Laith Hussin
921659619
Feb 5-2022

Assignment 1 Documentation

Github Repository

https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-1---calculator-laith57th.git

Project Introduction and Overview

Using a code skeleton provided, I was required to implement a partially coded program for an evaluator class. I was also required to program a simple user interface that uses the evaluator class to perform various mathematical calculations using the correct operator hierarchy.

Scope of Work

Task		**Completed**
Initialize the methods for the Operator abstract class		**X**
	abstract int priority()	**X**
	abstract Operand execute(Operand op1, Operand op2)	**X**
	boolean isOperator(String token)	**X**
Statically declare the Operator subclasses in the operators HashMap		**X**
	Addition, subtraction, multiplication, division, operator, open and close parentheses operators.	**X**
Implement the Operator methods to each Operator subclass by returning their respective priority and correct execute algorithm.		**X**
	Use @override for each method to prevent the instantiation of each method more than once.	**X**
Implement the methods and constructor for the Operand Class		**X**
	Operand(String token)	**X**
	Operand(int value)	**X**
	Int getValue()	**X**
	Boolean isNumber()	**X**
Create a helper method in the evaluator class and complete the algorithm using the abstract Operator class and the Operand class.		**X**
	Void processOperators()	**X**

Using given EvaluatorTester class, test the Evaluator class’s eval function using all possible cases:		**X**
	(6 * 3 + 8 + 3)	**X**
	3 + (2 * 3 + (6 + 2))	**X**
	4 + (1 + 2)	**X**
	(6 + 8) + 9	**X**
Using the implemented Evaluator class, complete the algorithm for EvaluatorUI		**X**
	Implement the actionPerformed function to handle button presses and use the eval method to calculate incoming infix expressions.	**X**
Organize project		**X**
	Created a package to store Operator class and Operand class called Operation package.	**X**
	Created operators package within the operationPackage that holds the Operator subclasses.	**X**

Execution and Development Environment

To complete this project, I used Intellij on my macbook and windows PC.

Compilation Result

Using the terminal, I used the following commands on the project files:

```
> javac EvaluatorTester.java
> java EvaluatorTester
> javac EvaluatorUI.java
> java EvaluatorUI
> javac Evaluator.java
```

The program ran as expected and no error messages were displayed

Assumptions

I assumed that incoming infix expressions were valid. For example, no extra parentheses, no trailing operators (i.e. ++, --), and every open parentheses had its corresponding closed parentheses. I also assumed all operands are integer values.

Implementation

Operator class

I began working on this project by implementing the operator class. Looking at the Evaluator class, I knew I needed a working operator and operand classes before touching the Evaluator algorithm. Following the instructions on the assignment page and hints in the code, I initialized a HashMap to hold the operators I am interested in keyed by its corresponding subclass that extends the abstract operator class. Then I simply left the priority() and execute(Operand, Operand) to be implemented in each subclass with their respective values and calculation algorithm.

Operator subclasses

For the implementation of the subclasses, I used @override to prevent the instantiation of either operator method more than once. Then, following the instructions for this assignment, I returned the respective priority value to each operator subclass. Finally, for the execute method, I created an Operand object that holds the correct calculation for each operator subclass.

Operand

The Operand class was mostly easy to implement. Two constructors, one holds a String token and the other holds the corresponding integer value of the String, one getter method that returns the integer value of the Operand object, and the boolean isNumber that checks if scanned token is a string value. The boolean method was a bit tricky, however, after a few attempts, I ended up using a try and catch to parse the token and check whether it indeed is a string token.

Evaluator

After implementing the Operator class, Operand class, and the operator subclasses, I was ready to begin working on the Evaluator class. After a few thinking sessions, I began to understand the overall algorithm and how everything works together. I noticed that I'll be using the "process operators" algorithm more than once, so I decided to create a method that holds these lines of code. Next, I opened the assignment instructions document and used a few if/else if/else statements in the first while loop that hold the different conditions specified in the document. After the first while loop ends and the expression is fully transformed into values inside either the operand or operator stacks, if there are any operators left in the operator stack other than parentheses, process them until the operator stack is empty then return the top of the operand class as the answer.

