Tema 9: Ficheros de disco

1. Leer de un fichero binario

Llamamos archivos binarios a aquellos que no son de texto plano. Algunos ejemplos pueden ser: archivos de audio, de vídeo, de Word, de imagen, etc.

Mediante java podemos leer un archivo binario byte a byte y hacer lo que queramos con la información que contiene. Aunque, por supuesto, sólo lo podremos hacer si conocemos la manera en que está codificada esa información.

Para trabajar con archivos de disco debemos importar el paquete *java.io*:

- `import java.io.*;`

Empezamos creando un objeto de la clase FileInputStream, pasándole como parámetro la ruta del archivo que queremos leer:

- `FileInputStream archivo = new FileInputStream("ficheros/miarchivo.txt");`

En esta operación puede ocurrir que no se encuentre el archivo. Por eso java nos obliga a escribir esta orden en un bloque try-catch, capturando una posible IOException.

El archivo hay que ir leyéndolo byte a byte (con un bucle) desde el principio, con el método *read*:

- `caracter = archivo.read();`

Sabremos que hemos llegado al final del archivo cuando el método anterior devuelva -1.

Por último, los archivos que abrimos hemos de cerrarlos con el método *close( )*.

- `archivo.close();`

Ejemplo: Programa que lee un archivo del disco y va mostrando el valor de cada byte y, al final, el nº total de bytes leídos. Usamos una variable booleana llamada *eof* (end of file) como semáforo para parar de leer cuando lleguemos al final del archivo).

```
public class LectorByte {

	public static void main(String[] args) {
		try {
			FileInputStream archivo = new FileInputStream(
					"ficheros/miarchivo.txt");
			boolean eof = false;
			int contador = 0;
			int caracter = 0;

			while (!eof) {
				caracter = archivo.read();
				if (caracter == -1)
					eof = true;
				else {
					System.out.print(" " + caracter);
					contador++;
				}
			}

			System.out.println("\nTotal bytes: " + contador);
			archivo.close();
		} catch (IOException e) {
			System.out.println("Errorcillo: " + e.toString());
		}

	}
}
```

2. Escribir en un fichero binario

Para acceder a un archivo en modo escritura usaremos un objeto *FileOutputStream*:

- `FileOutputStream salida = new FileOutputStream("ficheros/salida.odt");`

Si no queremos sobreescribir el archivo sino añadir bytes al final (append):

- `FileOutputStream salida = new FileOutputStream("ficheros/salida.odt", true);`

Para guardar un byte dentro del archivo usaremos el método *write( )*:

- `salida.write(caracter);`

Igual que cuando abrimos un archivo para lectura, al final hay que cerrarlo:

- `salida.close();`

Cuando creemos un archivo de esta manera hay que tener en cuenta que no lo veremos en el panel de Eclipse, ya que no lo hemos creado con él. Habrá que refrescar el listado de archivos pulsando F5.

Ejemplo: Programa que lee byte a byte un documento de Libreoffice *entrada.odt* y lo copia en *salida.odt*.

```
public class Copia {

	public static void main(String[] args) {
		System.out.println("Accediendo al disco ...");
		try {
			FileInputStream entrada = new FileInputStream(
					"ficheros/entrada.odt");
			FileOutputStream salida = new FileOutputStream(
					"ficheros/salida.odt");
			boolean eof = false;
			int caracter = 0;

			while (!eof) {
				caracter = entrada.read();
				System.out.println(" " + caracter);
				if (caracter == -1)
					eof = true;
				else {
					salida.write(caracter);
				}

			}

			entrada.close();
			salida.close();

		} catch (IOException e) {
			System.out.println("Errorcillo: " + e.toString());
		}
	}

}
```

3. Uso de buffers

Tal como hemos accedido a los archivos hasta ahora cada vez que ejecutamos una operación de escritura o lectura el sistema accede al disco, lo que puede ser muy costoso en tiempo.

Una solución es utilizar buffers. Al usar buffers, la lectura y escritura se hace sobre un espacio en la memoria RAM que hace de intermediario con el fichero del disco. De esta manera se accede al disco para leer o escribir cantidades grandes de información de golpe, aunque la operación que nosotros ejecutamos sea de lectura o escritura de un sólo byte.

