CSC 413 Project 1 Documentation

Summer 2024

Ulices Gonzalez

923328897

413.01

https://github.com/csc413-SFSU-SU2024/calculator-ulicessgg.git

Table of Contents

1 Introduction

1.1 Project Overview

The calculator project shows users a menu on their screen with basic calculator functions with numbers 0 through 9 and the main operations of addition, subtraction, multiplication, division, and exponential calculation. Parentheses are also available to use for more complicated calculations. When the calculator is used each button sends its information back to the calculator as it saves the info and when '=' is entered it will send all the numbers and operators entered to be used to calculate. For example, if a user enters (6*16)-(10+9) it will calculate inside the parentheses first and then do the rest to get 77. To do each calculation the calculator uses each operator separately and will use numbers entered with them to find the correct value. The other functions available are 'C' which will clear the calculator screen and 'CE' which will clear the last operator or number entered.

1.2 Technical Overview

The calculator is split into four primary files that handle the processes allowing users to evaluate different equations. Evaluator UI begins this process by constructing the user interface through its Constructor which allows users to give input through the form of button presses. When a user enters using the interface their input will then be read by a switch statement that will follow the process accordingly whether or not '=', ' C', or 'CE' have been entered. Each respectively performs a different case with the primary one being '=' which will then create an evaluator object and then call the evaluateExpression function. Using the delimiters " +/*-^()" evaluateExpression will iterate through two stacks, one for operators and the other for operands. When either or are found they will be pushed to their respective stacks. If a parenthesis is found it will be pushed to the operator stack if an open parenthesis but if closed it will then lead to the evaluation of operands present in the stack which will work backward until the open parenthesis is reached again. If no parentheses are present then the evaluator will move on to calculate based on members present in both the operand and operator stacks in the same fashion as was done with the parentheses but without them being present. After this Loop concludes it will then run one final calculation to account for the remaining sub-expressions present in the stacks and then return the final value to the user. Calculations are run by using the various operator subclasses created through the abstraction of the operator class. Each one is implemented with its respective priority level which is returned to the user one called and during the process of the evaluateExpression function they are used to return values after executing their respective process when two operands are passed to the operator. Each operator subclass is accessed through the operator abstraction that has a hashmap implemented that allows for each operator to be returned based on the token entered by the user.

1.3 Summary of Work Completed

To get the program running I implemented the operand and operator classes and using the operator abstraction implemented each subclass for operators required to run the calculations. From there the rest of my work included the implementation of a parentheses check in the evaluateExpression function, the correction of the new operator object, and the sub-expression handling at the end of the function. My work concluded with the action performed function to implement a switch statement to handle '=', 'C', and 'CE' and other button entries.

2 Development Environment

Version of Java Used: Oracle OpenJDK Version 22

IDE Used: Intellij IDEA 2023

3 How to Build/Import Your Project

1. In the command line enter
    a. git clone https://github.com/csc413-SFSU-SU2024/calculator-ulicessgg.git
    b. or download the zip folder from the repository linked
2. If using Intellij you can run the IDE and on the top right click Open.
    a. From here navigate to the directory in which the calculator folder is present and click ok
3. Once the project folder is open navigate to the EvaluatorUI Class folder
    a. To build Ctrl/CMND+F9

4 How to Run Your Project

1. After building the project you can run the calculator with the following
    a. To run with keystrokes use Shift+F10 or
    b. To run with the mouse and the IDE UI click the run button on the top right

