Second quiz of Computer Programming (PRG – 11543) — ETSInf
Thursday June 7th, 2012
2 hours and 30 minutes

1. 1.5 points The following method asks the user for an integer value. The value is read as an `String`, then the `String` is converted into the integer value by using `Integer.parseInt()`.

```
public static int readInt() {
    Scanner keyboard = new Scanner( System.in );
    System.out.print( "Type an integer value: " );
    String str = keyboard.nextLine().trim();
    int value = Integer.parseInt( str );
    return value;
}
```

There is a problem, if the contents of the `String` is not an integer number properly written, then, a `NumberFormatException` can be thrown and the reading process aborts. For example if the `String` contains a letter.

**We ask you** to modify the method `readInt()` for preventing that the program be interrupted if a NumberFormatException is thrown during the conversion. In such a case the method has to ask again for the value, until the user types a correct integer number.

**Solution:**

```
public static int readInt() {
    Scanner keyboard = new Scanner(System.in);
    int value=0;
    boolean finish = false;
    while( !finish ) {
        try {
            System.out.print( "Type an integer value: " );
            String str = keyboard.nextLine().trim();
            value = Integer.parseInt( str );
            finish = true;
        }
        catch ( NumberFormatException e ) {
            System.out.println( "ERROR" );
        }
    }
    return value;
}
```

2. 1.5 points Given a text file named `origin.txt` and an integer value to be used as `threshold`, **we ask you** designing a method that creates a new file with the name `target.txt` and writes into it all the lines from `origin.txt` whose length is longer than or equal to the `threshold`.

If the file `origin.txt` was empty, then, the new file will be empty as well.

**Solution:**

```
public static void newFile(int threshold) throws IOException {
    String sin = "origin.txt"; String sout = "target.txt";

    Scanner fin = new Scanner( new File( sin ) );
    PrintWriter fout = new PrintWriter( new File( sout ) );
    while( fin.hasNextLine() ) {
        String aux = fin.nextLine();
        if (aux.length() >= threshold) fout.println(aux);
    }
    fin.close(); fout.close();
}
```

3. 3.0 points We need to add two new methods to class `ListIntLinked`. One constructor for creating a new linked list, whose contents must be the same numbers stored in the integer array passed as a parameter, and a method named `toArray()` for obtaining an array of integers as representation of the list. The order of the numbers must be maintained in both cases. **This new operations must be programmed without using previously existing methods in the class**.

   Thanks to these new methods we will be able to sort a list with point of interest by means of the following steps: (1) transforming the list into an array, (2) sorting the array by using one of the studied sorting methods, and (3) creating a new list from the sorted array.

   We ask you to solve the following tasks:

   1. Method `toArray()`, should return an array with the same integer values stored in the list and in the same order.
   2. Constructor `ListIntLinked( int [] a )`, that given an integer array creates a new list with the same integer values stored in the array and in the same order.

3. Suppose we already have available the class `ListIntLinked` with the methods described above, and also suppose we have available the class `CheckLists` with a quick method for sorting arrays, whose profile is the following:

   ```
   public static void quickSort( int[] a, int begin, int end )
   ```

   you have to write the sequence of instructions for sorting a list referenced by `list1`, a reference variable to objects of class `ListIntLinked`.

**Solution:**

```
public int [] toArray()
{
    int[] A = new int[ size ];
    int k; NodeInt p;
    for( p=first, k=0; p!=null; p=p.next, k++ ) A[k] = p.datum;
    return A;
}

public ListIntLinked( int[] a )
{
    this.first = null;
    for( int k=a.length-1; k>=0; k-- )
        first = new NodeInt( a[k], first );

    this.size = a.length;
    // The interest point at the beginning
    this.cursor = first; this.prevCursor = null;
}

// Sequence of instructions for sorting list1:
  int [] A = list1.toArray();
  quickSort( A, 0, A.length-1 );
  list1 = new ListIntLinked( A );
```

4. 1.5 points Starting from the implementation of class `StackIntLinked` that we studied in class, and making use of its internal data representation, that is, manipulating their private attributes, we ask you to write a method for swapping the value in the top of the stack with the value in the bottom of the stack.

   ```
   public void topBottom()
   ```

   It is a precondition that the stack be non empty.

**Solution:**

```
/** PRECONDITION: !isEmpty() */
public void topBottom() {
```

```
    NodeInt tmp = top;
    int e = tmp.datum;
    while( tmp.next != null ) tmp = tmp.next;
    tope.datum = tmp.datum;
    tmp.datum = e;
}
```

5. 2.5 points Given two lists with point of interest:

```
ListIntLinked   la, lb;
```

that are in ascending order and not contain duplicate values. **We ask you** to write a method named `intersection()` that returns a new list (an object of the same class), that contains the integer values that are contained in the input lists: `la` and `lb`. This new list should not contain duplicate values.

If `la` and `lb` have no values in common, then, the new list must be empty.

**Example:** If the contents of input lists are:

```
la = 2 4 6 7 9 11 33 45 67 112 129 310 516 555 610
lb = 8 11 22 33 44 45 46 112 113
```

The result list must contain:

```
11 33 45 112
```

**Solution:**

```
/** PRECONDITION: la and lb are in ascending order
                  and without duplicate values */
public static ListIntLinked intersection( ListIntLinked la,
                                          ListIntLinked lb )
{
    ListIntLinked lc = new ListIntLinked();
    la.begin(); lb.begin();
    while( !la.atTheEnd() && !lb.atTheEnd() ) {
        int a = la.get(); int b = lb.get();
        if ( a < b ) la.next();
        else if ( b < a ) lb.next();
        else {
            // This insert operation puts the new value after the cursor
            lc.insert(a);
            la.next(); lb.next();
        }
    }
    return lc;
}
```