En la práctica esto lo conseguimos creando un objeto *BufferedInputStream* o *BufferedOutputStream* a partir del *FileInputStream* o *FileOutputStream*. A partir de ahí usaremos el *BufferedInputStream* o *BufferedOutputStream* en lugar de los anteriores.

Es imprescindible que antes de cerrar nuestro programa cerremos el objeto buffer, ya que si no, algunos datos en memoria no se habrán llegado a guardar en el disco.

Ejemplo: Programa que guarda en un fichero los números del 1 al 50 y después los lee, usando buffers.

```
public class Bufferes {

	static String archivo = "ficheros/numeros.dat";

	public static void main(String[] args) {
		// Escribimos los números del 1 al 50 en un archivo de disco.
		escribir();
		// Leemos el contenido del anterior archivo y lo mostramos en la
		// consola.
		leer();
	}

	public static void escribir() {
		try {
			FileOutputStream fichero = new FileOutputStream(archivo);
			BufferedOutputStream buffer = new BufferedOutputStream(fichero);

			for (int i = 1; i <= 50; i++) {
				buffer.write(i);
			}

			buffer.close();
		} catch (IOException e) {
			System.out.println("Error: " + e.toString());
		}

	}
```

```
	public static void leer() {
		try {
			FileInputStream fichero = new FileInputStream(archivo);
			BufferedInputStream buffer = new BufferedInputStream(fichero);
			boolean eof = false;
			int caracter = 0;

			while (!eof) {
				caracter = buffer.read();
				if (caracter == -1)
					eof = true;
				else
					System.out.println(caracter + " ");
			}

			buffer.close();
		} catch (IOException e) {
			System.out.println("Error: " + e.toString());
		}
	}

}
```

Actividad: Modifica el programa anterior para que los métodos escribir( ) y leer( ) devuelvan un booleano que indique si ha ido bien (true) o mal (false).

Actividad: Modifica el programa anterior para no utilizar bloques try-catch sino la sentencia *throws*.

4. Lectura de streams de texto

Para leer específicamente archivos de texto, disponemos de algunas clases especiales:

- FileReader
- BufferedReader

Podemos utilizar los siguientes métodos de BufferedReader para leer caracteres:

- read( ): Lee un carácter (char) del archivo. Devuelve -1 si ya no quedan caracteres en el archivo.
- readLine( ): Lee una línea completa del archivo. Devuelve *null* si ya no quedan líneas en el archivo.

Al terminar de trabajar con el archivo hay que cerrarlo:

- close( )

Ejemplo: Programa que lee el propio archivo de código fuente y lo muestra por la consola.

```
import java.io.*;

public class LeerCodigo {

	public static void main(String[] args) {
		String linea = "";

		try {
			FileReader archivo = new FileReader("src/LeerCodigo.java");
			BufferedReader buffer = new BufferedReader(archivo);
			boolean eof = false;

			while (!eof) {
				linea = buffer.readLine();
				if (linea == null)
					eof = true;
				else
					System.out.println(linea);
			}

			buffer.close();
		} catch (IOException e) {
			System.out.println("Error de E/S: " + e.toString());
		}
	}
}
```

5. Escritura de streams de texto

Para escribir específicamente archivos de texto, disponemos de algunas clases especiales:

- FileWriter (incluiremos como segundo parámetro *true* si queremos escribir al final del archivo, en lugar de sobreescribir).
- BufferedWriter

Podemos utilizar los siguientes métodos de BufferedWriter para escribir caracteres:

- write( ): Escribe una cadena de texo en el archivo.
- newLine( ): Inserta un salto de línea en el archivo.

