TASK 4: Method Design & Modular Calculator.

1.Modular Programming

Modular programming means dividing a large program into smaller, independent methods. Each method performs a single task such as addition, subtraction, multiplication, or division.

Advantages:

- Improves code readability
- Makes debugging easier
- Allows code reusability
- Follows industry best practices

2. Methods in Java

A method is a block of code that performs a specific operation and is executed when it is called.

Syntax:

```
returnType methodName(parameters) {
   // method body
}
```

Example:

```
static double add(double a, double b) {
   return a + b;
}
```

3. Method Overloading

Method overloading occurs when multiple methods have the same name but different parameter lists.

In this task:

- add(int, int)
- add(double, double)

This is called compile-time polymorphism and improves flexibility and readability.

4. Return Type vs Void

- A method with a return type sends a value back to the caller.
- A void method does not return any value.

Example:

```
return a + b;   // return type
```

```
void printLine() { }  // void method
```

5. Pass-by-Value in Java

Java uses pass-by-value, which means a copy of the variable is passed to the method.
Changes made inside the method do not affect the original variable.

This is demonstrated using the changeValue() method in the program.

6. Exception Handling (Division by Zero)

If division is performed with zero, it causes a runtime error.
To avoid this, exception handling is used.

```
if (b == 0) {

    throw new ArithmeticException("Division by zero is not allowed");

}
```

This makes the program safe and prevents crashing.

7. Utility Methods

Utility methods are reusable methods used multiple times in a program.
In this task, printLine() is a utility method used to improve output formatting.

8. Stack Memory

Stack memory stores:

- Method calls
- Local variables
- Parameters

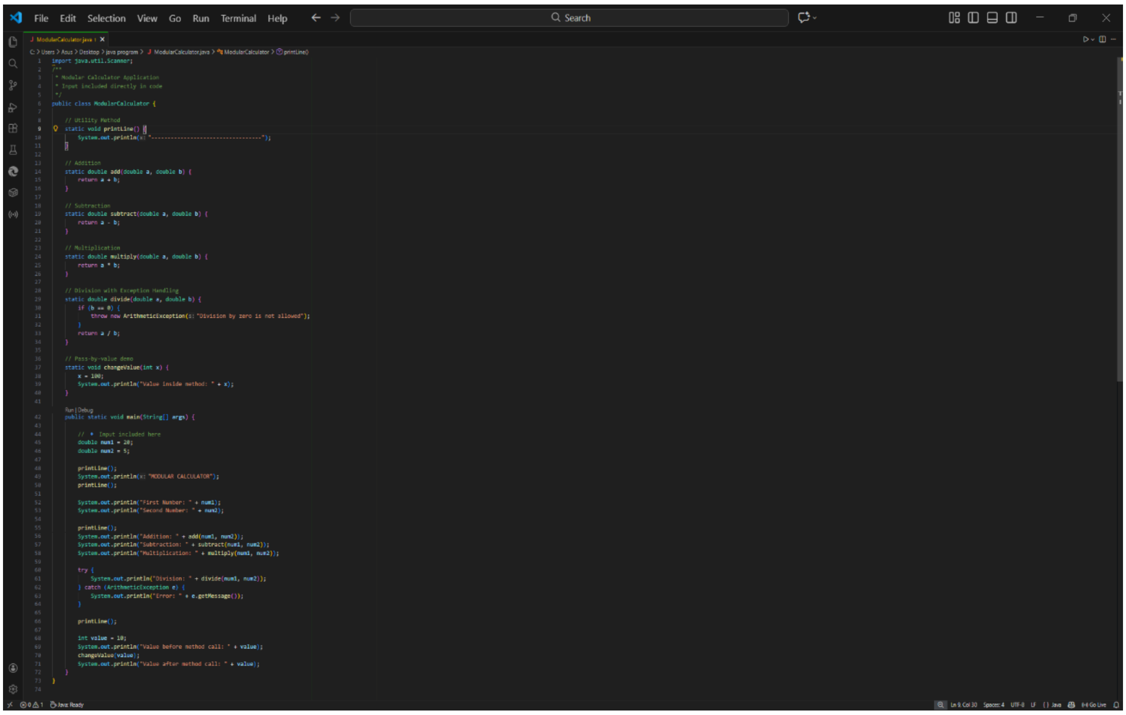
It works on LIFO (Last In First Out) principle and provides fast memory access.

