

**PRUEBA 3 PROGRAMACIÓN**  
**Junio 2007**  
**INGENIERÍA INFORMÁTICA**



UNIVERSIDAD CARLOS III DE MADRID

LEA ATENTAMENTE ESTAS INSTRUCCIONES ANTES DE COMENZAR LA PRUEBA:

- Rellene todas las hojas a bolígrafo, tanto los datos personales como las respuestas. No use bolígrafo rojo.
- No olvide rellenar el NIA y el grupo real al que pertenece.
- El tiempo máximo de realización es de 1 hora.
- El único material permitido sobre la mesa es la hoja de test y un bolígrafo

NO PASE DE ESTA HOJA, hasta que se le indique

Apellidos	Nombre	
Firma	NIA	Grupo

**PARTE 1: CUESTIONES**

**Pregunta 1 (1 Punto).**- Indicar si la siguiente afirmación es cierta, y explicar brevemente por qué.

*“El constructor de la clase File se puede usar para crear tanto directorios como ficheros”*

Falso, el constructor de la clase `File` no crea nada, tan solo asocia un objeto de esa clase a un fichero o a un directorio, pero no los crea. De hecho el directorio o fichero no tienen ni que existir. Para crear el directorio asociado a un objeto `File` habría que usar el método `mkdir()` y para crear un fichero el método `createNewFile()`

---

**Pregunta 2 (1 Punto).**- Indicar si la siguiente afirmación es cierta, y explicar brevemente por qué.

*“Se produzca o no una excepción, todas las sentencias de código de un bloque try/catch se ejecutarán”*

Falso. Si no se produce ninguna excepción sólo se ejecutarán las sentencias del `try`, ignorándose las del `catch`. Si se produce una excepción en una sentencia del bloque `try`, el resto de sentencias del `try` se ignoran y se ejecutan las del `catch`.

---

**Pregunta 3 (1 Punto).**- Indicar si la siguiente afirmación es cierta, y explicar brevemente por qué.

*“El orden en el que se colocan las sentencias catch dentro de un bloque try es irrelevante ya que Java buscará la más conveniente en caso de producirse una excepción.”*

Falso. Java va recorriendo secuencialmente de arriba abajo las sentencias `catch` asociadas a un determinado bloque `try` y ejecuta el primer `catch` que trata esa excepción o una superclase de esa excepción. Por lo tanto es obligatorio (Java no compila si no) poner primero los `catch` que tratan las excepciones más particulares y luego los que tratan excepciones genéricas que engloban a esas particulares.

**Pregunta 4 (1 Punto).**- Implementar un método que copie el contenido de un fichero que se recibe como parámetro de entrada en un nuevo fichero llamado "nuevo.dat". Se deben gestionar las excepciones pertinentes.

Hay múltiples respuestas para esta pregunta, un ejemplo con ficheros binarios y suponiendo que el fichero se puede leer entero en memoria, es:

```
public void copiar (String fichero) {  
    try {  
        File fichero1, fichero2;  
        fichero1 = new File (fichero);  
        fichero2 = new File("nuevo.txt");  
        byte [] a = new byte [(int)fichero1.length()];  
        FileInputStream fis = new FileInputStream(fichero1);  
        FileOutputStream fos = new FileOutputStream(fichero2);  
        fis.read(a);  
        fos.write(a);  
        fis.close();  
        fos.close();  
    }  
    catch (IOException e){  
        System.out.println("Error de E/S");  
    }  
}
```

**Pregunta 5 (1 Punto).- Dado el siguiente método**

```
public class ExceptionA extends Exception {}
public class ExceptionB extends Exception {}
public class ExceptionC extends ExceptionA {}
public class ExceptionD extends ExceptionA {}
public class Cuestion {
    public void metodo1(int x) throws ExceptionA, ExceptionB{
        if (x%2==0) metodo2(x);
        else metodo3(x);
    }
    public void metodo2(int x) throws ExceptionA{
        if (x<0) throw new ExceptionC();
        else if (x>6) throw new ExceptionD();
    }
    public void metodo3(int x) throws ExceptionB{
        if (x<0) throw new ExceptionB();
    }
}
public class Cuestion3 {
    public static void main (String [] args){
        Cuestion c = new Cuestion();
        try{
            c.metodo1(7);
            c.metodo1(8);
            c.metodo1(-2);
        }
        catch (ExceptionD e){
            System.out.println("D");
        }catch (ExceptionC e){
            System.out.println("C");
        }catch (ExceptionA e){
            System.out.println("A");
        }catch (ExceptionB e){
            System.out.println("B");
        }finally {
            System.out.println("adios");
        }
    }
}
```

**Explicar** la salida por pantalla tras ejecutar la clase Cuestion3.

El método main de la clase Cuestion3 hace una llamada a metodo1(7) de la clase Cuestion, este método llama a metodo2() si el parámetro es par y a metodo3() si es impar. Como 7 es impar, se llama a metodo3() que comprueba si es menor que cero. Como no lo es, vuelve sin hacer nada. A continuación se vuelve a hacer lo mismo con metodo1(8). Como 8 es par, se llama a metodo2(), que al recibir un valor mayor que 6 lanza la excepción D (hay que notar que en la cabecera de metodo2() se dice que se lanza la excepción A, pero como es una superclase de D, no hay problema)

Esta excepción se captura en el bloque catch correspondiente y se imprime por pantalla "D", a continuación se ejecuta el finally y se imprime "adios".

