Data Structures

Assignment #3 – Stack

The goal for this assignment is to implement ***Stack ADT*** using ArrayList or LinkedList implementation. Then, write a program to use an object of the developed Stack class to solve simple and practical problems. Please **name the programs as indicated** and add proper **program headers** and **output labels and comments.**

In our lecture, we have discussed handling boundary conditions in assignment 3. You can either throw an exception and catch it, or handle the boundary conditions without exceptions. No matter how you handle them, please make sure that your program will not be stopped by the improper input (your program should continue giving the user options till the use chooses 0 to exit).

**Part 1:** (20 points)

Develop a **new class**, called **MyStackYourname**, to implement, among others, key stack operations [***push(e), pop(), peek(), size(), isEmpty(),toString()***]we discussed in the class. Class Stack needs to be defined as generic stack with type <E> so that we can create stack objects that can hold data of different types, such as integer stack, string stack, char stack, double stack, etc.

**MyStackYourname** Class must have these methods

* **methodname(argument): returnType**
* isEmpty(): boolean Returns true if this stack is empty
* size(): int Returns the number of elements in this stack
* peek(): E Returns the top element in this stack
* pop(): E Returns and removes the top element in this stack
* push(E element): void Adds a new element to the top of this stack
* toString(): String Returns the String with all elements in current stack

//Based on your choice, toString will return the String with all elements in order [top to bottom] or [bottom to top]

//For this toString method, you are allowed to iterate through the linkedlist/arraylist as normal, and you don’t have to treat it like a stack where you cannot see or access any elements except the top.

Please consider how to handle boundary conditions (stack empty, etc.) for these methods. You should write your own code to handle those boundary conditions instead of just letting the code throw a NullPointerException or ArrayIndexOutOfBoundsException.

**Part 2:** (10 points)

Next, develop a simple test program called **TestStackYourname** to test each stack operation listed above and defined in your class **MyStackYourname**. Try **String type stack for the test program**. Use proper labels.

Make sure to allow the user to enter the stack content using interactive menu (embedded inside the sentinel loop to allow re-run):

**-----MAIN MENU-----  
0 - Exit Program  
1 – Push // Push should have user input  
2 - Pop  
3 – Peek (Top)  
4 - Size  
5 – Is Empty?  
6 – Print Stack**

**Choose menu:**

**Again, you cannot change method names, return type and parameter types.**

To evaluate your Class’s methods, the instructor may use another Test Program and will use your methods.

**// We will use main method (check Part 5) for testing both Part 3 and 4 at once**  
**Part 3:** (10 points)

Implement a class, call it **ExprYourname**, which will be used for transforming the expressions (infix to postfix) and evaluate postfix.

Write a method

public static String **infixToPostfix**(String infix){

// Write appropriate codes here

// Return postfix as a returned String of this method

}

Use your **MyStackYourname** to write the procedure of transformation from **infix to postfix**. Use the algorithm of infix to postfix translation which is introduced in the lecture, translate infix expression to postfix expression.

Assumption:

Input infix string doesn’t have whitespaces.

All input number is one-digit number (positive digit 0 to 9)

Input infix can include parenthesis ( )

Operator + - \* / ^ can be used

The output postfix string doesn’t have whitespaces.

**Part 4:** (10 points)

In the **ExprYourname** class, implement another method, call it **postfixEval**. Using class **MyStackYourname**, the **postfixEval** method should evaluate a postfix expression and return the correct calculated result.

Write a method

public static double **postfixEval**(String postfix){

// Write appropriate codes here

// Return calculated result value

}

Assumption:

Input postfix string doesn’t have whitespaces.

All input number is one-digit number (positive digit 0 to 9)

Operator + - \* / ^ can be used.

**Part 5:** (10 points)

In the **ExprYourname** class, write a main method which tests above two methods (Part 3 and Part 4)

Write a method

public static void **main**(String[] args){

// What is the process of main method.

// - User will insert infix

// - Invoke infixToPostfix to transform it to postfix expression

// - Invoke postfixEval to evaluate postfix expression

// - Show the calculated result

// Write appropriate codes here

// Allow the user re-run the program

}

// Examples

Enter an infix: (7+3)\*(3-1)/2 //Red characters are user input

Postfix Expression: 73+31-\*2/

Result value: 10.0

Do you want to continue? (Y/N) Y

Enter an infix: (5-6)/4\*2-(5+2)

Postfix Expression: 56-4/2\*52+-

Result value: -7.5

Do you want to continue? (Y/N) Y

Enter an infix: 3+2^(2+3)

Postfix Expression: 3223+^+

Result value: 35.0

Do you want to continue? (Y/N) N

**Submission:**Before submitting your programs, make sure you review the assignment submission requirements and grading guidelines on the course webpage. The grading guidelines explain some of the common errors found in programming assignments.

1. **One document** with screenshots to show all your results, **all individual .java** files (**MyStackYourname.java, TestStackYourname.java, ExprYourname.java**).
2. The assignment is due no later than **11:59 PM** on the due day posted in D2L.
3. Please compile and run your java files (only the .java files) right before you upload to the assignment submission folder in D2L.