Recovering of second quiz – PRG – ETSInf
Date: June 19th, 2012 – 2 hours

1. (1.5 points) `cat` is a command line utility of Unix systems which concatenates files and prints them on standard output. The contents of files is printed in the same order that file names appear as arguments. If a given file does not exist, `cat` prints an error message on standard error. As an example, if we run the `cat` command as follows:

```
cat Hola.java Adios.java Result.txt
```

when the file `Adios.java` doesn't exist, `cat` command prints on screen the following contents:

```
class Hola {
    public static void main(String[] args) {
        System.out.println("Hola a todos");
    }
}
cat: Adios.java: File or directory not found
Hola a todos
```

The version of the this class which is shown below is incomplete because of some possible exceptions that are not caught or propagated.

```
import java.util.*;
import java.io.*;
public class Cat {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++) {
            Scanner sf = new Scanner(new File(args[i]));
            while(sf.hasNext())
                System.out.println(sf.nextLine());
            sf.close();
        }
    }
}
```

You have to improve this version in order to obtain the same behavior of the Unix command `cat`. **We ask you** to rewrite this class for catching the exception `FileNotFoundException` inside the method `main()` by means of `try-catch`. This exception can be thrown by the constructor of `Scanner` class. Any other exception must be propagated.

Solution:

```
import java.util.*;
import java.io.*;
public class Cat
{
    public static void main(String[] args)
        throws Exception
```

```
    {
        for( int i=0; i<args.length; i++ ) {
            try {
                Scanner sf = new Scanner( new File( args[i] ) );
                while( sf.hasNext() )
                    System.out.println( sf.nextLine() );
                sf.close();
            }
            catch( FileNotFoundException fnfe )
            {
                System.err.println( "cat: " + args[i] +
                        ": file or directory not found" );
            }
        }
    }
}
```

2. (1.5 points) Given a text file named `origin.txt`, and a word (an object of `String` class), `we ask you` the design of a method for creating a new file named `target.txt` that must contain all the lines from `origin.txt` starting with the specified `word`. The lines saved into `target.txt` must be written in the same order they appear in `origin.txt`.

   If the file `origin.txt` was empty then the new file `target.txt` must be created empty.

   Notice: You can use the method `startsWith( String )`, defined in the class `String`, with the following profile:

   ```
   public boolean startsWith( String str )
   ```

   This method returns `true` if the current string begins with the string contained in `str`, the argument of the method, and returns `false` otherwise.

Solution:

```
public static void filter( String word )
    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.startsWith( word ) ) fout.println( aux );
    }
    fin.close(); fout.close();
}
```

3. (1.5 points) Considering the implementation of classes `NodeInt` and `StackIntLinked` explained and used in class. What prints on screen the following program?

```
public class Stacks {
    public static void main(String[] args)
    {
        StackIntLinked p1 = new StackIntLinked();
        for( int i=1; i<=10; i++ ) p1.push(i);
        StackIntLinked p2 = new StackIntLinked();
        while(!p1.isEmpty()) {
            int value = p1.pop();
            if ( value%2 == 0 )
                p2.push( value );
            else
                System.out.print( " " + value );
        }
        while( !p2.isEmpty() )
            System.out.print( " " + p2.pop() );
    }
}
```

Solution: 9 7 5 3 1 2 4 6 8 10

4. (3 points) For representing words as linked sequences of letters (`char` variables), we suppose that class `NodeChar` is already implemented. This class is similar to `NodeInt` class, but for storing values of type `char` instead of type `int`, i.e., characters instead of integer numbers. The attributes of `NodeChar` class are `char value` and `NodeChar next`. This class also has implemented the common methods: constructors, getter methods, setter methods and the `toString()` method. **We ask you** to write an static method named `correct()` inside a class included in the same packaged that class `NodeChar` and with the following profile:

```
public static NodeChar correct( NodeChar p )
```

Remember! When the class is defined in the same package of class `NodeChar`, and the attributes of class `NodeChar` are declared as *friendly*, these attributes can be accessed from the new designed class.

The method `correct( NodeChar p )` must correct the word represented by a linked sequence of objects of class `NodeChar`. The corrections to be made are the following: the subsequence of two symbols `n` `y` must be converted into the symbol `ñ`. This means substituting the symbol `n` by the symbol `ñ` and removing the symbol `y`. For example, words "cuca**ny**a" and "**ny**o**ny**ería" will be translated into "cuca**ñ**a" and "**ñ**o**ñ**ería" respectively.

Solution:

```
/** The word has one character at least */
public static NodeChar correct( NodeChar p )
{
    NodeChar aux = p.next, ant = p;
    while( aux != null ) {
```

```
        if ( ant.value == 'n' && aux.value = 'y' ) {
            ant.value = 'ñ';
            ant.next = aux.next;
            aux = ant.next;
        } else {
            ant = aux;
            aux = aux.next;
        }
    }
    return p;
}
```

5. (2.5 points) Given a list with point of interest `ListPIIntLinked l`, **we ask you** to write a static method `void removeNegatives( ListPIIntLinked l )` for removing the negative values stored in the list `l` given as a parameter. You must use the public methods already implemented in the class `ListPIIntLinked`. You are not allowed to use the internal representation of class `ListPIIntLinked` for solving this problem.

   Example: If `l = 3 -2 5 -7 -8 1 -10`, after running `removeNegatives( l )` the contents of `l` must be `3 5 1`.

Solution:

```
public static void removeNegatives( ListPIIntLinked l )
{
    l.begin();
    while( !l.atTheEnd() ) {
        if ( l.get() < 0 )
            l.remove();
        else
            l.next();
    }
}
```