Al terminar de trabajar con el archivo hay que cerrarlo:

- close( )

Ejemplo: Programa que guarda en un archivo de texto el nombre y la puntuación del usuario de un juego.

```
import java.io.*;

public class GuardarPuntuaciones {

	public static void main(String[] args) {
		final String nombre = "charlie";
		final int puntos = 10;

		try {
			FileWriter archivo = new FileWriter("ficheros/puntuaciones.txt");
			BufferedWriter buffer = new BufferedWriter(archivo);

			buffer.write(nombre);
			buffer.newLine();
			buffer.write(String.valueOf(puntos));

			buffer.close();

		} catch (IOException e) {
			System.out.println("Error de E/S: " + e.toString());
		}
	}

}
```

6. Acceder a la carpeta personal o a la carpeta actual del usuario

El usuario con el que hemos abierto sesión en el ordenador dispone de una carpeta personal. Podemos acceder a la ruta (String) de esta carpeta con el método:

```
System.getProperty("user.home");
```

Podemos acceder a la ruta (String) desde donde estamos ejecutando la aplicación con el método:

```
System.getProperty("user.dir");
```

7. Manejo de archivos y carpetas

Para manipular carpetas y archivos podemos utilizar un objeto File, al cual le pasamos la ruta de un archivo o una carpeta. Sus métodos suelen devolver un valor booleano que indica si la operación se ha realizado (true) o no (false).

Por ejemplo para crear una carpeta llamada "configuración" dentro de nuestra carpeta de proyecto:

```
File carpeta = new File("configuracion");
boolean creada = carpeta.mkdir();
```

También podemos crear una jerarquía de carpetas usando el método *mkdirs( )* en lugar de *mkdir( )*:

```
File carpeta = new File("herramientas/imagenes/configuracion/xml");
boolean creada = carpeta.mkdirs();
```

Podemos borrar un archivo o una carpeta:

```
File archivo = new File("ficheros/archivo_borrar.txt");
File carpeta = new File("ficheros/carpeta_borrar");

archivo.delete();
carpeta.delete();
```

Podemos cambiar el nombre de un archivo o carpeta:

```
File archivoOrigen = new File("ficheros/miarchivo.txt");
File archivoDestino = new File("ficheros/miarchivonuevo.txt");
boolean cambiado = archivoOrigen.renameTo(archivoDestino);
```

Ejemplo 1: Programa que crea una carpeta dentro de la carpeta de proyecto, llamada "configuración".

```
public class Ejemplo1 {

	public static void main(String[] args) {
		File carpeta = new File("configuracion");
		boolean creada = carpeta.mkdir();
		if (!creada) {
			System.out.println("No se ha podido crear la carpeta");
		}
	}

}
```

Ejemplo 2: Programa que crea de golpe una jerarquía de carpetas dentro de la carpeta de nuestro proyecto.

```
public class Ejemplo2 {

	public static void main(String[] args) {
		File carpeta = new File("herramientas/imagenes/configuracion/xml");
		boolean creada = carpeta.mkdirs();
		if (!creada) {
			System.out.println("No se han podido crear las carpetas");
		}
	}

}
```

Ejemplo 3: Programa que cambia el nombre de un archivo de nuestra carpeta de proyecto. La ruta de este archivo es *"ficheros/miarchivo.txt"*.

```
public class Ejemplo3 {

	public static void main(String[] args) {
		File archivoOrigen = new File("ficheros/miarchivo.txt");
		File archivoDestino = new File("ficheros/miarchivonuevo.txt");

		boolean cambiado = archivoOrigen.renameTo(archivoDestino);
		if (!cambiado) {
			System.out.println("No se ha podido cambiar el nombre del
archivo");
		}
	}
}
```

Ejemplo 4: Programa que borra un archivo y una carpeta e informa del resultado.

```
public class Ejemplo5 {

	public static void main(String[] args) {

		File archivo = new File("ficheros/archivo_borrar.txt");
		File carpeta = new File("ficheros/carpeta_borrar");

		if (archivo.delete())
			System.out.println("Se ha borrado el archivo.");
		else
			System.out.println("No se ha podido borrar el archivo");

		if (carpeta.delete())
			System.out.println("Se ha borrado la carpeta.");
		else
			System.out.println("No se ha podido borrar la carpeta.");

	}
}
```