9. Importance of This Task

This task helps in understanding:

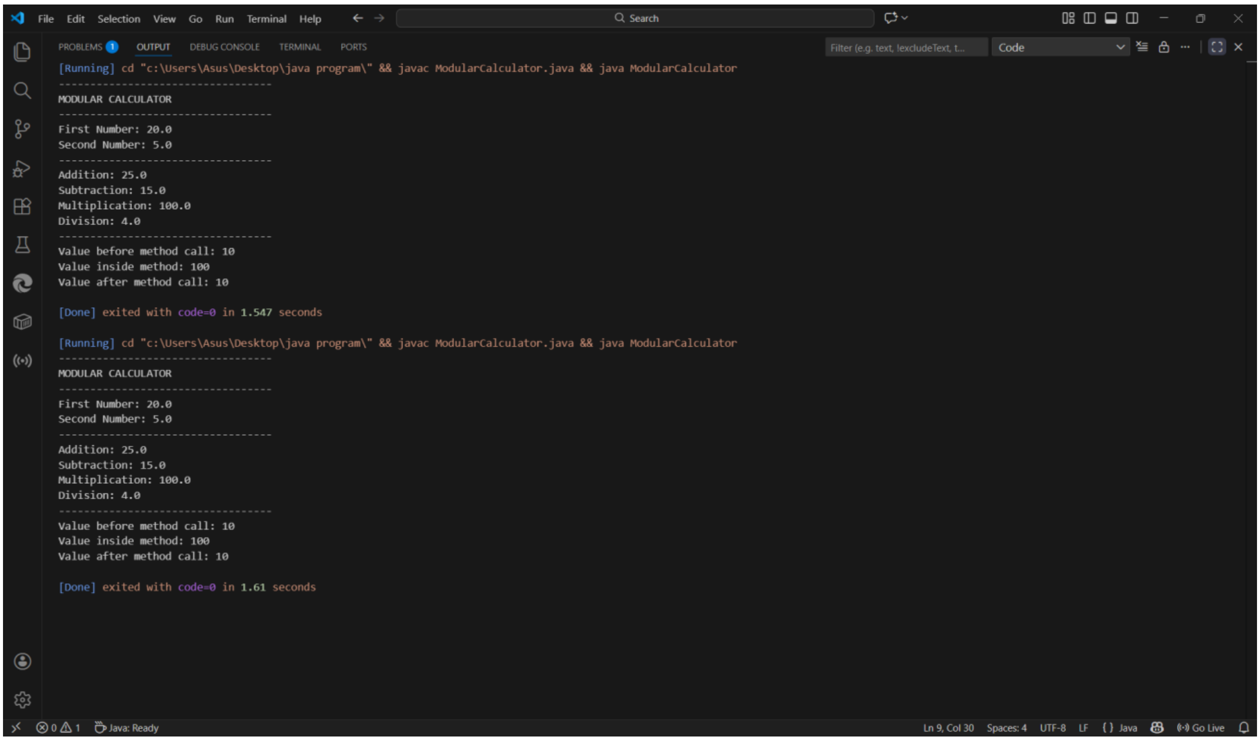
- Core Java concepts
- Clean and modular coding
- Interview-oriented programming practices
- Real-world application structure

Code:-



```java
import java.util.Scanner;
/**
 * Modular Calculator Application
 * Input included directly in code
 */
public class ModularCalculator {

    // Utility Method
    static void printLine() {
        System.out.println("----------------------------------");
    }

    // Addition
    static double add(double a, double b) {
        return a + b;
    }

    // Subtraction
    static double subtract(double a, double b) {
        return a - b;
    }

    // Multiplication
    static double multiply(double a, double b) {
        return a * b;
    }

    // Division with Exception Handling
    static double divide(double a, double b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero is not allowed");
        }
        return a / b;
    }

    // Pass-by-value demo
    static void changeValue(int x) {
        x = 100;
        System.out.println("Value inside method: " + x);
    }

    public static void main(String[] args) {

        // 🔹 Input included here
        double num1 = 20;
        double num2 = 5;

        printLine();
        System.out.println("MODULAR CALCULATOR");
        printLine();

        System.out.println("First Number: " + num1);
        System.out.println("Second Number: " + num2);

        printLine();
        System.out.println("Addition: " + add(num1, num2));
        System.out.println("Subtraction: " + subtract(num1, num2));
        System.out.println("Multiplication: " + multiply(num1, num2));

        try {
            System.out.println("Division: " + divide(num1, num2));
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }

        printLine();

        int value = 10;
        System.out.println("Value before method call: " + value);
        changeValue(value);
        System.out.println("Value after method call: " + value);
    }
}
```

Output:-



```
[Running] cd "c:\Users\Asus\Desktop\java program\" && javac ModularCalculator.java && java ModularCalculator
----------------------------------
MODULAR CALCULATOR
----------------------------------
First Number: 20.0
Second Number: 5.0
----------------------------------
Addition: 25.0
Subtraction: 15.0
Multiplication: 100.0
Division: 4.0
----------------------------------
Value before method call: 10
Value inside method: 100
Value after method call: 10

[Done] exited with code=0 in 1.547 seconds

[Running] cd "c:\Users\Asus\Desktop\java program\" && javac ModularCalculator.java && java ModularCalculator
----------------------------------
MODULAR CALCULATOR
----------------------------------
First Number: 20.0
Second Number: 5.0
----------------------------------
Addition: 25.0
Subtraction: 15.0
Multiplication: 100.0
Division: 4.0
----------------------------------
Value before method call: 10
Value inside method: 100
Value after method call: 10

[Done] exited with code=0 in 1.61 seconds
```