**Lab Assignment #4 – Using ADT Stacks, Queues, and Lists**

Due Date: Friday, Week 8

Purpose: The purpose of this Lab assignment is to:

1. Design algorithms that describe operations on ADT stacks, queues, and lists
2. Implement and test appropriate methods in Java or Python

References: Read the course’s text chapter 6, 7 and the lecture slides. This material provides the necessary information that you need to complete the exercises.

Be sure to read the following general instructions carefully:

- This assignment must be completed individually by all the students.

**Exercise 1**

**If your first name starts with a letter from A-J inclusively:**

Suppose we want to extend the **PositionalList** ADT with a method, ***indexOf(p)***, that returns the current index of the element stored at position p. Write this method using only other methods of the **PositionalList** interface (not details of our LinkedPositionalList implementation).

Write the necessary code to test the method. **Hint:** Count the steps while traversing the list until encountering position p.

**If your first name starts with a letter from K-Z inclusively:**

Suppose we want to extend the PositionalList abstract data type with a method, ***findPosition(e)***, that returns the first position containing an element equal to e (or null if no such position exists).

Implement this method using only existing methods of the PositionalList interface (not details of our LinkedPositionalList implementation). Write the necessary code to test the method.

**Hint:** use the *equals* method to test equality.

(5 marks)

**Exercise 2**

**If your first name starts with a letter from A-J inclusively:**

Implement a method with signature *transfer(S, T)* that transfers all elements from stack S onto stack T, so that the element that starts at the top of S is the first to be inserted onto T, and the element at the bottom of S ends up at the top of T. Write the necessary code to test the method.

(2 marks)

**If your first name starts with a letter from K-Z inclusively:**

Write a recursive method for removing all the elements from a stack S. Write the necessary code to test the method.

**Hint:** First check if the stack is already empty.

**Exercise 3**

Implement a method with signature ***concatenate(LinkedQueue<E> Q2)*** for the **LinkedQueue<E>** class that takes all elements of Q2 and appends them to the end of the original queue. The operation should run in **O(1)** time and should result in Q2 being an empty queue. Write the necessary code to test the method. **Hint:** You may just modify the SinglyLinkedList class to add necessary support.

(3 marks)

**Evaluation:**

|  |  |
| --- | --- |
| **Functionality:**   * Correct implementation of requirements for implementing and testing methods * Code demonstration and brief explanation in a short video | 70%  10% |
| **Object-Oriented design**:   * Correct design of classes and methods similarly to chapter 6, 7 examples. * Correct use of generics * Correct use of naming guidelines for project, classes, variables, methods. | 15%  5% |
| **Total** | 100% |

**Naming and Submission Rules:**

You must **name your Eclipse project** according to the following rule:

**YourFullname\_COMP254Labnumber**. Example: **JohnSmith\_COMP254Lab4**

You must name package names **com.exercisenumber.yourfirstname.yourlastname**, for example: **com.exercise1.john.smith**

Provide your **student number and full name as a comment** at the top of main method for each exercise.

**Archive your project in a zip file** named according to the following rule:

**YourFullname\_COMP254Labnumber.zip**

Example: **JohnSmith\_COMP254Lab4.zip**

Upload the zip file on eCentennial using the Assignment link.