CSC 413 Project Documentation

Fall 2018

Ahmad El Shakoushy

Assignment #1: Expression Evaluator

915814671

CSC 413-01

https://github.com/csc413-01-fa18/csc413-p1-aelshako

Table of Contents

1 Introduction

1.1 Project Overview

The purpose of this project was to make a calculator that is able to calculate the result of simple mathematical expressions. Each valid expression can only include operators or operands. Operands are simply integers in the implementation of this project. On the other hand, operators are the typical mathematical operation symbols (+ , - , / , ^, *) as well as parenthesis which are used to enclose certain parts of the expression. The first step was to get the calculator working without a graphical interface. A set of what are known as 'unit tests' was utilized in order to test the program with many different cases. Naturally, the next step of the design process was to make the program function with what's known as a graphical user interface (GUI). A GUI simply means that any user can use the expression calculator without any background knowledge or technical experience of computer science.

1.2 Technical Overview

This section will serve as a brief technical overview of the project. Please visit section 7 of this documentation for a more detailed explanation. The purpose of this project was the design and implementation of an infix expression calculator. There were multiple important classes within the implementation: Operator(abstract), Operand, Evaluator, EvaluatorUI. One of the most important classes in the project is the Operator class. The Operator class is an abstract class that includes an instance of a Hash Map. Each operator object belongs to a subclass of the Operator class. In other words, there is a subclass for the addition operator, the subtraction operator, and so on. The Hash Map is used to retrieve instances of those subclasses where the key is a string token holding the operator.

The Operand class includes a constructor to make an Operand object from a string token. It also includes a value variable, and its respective getters and setters. Lastly, it includes a check function to check if a string token represents a valid operand.

The actual infix expression evaluation is done within the Evaluator class via the utilization of two stacks: one for Operators and one for Operands. The Evaluator class also uses a StringTokenizer to split the input expression which is read in as a string into operators and operands. EvaluatorUI is simply the class that ties the logic in the Evaluator function with a GUI.

1.3 Summary of Work Completed

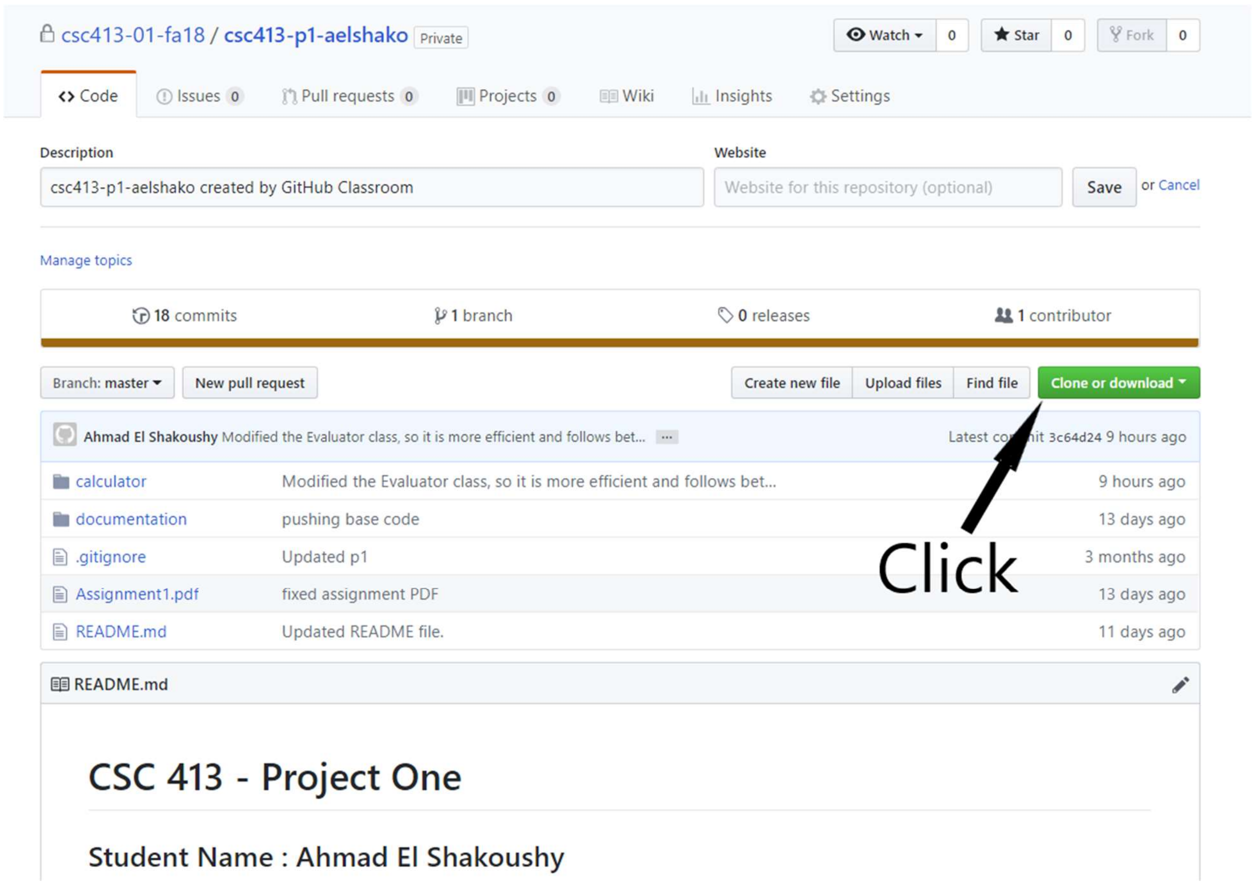
I have completed the implementations of all of the aforementioned classes. I've run all of the unit tests as well as the entire test collection. My implementation succeeded in passing all the tests. The EvaluatorUI class is also fully functioning with the inclusion of error checking for invalid expressions.

