Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 4 – Building a Simple Calculator

Problem Description

In the previous assignment, you wrote a Java program that prompted the user for an infix expression and converted the result to postfix, displaying the result in the console window. In this assignment we will i) build a simple interpreter that takes the postfix result as input and returns the numerical value of the expression as output, and ii) build a simple Graphical User Interface (GUI) that functions as a simple calculator.

Part I: The Postfix Interpreter

Consider the following example: 5*(3+4)/2

The corresponding Postfix expression is: 5 3 4 + * 2 /

Assume that the Postfix interpreter places the resulting expression in a queue, from left to right (in the example above, 5 is at the front of the queue and / is at the rear). The following algorithm can then be used to compute the value of this expression:

Algorthim PostEval
Input – queue hlding the postfix expression
Output – value of the postfix expression
Data - a stack to hold intermediate expressions

Steps
1. dequeue a token from the postfix queue
2. if token is an operand, push onto stack
3. if token is an operator, pop operands off stack (2 for binary operator); push result onto stack
4. repeat until queue is empty
5. item remaining in stack is final result

With above example

Token	Stack
5	5
3	5 3
4	5 3 4
+	5 7
*	35
2	35 2
/	17.5

The postfix expression is made up of elements that are strings, so when it comes to interpreting strings as values some conversion will be necessary. Assume that we are on the third line of the above example; the stack contains 5 3 4 and a + operator is dequeued from the input queue. Consider the following snippet of Java code:

```
Double result;
String token = PostFix.dequeue();
String OP_A = stack.pop();
String OP_B = stack.pop();
Double A = Double.parseDouble(OP_A);
Double B = Double.parseDouble(OP_B);

switch (token) {
case "+":
      result = B+A;
      break;
etc…
}

stack.push(Double.toString(result));
```

Operators and operators are Strings, so when it comes to evaluating an expression, the string containing the operator has to be converted to a numerical value. For this assignment it is best to do all calculations using type Double. The parseDouble method takes a string as input and returns its value expressed as type Double. Similarly, the toString method is used to convert a Double to a string. Notice how these steps are performed above.

Implementation:

It is generally a good idea to separate the GUI from the processing elements of a program. For this reason create a class called postFix() which handles all processing with respect to Infix conversion and Postfix evaluation. It should export/include the following methods:

public double doExpression(Queue Qin) where Qin is the queue containing the Infix expression to be evaluated. It returns the numerical value of the expression as type double.

private double PostEval(Queue PostFix) where PostFix is a queue containing the PostFix expression to be evaluated. PostEval() is called by doExpression which is responsible for Infix to Postfix conversion.

public void parse(String arg, Queue Qin) takes two arguments, a String containing an Infix expression and a Queue into which each token in the string is written, from left to right. This is subsequently interpreted by doExpression.

Testing:

The following test program can be used to make sure your implementation of the postFix class works as specified:

```
public class JCalc extends ConsoleProgram {

// Create program objects

	Queue Qin = new Queue();		// Queue for input expressions
	postFix pf = new postFix();		// Postfix converter and interpreter

// Entry point

	public void run() {
		println("Infix expression evaluator, enter expression of blank line to exit.");

		while (true) {
			String input = readLine("expr: ");		// Get expression
			if (input.equals("")) break;			// Terminate on blank line
			pf.parse(input, Qin);				// Parse
			double result = pf.doExpression(Qin);		// Evaluate
			println(input+" = "+result);			// Display result

		}
		println("Program terminated.");
	}
}
```

Examples:

```
Infix expression evaluator, enter expression of blank line to exit.
expr: 3*5/2*(6.32-4.85)
3*5/2*(6.32-4.85) = 11.025000000000006
expr: 355/113
355/113 = 3.1415929203539825
expr: 1+2+3+4+5
1+2+3+4+5 = 15.0
expr: 1/2+1/4+1/8+1/16+1/32+1/64
1/2+1/4+1/8+1/16+1/32+1/64 = 0.984375
expr: 1.00000001/2*2
1.00000001/2*2 = 1.00000001
expr:
Program terminated.
```

Part II: A Simple Calculator Applet

In the Jcalc example, most of the heavy lifting is done by the postFix class. From here it is pretty straightforward to build a simulation of a physical 4-function calculator using the same techniques as in the TemperatureConverter example in the notes. The minimum specification is shown below in Figure 1.

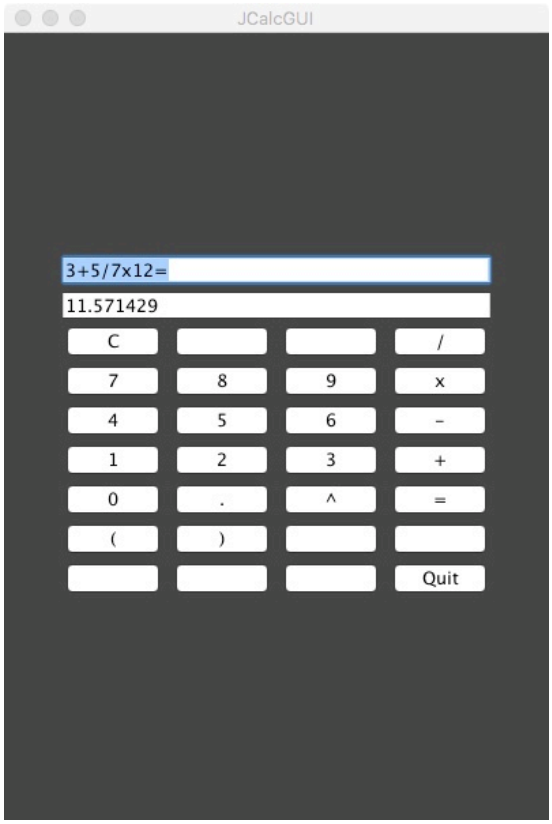

Figure 1

Unlike the more familiar calculator where an expression is evaluated as it is entered, in this version the user enters an Infix expression in the input box using using either the calculator buttons or the keyboard. When the user types enter on the keyboard or presses the = key, the expression is evaluated and the numerical result displayed in the lower display as shown in the figure. Your calculator must implement the 4 basic functions and support parentheses at minimum. However, you are encouraged to extend your interpreter to support unary operators, e.g. -5, and other functions.

The TemperatureConverter example has all the basic structures you will need for this implementation.

Instructions

1. Implement the postFix class and use it in conjunction with the Jcalc example above to replicate the examples shown earlier. Save these results in a file called JCalc.txt or JCalc.pdf depending on your screen capture mechanism.
2. Implement a class called JCalcGUI which implements your calculator applet. We will build and run this application from your source code and verify that it works correctly. It is suggested that you use the ACM classes for implementing the GUI. If you choose not to do so, make sure that your source code will build correctly using the standard Java configuration.
3. Your submission should consist (at minimum) of the following files:

   Stack.java
   Queue.java
   listNode.java
   JCalc.java
   JCalcGUI.java
   JCalc.txt or JCalc.pdf

   Upload your files to myCourses as indicated – include all the source code necessary to build your applications.

fpf/March 5, 2018