Recuperació Segon Parcial de PRG - ETSInf
Data: 19 de juny de 2012. Duració: 2 hores

1. (1.5 punts) El comandament `cat` dels sistemes Unix escriu en l'eixida estàndard el contingut de cadascun dels fitxers donats com arguments, en el mateix ordre en què van ser donats. Si algun dels fitxers especificats no existeix, mostra un missatge en l'eixida d'error. Per exemple, l'execució del comandament `cat Hola.java Adeu.java Result.txt`, sabent que el fitxer `Adeu.java` no existeix, mostra per pantalla:

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

La classe `Cat` que es mostra a continuació està incompleta pel que fa a captura o propagació d'excepcions. Es desitja que aquesta classe tinga un comportament similar al comandament `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();
        }
    }
}
```

Es demana reescriure aquesta classe perquè es capture dintre del mètode `main` mitjançant `try-catch` l'excepció `FileNotFoundException`, que, com és conegut, pot ser llançada pel constructor de la classe `Scanner`. Qualsevol altra excepció cal que es propague.

Solució:

```
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 punts) Donat cert fitxer de text anomenat en el sistema "origen.txt", així com cert `String paraula`, **es demana** dissenyar un mètode que escriga en el sistema un nou fitxer de text "desti.txt"que continga exclusivament i amb el mateix orde d'aparició, totes les línies de text del fitxer original que comencen per `paraula`.

   Si el fitxer origen estiguera buit s'haurà de generar un de nou també buit.

   Nota: Es pot utilitzar el mètode `startsWith(String)`, definit en la classe `String`, amb el següent perfil:

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

   que torna `true` si el `String` actual comença per `cad`. En cas contrari, torna `false`.

Solució:

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

3. (1.5 punts) Considerant la implementació de les classes `NodeInt` i `PilaIntEnla` explicades en classe, què mostra per pantalla el següent programa?

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

Solució: 9 7 5 3 1 2 4 6 8 10

4. (3 punts) Per tal de representar paraules com seqüències enllaçades de valors de tipus `char`, es suposa ja implementada la classe `NodeChar` (anàloga a la classe `NodeInt` vista en classe), amb atributs `dada` de tipus `char` i `seguent` de tipus `NodeChar`, i amb les dues operacions constructores habituals definides en aquest tipus de classe. **Es demana** escriure un mètode estàtic `corregir`, en una classe inclosa en el mateix paquet que la classe `NodeChar`, amb el següent perfil:

```
public static NodeChar corregir(NodeChar p)
```

que, donada una paraula `p` (de tipus `NodeChar`, amb al menys 1 caràcter), la corregisca substituint totes les ocurrències del parell de caràcters consecutius **'n' 'y'** pel caràcter **'ñ'**. Per exemple, les paraules "cuca**ny**a" i "**ny**o**ny**ería", serien "cuca**ñ**a" i "**ñ**o**ñ**ería".

Solució:

```
/** la paraula té al menys 1 caràcter */
public static NodeChar corregir(NodeChar p) {
    NodeChar aux = p.seguent, ant = p;
    while(aux!=null) {
        if (ant.dada=='n' && aux.dada = 'y') {
            ant.dada = 'ñ';
            ant.seguent = aux.seguent;
            aux = ant.seguent;
        }
        else {
            ant = aux;
            aux = aux.seguent;
        }
    }
    return p;
}
```

5. (2.5 punts) Donada una llista amb punt d'interès `LlistaPIIntEnla l`, **es demana** escriure un mètode estàtic `eliminarNeg` que elimine els valors negatius d'aquesta llista, fent ús exclusivament de les operacions públiques definides en la classe `LlistaPIIntEnla`, sense accedir a la seua representació interna.

   Exemple: Si inicialment es té que `l = 3 -2 5 -7 -8 1 -10`, com a efecte de l'execució del mètode que es demana, la llista resultant serà `l = 3 5 1`.

Solució:

```
public static void eliminarNeg(LlistaPIIntEnla l) {
    l.inici();
    while(!l.fi()) {
        if (l.recuperar()<0)
            l.eliminar();
        else l.seguent();
    }
}
```