2 Development Environment

a. Version of Java used: Java 10.0.2

b. IDE used: IntelliJ Idea

3 How to Build/Import your Project


csc413-01-fa18 / csc413-p1-aelshako Private
Watch 0 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings
Description
Website
Website for this repository (optional)
Save or Cancel
Manage topics
18 commits 1 branch 0 releases 1 contributor
Branch: master New pull request Create new file Upload files Find file Clone or download
Click
README.md

Description: csc413-p1-aelshako created by GitHub Classroom

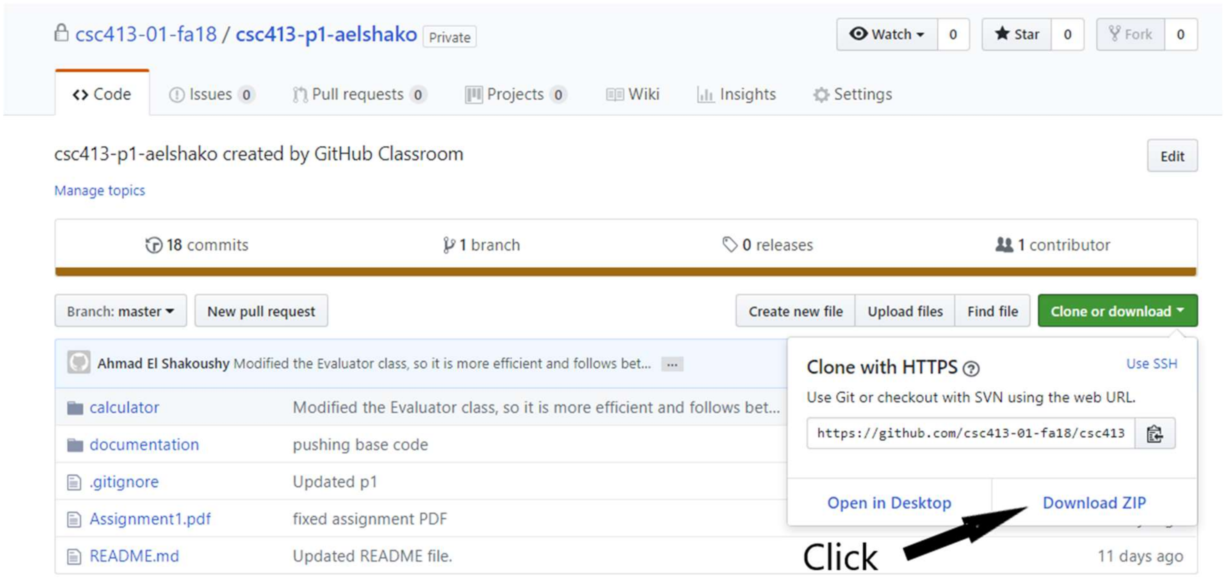
Ahmad El Shakoushy Modified the Evaluator class, so it is more efficient and follows bet... Latest commit 3c64d24 9 hours ago

calculator	Modified the Evaluator class, so it is more efficient and follows bet...	9 hours ago
documentation	pushing base code	13 days ago
.gitignore	Updated p1	3 months ago
Assignment1.pdf	fixed assignment PDF	13 days ago
README.md	Updated README file.	11 days ago

CSC 413 - Project One

Student Name : Ahmad El Shakoushy

Figure(1a): Click the green 'Clone or download' button in the repo link.


csc413-01-fa18 / csc413-p1-aelshako Private
Watch 0 Star 0 Fork 0
Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings
Edit
Manage topics
18 commits 1 branch 0 releases 1 contributor
Branch: master New pull request Create new file Upload files Find file Clone or download
Clone with HTTPS
Use SSH
Open in Desktop
Download ZIP
Click

csc413-p1-aelshako created by GitHub Classroom

Use Git or checkout with SVN using the web URL.

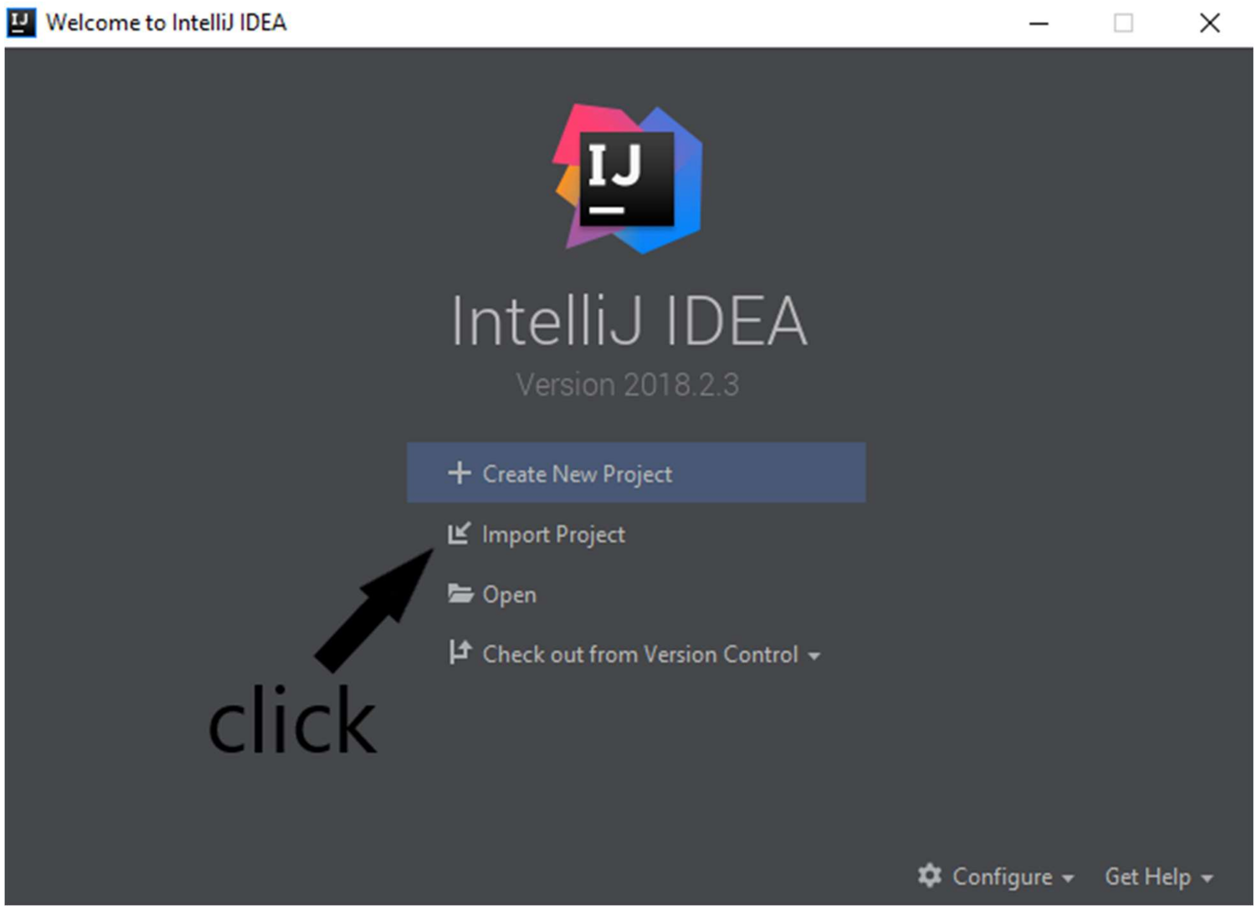
```
https://github.com/csc413-01-fa18/csc413
```

