CSC 413 Project Documentation

Fall 2019

Nimiksha Mahajan

917219230

CSC 413.01

https://github.com/csc413-01-fall2019/csc413-p1-Nimiksha.git

Table of Contents

1 Introduction:

The aim of this assignment was to build an expression evaluator in JAVA. This expression evaluator is meant to evaluate mathematical expressions based on the precedence of the operators used (The priority of the expressions are: "(" is -1; ")" is 0; +/- is 1; */ / is 2 and ^ is 3). Additionally, this program has a GUI designed to display a 4 X 4 calculator where the user can input an expression and get the desired result.

1.1 Project Overview

This project solves infix expressions composed of operands A, B and C, and operators + and *, such as (A+B*C), which follow a certain set of rules based on the priority of the operator between the operands. To solve the expression correctly by accounting for precedence, we convert the infix expression to a prefix expression where like +*ABC, by using Stacks. We push and pop elements on 2 stacks: The Operand stack and the Operator stack.

1.2 Technical Overview

For this project, the main data structures used are Hash Maps and Stacks.

The Hash Maps were used to build a hash map of all the operators and an object for their respective Operator class. The operator is stored as the key and the value is set as the object. The Operator class for each operator returns the priority and the executed result for that operator. For example, the 'AddOperator' object has a priority of 1 and it returns result for 'A + B'.

The Stacks were used to store values of the operand and the operator from the user input. Two stacks were implemented for this program: The Operator stack and the Operand stack. We use tokens to evaluate the user input values, and if the token is an operator (+-*/()), it is pushed onto the Operator stack, and if it is an operand (1,2,3,4,…..) it is pushed onto the Operand stack.

To evaluate the expressions, the program first pops the operator from the Operator stack and then it pops 2 operands from the Operand stack, since we are using binary operators that need 2 values to compute a result. Then this result is pushed onto the Operand stack and the process continues till the entire stack is evaluated. This process is abstracted in the ComputationOp () and after the check if some implementation needs to be performed, this method is called.

To further break down the process of evaluating the stacks, and the precedence of operators:

- First, the program checks if the operator stack is empty or not. If it is empty, there is a call to the Operator object newOperator and push the operator to the Operator stack.
- After that program ensures that there is something in the operator stack, it check to see if the operator is a "(", an opening parenthesis. If it is an opening parenthesis, the program adds the next operator on the stack, and continues.
- When the stack hots a ")", it peeks at the stack and continues till it finds the corresponding opening parenthesis, then evaluating the expression inside the parenthesis. It performs the evaluation by calling the ComputationOp ().
  Once the expression is evaluated, it pops the operator from the stack and continues the traversal down the stack.
- The program also checks that if the stacks are not empty, and if not, it calls the ComputationOp () and pushes the result onto the Operand stack.

This details the workings on the eval () which is responsible for the evaluation of all the expressions. The function returns the value that is at the top of the Operand stack, which is the value achieved after evaluating all the operators based on their level of precedence.

1.3 Summary of Work Completed

To summarize the work done in the program, the Expression Evaluator takes an arithmetic expression from the user and evaluates it based on the components of that expression. Given it is mathematical expression, it has operators with a certain order of precedence, and the evaluator accounts for that.

The Evaluator program returns the correct results and is able to pass all the tests. The GUI works as well, with all the features working correctly. The GUI calculator also has working 'C' and 'CE' button, which delete the complete expression and the last inputted number respectively.

2 Development Environment

The development environment used for this project was IntelliJ Ultimate Version 2019 2.1, with Project SDK, JAVA version 1.8.0_121.

3 How to Build/Import your Project

To build this project, clone the GitHub repo https://github.com/csc413-01-fall2019/csc413-p1-Nimiksha.git

Then import the cloned project on IntelliJ (Open the calculator folder).

4 How to Run your Project

In IntelliJ, build the project and you can run the **EvaluatorUI** to get the GUI.

To ensure that the program is running correctly, you can run the **tests**.

And if you want to run the program without GUI, run the **EvaluatorDriver** class.

