# *CSC 413 Project Documentation*

# *Fall 2019*


***Maria Seljak***

***915736307***

***Section 01***


***https://github.com/csc413-01-fall2019/csc413-p1-lseljak92***

# Table of Contents

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

# 1 Introduction

## 1.1 Project Overview

The purpose of this project consisted on creating an object that evaluates mathematical expression as well as a GUI calculator that applies the functionality of the evaluator object. The approach was using object-oriented design.

## 1.2 Technical Overview

Before starting to develop this assignment, I first mapped out the logic of the evaluator class based on the "Evaluation of Infix expressions" article provided on the assignment's instructions document. This let me organize my thoughts and ideas before starting to implement them.

Encapsulation was crucial when defining the access modifiers for the classes and objects. The main goal was to make sure that none of the data fields from the different classes cannot be directly accessed.

The two main classes to implement this project are the Operand and Operator classes. The Operand class includes two constructors: one takes a String as a parameter and converts it to an integer value when instantiating the object, while the other one takes an integer as the parameter and uses that value to instantiate the object. The methods from this class consist on a getter to provide access to the integer value of the object and a Boolean function that takes a String as a parameter and tests if the input is valid. To do so I decided to use the try-catch block to handle exceptions.

The Operator class is an abstract class. It includes a private static HashMap that contains all operators used throughout the program. Each one of these operators are unique objects that extend the Operator class. Therefore, each operator implements this class' methods, which are the priority function, that returns the object's priority integer and the execute function, which takes two Operands as parameters and return the calculates the result from the specific operation. The operators are the following: AddOperator, DivideOperator, MultiplyOperator, PowerOperator, SubtractOperator. There were also included two additional operator classes that represent the left and right operators. In order to follow the program's logic, these operators priorities are less than the other ones since they're not used to return any calculation. In order to validate the operators, the Operator class includes a Boolean function. The HashMap data structure make it possible to search for a specific key in O(1) time, which makes the running time of the program more efficient. Even if there were more operators to be added to the HashMap this constant time would not change.

Finally, in the Evaluator file the implementation of the expression is developed. It consists of two stacks: the operatorStack and the operandStack. Before adding any item to the stacks they're first tested to reassure they're valid inputs. After its validation, following the algorithm cited at the beginning of this documentation, I decided to test each possible scenario of calculations using an if-conditional. I also decided to develop a method that calculates the expressions depending on the token, since it's frequently used throughout the program. This makes it possible to avoid repeated code and make the program more readable. Using that method, I also decided to implement a method to clear the stack.

### 1.3 Summary of Work Completed

In the Evaluator class I was able to implement the algorithm that evaluates infix expressions. All the tests passed. However, I wanted to add the functionality of operations with negative numbers, but I was not able to achieve this goal. The Operand and Operator classes have been completed. The GUI calculator runs. In the implementation of the action listener I used an if-statement that links all buttons to a specific text field. I believe I could've approached it in a different way that would make code more maintainable and easier to read.

## 2 Development Environment

This assignment was developed in the InteliJ IDE using the JAVA version 8 update 221.

## 3 How to Build/Import your Project

In order to build/import the project, first clone or download the repository to your local machine. For guidance on cloning, this link provides a thorough explanation of the steps. When opening InteliJ or any other preferred IDE select "Import project" and choose the "calculator" folder as the root of the project. After completing the import process, search on the top bar of the IDE the "Build" icon and press it. This makes it possible to inspect the code as well as to provide and check the dependencies.

## 4 How to Run your Project

In order to execute the program search for the "Run" icon on the top of the IDE. If using InteliJ, on the left side of this icon there's a dropdown menu where any file can be chosen to run. In order to execute the GUI calculator, select the "EvaluatorUI" file. This will open a new window with the calculator interface. Using this the expression evaluator can be tested.

It is also an option to run the evaluator object without opening the GUI. To do so, select the "EvaluatorTest" file from the "Test" folder. This will run over pre-set tests that check the functionality of the object.

