CSC 413 Calculator Documentation

Spring 2021

Ernesto Ruiz

917385591

CSC413-02

https://github.com/csc413-sp21/csc413-p1-unique-Creations

Table of Contents

1 Introduction

1.1 Project Overview

This project creates a calculator program that utilizes a graphical user interface for simplicity. When the program is started, the interface will appear in the screen as a form of a calculator. By pressing the buttons on the screen, the interface will not only display the buttons you have pressed but also the result of the expression once the “=” button is pressed. The button “C” clears the entire expression while the button “CE” clears the expression up to the last operator.

1.2 Technical Overview

The Calculator program is composed of an Operator Abstract class, Evaluator class, Operand class, and EvaluatorUI class. The Operator class initializes a private HashMap operators desired as keys and instances of operators as values. The operator subclasses are then implemented to extend the Operator abstract classes. The Operand class takes strings, checks if the string is an integer, then can be used to get the value of the integer throughout the Evaluator class, which is used to evaluate the complete expression entered by the user. The expression is chosen using the EvaluatorUI class which is implemented to display an interactive calculator to the user, enter user’s expression, and display the result of the expression.

1.3 Summary of Work Completed

To fully implement a functional Calculator project, code was added to the functions in the Operand, Operator, Evaluator and EvaluatorUI classes. Although some of the evaluateExpression was already implemented in the Evaluator class, some logic was added to properly process the user’s expression and return the correct result. The Operand class’ constructors and functions are now functional as well as the actionPerformed function in the EvaluatorUI. Subclasses were created for the instances in the HashMap located in the Operator class. These subclasses now fully implement the abstract functions in the Operator class. The public static functions are also now fully implemented in the class. Some logic modifications are still required in the Evaluator class to process more complicated expressions using parentheses.

2 Development Environment

Java Version: openjdk-15.0.2

IDE: IntelliJ IDEA 2020.3.2

3 How to Build/Import your Project

To import the project into the IntelliJ IDE:

Start the IntelliJ IDE > Open > csc413-p1-unique-Creations /Calculator

The Calculator folder will be the root directory. This allows for the project to be properly built in the IDE. Using other folders may result in issue running the code.

4 How to Run your Project

There are multiple ways of running the project. The configuration should be set to the desired file with a main function, such as the EvaluatorDriver file or the EvaluatorUI class and the working directory should be set to the same path used to Import the project into the IDE. Once this is complete, the file's main should be set as the class path, making the project runnable.

