CSC 212 Homework # 2
Lists & Queues
Due date: 30/10/2016 at 13:00

Guidelines:	This is an individual assignment. The homework must be **submitted electronically through LMS**. **Hard copy submissions are not accepted**.

Problem 1

1. Write static method findIth, user of the ADT *List*, to make i-th element in a list (for example, 2nd, 4th or 5th node) the current (numbering starts from 1). Assume that the position i is valid. Method: *public static <T>void findIth (List <T>l, int i)*

   Example 1.1. *Assuming the list l: $A \to B \to C \to D \to E \to F$, then calling* findIth(l, 2) *puts the current on B,* findIth(l, 5) *puts the current on E.*

2. Write the method *public static <T>void moveBack(List<T>l, int p)*, user of the ADT *List*, which moves the current backwards with p positions. Assume that l is not empty and that p is a valid position. **Do not use any auxiliary data structure.**

   Example 1.2. *Assuming the list l: $A \to B \to C \to D \to \boldsymbol{E} \to F$, where the current element is the one in bold, then calling* moveBack(l, 2) *results in $A \to B \to \boldsymbol{C} \to D \to E \to F$, calling* moveBack(l, 4) *results in $\boldsymbol{A} \to B \to C \to D \to E \to F$.*

Problem 2

1. Write the method *public void removeK(int k)*, member of the class *LinkedList*, which removes the k elements that are after the current (current is not removed). Assume that the list is not empty and k is valid. **Do not call any of the methods of the class *LinkedList* and do not use any additional data structures.**

Example 2.1. *If* $l : A \rightarrow B \rightarrow \boldsymbol{C} \rightarrow D \rightarrow E \rightarrow F \rightarrow G$*, where* C *is the current, then calling* l.removeK(2) *results in* $l : A \rightarrow B \rightarrow \boldsymbol{C} \rightarrow F \rightarrow G$*, whereas calling* l.removeK(4) *results in* $l : A \rightarrow B \rightarrow \boldsymbol{C}$*.*

2. Implement the following method in the class *LinkedList*:

   - Procedure *insertBeforeCurrent(T e)*
   - Requires: The list l should not be full.
   - Results: The new element e is inserted before the current and the new element is made the current.

Problem 3

1. Write the method *direction*, member of the class *DoubleLinkedList*, which accepts an element e and returns the direction of e relative to the current pointer. It returns -1 if e is left of current, 0 if e is exactly on current, and 1 if e is right of current. Assume that e exists in the list. The method signature is *public int direction(T e)*. **Do not call any methods of the class *DoubleLinkedList* and do not use any other data structure**.

   Example 3.1. *Assume the list is:* $A, B, C, D, E, \boldsymbol{F}, G, H$ *and current is on* F*. Calling direction("B") returns -1. Calling direction("F") returns 0. Calling direction("G") returns 1.*

2. Write a method that takes as input two double linked lists and returns true if they contain the same elements but in reverse order, otherwise returns false. The method's signature is: *public static boolean areReversed(DoubleLnkedList<T>l1, DoubleLinkedList<T>l2).*

Problem 4

1. Write the method *public static<T>void remove(Queue<T>q, int[] pos, int k)*, which removes all the elements of q located at the positions indicated in *pos* (k is the size of *pos*). Assume that *pos* is sorted in increasing order with no duplicates and contains only valid positions. The numbering of the positions starts from 0 at the head. The method must run in $O(n)$, where n is the size of q (not $O(kn)$).

   Example 4.1. *If* $q : A, B, C, D, E, F, G, H$ *and* $pos : 1, 2, 5$*, then after calling* remove(q, pos, 3)*,* q *becomes* A, D, E, G, H*.*

2. Write an array implementation of the ADT PQueue. The *serve* method must run in $O(1)$, *enqueue* in $O(n)$.

Problem 5 [Priority queue]

A store announces a sale campaign whereby any customer who buys two items gets 50% off on the cheaper one. If the customer buys more than two items, he/she must group them into pairs of two to indicate the items that the offer should apply to.

1. Suppose you want to buy n items in total. Write a method that will give you the best pairing of the items (the one with the minimum price). The method's signature is:

   `public static LinkedList<ItemPair> minPairing(LinkedList<Item> items).`

2. If you leave it up to the store owner, he/she will try to pair the items in order to obtain the maximum price. Write a method that will help the store owner achieve this. The method's signature is:

   `public static LinkedList<ItemPair> maxPairing(LinkedList<Item> items).`

3. How much will you gain if you use your method (instead of the shop owner's method) for the following list of item prices: 60 SAR, 100 SAR, 400 SAR, 600 SAR, 200 SAR, 80 SAR.

```
public class Item {
        private int id;
        private double price;
        public Item(int id, double price) {
                this.id = id;
                this.price = price;
        }
        int getId() {
                return id;
        }
        double getPrice() {
                return price;
        }
}
```

```
public class ItemPair {
        public Item first;
        public Item second;
        public ItemPair(Item first, Item second) {
                this.first = first;
                this.second = second;
        }
}
```