's the most commonly used integer data type.
- Range: -2,147,483,648 to 2,147,483,647.
    - **long:**

- This is a 64-bit signed two's complement integer.
- It's used when you need a range of values larger than those provided by int.
- Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- **float:**

- This is a 32-bit single-precision IEEE 754 floating-point number.
- It's used for representing single-precision floating-point numbers.
- It should not be used for precise values, such as currency.

- **double:**
- This is a 64-bit double-precision IEEE 754 floating-point number.
- It's used for representing double-precision floating-point numbers and is generally the default choice for floating-point values.

- **char:**
- This is a 16-bit Unicode character.
- It represents a single character.
- Range: '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).

- **boolean:**
- This represents a logical value that can be either true or false.
- It's used for boolean variables and expressions.

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

In Java, there are different types of variables, for example:

- **boolean Data Type**
- The boolean data type represents a logical value that can be either true or false.
- **Syntax:**
- boolean booleanVar;
- **byte Data Type**
- The byte data type is an 8-bit signed two's complement integer. The byte data type is useful for saving memory in large arrays.
- **Syntax:**
- byte byteVar;
- **short Data Type**
- The short data type is a 16-bit signed two's complement integer. Similar to byte, a short is used when memory savings matter, especially in large arrays where space is constrained.
- **Syntax:**
- short shortVar;
- **int Data Type**
- It is a 32-bit signed two's complement integer.
- **Syntax:**
- int intVar;
- **long Data Type**
- The long data type is a 64-bit signed two's complement integer. It is used when an int is not large enough to hold a value, offering a much broader range.
- **Syntax:**
- long longVar;
- **loat Data Type**
- The float data type is a single-precision 32-bit IEEE 754 floating-point
- **Syntax:**
- float floatVar;
- **double Data Type**
- The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice. The size of the double data type is 8 bytes or 64 bits.
- **Syntax:**
- double doubleVar;
- **char Data Type**
- The char data type is a single 16-bit Unicode character with the size of 2 bytes (16 bits).
- **Syntax:**
- char charVar;

5. What is type casting in Java? Explain with an example.
   Typecasting in Java is the process of converting one data type to another data type using the casting operator.
   To enable the use of a variable in a specific manner, this method requires explicitly instructing the Java compiler to treat a variable of one data type as a variable of another data type.
   Syntax:
   <datatype> variableName = (<datatype>) value;
   E.g. narrowing double to float to long to int

6. Discuss the concept of wrapper classes and their usage in Java.
A wrapper class in java is the one whose object contains primitive data types when a object is created to a wrapper class,it contains the field and in this fiel a primitive data type is stored.
Use
The wrapper class in Java is used to convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method.

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

Static Typing:

- **Type checking:** Occurs at compile time. This means the compiler analyzes the code before it's executed to ensure that variables are used with their correct data types.

Dynamic Typing:

- **Type checking:** Occurs at runtime.
  This means the type of a variable is determined while the program is executing

Java is astatically typed language i.e. type checking is performed at compile time.
□

Coding Questions on Data Types:

1. Write a Java program to declare and initialize all eight primitive data types and print their values.
   The code is as follows

   public class PrimitiveDataTypes {

```
        public static void main(String[] args) {

            byte byteVar = 127;
            System.out.println("byte: " + byteVar);

            short shortVar = 32767;
            System.out.println("short: " + shortVar);

            int intVar = 2147483647;
            System.out.println("int: " + intVar);

            long longVar = 92233720368547758O7L;
            System.out.println("long: " + longVar);


            float floatVar = 3.14159f;
            System.out.println("float: " + floatVar);

            double doubleVar = 2.718281828459045;
            System.out.println("double: " + doubleVar);

            char charVar = 'A';
            System.out.println("char: " + charVar);

            boolean booleanVar = true;
            System.out.println("boolean: " + booleanVar);
        }
    }
```

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

```
import java.util.Scanner;

public class ArithmeticOperations {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.print("Enter the first integer: ");
        int num1 = input.nextInt();
```

```
        System.out.print("Enter the second integer: ");
        int num2 = input.nextInt();


        int sum = num1 + num2;
        int difference = num1 - num2;
        int product = num1 * num2;
        int quotient = num1 / num2;
        int remainder = num1 % num2;

        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.println("Quotient: " + quotient);
        System.out.println("Remainder: " + remainder);

        input.close();
    }
}
```