Ahmad El Shakoushy Modified the Evaluator class, so it is more efficient and follows bet...

calculator	Modified the Evaluator class, so it is more efficient and follows bet...	
documentation	pushing base code	
.gitignore	Updated p1	
Assignment1.pdf	fixed assignment PDF	
README.md	Updated README file.	11 days ago

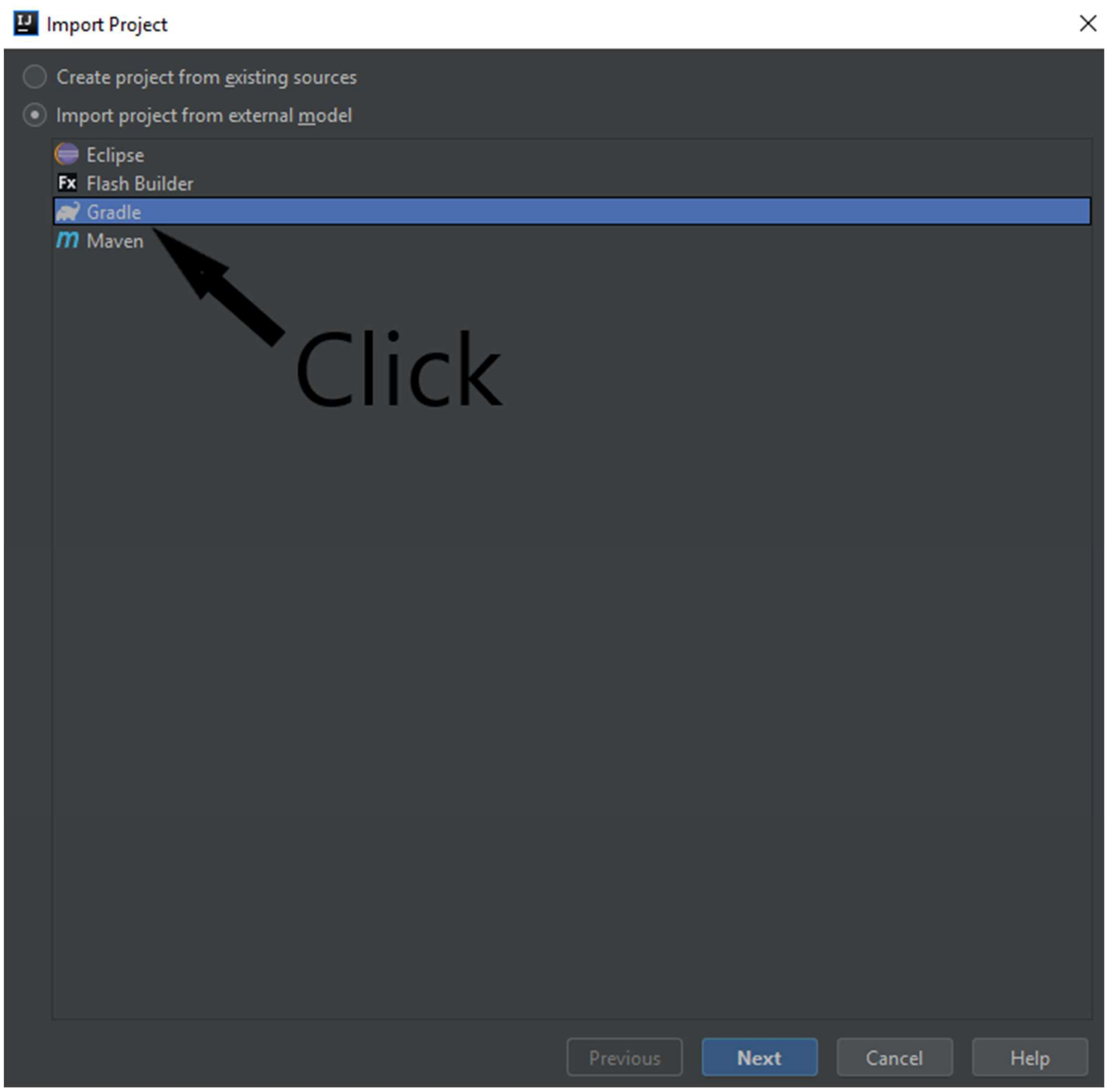
Figure(1b): Click the 'Download ZIP' button