Actividad 5: Programa que lee un archivo de texto y lo pasa a mayúsculas. Para eso crea un archivo temporal con el original pasado a mayúsculas; luego borra el original y renombra el temporal.

```
public class Transformar {

	public static void main(String[] args) {

		// Nombres de los archivos con los que vamos a trabajar
		String archivoOriginal = "ficheros/miarchivo.txt";
		String archivoTemporal = "ficheros/miarchivo.txt.tmp";
		boolean eof = false;
		String linea = "";

		try {
			// Creamos objetos File que los representen
			File origen = new File(archivoOriginal);
			File temporal = new File(archivoTemporal);

			// Creamos los flujos de entrada y salida
			FileReader fr = new FileReader(origen);
			BufferedReader br = new BufferedReader(fr);

			FileWriter fw = new FileWriter(temporal);
			BufferedWriter bw = new BufferedWriter(fw);

			// Vamos leyendo el archivo de origen caráctar a carácter, pasando
			// cada carácter a mayúsculas y escribiéndolo en el archivo
			// temporal.
			while (!eof) {
				linea = br.readLine();
				if (linea == null)
					eof = true;
				else
					bw.write(linea.toUpperCase());
			}

			br.close();
			bw.close();

			// Ahora borramos el archivo original. Si no ocurre ningún error,
			// renombramos el archivo temporal como el original.
			boolean borrado = origen.delete();
			if(borrado)
				temporal.renameTo(origen);
		} catch (IOException ioe) {
			System.out.println("Error de E/S: " + ioe.toString());
		}

	}
}
```

8. JFileChooser

Podemos usar el componente JFileChooser para elegir el archivo que queremos abrir en cualquier ubicación del equipo. Este componente muestra una típica ventana para elegir la carpeta y el archivo que nos interesa, y ofrece también las operaciones habituales, como crear carpetas o renombrar archivos.

La idea es que este cuadro nos devuelva el nombre de un archivo para luego hacer alguna operación con él con las clases vistas anteriormente.

Creamos un objeto JFileChooser:

- `JFileChooser eleccion = new JFileChooser();`

Llamamos a su método *showOpenDialog( )* para que se muestre el cuadro en modo "Abrir" o *showSaveDialog( )* para que se muestre en modo "Guardar".

- `int resultado = eleccion.showOpenDialog(null);`

Este método devuelve un entero que indica si el usuario ha pulsado el botón "Abrir" o el botón "Cancelar". Puede comprobarse comparando con las constantes:

- `JFileChooser.APPROVE_OPTION`
- `JFileChooser.CANCEL_OPTION`

El archivo (un objeto File) que el usuario ha seleccionado puede obtenerse del método *getSelectedFile( )*. Podemos sacar el nombre con *toString( )*.

```
if (resultado == JFileChooser.APPROVE_OPTION)
	System.out.println(eleccion.getSelectedFile().toString());
else
	System.out.println("No se ha elegido nada.");
```

De este modo hemos obtenido el nombre del archivo que el usuario quiere abrir. Ahora habría que utilizar lo que ya sabemos para abrir el archivo y hacer algo con su contenido.

Ejemplo: Programa que muestra una ventana con dos botones "Abrir" y "Guardar" que abren paneles para elegir ficheros. Muestran por consola el nombre del archivo elegido.

```
public class AbrirArchivo extends JFrame {

	AbrirArchivo() {
		super("Elegir archivo");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(300, 100);

		setLayout(new FlowLayout());

		JButton btnAbrir = new JButton("Abrir");
		JButton btnGuardar = new JButton("Guardar");

		btnAbrir.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent arg0) {
				JFileChooser eleccion = new JFileChooser();
				int resultado;

				resultado = eleccion.showOpenDialog(null);

				if (resultado == JFileChooser.APPROVE_OPTION)

	System.out.println(eleccion.getSelectedFile().toString());
				else
					System.out.println("No se ha elegido nada.");
			}
		});

		btnGuardar.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {
				JFileChooser eleccion = new JFileChooser();
				int resultado;

				resultado = eleccion.showSaveDialog(null);

				if (resultado == JFileChooser.APPROVE_OPTION)

	System.out.println(eleccion.getSelectedFile().toString());
				else
					System.out.println("No se ha elegido nada.");
			}

		});

		add(btnAbrir);
		add(btnGuardar);

		setVisible(true);
	}

	public static void main(String[] args) {
		new AbrirArchivo();
```

```
        }
}
```

Actividad: Crea un programa que pregunte al usuario el nombre del archivo de texto que quiere abrir y muestre el contenido en un JTextArea.