8. TEMA 8: Estudio de flujos de entrada/salida y ficheros.

Este tema es importante en cuanto al control de los flujos de información de dispositivos tanto de entrada, salida o ambos.

Aprenderemos como capturar información de estos flujos y también como enviarla, además aprenderemos como crear un programa que tenga datos persistentes.

Se aprenderá a manejar ficheros, leer de disco y escribir en disco.

También se verá como usar estos flujos para leer datos de una fuente como por ejemplo una página web.

8.1 Tipos de flujos.

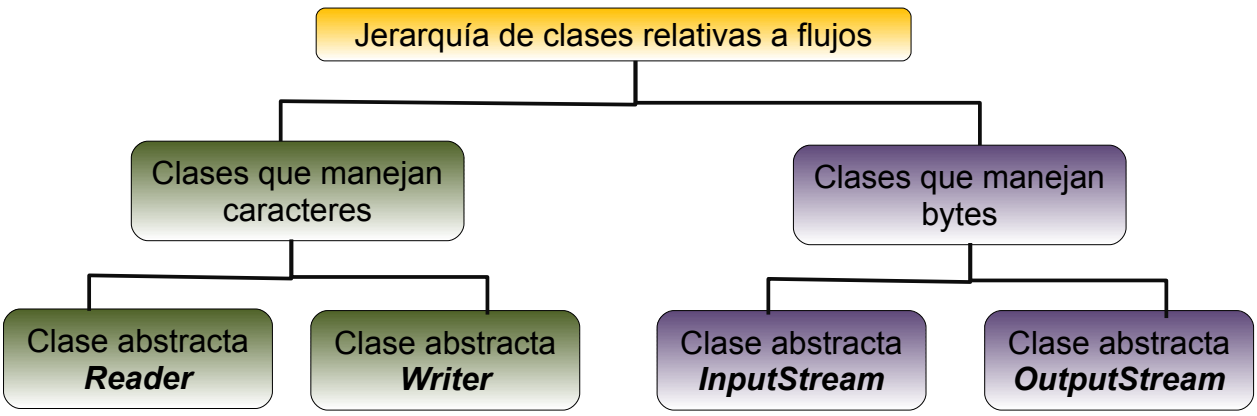
Existen dos tipos de flujos bien diferenciados, los de entrada de datos y los de salida de datos. Si somos capaces de controlar ambos podemos leer datos de cualquier fuente y escribir también en cualquier fuente, siempre que el hardware nos lo permita.

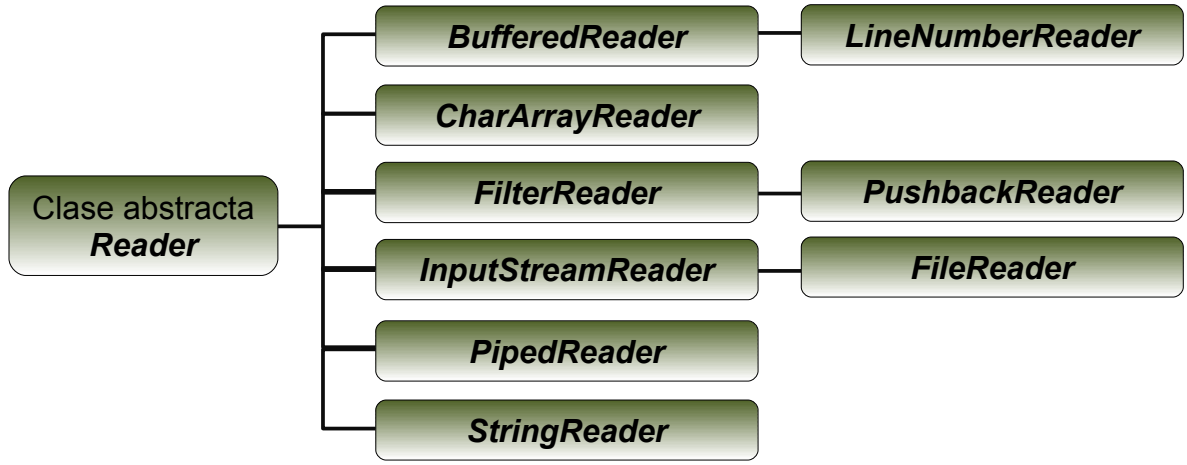
En Java disponemos de diferentes objetos que nos ayudan en esta tarea facilitándonos el acceso a la lectura y escritura. Entre estos objetos encontramos 3 que vienen predefinidos que corresponden a:

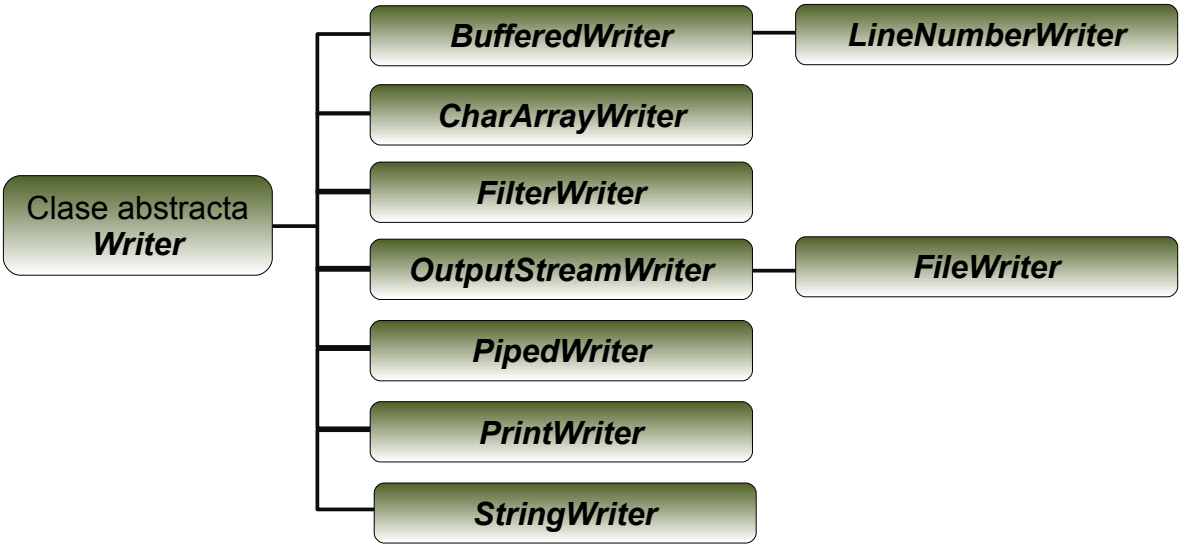
- System.in: Hace referencia a la entrada estándar de datos.
- System.out: Hace referencia a la salida estándar de información.
- System.err: Hace referencia a la salida estándar de información de errores.

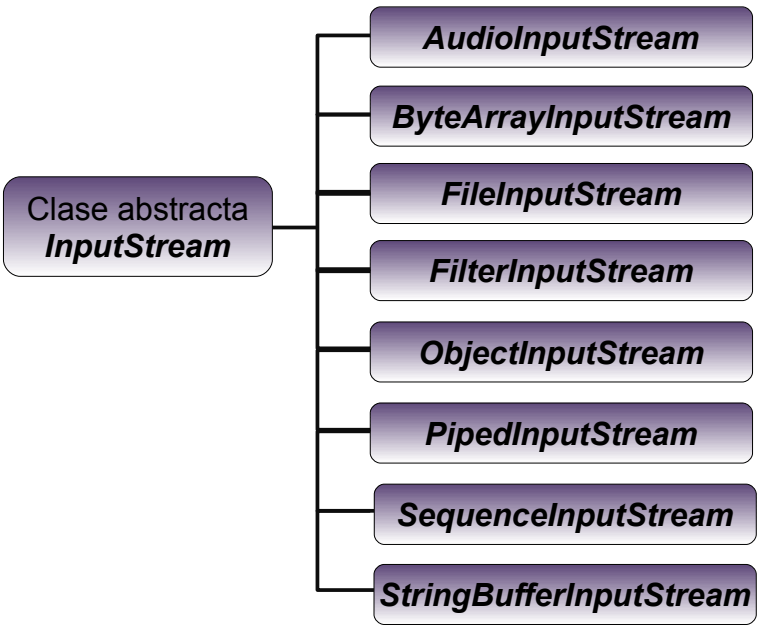
Así mismo en Java también disponemos de dos partes bien diferenciadas en los flujos de entrada y en los de salida, unos están orientados a los bytes y otros a los caracteres.

