Assignment 1

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 Oracle) in 1995. It is widely used in web development, mobile applications, enterprise software, cloud computing, and IoT.

Significance:

- Platform Independence (WORA - Write Once, Run Anywhere)
- Security through JVM and memory management
- Scalability for enterprise applications
- Multithreading Support for high-performance applications
- Robust Ecosystem with frameworks like Spring, Hibernate, Jakarta EE

---

2. Key Features of Java:

1. Simple & Easy to Learn – Similar to C++, but removes complex features like pointers.
2. Object-Oriented – Supports OOP principles (Encapsulation, Inheritance, Polymorphism, Abstraction).
3. Platform-Independent – Runs on any OS via JVM.
4. Secure – Provides built-in security mechanisms.
5. Robust – Automatic memory management and exception handling.
6. Multithreaded – Supports parallel processing.
7. Distributed Computing – Supports networking and remote method invocation.
8. High Performance – Uses Just-In-Time (JIT) compiler for faster execution.

---

3. Compiled vs. Interpreted Languages & Where Java Fits

Feature	Compiled Languages (C, C++)	Interpreted Languages (Python, JS)	Java
Execution	Converts code to machine code	Executes line-by-line	Uses both (Compiles to Bytecode & runs on JVM)
Speed	Fast	Slower	Optimized with JIT Compiler
Portability	OS-dependent	OS-independent	Platform-independent

Java is hybrid – it first compiles code to Bytecode, then JVM interprets and executes it.

---

4. Platform Independence in Java

Java follows Write Once, Run Anywhere (WORA) principle.

1. Java compiler converts source code (`.java`) into Bytecode (`.class`).
2. Bytecode is executed by the JVM (Java Virtual Machine), which is OS-specific.
3. This allows Java programs to run on Windows, Linux, macOS, and Android without modification.

---

5. Real-World Applications of Java

1. Web Development – Spring Boot, JSP (e.g., Amazon, LinkedIn)
2. Mobile Applications – Android apps (e.g., WhatsApp, Instagram)
3. Enterprise Applications – Banking & ERP Systems
4. Big Data & Analytics – Apache Hadoop, Apache Spark
5. Cloud Computing – AWS, Google Cloud, Azure
6. IoT (Internet of Things) – Smart home devices, automation systems
7. Game Development – LibGDX, jMonkeyEngine
8. Scientific Computing – Bioinformatics, AI research

Java Programming Basics: 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 in 1995. The main goal was to create a portable, platform-independent language for embedded systems.

---

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

Java was initially called "Oak", named after an oak tree outside James Gosling’s office. Later, it was renamed Java because Oak was already trademarked. The new name "Java" was inspired by Java coffee, symbolizing energy and speed.

---

3. Evolution of Java Versions

Version	Release Year	Key Features
Java 1.0	1996	First official release, Applets introduced
Java 2 (J2SE 1.2-1.4)	1998-2002	Swing, Collections framework, JVM improvements
Java 5	2004	Generics, Enhanced for-loop, Auto-boxing
Java 6	2006	Performance improvements, Scripting API
Java 7	2011	Try-with-resources, Diamond operator
Java 8	2014	Lambda expressions, Streams API, Functional programming
Java 9	2017	JPMS (Java Module System), JShell (REPL)
Java 10-12	2018-2019	Local variable type inference (var)
Java 13-16	2019-2021	Pattern Matching, Sealed Classes
Java 17 (LTS)	2021	Improved performance, new garbage collector
Java 21 (LTS)	2023	Virtual Threads, Record Patterns, Performance Enhancements

---

4. Major Improvements in Recent Java Versions

1. Java 8 (2014) – Lambda expressions, Streams API, Functional programming
2. Java 9 (2017) – JPMS (Module System), JShell
3. Java 10-11 (2018-2019) – `var` keyword, New Garbage Collector
4. Java 14-16 (2020-2021) – Pattern Matching, Records, Sealed Classes
5. Java 17 (2021 - LTS) – Improved Garbage Collection, Performance Boost
6. Java 21 (2023 - LTS) – Virtual Threads, Structured Concurrency, Generational GC

---

5. Java vs. C++ vs. Python: Evolution & Usability

Feature	Java	C++	Python
Paradigm	Object-Oriented, Platform-Independent	Object-Oriented, Performance-Oriented	Multi-Paradigm, Easy-to-Learn
Memory Management	Automatic (Garbage Collector)	Manual (Developers handle memory)	Automatic
Performance	Fast (JIT Compiler)	Very Fast (Compiled)	Slower (Interpreted)
Ease of Use	Moderate	Complex (Pointers, Manual Memory)	Very Easy
Application Areas	Web, Enterprise, Mobile, Cloud	System Software, Gaming, Performance-Critical Apps	AI, Data Science, Automation

Conclusion:

- Java balances performance and simplicity, making it ideal for large applications.
- C++ is best for performance-intensive tasks (OS, Game Engines).
- Python is preferred for AI, Data Science, and automation due to its simplicity.

