CSC 413 Project Documentation

Spring 2020

Duy Nguyen

Student ID: 917446249

CSC 413.03

GitHub Repository Link

https://github.com/csc413-03-spring2020/csc413-p1-duynguyen2

Table of Contents

1 Introduction

1.1 Project Overview

This is a calculator program that works like a simple calculator would. It can add, subtract, multiply, etc. only whole numbers. It can also use parenthesis to specify certain operations to be done first. It does throw errors when there are too many of one operator in a row, missing parenthesis or not enough numbers. It takes the texts and scans through to get rid of unnecessary spaces and does each operation depending the PEMDAS order (Parenthesis, Exponents, etc.).

1.2 Technical Overview

This is an Expression Evaluator, which takes strings of mathematical expressions and computes them. It looks at each character of the expression as a token and converts them to either numbers or operators. It throws errors if there are unbalanced expressions (missing parenthesis, too many plus signs, etc). It will take only positive inputs, but not negative inputs due to possibly mistaking the negative as a minus sign. It has three main classes; the Operand, Operator and Evaluator classes. The Evaluator UI is to allow for an interface for the Evaluator to use and function like a desktop calculator. The Operand class will turn tokens into integer values as long as the token is an integer. The Operator will compute a mathematical operation with two operands corresponding to the operator token and return the resultant operand. The Evaluator class has methods that takes a string expression and solve it if it is a valid expression while ignoring spaces, otherwise it will throw an exception.

The Operator class is an abstract class that has a hash map of all the simple math operators and checks tokens to see if they are mathematical operators and execute them accordingly. The hash map is where all the operators get put into so the Evaluator class can check if it exists and is an actual operator. Using a hash map instead of a list is helpful, since order doesn't matter, and the operators act as keys with a value, so it is easier to grab the key needed rather than a bunch of if statements. The methods are getOperator, check, execute and priority. The check method takes a string argument returns a Boolean if the string has a key in the hash map and it is implied to use the check method before using the other ones. The getOperator takes a string argument and returns the operator corresponding to it. The priority method is an abstract method that returns an integer value that will be used to determine which operator should be done first; the higher the number, the more priority. The execute method is an abstract method that takes two Operand objects and computes the math corresponding to the operator (ex. The plus operator will add them and return the sum).

There are 7 sub-classes of it that are all simple mathematical operators, including parenthesis. Each of them has to have the priority method and the execute method from the Operator class. The priority order of the operators is as follows in the table below:

Operators	Priority
(	-1
)	0
+, -	1
*/	2
^	3

The parenthesis classes are the only ones where the execute methods return null since they should not do anything. They need to have a priority to check and find them in the expressions. Their priority values can be anything as long as they are not the same as the other operators and less than 1. Other than that, the execute methods do what you would expect them to for the other ones.

The Operand class is another abstract class. The word operand is to define the numbers that will be used in the expressions. It takes tokens and converts them to integer values as long as the token is an actual integer. The variables that the Operator sub-classes need to actually do operations are Operand variables. This class has two constructors, one takes an integer value and one takes a string value and parses it into a signed decimal integer. It has a getter method called getValue, which returns the integer. There is also a check method that will check a string token to see if it can be parsed, otherwise it throws an exception. The check is implied to be used before the Operator constructor with a string argument.

The Evaluator class utilizes all of these other classes. There are two stacks, operandStack and operatorStack, and delimiters to check for, as well as an expressionTokenizer which is a string tokenizer. It takes expressions as string tokens and converts each character token as either an operand or an operator variable. Then once they get found as one or the other, then they get pushed into the corresponding stack. It will push any new operand or operator into the stack if those stacks are empty. If the operator stack has at least one operator and the operand stack has at least two operands, then it will create an operator value assign it with the operator at the top of the operator stack and create two operand variables and pop two from the operand stack. The operand stack has the left most operand from the expression at the bottom, meaning the first operand is below the second one in the stack so, the second one needs to be popped and then the first one. The operator variable will use the execute method with the two operand variables as arguments and return the result and push it into the operand stack. The only operators that should and will never execute are the parenthesis. Afterwards, it checks if the token is the right parenthesis, if it is then it will go through the stacks and process the inside expression until it finds the corresponding left parenthesis. This keeps going until the operator stack is empty and the operand stack should have only one value and that is the result, then returns it.

The EvaluatorUI is the GUI that acts like a desktop calculator. It computes the expressions entered in through button inputs. It uses JFrame the ActionEvent class to create the GUI and be able to interact with it. If any button except the C, CE and equal sign buttons get pressed, it creates an expression and adds it to the text field. Once the equal signed is pressed, it will put the expression into a string and create an Evaluator variable and do the evaluation.

1.3 Summary of Work Completed

First, I looked through the given code to figure out what does what, the classes, functions, etc. that need to be added and what they should look like and leaving some notes to help sort it out. I created the proper bodies for the methods in the Operator and Operand classes. I created a class for each operator and gave them priorities starting from -1 to 3, and operators were assigned priorities given by the outline. I stuck to the naming convention suggested to give myself less work in changing the given code. I added a RightParenthesisOperator and LeftParenthesisOperator to give them priority (i.e. -1, 0), which are there so the Evaluator can recognize them as operators and the stack can just push and pop them since they shouldn't ever execute. I also rearranged the Evaluator class a little bit by creating a method with the lines of code that actually do the math and added the parenthesis to the delimiters and deleted a couple of comments to confuse myself less whilst coding. For the UI, I have to check what is

being entered, and if it's anything but the C or CE buttons, go ahead and display it, then if it's the equal sign, create an Evaluator variable and compute it, then display the result but if it's invalid then display a message. The CE button clears the expression until right before the last operator and the C button clears the entire text field.

