OOPJ Assignment 01

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, and platform-independent** programming language developed by **Sun Microsystems** (now owned by **Oracle**) in 1995. It is designed to be **secure, portable, and robust**, making it a preferred choice for various applications, including **web development, mobile applications, enterprise software, and cloud computing**.

🔹 Significance of Java in Modern Software Development:

- **Cross-Platform Compatibility** → Write Once, Run Anywhere (**WORA**) principle.
- **Scalability** → Used in large-scale enterprise applications.
- **Security** → Provides built-in security features like bytecode verification and access control.
- **Multithreading** → Efficient for concurrent programming.
- **Community Support** → One of the largest developer communities.

---

2. List and Explain the Key Features of Java.

Java is widely used because of its rich set of features. Some of the key features include:

✅ Key Features of Java:

1. **Platform Independence** → Java code runs on any OS using the **Java Virtual Machine (JVM)**.
2. **Object-Oriented** → Java follows OOP principles like **encapsulation, inheritance, and polymorphism**.
3. **Robust and Secure** → Java includes exception handling and security mechanisms.
4. **Multithreading** → Java supports multiple threads for parallel execution.
5. **Automatic Memory Management (Garbage Collection)** → No need for manual memory allocation/deallocation.
6. **High Performance** → **JIT (Just-In-Time) Compiler** improves execution speed.

7. **Distributed Computing** → Supports networking and remote method invocation (**RMI**).

3. What is the Difference Between Compiled and Interpreted Languages? Where Does Java Fit In?

Compiled Languages	Interpreted Languages
Converts the entire source code into machine code before execution.	Converts and executes code line-by-line at runtime.
Faster execution.	Slower execution due to real-time interpretation.
Examples: **C, C++**.	Examples: **Python, JavaScript**.

🔹 Where Does Java Fit In?

Java is **both compiled and interpreted**:

- **Compiled** → Java code is first compiled into **bytecode** using the `javac` compiler.
- **Interpreted** → The **JVM (Java Virtual Machine)** interprets the bytecode and runs it on any OS.

Thus, Java achieves a **balance between speed and portability**.

4. Explain the Concept of Platform Independence in Java.

Platform independence means that **Java programs can run on any operating system (Windows, Linux, Mac) without modification**.

🔹 How Java Achieves Platform Independence:

1. Java code is compiled into **bytecode** (`.class` file) instead of machine code.
2. This **bytecode** runs on any system with a **Java Virtual Machine (JVM)**.
3. Since the JVM is available for different operating systems, Java programs can run anywhere.

💡 **Write Once, Run Anywhere (WORA)** is the fundamental principle behind Java's platform independence.

5. What Are the Various Applications of Java in the Real World?

Java is used in multiple domains due to its **stability, security, and cross-platform support**.

🔹 Real-World Applications of Java:

1. **Web Development** → Used in frameworks like **Spring Boot, JSP, and Servlets**.
2. **Mobile Applications** → **Android development** is powered by Java.
3. **Enterprise Software** → Java is widely used in **banking, healthcare, and ERP systems**.
4. **Cloud Computing** → Java is used for scalable cloud-based solutions.
5. **Big Data & Machine Learning** → Technologies like **Hadoop, Apache Spark** support Java.
6. **Game Development** → Java is used in game engines like **LibGDX and jMonkeyEngine**.
7. **Embedded Systems & IoT** → Java runs on smart devices, routers, and microcontrollers.

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 **1991**. It was officially released on **May 23, 1995**. Java was designed to be a versatile, object-oriented, and platform-independent programming language.

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

Initially, Java was called **Oak**, named after an oak tree outside James Gosling's office. The name was later changed to **Java** because "Oak" was already a registered trademark. The name "Java" was inspired by Java coffee, symbolizing energy and enthusiasm.

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

Java has undergone significant changes over the years, with each version introducing new features and improvements. Below is a brief overview of Java's evolution:

- **Java 1.0 (1995)**: Initial release with basic object-oriented programming features.
- **Java 1.2 (1998)**: Introduction of the Swing GUI toolkit and Collections framework.
- **Java 5 (2004)**: Added generics, enhanced for-loop, auto-boxing, and annotations.