- Extract the zip folder to a known location

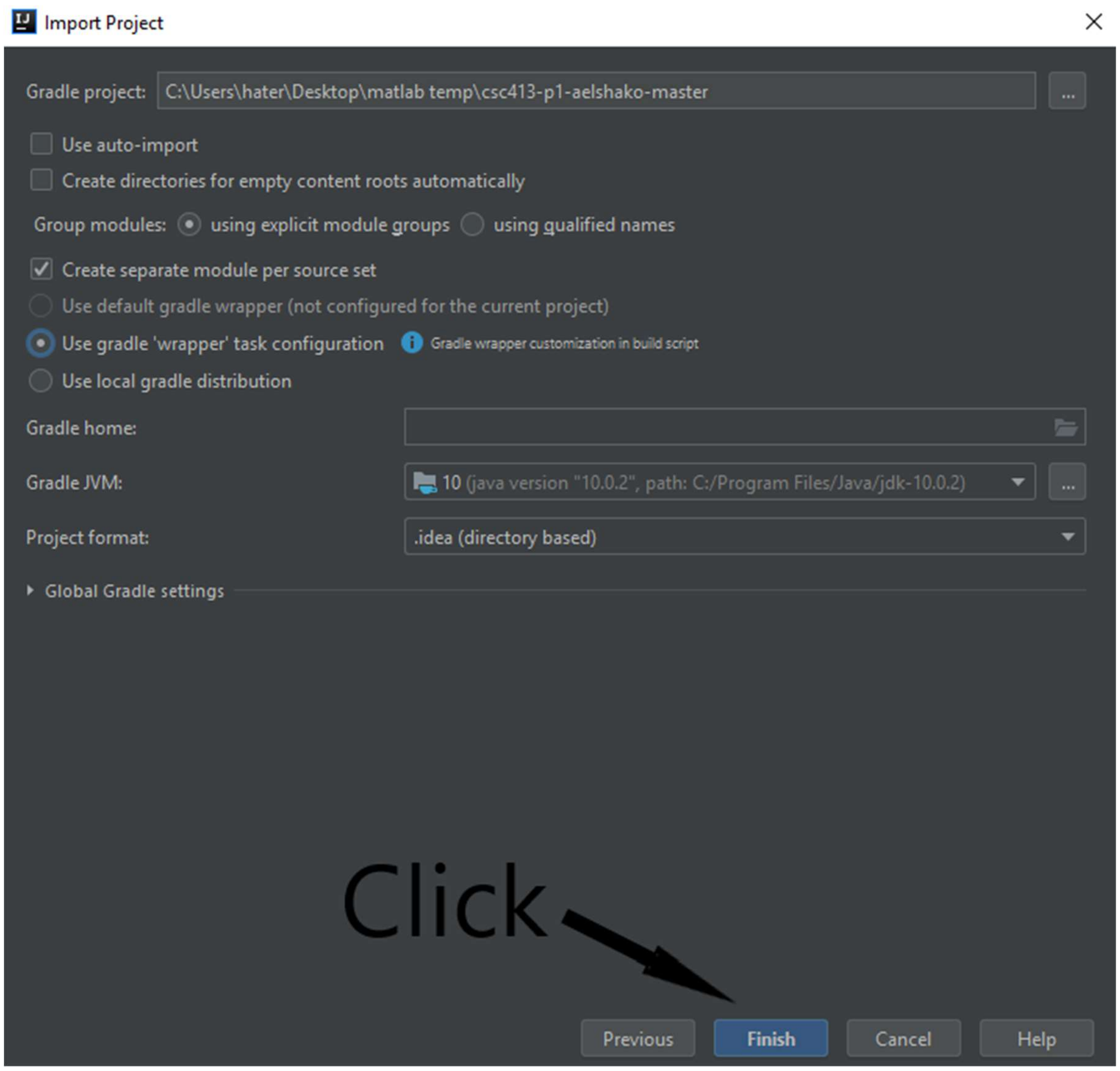


Figure(1c): Click the 'Import Project' button.

- On next menu, select the parent folder which you received from the previous extraction and hit 'OK'.



Figure(1c): Select 'import from external model' and click 'Gradle' then 'Next'. **(NOTE: <u>Make sure that you have Gradle installed before proceeding.)</u>**


Import Project

Gradle project: C:\Users\hater\Desktop\matlab temp\csc413-p1-aelshako-master

[ ] Use auto-import

[ ] Create directories for empty content roots automatically

Group modules: [✓ using explicit module groups] using qualified names

[✓] Create separate module per source set

Use default gradle wrapper (not configured for the current project)

[✓ Use gradle 'wrapper' task configuration] Gradle wrapper customization in build script

