Calculator And GUI

CSC 413 Summer 2017

Professor Anthony J Souza

Developer's Guide

Richa Mirashi
CSC 413 Summer 2017

Contents

Introduction:

This document provides information about GUI calculator. It elaborates the functionality of the calculator and classes required for developing the calculator. This document contains instructions to compile and execute the program, assumptions, and results and conclusion sections as well.

The GUI calculator is an application used to evaluate a mathematical expression. Evaluator class plays a crucial role in this project. EvaluatorUI and EvaluatorTester classes use the Evaluator class to evaluate a mathematical expression. The Evaluator Class receives input (integer values only) from the user, either from the command line or by mouse clicks, and evaluates an expression by using Operator classes (+, -, *, /, ^) and parentheses.

Scope of Work:

The scope of work for this project was to complete Evaluator, Operator and Operand classes and to develop a calculator that takes only integer values to solve any mathematical expression. Parentheses functionality and EvaluatorUI class implementation were optional. In EvaluatorUI class, the framework was already given, we only had to implement an actionPerformed() method as part of the Java AWT framework.

Instructions to compile and execute the program:

Github link: https://github.com/CSC413-SFSU/expre-eval-richamirashi.git

Clone Assignment 1 project from above link.

How to run project by GUI:

Open NetBeans and open Assignment 1 project. Open the EvaluatorUI class. Click on ‘Run’ tab and then select ‘Build Project’ or ‘Clean and Build Project’ option. Then, click on Run tab -> Run File. A calculator will pop up on the screen. Enter mathematical expressions and then click on ‘=’ button. The output or an answer of a mathematical expression will be displayed in a Text field of the calculator. ‘C’ and ‘CE’ buttons are used to clear the whole expression and to clear the existing value respectively.

By EvaluatorTester class:

Instead of the EvaluatorUI class, open the EvaluatorTester class. Enter the mathematical expressions that you want to test in args (line number 4) as below (refer Fig 3.1):

```
1   public class EvaluatorTester {
2     public static void main(String[] args) {
3       Evaluator evaluator = new Evaluator();
4       args = new String[]{"3/(2-3)", "2+3", "2-1", "(2*3)^2"};
5       for ( String arg : args ) {
6         try {
```

Fig 3.1 Input from the EvaluatorTester class

Save the program. Click on 'Run' tab and then select 'Build Project' or 'Clean and Build Project' option. Then, click on Run tab -> Run File. You will find the correct output in the console (refer Fig 3.2).

```
-----------
Processing expression = 3/(2-3)
-1.0
-3.0
3/(2-3) = -3
-----------
Processing expression = 2+3
5.0
2+3 = 5
-----------
Processing expression = 2-1
1.0
2-1 = 1
-----------
Processing expression = (2*3)^2
6.0
36.0
(2*3)^2 = 36
BUILD SUCCESSFUL (total time: 0 seconds)
```

Fig 3.2 Console output after running the EvaluatorTester class from NetBeans

By command line:

Open the command prompt. Change your current directory to Assignment1 -> build -> build -> classes

In classes folder, you will find all the classes files.

To run GUI calculator from the command prompt, use the following command:

java EvaluatorUI

After entering above command on the command prompt, a calculator will pop up. Click on the buttons to form a mathematical expression that you want to evaluate in the calculator and then click on '=' button to get the answer of the expression (refer Fig 3.3).

