CSC 212 Programming Assignment # 1
Implementing and Using Lists
Due date: 14/02/2019

Guidelines:	This is an **individual** assignment. The assignment must be submitted to **Web-CAT**

The goal of this assignment is to implement and use the ADT Comparable-List. The assignment is divided into two parts. In the first part, you will implement the ADT Comparable-List using linked and array representations. In the second part, you will write methods that use these implementations:

1. Given the following specification of the ADT Comparable-List, implement this data structure using both array and linked representations. You should write two classes `ArrayCompList` and `LinkedCompList` that both implement the interface `CompList`. You may use code from the lecture notes for the methods similar to those discussed in class.

Specification of ADT Comparable-List

- empty (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **output**: flag.
- full (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L has reached the maximum then flag is set to true otherwise false. **output**: flag.
- findFirst ( ): **requires**: list L is not empty. **input**: none. **results**: first element set as the current element. **output**: none.
- findNext ( ): **requires**: list L is not empty. Current is not last. **input**: none. **results**: element following the current element is made current. **output**: none.
- findPrevious( ): **requires**: list L is not empty. Current is not head. **input**: none. **results**: element preceding the current element is made current. **output**: none.
- last (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the last element is the current element then flag is set to true otherwise false. **output**: flag.
- first (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the first element is the current element then flag is set to true otherwise false. **output**: flag
- retrieve (Type e): **requires**: list L is not empty. **input**: none. **results**: current element is copied into e. **output**: element e.
- update (Type e): **requires**: list L is not empty. **input**: e. **results**: the element e is copied into the current node. **output**: none.

- insert (Type e): **requires**: list L is not full. **input**: e. **results**: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.
- remove ( ): **requires**: list L is not empty. **input**: none. **results**: the current element is removed. If L is empty, current will point to null. If the next element exists, it is made current, else the first element is made current. **output**: none.
- compareToPrevious (int c): **requires**: list L is not empty and current is not first. **input**: none. **results**: compares the current element to the previous element. The method returns a negative integer, zero, or a positive integer as the previous element is less than, equal to, or greater than the current element. **output**: flag
- swap(): **requires**: list L is not empty and current is not first. **input**: none. **results**: swaps the value of the current element with the value of the previous element. **output**: none.

2. Write a class called `CompListUtils` having the following static methods:

   (a) **public static** `<T extends Comparable<T>>` **void** `bubbleSort(CompList<T> l)`: A static method that sorts the elements of the list `l` in an increasing order using the bobble sort algorithm. DO NOT CALL THE METHOD retrieve() IN THIS METHOD.
   Example: If $l : 1, 2, -1, 10, 3, 44, 1, -10, 222, 1, -10$ then calling `bubbleSort(l)` will make $l = -10, -10, -1, 1, 1, 1, 2, 3, 10, 44, 222$.
   If $l = a, c, z, t, y, j, k, a, k, l, e, v, t$ calling the `bubbleSort(l)` will make
   $l = a, a, c, e, j, k, k, l, t, t, v, y, z$.

   (b) **public static** `<T extends Comparable<T>> CompList<T> combine(CompList<T> l1, CompList <T> l2,` **int** `k)`: A static method that combines the two lists `l1` and `l2` based on `k`. The method returns a new list which contains a merge of the first `k` elements in list `l1` and the last `k` elements in list `l2`.
   Example: If $l1 : A, B, C, D$ and $l2 : H, I, J, K$ then calling `combine(l1, l2, 3)` will return A, B, C, I, J, K. Also, calling `combine(l1, l2, 5)` will return A, B, C, D, H, I, J, K.

1 Deliverable and rules

You must deliver:

1. Source code submission to Web-CAT. You have to upload the following classed in a zipped file:

   - `LinkedCompList.java`
   - `ArrayCompList.java`
   - `CompListUtils.java`

   Notice that you should **not upload** the interface `CompList`.

The submission **deadline** is: **14/02/2018**.

You have to read and follow the following rules:

1. The specification given in the assignment (**class and interface names, and method signatures**) must not be modified. Any change to the specification results in compilation errors and consequently the mark zero.

2. All data structures used in this assignment **must be implemented** by the student. The use of Java collections or any other data structures library is strictly forbidden.

3. This assignment is an individual assignment. Sharing code with other students will result in harsh penalties.

4. Posting the code of the assignment or a link to it on public servers, social platforms or any communication media including but not limited to Facebook, Twitter or WhatsApp will result in disciplinary measures against any involved parties.

5. The submitted software will be evaluated automatically using Web-Cat.

6. All submitted code will be automatically checked for similarity, and if plagiarism is confirmed penalties will apply.

7. You may be selected for discussing your code with an examiner at the discretion of the teaching team. If the examiner concludes plagiarism has taken place, penalties will apply.