**44-542 Object-Oriented Programming**

**Lab09: Recursion Lab**

**Objective: Covers the usage of Recursion and provide practice with basic data structure. In addition, the opportunity to design and implement solution from scratch is provided.**

**NOTE:**

* Do not hard code any values.
* Go through the API provided for all Interfaces, classes details.
* Read every instruction carefully and follow them strictly.
* Do not change the name of the attributes, and methods given in API.
* You need not generate Javadoc for this project.
* @author annotation must contain your full name for all the classes in this project.
* Use @override annotation for every relevant method in all the classes of your project.
* Do everything in the same project.

**The lab…**

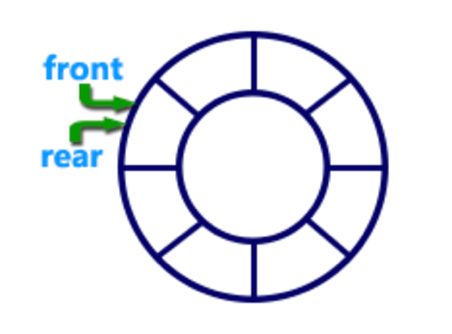
We begin with a warmup which you have hopefully done already as part of your study after the class discussion on recursion.

1. Worksheet: “recursion.docx” found in module Wk 11.
2. Create a package in ***“Lastname*\_Lab08Recursion”** project and name it as recursion.
3. Create a main class and name it as RecursionWorksheet.
4. Now follow the instructions provided in worksheet and complete the worksheet as per instructions.

The next three components should be developed in the same project. However, you are given the opportunity to name the packages and files yourself. It will be critical that the files created are all given meaningful names so that the grader can easily identify which files are relevant to each part of the question. Do not use the data structures from the Java Collections Framework in the following questions unless otherwise instructed. No sample output or input is provided – you have some flexibility to do as needed, but make sure it is easy to use and obvious regarding what is happening. It is up to you to test the code.

1. Use an array of 50 int values and populate it with random integer values. Write a sort method that will recursively sort the array. Print the contents of the array before and after sorting to demonstrate that your solution works.
2. Write a program that reads a String in from the keyboard and then recursively determines if the string is a palindrome. A palindrome is a string that reads the same from left to right as it does from right to left. Examples include. “Anna”, “Hannah”, “a dog, a plan, a canal, pagoda”, “A man, a plan, a cat, a ham, a yak, a yam, a hat, a canal-Panama!”. [Hint. Convert the string to all lowercase or all uppercase after reading it in, remove all spaces and punctuation.]
3. *5 BONUS POINTS. THIS PART IS OPTIONAL.*

Use an array of 20 int values to represent a circular queue. Values are added to the queue at the rear and removed at the front of the queue. The index of the head and tail elements are stored in appropriate variables. You can think of a circular queue as an array that bent into a circle so that the beginning and end of the array are joined.



When a value is added to the circular queue, it is inserted at the rear and the rear pointer is increased by one. If the rear pointer was referencing the last element in the array, then increasing its value by one results in it wrapping to the beginning of the array.

When a value is removed from the circular queue, it is removed from the front of the queue and the front pointer is increased by one (also wrapping back to the beginning of the array if it currently references the end of the array.

A circular queue can hold as many values as the length of the array (20 int values in our case).

A queue is full when the array holds 20 elements.

The queue is empty with the array holds 20 elements

You can read about a circular queue (also called a circular buffer) at <https://en.wikipedia.org/wiki/Circular_buffer>.

In this part of the lab you need to design, build a test a circular queue of 50 int values. It needs to have the following methods (you need to determine the parameters needed):

* Insert: add an element to the circular queue if it is not already full; throw an exception otherwise.
* Remove: remove the element at the front of the queue if it is not empty; throw an exception otherwise.
* Retrieve: return the value at the front of the queue if it is not empty, but do not remove it; throw an exception otherwise.
* Length: Returns the number of elements in the circular queue.
* Print: Print the contents of the queue with the front of the queue shown first. [THIS MUST BE DONE RECURSIVELY].
* IsFull: returns a Boolean value indicating if the queue is full.
* IsEmpty: returns a Boolean value indicating if the queue is empty.

Make sure you develop a main method that demonstrates that each of the methods above work properly and that the queue properly handles attempts to insert an element into a full queue and remove an element from an empty queue.

**Submit your solution to the lab by following the steps below:**

* Save your files in NetBeans.
* Zip your entire Project. (It should be called **Lastname\_lab08Recursion**.zip where **Lastname** is your last name.)
* Submit the Zip file to the **lab08Recursion** dropbox on Canvas.
* Download the Zip file which you have submitted.
* Look into the Zip file and verify the class files are updated. If not resave your project in NetBeans and resubmit.