

CSC 413 Project Documentation

Summer 2022

Timmy Tram

921102465

CSC 413-01

<https://github.com/csc413-SFSU-Souza/csc413-p1-TimmyTram>

Table of Contents

1	Introduction	3
1.1	Project Overview	3
1.2	Technical Overview	3
1.3	Summary of Work Completed	3
2	Development Environment	3
3	How to Build/Import your Project	3
4	How to Run your Project	6
5	Assumption Made	10
6	Implementation Discussion	10
6.1	Class Diagram	10
7	Project Reflection	11
8	Project Conclusion/Results	12

1 Introduction

1.1 Project Overview

The purpose of this project was to create a basic, but functional calculator application that users can use to emulate a physical calculator. This means this calculator follows PEMDAS and can perform parentheses, exponents, multiplication, division, addition and lastly subtraction.

1.2 Technical Overview

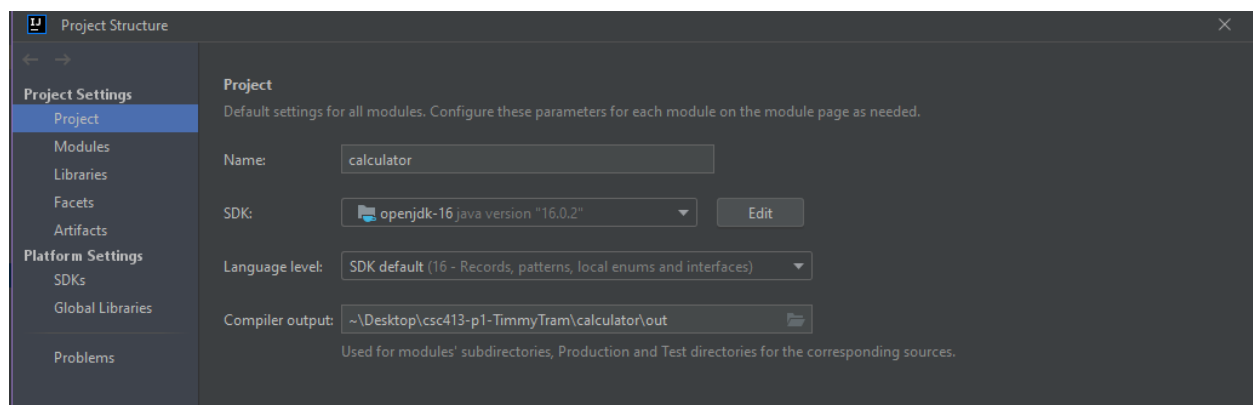
This project's focus was to implement an expression evaluator that takes in an expression from buttons on a GUI that then tokenizes the expression to run an algorithm that uses two stacks that can break down the expression into simpler pieces and eventually evaluate the expression that mathematically follows PEMDAS and outputs the result back onto the GUI. These simple pieces were operands and operators. The idea of an operator is abstract, but usually the operators have different levels of precedence and perform different kinds of actions. This means all PEMDAS operations derive from this abstract idea of an operator.

1.3 Summary of Work Completed

In this project I was able to fully implement the Operand class, the abstract Operator class and created sub-classes that extend the Operator class such as, AddOperator, SubtractOperator, DivideOperator, MultiplyOperator, PowerOperator, LeftParenthesisOperator, and RightParenthesisOperator. Additionally, I was able to complete the algorithm given in the Evaluator.java file as well as complete the EvaluatorUI.java file. I was able to pass all the given unit tests and the GUI functions as intended with the C button clearing the text field and CE acting as a backspace.