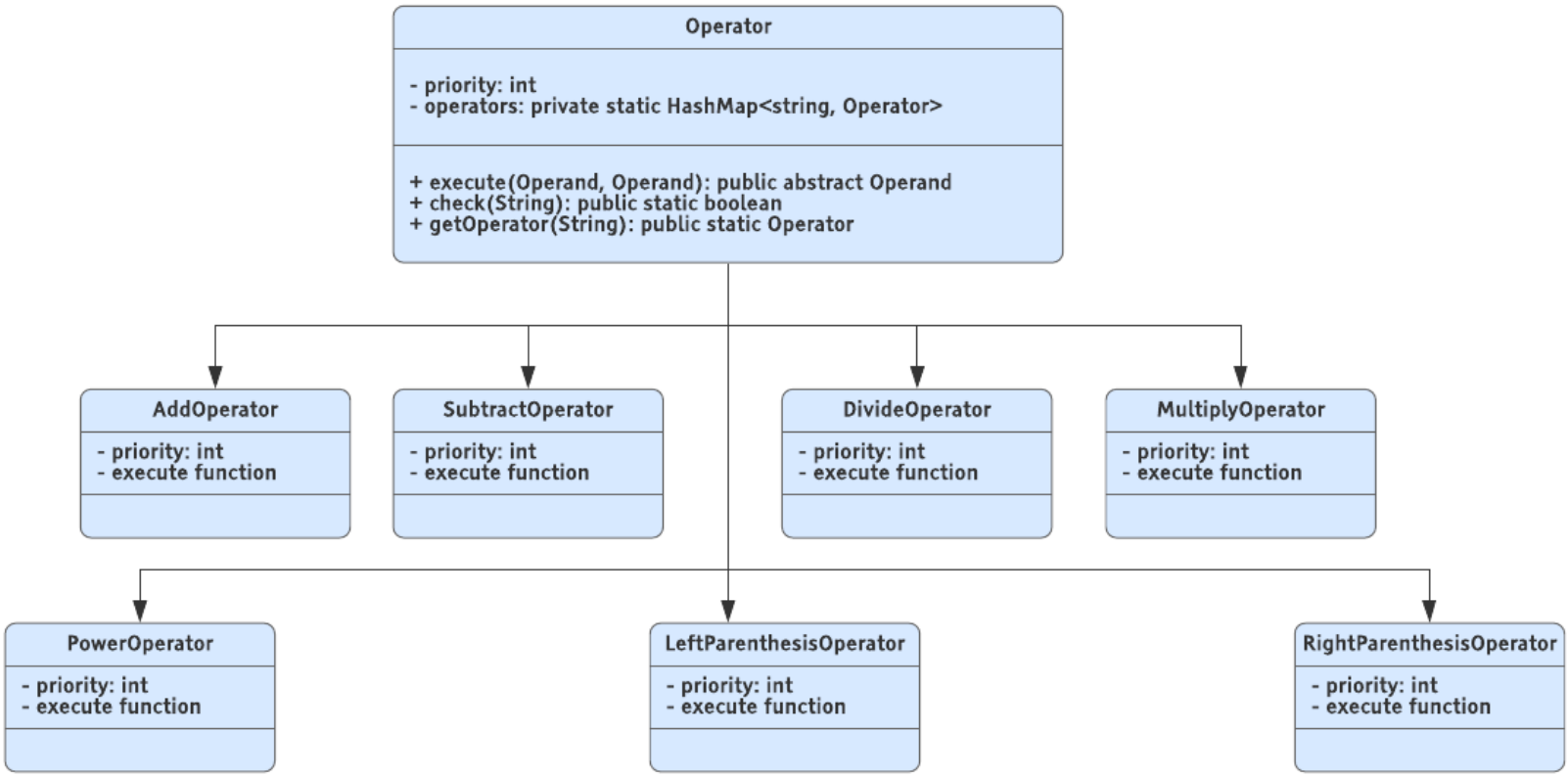
## 5 Assumption Made

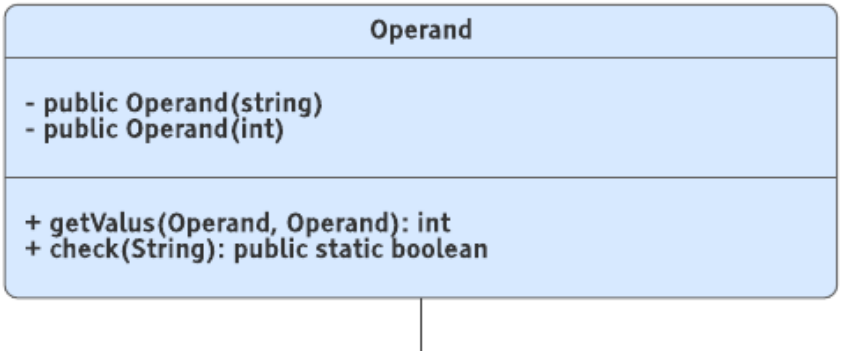
Throughout the project I had a hard time evaluating expressions that had parenthesis. At the beginning I assumed that their priority should be greater than the ones from the operators, so I gave them 4 and 5 to the left and right parenthesis, respectively. Consequently, whenever I tried running the program, if the expressions contain parenthesis, the program constantly crashed with the EmptyStackException. I never thought the problem came from the priority integer value, so I made numerous changes to the Evaluator conditional that tests the expressions. Finally, after debugging several times, I went over the algorithm again, trying to understand what the role of the parenthesis was. That's when I realized that I was using wrong values. I changed them to -1 (left) and 0 (right). After these changes, all tests finally passed. This experience made me aware of the importance of understanding the logic before implementing any code.

# 6 Implementation Discussion

My goal was to implement a Structural Design Pattern, more specifically, the Composite Pattern. This pattern relates to the part-whole hierarchy used in this project. For example, each operator is structured in a different way, but they're actually implemented in the same way.

## 6.1 Class Diagram



# 7 Project Reflection

In my personal experience, this first assignment was a great way to start implementing good design patterns when coding as well as efficient thinking and planning before any coding is started. I believe many implementations could be improved. However, the functionality of the programs passes all the tests that were set at the beginning of the assignment.

# 8 Project Conclusion/Results

Even though I faced challenges throughout the project, I was able to develop a functional program that fulfils the assignment's requirements. I focused on practicing and using object-oriented programming. I also applied appropriate access modifiers for each one of the objects and classes.

I learned the importance of planning before programming in order to have a clearer and larger picture of the overall program. This also made me realize of future improvements that this program can have. For example, the developed application currently doesn't recognize negative numbers, which could be an additional test case that would allow the application to be fully functional.