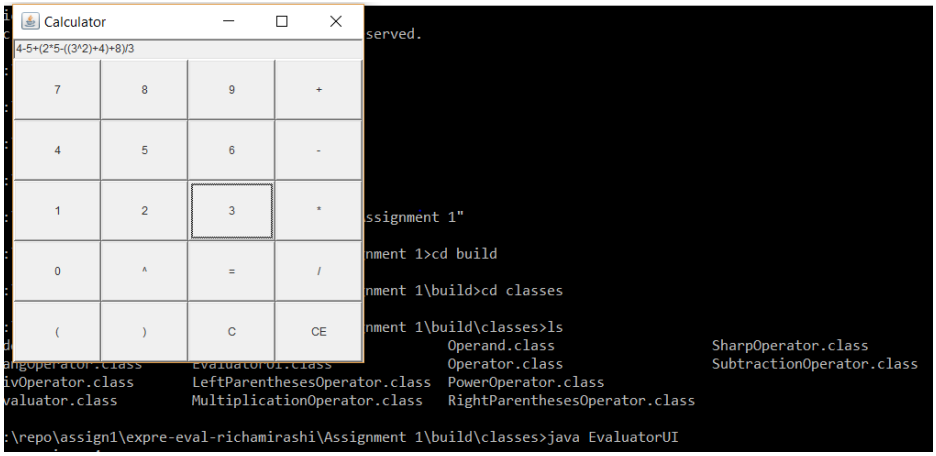

Fig 3.3 The EvaluatorUI class compilation and execution from the Command line

To run the EvaluatorTester file from the command prompt, first, comment line number 4 (args = new String[]{"3/(2-3)", "2+3", "2-1", "(2*3)^2"};) from EvaluatorTester class from NetBeans (If commented, no need to do this) and then click on Run tab -> Build Project.

Then, in the command prompt, be in the same directory (i.e. in the classes folder) and type the following command there.

java EvaluatorTester mathematical expressions

Mathematical expressions are the expressions that you want to evaluate (refer Fig 3.4). Please use double quotes (““) while entering any mathematical expression in the command prompt.

```
D:\repo\assign1\expre-eval-richamirashi\Assignment 1\build\classes>java EvaluatorTester "2+3" "3-4" "((2*3)-(3*4))^2" "4 - 10"
-----------
Processing expression = 2+3
5.0
2+3 = 5
-----------
Processing expression = 3-4
-1.0
3-4 = -1
-----------
Processing expression = ((2*3)-(3*4))^2
6.0
12.0
-6.0
36.0
((2*3)-(3*4))^2 = 36
-----------
Processing expression = 4 - 10
-6.0
4 - 10 = -6
```

Fig 3.4 The EvaluatorTester compilation and execution from the Command line

Assumptions:

NetBeans IDE 8.2 was used as a development environment. The following assumptions were made while developing a calculator:

1. A GUI calculator evaluates an expression only when '=' button is pressed. When running from the command line or from the EvaluatorTester class, a mathematical expression should not have '=' operator as it is an invalid operator.
2. All operators are binary, they work with two operands only.

Implementation Details:

The workflow of all the classes is described as below (refer Fig 5.1):

