# *CSC 413 Project Documentation*

# *Fall 2018*


***Anmol Gondara***

***916110499***

***CSC 413.01, Fall 2018***


***https://github.com/csc413-01-fa18/csc413-p1-anmol2727***

# Table of Contents

# 1 Introduction

## 1.1 Project Overview

In this project, the goal was to create a calculator that evaluates mathematical expressions of the following operations: addition (+), subtraction (-), multiplication (*), division (/), powers (^), and parentheses (( )). To achieve this, it was important to do two things; one, evaluate the operations listed above, and the second, create a user-friendly interface (such as the one on your phone) that makes it easy for anyone to evaluate expressions. In the end, you have a beautifully designed calculator in which you are able to perform PEMDAS in a matter of seconds.

## 1.2 Technical Overview

Technically speaking, the goal of this project was to implement object-oriented design into two separate programs; ultimately creating a calculator that evaluates expressions using a user interface. The first program consists of an object that evaluates mathematical expressions dealing with PEMDAS (parentheses, exponents, multiplication, division, addition, and subtraction). More specifically, an algorithm was designed that allowed the program to calculate given expressions. The second program was a GUI (graphical user interface) that was based off program one, and allowed users to enter expressions and solve them in an application.

## 1.3 Summary of Work Completed

While working to complete the project, I took Professor Souza's advice and began with the Operand class and followed up with the creation of the operator subclasses of the superclass Operator. Next, I implemented the Operator class and then moved onto the algorithm design, which was to be executed in the Evaluator file. After passing all of the given tests, I worked on understanding and completing the EvaluatorUI file, which in the end resulted in a fully functional user interface.

It made the most sense to work in this order because all of the classes were dependent upon each other. In the Evaluator class, the algorithm design is based upon the Operand and Operator stacks. To successfully implement the algorithm, I constructed the operand from a token as well as an integer and retuned the value. In the Operator class, I declared a HashMap to store operators as values, to later be accessed in the Evaluator file. In both the Operand and Operator files, I checked to see if each was a valid token. Lastly, in

the operator subclasses, I retuned the priorities of all operators as well as the solutions.

## 2 Development Environment

a. Version of Java Used:

1.8.0_101

b. IDE Used:

IntelliJ IDEA 2018.2.3 (Ultimate Edition)

## 3 How to Build/Import your Project

Step 1:

First I accepted the invitation to create my repository by clicking on the link provided on iLearn.

Step 2:

I opened the Terminal application on my computer and typed 'cd ~/Desktop' to change my directory to my desktop. Then I typed 'git clone <link>' where <link> was my GitHub repository link with '.git' attached to the end. This step cloned my repository to my desktop.

Step 3:

I went on the class syllabus and clicked the link to download IntelliJ IDEA (Ultimate Edition). After receiving the license for Ultimate Edition, I downloaded and installed the IDE to my computer.

Step 4:

I installed Gradle, after learning that it was not already installed on my computer, by going on gradle.org/install/and following the steps.

Step 5:

To begin importing the project to my IDE, I opened IntelliJ and clicked 'Import Project' on the startup screen, followed by selecting the 'calculator' folder inside my repository, which I cloned to my desktop in step 2. Next, I clicked 'Import project from external model', and then chose 'Gradle'. After clicking 'Next', I selected 'Use gradle wrapper task configuration' and attached the gradle folder, which I installed in step 4 to the 'Gradle home' box on the screen. To finish the

process, I clicked 'Next' and then 'Finish', which successfully completed the importing procedure.

## 4 How to Run your Project

Running Tests:

Before running my project I ran the tests that were provided by Professor Souza. To run the tests, I clicked on the 'src' folder, which was on the panel on the left hand side of the IDE. Then I clicked 'test', 'java', double clicked on each test file to open them, and lastly clicked on each of the green play buttons in the green circles labeled as 'Run Test.' These green play buttons were located right next to the 'number of line' area on the left hand side.

Running the Project:

To run the project, I clicked on the 'src' folder, which was on the left panel on the left hand side of the IDE. Then I clicked 'main', 'edu.csc413.calculator', 'evaluator', and then double clicked the 'EvaluatorUI' to open the file. I then clicked the green play button located on line 27 and chose 'Run 'EvaluatorUI.main()'.' This step opened the calculator application and I was able perform actions on the interface, which meant I had successfully ran the project.

## 5 Assumption Made

Assumption 1:

When writing the PowerOperator subclass, I created a helper function that would allow the power operation to take place, using a while loop. My assumption was that I could declare the access level as Public, but when I showed Professor Souza my PowerOperator subclass, he told me to change it to Private because it is a helper to the Power class and only the power class. This means that it is only accessible in the class it is declared in.

Assumption 2:

While finishing up the Operator file and moving on to the Evaluator file, I assumed that we did not have to include Parenthesis operators in the HashMap, within the Operator file. I assumed this could be achieved in the Evaluator file itself. As I began designing, implementing, and testing my algorithm in the Evaluator file, I realized that to account for "(" and ")" I had to declare them as subclasses of the superclass Operator and declare them as apart of the HashMap in static otherwise I would get errors while running my program because not only did they not have a set priority, they were not returning anything.

