CSC 413-02

Infix Expression Calculator

Joshua Agulto

915304694

https://github.com/csc413-02-spring2019/csc413-p1-joshagulto

Introduction

Project Overview

This project was designed to test students on their knowledge on stacks. This project utilizes many different java aspects such as stacks, hashMaps, object-oriented classes. The project was given with the code almost completed. The students job was to create operator classes for each operator, i.e. *, +, -, /, and finish an almost completed evaluator, operand, and evaluator UI class.

Technical Overview

The purpose of this project was to utilize stacks to create an infix expression calculator. The idea of this project is to create a calculator that could accept math expression in infix form, i.e. 2 + 2 or 2 / 52. The calculator is designed to accept strings from the users, in the form of an infix expression, and converts the strings into two stacks, operator stack and operand stack, and then calculates the formula. The program uses a simple UI that allows the user to input strings using an on-screen number pad. Unfortunately, this program does not have keyboard support.

Completion

Unfortunately, the project was not fully completed. All the operator classes, the operand class, most of the evaluator class, are completed. The project passes most of the tests for the evaluator but there were scenarios where evaluator would not display the correct answer.

Build Information

This program was written using IntelliJ IDEA 2018.3.4 from JetBrains. The Java Development Kit that was used was jdk11.0.2.

You can download this project by using the specified link above. With the link, you have the option to download the repository in a zip file or use the link to clone the file via Git Bash. When the file is downloaded and unarchived, you can import the folder with any Java IDE. With most IDE's, you can use File > Import or File > Open Project to import the folder. Once the project is completely imported into your IDE of choice, all files will be accessible. In order to run the program, you can simply open the EvaluatorUI.class and press run. The GUI is simple and will immediately initiate the number pad on screen.

Implementation

This project uses stacks that utilize to object-oriented classes, Operator and Operand. The operator class is an abstract class that recognizes the string inputs the user gives and translates them into multiple operators. The operators used were addOperator, subtractOperator, multiplyOperator, divideOperator, powerOperator, openOperator, and closeOperator. All seven of these classes are subclasses that extend the main abstract operator class. The main operator class is simply an idea that all the sub class operators inherit. All operators come with a priority and an execute function. Each operator priority vary depending which operators will be executed first: add and subtract have priority 1, multiply and divide have priority 2, and power has priority 3. The openOperator and closeOperator are the parenthesis. Because their implementation differs from the regular operators, their priority is completely arbitrary. Each operator, excluding open and close, have their own execute function respective to their math operation. The operand class is a simple class that holds a private value and a simple function that returns that private value. The evaluator class implements both the operator and operand classes using stacks. The evaluator class translates a string and tokenizes the string per character and assigns each character to their respective class: numbers are instantiated as operands and operators are instantiated as operators. Using stacks and a simple algorithm, the program then processes all the operands and operators and returns a value that is displayed on screen. Evaluator has two methods: eval and process. Eval is the

main method that is called by the UI that handles all the sorting of strings and parsing. Process is a short method that pops two operands and one operator and executes these operands.

<u>Reflection</u>

My initial assumptions about this project is that it was going to a piece of cake. I have a decent coding background, so I saw no difficulty doing this project. When I first started this project, I was completely confused. Although I may know how to code and exactly what methods and functions, I should be using, I had difficulty understanding the algorithm used. I was initially using the algorithm at first, but I could not understand mathematically how it would work so I could not translate it into code. One problem with the code I had was I, for some reason, did not read the section saying, "If none of the above cases apply, process an **operator". Because I did not read this one step, I could not understand the algorithm for hours. Prior to finally understanding the code, I tried to come up with my own algorithm that could also suffice the program, although that failed miserably.**

One thing I wish I could have done was start the project sooner. Unfortunately, I do not have a laptop, so I am not readily able to code programs. Because I only have a PC at home, I can only do coding assignments before and after school and before and after work. Because of my schedule, I ended up procrastinating hard on the project overestimating my own personal knowledge. I assumed the project was going to be a lot simpler and easier, but I ultimately let myself down. My project was not bad, but it does still have some work. One thing I was not able to finish was the eval method. Within the method, I noticed that some tests were not resulting in the proper solution. I noticed that my process method was doing operators from the top of the stack to the bottom. This situation only works when the precedence of the operator increases as the expression goes on. If there was a case where there were 3 add/subtract operators and then 2 multiply/divide operators, it would execute the operands from the top of the stack to the bottom rather than the bottom of the multiply/divide stack to the top and then the bottom of the add/subtract stack to the top. I attempted to make my own algorithm that could understand that discrepancy my program had but in multiple cases I could not get the program to result in the proper solution.

The program does properly work with simple equations that only have one operator per level of priority. If there was a case where there were multiple operators of the same priority in a row, the program would not spit out the proper result. In the end, I am very disappointed in myself as a programmer that I could not finish the project. I have learned that deciding to procrastinate prior to understanding what the project entails is probably the worst decision. One thing I am glad about is that I decided to mess up in the beginning. Because of this, I know will not make the same mistakes again.