CSE215

Summer 2020

Project Report

JavaCalc: A Basic Calculator Using Java Programming Language

Submitted by

Sumit Saha 1931415042
Raiyan Rahman 1931340042
Md. Iftekharul Alam 1931476642

Section : 05
Date : 30 September 2020

Submitted to

Ziaul Hossain (ZHO)

Senior Lecturer
ECE Department

Table of Contents

1. Introduction

JavaCalc is a user-friendly Java application designed to perform basic arithmetic operations, including addition, subtraction, multiplication, and division. Built using Object-Oriented Programming (OOP) principles, JavaCalc emphasizes modularity and maintainability through its structured class hierarchy. The application features an abstract “Operation”class, from which specific operation classes inherit, ensuring code reusability and clarity. JavaCalc allows users to interact with the program through a simple command-line interface, where they can input numbers and select operations. It includes robust error handling to manage invalid inputs, such as division by zero and unsupported operations, providing users with informative feedback. This project serves as an excellent example of applying OOP concepts in Java, making it an ideal tool for both learning and everyday calculations.

2. Objectives

- Implement a calculator application using Java.
- Apply OOP concepts such as encapsulation, inheritance, polymorphism, and abstraction.
- Ensure code modularity and ease of maintenance.
- Handle exceptions gracefully and provide user-friendly error messages.

3. System Design

3.1. Overview

The calculator application is structured around an abstract ***`Operation`*** class, from which concrete operation classes inherit. The ***`Calculator`*** class manages the execution of these operations, while ***`CalculatorMain`*** serves as the entry point for user interaction.

3.2. Class Diagram

Below is a textual representation of the class hierarchy:

- **Operation** (Abstract Class)
  - Fields:
    - ***`double number1`***
    - ***`double number2`***
  - Methods:
    - ***`abstract double calculate()`***
- **Addition** (Extends Operation)
  - Methods:
    - ***`double calculate()`***
- **Subtraction** (Extends Operation)
  - Methods:
    - ***`double calculate()`***
- **Multiplication** (Extends Operation)
  - Methods:
    - ***`double calculate()`***
- **Division** (Extends Operation)
  - Methods:
    - ***`double calculate()`***
- **Calculator**

        - Methods:
            - *`double executeOperation(String operation, double num1, double num2)`*
- **CalculatorMain**
    - Methods:
        - *`public static void main(String[] args)`*

4. Implementation

4.1. Operation Class

The ***`Operation`*** class is an abstract class that defines the blueprint for arithmetic operations. It encapsulates the common properties and methods shared among all operations.

```java
package calculator;

public abstract class Operation {
    protected double number1;
    protected double number2;

    public Operation(double number1, double number2) {
        this.number1 = number1;
        this.number2 = number2;
    }

    public abstract double calculate();
}
```

4.2. Addition Class

The ***`Addition`*** class extends the ***`Operation`*** class and implements the addition logic.

```
package calculator;

public class Addition extends Operation {

    public Addition(double number1, double number2) {
        super(number1, number2);
    }

    @Override
    public double calculate() {
        return number1 + number2;
    }
}
```

4.3. Subtraction Class

The ***`Subtraction`*** class extends the ***`Operation`*** class and implements the subtraction logic.

```
package calculator;

public class Subtraction extends Operation {

    public Subtraction(double number1, double number2) {
        super(number1, number2);
    }

    @Override
    public double calculate() {
        return number1 - number2;
    }
}
```

4.4. Multiplication Class

The ***Multiplication*** class extends the ***Operation*** class and implements the multiplication logic.

```
package calculator;

public class Multiplication extends Operation {

    public Multiplication(double number1, double number2) {
        super(number1, number2);
    }

    @Override
    public double calculate() {
        return number1 * number2;
    }
}
```

4.5. Division Class

The ***Division*** class extends the ***Operation*** class and implements the division logic, including handling division by zero.

```
package calculator;

public class Division extends Operation {

    public Division(double number1, double number2) {
        super(number1, number2);
    }

    @Override
    public double calculate() {
        if (number2 == 0) {
            throw new ArithmeticException("Division by zero is not
allowed.");
        }
        return number1 / number2;
    }
}
```