2 Development Environment

- a. Version of Java used: openjdk-16 or Java version 16.0.2



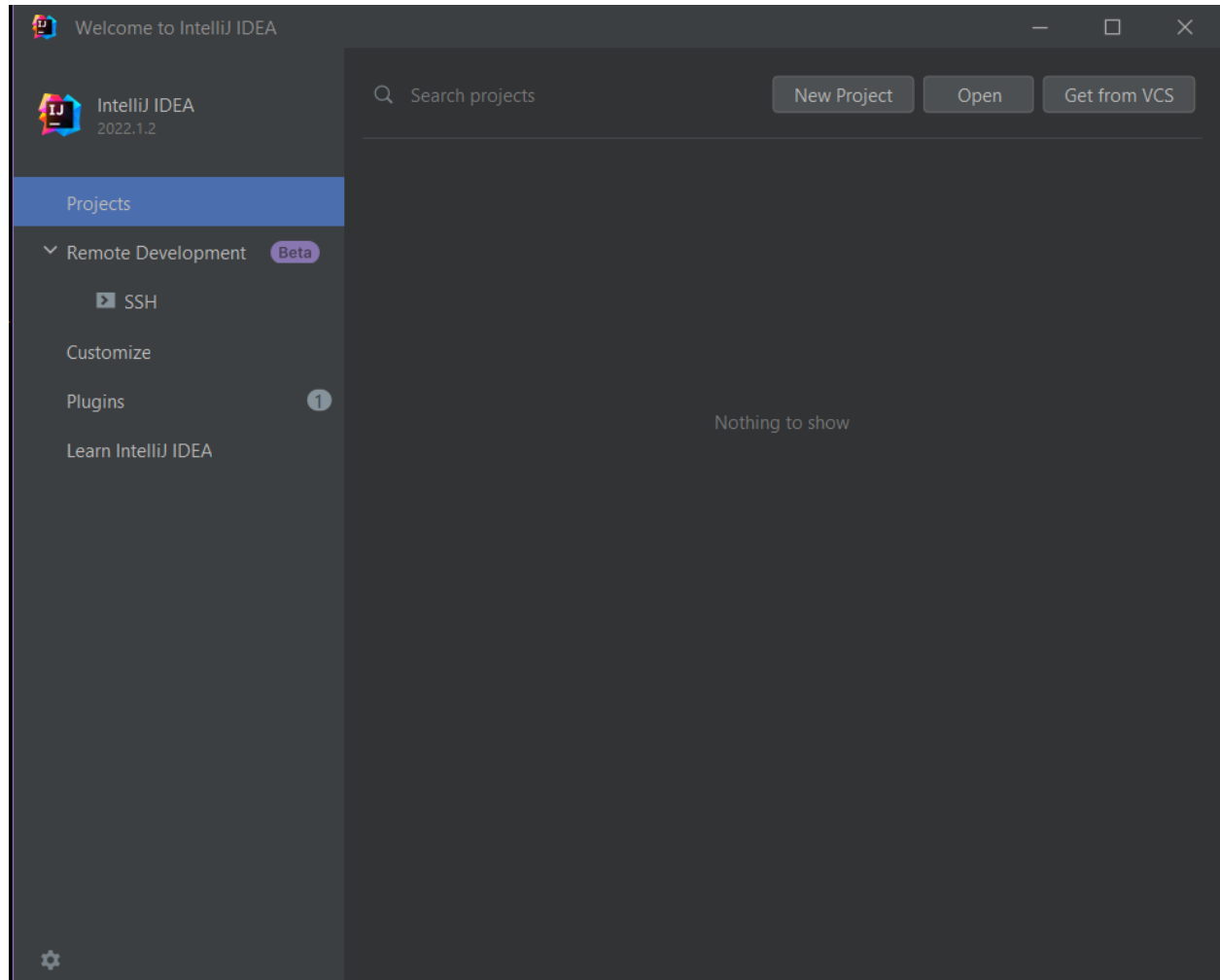
- b. IDE used: IntelliJ IDEA Ultimate Edition 2022.1.2

