Computer Programming – Theory – ETSINF – Academic year 2018/2019
Retake of second mid term exam – June 11^{th}, 2019 – duration two hours

Note: The maximum mark for this exam is 10 points, but its weight in the final grade of PRG is **3 points**.

1. 3 points The Association of Tennis Professionals (ATP) provided us with a file containing data about tennis players. Each line contains the data of one player. First column contains the last name, second column its age in years, third column the accumulated points, and the fourth column the number of championships the player participated during the current year. The file can contain errors because of (a) the information is not complete (missing data) or even there are more columns than four, or (b) age or points or number of championships are not integers o are negative values.

   You have to design and implement a method with two objects of the class String as parameters, the first parameter is `fileIn` and the second one is `fileOut`, with the names of input and output files respectively. The method **must** read all data contained in `fileIn` and show an error message for each line with missing or incorrect data. The message should provide information about the detected error. Additionally, the method must propagate all the possible exceptions of the class `FileNotFoundException` (remember that this class is one of the checked exceptions), that could be thrown if any error appears when trying to open the files. An example of input file is:

```
Djokovic    31  12115   17
Nadal       -32 7945    16
Federer     37  5770    17
Thiem       25  4845    24  4
Zverev      22  4745
Nishikori   29  3860    23.5
Tsitsipas   20  3790    28
Anderson    32  3755.3  17
```

   And the contents of the output file should be:

```
Error line 2: Negative value.
Error line 4: Unexpected number of columns.
Error line 5: Unexpected number of columns.
Error line 6: Invalid format for an integer.
Error line 8: Invalid format for an integer.
```

   If columns in each line can be delimited by tabs in addition to white spaces, then you can use the following instruction to split the lines and get all the columns in separated strings:

```
String [] tokens = line.split("([ \t])+");
```

   For instance, if `line` contains `"Djokovic 31 12115 17"`, the array `tokens` will contain `["Djokovic", "31", "12115", "17"]`.
   You can use the method `parseInt()` of the class `Integer` for converting the contents of an string to an integer. This method can thrown an exception of the class `NumberFormatException` if the string passed as parameter does not contain a sequence of characters with the format valid for integers.

Solution:

```
public static void checkForErrors( String fileIn, String fileOut )
    throws FileNotFoundException
{
    File fE = new File(fileIn);
    File fS = new File(fileOut);
    Scanner input = new Scanner(fE);
    PrintWriter output = new PrintWriter(fS);
    int cont = 0;
    while(input.hasNext() ) {
        try {
            String line = input.nextLine();  cont++;
```

```
            String[] tokens = line.split("([ \t])+");
            if ( tokens.length != 4 ) {
                output.println( "Error line " + cont + ": " + "Unexpected number of columns." );
            } else {
                int age = Integer.parseInt(tokens[1]);
                int points  = Integer.parseInt(tokens[2]);
                int championships  = Integer.parseInt(tokens[3]);
                if ( age < 0 || points < 0 || championships < 0 ) {
                    output.println( "Error line " + cont + ": " + "Negative value." );
                }
            }
        }
        catch( NumberFormatException e ) {
            output.println( "Error line " + cont + ": " + "Invalid format for an integer." );
        }
    }
    input.close();
    output.close();
}
```

2. 3.5 points Write a new non-static method in the class `ListIntLinked` with the following profile and precondition:

```
/** Precondition: list has the values stored in ascending order */
public void removeGreaterThan( int e )
```

Given an integer value `e` passed as parameter, the method modifies the list by removing all the values in the list greater than `e`. After completing the task, this method will set the cursor at the beginning, i.e. referencing the first element of the list.

Two examples taking as starting point a list `l` with this contents: `[10] 12, 14 15`

– First example: if we execute `l.removeGreaterThan(12)`, then the list will be `[10] 12`

– Second one: if we execute `l.removeGreaterThan(9)`, then the list will be empty.

MANDATORY: Only use the attributes of the class `ListIntLinked`, no other methods of this class. Methods of class `NodeInt` are allowed, but you can also use the attributes of the class `NodeInt`.

Solution:

```
public void removeGreaterThan( int e )
{
    if ( first != null && first.getValue() > e ) {
        first = last = null;
        size = 0;
    } else {
        int counter = 0;
        cursor = first;
        while( cursor != null && cursor.getValue() <= e ) {
            counter++;
            cursor = cursor.getNext();
        }
        if ( cursor != null ) {
            cursor.getPrevious().setNext(null);
            last = cursor;
            size = cont;
        }
    }
    cursor = first;
}
```

3. [3.5 points] Write an static method that can be in any class different from the class `QueueIntLinked`, with the following profile:

```
public static void moveBack( QueueIntLinked q, int x )
```

such that searches the first occurrence of `x` within the queue `q`,

- if it exists then the first occurrence of `x` in the queue is moved to the end of the queue.
- Otherwise the queue remains untouched.

MANDATORY: As stated before, this method will be implemented within a class different from `QueueIntLinked`, so, only the methods of this class can be used.

Solution:

```
public static void moveBack( QueueIntLinked q, int x )
{
    int n = q.size();
    int i = 0;
    while( i < n && q.element() != x ) {
        q.add( q.remove() );
        i++;
    }
    if ( i < n - 1 ) {
        x = q.remove();
        for( j = i + 1; j < n; j++ ) {
            q.add( q.remove() );
        }
        q.add(x);
    }
}
```

APPENDIX

Attributes of the class `ListIntLinked` and methods of the class `QueueIntLinked`.

```
public class ListIntLinked {
    private NodeInt first;
    private NodeInt last;
    private NodeInt cursor;
    private int size;
    ...
}

public class QueueIntLinked {
    ...
    public QueueIntLinked() { ... }
    public void add( int x ) { ... }
    public int remove() { ... }
    public int first() { ... }
    public boolean isEmpty() { ... }
    public int size() { ... }
}
```