Java Assignment-1 (7PM BATCH)

Md salick rahman

Ques. 1. When was java developer and who is the inventor of java? Why we use java for programming?

Ques. 2. What are the features of java?

Ques. 3. what is JDK, JRE, JVM?

Ques. 4. what is the structure of java program. write a program to print hello world.

Ques. 5. Define Variables in java. Explain different scope of variable.

Ques. 6. Define data types in java.

Ques. 7. What is type casting in java?

Ques. 8. Write a program to add two number by taking input from user.

Ques. 9. Define operator and its type?

Ques. 10. Write a program individually to perform operators' operation.

Solution: -

Ques. 1. What was java developer and who is the inventor of java? Why we use java for programming?

Sol: - A Java developer is a software engineer or programmer who specializes in developing applications and software using the Java programming language. Java developers work on various types of projects, from building enterprise-level applications to developing mobile apps, web applications, and even games. Their responsibilities often include designing, coding, testing, and maintaining Java-based applications. Java was invented by James Gosling and his team at Sun Microsystems (now part of Oracle Corporation). The language was originally developed in the early 1990s, with the first public release in 1995. James Gosling is often referred to as the "father of Java" due to his pivotal role in its creation.

Uses of java for programming: -

1) **Platform Independence (Write Once, Run Anywhere)**
2) Object-Oriented Programming
3) Rich Standard Library
4) Security Features
5) Multithreading
6) Robustness and Reliability
7) Large Developer Community and Ecosystem
8) Enterprise Applications
9) Android Development

Ques. 2. What are the features of java?

Sol: -

1) Platform Independence: Write Once**,** Run Anywhere: Java programs are compiled into bytecode, which can be executed on any device or operating system that has a Java Virtual Machine (JVM).
2) Object-Oriented: Java is an object-oriented programming language, meaning it organizes software design around data, or objects, rather than functions and logic.
3) Simple: Java's syntax is designed to be easy to learn and use, especially for programmers who are familiar with C or C++.
4) Secure: Java provides a secure environment for developing and running applications.
5) Robust: Java is designed to be robust, with strong memory management, automatic garbage collection, and exception handling.
6) Multithreaded: Java has built-in support for multithreading, which allows multiple threads to run concurrently, making it easier to build applications that perform many tasks at once, such as real-time simulations, video games, and multimedia processing.
7) Portable: Java's platform independence and its ability to run on any system that supports the JVM make it highly portable.
8) High Performance: While Java is interpreted, it achieves high performance using Just-In-Time (JIT) compilers that convert bytecode into native machine code during runtime.
9) Distributed: Java is designed for the distributed environment of the internet. It provides built-in support for networked applications through libraries like `java.net` for TCP/IP protocols and `Remote Method Invocation (RMI)` for calling methods across the network.

10) Dynamic: Java is dynamic in nature, capable of loading classes at runtime that are not known until execution.
11) Architecture-Neutral: Java's bytecode is architecture-neutral, meaning it can be executed on any processor or operating system that has a JVM.
12) Automatic Garbage Collection: Java automatically handles memory deallocation using its garbage collector, which frees up memory that is no longer in use, reducing the likelihood of memory leaks and other memory-related issues.
13) Rich Standard Library: ava comes with a comprehensive standard library (Java API) that provides a wide range of classes and methods for everything from data structures, mathematics, and networking to graphical user interfaces (GUIs), making it easier to develop applications.
14) Scalability: Java is suitable for building both small-scale and large-scale applications. Its design and features allow developers to create scalable systems, making it a popular choice for enterprise-level software.

Ques. 3. what is JDK, JRE, JVM?

Sol: -

1. JVM (Java Virtual Machine)

JVM is an abstract computing machine or virtual machine that enables a computer to run Java programs. It interprets the compiled Java bytecode and executes it on the underlying hardware.

2. JRE (Java Runtime Environment)

JRE is a package that provides the necessary environment to run Java applications. It includes the JVM, core libraries, and other components required for running Java programs.

3. JDK (Java Development Kit)

JDK is a full-featured software development kit used for developing Java applications. It includes the JRE, along with additional tools and utilities required for Java development.

Ques. 4. what is the structure of java program? write a program to print hello world.

Sol: -

The structure of a Java program typically includes the following key components:

1. **Package declaration** (if applicable)- package com.example;
2. **Import statements** (if needed)- import java.util.Scanner;
3. **Class declaration** (required)-

   public class HelloWorld {

   // Class body

   }

4. **Main method** (required for execution)-

   public static void main(String[] args) {

   // Code to be executed

   }

5. **Statements and logic** (inside the main method)- System.out.println("Hello, World!");

   Program to print hello word

```java
package java_assignment_1;

public class print_hello_word {

	public static void main(String[] args) {

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

	}

}
```

Ques. 5. Define Variables in java. Explain different scope of variable.