5 Assumption Made

The assumptions made for the assignment were:

- Only the +, -, *, /, ^, (and) operators were available. All these operators are binary
- The priority for these operators is:

  “(“ is -1

  “)” is 0

  +/- is 1

  */ / is 2

  ^ is 3
- The operands are only integers. We do not use double or float values.
- Constants are not used in the expression

6 Implementation Discussion

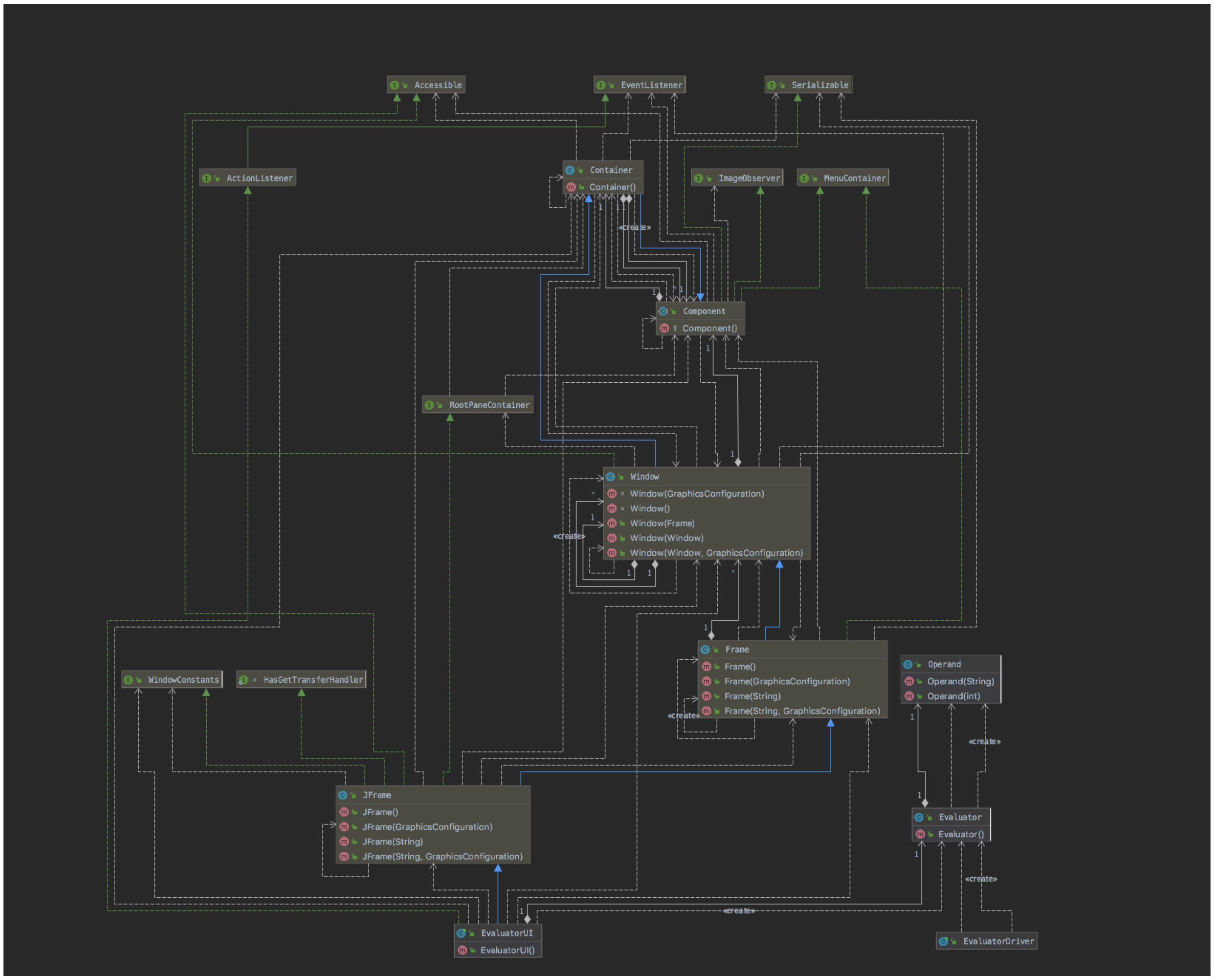
In this program not only did we implement data structures, but also implemented important JAVA Object Oriented Programming (OOP) concepts:

