Programming Project #3

CIS 296 – University of Michigan – Dearborn
Prof. John P. Baugh

Points: _____ / 125
Due Date: November 11, 2020 at 11:59 p.m.

Objectives

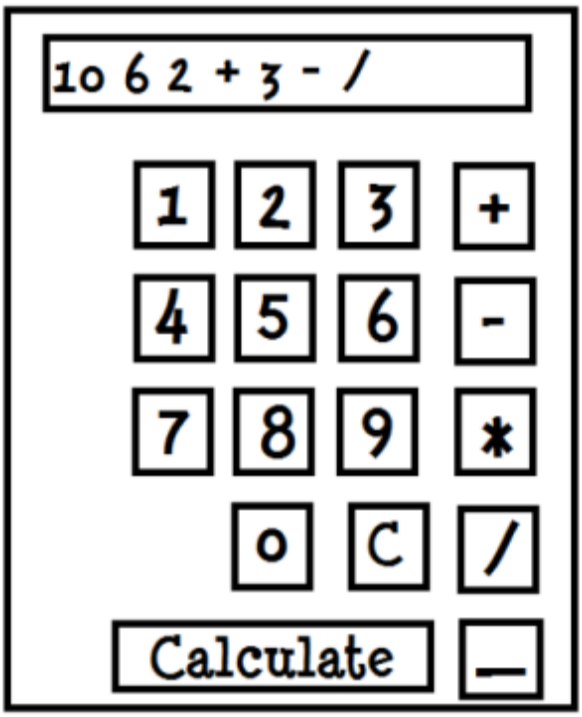
- To understand GUI applications better
- To understand the Stack ADT better
- To implement a program that performs mathematical computations

Instructions

For this assignment, you are going to implement a postfix calculator using a Stack. The application must be a GUI application, allowing the user to enter operands (the numbers) and operators as a string into the display window. When the user clicks the "Calculate" button, the application will attempt to perform the calculation. If it cannot, an appropriate error should be displayed. Examples of exceptions would be a division by zero attempt, or an incorrectly formed postfix string for an equation.

The user can only enter numbers using the buttons on the calculator. He/she may not use the keyboard to enter the numbers or operators.

The calculator's mock-up interface is as follows:



The calculator allows the user to

- Enter numbers (0 … 9)
- Enter operators (+, -, *, /)
- Calculate the current problem on the screen display
- Clear the display (with the C button)
- Enter a space (_) between numbers

How does Postfix notation work?

In postfix notation the operators follow their operands; for instance, to add 3 and 4, one would write "3 4 +" rather than "3 + 4". If there are multiple operations, the operator is given immediately after its second operand; so the expression written "3 − 4 + 5" in conventional notation would be written "3 4 − 5 +" in postfix: first subtract 4 from 3, then add 5 to that. An advantage of postfix is that it eliminates the need for parentheses that are required by infix.

While "3 − 4 * 5" can also be written "3 − (4 * 5)", that means something quite different from "(3 − 4) * 5". In postfix, the former could be written "3 4 5 * −", which unambiguously means "3 (4 5 *) −" which reduces to "3 20 −"; the latter could be written "3 4 - 5 *" (or 5 3 4 - *, if you wish to keep similar formatting), which unambiguously means "(3 4 -) 5 *".

Standard expression (infix notation)	Postfix Notation	Value
1 + 3	1 3 +	4
10 / 5	10 5 /	2
10 / (6 + 2 -3)	10 6 2 + 3 - /	2

How can I use a Stack to implement the calculator?

For the following pseudocode, consider a “symbol” as a number (0…9) or an operator (+, -, *, /). When I mention the entire string, I’m talking about the postfix expression (for example, 10 6 2 + 3 - /). In this instance, the symbols would be 10, 6, 2, +, 3, -, /

Pseudo-code might look like the following:

```
read in a symbol (number or operator)
As long as you’re not at the end of the entire string yet
      if the symbol is +
            pop the last 2 values from the stack
            push back their sum
      else if the symbol is *
            pop the last 2 values from the stack
            push back their product
      else if the symbol is –
            pop the last 2 values from the stack
            push back the difference (second value – first value)
      else if the symbol is /
            pop the last 2 values from the stack
            push back the quotient (second value / first value)
```

```
      else if the symbol is a number
            convert to a double and push it on the stack;
read the next symbol, repeat above steps
do until you're at the end of the string
```

Obviously, **you should skip spaces**. They are essentially there as delimiters between symbols (numbers and operators.) Check out the `String` class's `split( )` method, which converts a `String` into an array of `Strings`.

Deliverables / Submission Instructions

- You need to turn in your source code (just the .java files) **zipped up**
    - They should be submitted to Canvas under the appropriate assignment directory. **Just name the zip folder/file Proj3.zip** – Canvas automatically attaches your name to it.