# 6 Implementation Discussion

Operand Class:

I constructed an operand from a string token by declaring a variable value followed by setting the value of the parameter 'token' to the parseInt() function so that it could analyze a string and return an int.

I then constructed an operand from an integer by setting the value of the parameter 'value' to the same variable. Both of these constructions of operands were followed by the return of value, which could be accessed through the public method getValue(). This can be better seen in the class diagram in 6.1.

Lastly, I checked to see if the given token is a valid operand by applying a try-catch block to throw an exception when the program tries to convert a string to an int.

Operator Subclasses:

I created a total of seven operator subclasses: AddOperator(1), SubtractOperator(2), MultiplyOperator(3), DivideOperator(4), PowerOperator(5), OpenParenthesisOperator(6), and CloseParenthesisOperator(7). In all of these subclasses, I implemented the abstract methods – priority() and execute()—which were declared in the Operator class. In the priority methods, I returned the given priorities from the assignment PDF document. For the two parenthesis operators, I returned the lowest of priorities – 0 – as was stated to us in the Evaluator file on lines 30-32. In the execute methods, for operators 1-4, I created a variable which stored the solution and assigned it to a new instance of operand which in return performed the operation using the getValue() method declared in the Operand class (see above). For operator 5, I created a helper function to help me return the value of power in the execute() method. I achieved this by creating a simple do-while loop that would return the power outcome. For operators 6 and 7, I returned the value null within execute().

Operator Class:

I declared a HashMap to contain key and value as String and Operator, which I assigned with the name 'operators.' I then created a static initialization box for the 'private static HashMap' I just created. In this, I set a new instance of HashMap to allow me to access operators as values. Next, I stored operators so that I would be able to check if a token is a valid operator, just like in the directions given to us on lines 18 and 19.

Next I made sure to check if a given token is a valid operator by utilizing the containsKey function in a conditional statement, set to return true if the token is indeed valid and false otherwise.

Lastly, I created a public method to allow looking up Operators by token. This will be used in the Evaluator file.

Evaluator Class:

I added '(' and ')' as delimiters to the given attribute, to account for the parentheses.

I proceeded to fix line 76, which was given to us as 'Operator newOperator = new Operator();' because this piece of code gave an error and didn't allow me to run any tests. I changed it so that newOperator calls for the getOperator method, which is declared in the Operator file and allows Evaluator to look up Operators by token.

My next step was to add a conditional statement for when a token equals '(', in the first while loop. If the token does equal '(' then an Operator object is created and pushed to the operator stack as 'up to date Operator'. On the other hand, for when a token equals ')', I added a conditional statement that processes operators until the matching '(' is coincided. To implement this, I called for a helper method I created that runs a do-while loop. Inside the loop, the declared attribute operandStack is popped twice for op1 and op2. The order here matters because if we assume 1 + 2 and we push 1 and then 2, 2 comes to the top of the stack. When we pop the stack, we get 2 and then 1, which means we assign op2 to the first pop, and op1 to the second pop (achieved using the execute method). The result is then pushed onto the operandStack. Lastly, I popped the '(' operator.

I then continued to account for when the operatorStack is empty. If that were to be the case, I made sure the Operator object is created from the token using newOperator and then pushed to the operator Stack. If the operatorStack is not empty while the operator's precedence is greater than the precedence of the Operator at the top of the Stack, then the declared attribute operandStack is popped twice for op1 and op2. Using the execute method we assign op2 to the first pop, and op1 to the second pop. Finally pushing the result onto the operandStack.

