CSC 413 Project Documentation

Fall 2018

RUSSELL WONG

916507662

413.01

https://github.com/csc413-01-fa18/csc413-p1-russ3llwong

Table of Contents

1 Introduction

1.1 Project Overview

This project simulates a simple calculator. In other words, it can evaluate mathematical expressions. When users run the project, they can enter expressions to be evaluated. In addition, there is a user interface included in the project as well. It has the appearance of a simple calculator, which is easy to use.

The program analyzes a mathematical expression by deconstructing the expression into individual numbers and mathematical operators. It scans through the expression and computes the correct result according to the order and priorities of the operators.

After evaluating the expressions, the program will display the accurate results. Users can then decide to evaluate more expressions or stop the program.

1.2 Technical Overview

As mentioned above, this project simulates a simple calculator which evaluates mathematical expressions. The project applies the algorithm of evaluation of infix expressions to evaluate the mathematical expressions.

In order to do so, it scans through strings of mathematical expressions using a tokenizer. The tokenizer will break down a mathematical expression into individual operands and operators. New operand and operator objects will be instantiated, depending on what tokens are scanned, and then they will be pushed into stacks. The expression will be evaluated as the tokenizer scans across the expression. After the tokenizer has done its job, the program will process the remaining operands & operators to determine the correct result.

Besides that, there is also a Graphical User Interface (GUI) in the project as well. When users run the GUI, they can interact with a simple calculator to evaluate mathematical expressions. The GUI will receive input from the users as the buttons get pushed, hence updating the text field whenever a button is pushed. When users push the "=" button, an instance of the *Evaluator* class will be created to evaluate the expression. The final result will then be displayed in the text field of the GUI calculator.

1.3 Summary of Work Completed

I started by programming the utility classes the *Evaluator* class uses. Since the *Operand* class is simple and is not dependent on other classes, I worked on it first before moving onto the *Operator* class. I completed the constructors, the *getValue()* method and the *check()* method which determines if a token is a valid operand.

In the Operator class, I declared a *HashMap* with the mathematical operators stored as keys and all of the *Operator* subclasses stored as values. After that, I completed its *check()* method as well. Then, I created 6 subclasses for the *Operator* superclass, for the operators "+", "-", "*", "/", "^", and "(". I proceeded to implement their *priority()* and *execute()* methods, which returns the precedence of an operator and performs a mathematical calculation respectively.

Next, I implemented the algorithm of evaluation of infix expressions within the *Evaluator* class, which was already partially complete. I added "(" operator to the delimiters string and handled every possible condition for the token scanned in order for the algorithm to work.

Lastly, I completed the "*actionPerformed*" method in the *EvaluatorUI* class by reusing the algorithm implemented within the *Evaluator* class.

2 Development Environment

Version of Java Used: Java 10.0.2

IDE Used: IntelliJ IDEA 2018.2.3. (Ultimate Edition)

3 How to Build/Import your Project

1. Ensure that you have an IDE and an updated version of Java, preferably version 8 or above.
2. Install Gradle if you do not have it in your computer.
3. In your IDE, on the top menu tab, go to “File”, then “New” and then “Project from Existing Sources…”
4. Select the “calculator” folder as the root source.
5. After that, select “Import project from external model”.
6. Under the list of external models, select “Gradle”.
7. Next, ensure that the “Gradle home” directory points to where your Gradle files are in your computer. (e.g. C:/User)
8. The “Gradle JVM” directory should point to your newest JDK’s folder.
9. Lastly, proceed with the instructions prompted and the project will be built.

4 How to Run your Project

After having the project built or imported, you have two options for running the project. You can either run the *EvaluatorDriver* class or the *EvaluatorUI* class.

EvaluatorDriver:

1. Regardless of what IDE is used, just click “Run” for the EvaluatorDriver file.
2. In the output console, you will prompted to enter an expression.
3. Enter a valid expression and click “Enter” for the program to evaluate the expression.
4. The program will print the expression that was entered and the result of it.
5. You may now either enter another expression to be evaluated or exit/end the program by clicking “Stop”.

EvaluatorUI:

1. Regardless of what IDE is used, just click “Run” for the EvaluatorUI file.
2. After the program compiles successfully, a GUI (calculator) will appear in a new window.