4.6. Calculator Class

The ***Calculator*** class selects and executes the appropriate operation based on user input.

```java
package calculator;

public class Calculator {

    public double executeOperation(String operation, double num1, double 
num2) {
        Operation op;

        switch (operation.toLowerCase()) {
            case "add":
                op = new Addition(num1, num2);
                break;
            case "subtract":
                op = new Subtraction(num1, num2);
                break;
            case "multiply":
                op = new Multiplication(num1, num2);
                break;
            case "divide":
                op = new Division(num1, num2);
                break;
            default:
                throw new UnsupportedOperationException("Invalid 
operation.");
        }

        return op.calculate();
    }
}
```

4.7. CalculatorMain Class

The ***`CalculatorMain`*** class handles user interaction and serves as the application's entry point.

```
package calculator;

import java.util.Scanner;

public class CalculatorMain {

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

        System.out.println("Welcome to the Calculator");
        System.out.print("Enter first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter second number: ");
        double num2 = scanner.nextDouble();

        System.out.print("Enter operation (add, subtract, multiply, divide):
");
        String operation = scanner.next();

        try {
            double result = calculator.executeOperation(operation, num1,
num2);
            System.out.println("Result: " + result);
        } catch (ArithmeticException | UnsupportedOperationException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

5. Object-Oriented Concepts Applied

5.1. Encapsulation

Encapsulation is achieved by bundling the data (***`number1`***, ***`number2`***) and methods (***`calculate()`***) within classes. The ***`Operation`*** class and its subclasses encapsulate the data and behavior specific to each operation.

5.2. Inheritance

Inheritance is used to create a hierarchy of classes. The concrete operation classes (***`Addition`***, ***`Subtraction`***, ***`Multiplication`***, ***`Division`***) inherit from the abstract ***`Operation`*** class, reusing common code and allowing for polymorphic behavior.

5.3. Polymorphism

Polymorphism allows the ***`Calculator`*** class to interact with different types of operations through a common interface. By treating all operations as instances of the ***`Operation`*** class, the ***`Calculator`*** can execute any operation without knowing its specific type.

5.4. Abstraction

Abstraction is implemented through the ***`Operation`*** abstract class, which provides a template for concrete operation classes. This hides the implementation details from the ***`Calculator`*** class and promotes a clear separation of concerns.

6. Testing and Results

The application was rigorously tested with various inputs to ensure accuracy and robustness.

Test Case 1: Addition of Two Positive Numbers

- **Input:**
    - Number1: `5`
    - Number2: `3`
    - Operation: `add`
- **Expected Output:** `8`
- **Actual Output:** `8`
- **Result:** Pass

Test Case 2: Subtraction Resulting in Negative Number

- **Input:**
    - Number1: `2`
    - Number2: `5`
    - Operation: `subtract`
- **Expected Output:** `-3`
- **Actual Output:** `-3`
- **Result:** Pass

Test Case 3: Multiplication with Zero

- **Input:**
    - Number1: `0`
    - Number2: `10`
    - Operation: `multiply`
- **Expected Output:** `0`

- **Actual Output:** ***`0`***
- **Result:** Pass

Test Case 4: Division by Zero

- **Input:**
  - Number1: ***`10`***
  - Number2: ***`0`***
  - Operation: ***`divide`***
- **Expected Output:** Error message "Division by zero is not allowed."
- **Actual Output:** Error message "Division by zero is not allowed."
- **Result:** Pass

Test Case 5: Invalid Operation Input

- **Input:**
  - Number1: ***`4`***
  - Number2: ***`2`***
  - Operation: ***`modulo`***
- **Expected Output:** Error message "Invalid operation."
- **Actual Output:** Error message "Invalid operation."
- **Result:** Pass

7. Conclusion

The Calculator project successfully demonstrates the application of key Object-Oriented Programming concepts in Java. The use of abstraction, encapsulation, inheritance, and polymorphism results in a modular and extensible codebase. Exception handling ensures that the application can gracefully handle invalid inputs and operations, providing a robust user experience.

8. References

- Oracle Java Documentation: https://docs.oracle.com/javase/8/docs/
- **Effective Java**, Joshua Bloch, Addison-Wesley Professional