Foundations of Computer Science - Group 2
Programming test dated 16-07-2018

STACK OF PARAMETRIC DATA ITEMS

==========

A generic stack is a collection of parametric data elements in which the insertion and removal of elements occurs according to the LIFO (Last In First Out) policy. It is defined by the following interfaces:

```
public interface Stack<T> extends Container // Stack ADT
{
   /**
      inserts the specified item on top of this stack
      @param o the specified item
   */
   void push(T o);

   /**
      inspect the item at the top of this stack
      @return the item at the top of this stack
      @throws EmptyStackException if this stack is empty
   */
   T top() throws EmptyStackException;

   /**
      removes the item at the top of this stack
      @return the item removed
      @throws EmptyStackException if the stack is empty
   */
   T pop()throws EmptyStackException;
}
public interface Container
{
   /**
      checks if this container is empty
      @return true if this container is empty, otherwise false
   */
   boolean isEmpty();

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

   /**
      provides the number of data items in this container
      @return the number of data items in this container
   */
   int size();
}
```

where the EmptyStackException class is the following

```
public class EmptyStackException extends RuntimeException
{
```

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

Write the code of the class

```
public class S<T> implements Stack<T>
{
   public String toString() {...}
}
```

where

- the constructor initializes an empty stack
- the toString method returns a string containing the textual descriptions of the elements of the stack in LIFO sequence separated by a space.

The evaluation of the test will take into account the time complexity of the class S methods, considering the solution in which all the methods defined in the given interfaces have asymptotic time complexity equal to O(1) at least in the average case to be excellent.

Write then the code of the Extended Stack subclass of comparable elements

```
public class SE <T extends Comparable> extends S <T>
```

where:

- the constructor initializes an empty extended stack
- the `public Stack clone()` method returns a distinct stack containing the same elements of this stack, in the same sequence
- the `public Stack reverse()` method returns a stack containing the same elements as this stack, but in reverse sequence
- the `public Object[] toArray()` method returns the elements in LIFO sequence in an array
- the `public Object[] toSortedArray()` method returns an array containing the stack elements sorted according to their natural order

During the correction the code will be tested with the commands below:

```
$ rm * .class
$ javac SETester.java
$ java SETester < numbers.txt
```

where `numbers.txt` is the attached file and the `SETester.java` class is the following:

```
public class SETester
{
   public static void main(String[] args)
   {
      // creating of a new extended stack
      SE<String> s = new SE();

      // reading data from standard input
      java.util.Scanner in = new java.util.Scanner(System.in);
      while (in.hasNext())
         s.push(in.next());              //test of the push method
      in.close();

      // printing messages on standard output
      System.out.print("SIZE = " + s.size() + "\
n");            // test of size
      System.out.print("TOSTRING = " + s.toString() + "\n");
//test of toString
```

```
      System.out.print("CLONE = ");
      Stack s1 = s.clone();                        // test of clone
      System.out.println(s1.toString());       //test of to string

      System.out.print("REVERSE = ");
      s1 = s.reverse();                           // test of reverse
      System.out.println(s1.toString());
//test of to string

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

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

      System.out.print("SIZE = " + s1.size() + "\");// test size
   }
}
```

The attached OK.txt file contains the printout on standard output when the SETester class runs and reads data from the numbers.txt file.
Generally, the tests that give rise to compilation errors are considered insufficient.
During the correction, the following maximum scores will be assigned to the methods:
`S: push 6, pop 6, top 4, toString 6 (Tot = 22)`
`SE: clone 1, reverse 2, to Array 2, toSortedArray 2 (Tot = 7)`
one point can be assigned for the programming style and 2 points for the method time complexity.
In the realization of the S and SE classes, it is not permitted:

- to add non-private variables and / or methods;
- to use classes of the Java Platform API except those of the java.lang package.

At the end of the test, leave the files produced and the files provided by the teacher in the working directory.
The files produced must contain as the first line a comment with the name and surname of the student, roll number, date and number of the work station.
FOUNDATIONS OF COMPUTER SCIENCE (Group 2)
Name _____Roll no. _____ PC no. ADT_____

I submit my code consisting of the following files: [] S.java [] SE.java [] other_____ Signature _____
I do not submit my code and I withdraw from the exam.
Signature _____