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

```
public class TypeCasting {


    public static void main(String[] args) {
System.out.println("Implicit Type Casting (Widening):");
        int intValue = 100;
        long longValue = intValue;
        float floatValue = longValue;
        double doubleValue = floatValue;


        System.out.println("int: " + intValue);
        System.out.println("long: " + longValue);
        System.out.println("float: " + floatValue);
        System.out.println("double: " + doubleValue);


        System.out.println("\nExplicit Type Casting (Narrowing):");
```

```
        double anotherDouble = 123.456;
        float anotherFloat = (float) anotherDouble;
        long anotherLong = (long) anotherFloat;
        int anotherInt = (int) anotherLong;
        short anotherShort = (short) anotherInt;
        byte anotherByte = (byte) anotherShort;

        System.out.println("double: " + anotherDouble);
        System.out.println("float: " + anotherFloat);
        System.out.println("long: " + anotherLong);
        System.out.println("int: " + anotherInt);
        System.out.println("short: " + anotherShort);
        System.out.println("byte: " + anotherByte);


        double largeDouble = 12345678910.11;
        int smallerInt = (int) largeDouble;
        System.out.println("\nData loss example: large double to int: " + smallerInt);
    }
}
```

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

```
public class WrapperClassConversions {

    public static void main(String[] args) {


        int intValue = 10;
        Integer integerObject = Integer.valueOf(intValue);
        double doubleValue = integerObject.doubleValue();
        System.out.println("Integer " + intValue + " to Double: " + doubleValue);

        double doubleValue2 = 15.75;
```

```
        Double doubleObject = Double.valueOf(doubleValue2);
        int intValue2 = doubleObject.intValue();

        System.out.println("Double " + doubleValue2 + " to Integer: " + intValue2);

        double doubleValue3 = 15.75;
        Double doubleObject3 = Double.valueOf(doubleValue3);
        int intValue3 = (int) Math.round(doubleObject3); // Convert to integer
    using rounding.
        System.out.println("Double " + doubleValue3 + " to Integer (Rounded): " +
    intValue3);
      }
    }
```

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

```
public class SwapNumbers {

  public static void main(Stringargs) {
    int num1 = 10;
    int num2 = 20;

    System.out.println("Before swap:");
    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);

    int temp = num1;
    num1 = num2;
    num2 = temp;

    System.out.println("\nAfter swap (using temp):");
    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);

    num1 = 10;
    num2 = 20;
    num1 = num1 + num2;
    num2 = num1 - num2;
    num1 = num1 - num2;

    System.out.println("\nAfter swap (without temp):");
```

```
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

        num1 = 10;
        num2 = 20;

        num1 = num1 ^ num2;
        num2 = num1 ^ num2;
        num1 = num1 ^ num2;

        System.out.println("\nAfter swap (without temp, using XOR):");
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

    }
}
```

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 VowelOrConsonant {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a character: ");
        char ch = input.next().charAt(0);

        ch = Character.toLowerCase(ch);

        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(ch + " is not a letter.");
        }

        input.close();
    }
}
```

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

```
public class EvenOrOdd {

    public static void main(String[] args) {

        if (args.length == 0) {
            System.out.println("Please provide a number as a command-line argument.");
            return;
        }

        try {
            int number = Integer.parseInt(args[0]);

            if (number % 2 == 0) {
                System.out.println(number + " is even.");
            } else {
                System.out.println(number + " is odd.");
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please provide a valid integer.");
        }
    }
}
```

Part 4: Java Development Kit (JDK)

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

JDK (Java Development Kit)

- The JDK is a software development kit that provides the tools necessary to develop Java applications.
- It includes:
- JRE (Java Runtime Environment)
- A compiler (javac)
- A debugger (jdb)
- Other development tools
- Essentially, if you want to write Java programs, you need the JDK.
  Differences are as followed
- The JDK is for development; the JRE is for running Java applications.

- The JDK includes the JRE
- The JVM is a component of the JRE
- The JRE provides the environment for the JVM to run

2. Explain the main components of JDK.
The components are as followed
Java Compiler (javac):

- This is the core component that translates Java source code (.java files) into bytecode (.class files).

Java Runtime Environment (JRE):

- As mentioned earlier, the JRE is a crucial part of the JDK. It provides the runtime environment for executing Java applications.

Java Archive (jar) Tool:

- This tool allows you to package multiple Java class files and related resources into a single archive file (.jar file).

Java Debugger (jdb):

- This tool helps developers find and fix errors (bugs) in their Java code.

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

1. Download the JDK:

Go to the official Oracle website or the OpenJDK website. Oracle JDK might require an Oracle account, while OpenJDK is typically open-source.

Oracle JDK: Search for "Oracle JDK download"

OpenJDK: Search for "OpenJDK download"

Choose the appropriate JDK version for your operating system (Windows, macOS, Linux) and architecture (x64, ARM, etc.).

Download the installer or archive file.

2. Install the JDK:

Run the downloaded installer (.exe file).

Follow the installation wizard, accepting the default settings or customizing the installation path as needed.

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”);
}
```

