CSCI 310, Data Structures

Homework 11 – RPN Calculator (*20 Points*)

Note: Homework assignments HW 10 and HW 11 are all related. Your solution to HW 10 is used in HW 11.

For this assignment you are to implement a basic RPN[1] calculator that supports integer operands and the binary integer operators +, -, *, /, and %. The style of arithmetic expressions our calculator will evaluate is also called a post-fix notation. Stacks are great for doing this job! Your task is to write a program that uses your stack from Homework 10 to perform these calculations as specified below.

Calculator Specification

Your program should be called RPNCalc and work as follows:

- The user enters input through System.in consisting of *operands* and *operators*, presumably in post-fix notation. The user can also enter instructions to get information about the current state of the stack and quit the program.
- Operands
  - All operands are integers.
  - If the user enters a valid integer, it is pushed onto the stack.
- Operator
  - Valid operators are +, -, *, /, %
  - If the user enters a valid operator, two integers are popped off the stack, the requested operation is performed. The result of the operation is pushed back onto the stack and is also printed, followed by a new line.
- Instructions
  - Valid instructions are ?, ^, $, !, which do the following:
    - ? (Question mark) Prints the current state of the stack using its toString method followed by a new line.
    - ^ (Caret) Pops the top element off the stack and prints only that element (not the entire stack) followed by a new line.
    - $ (Dollar sign) Removes all elements from the stack using its clear method.

[1] https://en.wikipedia.org/wiki/Reverse_Polish_notation

- **!** (Exclamation point or bang) Prints the message "Goodbye" and exits the program.

Error Conditions

There are several error conditions that you program must handle gracefully for full credit. Two examples are described below. You will have to figure out any further error conditions for yourself.

- If the user enters blah or 3.1 or anything else that doesn't make sense for an integer calculator, your program should make it clear that it can't do anything helpful with that input; *but it should not stop at that point*.
- If the user requests an operation for which there are not enough operands on the stack, your program should notify the user of the problem but leave the stack unchanged; again, it should certainly not stop at that point.

All error messages should be printed to the standard error device (System.err) instead of the standard output device (System.out). Of course, the regular input and output is done through standard input and standard output as usual. All error messages must start with the symbol # (that's a hash sign) and a space and be followed by a new line.

What to Turn In

Save all of the files for your program in a single folder, then create a zip file of that folder and submit the zip file through Blackboard.

Your project should include the following files:

a. StackGeneric_Interface.java – You should download this file from Homework 10 and use it as-is. Make no changes to this file.

b. DNode_Generic.java – The class you wrote for Homework 10.

c. DLinked_StackGeneric.java – The class you wrote for Homework 10.

d. EmptyStackException.java – The class you wrote for Homework 10.

e. RPNCalc.java – The program you are to write for this assignment.

Sample Runs

The following pages show three sample runs that will hopefully help you understand what you are trying to achieve. I've colored what I typed in green to help you follow along.

Sample Run 1 – Basic normal operation

```
RPN Calculator
?
TOS []
15
?
TOS [15]
5
?
TOS [5, 15]
/
3
?
TOS [3]
10
20
30
?
TOS [30, 20, 10, 3]
+
```

?	The first command displays the initial empty stack: TOS []
15	Pushes 15 on the stack
?	Prints the stack: TOS [15]
5	Pushes 5 on the stack
?	Prints the stack: TOS [5, 15]
/	Pops 5 and 15, divides 15 ÷ 5, prints the result, and pushes it back on the stack
?	Prints the stack: TOS [3]
10	Pushes 10 on the stack
20	Pushes 20 on the stack
30	Pushes 30 on the stack
?	Prints the stack: TOS [30, 20, 10, 3]

+	Pops 30 and 20, adds them, prints the sum, and pushes it back on the stack
?	Prints the stack: TOS [50, 10, 3]
^	Pops the 50 and prints it
?	Prints the stack: TOS [10, 3]
$	Clears the stack
?	Prints the now empty stack: TOS []
!	Exits the program

Sample Run 2 – The same example run the "fast" way

```
RPN Calculator
? 15 ? 5 ? / ? 10 20 30 ? + ? ^ ? $ ? !
TOS []
TOS [15]
TOS [5, 15]
3
TOS [3]
TOS [30, 20, 10, 3]
50
```

As you can see, if the entire sequence of integers, operators, and instructions is entered on a single line, they are all executed in order. It’s like having our own little programming language!

Run 3 – Handling errors

In this example I've colored what I typed in green, and I've also colored the error messages in yellow, just to help you follow along. Your program doesn't need to use colors (but remember to print the error messages using System.err).

```
RPN Calculator
1
?
TOS [1]
duck
# Invalid input: "duck" is not recognized.
?
TOS [1]
1.2
# Invalid input: "1.2" is not recognized.
?
TOS [1]
+
# Need 2 numbers for an operation.
?
TOS [1]
^
1
```

1	Pushes 1 on the stack
?	Prints the stack: TOS [1]
duck	This is an error. The appropriate message is printed.
?	Prints the stack: TOS [1]. Notice the error didn't keep us from continuing.
1.2	This is another error. The appropriate message is printed.
?	Prints the stack: TOS [1]. Again, the error didn't keep us from continuing.
+	This is an error (the stack doesn't have 2 numbers to add).
?	Prints the stack: TOS [1]. The error didn't keep us from continuing.
^	Pops the 1 and prints it.

?	Prints the now empty stack: TOS []
^	Error, cannot pop an empty stack. Prints the error message.
?	Prints the stack: TOS []. The error didn't keep us from continuing.
5	Pushes 5 on the stack.
?	Prints the stack: TOS [5]
!	Exits the program