Sol: - Variables - In Java, variables are containers used to store data values. Each variable in Java has a specific data type that determines the kind of data it can hold, such as integers, floating-point numbers, characters, or strings.

Types of Variables in Java:

1. **Local Variables**:
   - These are declared inside a method, constructor, or block.
   - Local variables are only accessible within the method or block in which they are declared.
   - They do not have a default value, so they must be initialized before use.

2. **Instance Variables (Non-Static Fields)**:
    - These are declared inside a class but outside any method, constructor, or block.
    - Each instance of the class (object) has its own copy of the instance variables.
    - Instance variables have default values if not explicitly initialized (e.g., `0` for int, `null` for objects).

3. **Class Variables (Static Fields)**:
    - These are declared with the `static` keyword inside a class, but outside any method, constructor, or block.
    - Class variables are shared among all instances of the class, meaning there is only one copy of each class variable.
    - Like instance variables, class variables also have default values.

Scope of Variables in Java

The scope of a variable refers to the region of the program where the variable is accessible.

1. **Local Variable Scope**:
    - The scope of a local variable is limited to the method, constructor, or block in which it is declared.
    - Once the method or block exits, the local variable is no longer accessible and is removed from memory.

2. **Instance Variable Scope**:
    - Instance variables are accessible by all methods, constructors, and blocks within the class.
    - They exist if the object of the class exists and are destroyed when the object is destroyed.

3. **Class Variable Scope**:
    - Class variables (static variables) are accessible from any method, constructor, or block in the class.
    - They exist for the duration of the program and are shared across all instances of the class.

Ques. 6. Define data types in java.

Sol: - In Java, data types specify the size and type of values that can be stored in a variable. Java has two categories of data types:

1. **Primitive Data Types**
2. **Reference (Non-Primitive) Data Types**

1. Primitive Data Types

Primitive data types are the most basic data types built into the Java language. There are eight primitive data types:

- **byte**
  - Size: 8-bit
  - Range: -128 to 127
  - Use: To save memory in large arrays, where the memory savings is most needed.
- **short**
  - Size: 16-bit
  - Range: -32,768 to 32,767
  - Use: Like `byte` but can store larger values.
- **int**
  - Size: 32-bit
  - Range: -2^31 to 2^31-1
  - Use: Default data type for integer values.
- **long**
  - Size: 64-bit
  - Range: -2^63 to 2^63-1
  - Use: When a wider range than `int` is needed.
- **float**
  - Size: 32-bit
  - Range: Varies (up to 7 decimal digits)
  - Use: For single-precision floating-point numbers.
- **double**
  - Size: 64-bit
  - Range: Varies (up to 15 decimal digits)
  - Use: For double-precision floating-point numbers (default for decimal values).
- **char**
  - Size: 16-bit
  - Range: 0 to 65,535 (represents a single 16-bit Unicode character)
  - Use: For storing characters.
- **Boolean**
  - Size: 1-bit (not precisely defined, but typically treated as a single bit)
  - Values: `true` or `false`
  - Use: For simple flags that track true/false conditions.

2. Reference (Non-Primitive) Data Types

Reference types are more complex and can refer to objects or arrays. They include:

- **Classes**
  - A blueprint for creating objects. Example: `String`, `Scanner`, or any custom class.
- **Interfaces**
  - A reference type like a class that can contain only constants, method signatures, default methods, static methods, and nested types.
- **Arrays**
  - A collection of elements of the same type. Example: `int []`, `String []`.
- **Enums**
  - Special classes that represent a group of constants (unchangeable variables, like `final` variables). Example: `Enum Days {MONDAY, TUESDAY, ...}`.

Ques. 7. What is type casting in java?

Sol: - Type casting in Java is the process of converting one data type into another. It can be broadly classified into two types:

1. **Implicit (Automatic) Type Casting**
2. **Explicit Type Casting**

1. Implicit (Automatic) Type Casting

Implicit type casting occurs when a smaller data type is converted into a larger data type automatically by the Java compiler. This is also known as **widening** or **upcasting**. It happens when there is no risk of losing data during the conversion.

For example:

- `byte` to `short`
- `short` to `int`
- `int` to `long`
- `float` to `double`

2. Explicit Type Casting

- Explicit type casting is required when you are converting a larger data type into a smaller data type, also known as **narrowing** or **down casting**. Since this process can lead to data loss, the programmer must specify the type explicitly.