3 How to Build/Import your Project

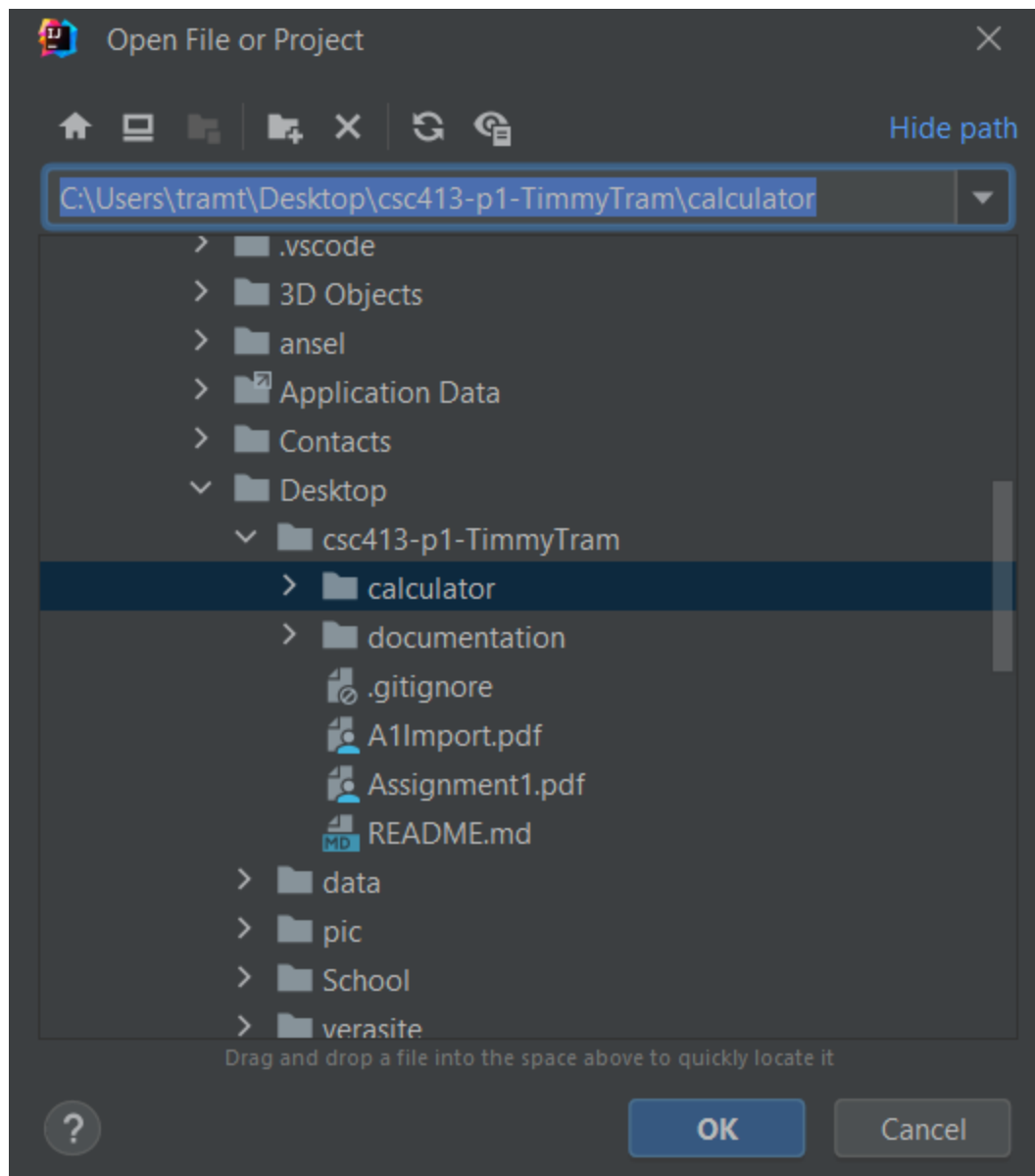
Open a terminal of your choice and type: `git clone https://github.com/csc413-SFSU-Souza/csc413-p1-TimmyTram.git`

```
Command Prompt
C:\Users\tramt\Desktop>git clone https://github.com/csc413-SFSU-Souza/csc413-p1-TimmyTram.git
```

