Assignment 01 - Expression Evaluator

Dylan Burns

922717234

CSC 413 - Fall 2022

Repository Link: https://github.com/csc413-SFSU-Souza/csc413-p1-Dylan-Burns

Introduction

a. Project Overview

The project is to design an application that computes equations provided by the user through a graphical user interface. The user inputs an equation and the application splits up the input into operators and operands. Operators are processed according to their precedence within the equation. For instance, Addition (+) and Subtraction(-) have a lower priority than Multiplication(*) and Division(/). The application must compute operations in order of precedence to be accurate.

b. Technical Overview

The calculator application is composed of various classes that split the application into well-defined operations. First, we have the Evaluator class which includes a single method to evaluate the expression the user inputs. This method verifies the input is valid, splits and stores operators and operands into two different data structures (Stacks) and then retrieves the necessary components as needed to compute the sum of the expression. If the expression is valid, the method parses the string and pushes operators and operands into their stack. The program then computes the expression according to the operator priority which is defined within each Operator subclass. The *evaluateExpression* method in *Evaluator.java* begins to separate the expression into separate components, computes the sum of each operations according to its priority, and then returns the value back to the original expression to continue evaluating the expression until each operation has been computed, thus resulting in a total sum of the expression the user input. The *Operator* class contains 7 subclasses (*AdditionOperator, SubtractionOperator, MultiplicationOperator, DivisionOperator, OpenParenthesisOperator, PowerOperator,* and *StartExpressionOperator*). Each plays its own role within the *Evaluator* class by computing the expression passed to it and returning the value back to the *evaluateExpression* method. We have created separate classes for each type of operation to achieve a cohesive design and ensure our code is robust and maintainable. If one of these operations fail, we know which portion of the expression caused the error because the compiler outputs the class that caused the program to fail. Our *Operator* class contains a hashmap of the various operators to look for while evaluating the expression. By determining the priority of each operator within the expression we are then able to separately compute each sum to reach the total sum of the expression.

c. **Summary of work completed**

My contribution to this assignment includes completing the required methods and classes as instructed in the Assignment outline. Aside from the programming portion I have also completed the documentation portion of the assignment which outlines the design and implementation of the application.

Development Environment

a. **Java Version :** openjdk 17.0.1 2021-10-19

b. **IDE :** IntelliJ IDEA

Build Instructions

You can download the zip file from my github repository. Go to intellij → File → Open → Downloads → csc413-p1-Dylan-Burns-main. Once you have opened the project in intelliJ you will need to add a JDK to the project structure. I chose to use *openjdk-17 version 17.0.1.* Once you have added the JDK you can run the *EvaluatorUI.java*. This will execute the main method and open a graphical user interface where you can test the application.

Run Instructions

Run *EvaluatorUI.java* and input the test cases provided by the *EvaluatorDriver.java* class.

Assumptions

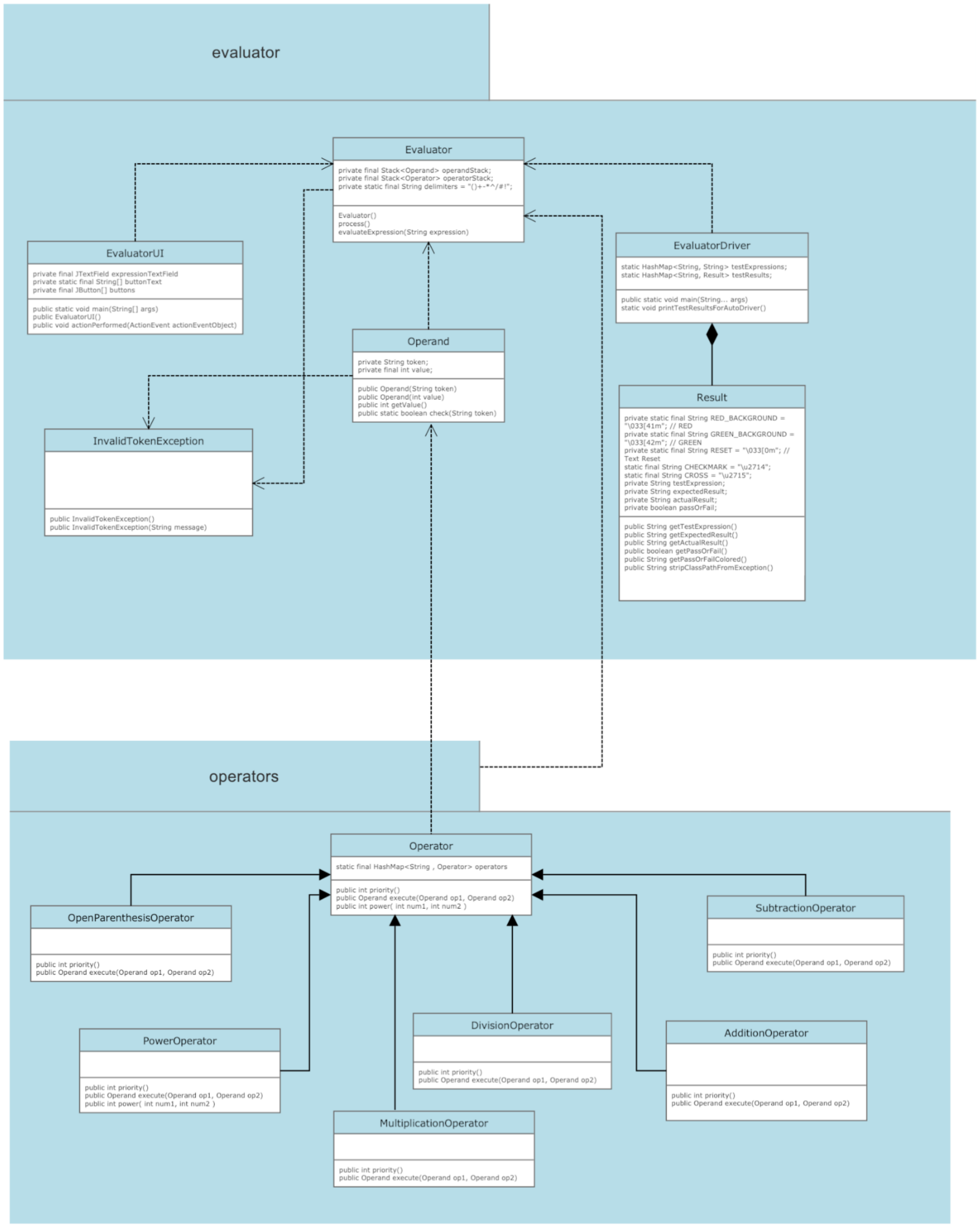
The application assumes you use the correct syntax, expect the output rounded to the closest whole number, and follow the build and run instructions provided above.