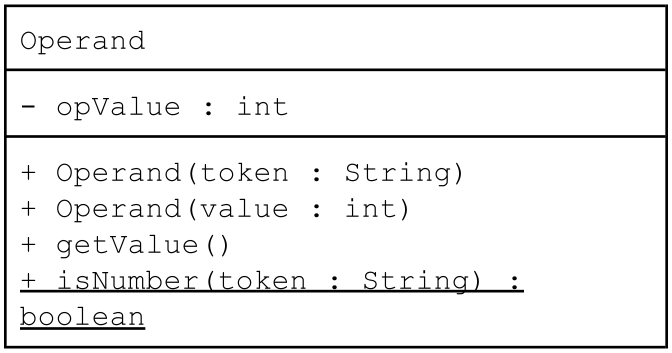
EvaluatorUI

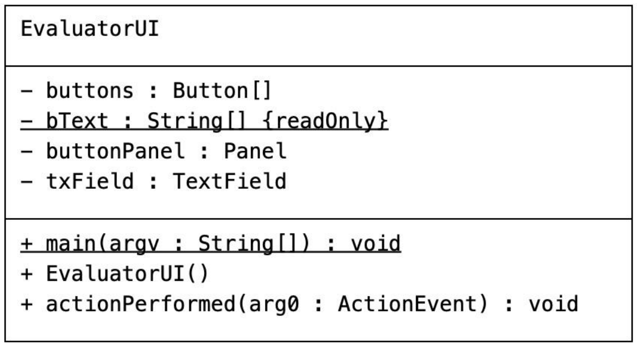
This class was fairly easy to implement, most of the code was already done for me. I did notice that there were three for loops that loop through the same thing so I decided to merge them into one to reduce runtime. For the actionPerformed function, I created a String that holds the button presses then checks if the button pressed is =, C, or CE. For the = button, I created an Evaluator object and used the eval method I previously implemented to calculate the String token. As for C and CE, I set the function to clear the textfield. Finally, for all other buttons, the function prints the number or operator on the screen.

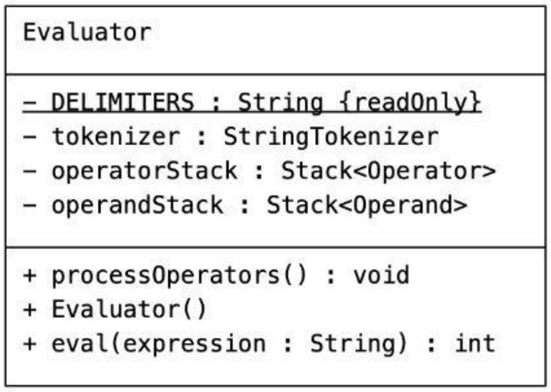
Code Organization

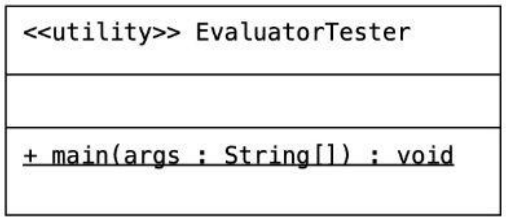
To better organize this project I moved the Operator and Operand classes into a package, created a different java file for each Operator subclass and placed all of them into a package within the Operator and Operand package.