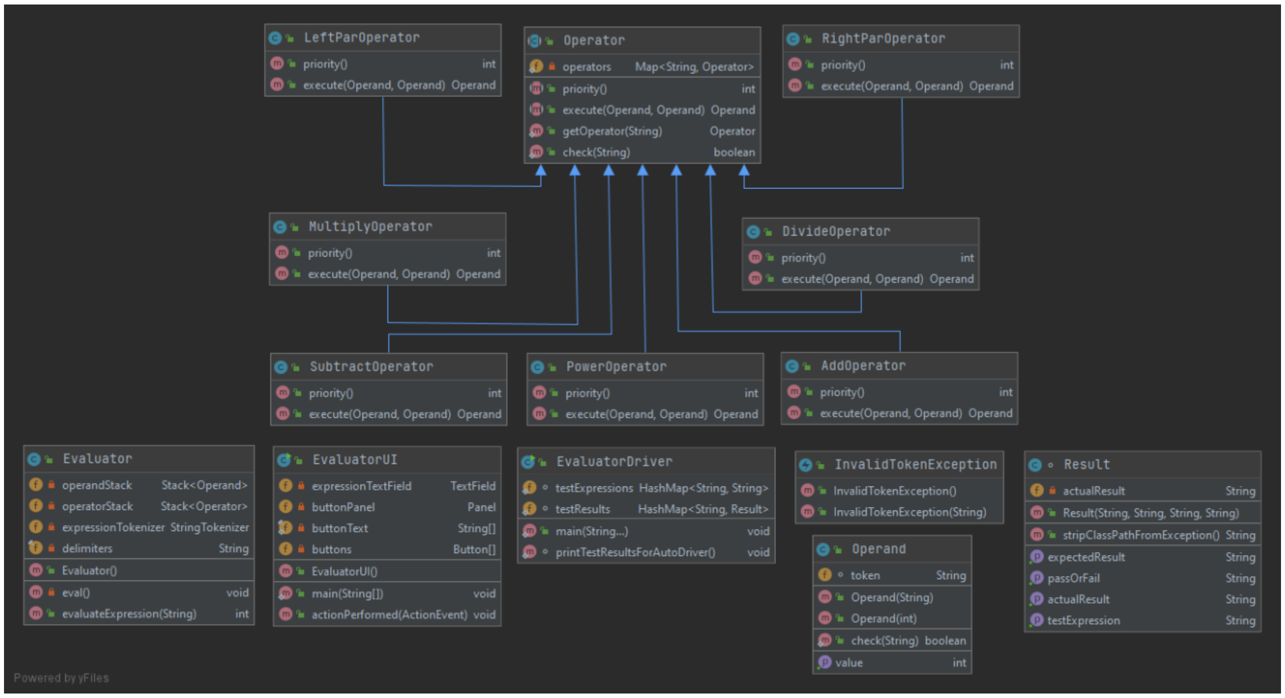
5 Assumption Made

A few modifications were made when implementing the classes. In the powerOperator subclass, I imported java.lang.Math to allow me to call the pow() function. This was done in order to keep my code short and easy to read, assuming we are able to import classes into the subclasses. When working with the Evaluator class, I assumed we were able to create our own function. Therefore, I added the eval() function to allow me to evaluate the expression throughout the code when necessary. The code inside the function was implemented using similar code to the block already provided in the evaluator class.

6 Implementation Discussion

When implementing the project, I attempted to keep the code as short as possible. This was the goal in order to make the code not only readable but also functional since larger blocks of code may sometimes result in unwanted outcomes. To make the evaluator easier to implement, I added the leftParOperator and rightParOperator to the HashMap located in the Operator class. This allows me to call the priority() function on the initialized operator in the Evaluator class. A few if statements were created in the Evaluator class to process the expression and a private eval() function was implemented to evaluate the operator with the both operands in the stack. A while method was added to the bottom of the evaluateExpression() function to evaluate the expression if all if statements are ignored. When dealing with the parentheses, 2 additional if statements were needed to process the information efficiently in that scenario. Further improvements are required to shorten the code and make the implementation more readable. The Operator and EvaluatorUI classes were both kept short with a bit more development in the UI class, since the calculator being displayed needs to allow the user to choose what is being displayed and what expression will be evaluated. Once the user clicks the "=" button, the evaluateExpression() function is called with the expression in the parameter, therefore, returning the result once complete.

6.1 Class Diagram


LeftParOperator
priority() int
execute(Operand, Operand) Operand
Operator
operators Map<String, Operator>
priority() int
execute(Operand, Operand) Operand
getOperator(String) Operator
check(String) boolean
RightParOperator
priority() int
execute(Operand, Operand) Operand
MultiplyOperator
priority() int
execute(Operand, Operand) Operand
DivideOperator
priority() int
execute(Operand, Operand) Operand
SubtractOperator
priority() int
execute(Operand, Operand) Operand
PowerOperator
priority() int
execute(Operand, Operand) Operand
AddOperator
priority() int
execute(Operand, Operand) Operand
Evaluator
operandStack Stack<Operand>
operatorStack Stack<Operator>
expressionTokenizer StringTokenizer
delimiters String
Evaluator()
eval() void
evaluateExpression(String) int
EvaluatorUI
expressionTextField TextField
buttonPanel Panel
buttonText String[]
buttons Button[]
EvaluatorUI()
main(String[]) void
actionPerformed(ActionEvent) void
EvaluatorDriver
testExpressions HashMap<String, String>
testResults HashMap<String, Result>
main(String...) void
printTestResultsForAutoDriver() void
InvalidTokenException
InvalidTokenException()
InvalidTokenException(String)
Operand
token String
Operand(String)
Operand(int)
check(String) boolean
value int
Result
actualResult String
Result(String, String, String, String)
stripClassPathFromException() String
expectedResult String
passOrFail String
actualResult String
testExpression String
Powered by yFiles

7 Project Reflection

Implementing this project was somewhat challenging. Not only did I have to remember Java, but also data structure concepts. The debug tool allowed me to see exactly how the code was running and gave me the ability to fix the code more efficiently. It was a good way of getting use to writing Java again. Most of the trouble I had was in implementing the Evaluator class. Writing the code to have the evaluator process the operators took some time to get correct. I created an Operator subclass for both parentheses to make it easier to code. Although I did implement the evaluator to evaluate the expression efficiently, I feel as if revisions are still necessary to make the code easier to read. There are always ways of not using a lot of if statements. Writing the rest of the code served as review and the EvaluatorUI was fun to code since I have not had experience with implementing a GUI with anything other than Swift. While there were some similarities in the way to implement the actions, there were also new syntax I had to learn with Java. Implementing this project was educational and helped me review on what I learned a couple of years ago.

8 Project Conclusion/Results

Writing code requires patience and practice. This assignment served as a good way to review and learn new things. Although implementing efficient code can be sometimes challenging, observing the code you wrote work the way one wants it to bring satisfaction. I hope to one day be able to build something that will not only bring joy to others but also prove necessary in our community. Being given assignments such as this one brings me closer to that goal. It gave me the opportunity to make mistakes and learn how to fix them. This assignment also gave me the opportunity to learn how to use GitHub. Although I have used it before, I never knew how to use it as efficiently as I do now. GitHub gave me the ability to save my work somewhere other than my current computer or cloud, it also gives me the ability to be ability to work with a team more efficiently whether it will be for work or for side projects. Assignments such as this one are necessary in order to expand our capabilities in writing efficient and readable code.