CSC 413 Calculator Documentation

Fall 2024

Dale Rivera

922792075

Section 1

GitHub Repository Link

Table of Contents

1 Introduction

1.1 Project Overview

This calculator app calculates basic math expressions that has a user-friendly interface. It can calculate basic arithmetic and supports parenthesis.

1.2 Technical Overview

This calculator app solves basic math operations using an algorithm that is similar to Dijkstra's shunting yard algorithm which uses two stacks to store operators and operands. Each math operator is a subclass and each calculation is done within these subclasses. The order of the calculations are based on their position on the stack and the operator's priority. Java Swing was used to create the user interface.

1.3 Summary of Work Completed

To get the assignment working, I had to create the operator subclasses and fix a few lines from the given skeleton code like the line that tries to instantiate an abstract class. One major issue I had was with the parenthesis. The issue was that I was trying to implement it like any other operator but after reading more about Dijkstra's two stack algorithm, I resolved the issue by treating the parenthesis as a flow control.

2 Development Environment

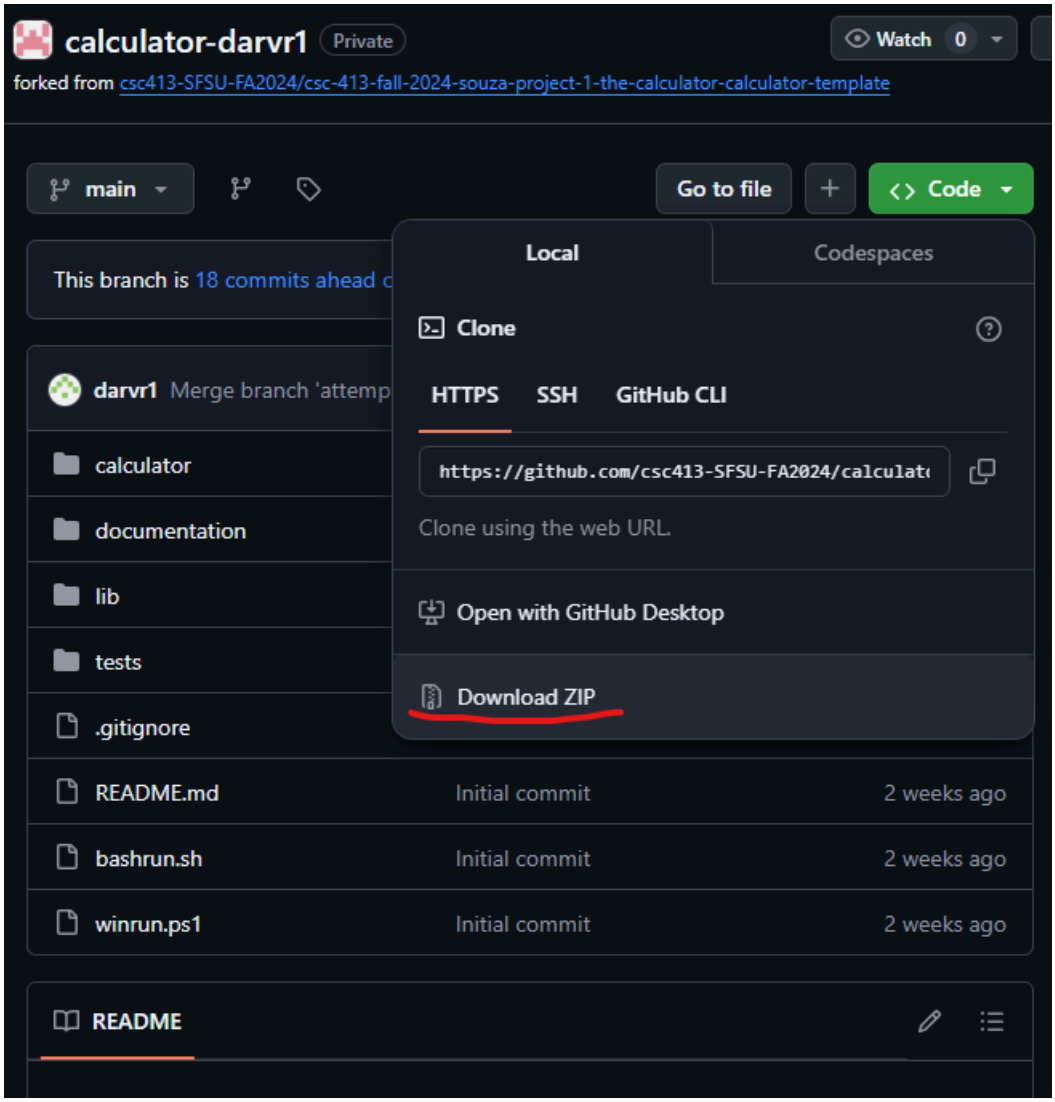
Java version: "22.0.1" 2024-04-16

IDE: IntelliJ IDEA 2024.2.0.2 (Ultimate Edition)