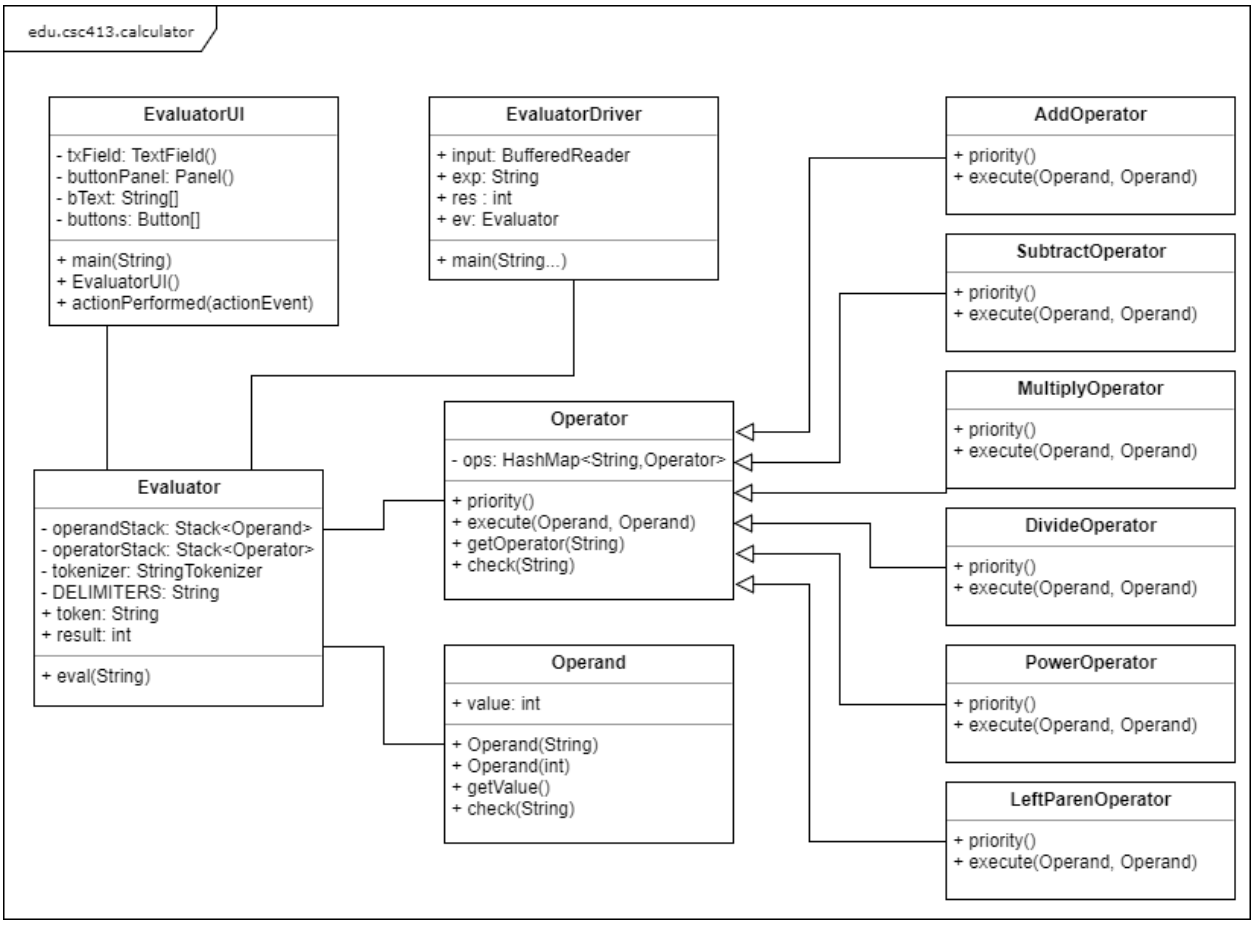
3. Enter a valid expression and click the “=” button for the calculator to evaluate the expression.
4. The calculator will clear the expression that was entered and return the result of the expression.
5. You may now either enter another expression to be evaluated or exit/end the program by closing the GUI or clicking “Stop” in the program.

5 Assumption Made

1. In the output console, the user input will not contain any invalid expressions. For example, “3+*1))-2”
2. In the output console, the user input will not contain any operands other than numerical operands, meaning alphabets will not be included in expressions.
3. In the output console, the only operators that will be used are “+”, “-”, “*”, “/”, “^”, “(“, and “)”
4. In the GUI, the user input will not contain any invalid expressions.
5. In the GUI, the user input will not contain any negative numbers.
6. The remainder when a number is divided is ignored. For example, evaluating 10/3 will result in a 3.

6 Implementation Discussion

6.1 Class Diagram



The UML diagram above shows the relationships between the classes in this project.

The *EvaluatorUI* and *EvaluatorDriver* are the classes to run if users want to evaluate mathematical expressions. The *EvaluatorDriver* class interacts with users via the output console whereas the *EvaluatorUI* class interacts with users via a GUI. They both depend on the *Evaluator* class as they need the *Evaluator* class to evaluate expressions.

