8. Ficheros Binarios

- Los ficheros binarios almacenan secuencias de dígitos binarios que no son legibles directamente por el usuario tal y como ocurre con los ficheros de texto.
- Estos archivos pueden contener datos de tipo básico (int, float, char, etc) y objetos.
- Ocupan menos espacio en disco que los anteriores.
- **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.

8. Ficheros Binarios

- Las clases Java que nos permiten trabajar con este tipo de ficheros son **FileInputStream** y **FileOutputStream**.

8.1 Lectura de ficheros binarios

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

8.1 FileInputStream

- La clase FileInputStream permite **leer bytes** de un fichero.
- Para crear objetos FileInputStream podemos utilizar 3 constructores.
- ***Constructores de la clase FileInputStream:***

Constructor and Description
FileInputStream(File file) Creates a FileInputStream by opening a connection to an actual file, the file named by the File object file in the file system.
FileInputStream(FileDescriptor fdObj) Creates a FileInputStream by using the file descriptor fdObj, which represents an existing connection to an actual file in the file system.
FileInputStream(String name) Creates a FileInputStream by opening a connection to an actual file, the file named by the path name name in the file system.

8.1 FileInputStream

- Lanzan una **excepción FileNotFoundException** si el fichero no existe.
- La clase proporciona el **método read() para leer bytes** del fichero:
  - **int read()**
  - **int read (byte buff[]):** Lee buff.length caracteres y los almacena en buff
  - **int read (byte [] buff, int desplazamiento, int n):** lee n caracteres de datos y los almacena en buff a partir de la posición buff[desplazamiento]
- El método read lanza una excepción **IOException**.

8.2 Escritura de ficheros binarios

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

8.2.1 FileOutputStream

- La clase FileOutputStream permite tener acceso al fichero para **escribir bytes**.
- Para crear objetos FileOutputStream podemos utilizar varios constructores.
- ***Constructores de la clase FileOutputStream:***

Constructor and Description
FileOutputStream(File file) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(File file, boolean append) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(FileDescriptor fdObj) Creates a file output stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system.
FileOutputStream(String name) Creates a file output stream to write to the file with the specified name.
FileOutputStream(String name, boolean append) Creates a file output stream to write to the file with the specified name.

8.2.1 FileOutputStream

- 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 existe y no se ha podido crear el fichero.

8.2.1 FileOutputStream

- La clase FileOutputStream proporciona el **método write() para escribir bytes** en el fichero.
- Este método lanza una **IOException**.
- Algunos de los métodos de esta clase son :
  - **void write(int b)**
  - **void write(byte[]b)**
  - **void write(byte []b, int desplazamiento, int n)**

- Lanzan una **IOException**

8.3 Ejemplo

```
public class EjemplolBinarios {
        public static void main(String[] args) throws FileNotFoundException, IOException
            File f = new File("C:\\FICHEROS\\BINARIO1.BIN");
            FileOutputStream fOut = new FileOutputStream(f);
            FileInputStream fIn = new FileInputStream(f);
            int i;
            for (i = 1; i <= 100; i++) {
                fOut.write(i);
            }
            fOut.close();
            while ((i = fIn.read()) != -1) {
                System.out.println(i);
            }
            fIn.close();
             //LA MISMA VERSIÓN CON CARACTERES, hay que abrir de nuevo los archivos
             fOut = new FileOutputStream(f);
             fIn = new FileInputStream(f);
             int letra;
             for (letra = 'a'; letra <= 'z'; letra++) {
                 fOut.write(letra);
             }
             fOut.close();
             while ((letra = fIn.read()) !=-1) {
                 System.out.println((char)letra);
             }
             fIn.close();
             System.exit(0);
         }
    }
```

8.4 Tratamiento de tipos primitivos

- Para leer y escribir datos de tipos primitivos: int, float, long, etc, se usan las clases **DataInputStream** y **DataOutpuStream** que proporcionan métodos adicionales para la lectura/escritura de datos.

8.4.1 DataInputStream

- Los métodos de esta clase son:

Type	Method and Description
boolean	readBoolean() See the general contract of the readBoolean method of DataInput.
Byte	readByte() See the general contract of the readByte method of DataInput.
Char	readChar() See the general contract of the readChar method of DataInput.
Double	readDouble() See the general contract of the readDouble method of DataInput.
Float	readFloat() See the general contract of the readFloat method of DataInput.
Int	readInt() See the general contract of the readInt method of DataInput.

8.4.1 DataInputStream

Long	**readLong()** See the general contract of the readLong method of DataInput.
Short	**readShort()** See the general contract of the readShort method of DataInput.
Int	**readUnsignedByte()** See the general contract of the readUnsignedByte method of DataInput.
Int	**readUnsignedShort()** See the general contract of the readUnsignedShort method of DataInput.
String	**readUTF()** See the general contract of the readUTF method of DataInput.