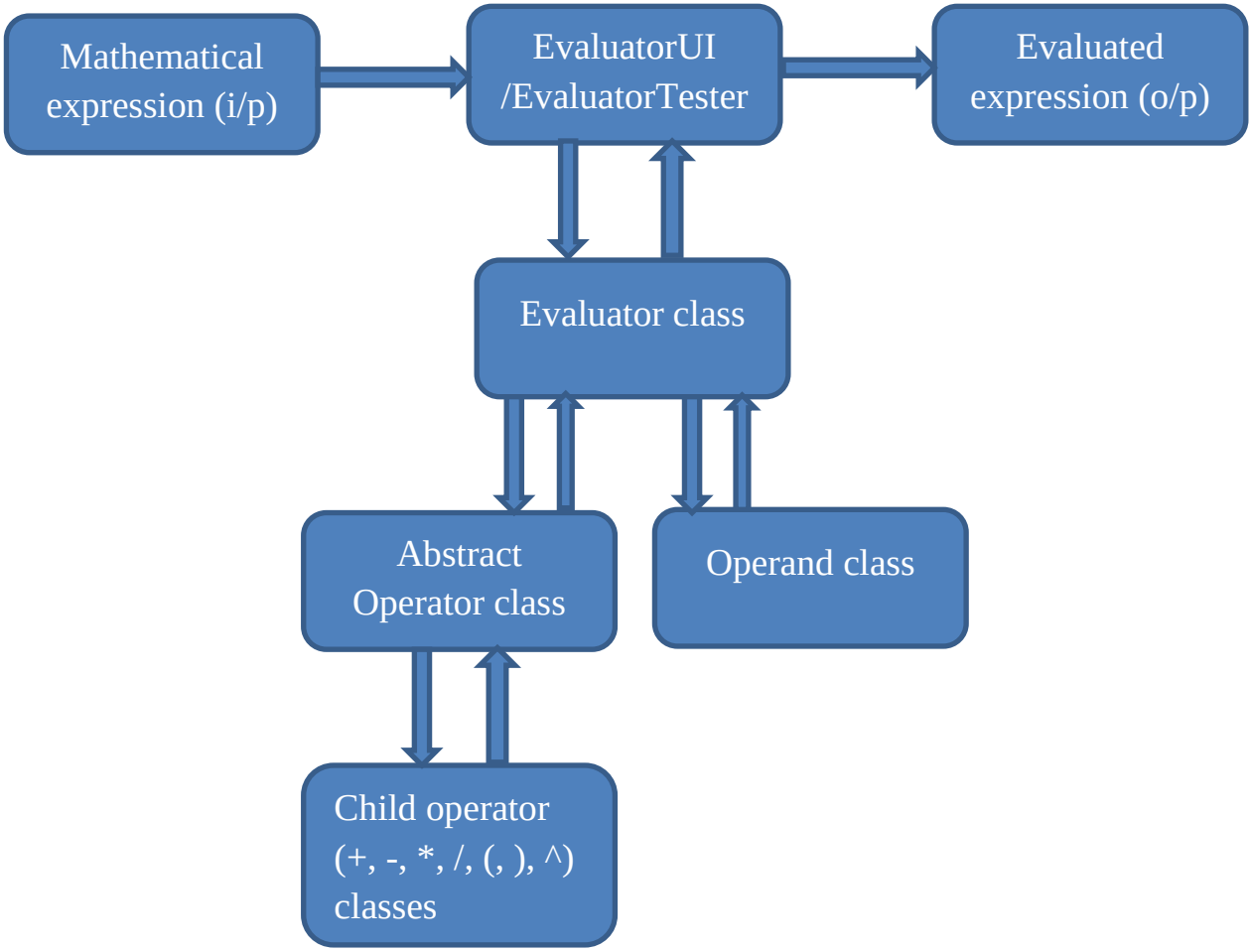

Fig 5.1 Workflow of a calculator project

Class Overview:

The EvaluatorUI and the EvaluatorTester classes receive user input (integer values only) in the form of a string expression and pass this expression to the Evaluator class. The Evaluator class calls the Operand class, and the Abstract Operator class and its subclasses or child classes.

EvaluatorUI Class:

The EvaluatorUI class is a part of Java AWT (Abstract Window Toolkit) framework. It creates the GUI calculator display by using Java Swing and AWT libraries. GUI calculator has 20 buttons (0-9 buttons, +, -, *, /, ^ operators, (,) parentheses, = operator, and C and CE buttons to clear the text field) and a text field to display a mathematical expression and the result. The EvaluatorUI class uses the ActionListener interface and calls an actionPerformed() method for every mouse click. In actionPerformed() method, the EvaluatorUI class instantiates the Evaluator class when the user clicks on '=' button and passes the entered expression (user input) as an argument to the 'eval()' method of the Evaluator class. The eval() method returns the evaluated expression (output) to the EvaluatorUI class and the output or the result is displayed in the text field of the calculator.

'C' button: Clears the whole expression.

'CE' button: Clears the existing value.

