CSC 413 Project Documentation

Spring 2019

Raya Farshad

917940106

CSC413-03

https://github.com/csc415-03-spring2019/csc413-p1-rayafsh.git

Table of Contents

[1 Introduction 3](#_Toc522827688)

[1.1 Project Overview 3](#_Toc522827689)

[1.2 Technical Overview 3](#_Toc522827690)

[1.3 Summary of Work Completed 3](#_Toc522827691)

[2 Development Environment 3](#_Toc522827692)

[3 How to Build/Import your Project 3](#_Toc522827693)

[4 How to Run your Project 3](#_Toc522827694)

[5 Assumption Made 3](#_Toc522827695)

[6 Implementation Discussion 3](#_Toc522827696)

[6.1 Class Diagram 3](#_Toc522827697)

[7 Project Reflection 3](#_Toc522827698)

[8 Project Conclusion/Results 3](#_Toc522827699)

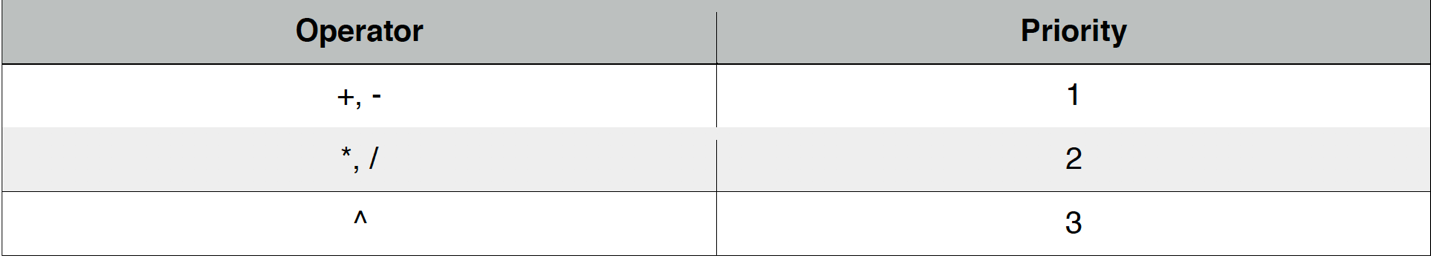
# Introduction

In this project we asked to create a calculator which calculates addition, subtraction, multiplication, division, power or mix of all these operations along with the Parenthesis which define the priority of those arithmetic operations.

## Project Overview

In the current project we are assuming that the operators ( +, -, \*, ^, / ) are written between the operands (numbers) which calls as infix expression.

Also, this approach is more confusing to the computers to assign the higher priorities to which operators. In order to make that clear, in our program we define given priority for each of operators, and we assign priority to the parenthesis.



The overall overview of what I have done in this project is:

We have two stacks:

* Operator stack to hold the operators (+, - , \*, / ,(), ^)
* Operand stack to hold the operands (numbers)

1. Step1, Initialize the first token you want to push at the end on the Operator stack I choose “A” to push it into the stack with the priority of 0.
2. Step2: Scan the expression and check if it has more token or not.

* If Yes -> I check if it is an operand (valid numbers)
* If No -> Process the Operator (Go to Process Step)

1. Step3: Check if the token is an operand

* If Yes, it’s a valid number push the token to the operand stack.
* If No, Go to step 4

1. Step4: Check if the token is an operator

* If Yes, Go to step 5
* If No, its an invalid token. Throw an exception.

1. Step5: now we have the operator token.

* first Check if the token is “)” then Go to Step Process. And then remove “(“ from the top of the stack
* Second Check if the token is “(“ then create an object and push it to the operator stack
* Third, 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
* Forth, if none of above happened: while operator’s precedence is less or equal than the precedence of the Operator at the top of the Stack AND the we are not reaching to the token we initially push to the stack “A” the Go to the Process step.

**Process step**

1. Pop the operand Stack twice (for each operand - note the order!!)
2. Pop the operator Stack
3. Execute the Operator with the two Operands
4. Push the result onto the operand Stack

## Technical Overview

We will implement a stack data structure to keep track of operator precedence. Stack data-structure operates on Last in First Out (LIFO) principle.

Also on this project we have an abstract class named Operator which has two abstract methods :

public abstract int priority();  
public abstract Operand execute(Operand op1, Operand op2 )

Also I create a static HashMap to map the String operators (+ - \*^/()A) with the associate Operator class object (AddOperator, SubtractOperator,…)

These Subclasses of Operator abstract class, have been made in order to override the priority and execute function for each operators individually (because we have specific priority and specific execute function for each of the operators).

