CSC413 Project Documentation

Summer 2019

Steve Tu

918460002

CSC413.01

https://github.com/csc413-01-summer2019/csc413-p1-steve1316

Table of Contents

1 Introduction

1.1 Project Overview:
Assignment 01 covers how using the concept of encapsulation allows us to hide the mechanisms behind evaluating mathematical expressions with proper priority given to certain operators.

1.2 Technical Overview:
We abstract the Operator parent class in order to hide its child subclasses and their functionality of the various operators of mathematics. We utilize a hash map containing strings of the operators. Attached to them are the values containing the code necessary to create a new concrete operator object from its subclass. The evaluator will then use stacks for both operand and operator to parse out whether or not the token string is an operand or operator and put them into their respective stacks. If the token is an operator, it checks via the priority function in the operator subclasses to make sure that all the operators in the operator stack is placed correctly. After all operands and operators are parsed from the given expression, it will then go through while loops in order to execute via the appropriate operator subclass the 2 operands that are popped out along with its operator from the stack.

1.3 Summary of Work Completed:
I filled out the Operand class to have its own private integer to set and get from. The Operand constructor is also able to parse the integer value from the token string. For the Operator class, I added functionality for parentheses in the hash map and abstracted the execute function. The check function will go through a regex of operators to see if the given token string is an operator or not. The getOperator function simply goes through if statements inside a for loop to match the given string token to a specific operator in the hash map so that the function can return that operator back to the caller. I also created new files for all 6 operator subclasses in order to fill out their priority and execute functions that they inherited from the parent Operator class. In the Evaluator file, I added checks to see if stacks are empty so that I can break out of while loops in order to avoid stack empty exception errors. Then I copied the given while loop to pop 2 operands and its associated operator in order to execute them and pasted them in areas where they can handle parentheses and to handle the entire expression. I also adjusted the new while loops to make sure that the priorities required are met, especially for parentheses.

2 Development Environment:
I used Java SE Development Kit 12.0.1 and IntelliJ IDEA.

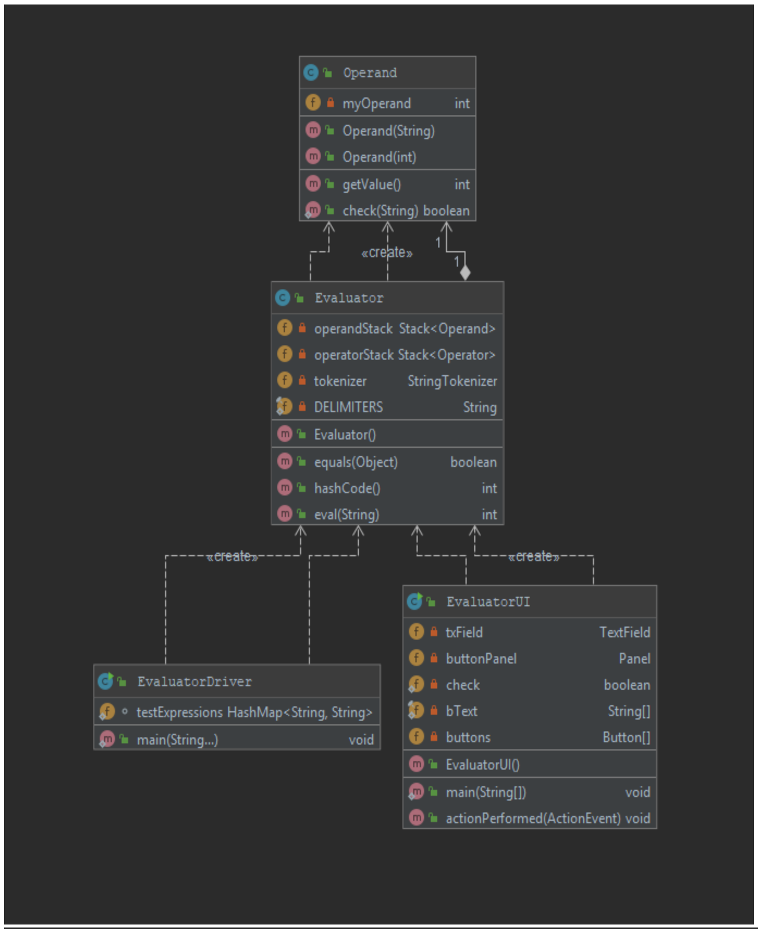
3 How to Build/Import your Project:
I imported to IntelliJ by fetching the repo via the Github Desktop program and then directing IntelliJ to the calculator folder. I made sure to have IntelliJ use JDK 12.0.1. After that, I can build my project by hovering over the Build menu and clicking on “Build Project”.