EvaluatorTester Class:

The EvaluatorTester class is a public class and it instantiates the Evaluator class. The EvaluatorTester class passes mathematical expressions to the Evaluator class and displays the evaluated answers in the console or in the command prompt after running this file.

Evaluator Class:

The Evaluator is a public class and the 'eval(String expression)' method of this class plays an important role in evaluating the expression. In the constructor, a HashMap is initialized to add all the operators in the HashMap.

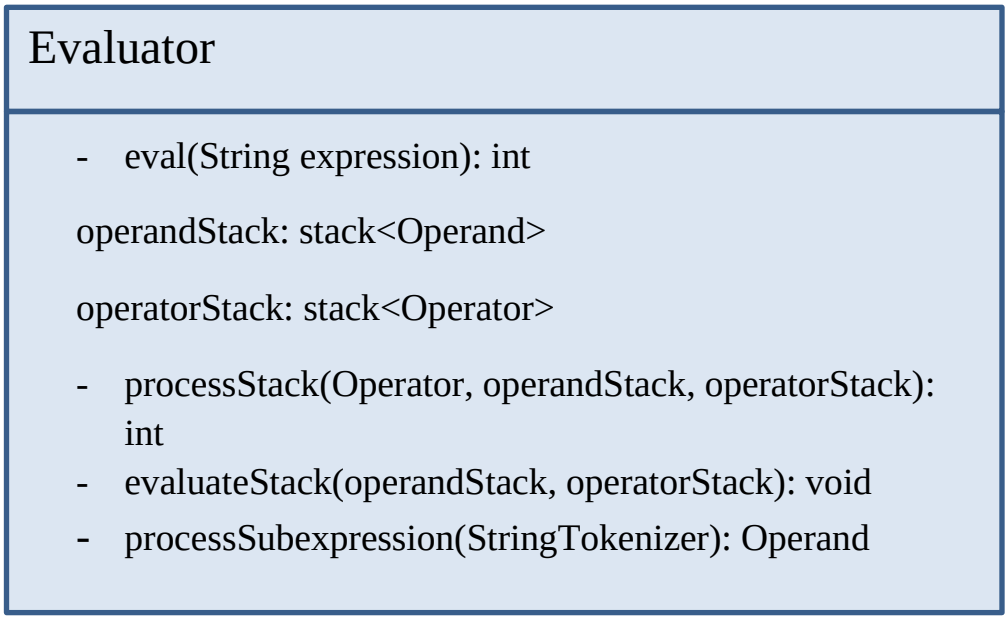

Fig 6.1 The Evaluator class methods

eval(String expression) method:

- In the 'eval()' method, a mathematical expression is received as an argument from the EvaluatorTester class or from the EvaluatorUI class. If the expression is empty, then the error message is displayed on the console.
- Two stacks, the operandStack and the operatorSTack are instantiated in the 'eval()' method.
- A StringTokenizer tokenizes the given expression into tokens.
- A sharp operator (#) is pushed onto the operatorStack. It actually does nothing and it has the lowest priority. This operator is inserted at the beginning of the operator stack and hence there is no need to check if the operator stack is empty.
- The while loop is executed till all the tokens are pushed onto the operatorStack and operandStack. In the while loop, every token is compared in the 'check()' method of the Operator or the Operand class. If the token is operand, then it is pushed onto the operandStack. If the ")" right parentheses occurs before "(" left parentheses in the expression, then an error message is displayed. If "(" left parentheses occurs first, then the 'processSubExpression()' method is called.

processSubExpression() method: In this method, for every left parentheses, a counter is incremented, and for every right parentheses, a counter is decremented. If the counter value is non-zero, then the expression is an invalid expression.

For example: ((2+3)-4 is an invalid expression as the counter value is 1 here.

If the counter value is zero, then the given expression is broken down into a subexpression.

