Name: Yan Peng
Student ID: 921759056
Sep 13th, 2021

Assignment 1 Documentation

GitHub Repository

Link for the assignment 1

https://github.com/sfsu-csc-413-fall-2021/assignment-1-calculator-yuqiao1205

Project Introduction and Overview

The evaluator project required me to implement an infix expression evaluator given a code skeleton, and to perform basic mathematical functions. The aim of this project is to understand object-oriented programming, Strategy pattern and how to use Java Swing. I handle the priority of operator and parentheses by using stack and the priority of operators. To isolate and encapsulate the operator execution and priority logic, I have implemented operator instance classes such as AdditionOperator, SubstractionOperator etc.

Scope of Work

Task					Completed
Implement the eval method of the Evaluator class					✓
Test the implementation with expressions that test all possible cases. The following expressions were used:					✓
simple	1+2	7*8	3^3		✓
	9-5	9/3			
nested	(2*(2+3))/2				✓
	((3-2)(2+4))*2				
special	1				✓
normal	(1+2)*(3+4)				✓
	1+(2+3*4/2)+1				✓
	2^2^3				✓
	2^(2^3)				✓
	2-3-4				✓
Implement the methods in the abstract Operator class					✓
	boolean check(String token)				✓
	abstract int priority()				✓
	abstract Operand execute(Operand operandOne, Operand operandTwo)				✓
	Lookup mechanism for operators to prevent instantiation of the same operator more than once				✓
Implement the methods in the abstract Operator class					✓
	Properly organize subclasses (I used a package operator)				✓
	Create a subclass of Operator, ArithmeticOperator				✓
	Implement subclasses for the required operands: multiplication, division, addition, subtraction, exponentiation, parentheses)				✓

Task		Completed
Implement Operand class		✔
	Constructors(String and int parameters)	✔
	boolean check(String token)	✔
	int getValue()	✔
Complete implementation of the Calculator UI in the EvaluatorUI class		✔
	Use the previously implemented Evaluator	✔
	Implement the actionPerformed method to handle button clicks	✔

Execution and Development Environment

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

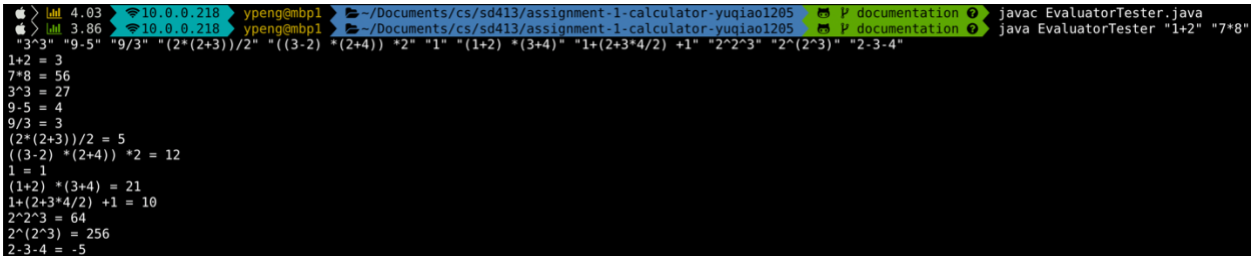
Compilation Result

Using the instructions provided in the assignment one specification:

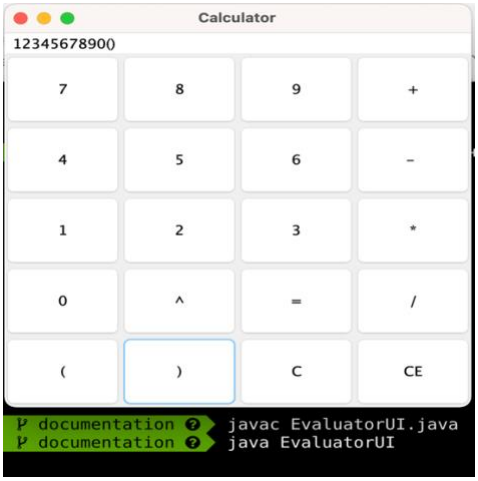
```
> javac EvaluatorTester.java
> java EvaluatorTester "1+2" "7*8" "3^3" "9-5" "9/3"
"(2*(2+3))/2" "((3-2)*(2+4)) *2" "1" "(1+2) *(3+4)"
"1+(2+3*4/2)+1" "2^2^3" "2^*(2^3)" "2-3-4"

1+2 = 3
7*8 = 56
3^3 = 27
9-5 = 4
9/3 = 3
(2*(2+3))/2 = 5
((3-2)*(2+4))*2 =12
1 = 1
(1+2)*(3+4)= 21
1+(2+3*4/2)+1 = 10
2^2^3 = 64
```

```
2^(2^3) = 256
2-3-4 = -5
```




