

# **Project1\_GUI Calculator & EvaluatorTester**

CSC 413 Fall 2015 Professor Yoon Developer's Guide

February 12, 2016

Version 0.1

Brendan Kelly

## Table of contents

Project Information:	1
Introduction:	3
Scope of Work:	3
Background/Given Resources:	3
Assumptions:	4
How to use the programs:	4
Class Overview:	5
Evaluator class:	6
Operator Classes:	7
Operand Class:	8
Project1 Class:	10
Main Class:	11

## Project Information:

### Introduction:

This is the dev's guide for the CSC 413 Spring 2016 Project 1- GUI Calculator and Command Line Expression Evaluator. This guide has information on how to use the programs and how they work. This will be useful to anyone intending to re-use this code.

These programs both solve expressions using the Evaluator algorithm, however one uses command line arguments and the other uses GUI. There are two JAR files:

1. 413project1a.jar- a command line mathematical expression evaluator.
2. 413project1b.jar- a GUI calculator that works with mouse click input.

This guide will cover both programs. Both of them calculating mathematical expressions. The Evaluator class uses Integer arithmetic meaning division results are truncated and supports (+,-,/,\*,^) and parentheses. For the command line program, expressions are entered in separated by spaces. Evaluator will check if the expression is valid as well.

The GUI Calculator has a basic UI with 20 buttons including all operand/operators as well as C and CE buttons that clear a single button click or a whole expression.

### Scope of Work:

The documentation for this project provided an unfinished Evaluator class and instructions on how to make the Operand and operator classes. It was also a requirement to add the functionality of the power and parentheses operators. For the GUI portion of the project it was necessary to implement the actionPerformed() method.

### Background / Given Resources:

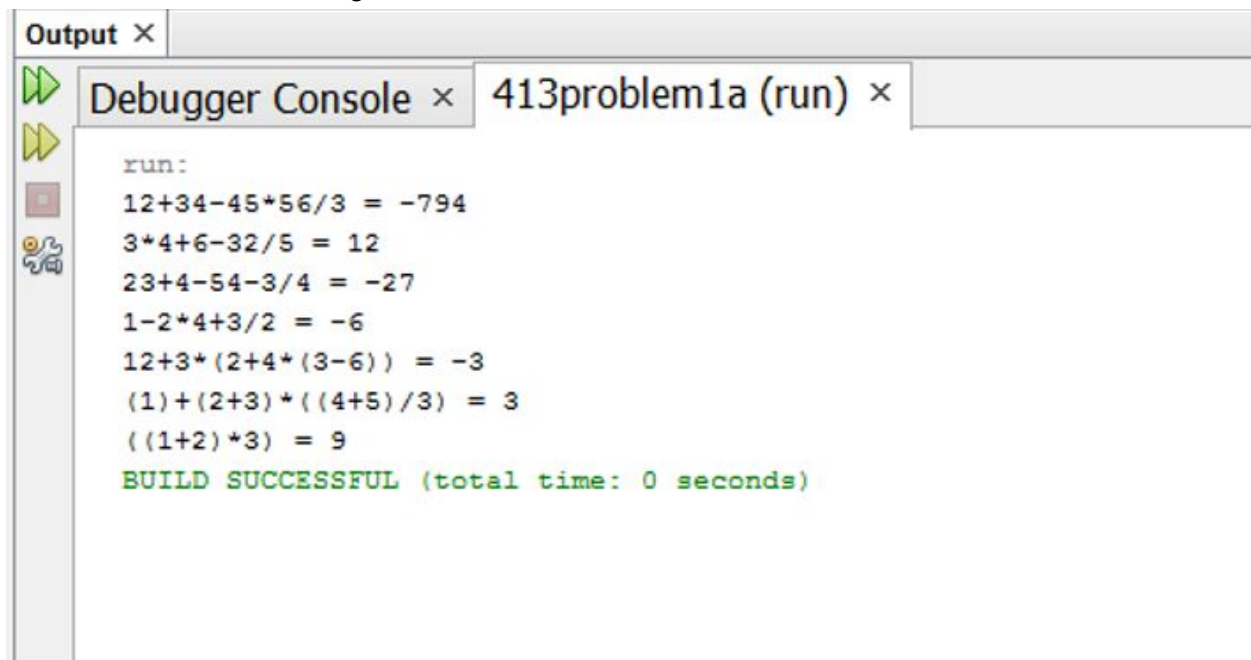
The code and the project description given for this project included an almost completed Evaluator class and instructions on how to complete Operand and Operator classes. The code given to construct the GUI calculator UI was given as well. The actionPerformed() method only needed to be implemented. The EvaluatorTester was provided and contains the main method and is found in Main.java. This project is also for just integer arithmetic. Division results are truncated.

