ITCS 2214 Data Structures
Assignment 2

Download the assignment from Moodle. Change the name of the folder to your first-name underscore last-name underscore assign2 (FirstName_LastName_Assign2). DO NOT change the structure of the files. If you write your code somewhere else, copy and paste it in the CircularLinkedList.java file. ENSURE you DO NOT paste over the package declaration at the top of the file. DO NOT change the interface. Use Main.java as your driver; put test code in this file to debug your data structure. It is ok if you leave the code in Main.java, I will use my own code. In the Read Me file, write a description of each method you implemented (what the method is suppose to do). Upload a zip folder to Moodle before the deadline.

Fill in the code in the CircularLinkedList.java file. You may use other private helper functions as you see fit. However, NO other instance or static variables should be used. You will NOT be graded on the efficiency of your code. In other words, just get the job done. My test code will invoke operations on your list, checking its state.

These are the methods you should implement:

a. CircularLinkedList(): initializes an empty unsorted list.

b. CircularLinkedList(boolean sorted): initializes an empty unsorted or sorted list based on the *sorted* parameter.

c. addToFront(T element): add the *element* to the head of the list in an unsorted list, or to its natural ordered position if the list is sorted. (3 points)

d. addToRear(T element): add the *element* to the tail of the list in an unsorted list, or to its natural ordered position if the list is sorted. (3 points)

e. addAfter(T element, T target): add the *element* after the *target* element in an unsorted list, or to its natural ordered position in a sorted list. If the target element is not in the list, add the *element* to the rear of the list in an unsorted list. (3 points)

f. addBefore(T element, T target): add the *element* before the *target* element in an unsorted list, or to its natural ordered position in a sorted list. If the target element is not in the list, add the *element* to the head of the list in an unsorted list. (3 points)

g. removeFirst(): remove and return the first element in the list. (1 point)

h. removeLast(): remove and return the last element in the list. (1 point)

h. remove(T target): remove and return the *target* element from the list. If the *target* element is not in the list, return null. (2 points)

p. clear(): deletes all the elements from the list. (2 points)

i. moveUp(T target): move the *target* element up one place towards the head of the list. The element above the target element will move towards the tail. The elements does not wrap around the head and tail of the list. (2 points)

j. moveDown(T target): move the *target* element down one place towards the tail of the list. The element below the target element will move towards the head. The elements does not wrap around the head and tail of the list. (2 points)

k. contains(T target): returns true if the *target* element is in the list, otherwise return false (2 points)

l. ascendingSort(): sorts the list in ascending order from head to tail. (2 points)

m. descendingSort(): sorts the list in descending order from head to tail. (2 points)

n. slide(int amount): slides all the elements down (towards the tail) or up (towards the head) in the list by the specified *amount* . If it is a positive number the elements slide down; they slide up if it is a negative number. The elements wraps around the head and tail of the list. (5 points)

o. shuffle(): randomly shuffles the elements in the list. (3 points)

o. isEmpty(): returns true if the list has no elements in it, false otherwise. (1 point)

p. size(): returns how many elements are in the list. (1 point)

n. toString(): returns the contents of the list from head to tail, with one element per line. (2 points)