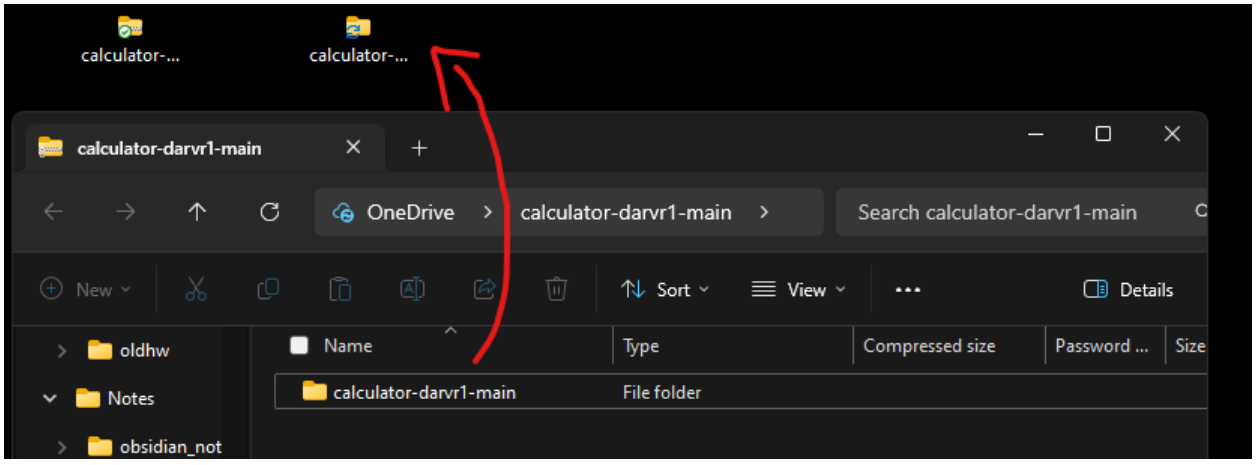
3 How to Build/Import your Project

This instruction is for windows.

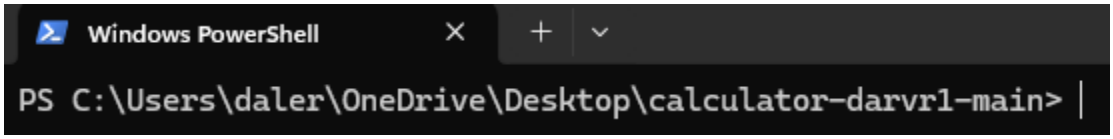
1. Download the zip file from GitHub.



2. Open the zip file then drag the content to your desktop (or extract it in your preferred way).



3. In powershell, set your directory to the downloaded file.



4. Compile from command line. **Due to formatting issues after converting to pdf, it is better to copy the commands in steps 4-5 from the README file.**

```
javac -d target .\calculator\evaluator\Operand.java
javac -d target .\calculator\operators\*.java
javac -d target .\calculator\evaluator\*.java
```

5. Compile test

```
javac -d target --class-path
"./;.\lib\junit-platform-console-standalone-1.9.3.jar"
.\tests\operator\*.java
javac -d target --class-path
"./;.\lib\junit-platform-console-standalone-1.9.3.jar"
.\tests\operand\*.java
javac -d target --class-path
"./;.\lib\junit-platform-console-standalone-1.9.3.jar"
.\tests\*.java
```

4 How to Run your Project

1. Run on the terminal.

   Due to formatting issues after converting to pdf, it is better to copy the commands in steps 1, 2, and 3 from the README file.