## Assumptions:

This program was created on a Microsoft Surface 3 Intel Core i5-4300U CPU @ 1.9GHz, 4GB RAM running Windows 8.1 64-bit OS. I'm using NetBeans IDE 8.0.2 and the algorithm used was a modified Shunting Yard theorem.

## How to use the programs:

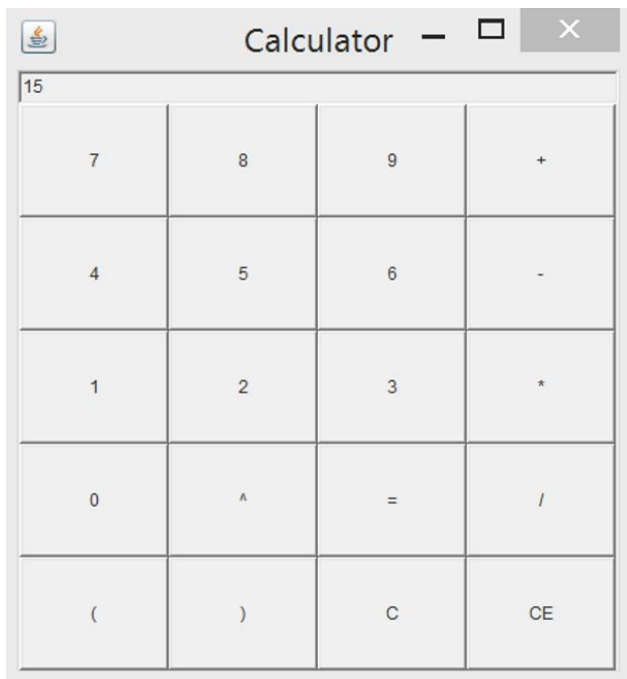
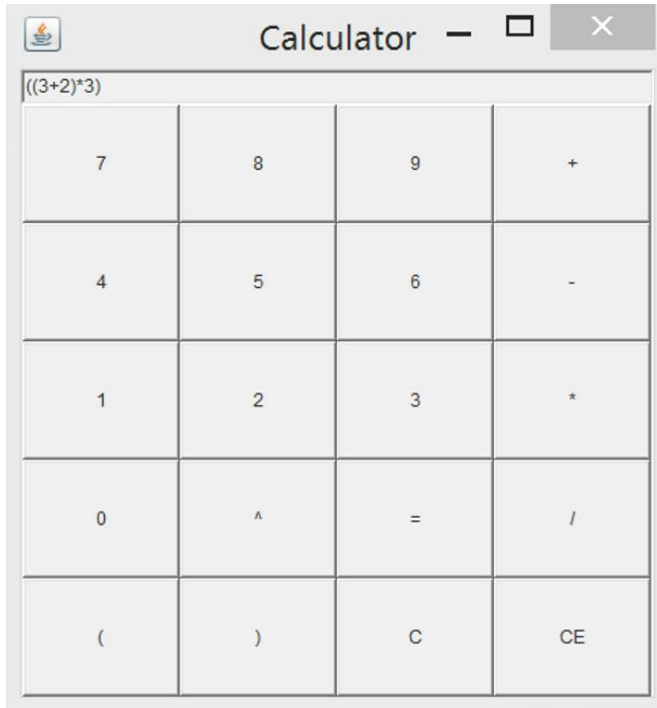
The command line Version, EvaluatorTester.jar can either be run on the command line or through the NetBeans IDE. During my testing of this program I used the project properties on Netbeans to enter in the arguments for the main method.



