Recovering Second Partial Exam - Theory - PRG - ETSInf
Academic year 2012/2013 – June 17th, 2013
Duration: 1 hour and 50 minutes.

1. $\boxed{\text{2.5 points}}$ Given two `String` objects `fileName` and `word`, you have to implement a static method for copying all the lines from the input file (`fileName`) containing the `word` into a file named *result.txt*, the output file. Lines in the output file should appear after the number of line they are in the input file. The method `should catch exceptions` of the class `FileNotFoundException`. A message must be printed to standard output if an exception is thrown.

   NOTE: You can use the method `contains()` of the class `String` in your solution.

   `s1.contains(s2)` returns `true` if `s1` contains `s2` as a substring, otherwise it returns `false`.

   Let us see an example of input file and output file if the word used is `ratón`. Input file:

```
El ratón de Federico
no tiene patas ni hocico.
Es ratón de ordenador,
de un ordenador muy listo.
```

   The output file (*result.txt*) would be:

```
1  El ratón de Federico
3  Es ratón de ordenador,
```

Solution:

```
import java.io.*;
import java.util.*;
public class Exercise1
{
    public static void copy( String fileName, String word )
    {
        try {
            Scanner s = new Scanner( new File( fileName ) );
            PrintWriter pw = new PrintWriter( new File( "result.txt" ) );
            int counter = 0;
            while( s.hasNext() ) {
                String line = s.nextLine();
                counter++;
                if ( line.contains( word ) ) pw.println( counter + "  " + line );
            }
            s.close();
            pw.close();
        }
        catch( FileNotFoundException e ) {
            System.out.println( "File not found!" );
        }
    }
}
```

2. \boxed{2.5 points} Let `SortedListIPIntLinked` be a class very similar to the class `ListIPIntLinked` that we studied in the unit 5. The objects of the class `SortedListIPIntLinked` are lists where integer numbers are stored in strictly increasing order. The constructor and all the methods of this new class behave like the ones of the known class, except the method `insert()`.

   To be done: implement the method `insert()` for the new class. **You can't use the existing methods of the class, you have to manage the references**.

   ```
   public void insert( int x )
   ```

   The `insert()` method for the new class must insert new values in the correct position in order to maintain the list sorted. The interest point must be placed to the right of the last inserted element.

   Solution:

   ```
   public void insert( int x )
   {
       if ( 0 == this.size ) {
           this.first = this.last = new NodeInt( x );
           this.size = 1;
           this.current = null;
       } else {
           this.current = this.first;
           while( this.current != null && this.current.datum < x ) {
               this.current = this.current.next;
           }
           if ( this.current == null || this.current.datum > x ) {
               NodeInt newItem = new NodeInt( x );
               if ( this.current == null ) {
                   this.last        = newItem;
                   newItem.previous = this.last;
                   this.last         = newItem;
               } else if ( this.current == this.first ) {
                   newItem.next        = this.first;
                   this.first.previous = newItem;
                   this.first          = newItem;
               } else {
                   newItem.previous = this.current.previous;
                   newItem.next     = this.current;
                   this.current.previous.next = newItem;
                   this.current.previous      = newItem;
               }
               this.size++;
           }
       }
   }
   ```

3. \boxed{2.5 points} Let `l1` be a list with interest point, a list of the class `ListIPIntLinked`, and let `x` be an integer number, you have to implement a static method `remove()` that returns a new list of the same class with all the elements in `l1` equal to `x`. These values must be removed from the list `l1`.

Let us see an example, if the list `l1` contains the values { `2, 3, 4, 3, 7, 5, 3` }, then, after executing `ListIPIntLinked l2 = remove( l1, 3 );` the created list `l2` will contain { `3, 3, 3` }, and the old list `l1` will contain { `2, 4, 7, 5` }.

You can only use the public methods of the class `ListIPIntLinked`.

Solution:

```
public static ListIPIntLinked remove( ListIPIntLinked l, int x )
{
    ListIPIntLinked l_aux = new ListIPIntLinked();
    l.begin();
    while( !l.atTheEnd() ) {
        if ( l.get() == x ) {
            l_aux.insert( x );
            l.remove();
        } else
            l.next();
    }
    return l_aux;
}
```

4. $\boxed{\text{2.5 points}}$ Given a stack `s` of the class `StackIntLinked` and given an integer value `x`, you have to implement a **static and recursive** method with the following profile:

```
public static void removeLowerThan( StackIntLinked s, int x )
```

The method should remove from `s` those elements lower than `x`. For example, if the list contains { `3, 6, 7, 2, 5, 4` }, after the call `removeLowerThan( s, 5 );` the stack will contain the values { `6, 7, 5` }, as you can see in the following representation:

```
 's' before the execution                             's' after the execution
         3
         6
         7
         2          ==== removeLowerThan(p,5) ====>          6
         5                                                   7
         4                                                   5
        ---                                                 ---
```

Only the public methods of the class `StackIntLinked` **can be used**.

Solution:

```
private static void removeLowerThan( StackIntLinked s, int x )
{
    if ( !p.isEmpty() ) {
        int aux = p.pop();
        removeLowerThan( s, x );
        if ( aux >= x ) p.push(aux);
    }
}
```