Foundations of Computer Science - a.a. 2022-2023
Programming Test - 1 Feb. 2023

GENERIC STACK OF PARAMETRIC TYPE ELEMENTS

The stack abstract data type is a container of items with Last In First Out (LIFO) data access mode. It is defined with the java interface below:

```
public interface Stack<T> extends Container // Generic stack ADT
{  /**
      Removes the object at the top of this stack and returns it
      @return the object at the top of this stack
      @throws EmptyStackException if this stack is empty
   */
   T pop()throws EmptyStackException;

   /**
      Pushes an item onto the top of this stack, if not null
      @param item the item to be pushed onto this stack
      @throws IllegalArgumentException if the item to be pushed is null
   */
   void push(T item);

   /**
      Returns the position of an object in the stack. If the object x is present
      in the stack, the method returns the distance from the top of the stack to
      the nearest occurrence of x. The topmost item on the stack is considered
      to be at distance 0.
      @param x the desired object
      @return the position from the top of the stack where the object is
              located; -1 indicates that the object is not in this stack.
   */
   int search(T x);

   /**
      Looks at the object at the top of this stack without removing it
      @return the object at the top of this stack
      @throws EmptyStackException if this stack is empty
   */
   T top() throws EmptyStackException;
}
```

where T is the parametric type of the stack items, Container and EmptyStackException are the interface and class, respectively, below;

```
public interface Container
{  /**
      Checks if this container is empty
      @return true if this container is empty
   */
   boolean isEmpty();

   /**  Makes this container empty  */
   void makeEmpty();

   /**
      Returns the size of this container
      @return  the number of items in this container
   */
   int size();
}
public class EmptyStackException extends RuntimeException
{
   public EmptyStackException() {}

   public EmptyStackException(String err) { super(err); }
}
```

Write first the code of the `public class S<T> implements Stack<T>` where the constructor constructs an empty stack.

Write then the code of the following extended stack class
`public class ES<T extends Comparable> extends S<T>` with the following components:

- constructor `public ES(T[] a)`: constructs an extended stack containing the elements in array a, where the elements are in LIFO sequence and the topmost element is at index `lenghth-1`
- method `public Comparable[] toArray()`: returns an array containing all the elements in this extended stack in the LIFO sequence, where the topmost item is at index 0
- method `public Comparable[] toSortedArray()`: returns an array containing all the elements in this extended stack sorted in ascending order according to their natural ordering
- method `public int search(T x)`: overrides the superclass method of the same name. It behaves in the same way as the method in the superclass, but it is implemented through a recursive approach.

During test evaluation, the code will be tested with the commands below:

```
$rm *.class
$javac STester.java
$javac ESTester.java
$java STester
$java EStester
```

where STester and ESTester are the classes below:

```
public class STester {
   public static void main(String[] args) {
      // constant arrays
      String[] CITIES =  {"Teheran", "Doha", "Cairo", "Madrid", "Stockholm"};
      String[] TARGETS = {"Paris", "Stockholm", "Cairo", "Doha"};

      // test stacks
      S<String> s1 = new S();
      S<String> s2 = new S();

      //push data into test stacks
      for (String s : CITIES) {
         s1.push(s);
         s2.push(s);
      }

      System.out.print("SIZE = " + s1.size() + "\n");     // test of size method

      System.out.print("TOP & POP = ");
      while (!s1.isEmpty()) {                          //test of isEmpty method
         System.out.print(s1.top() + " ");                 //test of top method
         s1.pop();                                         //test of pop method
      }
      System.out.println();

      System.out.println("SEARCH = ");
      String top = s2.top();
      for (String s : TARGETS) {
         int n = s2.search(s);
         if (n < 0) {
            System.out.println("   " + s + " is not found");
         }
         else {
            System.out.println("   " +s+ " is at distance " +n+ " from " + top);
```

```
         }

      }
      System.out.println("SIZE 1/2 = " + s1.size()+"/"+ s2.size()); // test of size
   }
}
public class ESTester {
   public static void main(String[] args) {
      // constant arrays
      String[] CITIES =  {"Teheran", "Doha", "Cairo", "Madrid", "Stockholm"};
      String[] TARGETS = {"Paris", "Stockholm", "Cairo", "Doha"};

      // test estended stack
      ES<String> s1 = new ES(CITIES);

      System.out.print("TOARRAY: ");
      Object[] v = s1.toArray();                      // test of toArray method
      for (Object o : v) {
         System.out.print(o + " ");
      }
      System.out.println();

      System.out.print("TOSORTEDARRAY: ");
      v = s1.toSortedArray();                    // test of toSortedArray method
      for (Object o : v) {
         System.out.print(o + " ");
      }
      System.out.println();

      System.out.println("SEARCH = ");
      String top = s1.top();
      for (String s : TARGETS) {
         int n = s1.search(s);                        // test of search method
         if (n < 0) {
            System.out.println("   " + s + " is not found");
         }
         else {
            System.out.println("   " +s+ " is at distance " + n + " from " + top);
         }
      }
      System.out.println("SIZE = " + s1.size()); // test of size method
   }
}
```

The `OKS.txt` and `OKES.txt` files attached provide the printouts on standard output when the `STester` and `SETester` classes run.
Normally, code that reports errors at compile time is considered insufficient.

During the code evaluation, the following maximum scores will be awarded:
S: push 7, pop 7, top 7, search 3 (Tot = 24)
ES: constructor 1, toArray 1, toSortedArray 2, search 2 (Tot = 6)
1 point can be assigned for the style and 1 point for the time complexity of method toSortedArray in the worst case.

In the implementation of the S and ES classes, it is not allowed
- to use non private instance variables
- to add non-private methods
- to use classes of the Java Platform API except for those ones of the java.lang package

At the end of the programming test, leave in the working directory the files you have produced.

File written by students must contain as first line a java comment including student first and family names, ID, date and number of the working station.

FOUNDATIONS OF COMPUTER SCIENCE
Name_____ID._____
Date _____Taliercio work station No._____

Undersign one of the following statements:

[ ] I deliver for evaluation the following files:
( )S.java ( ) ES.java ( ) other_____
Signature_____

[ ] I withdraw from the programming test.
Signature_____