ASSIGNMENT 1
CALCULATOR

Hadia Andar
915397384
CSC 413, Summer 2018
GITHUB LINK: https://github.com/csc413-01-su18/csc413-p1-handar
Collaborated with: Stephanie Santana

Introduction

a. Project Overview

The purpose of this assignment is to create two programs, with the first one being about an object that evaluates mathematical expressions by using hashmaps, delimiters, and strings and the second one being a functional GUI calculator that is made up of the first program. The calculator runs our Evaluator algorithm. I collaborated with Stephanie Santana.

We were given an already pre-written source code that we had to fill in to execute the test cases correctly. The code takes in a string that represents a mathematical expression. This string can come from either EvaluatorTester.java or custom-written by the user using the GUI calculator. This string is broken up using a stringTokenizer.

The Operand and the Operator class (along with its subclasses) are very important and what the Evaluator class will use to determine the order of operations. This order of operations is called “priority” in the code. Each operator is assigned a numerical priority number. The code takes each token in the string, checks if it is either an operand or operator. It pushes it onto the appropriate stack and evaluates based on Operator priority.

b. Technical Overview

We made a graphical user interface that creates an object of our abstract Evaluator class and built a fully-functional simple calculator. We used our Operator class that had the operator hashmap and the operand keys from the GUI, along with their functions that were in their appropriate subclasses, and produced an algorithm that works for all valid inputs.

c. Summary of work completed

In the end, we created a fully-functional simple calculator was written. The user can input in mathematical expressions and run test cases. Our EvaluatorTester class was successful in providing the correct output using the algorithm. The valid inputs prove that the Calculator is able to evaluate and calculate the correct results for the given strings and tokens.

Development environment

a. Java: 1.8.0_161; Java HotSpot(TM) 64-Bit Server VM 25.161-b12
b. Netbeans IDE 8.2

How to build or import your game in the IDE you used.
Importing:
File -> New Project -> Java Project with existing sources -> Name your project folder→ Select and import existing sources → -> Browse and choose Assignment 1 folder -> Finish

For running the EvalutorTester. Java → right click → run file

1. If the EvaluatorTester.java outputs the correct answers, then you can run the whole project
2. You can run the project from the Run Tab on menu Bar (select Run Program) or press the play button on the toolbar

Building and Running Project:
Go to EvaluatorUI.java. In the menu: Run -> Build Project -> Run Project

Assumptions Made when designing and implementing your project
Operand Class:

- the user inputs a valid string and that there are no negative numbers
- must create an int variable *intValue* to hold the operand value
- must create a hashmap to hold key and value of each operator
- the operators must have a priority similar to basic algebra rules (PEMDAS)
- the abstract Operator class must have subclasses from the defined keys and values to implement its functions
- the Evaluator is the most important part of the project since it has the algorithm for the calculator to process the strings and tokens received through user input and the EvaluatorTestor class
- Hashmaps are going to be very useful in the code to keep things organized