- **Java 8 (2014)**: Introduced lambda expressions, Streams API, and a new Date-Time API.
- **Java 11 (2018, LTS)**: Removed older APIs, introduced HTTP Client API, and enhanced performance.
- **Java 17 (2021, LTS)**: Added sealed classes, strong encapsulation, and new language improvements.
- **Java 21 (2023, LTS)**: Introduced virtual threads, sequenced collections, and improved garbage collection.

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

Recent Java versions have focused on **performance, security, and usability enhancements**. Some major improvements include:

- **Java 8**: Functional programming (Lambda expressions, Streams API), Default methods in interfaces.
- **Java 11**: Long-Term Support (LTS), improved garbage collection, HTTP Client API.
- **Java 17**: Sealed classes, new macOS rendering pipeline, strong encapsulation.
- **Java 21**: Virtual threads for better concurrency, sequenced collections, pattern matching enhancements.

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

Java, **C++**, and **Python** each have unique strengths, making them suitable for different applications. Below is a comparison:

Feature	Java	C++	Python
Performance	Moderate (JIT Compilation)	Fast (Compiled)	Slower (Interpreted)
Ease of Use	Moderate	Complex (Manual Memory Management)	Very Easy (Dynamic Typing)
Memory Management	Automatic (Garbage Collection)	Manual (New/Delete)	Automatic (Garbage Collection)
Platform Independence	Yes (JVM)	No	Yes (Interpreted)
Use Cases	Web, Enterprise, Mobile Apps	System Programming, Game Development	AI, Data Science, Web

Evolution Trends:

- **Java**: Balances performance and security, widely used in **enterprise and cloud computing**.
- **C++**: Offers **high performance and control**, mainly used for **system-level programming**.
- **Python**: Grows rapidly in **AI, data science, and scripting** due to its simplicity.

Part 3: Data Types in Java

1. Explain the importance of data types in Java.

Data types in Java define the **type of values** that a variable can store. They are essential for **memory management, data integrity, and type safety**.

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

- **Primitive Data Types**: Basic types such as `int`, `char`, and `boolean`. Stored directly in memory.
- **Non-Primitive Data Types**: Objects, arrays, and classes. They reference memory locations instead of storing values directly.

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

Java has eight primitive data types:

Data Type	Size	Default Value	Description
`byte`	1 byte	0	Stores small integers (-128 to 127)
`short`	2 bytes	0	Stores larger integers (-32,768 to 32,767)
`int`	4 bytes	0	Default integer type (-2^31 to 2^31-1)
`long`	8 bytes	0L	Large integer values (-2^63 to 2^63-1)
`float`	4 bytes	0.0f	Single-precision decimal numbers
`double`	8 bytes	0.0d	Double-precision decimal numbers
`char`	2 bytes	'\u0000'	Stores single characters (Unicode)
`boolean`	1 bit	false	Represents true/false values

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

```
int number = 10;
float pi = 3.14f;
char letter = 'A';
boolean isJavaFun = true;
```

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

Type casting is converting one data type into another. It is categorized as:

- **Implicit Casting (Widening Conversion)**: Automatic conversion from smaller to larger types.
- **Explicit Casting (Narrowing Conversion)**: Manual conversion from larger to smaller types.

Example:

```
int num = 100;
double doubleNum = num; // Implicit casting

double decimal = 99.99;
int intNum = (int) decimal; // Explicit casting
```

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

Wrapper classes allow primitive types to be used as objects. Examples include `Integer`, `Double`, and `Boolean`.

Example:

```
int num = 10;
Integer obj = Integer.valueOf(num); // Wrapping
int newNum = obj.intValue(); // Unwrapping
```

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

- **Static Typing**: Variable types are checked at **compile-time** (Java, C++).
- **Dynamic Typing**: Types are checked at **runtime** (Python, JavaScript).

Java is **statically typed**, meaning all variables must be declared with a specific type before use.