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



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

Assumptions

I assumed that all expressions provided to the Evaluator's eval method would be correct, so I did not implement any error handling for invalid tokens.

Implementation

Evaluator

As I implemented this algorithm, I noted a few cases of duplicated code. I have also enhanced algorithm to support nested bracket cases. Once I had the general algorithm working, I organized this code and created new methods such like

initializeStacks, parseExpression, processToken, processRemainingOperators and process methods, as the single purpose act of processing both operators and operands from their respective stack was required in multiple places in the eval method.

Operator and the Strategy Software Design Pattern

Implementation for this project required me to understand and implement the Strategy pattern as described in class. The Strategy pattern, a class behavior or its algorithm can be changed at run time.

The operator class provided the abstract base class for our implementation of this pattern, specifying an abstract method: execute, that would be implemented in derived classed to provide the varying behavior for each of the operators. The subclasses that were created were: AdditionOperator, SubtractionOperator, MultiplicationOperator, DivisionOperator, and PowerOperator. Other operators, appear as symbols, but do not execute anything will throw an exception if the execute method is called.

ArithmeticOperator

The ArithmeticOperator class is inherited from the Operator class to distinguish arithmetic operations such as + - * / ^ from non-arithmetic operations e.g., “(” and “)”.

Operand

The Operand class was straightforward to implement but did require some attention to the fact that an Operand could be instantiated with a String or an Integer. Since the logic I used for the String constructor and the check method were similar, I considered repurposing the check method to test for valid String input in this constructor, but decided to just include a try/ catch block in the constructor, since the purpose of the check method was to provide a boolean check on whether or not a String input was a valid operand, and the purpose of the constructor was to convert the operand to an integer.

