Expression Evaluator and Calculator GUI
Software Documentation
Table of Content

Expression Evaluator and Calculator GUI

Software Documentation

Contributor: Shan Kwan Cho
GitHub Repository: https://github.com/csc413-03-sp18/csc413-p1-ShanKwanCho.git

1. Introduction:

This project is called calculator project. The purpose of this project is to be able to evaluate and operate mathematical expressions. And this calculator project express GUI (graphic user interface).

1.1 Overview:

Users can input the numbers depends on their need by using the Operators (addition, subtraction, multiplication, division, etc). User can face the user interface of keypad of the calculator along the operations.

Note: This project is running on Mac OS High Serra and IDE is NetBeans
More info IDE of NetBeans: https://netbeans.org

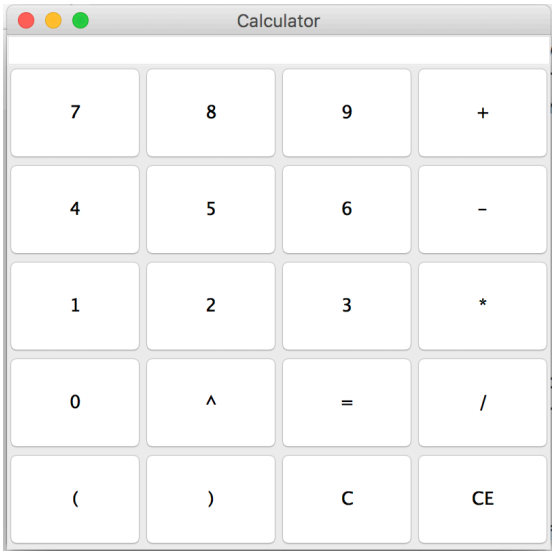

Fig1.1: Calculator User Interface

1.2 User Interface

The Calculator User Interface allows the user to solve the mathematical expressions by pressing particular operators.

1.3 Project Components:

Technically, calculator project includes five java files

1. Evaluator.java
2. EvaluatorTester.java
3. EvaluatorUI.java
4. Operand.java
5. Operator.java

1.4 Basic Structure (Hierarchy):

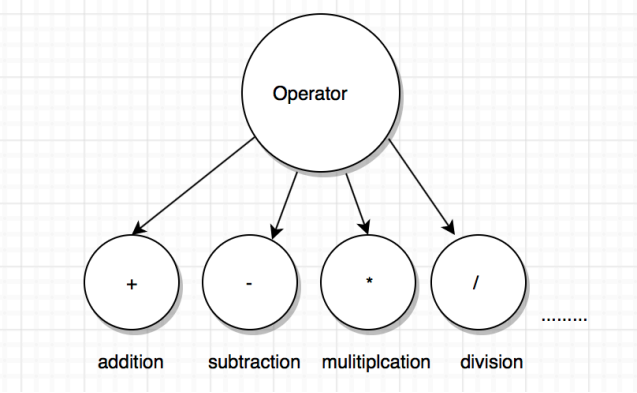

Fig 1.4.1: Opeartor.java

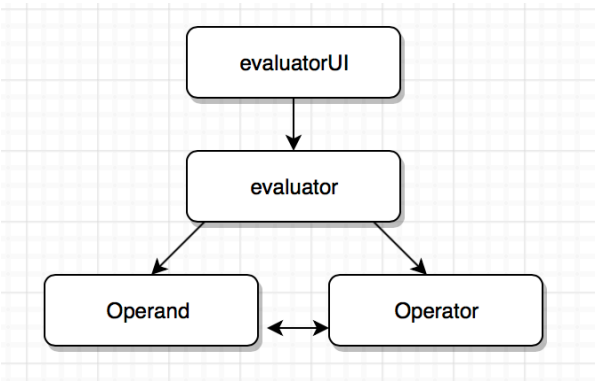

Fig 1.4.2: Overall Hierarchy (calculator)

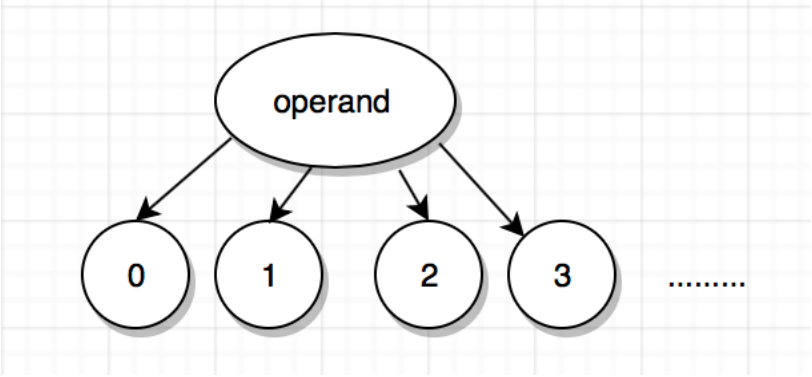

Fig 1.4.3: Operand.java

2. Technical Guide

2.1. Scope of Work

2.1.1 Algorithm of Calculator

In Calculator algorithm, we will use two stacks :

- Operand Stack: values(numbers) and
- Opeartor Stack: operators (+, -, *, / and ^).

Possible processes

1. Pop operand stakc once (value1)
2. Pop operator stack once (operator)
3. Pop opeator stack again (value 2)
4. Compute value1,operator and value2
5. Push the value obrained in operand stack

Take and perform the algorithms (i) to (vi) until the end of expression is reached,

i. If the character is an operand, push it onto the operand stack.

ii. If the character is an operator, and the operator stack is empty then push it onto the operator stack.

iii. If the character is an operator and the operator stack is not empty, and the character's precedence is greater than the precedence of the stack top of operator stack, then push the character onto the operator stack.

iv. If the character is "(", then push it onto operator stack.

v. If the character is ")", then "process" as explained above until the corresponding "(" is encountered in operator stack. At this stage POP the operator stack and ignore "("

vi. If cases (i), (ii), (iii), (iv) and (v) do not apply, then perform possible process as mentioned above.