The screenshot shows the NetBeans IDE's Output window. The window has a tab titled "413problem1a (run)" and a "Debugger Console" icon. The output text is as follows:

```
run:
12+34-45*56/3 = -794
3*4+6-32/5 = 12
23+4-54-3/4 = -27
1-2*4+3/2 = -6
12+3*(2+4*(3-6)) = -3
(1)+(2+3)*((4+5)/3) = 3
((1+2)*3) = 9
BUILD SUCCESSFUL (total time: 0 seconds)
```

For the GUI version when the Project1.java file is executed, a calculator UI with 20 buttons pops up and by entering in the expression using the buttons and then pressing the = button the result shows up in the text area.



## Class Overview:

The `EvaluatorTester` and `Project1_GUI` use the `Evaluator` public class in the `Evaluator.java` file. Also included are the `Operand` class and Each Operator class, including `AdditionOperator.java`, `SubtractionOperator.java`, `MultiplicationOperator.java`, `DivisionOperator.java`, `PoundOperator.java`, `ExclamationOperator.java`, `LeftParOperator.java`, `RightParOperator.java`, and `PowerOperator.java`.

Main.java includes the EvaluatorTester class which creates a new Evaluator object, anEvaluator, and evaluates the expression, printing out the expression and result.

The 413project1b contains a separate java file Project1.java which contains the class Project1. This creates the Calculator UI and implements the ActionListener class and actionPerformed methods to process mouse input .

## Evaluator class:

The evaluator class creates two instances of stack

1. opdStack of Operand objects.
2. oprStack of Operator objects.

The Operator constructor fills an instance of each operator into a hashmap named operators.

The operator priority is given in each of the subclasses and follow the documentation given in class except the out priority of the right parentheses is 1 instead of 2 in order for every operator to be able to be pushed on after the left parenthesis.

## Operator Priority table

Operator	in priority	our priority
!	1	1
#	0	0
+	2	2
-	2	2
*	3	3
/	3	3
^	4	4
(	5	1
)	2	2

The special operators are # and !. These operators are used to get rid of expensive checking of empty stack and to clear the stack respectively.

## Operator Class

```
public abstract class Operator {

    static final HashMap<String, Operator> operators = new HashMap<>();

    static {
        operators.put("+", new AdditionOperator());
        operators.put("-", new SubtractionOperator());
        operators.put("*", new MultiplicationOperator());
        operators.put("/", new DivisionOperator());
        operators.put("^", new PowerOperator());
        operators.put("(", new LeftParOperator());
        operators.put(")", new RightParOperator());
        operators.put("#", new PoundOperator());
        operators.put("!", new ExclamationOperator());
    }

    static boolean check(String tok) {

        if (operators.containsKey(tok)) {
            return true;
        } else {
            return false;
        }
    }

    abstract int getOutPriority();

    abstract int priority();

    abstract Operand execute(Operand opd1, Operand opd2);

}
```

## Special Processing for Parenthesis

In order to take care of the parentheses I had to use a method called `isOperable()` to make sure the algorithm didn't try to evaluate the parentheses operators. This would cause an empty stack exception. By doing this, all operators end up on the stack and the leftover parentheses are cleared out by the `!` operator with a while loop.

```

//taking care of right associativity
if ((oprStack.peek().priority() == 4 && newOpr.priority() == 4) || !(operable(oprStack.peek()))
    break;
} else {
    //popping all parantheses
    if (newOpr == Operator.operators.get("!")) {
        while (!(operable(oprStack.peek()))) {
            oprStack.pop();
        }
    }
}

```

By changing the second while loop to check if the old operator's in and out priority is greater than or equal to the new operator's and by adding the above code to the while loop. I can exit the inner loop to take care of right associativity and get rid of open and close parentheses after all of the operations are finished.