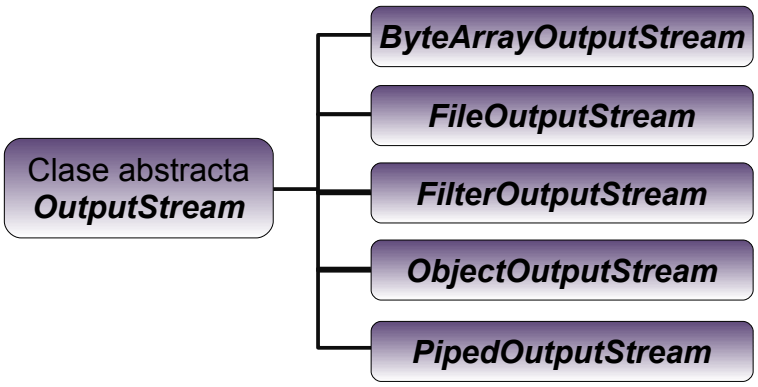
Las dos clases principales son Reader (para entradas) y Writer (para salidas) en cuanto a los caracteres e InputStream (Flujo de entrada) y OutputStream (Flujo de salida) en cuando a los bytes. De cada una de estas clases derivan otras que nos permiten refinar esta captura y nos ofrecen métodos mas específicos para leer o escribir datos. Como ejemplo se va a mostrar la jerarquía que se establece:




Clase abstracta
Reader
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
PipedReader
StringReader


Clase abstracta
Writer
BufferedWriter
LineNumberWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
PrintWriter
StringWriter


Clase abstracta
InputStream
AudioInputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream



8.2 Uso de flujos.

Mediante las clases mencionadas en el apartado anterior podemos hacer que nuestro programa se comunique con dispositivos externos. Uno de los primeros ejemplos que podemos observar para comprobar el potencial es leer el contenido de una página web. Aunque no es imprescindible comprender todo el código a continuación es importante analizarlo y ayudarse de los comentarios:

```
// Lectura de ficheros de internet
//Importamos las clases necesarias para realizar la lectura de ficheros de
internet
import java.net.MalformedURLException;
import java.net.URL;
import java.io.*;

public class ejemploLecturaWeb {
	public static void main(String[] args) {
		// direccion donde guardamos la URL
		URL direccion;
		// Capturamos las excepciones para la lectura de la url y el lector
de buffer
		try {
			// Establecemos la página que vamos a leer
			direccion = new URL("http://www.elcampico.org/index.php");
			String s = new String(); String html = new String();
			// Preparamos el buffer para la lectura de la página
			BufferedReader br = new BufferedReader(new
InputStreamReader( direccion.openStream()));
			// Leemos el fichero html hasta que llegamos al final
			while((s = br.readLine()) != null){
				html += s + "\n";
			}
			// Cerramos la lectura
			br.close();
			// Mostramos el resultado
			System.out.println(html);
		} catch (MalformedURLException e1) {
			System.err.println("Error al leer la página: ");
			e1.printStackTrace();
```

```
            } catch (FileNotFoundException e) {
                  System.err.println("Salida fichero no encontrado: " + e);
            } catch (IOException e) {
                  System.err.println("Salida error de entrada de datos: " + e);
            }
      }
```

Veremos varios ejemplos de uso. Entre ellos la comunicación con el teclado y la lectura y escritura de ficheros.