Reference : http://csis.pace.edu
More Info : http://csis.pace.edu

2.1.2 Calculator Development

Operator.java: First of all, a developer develop with the Operator.java and Operand.java. For Operator.java, create instance of Hashmap in operator class shown in Fig 1.1.

```
//create new instance of Hashmap in operator class
static HashMap <String, Operator> operators = new HashMap <String, Operator>();
```

Fig 2.1.2 :Hash Map for Operator

Note: HashMap will use as keys as token that developers' attention and values will be instances of the operator.

Operand.java: For Operand.java, create boolean check(String token) to return true if the specified token is an operand. Build the constructor of string token and int value and int getValue() to returns the integer value of the operand. And develop the class of operators (addition, substation, multiplication, division,..,etc) along with the override method(extended from Operator.java) in order to follow the pop the Operand Stack twice, execute the operator with two operands and push the result onto the operand stack. After all token are read, process Operators until the operator Stack is empty.
For more info, please look source codes.
Note: Order matters for each operand.

Evalutor.java: Put all the mapping operator to corresponding method and initialize the operator Stack(necessary operator of priority schema) and should be aware of the usual Mathemetical operators "+-*/" is less than priority. Operators allow to remove some comparisons from the loop, at the cost of one or two additional operations outside of the token processing loop. For more info, please look source codes.

EvalutorUI.java: Create the operator buttons total of 20 number from left to right, top to bottom And create the bText[ ] array that contains the corresponding buttons. As well Add the button to the button panel and listen mouse input. For more info, please look source codes.

EvalutorTester.java: No require to add anything unless the developer write his or her own test cases.

3.Testing and Execution

There are five testing case in sources files: Addition Test, Basic, Hard Test, Parentheses Test and UI Test.

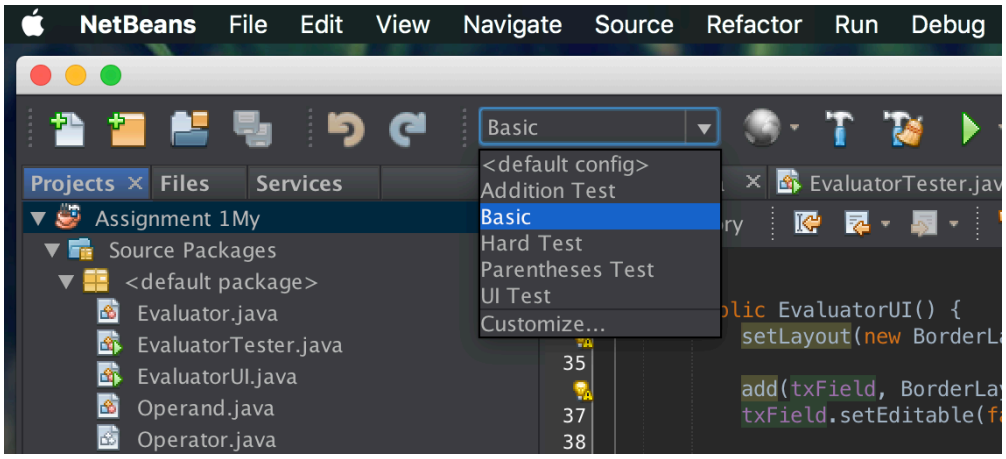

Fig: 3.1 Tests files

After testing successfully, each test can get the result of BUILD SUCCESSFUL (total Time: 0 seconds)

For errors and bugs, see section 8: Support, Contact and FAQ

4.Assumption

If the user input the operation of zero divided by five, in the real calculation will output undefined however this calculator will release the output of zero. Another assumption is that the users operate negative numbers and return the same output of the negative numbers.

5.Implementation discussion

For the implementation discussion, I had the error while running every tests and showing the error message shown below:

```
Exception in thread "main" java.util.EmptyStackException
        at java.util.Stack.peek(Stack.java:102)
        at java.util.Stack.pop(Stack.java:84)
        at Evaluator.eval(Evaluator.java:128)
        at EvaluatorTester.main(EvaluatorTester.java:6)
```

Fig: 5.1 exception in thread error message

Slack private channel: csc413-spring2018

Collaboration Panel:
I collaborated with one of the Slack member @Jed issuing with the errors shown.

Fig 5.2 collaboration panel on Slack

6.Results and conclusions

Calculator project have to implement with many operator methods. Especially, I learned the usage of HashMap, stack, token and override methods. And I am now able to deploy the pieces of code my code by reading the instructions, chopping into detail and figure it out. When I face with individual test bug, the best strategy is to run the tests whenever I insert updated code and clone as a backup for in case problems. I do believe that testing piece by piece is safer and save time to debug the corresponding errors and prevent overloading of errors. One of the useful tips that I learned throughout this project is that collaborating with same project collaborator and contribute and collaborate, debug the issue, errors and develop the particular area together. This strategy is very useful and good routine to keep in the future in software developing environments.

7.Credit and Reference

Algorithm: http://csis.pace.edu
Collaboration on Slack: @Jed jahmadi1@mail.sfsu.edu
Slack: private channel: csc413-spring2018
IDE NetBeans: https://netbeans.org

8. Support, Contact and FAQ

If you have issue with the calculator project, please let us know by email: scho4@mail.sfsu.edu

9. License

This project is licensed under [calculator project Version 1.0]