1.public class HelloWorld { ... }:

- This line declares a public class named HelloWorld.

2.public static void main(String[] args) { ... }:

- This line defines the main method.
- The main method is the entry point of a Java program. When you run a Java program, the JVM starts execution from the main method.

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

- This line prints the string "Hello, World!" to the console.
- System.out is a standard output stream object that represents the console

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

PATH Environment Variable:

- **Purpose:** The PATH variable tells the operating system where to find executable files.
- **Significance in Java:**
    - When you type java or javac in the command line, the operating system searches the directories listed in the PATH variable to find these executables.

CLASSPATH Environment Variable:

- **Purpose:** The CLASSPATH variable tells the Java Virtual Machine (JVM) where to find .class files (compiled Java bytecode) and other resources (like JAR files) that your Java program needs to run.
- **Significance in Java:**
    - When the JVM encounters a import statement or tries to load a class, it searches the directories and JAR files listed in the CLASSPATH to find the corresponding .class files

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

OpenJDK	Oracle JDK	
License	Free, open source, GNU General Public License	Closed source, commercial license
Maintenance	Maintained by Oracle, Red Hat, and the community	Maintained by Oracle
Support	Commercial support available from OpenJDK vendors	Commercial support available from Oracle

7. Explain how Java programs are compiled and executed.

Compilation

Java programs are compiled using the **Java Compiler (javac)**, which is part of the Java Development Kit (JDK).

When you run the command javac Example.java, the compiler checks your code for syntax errors. If no errors are found, the compiler converts the source code into **bytecode**, a platform-independent, intermediate representation.

The bytecode is saved in a .class file. For example, after compiling Example.java, you get Example.class.

Execution

The .class file (bytecode) is executed by the **Java Virtual Machine (JVM)**. The JVM is responsible for running the program on a specific operating system and hardware.

The command java Example instructs the JVM to load the Example.class file, interpret its bytecode, and execute it.

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

JIT compilation is the process of converting Java bytecode (the intermediate code produced by the Java compiler) into **native machine code** at runtime.

Unlike traditional compilation, which translates the entire program before execution, JIT compiles the bytecode into machine code **on-demand**, during program execution

Working

Initial Interpretation: When a Java program runs, the JVM interprets the bytecode and executes it line by line.

Optimization by JIT: For frequently executed portions of the bytecode (like loops or methods), the JIT compiler steps in and compiles them into native machine code.

Performance improvement

Execution Speed: Native machine code runs much faster than interpreted bytecode, reducing execution time for critical parts of the program.

Hotspot Optimization: JIT focuses on "hotspots" (frequently used code paths), applying advanced optimizations like inlining methods, loop unrolling, and dead code elimination.

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

The Java Virtual Machine (JVM) plays a crucial role in executing Java programs by serving as the runtime environment that bridges the gap between platform-independent bytecode and platform-specific machine code.

Class Loading

- The JVM starts by loading the .class files (bytecode) into memory. This is done by its **Class Loader** subsystem.
- It supports dynamic class loading, meaning classes are loaded only when they're needed, saving memory and improving efficiency.

b. Bytecode Verification

- Before executing bytecode, the JVM ensures it adheres to Java's safety and security standards.
- This step prevents malicious or erroneous code from crashing the system or compromising security.

c. Execution Engine

- The JVM's **Execution Engine** is responsible for executing the loaded bytecode.
- It uses both interpretation (executing bytecode line by line) and Just-In-Time (JIT) compilation to convert bytecode into native machine code for faster execution.

d. Memory Management

- The JVM handles memory allocation and deallocation automatically via its **Garbage Collector**. This reduces memory leaks and simplifies development since programmers don't need to manually manage memory.