PRG – ETSINF – THEORY – Academic year 2015-2016
Second partial exam – May 31st, 2016 – Duration: 2 hours

1. 1.5 points Let lS be an array of String objects with some positions set to null and other positions containing references to correct objects of the class String.

   If we use the following method for showing on screen the length of all the existing objects of the class String in lS, several exceptions can be thrown, in particular ArrayIndexOutOfBoundsException and NullPointerException.

```
public static void m1( String[] lS )
{
    int k = 0;
    boolean finished = false;
    while( !finished ) {
        System.out.print( "Position " + k + ": ");
        System.out.println( lS[k].length() + " characters" );
        k++;
    }
}
```

   But what we would like is to get the correct output **without exceptions**, as in the following example for an array with 6 positions and 4 references to correct objects of the class String:

```
Position 0: 4 characters
Position 1: 9 characters
Position 2: non existing String
Position 3: 11 characters
Position 4: non existing String
Position 5: 0 characters
Position 6: non existing String, end of array reached.
```

   It is requested to rewrite the method m1() to catch exceptions of the both above-mentioned classes, without using the attribute length of arrays nor the constant null. The new version of the method must solve the problem and its output should be as in the above sample.

Solution:

Two alternative solutions:

```
// First solution:
public static void m1( String [] lS )
{
    int k = 0;
    boolean finished = false;
    try {
        while( !finished ) {
            System.out.print( "Position " + k + ": " );
            try {
                System.out.println( lS[k].length() + " characters" );
            }
            catch( NullPointerException npe ) {
                System.out.println( "non exiting String" );
            }
            k++;
        }
    }
```

```
        catch( ArrayIndexOutOfBoundsException io ) {
            System.out.println( "non existing String, end of array reached." );
        }
    }

    // Second solution:
    public static void m1( String [] lS )
    {
        int k = 0;
        boolean finished = false;
        while( !finished ) {
            System.out.print("Position " + k + ": " );
            try {
                System.out.println( lS[k].length() + " characters" );
            }
            catch( NullPointerException npe ) {
                System.out.println( "non existing String" );
            }
            catch( ArrayIndexOutOfBoundsException io ) {
                System.out.println( "non existing String, end of array reached." );
                finished = true;
            }
            k++;
        }
    }
```

2. 2.5 points **It is requested** to implement an static method according to the following description:

- The method should:
  - receive as a parameter an object of the class `String` with the path of a file.
  - propagate exceptions of the class `FileNotFoundException`. This kind of exception can be thrown when it is not possible to open the file whose file name has been provided as a parameter.
  - create and return an object of the class `ListIntLinked` with all the integer numbers contained in the file.
- The text file can contain tokens that do not correspond to integer numbers.
  It is unknown the number of tokens which are incorrect integer numbers.
  The number of tokens per line is variable.
- All the correct integer numbers should be inserted into the list to be returned.
- When trying to process as an integer those tokens that do not correspond to integer numbers it can be thrown an exception of the class `InputMismatchException`.
  Your solution should properly deal with this kind of exceptions by showing on screen a message including the name of the exception and the token that caused it.
  Wrong tokens should not impede reading all the correct integer numbers contained in the file.

Solution:

```
    public static ListIntLinked read( String filename )
        throws FileNotFoundException
    {
        ListIntLinked l = new ListIntLinked();
        Scanner sc = new Scanner( new File( filename ) );
        while( sc.hasNext() ) {
            try {
```

```
            l.insert( sc.nextInt() );
        }
        catch( InputMismatchException e ) {
            System.err.println( e + "::" + sc.next() );
        }
    }
    sc.close();
    return l;
}
```

3. 3 points When working with queues, it can be of interest to remove some elements which are not in the first position. Recall that removing the first element in a queue is just dequeuing.

But we need a special method for removing elements which are not in the first position.

It is requested to add a new method to the class `QueueIntLinked` with the following profile:

```
public int dequeue( int x )
```

that will remove from the queue and return the first occurrence of `x` in the queue.

If the queue is empty the method should throw an exception of the class `NoSuchElementException` with the message `"Empty queue!"`. If `x` is not in the queue then the method should throw an exception of the same class with the message `"x is not contained in the queue!"`. `x` in the message should be the value of the parameter `x` of the method.

It is not allowed to use the methods of the class `QueueIntLinked` in the solution.

Solution:

```
// Solution for single linked sequences
public int dequeue( int x )
{
    if ( 0 == this.size ) throw new NoSuchElementException( "Empty queue!" );

    NodeInt previous = null, current = this.first;

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

    if ( null == current )
        throw new NoSuchElementException( x + " is not contained in the queue!" );

    if ( current == this.first ) {
        this.first = current.getNext();
    } else {
        previous.setNext( current.getNext() );
    }
    if ( current == this.last ) {
        this.last = previous;
    }

    --this.size;

    return x;
}
```

```
    // Solution for double linked sequences
    public int dequeue( int x )
    {
        if ( 0 == this.size ) throw new NoSuchElementException( "Empty queue!" );

        NodeInt current = this.first;

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

        if ( null == current )
            throw new NoSuchElementException( x + " is not contained in the queue!" );

        if ( current == this.first ) {
            this.first = current.getNext();
            this.first.setPrevious( null );
        } else if ( current == this.last ) {
            this.last = current.getPrevious();
            this.last.setNext( null );
        } else {
            current.getPrevious().setNext( current.getNext() );
            current.getNext().setPrevious( current.getPrevious() );
        }

        --this.size;

        return x;
    }
```

4. $\boxed{\text{3 points}}$ **It is requested** to implement a method with the following profile in a class different of `ListIntLinked`.

```
/** Precondition: list1 and list2 have no repeated elements */
public static ListIntLinked removeFromList1ElementsInList2( ListIntLinked list1,
                                                            ListIntLinked list2 )
```

The method should return a list with those elements contained in both lists and removing them from `list1`.

Example: if the contents of `list1` is $\{6, -5, 4, 8, -9\}$, and the contents of `list2` is $\{21, 8, 5, -9, -5, 16\}$, then the result of invoking `removeFromList1ElementsInList2( list1, list 2)` should be a list with $\{-5, 8, -9\}$ and the contents of `list1` should be reduced to $\{6, 4\}$.

Solution:

```
/** Precondition: list1 and list2 have no repeated elements */
public static ListIntLinked removeFromList1ElementsInList2( ListIntLinked list1,
                                                            ListIntLinked list2 )
{
    ListIntLinked result = new ListIntLinked();

    list1.begin();
    while( list1.isValid() ) {

        int x = list1.get();
        list2.begin();
        while( list2.isValid()  &&  list2.get() != x ) list2.next();

        if ( ! list2.isValid() ) {
            list1.next();
        } else {
            list1.remove();
            result.insert(x);
        }
    }
    return result;
}
```