Marc Castro
921720147
February 18, 2022

Github Repository

https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-1---calculator-mcastro16

Project Introduction and Overview

The evaluator project required me to implement an expression evaluator, basically a calculator. I was given a code skeleton and had to integrate that evaluator into a calculator user interface skeleton that I would complete.

Scope of Work

Task	Completed
Implement the eval method of the Evaluator class	X
Test the implementation with expressions that test all possible cases. The following expressions were used: ● 1+(1+2*3/4^5)+2 ● 1+2 ● 2*3	X X X X
Implement methods in the abstract Operator class ● boolean check(String token) ● abstract int priority() ● abstract Operand execute(Operand operandOne, Operand operandTwo) ● Lookup mechanism for operators to prevent instantiation of the same operator more than once	X X X X X
Implement Operator subclasses ● Properly organize subclasses (I used a package named operators) ● Implement subclasses for the required operands: multiplication, division, addition, subtraction, exponentiation, parentheses)	X X X

Implement Operand class ● Properly organize subclasses (I used a package named operators) ● Implement subclasses for the required operands: multiplication, division, addition, subtraction, exponentiation, parentheses)	X X X
Implement Operand class ● Constructors (String and int parameters) ● boolean check(String token) ● int getValue()	X X X X
Complete implementation of the Calculator UI in the EvaluatorUI class ● Use the previously implemented Evaluator ● Implement the actionPerformed method to handle button presses	X X X

Execution and Development Environment

I used the Atom IDE on my Windows Desktop to complete this assignment and tested it using the Windows Command Prompt via JDK compilation and running.

Compilation Result

Using the instructions provided in the assignment specification:
> javac Evaluator.java
> java Evaluator
> javac EvaluatorUI.java
> java EvaluatorUI

The compilation provided no warnings or error messages.

Assumptions

I assumed that all the expressions that were provided in the Evaluator's eval method would be correct. The invalid messages were also assumed to function properly. The provided Evaluator skeleton was also presumed to be correct and that I would only have to add code onto it and not alter any of the provided code.

Implementation

Evaluator

The provided Evaluator Class skeleton included an eval method that had to be implemented according to an algorithm that was talked about in class and linked from the specified file. As I worked with the code, I realized that there were cases where I could duplicate some code for specific functions. I first worked on trying to get the code to work for addition and subtraction since the ".put" functions were somewhat provided for those within the Operator class file, I just moved them into the Evaluator constructor so that it would apply to the entirety of the class.. I then implemented multiplication, division, and power. Once the code worked for those, I reworked the skeleton and changed a few of the while loops to account for reading parenthesis and running expressions within the parenthesis with priority over those that were outside of the parenthesis.

Operator and the Strategy Software Design Pattern

Implementation of the Operators required us to understand and implement the Strategy Pattern specified in class. The strategy allows us programmers to specify a family of algorithms as an abstract class and implement an algorithm that may be used in multiple scenarios during runtime. The abstract Operator class provided a base abstract method called execute, which would be called in our operator classes to provide outputs based on the operation it was specified to complete. The concrete classes that I created were the InitOperator, CloseExpressOperator, OpenParentOperator, CloseParentOperator, AdditionOperator, SubtractionOperator, MultiplicationOperator, PowerOperator, and DivisionOperator.

Operand

The Operand class was easy to implement for the most part. I did have to include an integer value and look up how to change a string to an integer for the two constructors that passed strings and integers. Other than that I implemented a try and catch method for the check method and just returned the value for the getValue method.

