1. Title Page:

Rajesh Panta
920636337
csc 413-01, summer
https://github.com/csc413-SFSU-SU2024/calculator-rajeshpanta.git

2. Introductions:

Project Overview:

The goal of this project is to develop a Calculator application designed to evaluate mathematical expressions using a graphical user interface (GUI). The purpose of this project is to demonstrate the application of object-oriented programming principles in creating an interactive tool that simplifies mathematical calculations for users.

Technical Overview

The project involves an object-oriented design for creating a mathematical expression evaluator and an associated Calculator GUI. The architecture comprises several key components:

i. Operand Class:

a. Purpose: Represents integer values within mathematical expressions.
b. Implementation: Includes a constructor for initialization and a method to retrieve the operand's value, ensuring encapsulation via private access modifiers.

ii. Operator Abstraction and Subclasses:

a. Abstraction: Defined as an abstract class or interface with a method execute (Operand op1, Operand op2) to perform operations using two operands.
b. Subclasses: Include specific operators like AddOperator, SubtractOperator, MultiplyOperator, DivideOperator, and PowerOperator, each implementing the execute method to perform their respective operations.

iii. Evaluator Class:

a. Functionality: Evaluates infix mathematical expressions using two stacks; one for operands and one for operators. Implements the evaluateExpression(String expression) method to parse and evaluate expressions.
b. Algorithm: Utilizes a stack-based approach for parsing and processing tokens from the expression, managing operator precedence, and handling parentheses to ensure correct evaluation order.

iv. Calculator GUI (EvaluatorUI.java):

a. Integration: Uses the Evaluator class to process input from the GUI, which includes buttons for digits and operators, and a display area for the result.
b. User Interaction: Allows users to input expressions directly through the GUI, which are then evaluated on-the-fly, displaying results immediately.

Summary of Work Completed

Class Implementation:

a. Operand Class: Successfully implemented to handle the storage and retrieval of integer values within expressions.
b. Operator Classes: Developed all required operator subclasses, each tailored to handle specific arithmetic operations, ensuring correct execution of expressions involving multiple types of operations.

Evaluator Class Completion:

- Completed the implementation of the Evaluator class, effectively parsing and evaluating complex mathematical expressions using a robust algorithm based on operator precedence and stack-based processing.

GUI Development:

- Designed and implemented the Calculator GUI using Java Swing, integrating the Evaluator class to provide real-time expression evaluation. The GUI layout was carefully crafted to provide a user-friendly interface with clear demarcation of operations and display.

Testing:

a. Conducted comprehensive unit testing for each component (Operand, Operator, Evaluator) to ensure reliability and correctness. The tests covered a wide range of scenarios, including simple operations, complex expressions, and error handling.
b. Ensured all components adhered to specified design restrictions, particularly with regard Integration and Final Testing:
c. Successfully integrated the Evaluator with the Calculator GUI, conducting final end-to-end testing to ensure the entire application functioned as expected, providing accurate results and responsive user interactions.
d. For the test to pass successfully added a import calculator.operators.operator_name to call and successfully pass the test:

3. Development Environment:
   a. OpenJDK version "21.0.2"
   b. The version of IntelliJ IDEA I am using is 2023.3.4 Ultimate Edition.

4. How to Import Your Project:

To import the Calculator project in IntelliJ IDEA on MacOs:
i. javac -d target ./calculator/evaluator/Operand.java

ii. javac -d target ./calculator/operators/*.java

iii. javac -d target ./calculator/evaluator/*.java

5. How to Run Your Project:

MacOS:

javac -d target --class-path "./:./lib/junit-platform-console-standalone-1.9.3.jar" ./tests/operator/*.java