For example: 3+(((2-1)*(3^2))-4)-1 ⟶ (((2-1)*(3^2))-4) ⟶ ((2-1)*(3^2)) ⟶ (2-1)*(3^2) ⟶ (2-1)

⟶ (3^2)

The subexpression is passed to the 'eval(subExpression)' method to evaluate the value of the subexpression. The 'processSubExpression' method calls the 'eval()' method recursively because an expression can have more than sub-expressions. The result of the sub-expression is pushed onto the operandStack.

(**NOTE**: The reason I instantiated the operatorStack and the operandStack into the eval method because the 'processSubExpression()' method treats every sub-expression as a new expression, evaluates it and provides the result of a subexpression (operand) value).

- Back to the 'eval()' method, if the token is neither operand nor any parentheses nor any invalid operator, the token has to be a valid operator. The priority of the newOperator (token) is compared with the priority of the topmost operator of the operatorStack. If the priority of the newOperator is greater, then it is pushed onto the operatorStack. The priority chart for operators is given below:

Operator	Priority
#	0
!	1
+, -	2
*, /	3
^	4

If the priority of the top operator of the operatorStack is greater, then the 'evaluateStack(operandStack, operatorStack)' method is called.

evaluateStack(operandStack, operatorStack): In this method, the topmost operator of the operatorStack and top two operands of the operandStack are popped. The 'execute()' method of the corresponding operator (child operator of the Abstract Operator class) is called and then the result is pushed onto the operandStack.

- Back to 'eval()' method. If any operators or operands are still left on both the stacks, then the Bang operator (!) is pushed onto the operatorStack. The Bang operator does not perform any operation and its priority is less than the mathematical operators. This operator is inserted to indicate that it's the end of an expression. After pushing the Bang operator onto the operatorStack, the 'eval()' method calls the 'processStack()' method to evaluate rest of the operators and operands from the stacks and returns the final result of the expression.

processStack(Operator sharpOperator, operandStack, operatorStack): The Bang operator is popped first from the operatorStack. Then, the 'evaluateStack()' method is called until we get the Sharp operator on the operatorStack (the Sharp operator indicates the beginning of an expression). When we get the Sharp operator on the operatorStack, it means that all the expression is evaluated and the operandStack has the result stored in it. Hence, pop the result from the operandStack and return it to the 'eval()' method. This result is returned to the EvaluatorUI and the EvaluatorTester classes from the 'eval()' method.

Operand Class:

The Operand class is a public class and it has two constructors. One constructor takes String token as an argument while another one takes the double value. The 'check (String token)' method is a public static method, implemented to check if the token is an operand.

Abstract Operator Class:

The Operator class is a public abstract class and it instantiates a HashMap for storing all the operator objects. The 'initializeOperatorMap()' method instantiates all the child operator classes and puts all the operator objects into a HashMap and returns the HashMap. The 'check (String token)' method is a public static method, implemented to check if the

token is a valid Operator. The ‘getOperatorMap(String token)’ method receives a string token and returns the corresponding Operator from a HashMap.

The abstract Operator class has two abstract methods, ‘priority()’ and ‘execute(Operand op1, Operand op2)’.