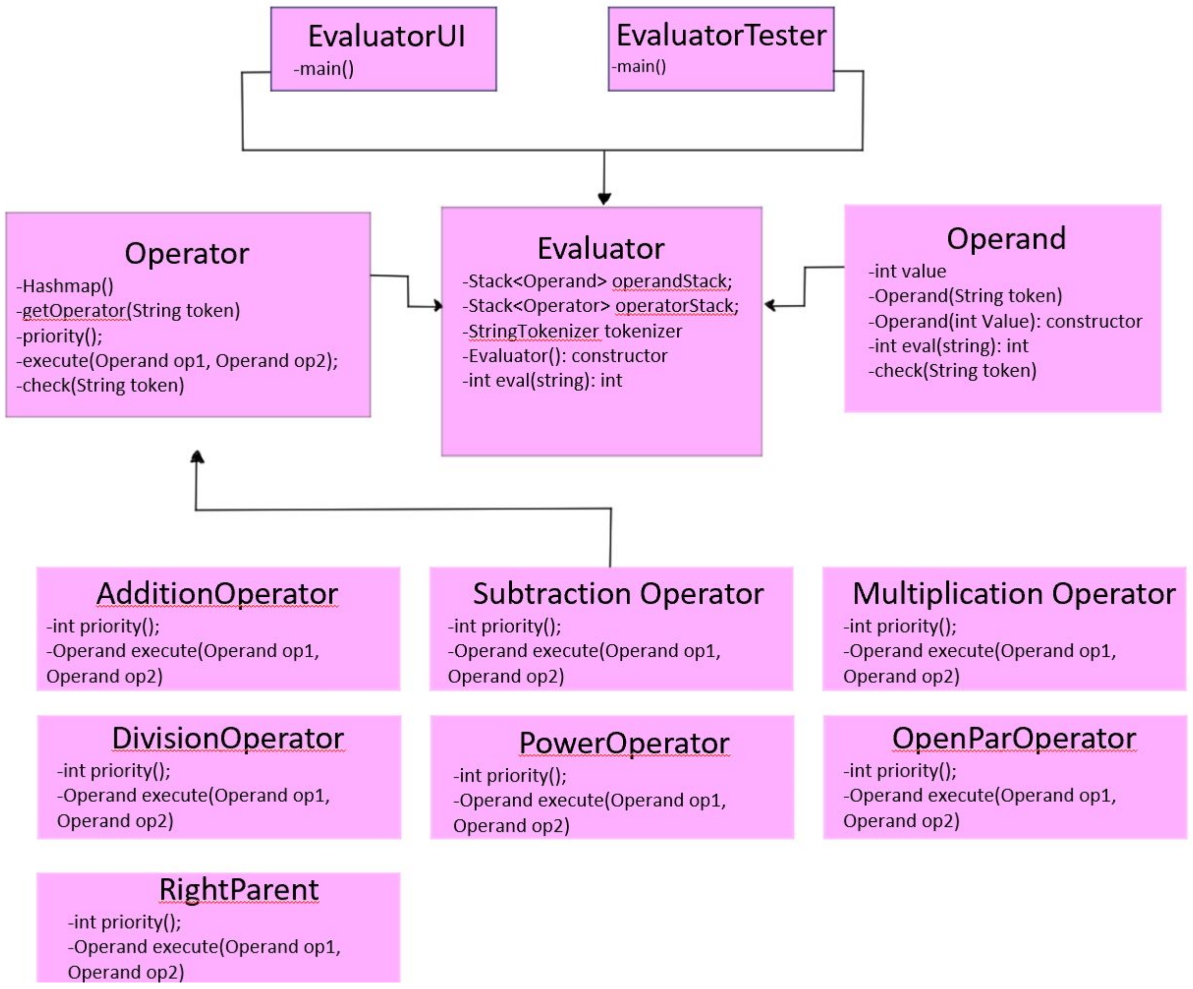
Operator Classes for The Following:

- AdditionOperator() -> priority of 2
- SubtractionOperator() -> priority of 2
- MultiplicationOperator() -> priority of 3
- DivisionOperator() -> priority of 3
- PowerOperator() -> priority of 4
- OpenParOperator() -> priority of 1
- RightParent() -> priority of -1

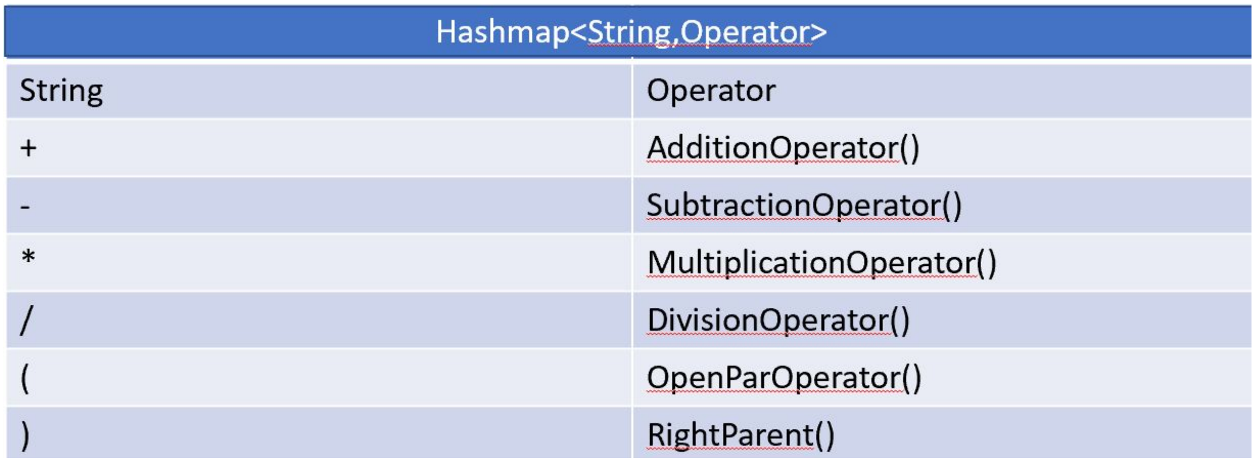
Other Assumptions:

- Organization is key
- Plan out the code before writing it all
- Don’t write the code out in one go

Implementation Discussion
The pink diagram below shows the hierarchy of the code. EvaluatorUI and EvaluatorTester act as the main classes because those are the classes that you run the program on. Below that are the 3 classes: Operator, Evaluator, and Operand that hold the main functions that hold up this program. And below that are the 7 Operator subclasses that implement the abstract Operator class's functions.


EvaluatorUI
-main()
EvaluatorTester
-main()
Operator
-Hashmap()
-getOperator(String token)
-priority();
-execute(Operand op1, Operand op2);
-check(String token)
Evaluator
-Stack<Operand> operandStack;
-Stack<Operator> operatorStack;
-StringTokenizer tokenizer
-Evaluator(): constructor
-int eval(string): int
Operand
-int value
-Operand(String token)
-Operand(int Value): constructor
-int eval(string): int
-check(String token)
AdditionOperator
-int priority();
-Operand execute(Operand op1, Operand op2)
Subtraction Operator
-int priority();
-Operand execute(Operand op1, Operand op2)
Multiplication Operator
-int priority();
-Operand execute(Operand op1, Operand op2)
DivisionOperator
-int priority();
-Operand execute(Operand op1, Operand op2)
PowerOperator
-int priority();
-Operand execute(Operand op1, Operand op2)
OpenParOperator
-int priority();
-Operand execute(Operand op1, Operand op2)
RightParent
-int priority();
-Operand execute(Operand op1, Operand op2)

We used hashmaps to our advantage with strings and functions as the parameters. A hashmap was implemented with string and operators in the abstract Operator class. The string was made up of a token from the delimiter that was defined in Evaluator.java. In the hashmap, every string in the delimiter is bound to an operator, so whenever in the hashmap the string +, -, *, /, ^, (, and) are called, the appropriate operator function is used. Each operator has its own subclass that overrides the priority() and execute() method in the operator class.



Hashmap<String,Operator>	
String	Operator
+	AdditionOperator()
-	SubtractionOperator()
*	MultiplicationOperator()
/	DivisionOperator()
(	OpenParOperator()
)	RightParent()

Each subclass had its own priority in the mathematical equation listed in the priority function. Within each subclass operator is the executed steps of the operator.

The Operand class's implementation is made up of the operand constructor that parsed the string token to an integer, a getValue function to get and return the operand value, and the boolean check to make sure that the string token is converted to an integer.

The Evaluator class's implementation has the two stacks, operandStack and operatorStack. Each stack would either peek, pop, or push an operator depending on its' priority value, and then pop the operands out of the stack. Within the class a stringTokenizer was designed to break up the string of user input into two different stacks, one being for the operand, and the other for the operator.

When everything has been implemented, there are two ways to run the project. The Evaluator class can first be tested using the EvaluatorTester.java to make sure that the logic and algorithm in the Evaluator class is correct. If none of results of the tester file are in error, then that means that an Evaluator object was successfully created. The second way to run the project is running it from EvaluatorUI which depends on user input in the calculator to evaluate the mathematical expression and to test the Evaluator object.

Project reflection

This project was focused on remembering the key programming components in Java that we had learned in CSC 210 and CSC 220. The Hashmap did not make sense to me at first, but after watching a lot of resource videos and looking at implementation examples, it made sense why the calculator needed it. It really helped to organize my code. I spent many hours coding this project and refreshing and brushing up my memory of its resources. The project was kind of confusing at first too because I had trouble finding a way to implement the "(" and ")" strings. I kept thinking "how does the computer just know how to use (and) in the mathematical expression?". But then I realized that it is all about what priority number I gave my Operators. I used if statements and while loops to my advantage in Evaluator.java to tell the program when to push and when to pop. So we essentially created a simple PEMDAS algorithm.

Project Conclusion and Results

The hardest part in the project was figuring how how to implement the parenthesis so the expression stayed in PEMDAS order. The Evaluator algorithm that was given to us took a long time to understand at first because we had to read through and process the hints and expectations that were given. The trick to processing through the parenthesis and everything else was to use while and if statements to compare operator priorities. The result was a simple calculator using hashmaps and stacks.