   Replace expression with the expression you want to calculate (**Wrap the expression in between quotation marks.** e.g. **“**(2+3)*5**”**).

```
java -cp target calculator.evaluator.Evaluator “expression”
```

2. Run unit test

```
java -jar .\lib\junit-platform-console-standalone-1.9.3.jar -cp
```

```
target --scan-classpath
```

3. Run GUI

```
java -cp target calculator.evaluator.EvaluatorUI
```

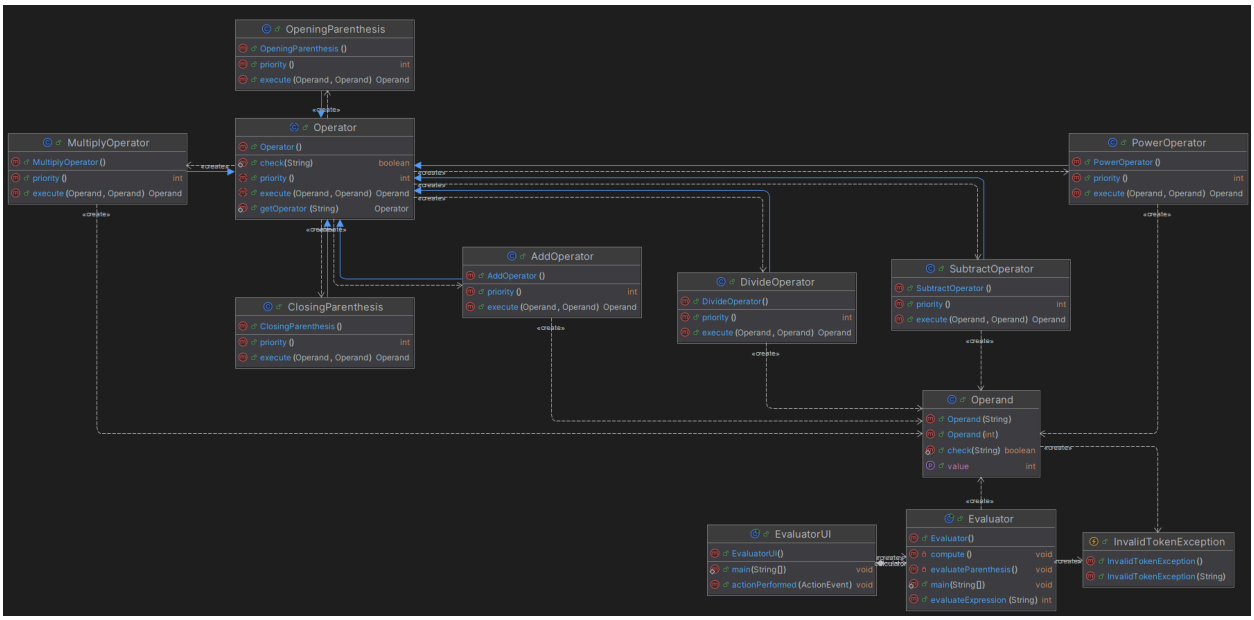
5 Assumption Made

1. The first number cannot be negative because the algorithm expects infix notation.
2. Operands must be integers.
3. We create subclasses for the parentheses.
4. Parentheses cannot be used for multiplication. Same reason as #1.

6 Implementation Discussion

A design choice I made when it comes to parentheses was to only push the opening part. The closing parenthesis serves as a signal to start evaluating inside the parentheses so an if statement is used to check for a closing parenthesis. When it finds one, it starts evaluating by popping the operands and operators from their respective stacks until it hits an opening parenthesis. Another design choice I made for the GUI was to not do anything to the front-end when a user inputs something invalid. My reason behind this is because what if someone inputs a long equation and accidentally misclicks the equals sign button. It feels awful when the calculator erases the entire expression when the user could've easily fixed it. However, it can be odd to other users as they may start thinking that the calculator isn't working properly but in reality they did not notice that there is an error in their expression. If I knew more about the Swing library, I would've made the expression field have a red tint if something invalid was inputted.

6.1 Class Diagram



This image can also be found in the documentation folder.

7 Project Reflection

Since I haven't used Java in a while, this project was a very good refresher on how to use the language but also OOP as well. I also had fun figuring out how Swing works since this was my first time working with it and I used to play with a graphics library when I first started learning programming. I did procrastinate a lot but whenever I worked on this project, I was in the zone and having fun.

8 Project Conclusion/Results

This project meets all the requirements that are listed on canvas. An improvement (or downgrade) I would add is to make CE function similar to Windows or older calculators where it would clear the entire entry instead of removing the last character. This was how I originally had it implemented but it was stated in class to not use any loops in `actionPerformed()`. It was possible to do it without any loops but I either need a long list of if-else statements or a deeply nested `Math.max()`. As a result, I replaced it with one line that deletes the last character.