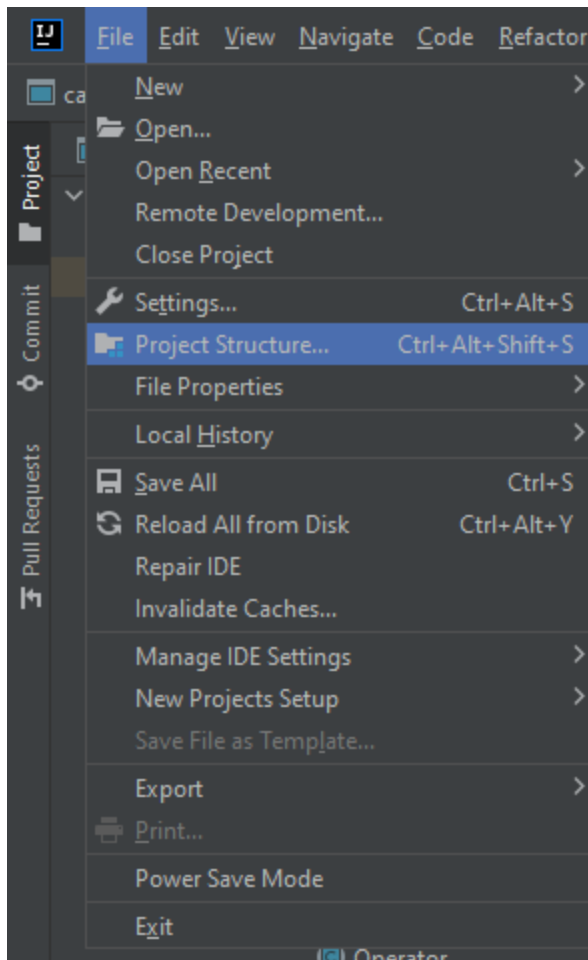
Select Open on IntelliJ IDEA



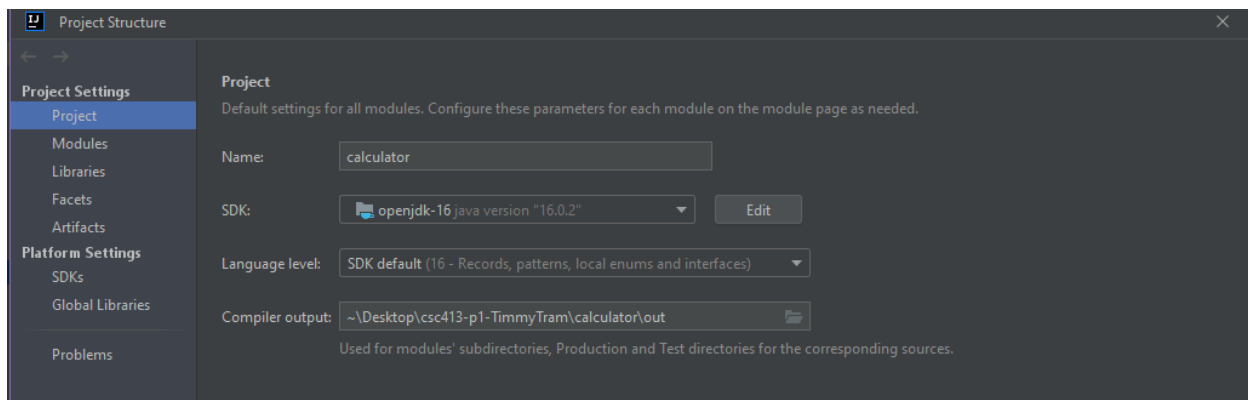
Select the calculator folder as the source under the csc413-p1-TimmyTram folder



Then under File click Project Structure.

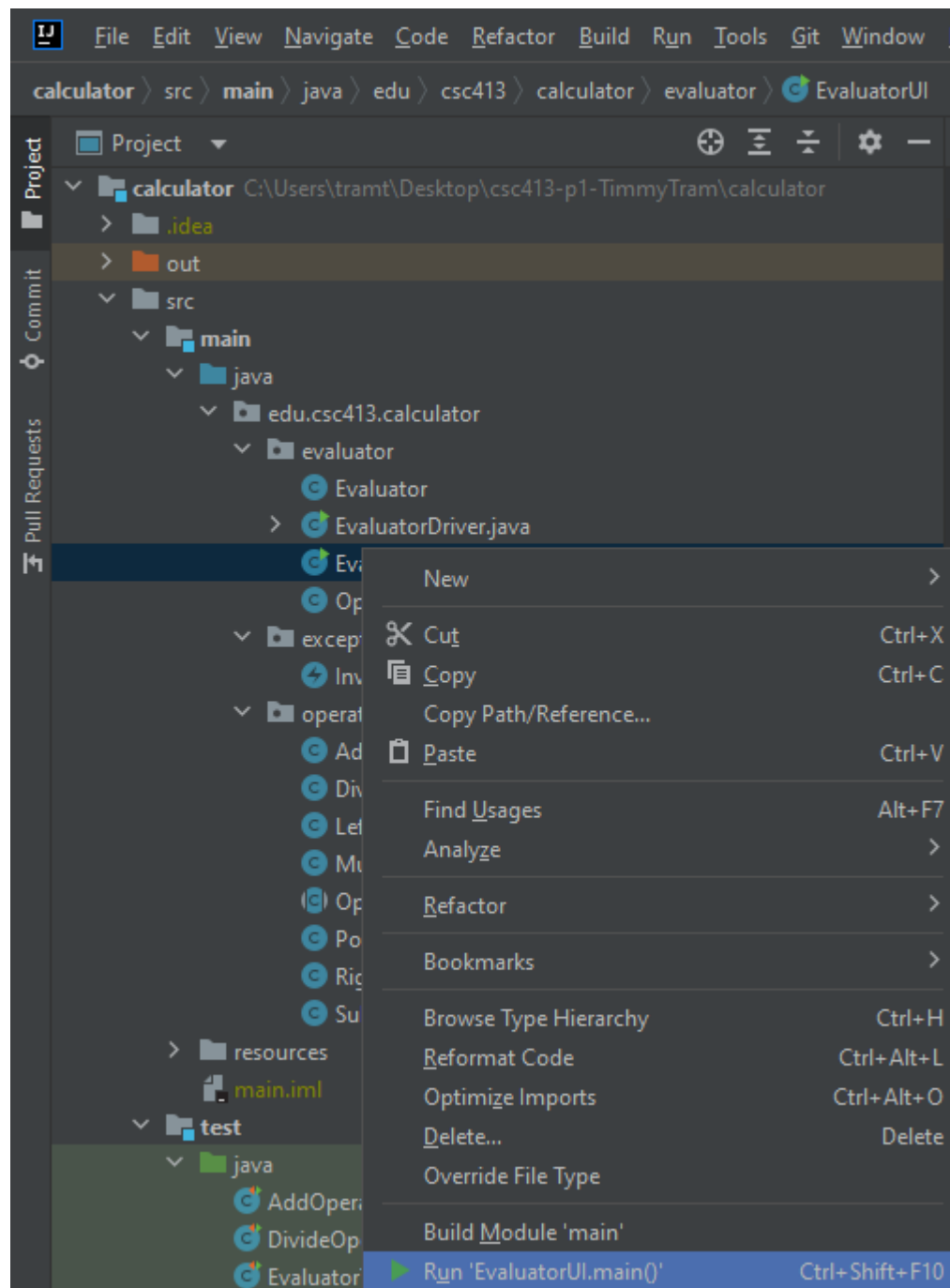


In project structure window, make sure the SDK is at least some version of Java 16 and set the language level to SDK default (16 – Records, patterns, local enums and interfaces) then hit apply and ok.

