PRG – ETSInf – THEORY – Academic year 2013/2014
Second Partial Exam – June 10th, 2014 – Duration: 2 hours

1. [2 points] Given a text file, we need to show its contents in the standard output line by line and converted to capital letters.

   We ask you to write a method with the following profile:

   ```
   public static void showInUpperCase( String fileName ) ...
   ```

   this method has to write in the standard output, line by line, the contents of the file whose name is provided as parameter. All characters in the file should appear in uppercase.

   In the case that the file name provided in the parameter `fileName` doesn't exist, the method should propagate exceptions of the class `FileNotFoundException`.

   NOTE: Remember that the method `toUpperCase()` of the class `String` returns a copy of the string with all the letters converted to uppercase.

   Solution:

   ```
   public static void showInUpperCase( String fileName ) throws FileNotFoundException
   {
       Scanner sf = new Scanner( new File( fileName ) );
       while( sf.hasNextLine() ) System.out.println( sf.nextLine().toUpperCase() );
       sf.close();
   }
   ```

2. [1 point] Given a `stack` (an object of the class `StackIntLinked`), you have to write a method for showing in the standard output the value at top. If the stack is empty an error message should be shown, but you cannot use methods `size()` or `isEmpty()` for checking if the stack is empty. These restrictions imply you have to catch exceptions of the class `NoSuchElementException` that methods `top()` and `pop()` throw when they are executed over an empty stack.

   Solution:

   ```
   static void showTopValue( StackIntLinked stack )
   {
       try{
           System.out.println( stack.top() );
       }
       catch( NoSuchElementException ex )
       {
           System.err.println( "ERROR: empty stack!" );
       }
   }
   ```

3. [2 points] We are implementing a class named `ExamForStructs` that imports the package `linear` and want to add a new static method called `incrementEvenNumbers()`. The method should have as parameter an object of the class `QueueIntLinked` and should return a new object of the same class where even numbers must be converted to odd numbers by means of adding one.

   If the contents of the input queue is:

```
<- 5  10  14  13  22  31 <-
```

then the contents of the new queue returned by the method should be:

```
<- 5  11  15  13  23  31 <-
```

NOTE: You can not use additional data structures, only the input queue received as parameter and the output queue created within the method can be used.

a) (1.75 points) Implement the method `incrementEvenNumbers()`.

b) (0.25 points) Modify the method in order to use queues whose internal representation is by means of arrays.

Solution:

```
a) An iterative solution (among others) is as follows:
  public static QueueIntLinked incrementEvenNumbers(QueueIntLinked q){
      QueueIntLinked q1=new QueueIntLinked();
      int n=q.size(), cont=0;
      for (int i=0; i<n; i++){
          if (q.first()%2==0) q1.enqueue(q.dequeue()+1);
          else                q1.enqueue(q.dequeue());
      }
      return q1;
  }

b) The solution is just to change the class name QueueIntLinked to QueueIntArray
in the profile of the method and in the declaration/creation of variable q1
```

4. 3 points We need to modify the behaviour of the `enqueue(int)` operation in class `QueueIntLinked`, in such a way that stored values are sorted in ascending order, i.e., the **first** value in the queue is the lowest one and the **last** value is the greatest one.

We ask you to implement a new version of the `enqueue(int)` method of the class `QueueIntLinked` in order to follow the previous definition.

Solution:

```
/** First possible version for en-queueing integers in ascending order. */
public void enqueue_v1( int value )
{
    NodeInt q = null;
    NodeInt p = first;

    while( p != null && value > p.getValue() ) { q = p; p = p.getNext(); }

    NodeInt node = new NodeInt( value );
    if ( q == null ) {
        node.setNext( first );
```

```
            first = node;
        } else {
            node.setNext( p );
            q.setNext( node );
        }

        if ( p == null ) last = node;
        size++;
    }

    /** Second possible version for en-queueing integers in ascending order. */
    public void enqueue_v2( int value )
    {
        NodeInt q = null;
        NodeInt p = first;

        while( p != null && value > p.getValue() ) { q = p; p = p.getNext(); }

        NodeInt node;
        if ( q == null )
            first = node = new NodeInt( value, first );
        else
            q.setNext( node = new NodeInt( value, p ) );

        if ( p == null ) last = node;
        size++;
    }
```

5. 2 points We need to implement a method named `count()` such that:

   - it receives as parameters a `list` as an object of the class `ListIPIntLinked` and two integer values `i` and `j`. It is assumed that both values define a range from `i` up to `j`.
   - It should return the number of values stored in the `list` belonging to range $[i, j]$.

   Suppose that the method is written in a different class to `ListIPIntLinked`.

Solution:

```
public static int count( ListIPIntLinked list, int i, int j )
{
    int counter = 0;
    list.begin();
    while ( !list.atTheEnd() ) {
        int x = list.get();
        if ( i <= x && x <= j ) counter++;
        list.next();
    }
    return counter;
}
```