Richard Robinson
SID: 917046226
CSC 413 Assignment 1
Due: Feb 14, 2017

Calculator Project

“It was a calculated risk, but I am bad at math.”

GITHUB: https://github.com/SFSU-CSC-413/assignment-1-Renaird

How to Make and Use a Jar:

1) Change to directory with .java files
2) Compile the java files with javac <CLASS_NAME>.java
3) Create a text document in this folder, name it manifest.
   --Add “Main-Class: <YOUR_MAIN_CLASS>” followed by a new line to this file
4) Type “jar cvfm <NAME_FOR_JAR_FILE>.jar manifest.txt *.class”
5) Run with java -jar <NAME_FOR_JAR_FILE>.jar

Note: in step 4 *.class is used, this will include all classes within the folder, for only uses certain classes one can write out the specific names of each class followed by a space instead.

Summary:

The goal of this project was to create a simple calculator with an object-oriented approach. This calculator uses an infix approach and is capable of:

-Multiplication
-Division
-Addition
-Subtraction
-Exponentiation
-Use of Parentheses

An operand stack and an operator stack are utilized. Each operator is an extension of an abstract class “Operator”. A hash table of operators is used to help determine if the characters within a string are a supported operator, an operand, or neither and then creates a corresponding object to be pushed upon the appropriate stack. Each supported operator is given a priority and evaluation corresponds to the given priority of operators. If a right parenthesis is found operations will be executed until the matching left parenthesis is found and then the left will be popped off the stack. A “#” operator is used to indicate that the stack is empty.

The basic evaluation operation pops two operands and an operator from their respective stacks and executes the evaluation of those components. The result is then pushed upon the operand stack.

This calculator works as intended and all valid inputs work.
The calculator works with integers, so rounding may occur upon division.

My project differs from the template in a few ways:

> *I have a executeOnce() method within my operator class which pops two operands and an operator and executes the evaluation of those components. The result is then pushed to the operand stack.*
>
> *The GUI is much different than the template*.
>
> op1 and op2 were renamed operandLeft and operandRight for clarity

Usage:
The EvaluatorUI is the GUI component to be used and the one I would recommend using. Alternatively, arguments can be passed to the EvaluatorTester class.

The calculator works using integer values.
This calculator has no sign operator, subtraction must be used to yield negative values.

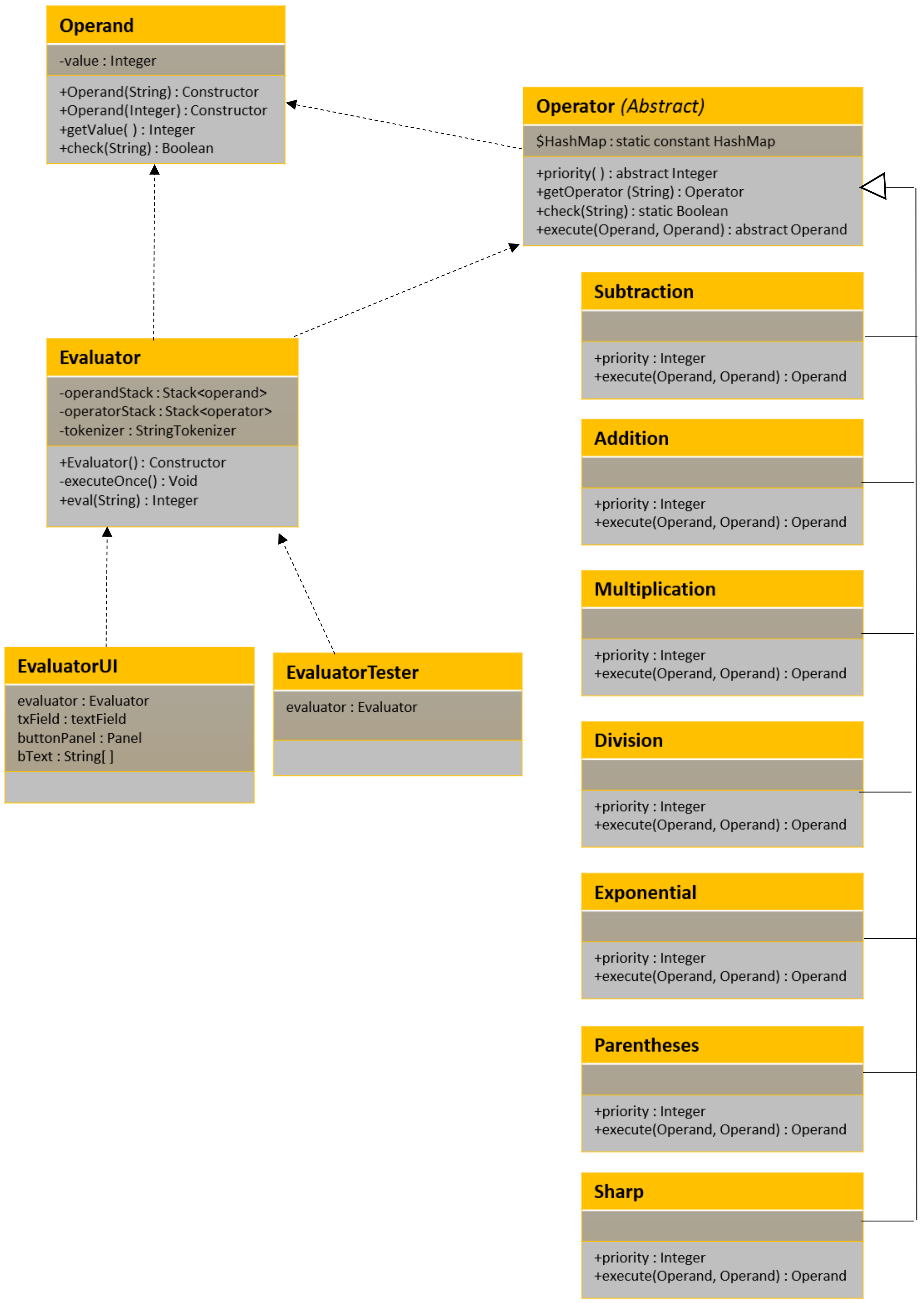
*It is assumed that the user will use the program correctly and this program was not designed to handle user error.

