Tunbutr, Bryant

CSCI 220
Data Structures I

Lab Project #2

INFIX TO POSTFIX CALCULATOR

Due Date
10/15/2013
Date Turned In
10/15/2013

Student Name:	___________ Bryant Tunbutr ____________________	Project Number:	_______1________
Project Name:	_______Calculator__________	Eclipse Version:	___Kepler________

Files: Calculator.java

Project specification

This software is intended to do calculations using stacks and operations including push, pop, peek.

The user inputs an expression, which gets converted to postfix, then evaluated. Both steps use stacks.

Program is complete except for some reason the postfix not 100%. The infix to postfix seems perfect, but the postfix evaluation gives correct results most of the time, but not all of the time.

I think I learned many lessons:

It is very important to work on a big project like this one step at a time, i.e. first make sure the program can just take in the numbers, then display, then get it to post fix.

Write notes about what each method does.

Label every variable as clearly as possible to avoid confusion and work better.

Printout of program input/output

```
/*  Program: Calculator
    Author: Bryant Tunbutr
    Class: Data Structures
    Date: 10/15/13
    Description: Infix and Postfix Calculator

    I certify that the code below is my own work.

   Exception(s): N/A

*/


public class CalculatorTest
{
   public static void main(String[] args){

      System.out.println("Author:  Bryant Tunbutr");
      System.out.println();
      System.out.println();

      // new object
      Calculator calc2 = new Calculator();

      // test cases

      String infix1 = "3+3";
      System.out.println("Infix: " + infix1);
      String postfix1 = calc2.infixToPostfixString(infix1);
      System.out.println("Postfix: " + postfix1);
      System.out.print("Result: " );
      calc2.postfixToOutputLong(postfix1);
      System.out.println();


      String infix2 = "(5-4)+4";
      System.out.println("Infix: " + infix2);
      String postfix2 = calc2.infixToPostfixString(infix2);
      System.out.println("Postfix: " + postfix2);
      System.out.print("Result: " );
      calc2.postfixToOutputLong(postfix1);
      System.out.println();
```

```java
        String infix3 = "(2-1+4)";
        System.out.println("Infix: " + infix3);
        String postfix3 = calc2.infixToPostfixString(infix3);
        System.out.println("Postfix: " + postfix3);
        System.out.print("Result: " );
        calc2.postfixToOutputLong(postfix1);
        System.out.println();


        String infix4 = "((5*4/2)+(4-3+4)-(6/3))*2";
        System.out.println("Infix: " + infix4);
        String postfix4 = calc2.infixToPostfixString(infix4);
        System.out.println("Postfix: " + postfix4);
        System.out.print("Result: " );
        calc2.postfixToOutputLong(postfix1);
        System.out.println();
    }
}
```

```
Author:  Bryant Tunbutr


Infix: 3+3
Postfix: 33+
Result: 6

Infix: (5-4)+4
Postfix: 54-4+
Result: 6

Infix: (2-1+4)
Postfix: 21-4+
Result: 6

Infix: ((5*4/2)+(4-3+4)-(6/3))*2
Postfix: 54*2/43-4++63/-2*
Result: 6
```

Typed in answers from main class

```
Enter your Infix or '0' to exit 1+3
Infix: 1+3
Postfix: 13+
Result: 4

5*2
Infix: 5*2
Postfix: 52*
Result: 10
```

```
Correct postfix but wrong answer below

(3*5)/(6/2)
Infix: (3*5)/(6/2)
Postfix: 35*62//
Result: 0




5*9
Infix: 5*9
Postfix: 59*
Result: 45
```

