Georgia State University
Department of Computer Science
CSC 2720 **J. L. Bhola**
Fall 2017
Programming Assignment 5
Due October 31st, 2017 at 11:00 p.m. to be uploaded to Folder #5 in iCollege.

Objectives:

To gain experience with the Stack Data Structure and the library - so use as many generic algorithms as possible.

Documentation:

1. Explain the purpose of the program as detail as possible - **8%.**
2. Develop a solution for the problem and mention algorithms to be used **-12%**
3. List data structures to be used in solution. - **5%.**
4. Give a description of how to use the program and expected input/output - **5%**
5. Explain the purpose of each class you develop in the program. - **5%.**

Programming:

1. For each method, give the pre and post conditions and invariant, if any - **10%**
2. Program execution **according to the requirements given 50%**
3. Naming of program as required **5%**

Description of Program

You are to write a program name **calc.java** that evaluates an infix expression entered by the user. The expression may contain the following tokens:
(1) Integer constants (a series of decimal digits).
(2) x (representing a value to be supplied later).
(3) Binary operators (+, -, *, / and %).
(4) Parentheses

Spaces between tokens are allowed but not required. The program will convert the expression to postfix (RPN) form and display the converted expression. The program will must also solve the postfix expression producing a final answer. The must run again and again repeatedly prompting the user for the value of **x**, displaying the value of the expression each time. When the user enters the letter **q** instead of a number, the program terminates.

The following example illustrates the behavior of the program (user input is in **bold and red**):
Porgram output is in **bold and green.**

```
Enter infix expression: (x + 1) * (x – 2) / 4
Converted expression: x 1 + x 2 - * 4 /

Enter value of x: 5
Answer to expression: 4

Enter value of x: 7
Answer to expression: 10

Enter value of x: q
```

If the infix expression contains an error of any kind, the program must display the message Error in expression (with an optional explanation) and then terminate. The following examples illustrate various types of errors:

```
Enter infix expression: 1 2 +
Error in expression!! No operator between operands. Also last token must be an operand.

Enter infix expression: 10.4
Error in expression!! Cannot accept floating point numbers.

Enter infix expression: 1 (  + 2)
Error in expression!! No operator between operand and left parentheses.

Enter infix expression: 5 – (x – 2))
Error in expression!! No matching left parentheses for a right parentheses.

Enter infix expression: 1 ** 2
Error in expression!! The * operator cannot be preceded by a * operator.
```

The output of your program must match the format illustrated in this example.

Here are some other additional requirements for this program:

(1) You must use stack objects during the translation from infix to postfix and during the evaluation of the postfix expression.

(2) Operators must have the correct precedence and associativity.

What to turn in:

-

1. All of the .java and the .class/jar files at the iCollege (D2L) website in the Dropbox folder for A5 no later than 11:00 p.m. on the due date.

<u>Hints:</u>

1. Do the program in stages. First, get the infix to postfix conversion working for binary operators. Next, implement the evaluation of the postfix expression. Finally, add code to check the infix expression for errors. Use the examples from the hand out as a starting point for the program, but keep in mind that this code does not handle associativity properly.

2. To detect errors in the infix expression, you will need to check for several situations:

A binary operator is preceded by an operator or an operand is preceded by an operand.
An illegal character is encountered (such as a period).
The last token in the expression is not an operand or a right parentheses
There is no left parentheses anywhere in the stack when a right parentheses is encountered.
The stack contains a left parenthesis when the expression ends.

3. Use a string to store the postfix expression. Use a stack of characters during the translation from infix to postfix.

4. Use the Scanner class to read the input string and extract each token. Careful when you are reading an unknown (eg. x) from the input expression.