CSC 212 Homework # 0
Revision
Due date: 06/10/2016

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

Remark. *For problems 1 and 2, you may refer to the textbook, Section 2.5.2.*

Problem 1 [Generic methods]

When a method takes as parameter a non-instantiated generic class, it is called generic. A generic method can be static or not.

Complete the generic method *search* below that takes as input a generic array *data* of size n and searches for the element e. If e is found, the index of its first occurrence is returned, otherwise -1 is returned.

```
public class Utils {
        // The <T> before the return type indicates that the
           method "search" is generic. Notice that the class "
           Utils" is not generic.
        public static <T> int search(T[] data, int n, T e) {
                ...
        }
}
```

Problem 2 [Generic interfaces]

Consider the following generic interface:

```
public interface Condition<T> {
        boolean test(T data);
}
```

1. Complete the method *search* below that takes as input a generic array *data* of size n and a *Condition cond*. The method returns the index of the first element in *data* that satisfies the condition (that is, for which cond.test() returns true), or -1 if no such element exists.

```
public class Utils {
        public static <T> int search(T[] data, int n, Condition<
           T> cond) {
                ...
        }
}
```

2. Using the method *search* above, write the method *searchEven* that finds the first even number in an array of integers. You need to write the appropriate implementation of the interface *Condition*.

```
public int searchEven(Integer[] data, int n) {
        ...
}
```

Problem 3

The following is the specification of a data structure called *GenericArray*:

- **Domain**:
  - Structure: linear.
  - Data type: generic.
- **Operations**: All operations will be done on a *GenericArray arr* of size n.
  - Procedure get(int i, T e). **Requires**: $0 \leq i < n$. **Results**: e is set to the element of *arr* at position i.
  - Procedure set(int i, T e). **Requires**: $0 \leq i < n$. **Results**: the element of *arr* at position i is set to e.

1. Complete the implementation of *GenericArray* below:

```
public class GenericArray<T> {
        ...
        public GenericArray(int n) {
                ...
        }
        public T get(int i) {
                ...
        }
        public void set(int i, T val) {
                ...
        }
}
```

2. Consider the class *Box*:

```
public class Box<T> {
        private T data;
        public Box(T data) {
                this.data = data;
        }
        public T get() {
                return data;
        }
        public void update(T data) {
                this.data = data;
        }
}
```

and a method that uses this class by creating an array of *Box<String>*:

```
public class ArrayOfBox {
        public static void main(String[] args) {
                Box<String>[] b = new Box<String>[3];
                b[0] = new Box<String>("A");
                b[0].update("B");
        }
}
```

What happens when you compile this code. Use the class *GenericArray* to solve this problem.

Problem 4

The following is the specification of the data structure *Pile*.

- **Domain**:
  - Structure: linear.
  - Data type: generic.
- **Operations**: All operations will be done on a *Pile p*.
  - Procedure empty(boolean flag). **Requires**: None. **Results**: *flag* is set to true if p is empty.
  - Procedure full(boolean flag). **Requires**: None. **Results**: *flag* is set to true if p is full.
  - Procedure add(T e). **Requires**: p is not full. **Results**: e is added at the end of p.
  - Procedure remove(T e). **Requires**: p is not empty. **Results**: e is set to the last element of p and this last element is removed.

The following is the Java implementation of *Pile* (**Do not complete the implementation**).

```
public class Pile<T> {
        ...
        public Pile(int maxSize) {
                ...
        }
        public boolean empty() {
                ...
        }
        public boolean full() {
                ...
        }
        public void add(T e) {
                ...
        }
        public T remove() {
                ...
        }
}
```

Write the method *compare* that compares two objects of type *Pile* and returns true if they are equal, false otherwise. The method signature is *public static* $<T>$ *boolean compare(Pile* $<T>$ *p1, Pile* $<T>$ *p2).* The two piles p1 and p2 must not change.