8.3 Entrada y Salida de teclado.

La entrada estándar de datos es el teclado y se representa mediante la clase "System", usando su atributo "in". Dicho atributo tiene una serie de métodos entre los cuales el mas relevante para nuestro caso va a ser:

- "int read()" que lee un byte y nos lo devuelve en código ASCII. Devuelve -1 cuando o hay más bytes para leer.

Veamos un ejemplo de uso de la misma:

```
import java.io.IOException;

public class ejemplo1 {
      public static void main(String[] args) throws IOException {
            int numChar;
            char caracter;
            System.out.print("Pulse una tecla y después intro:");
            numChar=System.in.read();
            caracter=(char)numChar;
            System.out.println("\nEl caracter es '"+caracter+
                        "' y su código UNICODE es:"+numChar);
      }
}
```

Ejercicio Auto-evaluación A8.1: *Con lo visto hasta ahora, ¿sabrías crear una clase que lea una frase y nos la muestre por pantalla?. Crea una clase llamada "ES.java" que contiene un método estático que se llame "leeDeTeclado()" que lea una frase de teclado y devuelva un String con la frase tecleada.*

Ejercicio Auto-evaluación A8.2: *Agregar a la clase "ES.java" otro método que se llame "leeDeTeclado(String)" que recibe un String con el mensaje que se muestra antes de empezar a teclear y nos devuelve el String tecleado, también agregar un método "leeNum(String)" que nos muestra el mensaje que le pasamos como parámetro y nos devuelve un número entero, y por último un método "leeNumReal(String)" que muestra el mensaje pasado como parámetro y nos vuelve un número double.* ***NOTA:*** *En caso de que el número introducido no sea válido debemos indicarlo por pantalla.*

El caso siguiente sería ver como utilizamos la salida estándar (pantalla) para mostrar

los datos, que como bien sabemos se hace con “System.out” el cual nos proporciona una serie de métodos para mostrar datos en pantalla como son “print” y “println”.

8.4 Ficheros.

Cuando ejecutamos un programa, mientras vamos introduciendo datos por teclado y almacenándolos en memoria lo estamos haciendo en una memoria volátil (RAM), si por el contrario queremos que esos datos vuelvan a estar disponibles de nuevo una vez iniciemos el programa debemos guardarlos en memoria secundaria, bien sea HDD un pendrive, CD, DVD, etc... Para ello es necesario hacer uso de los ficheros o bases de datos.

Los ficheros son una secuencia de datos almacenada en unidades de información. Cuando almacenamos un dato en un fichero a este dato se le suele llamar ***registro.*** Un registro a su vez puede estar formado por uno o varios datos que pueden ser de diferentes tipos, cada dato contenido en el registro se le conoce como ***campo.***

Por ejemplo una aplicación que almacena datos de alumnos, en este caso el registro es el alumno, y los campos son el nombre, la edad, etc...

En Java disponemos de la clase “File” la cual nos facilitará muchas tareas, tanto para recibir información acerca de los metadatos del fichero como para manipularlo. Cuando hablamos de la clase “File” no solo nos estamos refiriendo a ficheros sino también a directorios.