Java Programming Basics: Part 3 - Data Types in Java

1. Importance of Data Types in Java

Data types define what kind of data a variable can store, ensuring efficient memory usage and preventing errors. Java is a strongly typed language, meaning all variables must have a defined type.

Key Benefits:

- Ensures type safety (prevents incorrect data operations)
- Optimizes memory allocation
- Improves program reliability and performance

---

2. Primitive vs. Non-Primitive Data Types

Feature	Primitive Data Types	Non-Primitive Data Types
Definition	Predefined by Java	Defined by users
Memory Size	Fixed size	Varies based on object
Stored In	Stack memory	Heap memory
Operations	Directly stores values	Stores references to objects
Examples	`int`, `char`, `double`, `boolean`	`String`, `Array`, `Class`, `Interface`

---

3. Eight Primitive Data Types in Java

Data Type	Size	Default Value	Description
byte	1 byte	0	Smallest integer type (-128 to 127)
short	2 bytes	0	Larger than byte, used in legacy apps
int	4 bytes	0	Default for whole numbers (-2B to +2B)
long	8 bytes	0L	Used for very large numbers
float	4 bytes	0.0f	Single-precision decimal numbers
double	8 bytes	0.0d	High-precision decimal numbers
char	2 bytes	'\u0000'	Stores a single character (Unicode)
boolean	1 bit	false	Stores `true` or `false` values

---

4. Declaring and Initializing Data Types

```
// Primitive Data Types
```

```
int age = 25;

double price = 199.99;

char grade = 'A';

boolean isJavaFun = true;


// Non-Primitive Data Types

String name = "John";

int[] numbers = {10, 20, 30};
```

---

5. Type Casting in Java

Type casting is converting one data type into another.

Types:

- Widening (Implicit Casting) – Smaller to larger type (automatic)
- Narrowing (Explicit Casting) – Larger to smaller type (manual)

```
// Implicit Casting (Widening)

int num = 100;

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


// Explicit Casting (Narrowing)

double price = 99.99;

int roundedPrice = (int) price;  // double to int (manual)
```

---

6. Wrapper Classes in Java

Wrapper classes convert primitive types into objects.

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Usage Example:

```
// Autoboxing (Primitive to Object)
int num = 10;
```

```
Integer obj = num;


// Unboxing (Object to Primitive)

Integer obj2 = 20;

int num2 = obj2;
```

Why Use Wrapper Classes?

- Needed in Collections (ArrayList, HashMap, etc.)
- Allows null values (primitives cannot be null)
- Provides utility methods (e.g., `Integer.parseInt()`)

---

7. Static vs. Dynamic Typing & Java's Position

Feature	Static Typing (Java, C, C++)	Dynamic Typing (Python, JavaScript)
Type Checking	At compile-time	At runtime
Flexibility	Less flexible but safer	More flexible, error-prone
Performance	Faster	Slower due to runtime type checking

Java is a statically typed language, meaning type checking happens at compile-time, which ensures better performance and fewer runtime errors.

1. Coding Questions on Data Types

Q1.Declare and Initialize All Primitive Data Types

```
public class PrimitiveDataTypes {
    public static void main(String[] args) {
        byte b = 10;
        short s = 1000;
        int i = 100000;
        long l = 1000000000L;
        float f = 10.5f;
        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);
    }
}
```

Q2. 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 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();
    }
}
```

Q3.implement.Java program to demonstrate implicit and explicit type casting.

```
public class TypeCastingDemo {
```

```
    public static void main(String[] args){
        int num = 100;
        double d = num;
        System.out.println("Implicit Casting (int to double): " + d);
        double price = 99.99;
        int intPrice = (int) price;
        System.out.println("Explicit Casting (double to int): " + intPrice);
    }
}
```

4. Create a Java program that converts a given integer to a 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

        Double d = 25.5;
        Integer intVal = d.intValue(); // Convert Double to Integer

        System.out.println("Integer to Double: " + doubleObj);
        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 SwapNumbers {
  public static void main(String[] args) {
    int a = 5, b = 10;



    System.out.println("Before Swap: a = " + a + ", b = " + b);
    int temp = a;
    a = b;
    b = temp;
    System.out.println("After Swap (Using temp): a = " + a + ", b = " + b);


    a = 5; b = 10;
    System.out.println("\nBefore Swap: a = " + a + ", b = " + b);
    a = a + b;
    b = a - b;
    a = a - b;
    System.out.println("After Swap (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 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 (ch >= 'a' && ch <= 'z') {
            System.out.println(ch + " is a Consonant.");
        } else {
            System.out.println("Invalid input! Please enter a letter.");
        }

        sc.close();
    }
```

```
}
```

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

```
public class EvenOddCheck {

    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.");

        }

    }

}
```

C