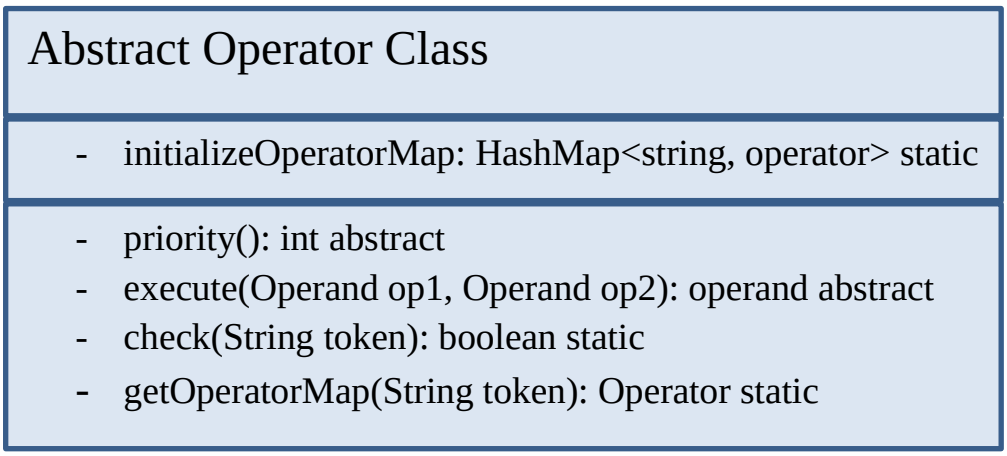

Fig 6.2 The Abstract Operator Class methods

Total nine child operator classes (refer Fig 6.3); five mathematical operators, two pseudo operators, and two parentheses operators; extend the abstract Operator class and implement the abstract methods of the Operator class into every single class.

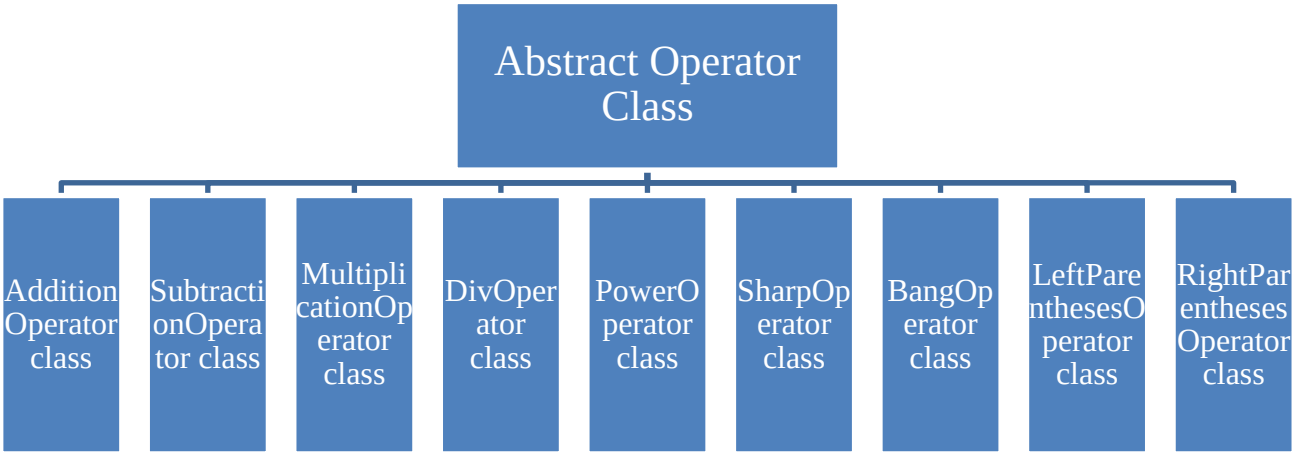

Fig 6.3 Relationship between the Abstract Operator class and its nine child classes

All abstract methods are overridden in nine sub-classes. The ‘priority()’ method sets the priority for every operator and returns it. Parentheses and pseudo operators do not have any priority.

The ‘execute(Operand op1, Operand op2)’ method receives two Operand values from the operandStack and performs the mathematical operation on them, and then, returns the result of the operation. This result is then pushed onto the operandStack by the ‘eval()’ method of the Evaluator class. Two pseudo operators do nothing in the ‘execute()’ method. Hence, if the

pseudo operators are passed into the mathematical expression, then command line or console should display an error message.

Results and Conclusion:

1. Developed a GUI calculator to evaluate a mathematical expression
2. This project is a best practice example to learn Java OOPs concepts such as Inheritance, Polymorphism, Abstraction, and also Java Swing and AWT libraries.
3. Invalid mathematical expressions helped to learn how to handle the exceptions in Java.
4. Future Scope: To develop this project to work with unary operators, float or double values.