Recuperación Segundo Parcial de PRG - ETSInf
Fecha: 19 de junio de 2012. Duración: 2 horas

1. (1.5 puntos) El comando `cat` de los sistemas Unix escribe en la salida estándar el contenido de cada uno de los ficheros dados como argumentos, en el mismo orden en el que fueron dados. Si alguno de los ficheros especificados no existe, muestra un mensaje en la salida de error. Por ejemplo, la ejecución del comando `cat Hola.java Adios.java Result.txt`, sabiendo que el fichero `Adios.java` no existe, muestra por pantalla:

```
class Hola {
    public static void main(String[] args) {
        System.out.println("Hola a todos");
    }
}
cat: Adios.java: No existe el fichero o el directorio
Hola a todos
```

La clase `Cat` que se muestra a continuación está incompleta en lo que respecta a captura o propagación de excepciones. Se desea que dicha clase tenga un comportamiento similar al comando `cat`.

```
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();
        }
    }
}
```

Se pide reescribir esta clase para que se capture dentro del método `main` mediante `try-catch` la excepción `FileNotFoundException`, que sabemos puede ser lanzada por el constructor de la clase `Scanner`. Cualquier otra excepción debe propagarse.

Solución:

```
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] +
                                   ": no existe el fichero o el directorio");
            }
    }
}
```

2. (1.5 puntos) Dado cierto fichero de texto llamado en el sistema "origen.txt", así como cierto `String palabra`, **se pide** diseñar un método que escriba en el sistema un nuevo fichero de texto "destino.txt"que contenga exclusivamente y en el mismo orden de aparición, todas las líneas de texto del fichero original que empiecen por `palabra`.

   Si el fichero origen estuviera vacío el nuevo fichero generado también deberá estar vacío.

   Nota: Se puede utilizar el método `startsWith(String)`, definido en la clase `String`, con el siguiente perfil:

   ```
   public boolean startsWith(String cad)
   ```

   que devuelve `true` si el `String` actual comienza por `cad`. En caso contrario, devuelve `false`.

Solución:

```
public static void nuevoFichero(String palabra) throws IOException {
    String sin = "origen.txt"; String sout = "destino.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(palabra)) fout.println(aux);
    }
    fin.close(); fout.close();
}
```

3. (1.5 puntos) Considerando la implementación de las clases `NodoInt` y `PilaIntEnla` explicadas en clase, ¿qué muestra por pantalla el siguiente programa?

```
public class Pilas {
    public static void main(String[] args) {
        PilaIntEnla p1 = new PilaIntEnla();
        for(int i=1; i<=10; i++) p1.apilar(i);
        PilaIntEnla p2 = new PilaIntEnla();
        while(!p1.vacia()) {
            int valor = p1.desapilar();
            if (valor%2==0) p2.apilar(valor);
            else System.out.print(" " + valor);
        }
        while(!p2.vacia())
            System.out.print(" " + p2.desapilar());
    }
}
```

Solución: 9 7 5 3 1 2 4 6 8 10

4. (3 puntos) Para representar palabras como secuencias enlazadas de valores de tipo `char`, se supone ya implementada la clase `NodoChar` (análoga a la clase `NodoInt` vista en clase), con atributos `dato` de tipo `char` y `siguiente` de tipo `NodoChar`, y con las dos operaciones constructoras habituales definidas en este tipo de clase. **Se pide** escribir un método estático `corregir`, en una clase incluida en el mismo paquete que la clase `NodoChar`, con el siguiente perfil:

```
public static NodoChar corregir(NodoChar p)
```

que, dada una palabra `p` (de tipo `NodoChar`, con al menos 1 carácter), la corrija sustituyendo todas las ocurrencias de la pareja de caracteres consecutivos **'n' 'y'** por el carácter **'ñ'**. Por ejemplo, las palabras "cuca**ny**a" y "**ny**o**ny**ería", serían "cuca**ñ**a" y "**ñ**o**ñ**ería".

Solución:

```
/** la palabra tiene al menos 1 carácter */
public static NodoChar corregir(NodoChar p) {
    NodoChar aux = p.siguiente, ant = p;
    while(aux!=null) {
        if (ant.dato=='n' && aux.dato = 'y') {
            ant.dato = 'ñ';
            ant.siguiente = aux.siguiente;
            aux = ant.siguiente;
        }
        else {
            ant = aux;
            aux = aux.siguiente;
        }
    }
    return p;
}
```

5. (2.5 puntos) Dada una lista con punto de interés `ListaPIIntEnla l`, **se pide** escribir un método estático `eliminarNeg` que elimine los valores negativos de dicha lista, haciendo uso exclusivamente de las operaciones públicas definidas en la clase `ListaPIIntEnla`, sin acceder a su representación interna.

   Ejemplo: Si inicialmente se tiene que `l = 3 -2 5 -7 -8 1 -10`, como efecto de la ejecución del método que se pide, la lista resultante será `l = 3 5 1`.

Solución:

```
public static void eliminarNeg(ListaPIIntEnla l) {
    l.inicio();
    while(!l.fin()) {
        if (l.recuperar()<0)
            l.eliminar();
        else l.siguiente();
    }
}
```