Sandesh Basnet

Link (https://github.com/csc413-03-sp18/csc413-p1-SandeshOnRails)

Project Introduction and Overview

This project was an introduction to Objected Oriented Design while programming to provide structure and scalability to the code. The project's goal was to implement an expression evaluator. The expressions would be tokenized by the String Tokenizer class in Java, and each of the tokens would be push into two different stacks, operator and operand, depending on their type. Type checking for the tokens is done in Operator and Operand class for each token. Operator is an abstract class that provides abstract methods that returns the priority and executes the operation depending on the type of the Operator. The abstract methods in the Operator class are implemented by the different Operator classes that extend the Operator abstract class, as the priority and execution of the different Operators are different. The Operator class has a static Hash Map that is used in the Evaluator class to map operator string and Operator object to it. The object of type operator is initialized in the Evaluator class using an interface in the Operator class that returns the type of Operator object (addition, division…). The Evaluator class had a skeleton of the algorithm, that reads all the tokens through a while loop, and makes checks for whether the token is an operator or operand. If the token is an operand then it is pushed to the operand Stack in the Evaluator class. We were assigned to implement the algorithm for the Operator Stack. I added checks for the token to be a right parenthesis first, and if the token is a right parenthesis, then I solve the expression inside the () until I reach the left parenthesis. If left parenthesis is encountered I exit the loop and pop it from the Operator Stack. If the token is not a right parenthesis, and the operator Stack still has operators and the operator on the top of the operator stack has higher precedence that the token recently read, and the token does not equals left parenthesis, perform execution on top two values of the operand stack popping the top of the operator stack. When all the tokens are read and pushed into their respective stacks, I finally check if there are any operators remaining in the stack, and if so, I pop the operand and the operator stack and solve the expression. The final value of the execution is pushed into the operand stack. Finally, I pop the operand stack for the final result and return it.

Instructions to Compile as Jar and Execute

Java –jar Assignment1.jar

Development Environment

NetBeans IDE 8.2 with JDK version 8.

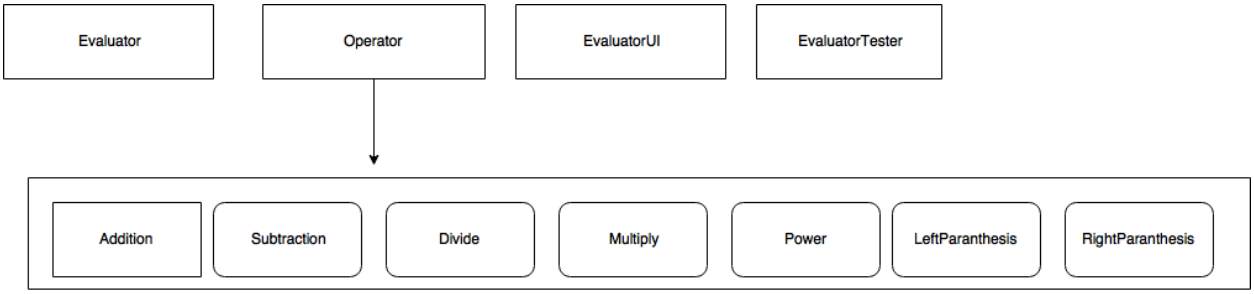
Assumptions

The project skeleton was provided, so I thought we needed to sub class Operators and not much of work would go on the Evaluator class. But as I worked on, my algorithm for evaluating expressions only solved non-braces expressions. I also had concerns about handling Arithmetic Exception.

Implementation

The Evaluator class uses abstract class Operator, and it's subclasses, which is the type of operator for each class, for defining it's algorithm. The evaluator class has two different operator and operand stacks that store the type of the operator objects. Java's String Tokenizer library is utilized to get tokenize the expression, and each token is checked if it's an operator or operand, and pushed into its respective stack. The interfaces in the operator and operand class check whether the token is an operand or operator. Upon recognizing an operand, the operand class gets the token gets pushed in the Operand Stack. If the token is an operator then static checkOperatorType() is called in the Operator class to initialize the Operator object depending on its type because Operator cannot be instantiated as it is an abstract class.

Below is the class hierarchy.



Evaluator

- OperatorStack<Operator>()- Stack to store Operator object depending on its type.
- OperandStack<Operand>() – Stack to store Operand object.
- Evaluator()- constructor that initializes the stacks and puts values into the static HashMap of Operator Class.
- Int Eval(String token) – Evaluates the token and returns the calculated value;

Operator

- Static HashMap<String, Operator>() – HashMap to store the type of Operator object and it's string value.
- Abstract int priority() – priority of the operator
- Static boolean check(String token) – check if the token is an operator
- Static Operator checkOperatorType(String token) – check the type of the operator.

Operand

- Private String token – private field to initialize the object with the string operand value.
- Operand(String token) – constructor to initialize the Operand object with it's string value.
- Operand(int value) – overloaded constructor to initialize the Operand object with the integer value.
- Int getValue() – returns the integer value of the Operator object.
- Static Boolean check(String token) – check if the token is an Operand.

  All the Operator subclasses extend and implement the abstract methods in the Operator Class.

EvaluatorUI

Implements Action Listener Java interface's actionPerfomed method. The actionPerformed method takes parameter ActionEvent. Using the ActionEvent object I got the string associated with the Event to set the Text Field in the UI. After the user triggers the event after the "=" button is clicked the Evaluator class is instantiated and the eval function of the Evaluator class takes the string in the text field as its parameter and the value returned by the eval function is displayed in the textfield. EvaluatorUI successfully evaluated the expression.

EvaluatorTester

I used an array and populated the array with the expressions of the Hard, SimpleExpression, and SimpleParanthesis test respectively, and evaluated each of the test cases. All the test cases was passed.

Results and Conclusion:

I was successfully able to pass all the test cases provided , and also I test my work with some expressions that I created, and it successfully passed those as well. The most challenging part for me was to figure out how to solve the embedded parenthesis as initially my algorithm only evaluated SimpleParanthesis, I used to the debugger to optimize my algorithm to pass the hard test.