Discussion:
There was nothing astonishingly difficult about this project, but the object-oriented approach implementing a hash table and various operand classes was a novelty to me and made me think of new ways of approaching problems. I will extend this concept of doing instead of asking “if” in the future and am extremely appreciative of this project for that reason.

My code is by no means optimized and does not error check for invalid arguments or syntax. I had a very positive experience with this project and will be capable of looking at future projects in a more object oriented fashion.

This could be made to check for errors and display an invalid input message through checking that a right parenthesis does not come before a left and that there is a right match for every left, and by checking if two operators are used in a row. It could be expanded upon by allowing the use of negative numbers through a use of a sign operator.

This project was extremely enjoyable

Operand
-value : Integer
+Operand(String) : Constructor
+Operand(Integer) : Constructor
+getValue( ) : Integer
+check(String) : Boolean
Operator (Abstract)
$HashMap : static constant HashMap
+priority( ) : abstract Integer
+getOperator (String) : Operator
+check(String) : static Boolean
+execute(Operand, Operand) : abstract Operand
Evaluator
-operandStack : Stack<operand>
-operatorStack : Stack<operator>
-tokenizer : StringTokenizer
+Evaluator() : Constructor
-executeOnce() : Void
+eval(String) : Integer
EvaluatorUI
evaluator : Evaluator
txField : textField
buttonPanel : Panel
bText : String[ ]
EvaluatorTester
evaluator : Evaluator
Subtraction
+priority : Integer
+execute(Operand, Operand) : Operand
Addition
+priority : Integer
+execute(Operand, Operand) : Operand
Multiplication
+priority : Integer
+execute(Operand, Operand) : Operand
Division
+priority : Integer
+execute(Operand, Operand) : Operand
Exponential
+priority : Integer
+execute(Operand, Operand) : Operand
Parentheses
+priority : Integer
+execute(Operand, Operand) : Operand
Sharp
+priority : Integer
+execute(Operand, Operand) : Operand

