

**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**

<b>Apellidos</b>	<b>Nombre</b>	
<b>Firma</b>	<b>NIA</b>	<b>Grupo</b>

**PARTE 1: CUESTIONES**

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

*“Una vez ejecutado el código especificado por `try` y/o `catch`, en el bloque `finally` se incluye código que se ejecuta cuando se produce una excepción. Este bloque es opcional, puede no incluirse un bloque `finally`.”*

Falso. El bloque `finally` es efectivamente opcional, pero si existe se ejecuta siempre, haya ocurrido o no una excepción.

---

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

*“Si se quiere crear un directorio nuevo, junto con sus directorios padres el método `mkdir()` de la clase `File` es suficiente. Si el directorio ya existe devuelve un error.”*

Falso. El método `mkdir()` sólo crea un directorio si existen sus directorios padres, si no existen no hace nada. Para crear también los directorios padres se usa el método `mkdirs()`. En cualquier caso, si el directorio ya existe, no da error, simplemente devuelve `false`.

---

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

*Existen excepciones que no estamos obligados ni a gestionar ni a lanzar.*

Verdadero. Son las excepciones no capturadas, que heredan todas de `RuntimeException`. Aunque puedan ocurrir dentro de un pedazo de código, Java no obliga a gestionarlas, ni siquiera a lanzarlas en la cabecera del método mediante `throws`.

**Pregunta 4 (1 Punto).- Dado el siguiente método**

```
public String metodo2(){
    String resultado = "";
    File fichero = new File ("fichero.dat");
    try {
        DataOutputStream dos = new DataOutputStream(new
FileOutputStream (fichero,true));
        dos.writeInt(4);
        dos.writeInt(6);
        dos.close();
        dos.writeInt(8);
    }
    catch (IOException e){
        resultado ="¡Vaya, algo falló!";
    }
    try {
        DataInputStream dis = new DataInputStream (new
FileInputStream (fichero));
        int b;
        b= dis.readInt();
        b+= dis.readInt();
        return resultado+String.valueOf(b);
    }
    catch (IOException e) {
        System.out.println("Otro error de E/S");
    }
    return resultado;
}
```

**Explicar** cuál sería el resultado por pantalla al ejecutar

```
System.out.println(metodo2());
```

En el primer try se abren dos flujos para escribir en el fichero "fichero.dat". Se escribe un 4, luego un 6 y a continuación se cierra el fichero, por lo que al intentar escribir el 8, salta la excepción y se guarda como resultado "¡Vaya, algo falló!"

A continuación se ejecuta el siguiente try que lee los dos primeros elementos del fichero y los suma, haciendo que resultado valga lo que valía más la suma (es decir 10).

Por último se devuelve resultado, por lo que por pantalla se imprime:

```
¡Vaya, algo falló!10
```

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

```
public void metodo1 (){
    try {
        File directorio1 = new File ("miDirectorio");
        File directorio2= new File (directorio1,"directorioHijo");
        File fichero1 = new File (directorio1,"fichero.dat");
        File fichero2 = new File (directorio2, "fichero.dat");
        if (!directorio2.exists()) directorio2.mkdirs();
        fichero1.createNewFile();
        fichero2.createNewFile();
        fichero1.renameTo(fichero2);
        System.out.println(fichero1.exists());
        System.out.println(fichero2.exists());
    }
    catch (IOException e){
        System.out.println("Ha ocurrido una excepción de E/S");
    }
}
```

Y suponiendo que al principio no existe ninguno de los directorios ni ficheros. **Explicar** cuál sería el resultado por pantalla y cómo quedarían los ficheros y directorios tras la ejecución del método.

**Respuesta:**

Las dos primeras sentencias crean dos objetos `File` para representar sendos directorios, siendo el segundo hijo del primero. Las dos siguientes, crean dos objetos `File` que representan dos ficheros del mismo nombre, situados cada uno en uno de los directorios anteriores.

En la siguiente sentencia se crea el `directorio2`, usando `mkdirs()`, lo que crea a su vez el `directorio1`. Luego se crean sin problemas `fichero1` y `fichero2`. A continuación se intenta renombrar `fichero1` a `fichero2`, pero como `fichero2` ya existe, no se hace nada. A continuación se imprime por pantalla si los ficheros existen, lo cual es cierto.

Por lo tanto el resultado es que se crearán todos los ficheros y directorios y el resultado por pantalla es:

```
true
true
```

**PARTE 2: PROBLEMAS**

**Problema 1 (3 Puntos).**- Dado el siguiente programa 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 que se acaba de crear.
3. Crear un método public void guardar (String fichero) en la clase Leopardo que guarde todos los atributos de la clase Leopardo en el fichero de texto que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.
4. Crear un método public Leopardo leer (String fichero) de la clase Leopardo que lea un fichero creado con el método anterior, que se le pasa por parámetro, y devuelva un objeto de la clase Leopardo. 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);
        PrintWriter pw = new PrintWriter (fos);
        pw.println(pelajeManchado);
        pw.println(habitat);
        pw.close();
        fos.close();
    }
    catch (FileNotFoundException e){
        System.out.println("No se encuentra el fichero");
    }
    catch (IOException e){
        System.out.println("Excepción al escribir el objeto");
    }
}

public Leopardo leer(String fichero){
    Leopardo resultado =
        new Leopardo("leopardo",1,"negro",false,"");
    try {
        FileReader fr = new FileReader (fichero);
        BufferedReader br = new BufferedReader (fr);
        resultado.pelajeManchado = (br.readLine()=="true");
        resultado.habitat = br.readLine();
    }
    catch (IOException e){
        System.out.println("Error al leer el fichero");
    }
    return resultado;
}
```

Como no se puede acceder a los atributos heredados de Felino por ser privados, sólo se pueden guardar los propios de Leopardo. Una solución más elegante (no se pedía en el examen) era haber hecho métodos públicos para acceder a los atributos privados de Felino y así poder guardarlos.

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

```
public int leeAño () {
    int resultado;
    try {
        BufferedReader br = new BufferedReader (new InputStreamReader
                                                (System.in));

        String lectura;
        lectura = br.readLine();
        resultado = Integer.parseInt(lectura);
        // Añadir el código que lanza la excepción aquí
        if (resultado > 2007) throw new MiExcepcion (resultado);

    }
    catch (IOException e) {
        System.out.println ("Error al leer del teclado");
        return 1900;
    }
    catch (NumberFormatException e) {
        System.out.println ("Error al convertir a entero");
        return 1900;
    }
    // Añadir la gestión de la excepción aquí
    catch (MiExcepcion e) {
        System.out.println ("el año " + e.dameValor() + " no es
                             correcto");
        return 1900;
    }
    return resultado;
}
```

1. Crear una excepción propia denominada `MiExcepcion` que tenga como atributo privado un número entero que se llame `valor` y un método `public int dameValor()` que devuelva el valor del atributo.
2. Crear un constructor para esta excepción que reciba como parámetro el valor del atributo `valor`. Crear también un constructor por defecto que ponga este valor a 0.
3. Cambiar el código del método `leeAño` para que si el valor de `resultado` es mayor que 2007, salte la excepción `MiExcepcion`.
4. Añadir al método `leeAño` el código necesario para que al saltar la excepción `miExcepcion` se imprima por pantalla "el año <valor> no es correcto" y el resultado que devuelva el método sea 1900.

```
public class MiExcepcion extends Exception {
    private int valor;

    public int dameValor() {
        return valor;
    }

    public MiExcepcion (int valor) {
        this.valor = valor;
    }
    public MiExcepcion () {
        this(0);
    }
}
```