6. Assumptions Made:

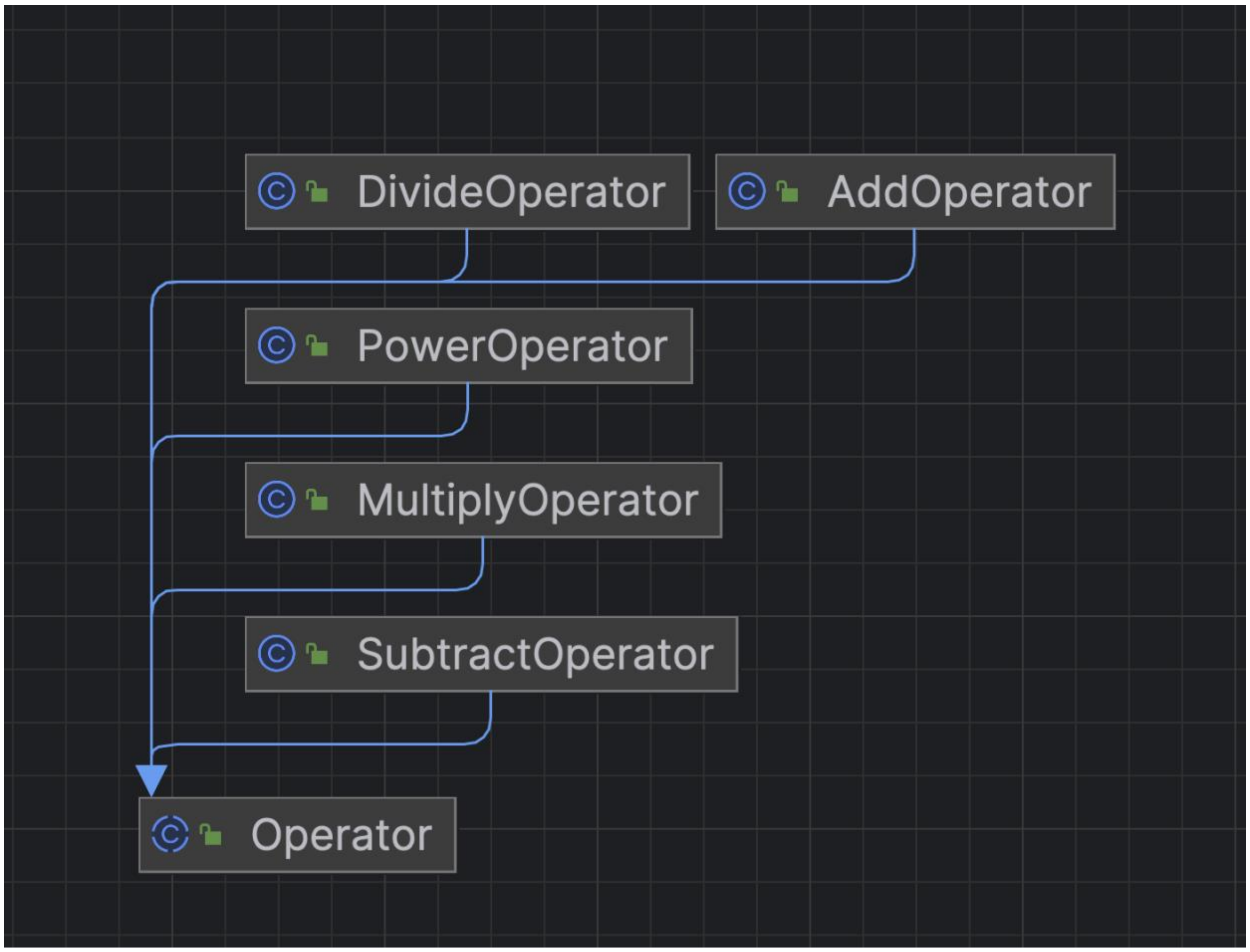
The implementation assumes that all user inputs are valid numeric values or operators. It also assumes that users understand basic arithmetic operations and the use of parentheses to specify order of operations.