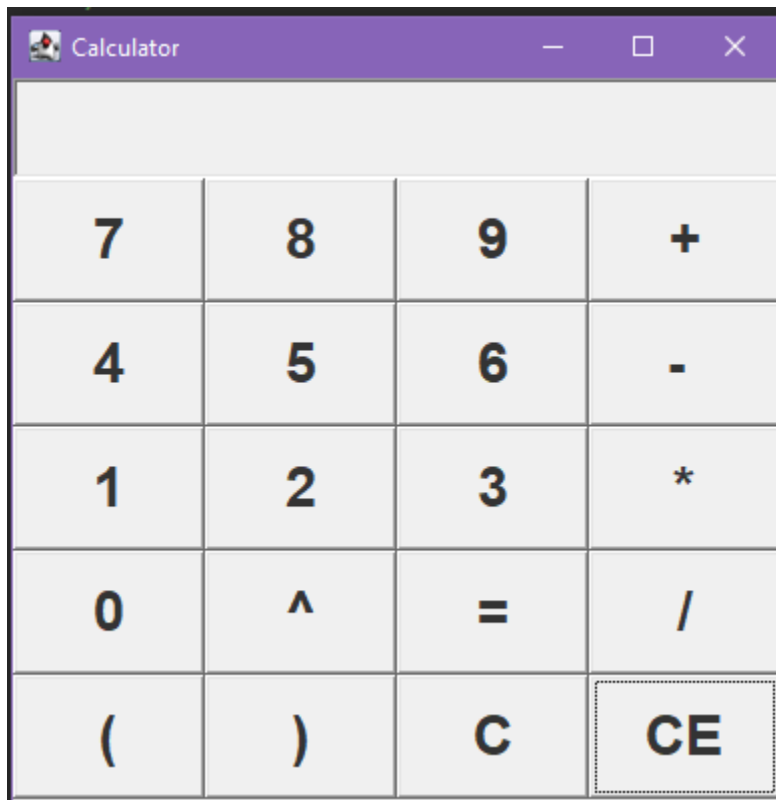


4 How to Run your Project

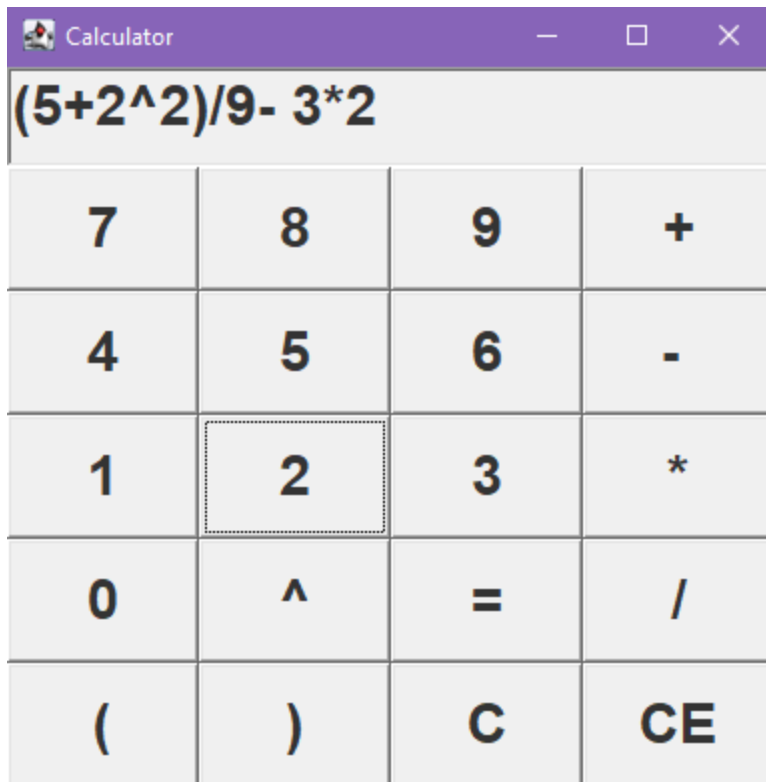
After successfully importing the project and configuring the SDK you can run the EvalutorUI.java file by right clicking the file and clicking Run "EvalutatorUI.main()"