Class Diagram

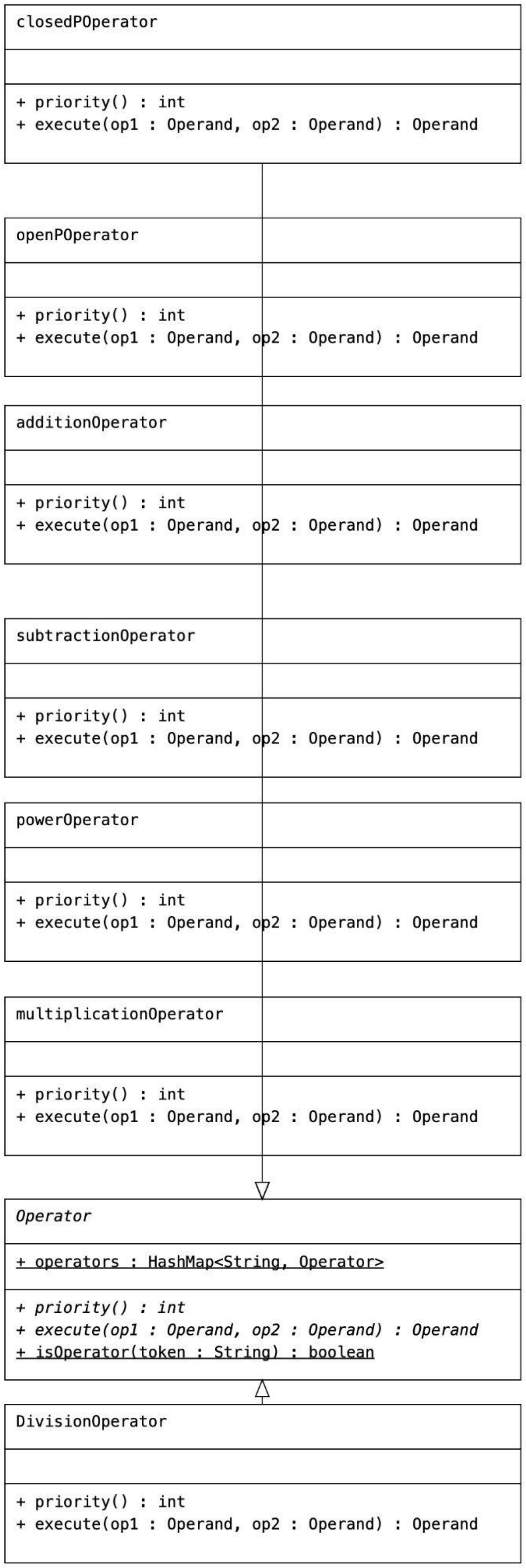

Operand
- opValue : int
+ Operand(token : String)
+ Operand(value : int)
+ getValue()
+ isNumber(token : String) : boolean


EvaluatorUI
- buttons : Button[]
- bText : String[] {readOnly}
- buttonPanel : Panel
- txField : TextField
+ main(argv : String[]) : void
+ EvaluatorUI()
+ actionPerformed(arg0 : ActionEvent) : void


Evaluator
- DELIMITERS : String {readOnly}
- tokenizer : StringTokenizer
- operatorStack : Stack<Operator>
- operandStack : Stack<Operand>
+ processOperators() : void
+ Evaluator()
+ eval(expression : String) : int



operationPackage.operator


closedPOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
openPOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
additionOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
subtractionOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
powerOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
multiplicationOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
Operator
+ operators : HashMap<String, Operator>
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand
+ isOperator(token : String) : boolean
DivisionOperator
+ priority() : int
+ execute(op1 : Operand, op2 : Operand) : Operand

Results and Conclusion

This project was a good review for java and object oriented programming. I also learned a lot about github and how it's used in the real world. I successfully implemented all the required classes and java files required in this project.

Challenges

The Operator class was especially difficult for me to implement. I haven't had good practice with abstract classes in the past and implementing the HashMap was a hassle since I didn't quite grasp the idea of statically assigning the values of the HashMap. This caused many problems in the eval method, especially. After placing many lines of text across the program, I was able to track and deduce the core of the issue. All in all, it was a good learning experience.

Future Work

I believe one way I can improve this program is to implement a better error checking routine inside the eval method and improve the EvaluatorUI by deleting the result in the text field as soon as a button is pressed for a new calculation.