Coding Questions on Data Types:

// 1. Declare and Initialize All Primitive Data Types
Public class PrimitiveDataTypes {
Public static void main (String[] args) {
Byte b = 10;
Short s = 100;
Int i = 1000;
Long l = 100000 L;
Float f = 10.5 f;
Double d = 99.99;
Char c = 'A';
Boolean bool = true;

```
    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);
}
```

}

~~~

```java

// 2. Perform All Arithmetic Operations on Two Integers
import java.util.Scanner;

public 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();

        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));

        sc.close();
    }
}
```

```
// 3. Demonstrate Implicit and Explicit Type Casting
public class TypeCasting {
    public static void main(String[] args) {
        // Implicit Type Casting (Widening)
        int intVal = 100;
        double doubleVal = intVal;
        System.out.println("Implicit Casting (int to double): " +
doubleVal);

        // Explicit Type Casting (Narrowing)
        double d = 99.99;
        int i = (int) d;
        System.out.println("Explicit Casting (double to int): " + i);
    }
}
```

```
// 4. Convert Integer to Double and Vice Versa Using Wrapper Classes
public class WrapperConversion {
    public static void main(String[] args) {
        Integer intObj = 50;
        Double doubleObj = intObj.doubleValue(); // Convert Integer to
Double
        System.out.println("Integer to Double: " + doubleObj);

        Double doubleVal = 75.5;
        Integer intVal = doubleVal.intValue(); // Convert Double to Integer
        System.out.println("Double to Integer: " + intVal);
    }
}
```

```
// 5. Swap Two Numbers Using and Without Using a Temporary Variable
import java.util.Scanner;

public 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();

        // Swapping using a temporary variable
        int temp = a;
        a = b;
        b = temp;
        System.out.println("After swapping (using temp variable): a = " + a + ", b = " + b);

        // Swapping 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);

        sc.close();
    }
}
```

```
// 6. Check Whether a Character is a Vowel or Consonant
import java.util.Scanner;

public class VowelOrConsonant {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a character: ");
        char ch = sc.next().toLowerCase().charAt(0);

        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            System.out.println(ch + " is a vowel.");
        } else if (Character.isLetter(ch)) {
            System.out.println(ch + " is a consonant.");
        } else {
            System.out.println("Invalid input. Please enter a letter.");
        }

        sc.close();
    }
}
```

```
// 7. Check 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;
        }

        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?

The **Java Development Kit (JDK)** is a software development kit that includes everything needed to develop and run Java applications. It differs from:

- **JVM (Java Virtual Machine)**: Runs Java bytecode but does not include development tools.
- **JRE (Java Runtime Environment)**: Includes the JVM and libraries to run Java programs but lacks development tools.

2. Explain the main components of JDK.

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

- **Java Runtime (** `**java**` **)**: Executes Java applications.
- **Java Libraries**: Pre-built classes and functions.
- **Debugger (** `**jdb**` **)**: Helps debug Java programs.
- **Javadoc Tool (** `**javadoc**` **)**: Generates documentation from comments.

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

1. Download the latest JDK from the official Oracle/OpenJDK website.
2. Install it following the on-screen instructions.
3. Set the `PATH` variable to include the `bin` directory.
4. Verify installation using:

    ```
    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!");
    }
}
```

- `public class HelloWorld` : Defines the class.
- `public static void main(String[] args)` : Entry point of the program.
- `System.out.println()` : Prints output.

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

- **PATH**: Specifies directories containing executable files like `javac` and `java` .
- **CLASSPATH**: Specifies directories containing Java class files and libraries.

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

Feature	OpenJDK	Oracle JDK

License	Open-source (GPL)	Commercial
Support	Community-driven	Oracle support
Updates	Frequent	LTS versions available

7. Explain how Java programs are compiled and executed.

1. **Write** Java code.
2. **Compile** using `javac` to generate bytecode (`.class` file).
3. **Execute** using `java`, which runs it on the JVM.

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

JIT compilation converts bytecode into native machine code **at runtime**, improving execution speed by optimizing frequently used code.

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

The **JVM** runs Java applications by:

- **Interpreting bytecode**.
- **Managing memory** with garbage collection.
- **Ensuring security** through sandboxing.