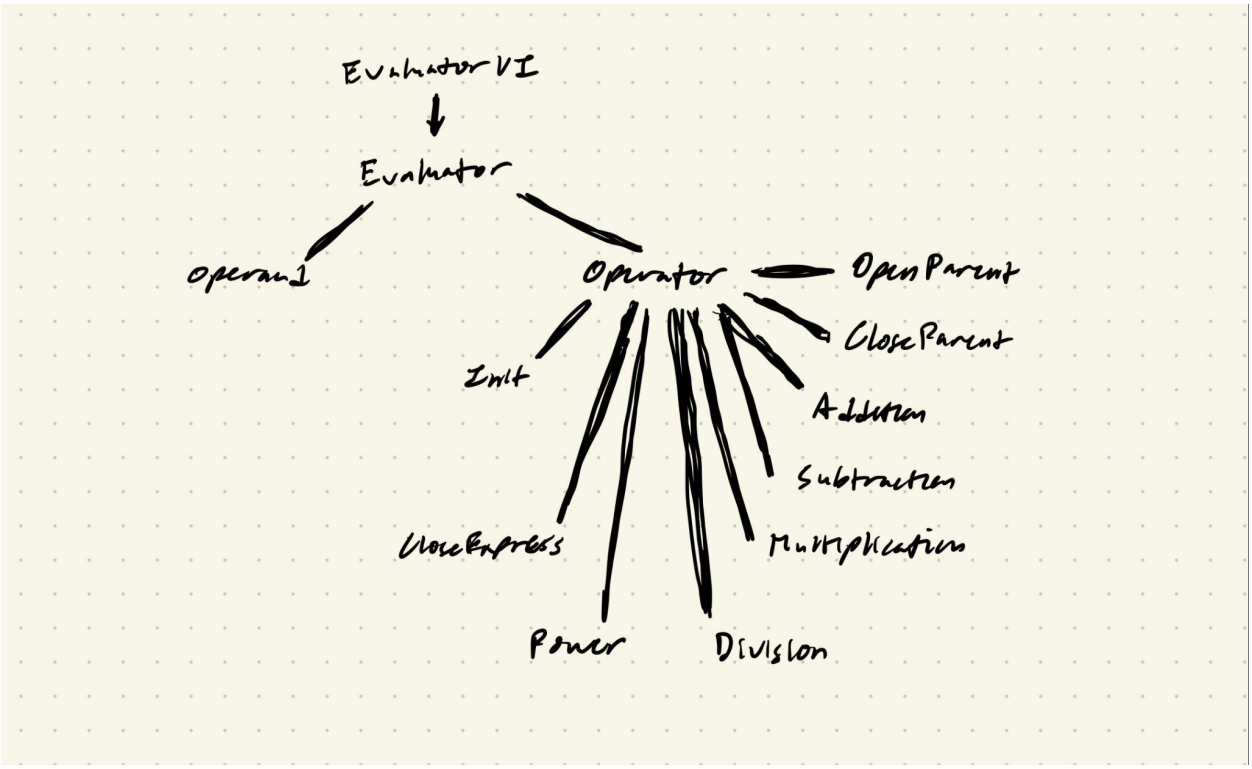
EvaluatorUI

The skeleton provided for the calculator required us to implement the actionPerformed method from the ActionListener interface. The professor went over the code for the framework and how to display all the buttons for the calculator. Within the actionPerformed method, I had it read first whether it was not "=", "CE", or "C". If it was not one of those three then the txField would be updated to include the specified button's text. If one of the three buttons were pressed, then the appropriate action would be performed. In the "=" button action, I took the text, passed it into the eval function found in Evaluator.java, converted that integer into a string, and passed it to the textField.

Code Organization

I moved all of the Operators and their derived classes into the same file.

Class Diagram



Results and Conclusion

This project was a great refresher to basic Java, as well as provided a challenge for me since I did transfer and was previously coding in C++ for projects like this. At my previous university, we were never really given a “Strategy Pattern”, so this was also another challenge that was somewhat hard to grasp but was good for me. Although late, I was able to implement all of the required features defined in the project specification and were able to fulfill all the requirements in the Scope of Work.

Challenges

The eval method was definitely the most challenging for me, I usually draw out how I want the code to run and what functionalities should run when necessary. However, this required drawing multiple interpretations of the code and diagnosing errors and bugs as they came up, which did help solve the assignment step by step. Implementing the parenthesis into equations and accounting for those was definitely difficult for me as well. I also had to include some extraneous variables in order to make the code get all of the tokens into the respective stacks because I would always run into errors with the stack not having enough operands to pop. Accounting for the notes provided by the professor and fixing those mistakes that the provided skeleton was making was something I had to wrap my head around. However, they did help with the logic as well as what I needed to do, it provided some sense of direction.

Future Work

Some future work could be to expand the functionality in EvaluatorUI and Evaluator to include some functions in calculus or trigonometry that are found in higher tier calculators. I also think that I could rework the code to be more efficient because although it is as efficient as it could be, I think the EvaluatorUI class code I created could be more optimized and the Operator class file could be broken up into different files for better organization.