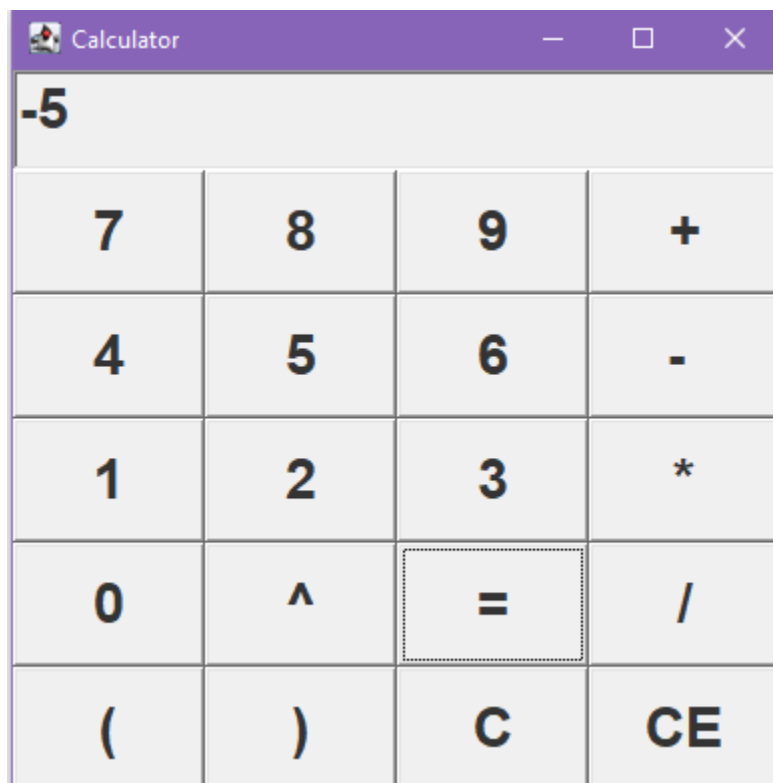
After clicking run you should be presented with a GUI



You should then be able to click on the buttons and create expressions



Pressing the "=" button will result in the evaluation of the given expression displayed in the text field

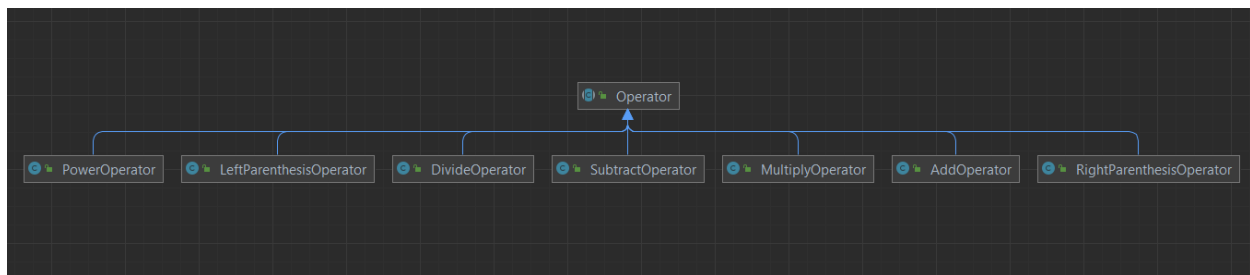
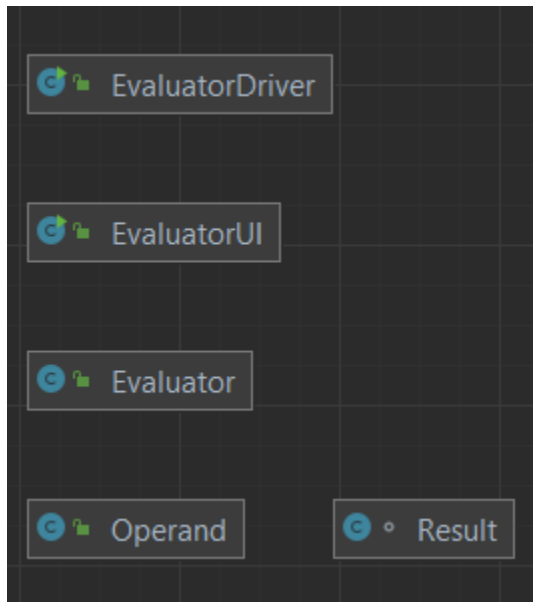


5 Assumption Made

In this project I assumed that we would only be dealing with integer numbers that are also non-negative. This means I would not expect fractions like $\frac{1}{2}$ or decimals such as 0.302 or negative numbers like -5 being inputted.

6 Implementation Discussion

6.1 Class Diagram



When designing this project, I initially looked at the unit tests to see what exactly I am expected to implement when it came to operators. I then realized implementing most of PEMDAS operators were simple since I only needed to implement two methods for each of the subclasses which was returning their priority and how they execute their specific operation. I then ran the respective unit test for each operator and was able to get it working so I knew I was on the right track.