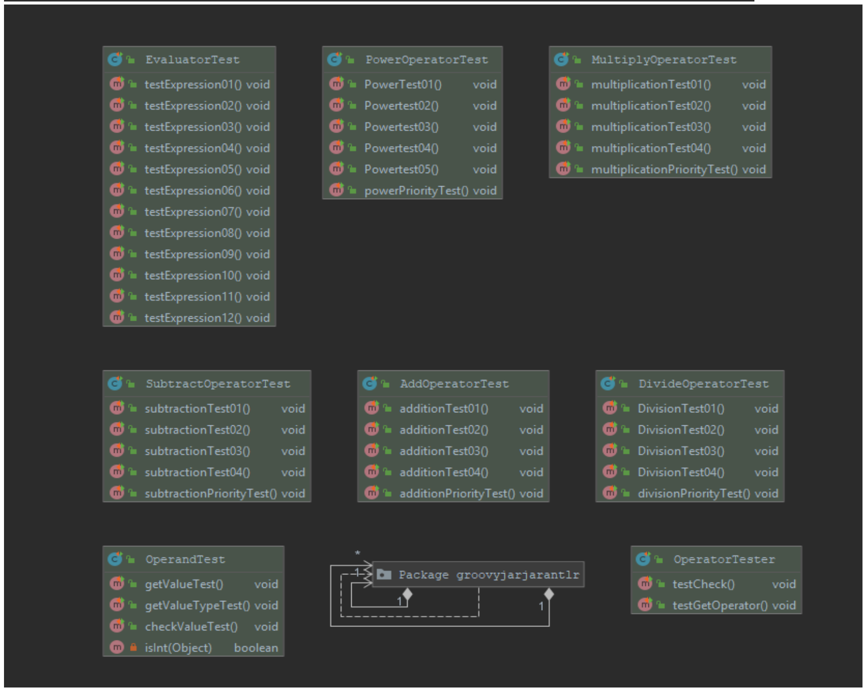
4 How to Run your Project:
I can run my project by hovering over the drop-down menu right next to the green Run button and selecting either the EvaluatorUI or the EvaluatorDriver. I use EvaluatorDriver with either no arguments, the “auto” argument to run the test expressions, and a user-defined expression. Or I can run EvaluatorUI to test the GUI and see if the expression is being sent properly to the Evaluator object and returns it back to the GUI correctly.

5 Assumption Made:
One major assumption that I made was that the given while loop that popped operands and operators in order to execute them was correct. If it was correct and working, I could save myself time by copying it to relevant areas in the code where I need to handle parentheses or where I need to handle the entire expression in the end. All that I needed to do was to properly check for priorities and empty stacks and I should be good to go for the Evaluator file. Another assumption that I made was that the “auto” test expressions were all valid and would cover any and all cases that the calculator could encounter as I did not check myself by giving it my own expressions.

6 Implementation Discussion

6.1 Class Diagram





7 Project Reflection:
This was a welcome refresher on how encapsulation and hash maps work. I was able to figure out how to properly incorporate hash map usage inside the Operator parent class due to a helpful individual on Slack. I also was able to refresh on inheritance and abstraction for the operator classes. I had limited experience with the Java GUI and it was very interesting going through the given GUI framework and seeing how it all comes together to form the calculator.

8 Project Conclusion/Results:
Overall, I was able to complete Assignment 01 in its entirety and I believe that I completed all the requirements given to me in regards to completing the Evaluator file and filling out the functionality for the Operand, Operator, and the subclasses to the Operator parent class.