EvaluatorUI

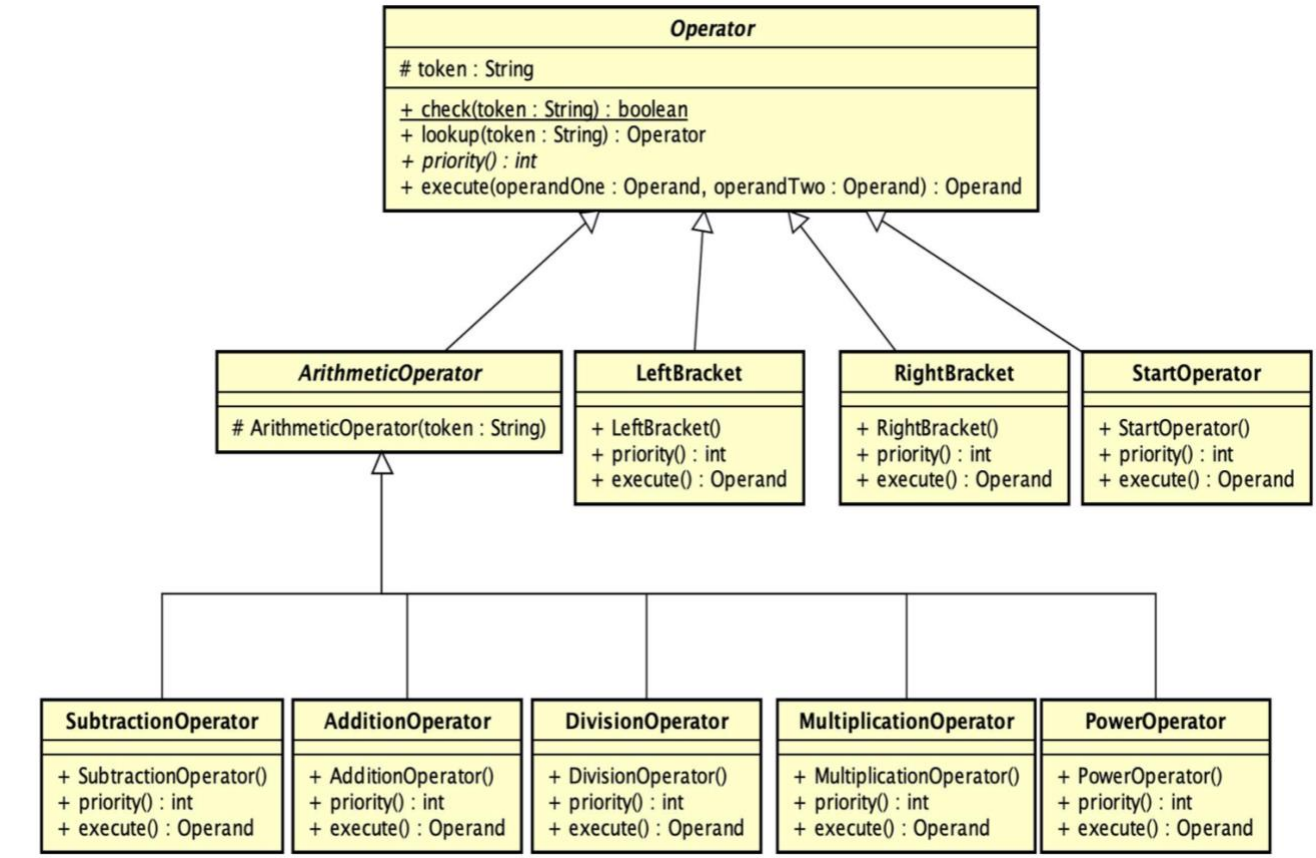
I implemented the actionPerformed method from the ActionListener interface. I created method for handling the special buttons "=", "C", and "CE".
The "CE" button clears the most recent entry in the text field and "C" button clears all the text field. The "=" button triggers evaluation of arithmetic expression string from the value of text field and evaluates that expression using the Evaluator class and its eval method. After evaluating the expression, the result is returned as a String, and placed back in the text field.

Code Organization

I created an operator package containing the family of operators, and I moved derived subclasses into the two sub-package arithmetic and common. Operator class, LeftBracket class, RightBracket class and StartOperator class are in the sub-package common and the sub-classes of ArithmeticOperator, operators (+, -, *, /, ^) are in the sub-package arithmetic. Having the intermediate class ArithmeticOperator makes it easy to distinguish the arithmetic operators from others in the code.

Class Diagram

The following class diagrams show the details of all the classes in this project, including the inheritance hierarchy



Evaluator

- operandStack : Stack<Operand>
- operatorStack : Stack<Operator>
- DELIMITERS : String

+ Evaluator()
+ eval(expression : String) : int
- parseExpression(expression : String) : void
- processToken(token : String) : void
- processOperator(token : String) : void
- processRemainingOperators() : void
- initializeStacks() : void
- process() : void
- evalUntilMatchingLeftBracket() : void

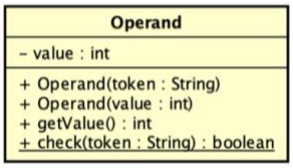
EvaluatorUI

- txField : TextField
- buttonPanel : Panel
- bText : String[]
- buttons : Button[]

+ main(args : String[]) : void
+ EvaluatorUI()
+ actionPerformed(event : ActionEvent) : void
- clearLastCharacter() : void
- evalExpression() : void

EvaluatorTester

+ main(args:String[]) : void



Results and Conclusion

This project served as a good refresher for Java, as well as an introduction to an object-oriented design pattern, Strategy pattern. I successfully implemented all the required features defined in the project specification.

Challenges

The eval method was challenging for me to implement. Especially when I tackled cases with left or right brackets in the stack, especially when nested. I considered how to organize packages to make code organization better. To organize the packages, I group related classes and think of it as a folder in a file directory. Some changes I made to the algorithm during development produce subtle errors that I found during testing. To address these, I analyzed the possible causes of the error messages and confirmed my hypotheses using the breakpoint debugger.