

**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.

*“Si queremos que una parte del código se ejecute siempre, es decir, tanto si hay como si no hay errores, debemos introducir ese fragmento de código dentro de un catch.”*

Falso. El catch, es el segundo bloque del código para gestionar excepciones. En la primer parte se debe introducir el código protegido, que puede crear una excepcion. Mientras que en la segunda, el catch, introduciremos lo que queramos hacer en caso de haber producido una excepción. Si queremos que cierto código se ejecute siempre, deberemos introducir otro bloque denominado finally tras el catch, cuyo contenido se ejecutará tanto si hubo como si no hub excepción.

---

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

*“Si un método puede producir una excepción y no queremos gestionarla, debemos indicar siempre explícitamente que lance la excepción al método invocador mediante throws.”*

Falso. Si la excepción pertenece al grupo de las capturadas, la afirmación es verdadera, pero si la excepción es del grupo de las no capturadas, no es necesario lanzarla explícitamente.

---

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

*“Para que se pueda serializar un objeto en java debe implementar la interfaz serializable y todos los objetos incluidos en él tienen que implementarla también.”*

Verdadero. Nos referimos con la expresión “serializar un objeto” al proceso de convertirlo en bytes para poder enviarlo y reconstruirlo posteriormente. Para que java sea capaz de convertir el objeto en bytes debe implementar la interfaz Serializable. Como la interfaz Serializable no tiene métodos basta con indicar *implements Serializable*. Si dentro de él hay atributos de otras clases, éstos a su vez deben ser serializables.

**Pregunta 4 (1 Punto).- Explicar** cuál sería el resultado por pantalla del siguiente programa si al ejecutarlo se escribe:

- a) java Ejercicio n
- b) java Ejercicio s
- c) java Ejercicio p
- d) java Ejercicio (sin ningún argumento)

```
public class EjerExcepciones {
    public static void main (String [] args){
        try {
            char valor=(args[0]).charAt(0);
            if (valor=='n') throw new Aexception();
            if (valor=='s') throw new Bexception();
            System.out.println("Estoy en try");
        }
        catch (Aexception a){
            System.out.println("Error tipo A");
        }
        catch (Bexception b) {
            System.out.println("Error tipo B");
        }
        catch(Exception c){
            System.out.println("Error tipo C");
        }
        finally{
            System.out.println("Estoy en finally");
        }
        System.out.println("Acabar");
    }
}

class Aexception extends Exception{
} //fin clase Aexception
class Bexception extends Exception{
} //fin clase Bexception
```

Respuesta:

- a)     Error tipo A  
        Estoy en finally  
        Acabar
- b)     Error tipo B  
        Estoy en finally  
        Acabar
- c)     Estoy en try  
        Estoy en finally  
        Acabar
- d)     Error tipo C  
        Estoy en finally  
        Acabar

**Pregunta 5 (1 Punto).- Explicar** cuál sería el error del siguiente programa:

- a) salida.writeInt() debe ser datos.writeInt()
- b) datos.close() debe ser salida.close()
- c) No hace falta capturar la IOException
- d) Sobra la línea import java.io.\*;

```
import java.io.*;
class EjerFicheros {
    public static void main(String [] args) {
        DataOutputStream salida;
        try {
            salida=new DataOutputStream(new
                FileOutputStream("datos"));

            for (int i=0;i<=10;i++) {
                salida.writeInt(i);
            }
            datos.close();
        }
        catch(IOException e) {
            System.out.println("Excepcion de
                entrada/salida: "+e.toString());
        }
    }
}
```

Respuesta:

**b) datos.close() debe ser salida.close()**

**PARTE 2: PROBLEMAS**

**Problema 1 (2 Puntos).-** Dado el siguiente programa java:

```
import java.io.*;
    //Cada vez que ejecutemos este programa, se incorporara
    //una nueva línea al fichero de log que se crea la primera
    //vez que se ejecuta
class Log {
    public static void main( String args[] ) throws IOException {
        RandomAccessFile miRAFile;
        String s ="Información a incorporar\nExamen de
        programación\n";

        // Añadir el código desde aquí

        // Añadir el código hasta aquí
    }
}
```

Escribir el código que:

- a) Crea un fichero de acceso aleatorio la primera vez que se ejecuta.
- b) Añade una nueva línea cada vez que se ejecute.

Respuesta:

```
// Abrimos el fichero de acceso aleatorio
    miRAFile = new RandomAccessFile( "/tmp/java.log","rw" );

// Nos vamos al final del fichero
    miRAFile.seek( miRAFile.length() );

// Incorporamos la cadena al fichero
    miRAFile.writeBytes( s );

// Cerramos el fichero
    miRAFile.close();
```

**Problema 2 (3 Puntos).-** Dado el siguiente código java:

```
public class Vehiculo {
    private String marca;
    private String color;
    protected static int unidadesKm0Concesionario;
    private int ident;

    public Vehiculo (String marc, String col){
        marca = marc;
        color = col;
        unidadesKm0Concesionario++;
        ident = unidadesKm0Concesionario;
    }

    public Vehiculo (){
        this ("indeterminado","desconocido");
    }
}
```

1. Crear la clase Coche que hereda de Vehículo y tiene dos atributos privados precio de tipo int y descapotable de tipo booleano.
2. Crear un constructor para la clase Coche 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. Modificar las clases Coche y Vehiculo para que se puedan serializar los objetos de estas clases.
4. Crear un método public void guardar (String fichero) en la clase Coche que guarde un objeto de la clase Coche en el fichero que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.
5. Crear un método public Coche leer (String fichero) de la clase Coche que lea un objeto de la clase Coche del fichero que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.

Solución:

1. Crear la clase Coche que hereda de Vehículo y tiene dos atributos privados precio de tipo int y descapotable de tipo booleano.

```
import java.io.*;

public class Coche extends Vehiculo {
    private int precio;
    private boolean descapotable;
}
```

2. Crear un constructor para la clase `coche` que recibe valor para todos los parámetros (incluidos los heredados). Hacer otro constructor por defecto sin parámetros que use el constructor con parámetros anterior.

```
public Coche (String marc, String col, int p, boolean d){
    super (marc, col);
    this.precio = p;
    this.descapotable = d;
}

public Coche (){
    this ("indeterminado","desconocido",0,false);
}
```

3. Modificar las clases `Coche` y `Vehiculo` para que se puedan serializar los objetos de estas clases.

Para que se puedan serializar objetos de la clase `Vehículo` y de la clase `Coche` hay que implementar la interfaz `Serializable` cambiando la definición de la clase `Vehículo`:

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

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

3. Crear un método `public void guardar (String fichero)` en la clase `Coche` que guarde un objeto de la clase `coche` en el fichero que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.

```
public void guardar (String fichero) {
    try {
        FileOutputStream r = new FileOutputStream (fichero);
        ObjectOutputStream s = new ObjectOutputStream (r);
        s.writeObject(this);
        s.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");
    }
}
```

4. Crear un método `public Coche leer (String fichero)` de la clase `Coche` que lea un objeto de la clase `coche` del fichero que se le pase por parámetro. Se deberán gestionar las excepciones pertinentes.

```
public Coche leer(String fichero){
    Coche resultado = new Coche();
    try {
        FileInputStream f = new FileInputStream (fichero);
        ObjectInputStream g = new ObjectInputStream (f);
        resultado = (Coche) g.readObject();
    }
    catch (ClassNotFoundException e){
```

```
        System.out.println ("No se ha reconstruido el objeto");
    }
    catch (IOException e){
        System.out.println("Error al leer el fichero");
    }
    return resultado;
}
}
```