CSC 413 Project Documentation

Summer 2021

Japheth Wun

921555593

CSC 413

https://github.com/csc413-su21/csc413-p1-finaldestroyer.git

- Introduction
- Project Overview

This program takes a mathematical expression from a user and produces a value from that expression. It will take a simple mathematical expression such as 1+2 to get 3 or 2*3 to get 6. Through the code it was supposed to accept a mathematical expression with and without parenthesis. The code itself would determine the order of operations within the given expression by using pemdas. After all the math was calculated, it would pop out the result for the given expression.

- ## Technical Overview

The program begins by asking the user for an expression. After reading the expression it will check if all the operators such as + , - , * , / and (,) ,exist with no other operators and check if all the operands are valid. After reading the expression, it will find any parenthesis and calculate the value found in the parentheses. By pushing it onto the stack, rinse repeat until no more parenthesis and then calculate this in the stack by popping out the numbers in the stack. When it pops two numbers out of the operand stack and one operator out of the operator stack, it will then check if it is valid and then gets the function for that operator and executes the operator and the two operands.

- ## Summary of Work Completed

Work was not yet completed.

To make this assignment/program work, I had to create classes for each of the operators such that addition, subtraction, division, multiplication, to the power of, Left and Right parenthesis had a class. In the Operator class, I created a hashmap that mapped its string expression to its mathematical function. When the addition was in the string, it would pop out the AddOperator function to calculate the expression. In the evaluator class, the function would read the string and push all numbers into a stack, but if it hits a left parenthesis, it would create a stack just until it hits a left parenthesis and calculate everything within that stack first. Then it would go back and calculate the rest of the stack and push the result so it would return and pop the result later on.

The biggest/most difficult problem was trying to get the program to recognize the parenthese, but it didn't work for me. So the only part that was left undone was taking parentheses correctly.

- # Development Environment

Java Version: 16.0.1

IDE Used: IntelliJ IDEA Community

- # How to Build/Import your Project

To build and import this project, it needs a java IDE, and then import the calculator class from the file which should import everything such as the operand, evaluator, and Operators classes.

- ## How to Run your Project

To make it run, you need to look at the EvaluatorUI and evaluator. To run the calculate program, you click on the run button next to the main function on the left of the EvaluatorUI. The program will then prompt you to enter a mathematical expression and will print the given expression and the result. If the expression is invalid, it will print out null.

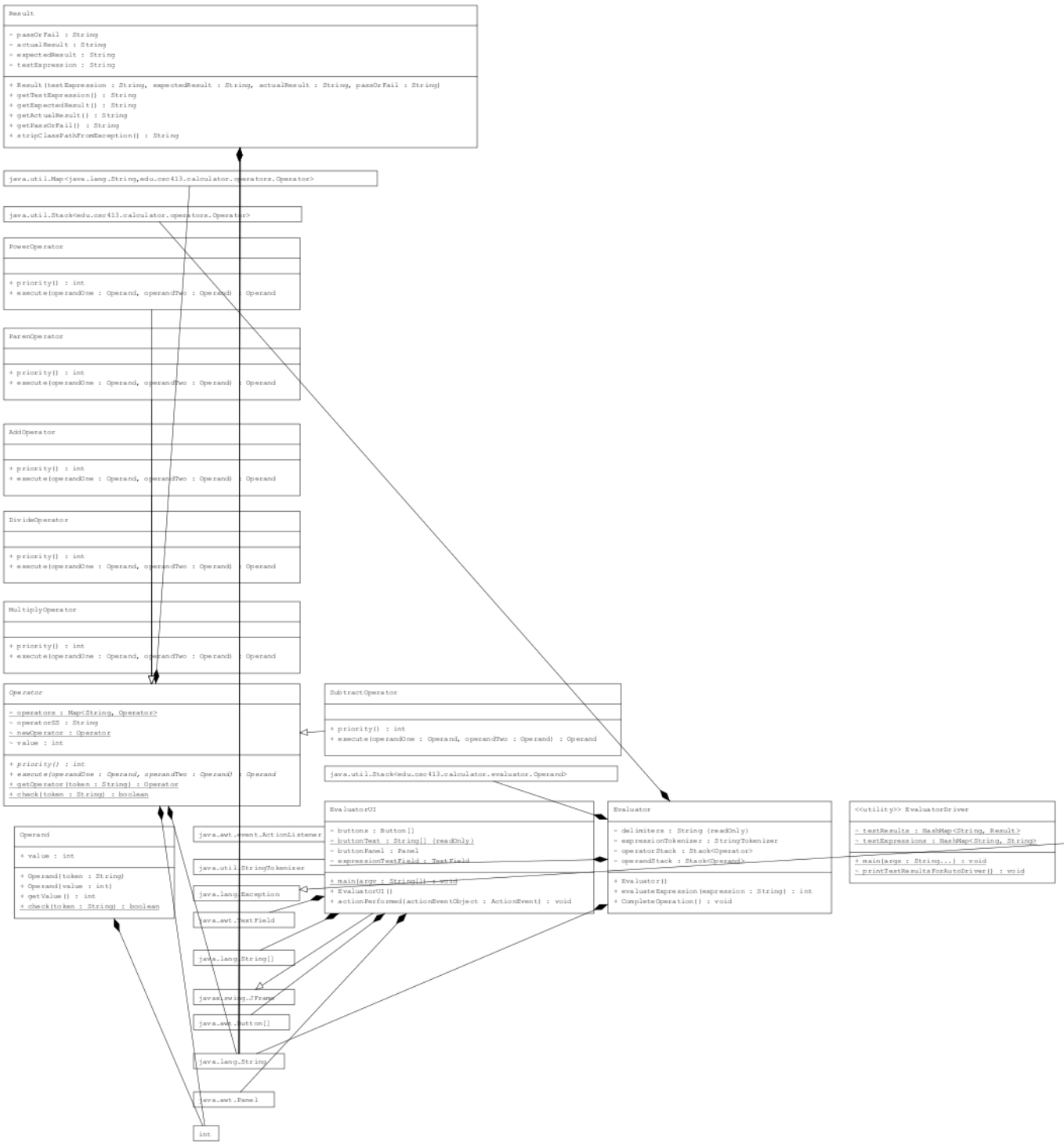
- ## Assumption Made

The assumptions made would be that the user would know how to enter numbers and operators for mathematical expressions. I assumed that if they were to enter a parenthesis, they would close the parenthesis. Another assumption would be that they only entered integers and no doubles, floats, or improper fractions. I also assumed that if they wanted to divide a number and it would equal an improper/proper fraction, they wanted it rounded.

- ## Implementation Discussion

Take in a mathematical expression and calculate it through the order of operations, PEMDAS .

- Class Diagram



- Project Reflection/Conclusion/Results

With only 1.5 weeks to finish this program, I was only able to complete 80% of the program. I wish I had more time to work on the program but we were on a time constraint due summer class going so fast. The first part of the program wasn't so difficult to understand but to put everything was a little more confusing. The hardest part was understanding when the user were to input a parentheses, it would create a new stack and push everything within the parenthesis into a stack and when it hit a right parenthesis, it would operate inside that stack and push it back to the original stack. After all that, it would calculate everything else through pemdas and when everything was done, it would pop out the result.