PROGRAMACIÓN

UNIDAD 9: Lectura y escritura de información

1. Introducción

Con frecuencia tendremos que guardar los datos de nuestro programa para poderlos recuperar más adelante. Hay varias formas de hacerlo. Una de ellas son los ficheros, que son relativamente sencillos. Otra forma más eficiente cuando es un volumen de datos muy elevado es usar una base de datos.

2. Ficheros de texto

Escribir en un fichero de texto

Un primer tipo de ficheros, que resulta sencillo de manejar, son los **ficheros de texto**. Son ficheros que podremos crear desde un programa en Java y leer con cualquier editor de textos, o bien crear con un editor de textos y leer desde un programa en Java, o bien usar un programa tanto para leer como para escribir.

Para manipular ficheros, siempre tendremos que dar tres pasos:

- Abrir el fichero
- Guardar datos o leer datos
- Cerrar el fichero

Si no abrimos fichero, obtendremos un mensaje de error al intentar acceder a su contenido.

Si no cerramos el fichero (un error frecuente), puede que realmente no se llegue a guardar ningún dato, porque no se vacíe el "buffer" (la memoria intermedia en que se quedan los datos preparados hasta el momento de volcarlos a disco).

2. Ficheros de texto

Escribir en un fichero de texto

La forma más sencilla de escribir texto en un fichero es mediante la clase FileWriter, que debemos importar desde java.io

Para abrir el fichero crearemos un objeto de la siguiente forma:

```
FileWriter f = new FileWriter("nombre_fichero");
```

Para escribir texto en el fichero utilizaremos el método *write(texto a escribir)*.

Para cerrar el fichero usaremos el método close()

Si el fichero no existe, se creará. Si existe, sobreescribirá su contenido. Si lo que queremos es añadir texto a un fichero existente sin borrar su contenido, crearemos el objeto de la siguiente forma:

```
FileWriter f = new FileWriter("nombre_fichero", true);
```

2. Ficheros de texto

Escribir en un fichero de texto

```
import java.io.FileWriter;
import java.io.IOException;

public class Ficheros {


    public static void main(String[] args) {
        // TODO code application logic here
        try {
            FileWriter f = new FileWriter("d:\\hola.txt");

            f.write("Uno");
            f.write("\nDos");
            f.close();
            f = new FileWriter("d:\\hola.txt", true);
            f.write("\nTres");

            f.close();

        } catch (IOException e)
        {
            System.err.println(e.getMessage());
        }
    }

}
```

2. Ficheros de texto

Leer de un fichero de texto

Para leer de un fichero de texto usaremos un objeto de la clase File que crearemos de la siguiente forma:

```
File fichero = new File("fichero_leer.txt");
```

Es conveniente comprobar si el fichero existe mediante el método exists() :

```
if (! fichero.exists() )
        {
            System.out.println("No he encontrado fichero_leer.txt");
            return;
        }
```

2. Ficheros de texto

Leer de un fichero de texto

La forma más sencilla de leer del fichero es mediante la clase Scanner que ya conocemos:

```
Scanner sc = new Scanner(fichero);
```

2. Ficheros de texto

Leer de un fichero de texto

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class Ficheros {

    public static void main(String[] args) {
        try {

            File fichero = new File ("d:\\hola.txt");
            if (!fichero.exists())
            {
                System.out.println("Fichero no encontrado");
            }
            else
            {
                Scanner sc = new Scanner(fichero);

                while (sc.hasNext())
                {
                    System.out.println(sc.nextLine());
                }
            }
        } catch (IOException e)
        {
            System.err.println(e.getMessage());
        }
    }
}
```

2. Ficheros de texto

Leer de un fichero de texto

También puede utilizarse la clase FileReader para leer desde un fichero de texto, pero sólo podemos leer carácter a carácter, no líneas completas:

```
FileReader fr = new FileReader(f);
```

2. Ficheros de texto

Clases BufferedReader y BufferedWriter

Las clases **BufferedReader y BufferedWriter** las podemos encontrar en **java.io.**

Estas clases tienen la misma función que **FileReader y FileWrite**r, leer y escribir en ficheros, pero **BufferedReader y BufferedWriter** optimizan estas funciones.

Se crean igual que **FileReader y FileWriter**, pero como parámetro insertaremos un objeto **FileReader** para **BufferedReader** y un objeto **FileWriter** para **BufferedWriter**.

```
BufferedReader br=new BufferedReader(new FileReader("D:\\fichero1.txt"));
BufferedWriter bw=new BufferedWriter(new FileWriter("D:\\fichero1.txt"));
```

2. Ficheros de texto

Clases **BufferedReader y BufferedWriter**

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Ficheros {
    public static void main(String[] args) {

        try
        {
            BufferedWriter bw=new BufferedWriter(new FileWriter("D:\\fichero1.txt"));
            BufferedReader br=new BufferedReader(new FileReader("D:\\fichero1.txt"));
            //Escribimos en el fichero
            bw.write("Esto es una prueba usando Buffered");
            bw.newLine();
            bw.write("Seguimos usando Buffered");
            //Guardamos los cambios del fichero
            bw.close();
            //Leemos el fichero y lo mostramos por pantalla
            String linea=br.readLine();
            while(linea!=null){
                System.out.println(linea);
                linea=br.readLine();
            }
            br.close();
        }catch(IOException e){
            System.err.println("Error E/S: "+e);
        }
    }
}
```

2. Ficheros binarios

Un fichero binario o de datos está formado por secuencias de bytes. Estos archivos pueden contener datos de tipo básico (int, float, char, etc) y objetos.