2 Development Environment

The IDE used to program this project is IntelliJ using JDK 12.

3 How to Build/Import your Project

This project is made with IntelliJ, so it is recommended to import the project into the same IDE since I have not tested it with any other IDE. To import the project, click the green button that says "Clone or Download" on the GitHub repo's home page. Then take the HTTPS or SSH link, but if you do not really understand SSH then use the HTTPS link. Copy the link and open your OS' terminal or Git Bash. Then in the terminal type "git clone [url copied]" and it will clone the repo. Alternatively, the github desktop app can be used to pull. Then open IntelliJ, or another IDE, and select Import Project and select the calculator folder as the source. Check the "Create project from existing sources" and go next. The default fields don't need to be touched so you can hit next. Make sure there are two items checked then hit next again. There should now be a list of JARs for the unit tests. Then there should be another 2 folders selected and then a JDK should be selected. If not, then install JDK 12 but if it still not there then hit the plus button to show it, otherwise continue. There shouldn't be any framework and the project should be imported properly. All the folders are there and can be selected. To build it, you can run them by right-clicking the files and clicking the Run option or selecting them and hitting the play button.

4 How to Run your Project

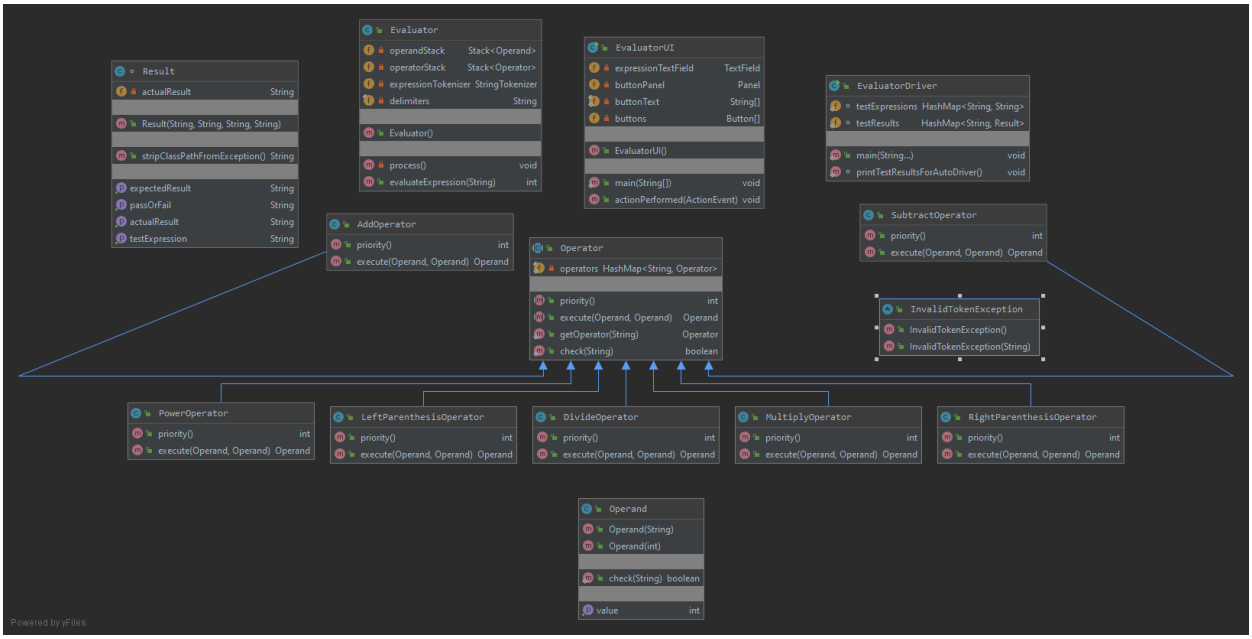
To run any of the code, you can click the arrows next to the folders until you see the C symbols. Right click any of the tests and hit run or select them and hit the play button. The only classes that can be run are the EvaluatorUI and the unit tests.

5 Assumption Made

- There are no negative inputs, only negative outputs
- Will not be tested in a way that tries to intentionally breaks the code
- Inputs and outputs are integers
- Every given expression is a string

6 Implementation Discussion

6.1 Class Diagram



7 Project Reflection

Starting this project took a while since I haven't coded at all in the past few months. The last time I did do code was in a different language, C++, so some of the general ideas were quick to grasp but getting back into the groove with Java instead took some time and made it a little tough. My knowledge of Java is mainly from CSC 210 and CSC 220, which only briefly covered things like hash maps, so I had to do some research on what they did, their methods and more. I found many helpful examples to help me get an idea for things like parsing and hash maps from geeksforgeeks.org, and though all of those parts were covered in lecture or already there, it helps me to know more so I can work around it better. I also had to remind myself of how stack worked and imagining each token should be stacking. One of the roadblocks was getting the Evaluator class to read the parenthesis tokens and then processing until it found the corresponding one. It would throw exceptions for any of the expressions with a parenthesis variable at first. I fixed it by utilizing priority values and making sure the parenthesis gets popped from the operator stack. This assignment was a bit difficult, especially because of the odd learning transition from CSC 210 to here. But all in all, it was not too bad; the coding portion of the assignment really took a few hours to finish and I spent time afterward messing around with different possible solutions and cutting down on the usage of resources. The UI is definitely the hardest to grasp since it isn't something that's really taught. I had never heard of ActionEvent, JFrame or TextField prior but after looking them up, it was not too hard to make up a solution.

8 Project Conclusion/Results

The Expression Evaluator works and properly can compute given expressions as long as they abide by the assumptions. If the expression is invalid, then it will throw exceptions. All the given unit tests should be passed if they are run.