Brandon Watnabe
921988506
February 16, 2022

Assignment 1 Documentation

Github Repository 1
Project Introduction and Overview 1
Scope of Work 2
Execution and Development Environment 3
Compilation Result 3
Assumptions 3
Implementation 3
Evaluator 3
Operator and the Strategy Software Design Pattern 3
Operand 4
EvaluatorUI 4
Code Organization 4
Class Diagram 4
Results and Conclusion 4
Challenges 5
Future Work 5

Github Repository

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

Project Introduction and Overview

For this project, I had to implement a class and its many subclasses, then use those to finish a partially completed calculator algorithm. Then I had to integrate that algorithm into a partially finished calculator UI.

Scope of Work

Task		Completed
Implement the eval method of the Evaluator class		X
Test the implementation with expressions that test all possible cases. The following expressions were used:		X
	1+(1+2*3/4^5)+2	X
	1+2	X
	2*3	X
Implement methods in the abstract Operator class		X
	boolean check(String token)	X
	abstract int priority()	X
	abstract Operand execute(Operand operand1, Operand operand2)	X
	Lookup mechanism for operators to prevent instantiation of the same operator more than once	X
Implement Operator subclasses		X
	Properly organize subclasses (I used a package named operators)	X
	Implement subclasses for the required operands: multiplication, division, addition, subtraction, exponentiation, parentheses)	X
Implement Operand class		X
	Constructors (String and int parameters)	X
	boolean check(String token)	X
	int getValue()	X
Complete implementation of the Calculator UI in the EvaluatorUI class		X
	Use the previously implemented Evaluator	X
	Implement the actionPerformed method to handle button presses	X

Execution and Development Environment

I used the IntelliJ IDE on my Windows to complete this assignment and tested in both the IDE and shell.

Compilation Result

Using the instructions provided in the assignment one specification:

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

No error messages or warnings were displayed, and the application ran as expected.

Assumptions

I assumed that the algorithm for computing the equations was correct and would provide me with the right answer if I did everything in order. I also assumed that the provided code would work and so I did not modify it much nor did I implement any exception handling.

Implementation

Evaluator

The Evaluator class was partially completed for me, but I needed to implement the algorithm provided in the assignment worksheet. Before starting this step, I laid out a plan of how to structure my code to adhere to the algorithm. I created a few methods to minimize repetition and to make it look cleaner. The performOperation method handled the calculations of the Operators and Operands. The processNewOperator method takes in the current Operator object that we are looking at and determines if it is a parethesis or another Operator. If it is an open parenthesis, it will be pushed to the stack. If it is a closed parenthesis, then it will call performOperation until it meets the open parenthesis again. If it is any of the other operators, then it will also be pushed to the stack.

Operator and the Strategy Software Design Pattern

Implementation for this project required us to understand and implement the Strategy pattern as described in class. The Strategy pattern allows a programmer to specify a family of algorithms as an abstract class, and to provide concrete implementation of that algorithm that may be used to vary behavior at runtime.

Implementing the Operator class and its subclasses required the use of the strategy design pattern that was discussed in class. The Operator class was an abstract class that was used as a blueprint for each different operator. These subclasses were: additionOperator, subtractionOperator, multiplicationOperator, divisionOperator, exponentiationOperator, openParenthesisOperator, and closedParethesisOperator. Structuring it this way allowed for each operator to perform similar tasks, but with different values and operations.

Operand

The Operand class took the least amount of time to implement, however, it was a bit tricky implementing the check method until I remembered that I can Exception e to clear the error and return false.

EvaluatorUI

The EvaluatorUI class was difficult for me to implement as I was not very familiar with JFC. It required me to implement the actionPerformed method which we were given an example of in class. That sample code was quite repetitive so I trimmed down all of the if else statements for all of the operators and operands into one else statement so it was more modular. I then had to implement the =, C, and CE buttons. The C and CE buttons both did the same thing as instructed. When a user inputs the equals button, it will use an Evaluator object to perform the calculation and returns the solution to the text field.

Code Organization

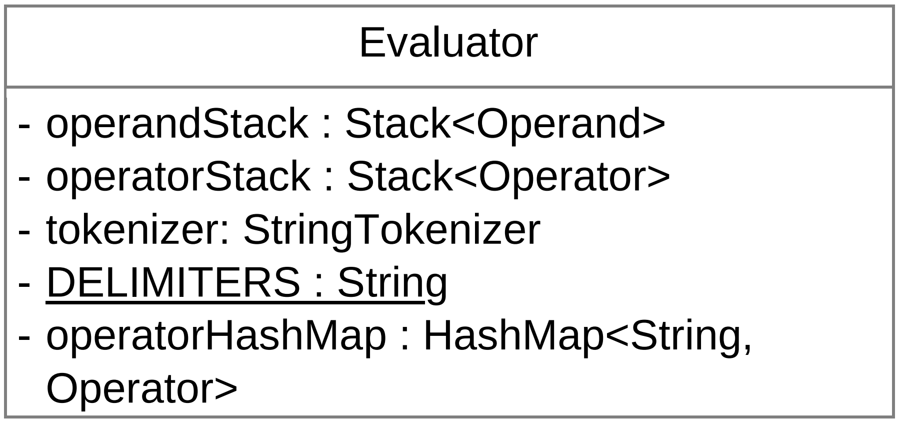
I compacted all of the operators into a package so that there are not a bunch of unorganized files in the main directory.

Class Diagram

The following class diagram shows the details of all of the classes in this project, including the inheritance hierarchy (of course you will need to provide all classes used with the appropriate level of detail - please ensure that each is easily legible, which may require splitting this diagram into multiple diagrams across multiple pages):

Results and Conclusion

This project helped me refresh my Java programming and challenged me in the UI portion as I was not familiar very with it. It also refreshed my memory on the Strategy design pattern that I learned in my Data Structures class.


Evaluator
- operandStack : Stack<Operand>
- operatorStack : Stack<Operator>
- tokenizer: StringTokenizer
- DELIMITERS : String
- operatorHashMap : HashMap<String, Operator>

+ Evaluator()
+ eval(expression : String) : String
- performOperation()
- processNewOperator()

Challenges

The most challenging part for me was figuring out how to implement the algorithm in the eval method. I thought I had it pretty planned out before starting to code it, but I ran into multiple issues that I overlooked in the planning phase. The debugger saved me many hours as I could go line by line and figure out where it is going wrong and why I keep getting errors. I eventually realized that I would have to create a placeholder operator so that it doesn't crash when using .peek()

Future Work

I think that a way to develop this calculator further would be to allow it to use more operators like a scientific calculator. Adding the functionality of using trig functions, roots, etc. would be challenging but very cool.