Static HashMap data structure allow us to have access to the function of the abstract class and the reason of initializing it as static was because When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are global variables. All instances of the class share the same static variable.

## Summary of Work Completed

The program works perfectly.

All the tests are passed, and the GUI works as expected.

# Development Environment

Following development environment made working with evaluator expression project possible:

1. Intellij IDEA version 2018.1(Ultimate version)
2. Jdk1.8.0\_161.jdk

# How to Build/Import your Project

Steps to import project:

1. From the git bash clone the repository from GitHub using the command:

**git clone** https://github.com/csc415-03-spring2019/csc413-p1-rayafsh.git

2. Open IntelliJ and select: **Import Project**

3. Navigate to the folder where the project folder was cloned. Inside the project, folder navigate to the folder path: **rayafarshad\** **csc413-p1-rayafsh**

4. Next select: Create project from existing resourses

5. Select: **Finish** to complete the Project Import Steps.

Steps to build the project:

1. In the IntelliJ under **Build** Menu select: **clean project** to guarantee clean build for the first time

2. Next under **Build** Menu select: **build project**

# How to Run your Project

Steps to Run Project:

1. Under **Run** Menu select: **Run ‘Evaluator Driver’**

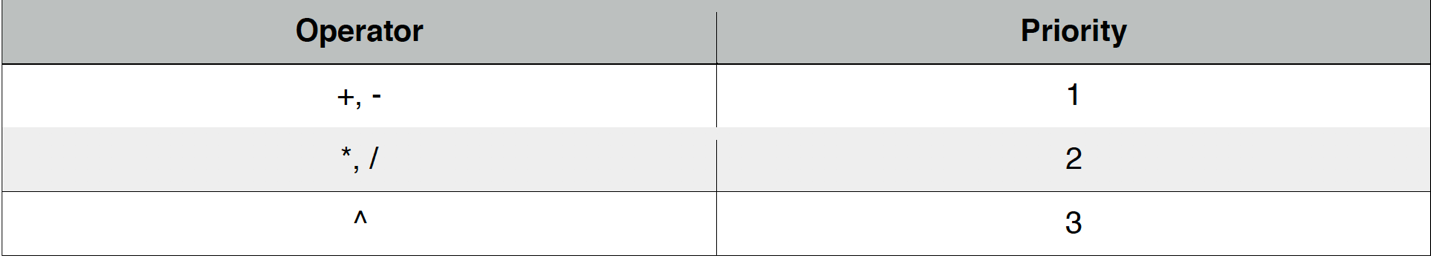
2. To run **EvaluatorTest** file navigate to test folder. Next select **EvaluatorTest** file then right click the mouse and select **Run ‘EvaluatorTest’.**

3. To run **EvaluatorUI** file navigate to evaluator folder. Next select **EvaluatorUI** file then right click mouse and select **Run ‘EvaluatorUI.main()’.**

# Assumption Made

The assumption made for the Operands and Operators are:

* 1. Permitted Operands: A, B, C, D…
  2. Permitted Operators: +, -, \*, /, ^, (, )
  3. All valid operators are binary.
  4. All values of operands are an integer.
  5. Blanks are permitted in expression
  6. Constants are not permitted in the expression
  7. Left and right parenthesis are permitted
  8. The assumption made for the Order of Operations are



* 1. And also the priority of right parenthesis “)” is: 4
  2. And also the priority of left parenthesis “(” is: -1
  3. The initial value I push into the operator stack is “A” with the priority of 0 and the operator name of InitializeClass().
  4. When both addition and subtraction operators are present in the expression, we will prioritize operation on Addition and Subtraction from left to right
  5. We should add space on our delimiters to define space as a delimiter

# Implementation Discussion

1.Encapsulation

The OOP concept means that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. A class can enforce the desired level of restriction to both the data and functions using access modifier such as private, public or protected keywords.

* 1. In the evaluator class I made sure that the attributes of the classes are properly encapsulated by using a private access modifier. Also, some of the methods such as our HashMap was made private.

2.Inheritance and composition

Is a type of relationship between classes known as inheritance and has a type of relationship between two classes is composition.

For example here we have a parent class Operator and different subclasses named Add operator, SubtractOperator and so on that inherits their functionalities from their parent class Operator.

2.1 Abstraction

Let us say we want to create a method named priority() and execute(op1, op2) and these two method has different functionality on each of child classes (**AddOperator, SubtractOperator, MultiplyOperator, DevideOperator, PowerOperator, LeftParentahasis, RightParentahasis, InitialClass…)** for this reason we define these two methods in the parent class as a abstract methods with no body and we override them on the subclasses.

