CSOC 2430: HW5

Your Task:

In this homework, we will learn how to evaluate arithmetic expressions. You need to implement three parts:

PART1: Write your OWN class stack. The class stack must have the following four public functions at least:

- **top()**: returns the element at the top of the stack without removal.
- **pop()** remove the element at the current top of the stack and returns it.
- **add()**: add one element on the current top of the stack.
- **size()**: returns the size of the stack

Hint: You can use Java ArrayList to implement the stack. You are not allowed to traverse the items of the Arraylist to reach the elements of the stack. Only the last element of the stack is accessible. So, do not define any other function that violates the definition of the stack (e.g. if you want to print all elements of the stack you must first pop them from stack then print it)

PART2: Write a function that converts an expression in infix to postfix format using YOUR class stack. It should support all operation of +, -, *, /, and ^ with parentheses and follow standard Math precedence ordering of BODMAS. To learn more about BODMAS see this link: BODMAS.
The algorithm can also be found in the following link:
Infix To Postfix
The operands are single digit numbers. The input IS in correct format and you ARE NOT required to check the input correctness. All operators are binay (have two operands) and you do not need to handle unary operators such as negation.
Examples:

- infix: **(2)** postfix: **2**
- infix: **2-3*4+5/6** postfix: **234*-56/+**
- infix: **(2+3)^4*5** postfix: **23+4^5***

Hint: Do not change the order of the operands even if the final evaluation result is equal. e.g postfix of '2+3' is **23+** NOT **32+**

PART3 Given an expression in postfix notation, Use YOUR stack to

1. Check whether the input indeed follows a true postfix
2. Evaluate the postfix expression

The algorithm can be found in this link: Postfix Evaluation
The operands are single digit numbers. The input may not be in correct format and you ARE required to check the input correctness. All operators are binay (have two operands) and you do not need to handle unary operators such as negation. If the input is valid it evaluates the experession otherwise it prints **nv** which means not valid. The operands in input file are integer (e.g. 3). However your output should be in floating point format with 1 decimal place (e.g. 3.0). If you do not follow the output format you will lose all the points for that test case this time.
Examples:

- postfix: **23+-** evaluation:**nv**
- postfix: **23+** evaluation: **5.0**
- postfix: **23+4^5*** evaluation: **3125.0**

Hint: Do not print the scientific notations of numbers! Try to find some challenging test cases.

Implementation Detail:

The solution needs to use your own stack. Do not use Java built-in stack. Your program reads the input string and store it in an array/arraylist then uses your stack to implement part 2 and 3.

Input

The program takes 3 arguments:
part number: 2 for part2 and 3 for part3,
expression: a string inside " " without any space and
Output File Name.

```
$ java HW5 2 "2-3*4+5/6" "itp_1.output"
```

Output

output is a text file that contains the postfix of the expression in this case.

234*-56/+

```
$ java HW5 3 "23+4^5*" "p_1.output"
```

Output

output is a text file that evaluates the postfix expression if it is valid otherwise prints nv to the file.

3125.0

Run the program on Linux:

Upload these test cases to the linux server. The input for each test case is in commands.txt file. The corresponding output for each command is in a .txt file. Read each command to find out its output file name. Create a directory on the Linux server, **its name must be hw5**

```
$ mkdir hw5
```

Change your current directory to the hw5

```
$ cd hw5
```

Run the shell script to compile the program

```
$ sh compile_java.sh
```

Run the shell script to run the test cases

```
$ sh test_java.sh
```