The *Evaluator* class has two utility classes——*Operand* class and *Operator* class. The *Operand* class does not depend on any other classes. On the other hand, the *Operator* class is an abstract superclass. The *priority()* and *execute()* methods are abstract methods. It has 6 subclasses, one for each mathematical operator, which will implement the abstract methods. By using inheritance, the *Operator* class hierarchy is easier to understand and manage.

The algorithm in the *eval* function in the *Evaluator* class uses the infix expression evaluation algorithm. The algorithm uses two stacks, one for the operands and another for the operators, to process the strings of expressions. The function first uses a tokenizer to tokenize a string of expression. Every token is then determined if it is an operand, an operator or an invalid token. The algorithm is explained in the following:

- If an operand token is scanned, an Operand object is created from the token, and pushed to the operand Stack
- If an operator token is scanned, and the operator Stack is empty, then an Operator object is created from the token, and pushed to the operator Stack
- If an operator token is scanned, and the operator Stack is not empty, and the operator's precedence is greater than the precedence of the Operator at the top of the Stack, then and Operator object is created from the token, and pushed to the operator Stack
- If the token is "(", and Operator object is created from the token, and pushed to the operator Stack
- If the token is ")", the process Operators until the corresponding "(" is encountered. Pop the "(" Operator.
- If none of the above cases apply, process an Operator.

Processing an Operator means to:

- Pop the operand Stack twice
- Pop the operator Stack
- Execute the Operator with the two Operands
- Push the result onto the operand Stack

After the whole expression is tokenized, the program will then process the remaining tokens left in the operand stack and operator stack. The actual computation is actually done in the *execute()* methods in the Operator subclasses. The *priority()* methods are used to determine the precedence of the operators, as this is important when determining when and which operators to process. The special case

In the end, the program will return the final result to the caller function, which could either be the *EvaluatorUI* or *EvaluatorDriver*.

7 Project Reflection

As this is the first project of this class, I initially did not have much idea on how and where to start. Admittedly, I was a little lost and overwhelmed at first. But after I started doing it, things were as bad as I thought.

Nonetheless, importing the project was a hassle, mainly because of "gradle". After successfully importing the project, I took some time to familiarize myself with the code written, as well as the instructions in the PDF file given. There were a few things that I had to do some research on, like the objects declared in the UI file. Besides that, I had to refresh myself on topics like inheritance & polymorphism.

The Operand & Operator classes did not take me too long to complete, except for figuring out the public method for the *hashMap*. I even declared a public method for users to obtain the entire *hashMap*, but then I realized that would allow people to modify the contents of the *hashMap*. So I went to read the assignment guide and realized the public method was only for outside classes to get an instance of an operator class via the key, which is a string token in this case. It probably would have taken longer if our instructor did not help us with the declaration of the *hashMap*. I then had to refresh myself on the declaration of subclasses in order to create the individual operator subclasses to inherit the methods from their parent class.

For this project, I spent most of my time on the Evaluator, I had to read the instructions repeatedly and study algorithm carefully to handle all the conditions. I struggled to understand the algorithm for a bit, but I was able to implement it after some trials & error. I initially handled the tests that did not involve the parentheses operators. After that, I proceeded to work on the parentheses operators. Not long after, I got to the point where I was able to pass 8 out of the 9 tests in the Evaluator Test. The last one, however, bothered me for a long time. After spending hours analyzing my algorithm and debugging, I realized it was due to me breaking the while loop when processing an operator. Next, I spent some time trying to understand the GUI as this is probably my second time doing a GUI. But in the end, I realized it was not as complicated as I expected it to be, at least not in this particular case. Though, I did have to read about the various objects used in the class as they were new to me.

This project really did the job as a refresher for me on Java. Moreover, it also gave me a brief idea of what to expect for the rest of the semester. It was a little challenging, but it was still manageable. One thing I definitely need to do for the following assignments is starting earlier because the documentation took longer than I thought.

8 Project Conclusion/Results

Through this project, I have learned a lot of things. Learned: Importing, evaluating, GUI and following instructions.

Last but not least, this project helped me to realize the significance of utilizing Debug Mode. I probably would not have been able to complete the project without it. By testing my algorithm in debug mode, I was able to identify the root causes of the errors I have encountered. Also, it saved me a lot of time in troubleshooting.

Thankfully, the algorithm works and the program compiles without error. The program passed all of the unit tests provided. Also, I am glad that the GUI Calculator is working as well. In short, the project is a success.