Por lo tanto el resultado por pantalla será:

D

adios

**PARTE 2: PROBLEMAS**

**Problema 1 (3 Puntos).**- Dado el siguiente código java:

```
public class Felino {
    private String especie;
    private int edad;
    private String colorPelaje;
    protected static int numeroEjemplares;
    private int id;

    public Felino (String esp, int ed, String color){
        especie = esp;
        edad = ed;
        colorPelaje = color;
        numeroEjemplares++;
        id = numeroEjemplares;
    }

    public Felino (){
        this ("indeterminado",0,"indeterminado");
    }
}
```

1. Crear la clase Leopardo que hereda de Felino y tiene dos atributos privados: pelajeManchado de tipo booleano que indica si el leopardo tiene o no manchas en el pelaje, y un atributo habitat de tipo String.
2. Crear un constructor para la clase Leopardo que recibe valor para todos los parámetros (incluidos los heredados). Hacer otro constructor por defecto sin parámetros que use el constructor con parámetros anterior.
3. Modificar la clase Leopardo y la clase Felino para que se puedan serializar los objetos de estas clases.
4. Crear un método public void guardar (String fichero) en la clase Leopardo que guarde un objeto de la clase Leopardo en el fichero que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.
5. Crear un método public Leopardo leer (String fichero) de la clase Leopardo que lea un objeto de la clase Leopardo del fichero que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.

```
import java.io.*;

public class Leopardo extends Felino{
    private boolean pelajeManchado;
    private String habitat;

    public Leopardo (String especie, int edad, String color, boolean
pelajeManchado, String habitat){
        super (especie, edad, color);
        this.pelajeManchado = pelajeManchado;
        this.habitat = habitat;
    }
}
```

```
public Leopardo () {
    this ("indeterminada",0,"indeterminado",false,"indeterminado");
}

public void guardar (String fichero) {
    try {
        FileOutputStream fos = new FileOutputStream (fichero);
        ObjectOutputStream oos = new ObjectOutputStream (fos);
        oos.writeObject(this);
        oos.close();
    }
    catch (FileNotFoundException e){
        System.out.println("No se encuentra el fichero");
    }
    catch (IOException e){
        System.out.println("Se ha producido una excepción al
        escribir el objeto");
    }
}

public Leopardo leer(String fichero){
    Leopardo resultado = new Leopardo();
    try {
        FileInputStream fis = new FileInputStream (fichero);
        ObjectInputStream ois = new ObjectInputStream (fis);
        resultado = (Leopardo) ois.readObject();
    }
    catch (ClassNotFoundException e){
        System.out.println ("No se ha podido reconstruir el
        objeto");
    }
    catch (IOException e){
        System.out.println("Error al leer el fichero");
    }
    return resultado;
}
}
```

Para que los objetos de ambas clases sean serializables, bastaría con modificar la cabecera de clase madre que quedaría así:

```
import java.io.*;
public class Felino implements Serializable {
    ...
}
```

No hace falta implementarla otra vez en Leopardo, puesto que si una clase implementa una interfaz sus hijas también lo harán.

**Problema 2 (2 Puntos).-** Dado el siguiente código java:

```
import java.io.*;
public class Problema3 {
    public static void main(String [] args){
        Jarra j1 = new Jarra();
        String linea;
        int litros;
        try{
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println(" Con cuantos litros quiere llenar la
jarra: ");
            linea = br.readLine();
            litros = Integer.parseInt(linea);
            j1.llenar(litros);
        }catch (IOException e) {
            System.out.println(e.toString());
        }catch (NumberFormatException e){
            System.out.println(e.toString());
        }
    }
}
```

1. Crear una excepción propia denominada `VolumenNegativo` que tenga como atributo privado un entero llamado `volumen`.
2. Crear un constructor para esta excepción que reciba como parámetro el valor del atributo `volumen`.
3. Crear una clase `Jarra` con dos atributos enteros que indiquen la máxima capacidad en volumen que tiene un objeto `Jarra` y la cantidad de litros que contiene actualmente una `Jarra`.
4. Crear dos constructores de la clase `Jarra`. Uno de ellos sin parámetros tal que cree una jarra vacía con una capacidad máxima de 3 litros. El otro constructor deberá crear la jarra vacía con una capacidad máxima dada por parámetro.
5. Crear dos métodos de la clase `Jarra`: `vaciar` y `llenar`. El primero vaciará la jarra completamente y el segundo llenará la jarra con el volumen que se le indique por parámetro. Si el volumen es negativo, este método lanzará una excepción de tipo `VolumenNegativo`. Si el volumen no es negativo, la jarra se llenará teniendo en cuenta los litros que ya tuviera y hasta un máximo de su capacidad.
6. Debido al nuevo código creado en los apartados anteriores, el código java facilitado no compila. ¿Por qué? Añadir el código que falta.



```
public class VolumenNegativo extends Exception {
    private int volumen;

    public VolumenNegativo (int vol){
        volumen = vol;
    }
}

public class Jarra {
    private int capacidad;
    private int litros;

    public Jarra (){
        capacidad = 3;
        litros = 0; // no haría falta porque es el valor por
defecto de Java
    }

    public Jarra(int cap){
        capacidad = cap;
        litros = 0;
    }

    public void vaciar (){
        litros = 0;
    }

    public void llenar (int volumen) throws VolumenNegativo {
        if (volumen<0) throw new VolumenNegativo(volumen);
        else litros = litros + volumen;
        if (litros>capacidad) litros = capacidad;
    }
}
```

Ahora no compila porque el método `llenar()` lanza la excepción `VolumenNegativo`, se puede añadir un `catch` para gestionar esta excepción o lanzarla haciendo `throws VolumenNegativo` en la cabecera del método `main`.