TCSS143
Fundamentals of Object-Oriented Programming-Theory and Application
Programming Assignment 4

DUE: October 31, 2013 by 12 midnight

The purpose of this programming project is to demonstrate the functionality of Stack structure.

REQUIREMENTS

You will submit a single source code file "**Calculator.java**" through the Programming Assignment 4 Submission link on Moodle.

Not only will you be graded on program correctness (Program executes correctly, proper use of methods, classes, inheritance, etc.) but also, good programming documentation techniques including proper indentation, correct locations of braces, meaningful identifier names, general comments prior to methods, specific comments on complex code, etc. If you are unsure (this should have been learned in 142), look inside the useful links folder in Moodle for links on these topics. Programs that fail to compile or execute will receive no credit.

DETAILS

You will create a simple simulated Calculator class that evaluates simple mathematical expressions represented in a given String. The String may contain any of the following tokens:

> Numeric values (double)
> Single characters: +, -, *, / ; for addition, subtraction, multiplication, and division, (,) left/right parentheses.
>
> For Example: (3 + (16 - (4 * 6)) + (12 - 4) + 7)
> Note: every item (or token) is surrounded by spaces.

Your calculator will use a Stack to process the expressions:

> When a number is read, insert it into an element (Element is described below) and push it on the stack.
> When an operator (+, -, *, /) is read, insert it into an element and push it on the stack.
> When a right parenthesis is read, you will pop off 3 elements, placing the first into a second operand, the second into an operator char (or String if you want to do it that way), and the third into a first operand. You will then evaluate the expression result = firstOperand operator secondOperand and place the result into an element and push it onto the stack. Continue until you have consumed all tokens in the String.
>
> In the event that there were not enough right parentheses to leave a single final result in the stack following the above process, you will now have to complete the expression remaining in the Stack by popping 3 elements, evaluating the expression, placing the result into an element, and pushing it onto the stack. Continue until only 1 element remains on the stack (this should be your final result which should be returned to the calling method).

As you can see, this is NOT a post-fix calculator. Had we taken that path (and it is much better) your code would have to interpolate regular expressions (in-fix) into the post-fix versions. Our calculator will not be as complex. Specific details about each Calculator method and fields begins on page 2.

Because we are pushing/popping both doubles and char's on the same Stack, we have to use an Object which will contain both and perform simple operations on the fields (get, set, etc.). This Object is so tightly bound to our Calculator class that we will include it at the bottom of the Calculator class file. The header for both is illustrated on the next page (note the Element class heading does not have the word public!)

```
import java.util.*;

public class Calculator{
  Element el;             // data to be pushed/popped on the stack
  Stack<Element> stack;   // The stack to hold the elements
  String expression;      // The expression String which is passed to
                          // method evaluate from the calling program
  double result;          // holds the final result

  public Calculator(){
  ...
} // End of class Calculator

class Element{
  double operand;   // Holds numberic values to be pushed on the stack
  char operator;    // Holds operator symbols to be pushed on the stack
  public Element(){
    operand = 0.0;
    operator = ' ';
  }
  ...
} // End of class Element
```

1. Calculator.java

public class Calculator {

Fields: An Element, Stack, String, double: Described above as comments in the code

Methods:
public Calculator ( )
Should set all fields to some stable state. This is your choice.

public double evaluate(String expression)
Receives an expression String as an argument and Returns the final result of the expression.

Most everything is done here (my version is 32 lines). You will want to use a Scanner to process all tokens in the expression String by placing doubles and operators into individual Elements and pushing them onto the stack. As described on page 1, you will be popping elements off when ‘)’ is encountered and pushing the result (in an element) back onto the stack.

When all tokens have been processed, you should drop out of the loop and enter another loop which will process any remaining expression left on the stack until a final result can be popped off.

The final result is returned to the calling program.

public double do3Pops( )
Pops 3 elements off the stack. The first will contain operand2, the second contains the operator, and the last contains operand1. do3Pops will then evaluate the expression and return the result to the calling method. This should be called in both the expression String processing loop and also in the loop that performs the final evaluation of expressions remaining on the stack.

You may create any other helper methods that you feel could reduce complexity or redundancy.

2. class Element

As mentioned above, you will create this class following the Calculator class at the bottom of the same file.
class Element {

Fields: see the lower section of code at the top of page 2.

Methods:
public Element( )
Implement as you see fit. You might also want to create a parameterized version for the expression String but, is not at all required.

Accessor/Mutator methods:
You will want to create one get method for each field (2 in all: “getOperator” and “getOperand”) and one set method for each field (2 in all: “setOperator” and “setOperand”).

Save this file under the name Calculator.java and upload it in Moodle through the Programming Assignment link.

Zipped together with these instructions are a driver program “Calc_Test.java” and an input file: “In4.text” .

All input expressions in the sample input file “In4.txt” are sans errors with each token surrounded by spaces.

That should take care of it. Of course, as already mentioned, add any helper methods you feel could be useful.

Sample input/output files:

In4.txt:

```
( 4 * 5 ) + 5 / 2.5
( 3 + ( 16 - ( 4 * 6 ) ) + ( 12 - 4 ) + 7 )
( 5 * ( 3 * ( 7 - 10 ) ) ) / (  2 +  ( 3 * 4 ) - 2 )
( ( 19.68 / 2.4 ) - ( 3  + ( 52.5 * -10 ) )
3 + 7 + ( 42.8 - ( ( ( 4 * 8 ) / 2 ) + 0.2 ) ) – 3
```

out4.txt:

```
( 4 * 5 ) + 5 / 2.5 = 22.0000
( 3 + ( 16 - ( 4 * 6 ) ) + ( 12 - 4 ) + 7 ) = 10.0000
( 5 * ( 3 * ( 7 - 10 ) ) ) / (  2 +  ( 3 * 4 ) - 2 ) = -3.7500
( ( 19.68 / 2.4 ) - ( 3  + ( 52.5 * -10 ) ) = 530.2000
3 + 7 + ( 42.8 - ( ( ( 4 * 8 ) / 2 ) + 0.2 ) ) - 3 = 33.6000
```

Good Luck and get started early.