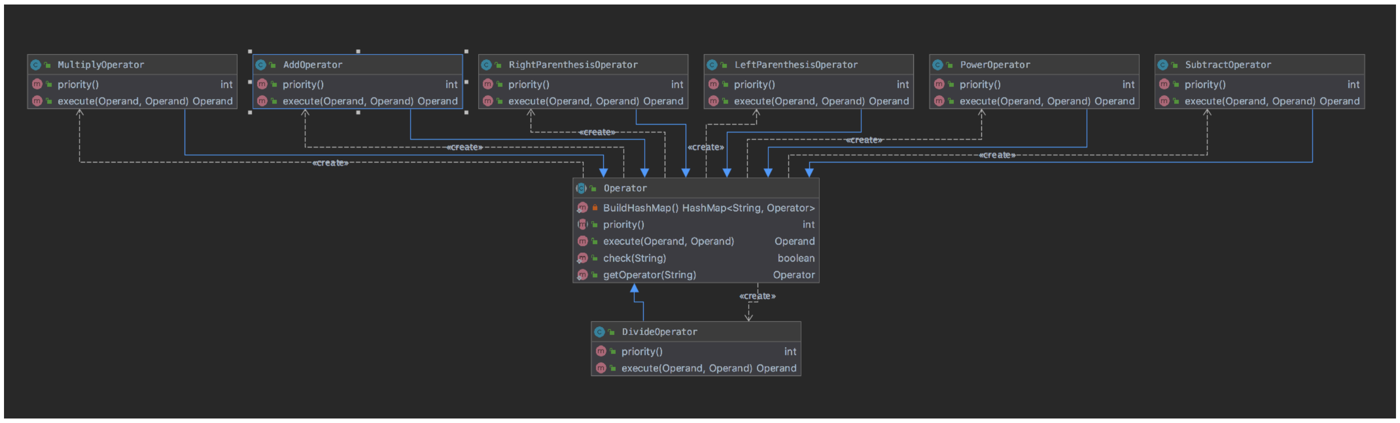
- Inheritance: This program uses inheritance in the operator package, where the Operator class is a super class with sub classes for the operators such as AddOperator, SubtractOperator, MultiplyOperator, DivideOperator, LeftParenthesisOperator, RightParenthesisOperator. The Operator class is the parent class and all the sub classes inherit its properties aka the methods.

- Polymorphism: The Operator class has abstract methods like priority () and execute (), and these abstract methods are inherited by the subclasses of the Operator class. These subclasses, even though, inherit the methods from the parent class, they have different functionality. For instance, the priority () returns a different value for all the operators and the execute () returns values based on different evaluations i.e. sum if its add or difference if its subtract etc.

- Encapsulation: Encapsulation is implemented in this program with the attributes of the Evaluator class declared as private. Also, the Hash Map with the operators is given private access. Encapsulation here plays an important role as it binds the data and functions and keeps them safe from unwarranted changes. It is inaccessible for change to other classes.

6.1 Class Diagram





MultiplyOperator
priority() int
execute(Operand, Operand) Operand
AddOperator
priority() int
execute(Operand, Operand) Operand
RightParenthesisOperator
priority() int
execute(Operand, Operand) Operand
LeftParenthesisOperator
priority() int
execute(Operand, Operand) Operand
PowerOperator
priority() int
execute(Operand, Operand) Operand
SubtractOperator
priority() int
execute(Operand, Operand) Operand
«create»
«create»
«create»
«create»
«create»
«create»
Operator
BuildHashMap() HashMap<String, Operator>
priority() int
execute(Operand, Operand) Operand
check(String) boolean
getOperator(String) Operator
«create»
DivideOperator
priority() int
execute(Operand, Operand) Operand

7 Project Reflection

As I reflect back on this assignment, I feel good about it. I was confused in the beginning but once I reviewed the Assignment document a good 5 times and read through the code that was given, I had an idea of what I needed to do. I worked on the Operand class first and then the Operator class. Working backwards helped me breakdown how the program should work. I was not too sure about my grasp on OOP objectives before, so I had the opportunity to review them in detail and that really helped. I had to review the shunting yard algorithm to understand the concept on infix, postfix and prefix expressions, and while that was a bit confusing, I do believe it helped me understand how to set up the eval (). One of my most frustrating moments was when my eval () wouldn’t work even though I was sure I had the concept down and it made sense in my head. I spent 6 hours working on it, only to find out I had commented out the .put for the 2 parenthesis in my HashMap (in the Operator class). It was both frustrating and funny, but I realized how these small mistakes are a part of a programmers' life and the joy I felt when my code actually worked is beyond words. The longs hours were worth that.

One of the most important things about this assignment for m was GitHub actually. I’ve no prior experience with GitHub and just maneuvering around it was challenging at first, but became fun later.

There were a lot of things I struggles with but there is a whole lot I learnt too!

8 Project Conclusion/Results

To conclude, this was a very interesting project where I learned a lot. I was able to implement a GUI, which I have never done and was able to work with OOP concepts, which I know are of essence.

I am happy with the results of my work because I ended up with a working calculator with a working GUI, that evaluates expressions based on the precedence of the operator. This assignment has left me kind of excited to work on the next one!