CSC 413 Project Documentation

Spring 2022

Abbas Mahdavi

918345420

CSC 413-02

https://github.com/csc413-SFSU-Souza/csc413-p1-AbbasMahdavi021

Table of Contents

1 Introduction

1.1 Project Overview

The purpose of this project is to create a calculator, which operates simple mathematical expressions, such as addition, subtraction, multiplication, etc., as well as dealing with expressions that are contained in parentheses. There are two ways where a user can use our calculator. One where the would be asked for expressions, in a command prompt. More interestingly, we have also implemented a simple calculator interface, which would work like any other calculator, with numbers 1 to 9, and expressions, such as *, /, +, () and -, to execute any operation in that range. An example would be: (5+3)/2, which after pressing the equals button, would show us the number 4.

1.2 Technical Overview

The calculator's evaluator class takes an infix expression as a String of tokens. Which is then parsed. The implemented GUI interface is also available to the same task. The expressions/tokens in this calculators are separated in two groups, the Operands, the strings of numbers that are turned into integers for evaluation, and well as operators: (**() +/*-^**), which are used to evaluate expressions. This is done in the Evaluator class, which contains two Stacks one for Operands and one for Operators, which then processes these tokens and returns a result, of the mathematical expression that was scanned. The scanned operands are pushed into operand stack, and the evaluator tokens are pushed into the evaluator stack when scanned. All of these tokens have set priority values that come into play here. Operators are pushed in such manner that when a closing parenthesis is scanned, and the operator stack is empty, it will not execute until a open parenthesis is scanned. We also check to see if operator stack is empty or there exists a open parenthesis, after which we can process a new operator. Similar if and else if statements are set in place, to basically ensure that the evaluators are pushed and popped in and order that makes sense. Then they are processed in the Evaluator class along with OperandOne and OperandTwo, which are 2 integers, that complete that certain mathematical expression, returning an integer as a solution to the math expression. Which is then shown to the user.

1.3 Summary of Work Completed

My contribution to the project was to the following classes: Evaluator, EvaluatorUI, and Operand classes in the evaluator folder. As well as the Operator class, and creating 7 new classes one for each evaluator/expression, in the operators folder.

The operators each are describe what the expression do, and what their priorities are. And they are brough together, in the Operator class.

I worked on the evaluator class the most, as was explained in the Technical Overview. The evaluatorUI which deals with the GUI, was also partly implemented by me, especially what the buttons on the GUI do and what they display. Lastly, the Operand class also needed work since in involved turning operands(numbers) from strings to integers and setting up a way where their values can be reached by the other classes in the program.

After all of which, the program is working correctly, tests passed, and so it is complete.

2 Development Environment

The version of java/SDK that was used to create this program is version 16.0

The IDE this program was made on was IntelliJ UE.

3 How to Build/Import your Project

To build this program, assuming we have correct clone the project from GitHub to our computer, we must do the following. We must make sure that the calculator folder is set as the root source of this project, when we open the project in IntelliJ. One way we can ensure this, in IntelliJ, clicking open, locating the assignment folder where we cloned it, going inside the assignment folder, and opening the calculator folder by clicking on it, and pressing open.

Next, we must ensure that the SDK is set. IntelliJ might prompt to "setup SDK" or download it if the program does not have it.

If all of this is done correctly, there should not be any red errors displayed (in the top-right corner of IntelliJ). There will be yellow warnings that are insignificant and can be ignored. None of the folder or classes on the left-hand panel should have any red underlines under them, and the classes should appear a blue color. If not, we have imported it wrong, and must open the folder again.

4 How to Run your Project

Next to see the project working, we can first try the Tests that are provided to us. Under src/Test folder, we can right click on the green java folder, and choose "Run 'All Tests'", which will run the tests. 60 tests will display as passed, one as failed which can be ignored.

Next, we can go to the evaluator folder, under, src/main/java/edu.cs.../evaluator, and right click on EvaluatorDriver.java, and run it. Which will open cdm, and asks for an input, of an expression. If a correct expression is entered, a correct result with be printed.

We can also run the evaluatorUI, in the same folder, which will bring up the calculator interface GUI, which can be used as any calculator, entering and expression and pressed the "=" button, returns the correct result, of given expression.

5 Assumption Made

One assumption is that the user enters valid expression, meaning no expression that would lead to ambiguity, or anything that involves decimals/ fractions. Since we are dealing with integer variables. So, a any fraction expression, like, 5/10, would return zero, since integers round down. 0.5 => 0. Also, we can assume that the program has a limit of how large of a number it can process. 11^11, would return a max result number of: 2147483647.

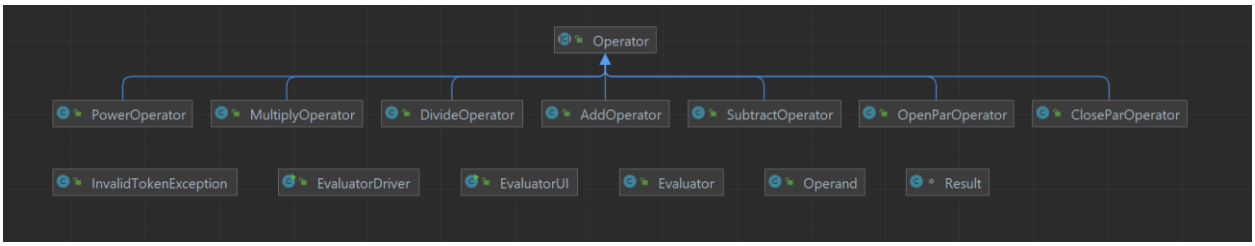
We must also assume that the entered inputs must always be positive. So, -6 + 2, would return null, since we don't have a way to differentiate between "-" the operator, vs "-" the sign.

6 Implementation Discussion

The main implantation worth mentioning is dealing with operator classes. I made a separate class for each of the operators, which includes the parentheses, and gave them their priority. I had to give a priority 0 to open par, and -1 to close par, as a way of setting them apart, as this was needed in the evalauotr.java class. This made easy to create a class that extends Operators, which fills out the abstract functions and adding a new entry to the hash map.

The classes in this project are visualized below, in a class diagram.

6.1 Class Diagram



7 Project Reflection

It is always a huge help to have an starter code. But it can sometimes be intimating when opening a file for the first time, and seeing lots of code written already, and having to go through them and trying how things are set up. So, I thought I would struggle with this assignment at first. But the code was cleared up the more I read and the more I coded myself. I loved working on the Operand class not because it was short, but because it felt like I finished a step and it gave me more confidence. I also really liked working on the evaluatorUI, and the GUI. It was genuinely fun to code the GUI and see it work the way I wanted. I was confused most about the Evaluaotr.java. And I could not get it working properly. I had to go back the change my operator classes that I had made for the close and open pars. But eventually they worked out fine.

The assignment is all around great, and knowing the purpose of it, being testing our knowledge and where we stand on java before heading to assignment 2, makes everything make sense. But if this was a stand-alone program but itself, it could have major improvements just a different evaluator class, that can, for example, deal with negative numbers, and more abstract expressions. But again, that's not the point of the project. So over all, I feel very happy completing this assignment on time, and seeing everything come together.

8 Project Conclusion/Results

This project is a complete infix expression evaluator with a GUI calculator interface. It can handle a range of (some what limited) mathematical expressions, with many combinations of the operators and parentheses. The evaluatorUI give correct results when proper expressions are entered. The GUI also works perfectly fine. All the tests also are passed, beside one that we can ignore.