Para poder leer el contenido de un fichero binario debemos conocer la estructura interna del fichero, es decir, debemos saber cómo se han escrito: si hay enteros, long, etc. y en qué orden están escritos en el fichero. Si no se conoce su estructura podemos leerlo byte a byte

2. Ficheros binarios

Escritura en ficheros binarios

Para escribir datos en un fichero binario utilizaremos las clases Java FileOutputStream y DataOutputStream derivadas de OutputStream.

FileOutputStream

La clase FileOutputStream permite tener acceso al fichero para escribir bytes.

Para crear objetos FileOutputStream podemos utilizar los constructores:

```
FileOutputStream (String ruta)
FileOutputStream (File objetoFile);
FileOutputStream (String ruta, boolean append)
FileOutputStream (File objetoFile, boolean append)
```

Si el parámetro append es true significa que los datos se van a añadir a los existentes. Si es false los datos existentes se pierden. Si se utiliza uno de los dos primeros constructores los datos existentes se pierden.

Los constructores lanzan una **excepción FileNotFoundException** si no se ha podido crear el fichero o no se ha podido abrir para escritura.

La clase FileOutputStream proporciona el **método write() para escribir bytes** en el fichero. Este método lanza una **IOException**.

2. Ficheros binarios

Escritura en ficheros binarios

DataOutputStream

A partir de un objeto FileOutputStream se puede crear un objeto DataOutputStream, que proporciona métodos para **escribir datos de tipo primitivo** en el fichero.

Para crear un objeto DataOutputStream se utiliza el constructor:

```
DataOutputStream(OutputStream nombre);
```

La clase proporciona **métodos writeXxx()** donde Xxx es el nombre del tipo primitivo. Lanzan una **IOException.**

2. Ficheros binarios

Escritura en ficheros binarios

Ejemplo

```
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

public class Ficheros {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        FileOutputStream fos = null;
        DataOutputStream salida = null;
        int n;

        try {
            fos = new FileOutputStream("d:/datos.dat");
            salida = new DataOutputStream(fos);
```

2. Ficheros binarios

Escritura en ficheros binarios

Ejemplo (continuación)

```
System.out.print("Introduce número entero. -1 para acabar: ");
            n = sc.nextInt();
            while (n != -1) {
                salida.writeInt(n); //se escribe el número entero en el fichero
                System.out.print("Introduce número entero. -1 para acabar: ");
                n = sc.nextInt();
            }
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
```

2. Ficheros binarios

Escritura en ficheros binarios

Ejemplo (continuación)

```
finally {
            try {
                if (fos != null) {
                    fos.close();
                }
                if (salida != null) {
                    salida.close();
                }
            } catch (IOException e) {
                System.out.println(e.getMessage());
            }
        }

    }
}
```

2. Ficheros binarios

Lectura de ficheros binarios

Para leer de un fichero binario utilizaremos las clases Java FileInputStream y DataInputStream derivadas de InputStream.

FileInputStream

La clase FileInputStream permite **leer bytes** de un fichero.

Para crear objetos FileInputStream podemos utilizar los constructores:

```
FileInputStream (String ruta)
FileInputStream (File objetoFile);
```

Ambos lanzan una **excepción FileNotFoundException** si el fichero no existe.

La clase proporciona el **método read() para leer bytes** del fichero.

El método read lanza una excepción **IOException**.

2. Ficheros binarios

Lectura de ficheros binarios

DataInputStream

A partir de un objeto FileInputStream podemos crear un objeto DataInputStream para **leer datos de tipo primitivo**.

Para crear un objeto DataInputStream se utiliza el constructor:

```
DataInputStream (InputStream nombre);
```

La clase proporciona **métodos readXxx()** donde Xxx es el nombre del tipo primitivo. Lanzan una excepción **IOException**.

Cuando un método readXxx() alcanza el final del fichero lanza una **excepción EOFException**.

