10.C. Ficheros.

1. Trabajando con ficheros.

1.3. Ficheros binarios y ficheros de texto (II).

Los **ficheros binarios** almacenan la información en bytes, codificada en binario, pudiendo ser de cualquier tipo: fotografías, números, letras, archivos ejecutables, etc.

Los archivos binarios guardan una representación de los datos en el fichero. O sea que, cuando se guarda texto no se guarda el texto en sí, sino que se guarda su representación en código UTF-8.

Para **leer datos de un fichero binario**, Java proporciona la clase **FileInputStream**. Dicha clase trabaja con bytes que se leen desde el flujo asociado a un fichero. Aquí puedes ver un ejemplo comentado.

```
package fileinputconbuffer;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class leerConBuffer {

    public static void main(String[] args) {
            int tama ;

    try{
      // Creamos un nuevo objeto File, que es la ruta hasta el fichero desde
      File f = new File("C:\\apuntes\\test.bin");

      // Construimos un flujo de tipo FileInputStream (un flujo de entrada desde
      // fichero) sobre el objeto File. Estamos conectando nuestra aplicación
      // a un extremo del flujo, por donde van a salir los datos, y "pidiendo"
      // al Sistema Operativo que conecte el otro extremo al fichero que indica
      // la ruta establecida por el objeto File f que habíamos creado antes. De
      FileInputStream flujoEntrada = new FileInputStream(f);
      BufferedInputStream fEntradaConBuffer = new BufferedInputStream(flujoEntrada);

      // Escribimos el tamaño del fichero en bytes.
      tama = fEntradaConBuffer.available();
      System.out.println("Bytes disponibles: " + tama);

      // Indicamos que vamos a intentar leer 50 bytes del fichero.
      System.out.println("Leyendo 50 bytes....");
```

```
        // Creamos un array de 50 bytes para llenarlo con los 50 bytes
        // que leamos del flujo (realmente del fichero)*/
        byte bytearray[] = new byte[50];

        // El método read() de la clase FileInputStream recibe como parámetro un
        // array de byte, y lo llena leyendo bytes desde el flujo.
        // Devuelve un número entero, que es el número de bytes que realmente se
        // han leído desde el flujo. Si el fichero tiene menos de 50 bytes, no
        // podrá leer los 50 bytes, y escribirá un mensaje indicándolo.
        if (fEntradaConBuffer.read(bytearray) != 50)
            System.out.println("No se pudieron leer 50 bytes");

        // Usamos un constructor adecuado de la clase String, que crea un nuevo
        // String a partir de los bytes leídos desde el flujo, que se almacenaron
        // en el array bytearray, y escribimos ese String.
        System.out.println(new String(bytearray, 0, 50));

        // Finalmente cerramos el flujo.Es importante cerrar los flujos
        // para liberar ese recurso. Al cerrar el flujo, se comprueba que no
        // haya quedado ningún dato en el flujo sin que se haya leído por la aplicación. */
        fEntradaConBuffer.close();

        // Capturamos la excepción de Entrada/Salida. El error que puede
        // producirse en este caso es que el fichero no esté accesible, y
        // es el mensaje que enviamos en tal caso.
    }catch (IOException e){
            System.err.println("No se encuentra el fichero");
    }
    }
}
```

Para **escribir datos a un fichero binario**, la clase nos permite usar un fichero para escritura de bytes en él, es la clase `FileOutputStream`. La filosofía es la misma que para la lectura de datos, pero ahora el flujo es en dirección contraria, desde la aplicación que hace de fuente de datos hasta el fichero, que los consume.

Autoevaluación

Señala si es verdadera o falsa la siguiente afirmación:

Para leer datos desde un fichero codificados en binario empleamos la clase FileOutputStream. ¿Verdadero o falso?

○ Verdadero.

Falso.