CSC 413 Project Documentation

Summer 2021

Yasin Hagos

918738176

01

[***https://github.com/csc413-su21/csc413-p1-yasinh23.git***](https://github.com/csc413-su21/csc413-p1-yasinh23.git)

Table of Contents

1 .Introduction

This project required us to set up a calculator that would solve infix expressions such as 1+2+3 or (2+4)*3. We had to determine a way to calculate these expressions by separating the operands(1,2,3,4...etc) from the operators(+-*^/..etc) and then bringing them back together following conventional precedence math rules such as PEMDAS to actually solve these expressions. Then we were tasked with providing a user with a visual calculator that looks like the ones computers have pre installed. The calculator's functionality was made possible by attaching the corresponding values to the buttons so when the user presses a button the right evaluations take place in the back end of the calculator.

1.2 Technical Overview

In this project I implemented an operand class that takes values as strings and integers and implements methods such as getvalue() and check() which get the value and check if what was provided is an operand. I then implemented an abstract operator class that initializes a hashmap that contains all the different types of operators. This abstract class also contains an abstract priority and execute function that keeps track of operator precedence priority and method execution for each operator. Similar to the operand class the operator class also contains a getOperator and check() method. Then I created the child classes of Operator such as addOperator,multiplyOperator…etc that hold implement the abstract methods contained in the parent class Operator. The evaluator class contains 2 stacks one for operands and the other for operators. The evaluator breaks down given expressions into tokens of operands and operators and places them in corresponding stacks. The algorithm to process an expression is (i) pop operand stack once (value1) (ii) pop operator stack once (operator) (iii) pop operand stack again (value2) (iv) compute value1 operator value2 (v) push the value obtained in operand stack. If the character is an operator and the operator stack is not empty, and the character's precedence is greater than the precedence of the stack.peek of operator stack, then push the character onto the operator stack.If the character is ")", then we do algorithm process (i)-(v) as explained before until the corresponding "(" is encountered in operator stack. At this stage POP the operator stack and ignore "(." If these cases don't apply then you would just perform the algorithm to process an expression (i)-(v).

1.2 Summary of work completed

In this project I then completed the remainder of the operator and operand class. I then created 6 subclasses for all the operators. I implemented the methods that in each operator subclass that determine how they behave. I applied the algorithm to the best of my ability in the Evaluator class and was able to get some of the tests to pass but not all. The remainder of the EvaluateUI class was also implemented.

2 Development Environment

I developed this program in Java using Intellij idea IDE

3 How to Build/Import your Project

1. Make sure Git is installed by entering “Git” in terminal. If there is no error, Git is installed.
2. Make sure Java is installed by entering command “Java --version” in terminal. The program has been tested for Java 12.
3. enter command: git clone “ https://github.com/csc413-su21/csc413-p1-yasinh23.git”

4 How to Run your Project

1. Open cloned folder in a Java IDE

2. Navigate to calculator/src/main/java/edu.csc413.evaluator/EvaluatorUI

3. Run EvaluatorUI file

4. Enter math problem

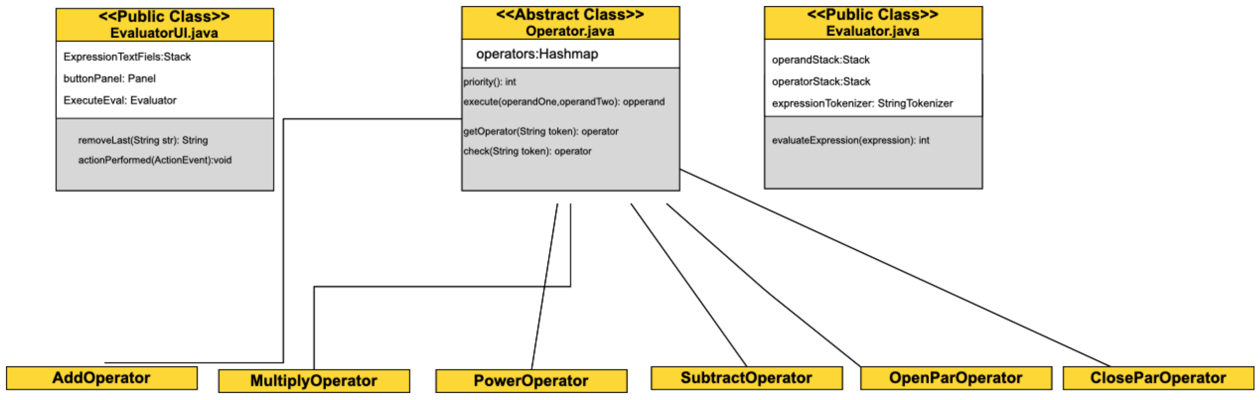
5 Assumption Made

I assumed coming into this project that the algorithm was going to be the hardest part which it was. The way the algorithm handled priorities especially with the parentheses was something I had to think about alot

6 Implementation Discussion

The implementation of classes was included creating new objects and the use of abstraction particularly in the case of the operator class. The abstract class provided its subclasses of operators to decide for themselves how to execute giving methods

6.1 Class Diagram



7 Project Reflection

I found this project to be pretty difficult for me until I really understood what was going on. The advice from the videos on what the best approach was to starting really helped out. I worked on the Operand class then Operator class before starting the Evaluator class which was extremely difficult for me. The way the algorithm handles the parenthesis was something that put me in a place of frustration. I ran into a lot of hiccups throughout the entirety of the project but just by searching through discord I could see that people ran into the same obstacles and someone in the class was able to help them out. I feel as if this project showed me that I need to get to work and sharpen my java skills in order to be successful in the remainder of this class.

8 Project Conclusion/Results

I was able to get most of the tests in this project to work but there is something still up with the way my algorithm is processing parenthesis. I'm not sure if the priority schema I'm using is the issue or if it has to do with the actual implementation of the algorithm. If I had more time I would probably go back and finish the project so that everything works properly.