Foundations of Computer Science
Exam dated 3 September 2025
Programming Test

QUEUE OF COMPARABLE ITEMS

The comparable item QUEUE abstract data type, a container of comparable elements implementing the first-in first-out (FIFO) data policy, is defined with the following interface

```
public interface Queue { // ADT Queue
   /**
      Returns true if the specified comparable item is contained in this queue
      @param item the item to check
      @return true if the specified comparable item is contained in this queue
   */
   boolean contains(Comparable item);

   /**
      Removes the front item from this queue and returns it.
      @return the removed front item of this queue or null if the this queue is empty.
   */
   Comparable dequeue();

   /**
      Inserts the specified comparable item into this queue.
      @param item the specified item to be inserted.
      @throws java.lang.IllegalArgumentException if the specified item is null.
   */
   void enqueue(Comparable item) throws IllegalArgumentException;

   /**
      Returns the front item of this queue.
      @return the front item of this queue or null if this queue is empty.
   */
   Comparable front();

  /**
      Checks if this queue is empty.
      @return true if this queue is empty.
   */
   boolean isEmpty();

   /**
      Makes this queue empty.
   */
   void makeEmpty();

   /**
      Returns the number of items in this queue.
      @return number of items in this queue.
   */
   int size();

   /**
      Returns the maximum item in this queue according to the natural ordering
      of the elements of this queue.
      @return the maximum data item in this queue or null if this queue is empty.
   */
   Comparable maxItem();
}
```

Code first the class

public class Q implements Queue where the constructor constructs an empty queue. Then code the following extended queue:
public class EQ extends Q where
- public EQ() constructs an empty extended queue
- public EQ(EQ q) constructs an extended queue with the same items as the specified extended queue and in the same FIFO sequence
- public T maxItem() overrides the superclass method of the same name using a RECURSIVE implementation.
- public Comparable[] toArray() returns an array that contains the elements of this extended queue in the same FIFO sequence as in this extended queue with the front item at index 0.

In the code evaluation, the following commands will be used

```
$ javac QTester.java
$ java QTester
$ javac EQTester.java
$ java EQTester
```

Where the QTester and EQTester test classes are below.

```
public class QTester {
   public static void main(String[] args) {
      // define constants
      String[] rivers = {"Lena","Ganges","Mekong","Danube","Orinoco","Yukon","Don"};

      // define new queue
      Queue q = new Q();

      // insert items into new queue
      for (String s : rivers) q.enqueue(s);

      // print size
      System.out.println("SIZE " + q.size());

      // print items
      System.out.print("ITEMS: ");
      int size = q.size();
      for (int i = 0; i < size; i++) {
         String item = (String)q.dequeue();
         System.out.print(item + " ");
         q.enqueue(item);
      }
      System.out.println();

      // get max item
      String maxItem = (String)q.maxItem();
      System.out.println("MAXITEM: " + maxItem);

      // remove elements
      System.out.print("FRONT/REMOVE ");
      while (!q.isEmpty()) System.out.print(q.front() + "/" + q.dequeue() + " ");
      System.out.println();

      // print size
      System.out.println("SIZE " + q.size());
   }
}
public class EQTester {
   public static void main(String[] args) {
      // constants
      String[] rivers = {"Lena","Ganges","Mekong","Danube","Orinoco","Yukon","Don"};

      // define new empty queue
      EQ q1 = new EQ();
```

```
      // insert items into new queue
      for (String s : rivers) q1.enqueue(s);

      // define new queue non-empty queue
      EQ q = new EQ(q1);

      // print size
      System.out.println("SIZE " + q.size());

      // print items
      Object[] items = q.toArray();
      System.out.print("ITEMS BEFORE: ");
      for (Object e : items) System.out.print(e + " ");
      System.out.println();

      // get max item
      String max = (String)q.maxItem();
      System.out.println("MAX ITEM: " + max);

      items = q.toArray();
      System.out.print("ITEMS AFTER: ");
      for (Object e : items) System.out.print(e + " ");
      System.out.println();

      // print size
      System.out.println("SIZE " + q.size());
   }
}
```

The OKQ.txt and OKEQ.txt Text files display printouts on standard output when the QTester and EQTester classes are run.

The following scores will be assigned in the code evaluation:

Q: contains 3, dequeue 6, enqueue 6, front 6, maxItem 3 (TOT 24),

EQ: EQ(EQ q) 1, recursive maxItem 2, toArray 2, (TOT 5)

one point can be assigned for the style and two points for the time complexity of the queue primitives (enqueue/dequeue) when their asymptotic time complexity is O(1) in the average case.

Code that reports compile-time errors is usually considered insufficient.

In the submitted code, the student may not

- use instance variables with access specifiers other than private
- add methods other than private
- use classes from the Java Platform API of packages other than java.lang.

Student files must contain a Java comment as the first line that includes the student's first and last name, ID, date, and workstation number.

FOUNDATIONS OF COMPUTER SCIENCE

Name_____ID._____

Date _____Taliercio work station No._____

Undersign one of the following statements:

□ I submit the following files for evaluation:

( )Q.java ( ) EQ.java ( ) other_____

□ I withdraw from the programming test.

Signature_____