I then cleaned up the evaluator algorithm by moving the body of the while loop into a private function called process() which does the same steps as I read in the given article on how to evaluate infix expressions. This is because I knew I would probably have to reuse this code multiple times, so I wanted to keep things DRY. I then completed a part of the algorithm that suggested that we process the stack until it was empty and was able to complete most of the basic equations like $1+2*3$ successfully and consistently.

The hardest part when it came to implementation was the parenthesis. I wondered whether it would be better to have them as one singular operator class or to have them as two separate operator classes. Additionally figuring out their priority and execute function was confusing since theoretically, the parenthesis operator does not actually execute anything, so I kept it as is.

Then I started work on the evaluator algorithm to get it to properly evaluate parentheses. I created a single if else statement to deal with the closing/right hand parenthesis since it was the most important part of the parenthesis in comparison to the open/left hand parenthesis. My thought process was that if we encounter a ")" then we should process or evaluate each operand and operator on the stacks until we find the "(" operator and lastly pop the operator stack to get rid of the "(" . In the else statement, this is where we would normally process operators with different priorities, but I modified the while loop so that there was no way we accidentally processed a "(" and call an execute function on accident because it would return null. This means that "(" would simply just get pushed onto the operatorStack and is just used as a marker for the while loop in the if statement.

The last part of implementation was the actionPerformed function in the EvaluatorUI.java file. The purpose of this function was simply to perform an action when a button is pressed. There are technically only 4 unique different button actions we needed to perform.

The first unique one was the "=" where I simply needed to pass the expression to the Evaluator object and perform the algorithm and return the value back to the GUI. I also needed to place most of the code in that if body as a try and catch block since evaluateExpression can throw InvalidTokenExceptions and my catch was to simply just empty out the text field.

The next else if was simply to check if I clicked "C" which would set the text field equal to an empty string to clear the text field.

The last else if was to check "CE" where I would simply just display the substring of the current string in the text field minus the last character.

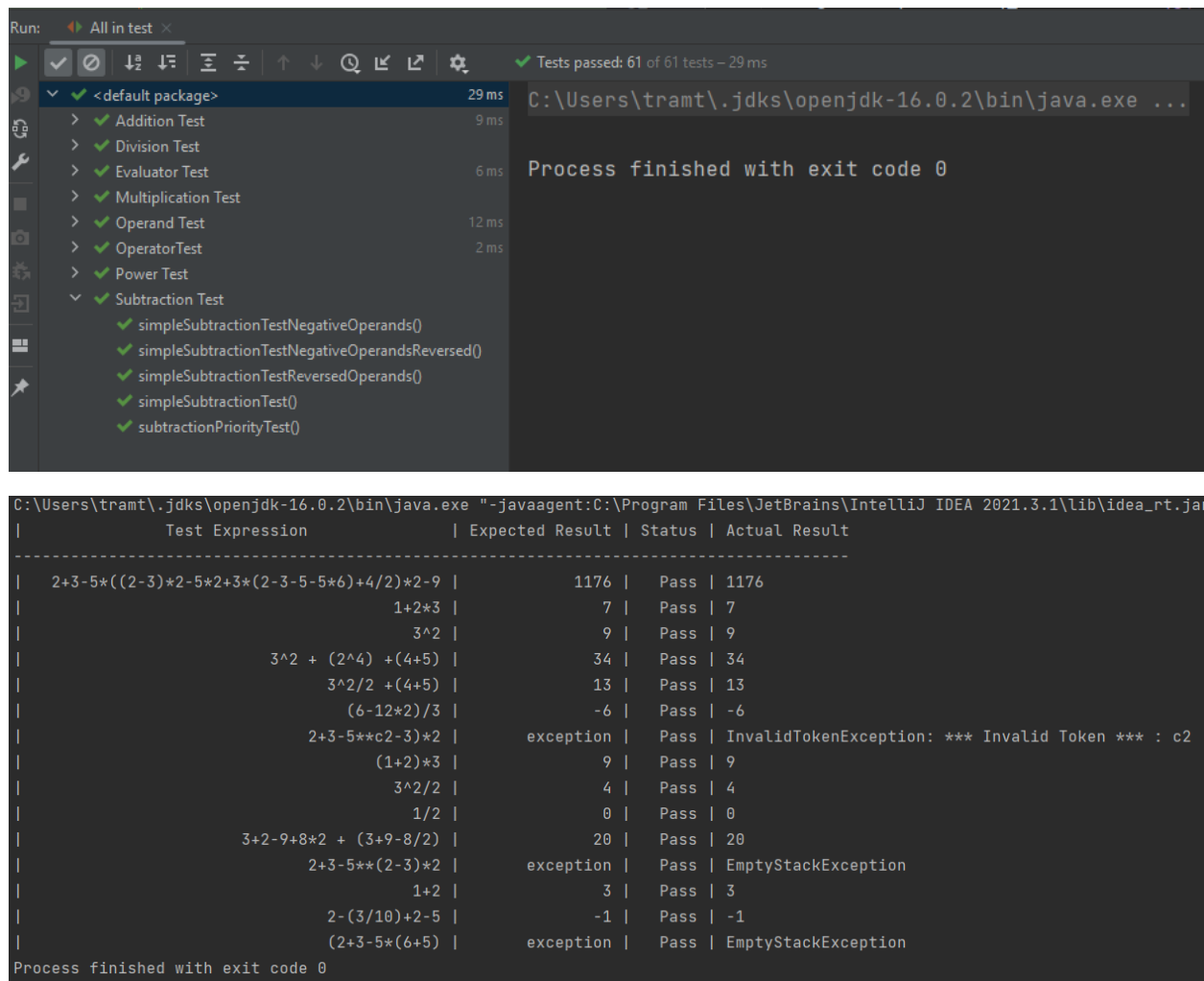
The else statement at the end covers all the numerical buttons and operators and I simply needed to get the current text field and concatenate the button's text to update the text field.