Ques. 8. Write a program to add two number by taking input from user.

Sol: -

```java
package java_assignment_1;

import java.util.Scanner;

public class Add_two_numbers_by_taking_input {

	public static void main(String[] args) {

		Scanner obj= new Scanner(System.in);
		int a,b;
		System.out.println("Enter two numbers");
			a=obj.nextInt();
			b=obj.nextInt();

			int c=a+b;


			System.out.println(c);


	}

}
```

Ques. 9. Define operator and its type?

Sol: - Operators in Java are special symbols or keywords that are used to perform operations on operands (variables or values). Java supports a variety of operators, categorized into several types based on their function and usage:

1. Arithmetic Operators

Arithmetic operators are used to perform basic arithmetic operations.

- **Addition (+)**: Adds two operands.
  - Example: `a + b`
- **Subtraction (-)**: Subtracts the second operand from the first.
  - Example: `a - b`
- **Multiplication (*)**: Multiplies two operands.
  - Example: `a * b`
- **Division (/)**: Divides the numerator by the denominator.
  - Example: `a / b`
- **Modulus (%)**: Returns the remainder of a division.
  - Example: `a % b`

2. Relational Operators

Relational operators are used to compare two values or expressions.

- **Equal to (==)**: Checks if two operands are equal.
  - Example: `a == b`
- **Not equal to (! =)**: Checks if two operands are not equal.
  - Example: `a! = b`
- **Greater than (>)**: Checks if the first operand is greater than the second.
  - Example: `a > b`
- **Less than (<)**: Checks if the first operand is less than the second.
  - Example: `a < b`
- **Greater than or equal to (>=)**: Checks if the first operand is greater than or equal to the second.
  - Example: `a >= b`
- **Less than or equal to (<=)**: Checks if the first operand is less than or equal to the second.
  - Example: `a <= b`

3. Logical Operators

Logical operators are used to perform logical operations and are commonly used in conditional statements.

- **Logical AND (&&)**: Returns `true` if both operands are `true`.
  - Example: `a && b`
- **Logical OR (||)**: Returns `true` if at least one of the operands is `true`.
  - Example: `a || b`
- **Logical NOT (!)**: Returns `true` if the operand is `false`.
  - Example: !a

4. Assignment Operators

Assignment operators are used to assign values to variables.

- **Simple Assignment (=)**: Assigns a value to a variable.
  - Example: `a = b`
- **Add and Assign (+=)**: Adds the right operand to the left operand and assigns the result to the left operand.
  - Example: `a += b` (equivalent to `a = a + b`)
- **Subtract and Assign (-=)**: Subtracts the right operand from the left operand and assigns the result to the left operand.
  - Example: `a -= b` (equivalent to `a = a - b`)
- **Multiply and Assign (*=)**: Multiplies the left operand by the right operand and assigns the result to the left operand.
  - Example: `a *= b` (equivalent to `a = a * b`)
- **Divide and Assign (/=)**: Divides the left operand by the right operand and assigns the result to the left operand.
  - Example: `a /= b` (equivalent to `a = a / b`)
- **Modulus and Assign (%=)**: Applies modulus and assigns the result to the left operand.
  - Example: `a %= b` (equivalent to `a = a % b`)

5. Unary Operators

Unary operators operate on a single operand.

- **Unary Plus (+)**: Indicates a positive value.
  - Example: `+a`
- **Unary Minus (-)**: Negates the value.
  - Example: `-a`
- **Increment (++)**: Increases the value of a variable by 1.
  - Example: `a++` or `++a`
- **Decrement (--)**: Decreases the value of a variable by 1.
  - Example: `a--` or `--a`
- **Logical NOT (!)**: Inverts the Boolean value.
  - Example: `!a`

6. Bitwise Operators

Bitwise operators are used to perform operations on binary representations of integers.

- **AND (&)**: Performs a bitwise AND operation.
  - Example: `a & b`
- **OR (|)**: Performs a bitwise OR operation.
  - Example: `a | b`
- **XOR (^)**: Performs a bitwise XOR operation.
  - Example: `a ^ b`
- **Complement (~)**: Inverts all the bits of the operand.
  - Example: `~a`
- **Left Shift (<<)**: Shifts the bits of the left operand to the left by the number of positions specified by the right operand.
  - Example: `a << b`
- **Right Shift (>>)**: Shifts the bits of the left operand to the right by the number of positions specified by the right operand.
  - Example: `a >> b`
- **Unsigned Right Shift (>>>)**: Shifts the bits of the left operand to the right by the number of positions specified by the right operand, filling the leftmost bits with zeros.
  - Example: `a >>> b`

7. Ternary Operator

The ternary operator is a shorthand for an `if-else` statement.

