PRG – ETSInf. THEORY. Academic Year 2015-220016. Second partial exam recovery.
GII – GIINF – GADE
June 17th, 2016. Duration: 2 hours.

1. 1.5 points Let `lS` be an array of objects of the class `String` where each string contains a floating-point value. If all the elements in the array contain the representation of a `double` as a string of digits, then the following code shows the contents of the array:

```
public static void m1( String [] lS )
{
    for( int i = 0; i < lS.length; i++ ) {
        System.out.print( "Pos: " + i + ": " );
        if ( lS[i].length() > 0 ) {
            double value = Double.parseDouble( lS[i] );
            System.out.println( "Value: " + value );
        } else {
            System.out.println( "Empty string." );
        }
    }
}
```

But if any of the strings does not exist or contains a value that is not a valid representation of a `double`, then some exceptions can be thrown. In particular `NullPointerException` or `NumberFormatException`.

Well, what we need is to get the output **without exceptions**. For instance, if the contents of the array is {`"1234.0"`, `"1.23456789E8"`, `null`, `"123xx9"`, `null`, `""` }, what we want is an output as follows:

```
Pos: 0: Value: 1234.0
Pos: 1: Value: 1.23456789E8
Pos: 2: Non existing string.
Pos: 3: Not a valid number.
Pos: 4: Non existing string.
Pos: 5: Empty string.
```

It is requested: to rewrite the method `m1()` for catching the above mentioned type of exceptions and get the desired output with no exceptions.

Solution:

```
public static void m1( String [] lS )
{
    for( int i = 0; i < lS.length; i++ ) {
        System.out.print( "Pos: " + i + ": " );
        try {
            if ( lS[i].length() > 0 ) {
                double value = Double.parseDouble( lS[i] );
                System.out.println( "Value: " + value );
            } else {
                System.out.println( "Empty string." );
            }
        }
        catch( NullPointerException nP ) {
            System.out.println( "Non existing string." );
        }
        catch( NumberFormatException nF ) {
            System.out.println( "Not a valid number." );
```

```
            }
        }
    }
```

2. 2.5 points **It is requested** to implement an static method for copying the elements stored in a stack, one per line, in a text file named "`stack-contents.txt`". The stack is an object of the class `StackIntLinked`. The method should receive as a parameter the stack. Finally, the two most important details: (1) **the numbers should be written to the file in the save order they were pushed in the stack**, and (2) **the stack must be in the original state at the end of the execution**.

   If the contents of the stack is $\{1, 2, 3, 4\}$ where 1 is the value at the top, the values must appear in the file in the following order $4, 3, 2, 1$.

   The method should return as a result an object of the class `File` referencing the file created on disk.

   The method also should catch exceptions of the class `FileNotFoundException` and show the proper error message in the case an exception of this class be thrown.

Solution:

```
public static File fromStackToFile( StackIntLinked p )
{
    StackIntLinked temp = new StackIntLinked();
    File file = new File( "stack-contents.txt" );
    try {
        PrintWriter pw = new PrintWriter( file );
        while( !p.isEmpty() ) { temp.push( p.pop() ); }
        while( !temp.isEmpty() ) {
            p.push( aux.pop() );
            pw.println( p.top() );
        }
        pw.close();
    }
    catch( FileNotFoundException e ) {
        System.out.println( "It was not possible to create the file." );
    }
    return res;
}
```

3. 3 points **It is requested** to add a new method with the following profile to the class `QueueIntLinked`

```
public void moveToBack( int x )
```

   such that:

   - searches the first occurrence of `x` in the queue and, if it is found, moves the element `x` to the end of the queue.
   - If `x` does not exist in the queue then the method should no modify the queue.

   Notice: It is only allowed to use the attributes of the class. It is explicitly forbidden to use the methods of the class `QueueIntLinked`. And it is also explicitly forbidden to use auxiliary data structures, even the use of arrays.

Solution: Version with single linked sequences.

```
/** If x is in the queue, the x is moved to the last position in the queue. */
public void moveToBack( int x )
{
    NodeInt temp = first, previous = null;

    while( temp != null  &&  temp.getValue() != x ) {
        previous = temp;
        temp = temp.getNext();
    }

    if ( temp != null  &&  temp != last ) {
        if ( temp == first) {
            first = first.getNext();
        } else {
            previous.getNext() = temp.getNext();
        }

        last.setNext( temp );
        temp.setNext( null );

        last = temp;
    }
}
```

Version with double linked sequences.

```
/** If x is in the queue, the x is moved to the last position in the queue. */
public void moveToBack( int x )
{
    NodeInt temp = first;

    // Searching
    while( temp != null && temp.getValue() != x ) temp = temp.getNext();

    // Moving to the last position
    if ( temp != null  &&  temp != last ) {
        if ( temp == first ) {
            first = first.getNext();
            first.setPrevious( null );
        } else {
            temp.getPrevious().setNext( temp.getNext() );
            temp.getNext().setPrevious( temp.getPrevious() );
        }

        temp.setPrevious( last );
        temp.setNext( null );
        last.setNext( temp );
        last = temp;
    }
}
```

4. 3 points **It is requested** to implement a method, in a class distinct of `ListIntLinked`, with the following profile and considering the precondition that the given lists have no duplicated items.

```
/** Precondition: list1 and list2 have no duplicated items. */
public static ListIntLinked difference( ListIntLinked list1,  ListIntLinked list2 )
```

The method should return a list with the elements of `list1` that are not contained in `list2`.

For instance, if the contents of `list1` is

$$\rightarrow 7 \rightarrow 3 \rightarrow 9 \rightarrow 6 \rightarrow 2$$

and the contents of `list2` is

$$\rightarrow 8 \rightarrow 9 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4$$

then the result of executing `difference( list1, list2 )` should be a list (an object of the class `ListIntLinked`) containing the elements

$$\rightarrow 7 \rightarrow 6$$

Solution:

```
/** Precondition: list1 and list2 have no duplicated items. */
    public static ListIntLinked difference( ListIntLinked list1, ListIntLinked list2 )
    {
        ListIntLinked result = new ListIntLinked();
        list1.begin();
        while( list1.isValid() ) {

            int x = list1.get();

            list2.begin();

            while( list2.isValid()  &&  x != list2.get() ) { list2.next(); }

            if ( ! list2.isValid() ) { result.insert(x); }

            list1.next();
        }
        return result;
    }
```