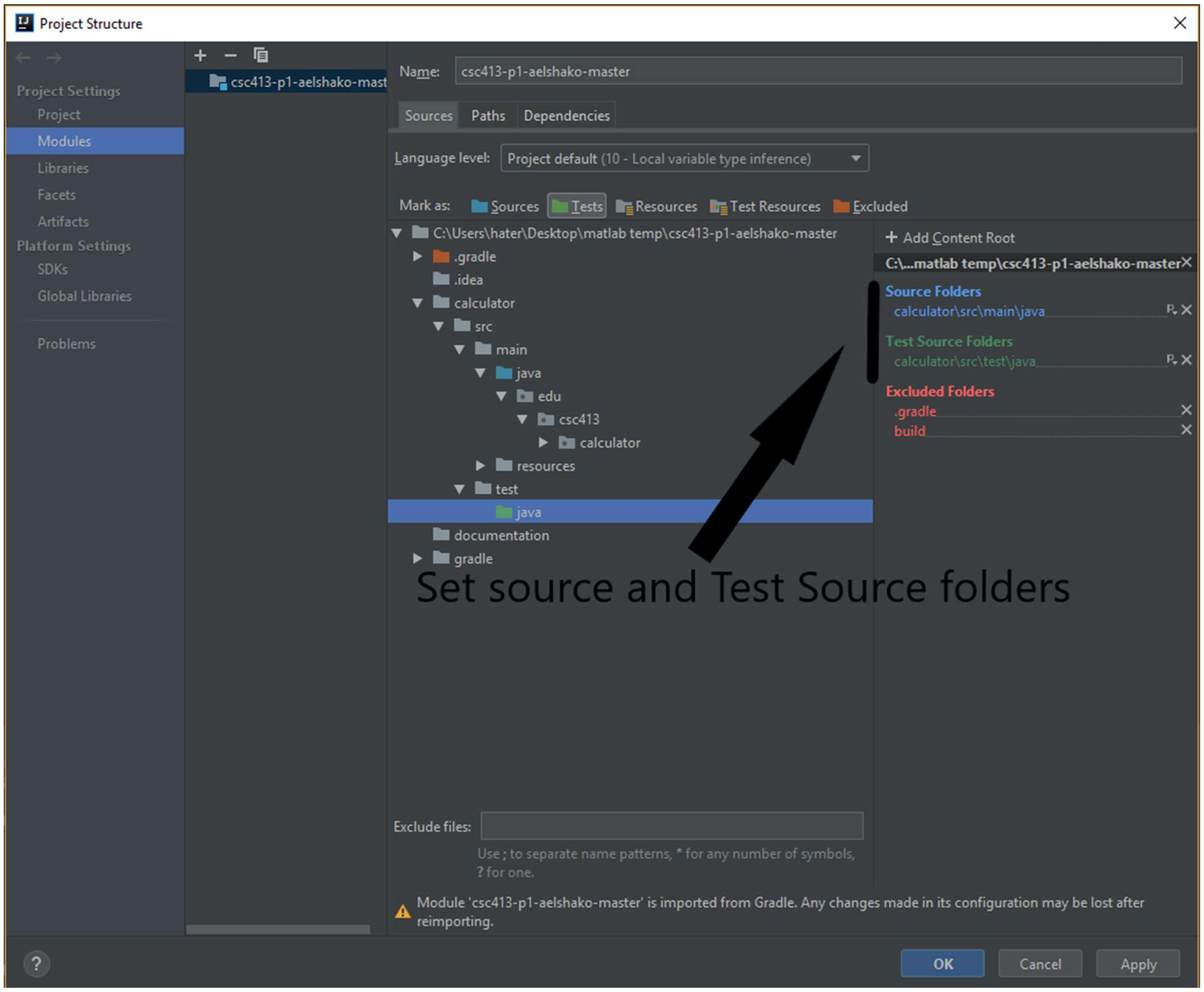
Use local gradle distribution

Gradle home:

Gradle JVM: 10 (java version "10.0.2", path: C:/Program Files/Java/jdk-10.0.2)

Project format: .idea (directory based)

Figure(1D): Verify that the 'Use gradle 'wrapper' task configuration' option is selected and hit 'Finish'.


Project Structure

csc413-p1-aelshako-mast

Name: csc413-p1-aelshako-master

Language level: Project default (10 - Local variable type inference)

Mark as: Sources Tests Resources Test Resources Excluded

```
▼ C:\Users\hater\Desktop\matlab temp\csc413-p1-aelshako-master
  ▶ .gradle
    .idea
  ▼ calculator
    ▼ src
      ▼ main
        ▼ java
          ▼ edu
            ▼ csc413
              ▶ calculator
        ▶ resources
      ▼ test
          java
    documentation
  ▶ gradle
```

+ Add Content Root

C:\...matlab temp\csc413-p1-aelshako-master

Source Folders
calculator\src\main\java

Test Source Folders
calculator\src\test\java

Excluded Folders
.gradle
build

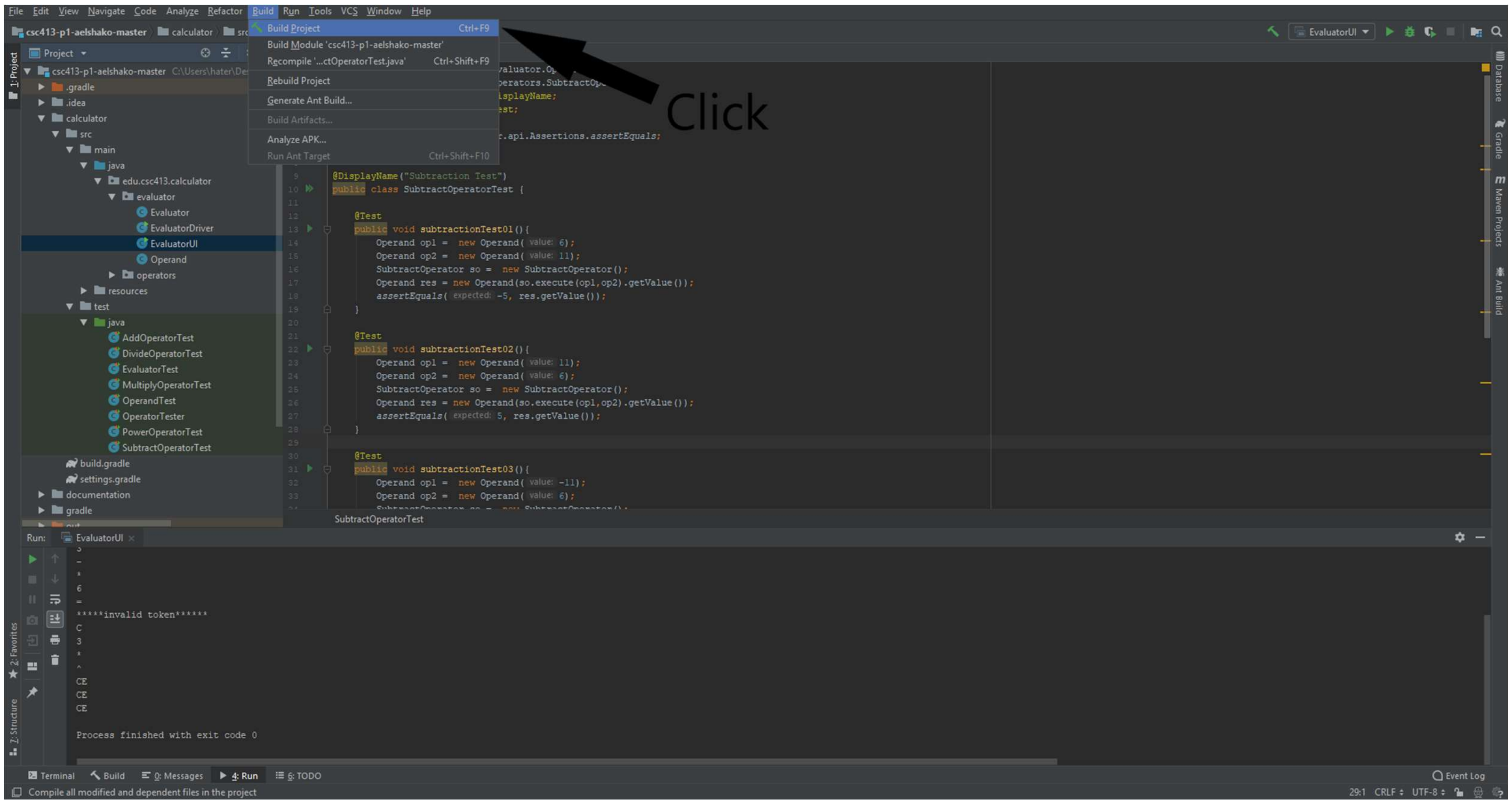
Set source and Test Source folders

Exclude files:

Use ; to separate name patterns, * for any number of symbols, ? for one.