## Code Segments that prepare for Operator Evaluation

The Operator following operator evaluation is unchanged because of the added lines of code above.

```

Operator oldOpr = ((Operator) oprStack.pop());
//check if oldOpr is a parentheses, if so it cannot operate on operands

Operand op2 = (Operand) opdStack.pop();
Operand op1 = (Operand) opdStack.pop();
opdStack.push(oldOpr.execute(op1, op2));

```

## Expression Evaluation and While loops

The following code is the finished Evaluator while loops that calculate each expression



```

oprStack.push(Operator.operators.get("#"));
String delimiters = "+-*/^#!() ";
StringTokenizer st = new StringTokenizer(expr, delimiters, true);
// 3rd arg is true to indicate to use the delimiters as tokens, too
// we'll filter out spaces
while (st.hasMoreTokens()) {
    if (!(tok = st.nextToken()).equals(" ")) { // filter out spaces
        if (Operand.check(tok)) { // check if tok is an operand
            opdStack.push(new Operand(tok));
        } else {
            if (!Operator.check(tok)) {
                System.out.println("*****invalid token*****");
                System.exit(1);
            }
            // Operator newOpr = new Operator(tok); // POINT 1
            Operator newOpr = Operator.operators.get(tok);

            //only for right associated
            while (((Operator) oprStack.peek()).priority() >= newOpr.priority() && ((Operator) oprStack.peek()).getOutPriority() >= newOpr.getOutPriority()) {
                // note that when we eval the expression 1 - 2 we will
                // push the 1 then the 2 and then do the subtraction operation
                // This means that the first number to be popped is the
                // second operand, not the first operand - see the following code

                //taking care of right associativity
                if ((oprStack.peek().priority() == 4 && newOpr.priority() == 4) || !(operable(oprStack.peek()))) {
                    break;
                } else {
                    //popping all parantheses
                    if (newOpr == Operator.operators.get("!")) {
                        while (!(operable(oprStack.peek()))) {
                            oprStack.pop();
                        }
                    }

                    Operator oldOpr = ((Operator) oprStack.pop());
                    //check if oldOpr is a parentheses, if so it cannot operate on operands

                    Operand op2 = (Operand) opdStack.pop();
                    Operand op1 = (Operand) opdStack.pop();
                    opdStack.push(oldOpr.execute(op1, op2));
                }
            }

            oprStack.push(newOpr);
        }
    }
}
oprStack.push(newOpr);
}

```

## The Operand Class:

The Operand class gets constructed using the string constructor, then through that constructor it is converted into an int and then calls the int constructor. This class also contains a check method that makes sure that a valid token is being used.

```

package pkg413problem1a;

/**
 *
 * @author brend_000
 */
public class Operand {

    private int val;

    Operand(String tok) {

        this(Integer.parseInt(tok));
    }

    Operand(int value) {
        val = value;
    }

    static boolean check(String tok) {
        boolean isOp = true;
        try {
            Integer.parseInt(tok);
        } catch (NumberFormatException e) {
            isOp = false;
        }
        return isOp;
    }

    int getValue() {
        return val;
    }

}

```

## Project1.java ActionPerformed Method

```
public void actionPerformed(ActionEvent arg0) { // You need to fill in this fuction

    if (arg0.getSource() == buttons[18]) {
        txField.setText(txField.getText().substring(0, txField.getText().length() - 1));
    } else if (arg0.getSource() == buttons[19]) {
        txField.setText("");
    } else if (arg0.getSource() == buttons[14]) {
        Evaluator e = new Evaluator();
        txField.setText("" + e.eval(txField.getText()));
    } else {
        for (int i = 0; i < buttons.length; i++) {
            if (arg0.getSource() == buttons[i]) {
                txField.setText(txField.getText() + bText[i]);
            }
        }
    }
}
```

In this Method I used a for loop to append the operand and operator buttons and separate if statements to take care of "C", "CE", and "=" Buttons

## Main Class

This class was given and I didn't change it.

```
public class Main {

    public static void main(String[] args) {
        // TODO code application logic here
        Evaluator anEvaluator = new Evaluator();
        for (String arg : args) {
            System.out.println(arg + " = " + anEvaluator.eval(arg));
        }
    }
}
```