5 Assumptions Made

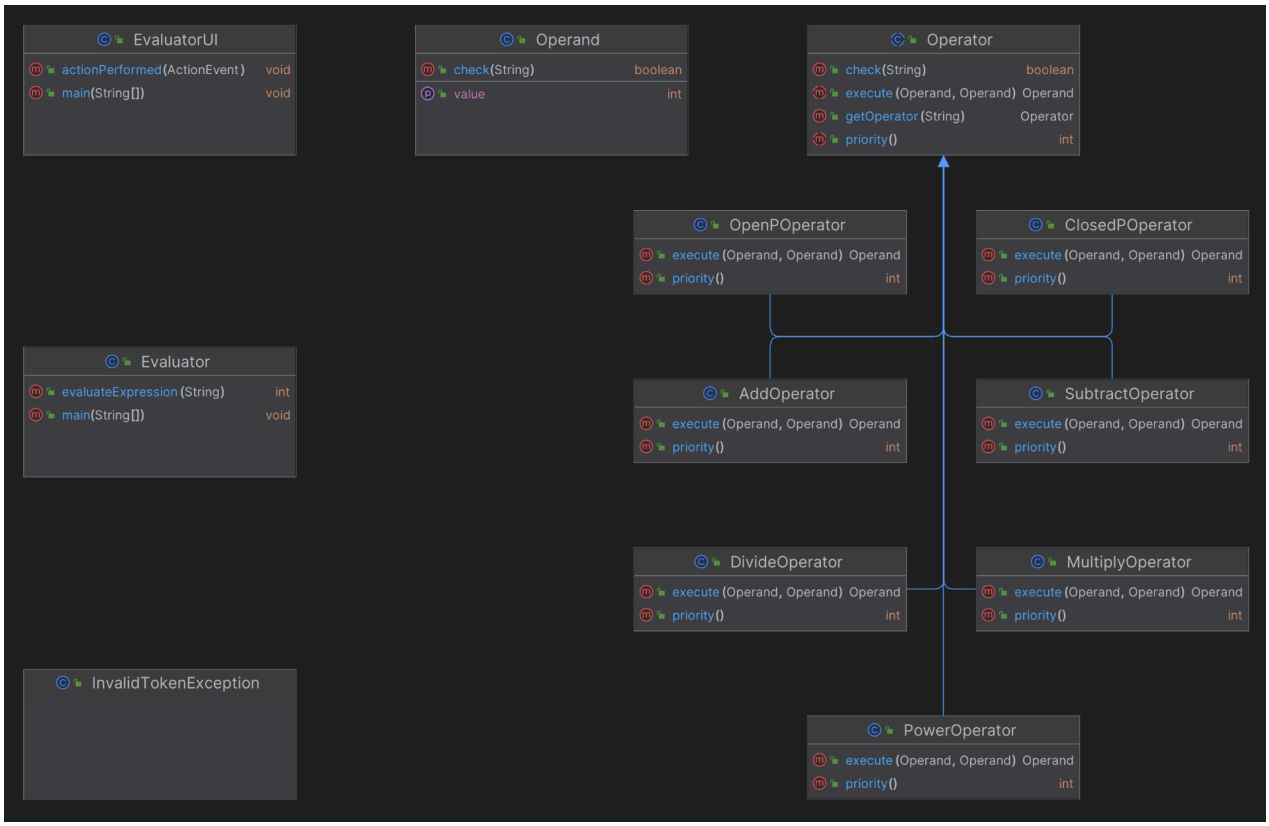
- No Decimals will be input
- No Negative integers will be accepted
- (and) can be treated as either operator subclasses or strings
- (and) will not perform multiplication
- Infinity will not be a returnable value for n/0
- The UI will not accept keystroke input
- Priorities follow the order of operations, PEMDAS. Except for parentheses
- Only a single hashmap would be created as opposed to one for each operator leading to the hashmap being a private static data structure
- String tokens must be converted into integers
- Operator subclasses will only have two defined functions while the abstraction of operator handles the remaining two (getOperator and check)
- evaluateExpression would need to be modified to handle (and) as new delimiters and operators
- Operands would each have a private integer value
- InvalidTokenException is not edited

6 Implementation Discussion

6.1 Design Choices

- Operands have private int value to store their numerical value.
- String input is converted to int for operand values.
- Try and catch statements using the conversion of string to int to check for valid operands
- getOperator will return the correct subclass from hashmap using get(token)
- Operators return predefined priority values and do not store them using integers
- EvaluatorUI actionPerformed uses a switch statement to determine the correct process for user input
- actionPerfomed uses a try and catch blocks for '='
- Implemented additional else if statements in evaluateExpression to handle (and) operators
-) when found will trigger a while statement to evaluate stack members until (is found
- newOperator is defined using getOperator
- The previously implemented while statement is repurposed to handle remaining subexpressions and once ended evaluateExpression returns calculated value to the UI

6.2 Class Diagram



7 Project Reflection

This project has been an interesting process since it's allowed me to refresh my skills in Java since I primarily work in C++ and C but it has shown me a lot of things that I could improve on. For the most part, syntactically I do need a bit of a refresher since there are a lot of features in Java that are allowed that I have only seen in documentation and textbooks but I haven't had to work with them. I think the most difficult part of this project that tested me on this was the abstraction since I'm accustomed to inheritance in C++, so that was the closest thing I could compare it to but I was a bit confused about how to properly implement subclasses from an abstract class. The only other thing I would say that I had not so much of an issue with but more so confusion on was exception since I haven't had to work with trying catch since I was in my community college which was a few years ago but for the most part it was a really easy process for me to get accustomed to especially when it came to the Boolean check methods and the UI implementation. Overall I'm really happy with my performance when it came to creating the calculator even though there were some areas I know for a fact I could have improved on it's at least given me a short list to pull from when it comes to improving my skills and catching up to make sure I can follow along the class.

8 Project Conclusion/Results

I have finished implementing all the necessary components of the calculator project. I was able to properly Implement all the necessary subclasses but also implemented the parentheses subclasses just in case in the future I can improve upon the evaluator to handle them more efficiently. There are a few edge cases that I haven't been able to solve but for the most part within the parameters of this project, they are more in line if we were to adapt this to work with decimals and other kinds of variables. As of now my project is working properly and can Output the correct values based on input fed to the UI. I was also able to clean up my diagram and have a better understanding of the workflow done behind the scenes using each class and how the valuator works properly as a standalone program from the UI program. As of now all of the operators work as expected including the parentheses which I have the most trouble with and have passed all unit tests provided to us.