- **Ternary Operator (? :)**: Returns one of two values based on a condition.
  - Example: `condition? value1: value2`
  - If `condition` is `true`, `value1` is returned; otherwise, `value2` is returned.

Ques. 10. Write a program individually to perform operators' operation.

Sol: -

Arithmetic operators

```
1. package java_assignment_1;
2.
3. public class arithmatic_operators {
4.
5.        public static void main(String[] args) {
6.
7.              int a = 10, b = 5;
8.
9.              System.out.println("a + b = " + (a + b));
10.         System.out.println("a - b = " + (a - b));
11.         System.out.println("a * b = " + (a * b));
12.         System.out.println("a / b = " + (a / b));
13.         System.out.println("a % b = " + (a % b));
14.       }
15.
16.}
```

Relational Operators

```
package java_assignment_1;

public class Relational_operators {

      public static void main(String[] args) {

            int a = 20, b = 5;

            System.out.println("a == b: " + (a == b));
        System.out.println("a != b: " + (a != b));
        System.out.println("a > b: " + (a > b));
        System.out.println("a < b: " + (a < b));
      }

}
```

Logical Operators

```
package java_assignment_1;

public class Logical_operators {

      public static void main(String[] args) {


        int a = 5;
        int b = 10;
        int c = 15;


        int sum1 = a + b;
        int sum2 = b + c;
```

```
        // Logical AND (&&)
        System.out.println("sum1 == 15 && sum2 == 25: " + (sum1 == 15 && sum2 ==
25));

        // Logical OR (||)
        System.out.println("sum1 == 20 || sum2 == 25: " + (sum1 == 20 || sum2 ==
25));

        // Logical NOT (!)
        System.out.println("!(sum1 == 15): " + !(sum1 == 15));
        System.out.println("!(sum2 == 20): " + !(sum2 == 20));

        // Logical XOR (^)
        System.out.println("(sum1 == 15) ^ (sum2 == 20): " + ((sum1 == 15) ^ (sum2
== 20)));

      }

}
```

Assignment Operators

```
package java_assignment_1;

public class Assignment_Operators {

      public static void main(String[] args) {

            int x = 10;
       int y = 5;

        // Simple assignment
        x = y;
        System.out.println("x = y: " + x);

        // Add and assign
        x += y;
        System.out.println("x += y: " + x);

        // Subtract and assign
        x -= y;
        System.out.println("x -= y: " + x);

        // Multiply and assign
        x *= y;
        System.out.println("x *= y: " + x);

        // Divide and assign
        x /= y;
        System.out.println("x /= y: " + x);

        // Modulus and assign
        x %= y;
        System.out.println("x %= y: " + x);

      }

}
```

Unary Operators

```java
package java_assignment_1;

public class Unary_operators {

	public static void main(String[] args) {

		int x = 10;
		 int y = -5;

		 // Unary Plus
		 System.out.println("+x: " + (+x));
		 // Unary Minus
		 System.out.println("-y: " + (-y));
		 // Pre-increment
		 System.out.println("++x: " + (++x));

		 // Post-increment
		 System.out.println("x++: " + (x++));
		 System.out.println("x after post-increment: " + x);

		 // Pre-decrement
		 System.out.println("--x: " + (--x));

		 // Post-decrement
		 System.out.println("x--: " + (x--));
		 System.out.println("x after post-decrement: " + x);

		 // Bitwise Complement
		 System.out.println("~y: " + (~y));

	}

}
```

Bitwise Operators

```java
package java_assignment_1;

public class Bitwise_operators {

	public static void main(String[] args) {
		int a = 42;
	  int b = 25;
	  // Bitwise AND
	  System.out.println("a & b = " + (a & b));

	  // Bitwise OR
	  System.out.println("a | b = " + (a | b));

	  // Bitwise XOR
	  System.out.println("a ^ b = " + (a ^ b));

	  // Bitwise NOT
	  System.out.println("~a = " + (~a));

	  // Bitwise Shift Left
```

```
        System.out.println("a << 2 = " + (a << 2));

        // Bitwise Shift Right
        System.out.println("a >> 2 = " + (a >> 2));

        // Bitwise Unsigned Shift Right
        System.out.println("a >>> 2 = " + (a >>> 2));

      }

}
```

Ternary Operator

```
package java_assignment_1;

public class Ternary_Operator {

      public static void main(String[] args) {

            int a = 10;
        int b = 20;


        int max = (a > b) ? a : b;


        System.out.println("The maximum value is: " + max);

      }

}
```