**class** **Operator** {

public abstract int priority();  
public abstract Operand execute(Operand op1, Operand op2 );

}

**class** **AddOperator** **extends** **Operator** {

@Override  
public int priority() {  
 return 4;  
}  
  
@Override  
public Operand execute(Operand op1, Operand op2) {  
 return op1+op2;  
}

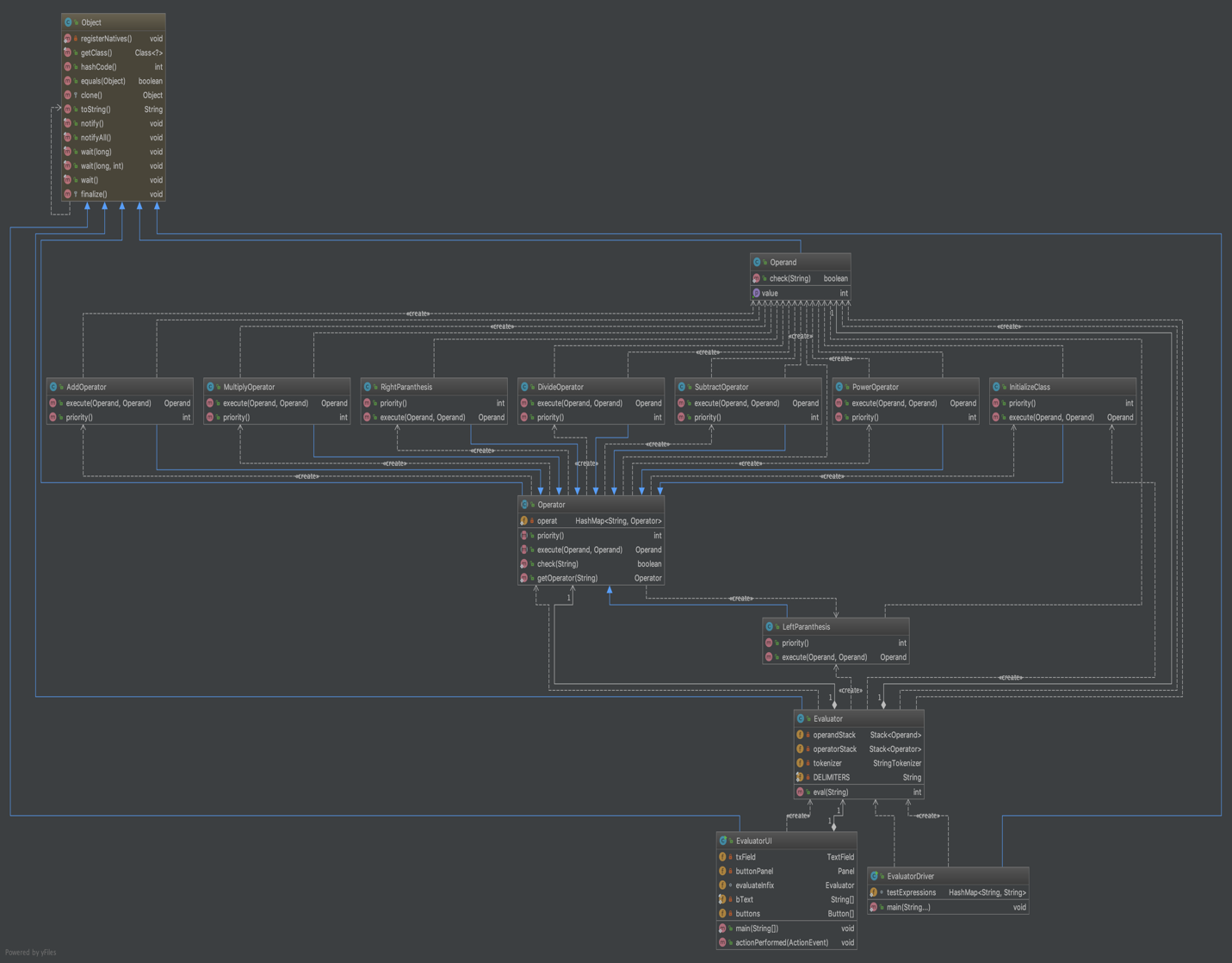
}

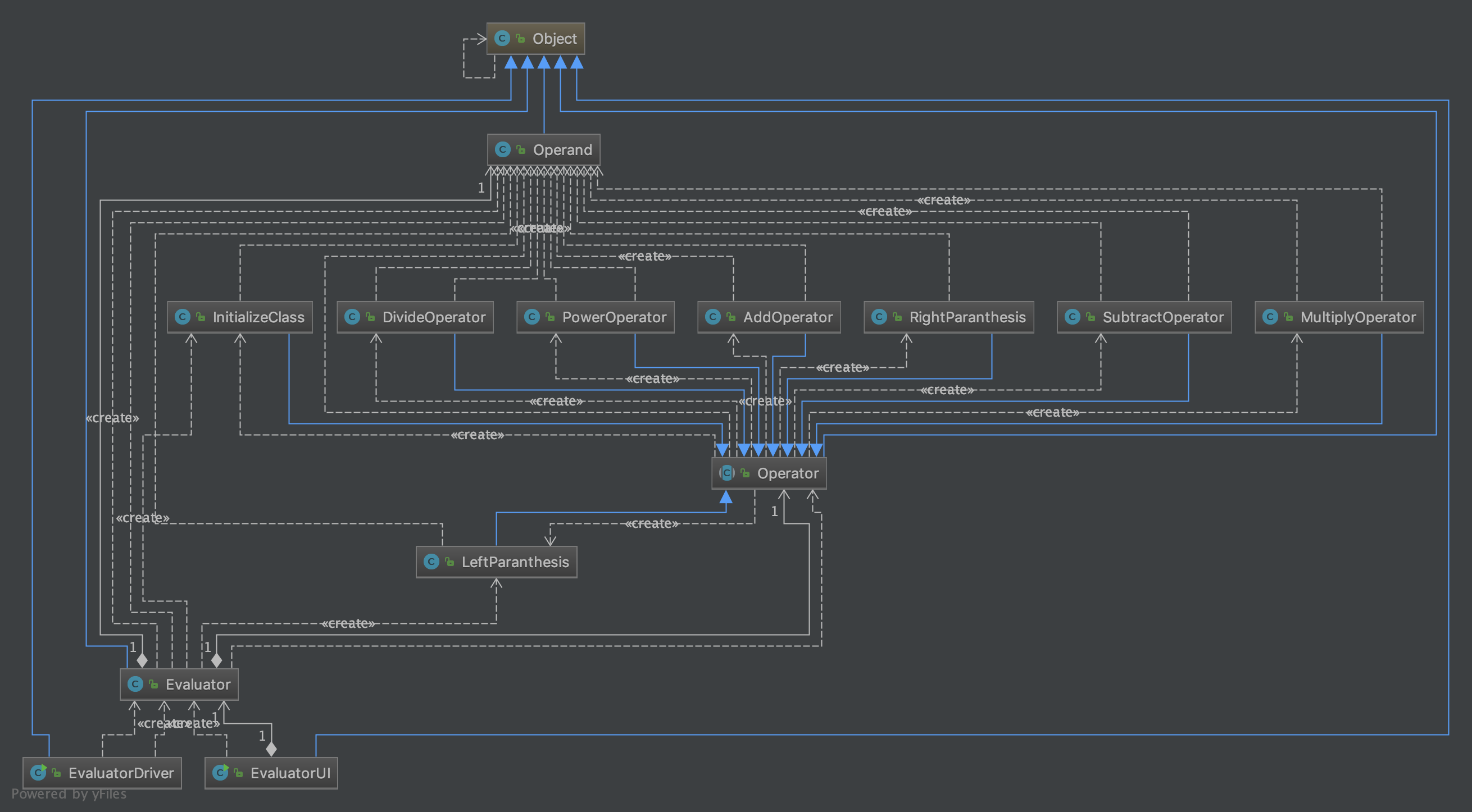
**class** **SubtractOperator** **extends** **Operator** {

@Override  
public int priority() {  
 return 3;  
}  
  
@Override  
public Operand execute(Operand op1, Operand op2) {  
 return op1-op2;  
}

}

## Class Diagram





# Project Reflection

During working on this project I first start with the evaluator class and I tried to implement the algorithm was given to us on the assignment description file. And I realized since we have abstract class I couldn’t instantiate any object from it so I needed to call the function get operator. After that I go to the operator and operand class to fill up the remaining function. I realized I have to abstract methods which I needed to override them on the subclasses. After I finish with that I tried to test some tests and I realized my code is getting out of the wile loop because it pop everything from the operating stack so I realize that I need to Initialize the operator stack with a specific priority. Also another Issue I has was to recognizing the spaces. So I added space and parenthesis to the delimiters.

Then I moved on to the evaluatorUI class to fix the GUI I check for each botton = or C or CE and put an else if function to take care of that.

# Project Conclusion/Results

I am glad that I work on this project it gives me a very great overview of the OOP concepts.

The test all completed perfectly and also the Gui works as it expected.