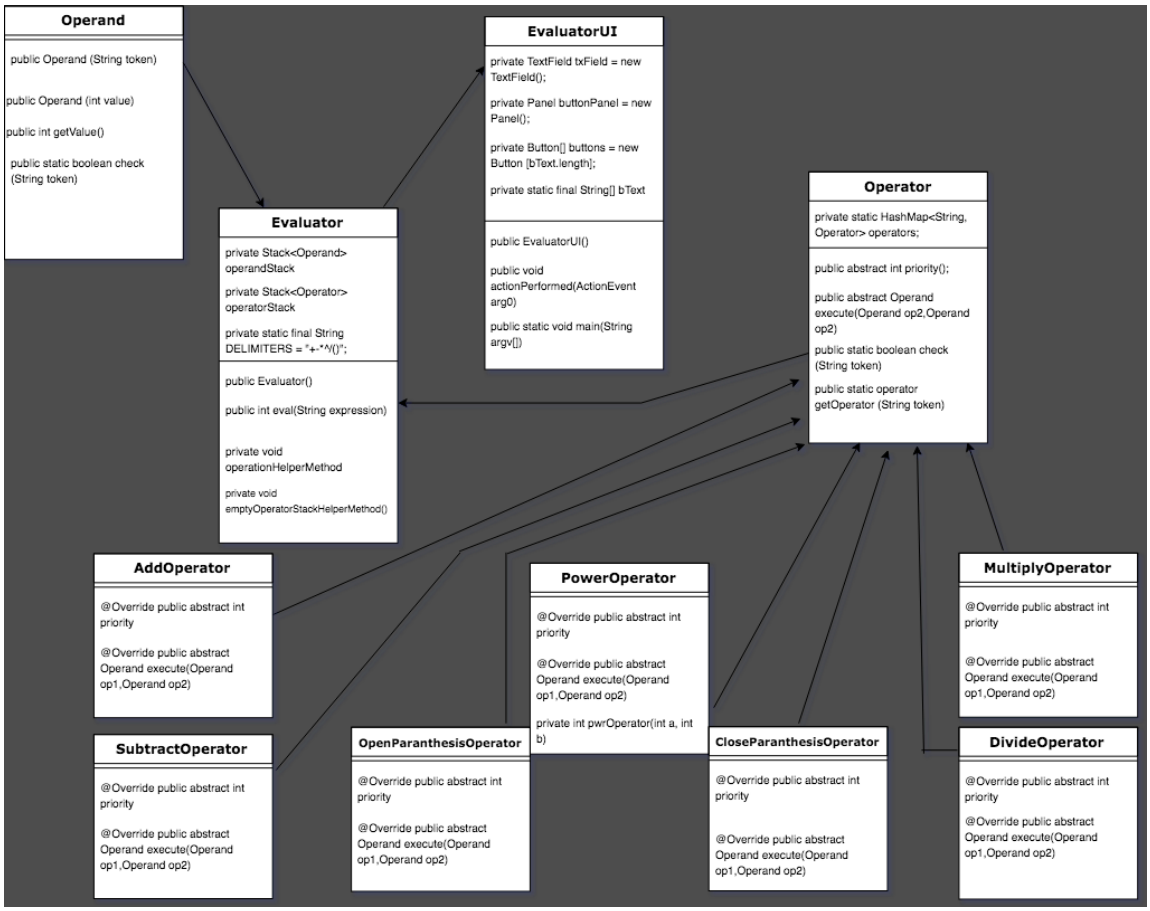
Lastly, once all of the tokens were scanned, I called for a helper method that runs a do-while loop within. In basic terms, this method performs the same operation that the method for ')' did but instead it runs while the operatorStack

is not empty. The loop processes Operators until the OperatorStack is empty. When the process is complete, a new instance of Operand is declared. I assigned it to pop the operandStack, which gets returned by getting the Value and ultimately completing the evaluation.

EvaluatorUI Class:

First I created a do-while loop that that returns a Boolean value. In essence, the loop acquires whenever a button is used/clicked. I played around with this a few times, until the UI would register the clicks of buttons. This process was a lot of trial and error. To make sure an 'invalid token' error did not pop up and crash the program, I implemented a conditional statement that makes sure = is not shown in the text field. To utilize the C button, I applied a conditional statement that clears the entire text field when C is used. To utilize the CE button, I applied a conditional statement that clears an expression when CE is used, meaning the very last entry. Lastly, I created a conditional statement that triggers the evaluation of an expression when = is used.

## 6.1 Class Diagram



## 7 Project Reflection

I learned a lot while working on this project, and look at the entire process as a great experience. Before this project I had not worked with Java for quite some time – just a little over a year now – as I have been concentrating on C++ and Python. As Professor Souza said when he assigned the project a couple of weeks ago, this was a great assignment to get reviewing done. Most of what I came across in this assignment I have already done in some capacity before – other than the GUI. I refreshed my memory of Java as I worked on this assignment by doing research on Google, using sites such as GeeksforGeeks and YouTube.

One thing that I would definitely like to point out is how much trouble I had importing the project. I have never worked with IntelliJ or Gradle before this assignment so I had to do a lot of research on how to get started. The hardest part about the whole procedure was installing Gradle onto my computer. I found a video on YouTube and followed along with the walkthrough and eventually got everything sorted out. I would say it took me almost three hours to get the project imported successfully, which tells me that I should manage my time better.

Overall, I enjoyed working on this assignment, but some advice I would give to myself is to start sooner. Though I finished this project with a good amount of time remaining, I still stressed myself out by not beginning to work on it right away. If Professor Souza had not extended the due date, I most likely would not have finished the assignment in full because I underestimated how much time the documentation takes.

## 8 Project Conclusion/Results

In the end, I successfully passed all of the test files after testing each file one by one. The tests that gave me the most trouble were in the EvaluatorTest. It turned out that my algorithm needed some more tweaking, but I eventually got it to output the correct solutions. Besides that set of tests, the EvaluatorUI results were giving me some problems because I kept on getting invalid token, which crashed my program. I had to take into account that we could not display the '=' sign in the text field. After writing a conditional statement for that case, I finally achieved the proper results.

To sum up this project, I would say that it was a much-needed experience. It gave me a great opportunity to review Java, and although there were days of great frustration, I ended up learning a lot, which will come in handy for the upcoming assignments.