2. Ficheros binarios

Lectura de ficheros binarios

Ejemplo

```
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class Binarios3 {
    public static void main(String[] args) {
        FileInputStream fis = null;
        DataInputStream entrada = null;
        int n;
```

2. Ficheros binarios

Lectura de ficheros binarios

Ejemplo (continuación)

```
try {
    fis = new FileInputStream("/ficheros/datos.dat");
    entrada = new DataInputStream(fis);
    while (true) {
        n = entrada.readInt();  //se lee  un entero del fichero
        System.out.println(n);  //se muestra en pantalla
    }
} catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
} catch (EOFException e) {
    System.out.println("Fin de fichero");
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

2. Ficheros binarios

Lectura de ficheros binarios

Ejemplo (continuación)

```
finally {
            try {
                if (fis != null) {
                    fis.close();
                }
                if (entrada != null) {
                    entrada.close();
                }
            } catch (IOException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

2. Ficheros binarios

Escritura y lectura de objetos en ficheros binarios

Java ha añadido una interesante faceta al lenguaje denominada serialización de objetos que permite convertir cualquier objeto cuya clase implemente el interface *Serializable* en una secuencia de bytes que pueden ser posteriormente leídos para restaurar el objeto original.

El interface *Serializable*

Un objeto se puede serializar si implementa la interfaz *Serializable*. Esta interfaz no declara ninguna función miembro, se trata de una interfaz vacía.

Para que un objeto sea serializable, todos los objetos que contenga también deben ser serializables.

2. Ficheros binarios

Escritura y lectura de objetos en ficheros binarios

Una vez que hemos definido una clase como serializable, la lectura y escritos de los objetos de dicha clase es muy parecida a las situaciones vistas anteriormente, así que no deberían de ser un problema entender su funcionamiento.

Para realizar la lectura y escritura de objetos vamos a utilizar las siguientes clases:

- **File**: Para crear o cargar el archivo.
- **FileInputStream** y **FileOutputStream**: Para leer o escribir los objetos.
- **ObjectInputStream** y **ObjectOutputStream**: Que serán los encargados de manejar la lectura y la escritura de los objetos.

Vamos a ver cómo se realizará la escritura y lectura de objetos mediante un ejemplo, para ello nos vamos a crear una clase Personas que implementa Seializable.

2. Ficheros binarios

Escritura y lectura de objetos en ficheros binarios

```java
import java.io.Serializable;


public class Personas implements Serializable{

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String nombre;
    private int edad;
    private String dni;
    public Personas(String nombre, int edad, String dni) {
        super();
        this.nombre = nombre;
        this.edad = edad;
        this.dni = dni;
    }
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }
    public String getDni() {
        return dni;
    }
}
```

2. Ficheros binarios

Escritura de objetos en ficheros binarios. ObjectOutputStream.

Los pasos a seguir son:

- Crear o cargar un fichero por medio de la clase **File**.
- Se lo pasamos a **FileOutputStream** para escribir.
- Este, se lo pasamos a **ObjectOutputStream** para que escriba.
- Escribimos el objeto en el fichero.
- Cerramos **ObjectOutputStream**.

El archivo a crear tiene que tener extensión **.obj**

Debemos capturar **IOException** que lanza el método "**writeObject(Object o)**".

2. Ficheros binarios

Escritura de objetos en ficheros binarios. ObjectOutputStream.

```
public static void setEscritura() throws IOException{
    File f=new File("datos.obj");//Creo el archivo
    FileOutputStream fos=new FileOutputStream(f);//Se lo paso
    ObjectOutputStream oos=new ObjectOutputStream(fos); //Este a su vez se lo pasamos
    oos.writeObject(new Personas("Juan",40,"258741369Z"));//escribo
    oos.writeObject(new Personas("Luis",35,"236985471B"));//escribo
    oos.writeObject(new Personas("Pilar",41,"777589632L"));//escribo
    oos.close();//cierro
}
```

2. Ficheros binarios

Lectura de objetos en ficheros binarios. ObjectInputStream.

Los pasos a seguir son:

- Cargar el fichero mediante la clase **File**.
- Se lo pasamos a **FileInputStream**.
- Este se lo pasamos a **ObjectInputStream**.
- Leemos los objetos.
- Cerramos **ObjectInputStream**.

Se deben capturar las excepciones **IOException** y **ClassNotFoundException** lanzadas por **ObjectInputStream** y el método **readObject()** respectivamente.

2. Ficheros binarios

Lectura de objetos en ficheros binarios. ObjectInputStream.

```
public static void getLectura() throws ClassNotFoundException, IOException{
    ObjectInputStream ois=null;
    try{
        File f=new File("datos.obj");
        FileInputStream fis=new FileInputStream(f);
        ois=new ObjectInputStream(fis);
        while(true){
            Personas p=(Personas) ois.readObject();
            System.out.println("Nombre:"+p.getNombre());
            System.out.println("Edad: "+p.getEdad());
            System.out.println("DNI: "+p.getDni());
            System.out.println("**************************");
        }
    }catch(IOException io){
        System.out.println("\n**********Fin**********");
    }finally{
        ois.close();
    }
}
```

Cargamos nuestro fichero en sus respectivas clases, creamos un bucle infinito con el **while(true)** y capturamos **IOException** para que cuando el archivo termine de leer objetos y salte, nos muestre por pantalla nuestro dialogo "**Fin**".

En el bloque **finally** cerramos **ObjetInputStream** ya que este bloque se ejecuta siempre, salte o no la excepción.