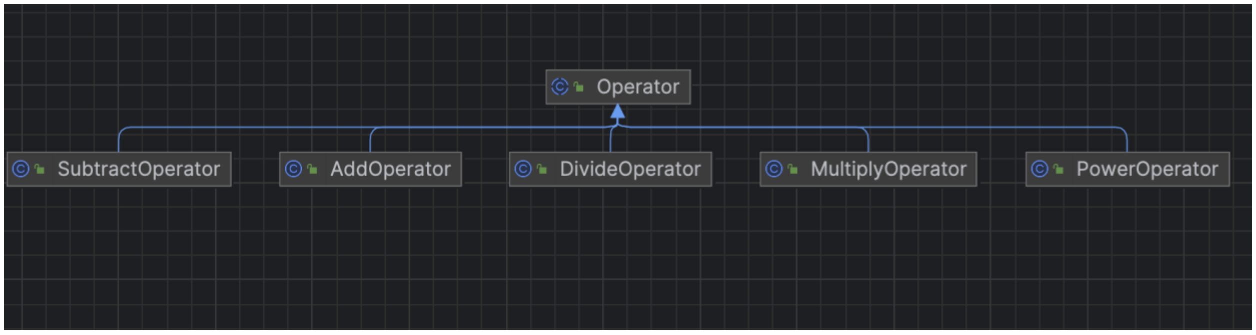
7. Implementation Discussion:
    a. Design choices made duing the implementing process:

The project consists of four main components: the Operand class, the Operator abstraction and its subclasses, the Evaluator class, and the Calculator GUI implemented in EvaluatorUI.java. The design utilizes object-oriented principles such as encapsulation, inheritance, and polymorphism to build a modular and scalable application. The Evaluator class acts as the core, utilizing other components to parse and evaluate mathematical expressions provided through the GUI.
The Operand class is designed to encapsulate an integer value, representing the operands used in mathematical expressions. It provides a constructor to initialize the operand's value and a method getValue() to retrieve this value, ensuring the internal state is protected and accessible only through this method. This approach adheres to the principle of encapsulation, using the private access modifier for the value field to restrict direct access from outside the class.
The Operator abstraction is implemented as an abstract class that defines a method execute(Operand op1, Operand op2) to be overridden by its subclasses like AddOperator, SubtractOperator, etc. Each subclass implements the execute method to perform the specific operation corresponding to the operator. This design leverages polymorphism, allowing the use of a single interface to interact with various operator types, and simplifies the management of different operations in the Evaluator class.
The Evaluator class is completed to parse and evaluate mathematical expressions using two stacks: one for Operand instances and one for Operator subclasses. The method evaluateExpression(String expression) implements the algorithm where tokens are processed sequentially: Numeric tokens are converted into Operand instances and pushed onto the operand stack. Operator tokens are compared against the top of the operator stack for precedence, and either pushed onto the stack or processed immediately depending on their priority. Parentheses

are handled by pushing them to the operator stack until a closing parenthesis is encountered, at which point operators are processed until the matching opening parenthesis is found. Challenges like managing operator precedence and handling nested expressions were addressed by carefully designing the loop that processes tokens and ensures correct execution order.
The Evaluator class is integrated into the Calculator GUI through EvaluatorUI.java, which provides a graphical interface for user input. Buttons for numbers and operators generate corresponding tokens, which are then passed as a string to Evaluator.evaluateExpression() when the equal button is clicked. The result is then displayed on the GUI. This setup demonstrates the practical application of the evaluator in a user-friendly interface.
Tests were implemented to verify each component individually and in combination. Tests for the Operand and Operator classes checked the integrity of operations and state management, while tests for the Evaluator class focused on verifying correct expression outcomes for a variety of scenarios, including complex nested expressions and different operator precedences. This project was a valuable exercise in applying object-oriented design principles to a practical problem.

b.





8. Project Reflection:

This project presented challenges in managing operator precedence and implementing the GUI interaction seamlessly. I learned a great deal about the practical application of object-oriented design patterns, especially in how they can facilitate the management of complex data interactions and improve code modularity and readability.

9. Project Conclusion and Results:

The application accurately evaluates a wide range of mathematical expressions, demonstrating the effectiveness of the stack-based parsing and evaluation algorithms. The Calculator GUI provides a user-friendly and responsive interface for real-time expression evaluation, designed to be intuitive and easy to use. Comprehensive testing ensured that all components function correctly and efficiently, with the application handling both normal and edge cases effectively. As a result, with all the combination of files the program runs successfully with passing all the test file.