Ex1

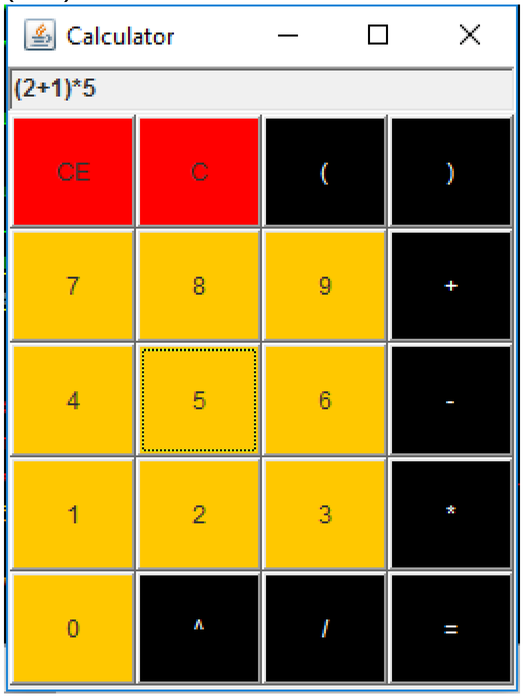
```
run:
(2-1)*5 = 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

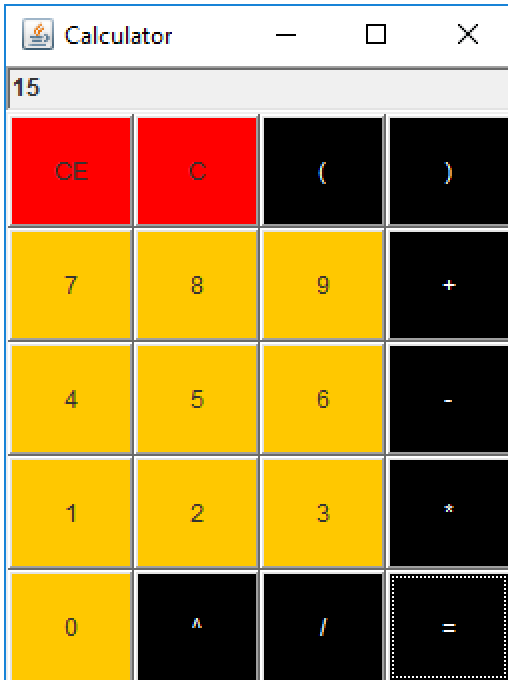
Ex2

```
run:
7^2+1 = 50
BUILD SUCCESSFUL (total time: 0 seconds)
```

EX. 3

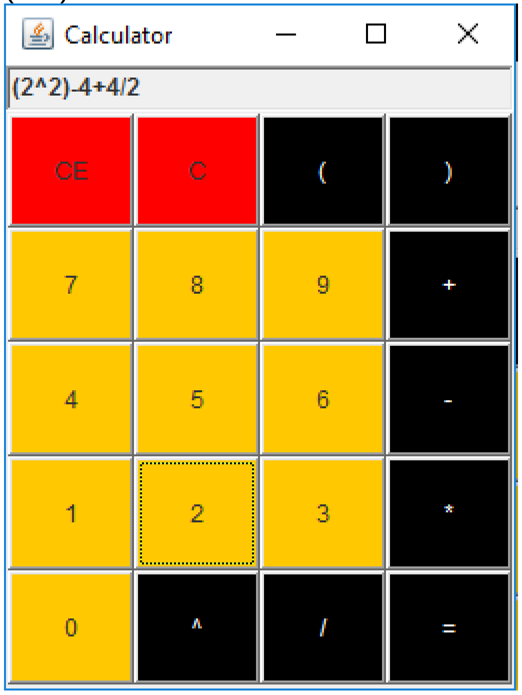
(2+1)*5 ---------------------------- > Evaluated

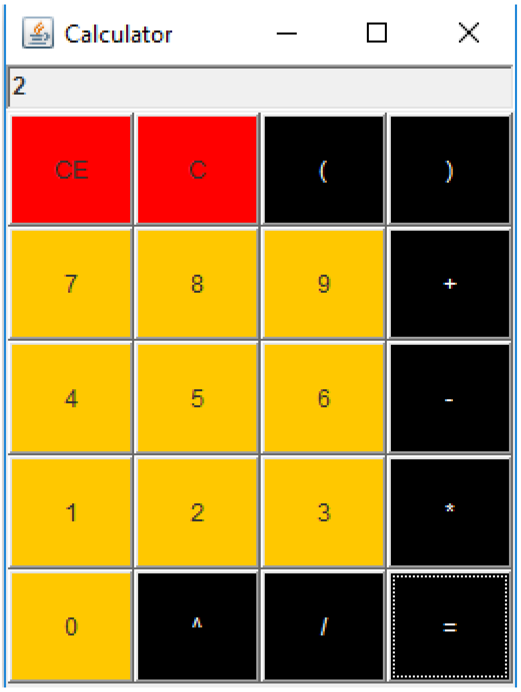

(2+1)*5


15

EX. 4

(2^2)-4+4/2 -------------------------------------- > Evaluated


(2^2)-4+4/2


2