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

8.4.2 DataOutputStream

- Los métodos de esta clase son:

Type	Method and Description
Void	**writeBoolean(boolean v)** Writes a boolean to the underlying output stream as a 1-byte value.
Void	**writeByte(int v)** Writes out a byte to the underlying output stream as a 1-byte value.
Void	**writeBytes(String s)** Writes out the string to the underlying output stream as a sequence of bytes.
Void	**writeChar(int v)** Writes a char to the underlying output stream as a 2-byte value, high byte first.
Void	**writeChars(String s)** Writes a string to the underlying output stream as a sequence of characters.
Void	**writeDouble(double v)** Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the underlying output stream as an 8-byte quantity, high byte first.

8.4.2 DataOutputStream

- Los métodos de esta clase son:

Void	**writeFloat(float v)** Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.
Void	**writeInt(int v)** Writes an int to the underlying output stream as four bytes, high byte first.
Void	**writeLong(long v)** Writes a long to the underlying output stream as eight bytes, high byte first.

Void	**writeShort(int v)** Writes a short to the underlying output stream as two bytes, high byte first.
Void	**writeUTF(String str)** Writes a string to the underlying output stream using modified UTF-8 encoding in a machine-independent manner.

8.4.3 Ejemplo

- Con la clase DataInputStream no podemos controlar el final de fichero del mismo modo (ver ejemplo)
- Otro problema es que la información sigue una repetición, pero no una estructura.
- Por ejemplo, qué pasaría si no supiésemos la edad de una de las personas? No podríamos saltarnos la escritura de esa edad porque se romperían las repeticiones y sería difícil localizar qué dato falta a la hora de leer del fichero.

8.4.3 Ejemplo

```
public class Ejemplo2Binarios {
        public static void main(String[] args) throws FileNotFoundException, IOException {
            File f = new File("C:\\FICHEROS\\BINARIO3.BIN");
            FileOutputStream fOut = new FileOutputStream(f);
            FileInputStream fIn = new FileInputStream(f);

            //Añadimos los Data_Stream
            DataOutputStream dOut = new DataOutputStream(fOut);
            DataInputStream dIn = new DataInputStream(fIn);
            String[] nombres = {"Luis", "Rosa", "Juan", "Pedro",
                    "Ana", "María", "Antonio", "Rubén", "Carlos", "Eva"};
            int[] edades = {18, 19, 18, 20, 21, 20, 18, 20, 22, 19};

            for (int i = 0; i < nombres.length; i++) {
                dOut.writeUTF(nombres[i]);
                //No se puede utilizar dOut.writeChars(String s) a no ser
                //que añadamos el "\n" expresamente nosotros
                //En este caso podríamos leer con readLine, pero está obsoleta
                dOut.writeInt(edades[i]);
            }
            dOut.close();
```

8.4.3 Ejemplo

```
            //El problema que tienen estos ficheros es que hay que recordar perfectamente el orden
            //en que se grabaron los datos para recuperarlos en el mismo orden
            String nombre;
            int edad;

            //En este tipo de flujo, si vamos a leer más allá del final, obtendríamos una excepción.
            //Por ello lo vamos a encerrar en un try catch para evitarlo
            try {
                while (true) {
                    nombre = dIn.readUTF();
                    edad = dIn.readInt();
                    System.out.println("Nombre: " + nombre + ", edad: " + edad);
                }
            } catch (Exception e) {
                //Aquí llegará al final del fichero, es decir, cuando intente leer más allá del final
            } finally //Se ejecuta tanto si hay errores como si no
            {
                dIn.close();
            }
        }
}
```

8.5 BufferedOutputStream

- También se puede manejar ficheros binarios de forma más eficiente haciendo uso de los Buffer.

```
public class Ejemplo3Binarios {
    public static double[][] matriz = {{2.0,3.0,4.0},{-2.0,-3.0,-4.0}};

    public static void main(String[] args) {

        File f = new File("C:\\ficheros\\ficheroB.dat");
        int row = matriz.length;
        int col = matriz[0].length;

        try {
            FileOutputStream fOut = new FileOutputStream(f);
            BufferedOutputStream bOut = new BufferedOutputStream(fOut);
            DataOutputStream dOut = new DataOutputStream(bOut);
            dOut.writeInt(row);
            dOut.writeInt(col);
            for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                    dOut.writeDouble(matriz[i][j]);
                }
            }
            dOut.close();
            } catch (IOException e) {}
            }

    }
```

8.6 BufferedInputStream

```
public static void leerFichero(File f) throws IOException{
    double[ ][ ] data;
    DataInputStream in = null;
    try{
        in = new DataInputStream(new BufferedInputStream( new FileInputStream(f)));
        int row = in.readInt();
        System.out.println("fila = " + row);
        int col = in.readInt();
        System.out.println("columna = " + col);
        data = new double[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                data[i][j] = in.readDouble();
                System.out.println("dato[" + i + "][" + j+ "] = " + data[i][j]);
            }
        }
    } catch (IOException e) {}
    finally{
        in.close();
    }
}
```