Source code

Calculator.java

```java
/*  Program: Calculator
    Author: Bryant Tunbutr
    Class: Data Structures
    Date: 10/15/13
    Description: Infix and Postfix Calculator

    I certify that the code below is my own work.

   Exception(s): N/A

*/

// import stack to be able to use push, pop, peek, etc.
import java.io.BufferedReader;
import java.util.Stack;

// import scanner to gather user input from keyboard
import java.util.Scanner;
import java.util.StringTokenizer;


public class Calculator
{
   // this calculator program works by using operands and operators
   // therefore, the first step will be to
   // differentiate between the operands and operators

   // these are the constant operators used in this program
   private static final String OPERATOR_STRING = "-+/*";

   // these are the constant operands used in this program
   private static final String OPERAND_STRING = "0123456789";

   // calculations are made in order of precedence, PEMDAS
   private int operatorPrecedenceInt(char operatorChar)
   {
       switch (operatorChar)
       {
       // these have lower precedence
       case '-':
        case '+':
            return 1;

        // these have higher precedence
        case '*':
        case '/':
            return 2;
        }
        return -1;
    }

    // check for operands
    // this checks whether user entered a valid operand by comparing to the
constants/numbers
    private boolean isOperandBool(char inputChar) {
        return OPERAND_STRING.indexOf(inputChar) >= 0;
    }

    // check for operators
    // checks whether the user input matches one of the stored operators
    private boolean isOperatorBool(char inputChar) {
        return OPERATOR_STRING.indexOf(inputChar) >= 0;
    }

    // this ensures that higher order operations are performed first
    private boolean higherPrecedenceOperatorBool(char operatorChar1, char
operatorChar2) {
        return operatorPrecedenceInt(operatorChar1) >=
operatorPrecedenceInt(operatorChar2);
    }

    // this method converts user input/infix into a postfix string
    public String infixToPostfixString(String userEnteredInfixString) {

        // first take the user entered input as a string
        // convert it to a char because this makes it possible to run methods
        // which check if the char is an operand, operator, operator precedence

        char[] userEnteredChar = userEnteredInfixString.toCharArray();

        // store the char in a stack
        Stack<Character> stackChar = new Stack<Character>();

        // use StringBuilder to create the post fix string
        StringBuilder postFixString = new
StringBuilder(userEnteredInfixString.length());

        // loop through the length of the user entry
        for (int i = 0, n = userEnteredChar.length; i < n; i++)
        {
            // evaluate each char
```

```
         char currentUserEnteredChar = userEnteredChar[i];

         // operator section
         // if char is an operator
        if (isOperatorBool(currentUserEnteredChar)) {

           // while the stack is not empty & the next char on the stack is not an
open (
           while (!stackChar.isEmpty() && stackChar.peek() != '(') {

              // if the top of the stack has a higher precedence than current char
              if (higherPrecedenceOperatorBool(stackChar.peek(),
currentUserEnteredChar)) {

                 // pop that top stack item & add it to the postfix string
                 postFixString.append(stackChar.pop());
              } else {
                 break;
              }
           }

           // if current char has lower precedence than top of stack push current
char
           // to top of stack
           stackChar.push(currentUserEnteredChar);
        }

        // parentheses section
        // push ( because it always has highest precedence
        else if (currentUserEnteredChar == '(')
        {
           stackChar.push(currentUserEnteredChar);
        }

        // if it is a closing parentheses
        else if (currentUserEnteredChar == ')')
        {
           // pop it from stack and output it
           // until an open parentheses (
           while (!stackChar.isEmpty() && stackChar.peek() != '(')
           {
              postFixString.append(stackChar.pop());
           }

           // pop all operators from the stack
           if (!stackChar.isEmpty()) {
              stackChar.pop();
           }
        }

        // operand section
        // if it is an operand
        else if (isOperandBool(currentUserEnteredChar))
        {
          // add it to output of postfix string
          postFixString.append(currentUserEnteredChar);
        }
      }
      // if stack has items, pop them and add  to postfix string
      while (!stackChar.empty()) {
        postFixString.append(stackChar.pop());
      }
      // return postfix string
      return postFixString.toString();
    }

    // this method calls the infix to postfix result so it can be used
    // for other methods, i.e postfix output to string
    public String getInfixToPostfixString(String infixString) {
      return infixToPostfixString(infixString);
    }

    // this method converts postfix into the evaluated expression output int
    public void postfixToOutputLong(String postfixString)
    {
      // create a new stack. It stores operands that still need
      // to be evaluated
      Stack<Long> postFixLong = new Stack<Long>();

      for (int i = 0; i < postfixString.length(); ++i) {
        // getting the next character of the string
        char currentPostFixChar = postfixString.charAt(i);

        // variables to store the operands are declared outside
        // of if/else for easier reuse
        long leftLong = 0, rightLong = 0;

           // check if the char is a digit
        if (Character.isDigit(currentPostFixChar)) {
              postFixLong.push((long) (currentPostFixChar - '0'));
        }
        else {
          // pop top 2 and perform operation
          leftLong = postFixLong.pop();
          rightLong = postFixLong.pop();

          // store operation result here
          long resultLong;

          // switch statement for operation
           switch(currentPostFixChar) {
                case '+':  resultLong = leftLong + rightLong;  break;
                case '-':  resultLong = leftLong - rightLong;  break;
                case '*':  resultLong = leftLong * rightLong;  break;
                case '/':  resultLong = leftLong / rightLong;  break;
                default:   resultLong = (long) Math.pow(leftLong,rightLong);

            }
            // push result
                postFixLong.push(resultLong);
        }
    }

    // print the result
    System.out.println(postFixLong.pop());
}

public static void main(String[] args)
{
    // store user input as string
     String userInputString;

     // value to exit loop
    final String SENTINEL_STRING = "0";

     // use scanner
    Scanner userInput = new Scanner( System.in );

    // tell user to enter operation
    System.out.print("Enter your Infix or '0' to exit ");

    // set string to user input
    userInputString = userInput.next();

    // run loop while not equal to sentinel value of 0
      while(!userInputString.equals(SENTINEL_STRING))
      {
      // new object
      Calculator calc1 = new Calculator();

      // print info and results
      System.out.println("Infix: " + userInputString);
      String postfix1 = calc1.infixToPostfixString(userInputString);
      System.out.println("Postfix: " + postfix1);
      System.out.print("Result: " );

      calc1.postfixToOutputLong(postfix1);

      // get next input from user
      userInputString = userInput.next();
      }
}
```

```
}
```