CSC 212 Homework # 0
Revision & Generics
Due date: 10/07/2017

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

Problem 1

Write a method `checkModify` that takes as input an integer parameter `n` and performs the following:

- If `n` is positive, the method returns `false`.

- If `n` is negative, the method returns `true` and changes `n` to `|n|`.

$$\cdots\cdots\cdots$$

Problem 2

Write a method that computes the value of a polynomial p at x:

$$p(x) = a_n x^n + a_{n1} x^{n1} + \ldots + a_1 x + a_0.$$

The signature of the method is *double evalPol(double A[], int n, double x)*, where A is the array containing the polynomial coefficients (a_0 is stored in $A[0]$, a_1 in $A[1]$, etc.), n is the degree of the polynomial and x is the value of the variable.

$$\cdots\cdots\cdots$$

Problem 3

1. What are generics in Java ? What are the advantages of using generics?

2. Write a generic method to exchange the positions of two different elements in an array.

3. Write a generic method for counting the number of times that a given variable occurs in an array.

4. Write the generic method `public static <T> void reverse(T[] A, int n)` that takes as input a generic array and reverse the order of its n first elements.

· · · · · · · · ·

Problem 4

In Java, it is not possible to create an array of a generic class. For instance, the statement `A<T>[] a = new A<T>[n];` results in a compilation error. To solve this problem it is possible to create a simple data structure that provides the same functionalities as an array (access by index). Complete the class below which uses an array to store data (the array is initially empty).

```java
public class GArray<T> {
        ...

        public GArray(int size) { // size is the length of the array
        }

        public T get(int i) { // Return the element at position i
        }

        public void set(int i, T e) { // Put e at index i
        }

}
```

· · · · · · · · ·

Problem 5

Consider the following class:
```java
public class Pair<T, U> {
        public T first;
        public U second;
        public Pair(T first, U second) {
                this.first = first;
                this.second = second;
        }
}
```
Write the method `public static <T, U> GArray<Pair<T,U>> pair(T[] A, U[] B, int n)`, which creates a `GArray` of pairs from the first `n` elements of `A` and `B` by pairing each element from `A` with the element from `B` at the same index.

· · · · · · · · ·

Problem 6

When writing a generic class or method, it is possible to constrain the generic type to implement a given interface. For instance, a generic sorting method must compare elements, which means that its generic type must be comparable. One way to enforce this is to require the generic type to implement the Java standard interface `Comparable`.

Example 6.1. *If we want the class* `A` *to be comparable, we define it as:*

```
public class A implements Comparable<A> {...}
```

Notice that the interface `Comparable` is a generic interface, with the parameter type indicating the type of objects that "this" object may be compared to. In the example, objects of class `A` can only be compared to objects of the same class.

The interface `Comparable` defines a single method `int compareTo(T o)`, which returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the parameter object.

1. Write the method `public static <T extends Comparable<T>> void sort(T[] A, int n, boolean incr)`, which sorts the `n` first elements of `A` in increasing order if `incr` is true, or decreasing order if `incr` is false.

2. Write a method `test` that creates an array of `String` and uses the method `sort` to sort it.

Remark. *Notice that when constraining the generic type* `T` *to be* `Comparable`, *we use* `extends` *and not* `implements`.

· · · · · · · · ·