Module 'csc413-p1-aelshako-master' is imported from Gradle. Any changes made in its configuration may be lost after reimporting.

Figure(1E): Set the Source and Test folders to be identical to what is shown. Press ‘Apply’, and then ‘OK’.



```
valuator.Op
perators.SubtractOp
isplayName;
est;

r.api.Assertions.assertEquals;

@DisplayName("Subtraction Test")
public class SubtractOperatorTest {

    @Test
    public void subtractionTest01(){
        Operand op1 =  new Operand( value: 6);
        Operand op2 =  new Operand( value: 11);
        SubtractOperator so =  new SubtractOperator();
        Operand res = new Operand(so.execute(op1,op2).getValue());
        assertEquals( expected: -5, res.getValue());
    }

    @Test
    public void subtractionTest02(){
        Operand op1 =  new Operand( value: 11);
        Operand op2 =  new Operand( value: 6);
        SubtractOperator so =  new SubtractOperator();
        Operand res = new Operand(so.execute(op1,op2).getValue());
        assertEquals( expected: 5, res.getValue());
    }

    @Test
    public void subtractionTest03(){
        Operand op1 =  new Operand( value: -11);
        Operand op2 =  new Operand( value: 6);
```

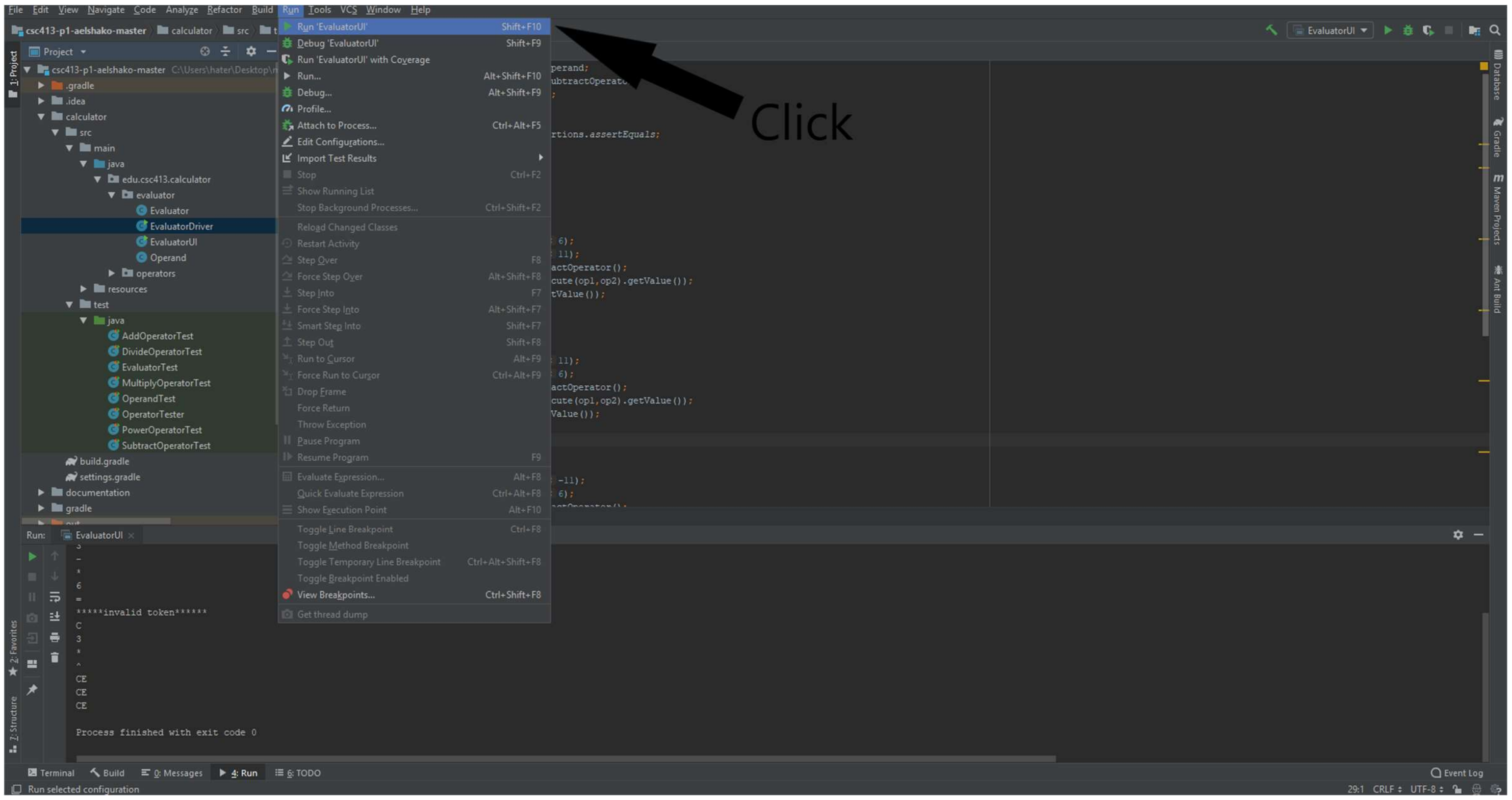
```
-
*
6
=
*****invalid token******
C
3
*
^
CE
CE
CE

Process finished with exit code 0
```

Compile all modified and dependent files in the project

Figure(F): Go to the upper left of the window. Press ‘Build’ and then ‘Build Project’.