Implementation

Before writing any code I reviewed the classes to understand their responsibilities within the application. Understanding what behaviors and attributes the class objects have is important so we don't write duplicate or error prone code. The calculator app needs to accept user input, parse the expression by separating the operands and operators, and evaluate individual portions of the entire expression according to the operator precedence.

The *Evaluator* class is responsible for accepting the expression input by the user, validating the syntax and operators, separating the expression for individual computation according to precedence by pushing the operations onto two separate stacks (one for operators and one for operands), and returning the value back to the expression to compute the remaining operations. The *Operator* class is responsible for determining if the expression contains valid operators and for each of its subclasses which represent the valid operators. Each subclass contains two methods (with the exception of the *PowerOperator* class). The first method *priority()* is responsible for returning the priority value of that operator. The second method *execute()* is responsible for computing the value of that operation with the two parameters passed to it (i.e. operand1 and operand2). The *Operand* class is responsible for validating that the user input contains valid operands, and parsing the string expression into integer values to be computed with their corresponding operator.

UML Diagram


evaluator
Evaluator
private final Stack<Operand> operandStack;
private final Stack<Operator> operatorStack;
private static final String delimiters = "()+-*^/#!";
Evaluator()
process()
evaluateExpression(String expression)
EvaluatorUI
private final JTextField expressionTextField
private static final String[] buttonText
private final JButton[] buttons
public static void main(String[] args)
public EvaluatorUI()
public void actionPerformed(ActionEvent actionEventObject)
EvaluatorDriver
static HashMap<String, String> testExpressions;
static HashMap<String, Result> testResults;
public static void main(String... args)
static void printTestResultsForAutoDriver()
Operand
private String token;
private final int value;
public Operand(String token)
public Operand(int value)
public int getValue()
public static boolean check(String token)
InvalidTokenException
public InvalidTokenException()
public InvalidTokenException(String message)
Result
private static final String RED_BACKGROUND = "\033[41m"; // RED
private static final String GREEN_BACKGROUND = "\033[42m"; // GREEN
private static final String RESET = "\033[0m"; // Text Reset
static final String CHECKMARK = "\u2714";
static final String CROSS = "\u2715";
private String testExpression;
private String expectedResult;
private String actualResult;
private boolean passOrFail;
public String getTestExpression()
public String getExpectedResult()
public String getActualResult()
public boolean getPassOrFail()
public String getPassOrFailColored()
public String stripClassPathFromException()
operators
Operator
static final HashMap<String , Operator> operators
public int priority()
public Operand execute(Operand op1, Operand op2)
public int power(int num1, int num2)
OpenParenthesisOperator
public int priority()
public Operand execute(Operand op1, Operand op2)
SubtractionOperator
public int priority()
public Operand execute(Operand op1, Operand op2)
PowerOperator
public int priority()
public Operand execute(Operand op1, Operand op2)
public int power(int num1, int num2)
DivisionOperator
public int priority()
public Operand execute(Operand op1, Operand op2)
AdditionOperator
public int priority()
public Operand execute(Operand op1, Operand op2)
MultiplicationOperator
public int priority()
public Operand execute(Operand op1, Operand op2)

Reflection

After completing the coding portion of the assignment I spent time creating the UML diagram and fixing any bugs in the application. One issue that I found when debugging was the *C* button which is responsible for removing single characters rather than deleting the entire entry. After about five minutes I realized I was overthinking and could simply set the text field to the same as before but with a substring from index *0* to *expressionTextfield.length - 1*. Another bug that caused issues within the application *CE* button. Initially I thought setting the text field to a single space (which I mistakenly thought was an empty string) would suffice. I realized that setting the text field to a single space would cause the future expression syntax to include that space, thus causing the program to parse the string incorrectly. If I had more time to work on the Assignment I would add additional user support to provide a "man" page that detailed the application assumptions. I would also include enhanced error handling for the application to avoid unwanted termination due to user error. If an input was invalid I would output error messages detailing what portions of the string caused the program to fail.

Conclusion and Results

To conclude, the calculator application runs as expected and I was able to navigate the various classes and understand the role they each played. Creating the UML diagram was a bit unusual as I have never done one before so there may be some mistakes, but I will continue to practice so I am prepared for future assignments. This assignment has helped me to get back into the groove of programming and using data structures. I took a break over the last few weeks of summer from my side projects and have noticed some areas that need refreshing. This was a fun application to work on and I look forward to the future programming assignments.