7 Project Reflection

This project was interesting since it has been a while since I worked with a project with this many files. I really liked the addition of unit tests since it was able to provide insight as to what is expected for each class to do. Additionally, it made to easier since I would not have to manually write the tests myself. I am satisfied with the code I wrote for the operator's package and Operand file as well as the actionPerformed function for the EvaluatorUI since I do not think I can make much major changes to it. The one thing I think needs improvement was probably the evaluator algorithm because I felt like the way I implemented it makes the parenthesis operator classes a bit useless since I never use the priority or execute function. I only really had those specific operators to add to the HashMap of operators so I could get their classes and compare it to a character for my conditional statements in order to process () properly.

8 Project Conclusion/Results

In conclusion, the project works as expected as I was able to pass all the unit tests and the GUI can perform the same calculations in the given unit tests as well. I additionally, tested with my own expressions like $((2+2))$ to make sure the algorithm can process the parenthesis properly which it was able to do. I also made sure to run auto on the EvaluatorDriver and it was able to provide all the expected results as well. Thus, this project works as expected.



The screenshot shows the IntelliJ IDEA interface. The top toolbar indicates 'Tests passed: 61 of 61 tests - 29 ms'. The left sidebar shows a tree view of tests under '<default package>'. The right pane shows the console output.

Test Results:

- > ✓ <default package> 29 ms
 - > ✓ Addition Test 9 ms
 - > ✓ Division Test
 - > ✓ Evaluator Test 6 ms
 - > ✓ Multiplication Test
 - > ✓ Operand Test 12 ms
 - > ✓ OperatorTest 2 ms
 - > ✓ Power Test
 - > ✓ Subtraction Test
 - ✓ simpleSubtractionTestNegativeOperands()
 - ✓ simpleSubtractionTestNegativeOperandsReversed()
 - ✓ simpleSubtractionTestReversedOperands()
 - ✓ simpleSubtractionTest()
 - ✓ subtractionPriorityTest()

Console Output:

```
C:\Users\tramt\.jdk\openjdk-16.0.2\bin\java.exe ...  
Process finished with exit code 0
```



```
C:\Users\tramt\.jdk\openjdk-16.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.3.1\lib\idea_rt.jar  
| Test Expression | Expected Result | Status | Actual Result |  
|-----|-----|-----|-----|  
| 2+3-5*((2-3)*2-5*2+3*(2-3-5*6)+4/2)*2-9 | 1176 | Pass | 1176 |  
| 1+2*3 | 7 | Pass | 7 |  
| 3^2 | 9 | Pass | 9 |  
| 3^2 + (2^4) + (4+5) | 34 | Pass | 34 |  
| 3^2/2 + (4+5) | 13 | Pass | 13 |  
| (6-12*2)/3 | -6 | Pass | -6 |  
| 2+3-5*c2-3)*2 | exception | Pass | InvalidTokenException: *** Invalid Token *** : c2 |  
| (1+2)*3 | 9 | Pass | 9 |  
| 3^2/2 | 4 | Pass | 4 |  
| 1/2 | 0 | Pass | 0 |  
| 3+2-9+8*2 + (3+9-8/2) | 20 | Pass | 20 |  
| 2+3-5*((2-3)*2 | exception | Pass | EmptyStackException |  
| 1+2 | 3 | Pass | 3 |  
| 2-(3/10)+2-5 | -1 | Pass | -1 |  
| (2+3-5*(6+5) | exception | Pass | EmptyStackException |  
Process finished with exit code 0
```