4 How to Run your Project



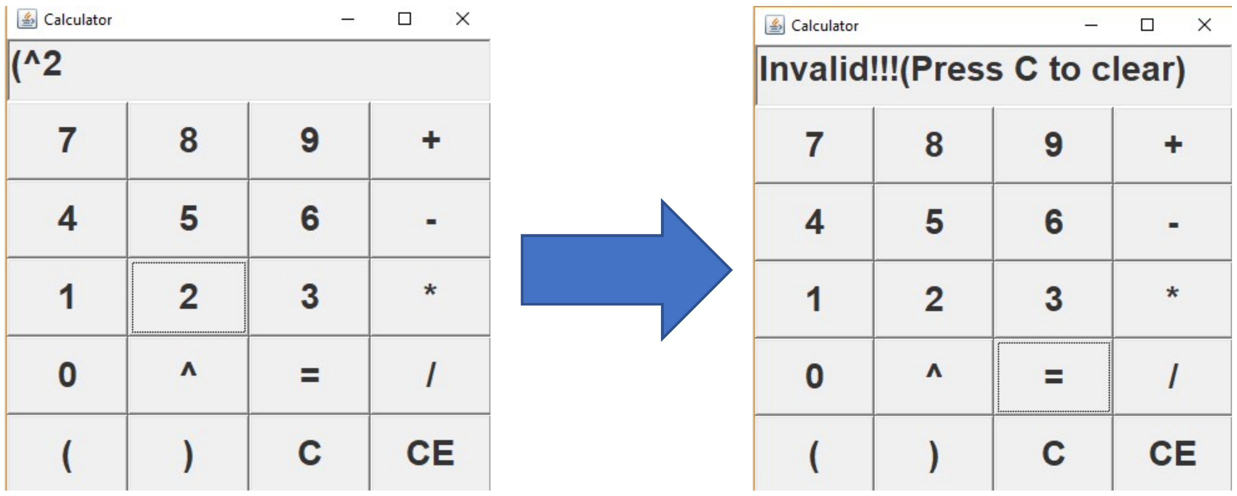
```
3
-
*
6
=
*****invalid token******
C
3
*
^
CE
CE
CE

Process finished with exit code 0
```

Figure(G): Go to the upper left of the window. Press ‘Run’ and then ‘Run ‘EvaluatorUI’’.

5 Assumption Made

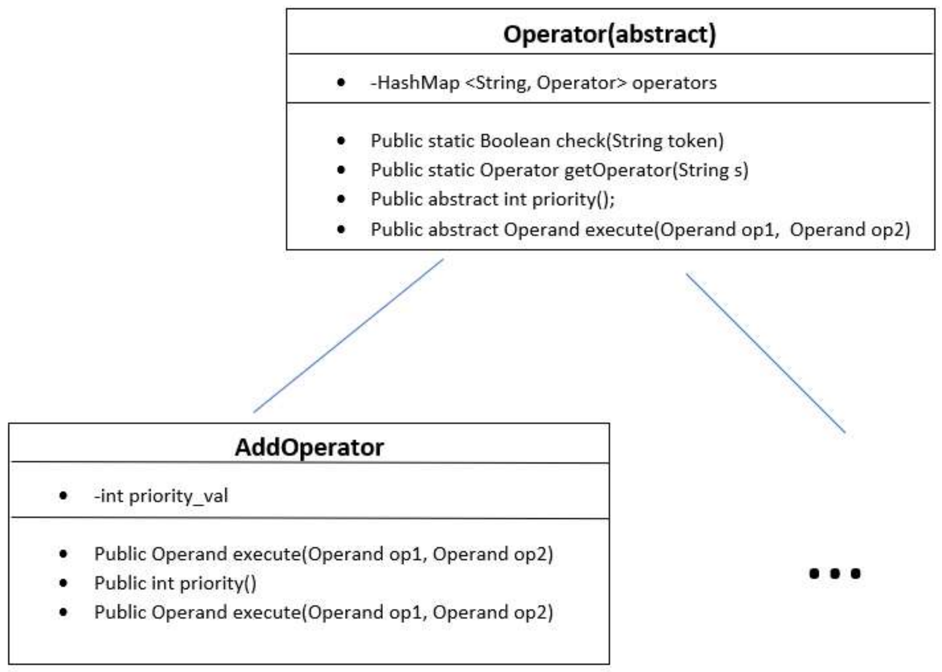
The only explicit assumption within the code is that all of the operands must be integers. I initially assumed that the users would be entering proper expressions, but I later revised my code to account for invalid expressions. In my EvaluatorUI, the calculator screen will notify the user that their expression is invalid as shown here:



6 Implementation Discussion

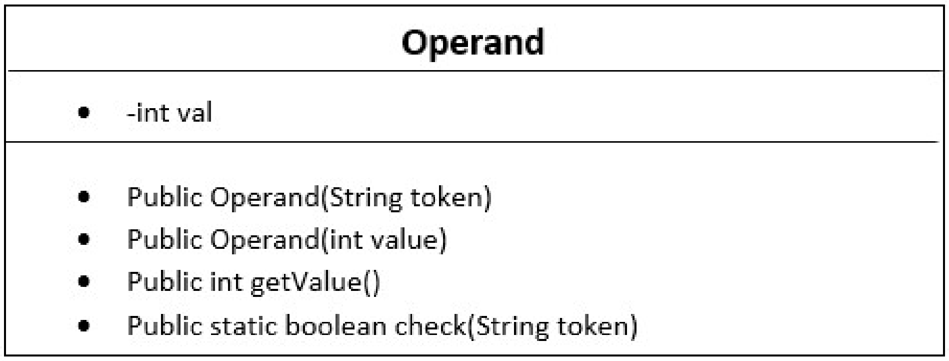
I feel that the implementation process and class hierarchy for this project is best explained visually. The following diagram shows the general structure for the Operator class and its various subclasses:

<u>Operator class:</u>


Operator(abstract)
-HashMap <String, Operator> operators
Public static Boolean check(String token)
Public static Operator getOperator(String s)
Public abstract int priority();
Public abstract Operand execute(Operand op1, Operand op2)
AddOperator
-int priority_val
Public Operand execute(Operand op1, Operand op2)
Public int priority()
Public Operand execute(Operand op1, Operand op2)
...

Figure(H): This figure shows the data fields(upper part of boxes) and the methods(lower part of boxes) for the Operator class and for the AddOperator class. The Operator class is an abstract class. It includes two static methods(check and getOperator). One of them checks if a String token corresponds to a valid operator, and the other returns an appropriate Operator object for that specific token by searching a Hash Map. The Operator class includes a private Hash Map where the keys are the String tokens of Operators and the values are actual instances of those operators. For example, one entry in the Hash Map is for the addition operation, so the key would be "+" and the actual value would be an object of the AddOperator type. The Operator class has two abstract methods. An abstract method is simply a method in an abstract class. Abstract methods only define what should be done, and not how it is done. Therefore, subclasses of the abstract Operator class provide implementation for the two abstract methods(priority and execute, respectively). It is very important to note that there is a total of 7 Operator subclasses: AddOperator, DivideOperator, MultiplyOperator, Power Operator, LeftParenthesisOperator, RightParenthesisOperator, and SubtractOperator. The above figure for the Operator class only shows the AddOperator subclass because all the other subclasses would have the same member names. Another important fact to note is that I've set each operator type(including parenthesis) to have a unique priority. This made it easier to write cases in checking if certain operators are present.

Operand class:



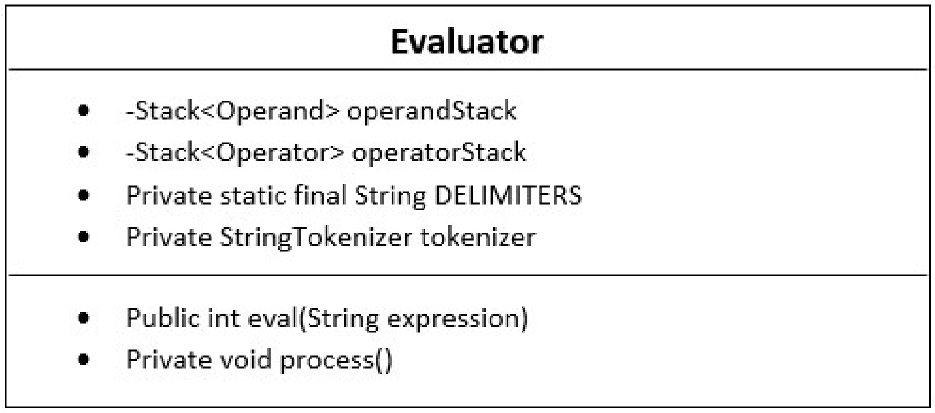
Operand

- -int val

- Public Operand(String token)
- Public Operand(int value)
- Public int getValue()
- Public static boolean check(String token)

Figure(I): This figure shows the structure of the Operand class. There is a private integer variable(val) which holds the actual integer value of the Operand. A public getter function(getValue) exists so we can get the value of a specific Operand object. The class includes two different constructors. One of the constructors takes in a String token representing an integer such as "3" while the other takes in an actual integer value such as 3. The last method in the class is the check method which is static, and simply returns a boolean to indicate if a String token corresponds to a valid operand.

Evaluator class:



Evaluator

- -Stack<Operand> operandStack
- -Stack<Operator> operatorStack
- Private static final String DELIMITERS
- Private StringTokenizer tokenizer

- Public int eval(String expression)
- Private void process()

Figure(I): This figure shows the structure of the Evaluator class. The Evaluator class is where the actual infix expression evaluation algorithm is implemented. In the next part of this report (Eval Explanation section), we can see an explanation that illustrates the flow of the code within the eval function. The Evaluator class includes two private Stack variables. One Stack is for Operands while the other is for Operators. The last two data fields of the class are a String of delimiters and a StringTokenizer which uses the delimiters to break up the expression. The Evaluator class includes two methods; one method(eval) is used to evaluate the actual expression while the other method(process) is a helper method. The process() method pops the Operand Stack twice, the Operator Stack once, and performs the appropriate operation on the two operands before pushing the result to the Operand Stack.

Eval Explanation: In this section of the report, I will explain my implementation of the eval() function as briefly and concisely as I can. Most of my explanation will be focused on explaining the underlying core algorithm as opposed to listing out each check and loop. The eval function runs for as long as more tokens exist. If the token represents an Operand, we make a corresponding Operand object and push it onto the operandStack. If the token is not representative of an operator, we throw a RunTimeException. At this point, we can deduce that the token represents a valid operator. Therefore, we create an operator object of that type. If the operator stack is empty or the operator is a left parenthesis: we add it right away to the operatorStack.

Otherwise, we have to do some additional checks. Our first check is to see if the operator is a right parenthesis. If so, we continuously process(explained previously) until we encounter the corresponding left parenthesis which is then removed from the operatorStack. The reason we do this is because we want to evaluate nested parenthesis correctly. Our last check is to repeatedly process if the new Operator has a lower priority than the operator at the top of the operatorStack. This ensures that we evaluate all higher priority operations before moving onto ones with lower priority. The last step in the algorithm is to continuously process until the operatorStack is empty. In other words, we process until there is nothing left to process.

7 Project Reflection

Working on this project was a very interesting experience for me. I hadn't worked with Java in over two years, but I feel that I am mostly up to speed after doing some reviewing. I initially struggled a little with the syntax but this project served as a very good tool in helping me get re-integrated with Java. The most time-consuming part of this project for me was learning how to properly use git and the IntelliJ Idea IDE. Writing this documentation took me a significant amount of time, but I feel that it is crucial in allowing me to re-visit why I made the design decisions that I did. Furthermore, I firmly believe that writing this documentation helped me build up multiple skills with regards to explaining software design and its principles. I look forward to having future assignments of this or a higher caliber to further push myself into understanding and describing better software development principles.

8 Project Conclusion/Results

I am proud to say that my project is fully functional. I did my best to make helper functions where possible to make my code more concise and readable. Additionally, I put a great deal of thought into the concept of encapsulation and how accessible my variables and functions should be. For example, I made my process() helper function to be private because there is no reason for it to be accessed outside of the class. My project successfully clears all of the junit tests, and I've implemented the GUI to include error checking for invalid expressions.