Veamos un sencillo ejemplo de como leer información acerca de un fichero:

```
import java.io.File;

public class ejemplo2 {
	public static void main(String[] args) {
		File f = new File("mifichero.txt");

		if(f.exists())
		{
			System.out.println("El fichero "+f.getName()+
				" ha sido abierto correctamente");
		}
		else
		{
			System.err.println("El fichero no existe.\n");
		}
		if (f.exists()){
			System.out.println("Información de '"+f.getName()+"'");
			System.out.println("La ruta es: "+f.getAbsolutePath());
			System.out.println("Tamaño: "+f.length()+" bytes.");
			System.out.println("Permite LEER:"+f.canRead()+"  ESCRIBIR:"+
					f.canWrite()+"  EJECUTAR:"+f.canExecute());
		}else{
			System.err.println("No hay información acerca del fichero.");
		}
	}
}
```

Así mismo también disponemos de multitud de métodos que nos proporciona File entre

los cuales podemos destacar los siguientes:

- isFile(): Indica si es un fichero.
- IsDirectory(): Indica si es un directorio.
- listFiles(): Devuelve un array de "Files" en caso de que este lo contenga.

Ejercicio Auto-evaluación A8.3: *Con lo visto hasta ahora crear una clase llamada GestorFicheros que tiene un método estático "abrirFichero(String)" que se le pasa un String con el nombre del fichero y nos devuelve el objeto File, y un método estático "getInfo(File)" que nos muestra la información acerca del fichero abierto, la información será: el nombre del fichero, su ruta absoluta, el tamaño y los permisos del mismo.* ***NOTA:*** *Invesitgar sobre los métodos que nos proporciona "File" para ver esta información.*

Ejercicio Auto-evaluación A8.4: *Agregar a la clase GestorFicheros un método "getList(File)" que recibe un File y en caso de ser un directorio nos muestra cuál es el contenido de ese directorio.* ***NOTA:*** *Investigar acerca del uso de "File" para tratarlo como un directorio.*

8.5 Lectura y escritura de ficheros de texto.

En este apartado nos vamos a centrar en las clases "FileReader" y "FileWriter" que nos permitirán leer y escribir respectivamente. A los constructores de ambas clases podemos pasarle tanto un objeto de tipo File con el fichero a leer o escribir, o un String con la ruta del fichero que queremos leer o escribir.

- La clase FileReader lo que hace es abrir el fichero y una vez abierto leer el fichero de forma secuencial carácter a carácter. En el caso de que el fichero no exista lanzará una excepción del tipo "FileNotFoundException".
- La clase FileWriter lo que hace es abrir el fichero y una vez abierto escribe en el fichero de forma secuencial carácter a carácter. En el caso de que el fichero no exista lo crea. En el caso de que el fichero que vamos a escribir ya exista se pueden hacer dos cosas:
  - Empezar desde el principio y borrar la información que contiene, que para ello instanciaremos el objeto de la siguiente manera:

    ```
    FileWriter f=new FileWriter("fichero.txt");
    ```

  - Añadir información al final del fichero, que para ello instanciaremos el objeto de la siguiente manera:

    ```
    FileWriter f=new FileWriter("fichero.txt",true);
    ```

Cuando abrimos un fichero es **muy importante realizar el cierre** del mismo, ya que si el fichero queda abierto podría quedar inutilizado. Para cerrar un fichero en el caso de las sentencias de arriba se haría con `f.close()`.

Para entender mejor como funcionan estas clases lo mejor es verlo con un ejemplo. Vamos a escribir y leer información de un fichero, para ello también nos valdremos de la

clase “ES.java” creada en los ejemplos:

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

public class ejemplo7 {
	public static void main(String[] args) throws IOException{
		// ESCRITURA
		FileWriter fw=new FileWriter("mifichero.txt");
		String cad="";
		int valor=0;
		cad=ES.leerDeTeclado("Escribe un texto:");
		fw.write(cad+"\n");
		fw.close();
		cad="";
		// LECTURA
		FileReader fr=new FileReader("mifichero.txt");
		while(valor!=-1){
			valor=fr.read();
			if (valor!=-1){
				cad=cad+(char)valor;
			}
		}
		System.out.println("El contenido del fichero es:\n"+cad);
		fr.close();
		// ESCRITURA AL FINAL
		fw=new FileWriter("mifichero.txt",true);
		cad="";
		valor=0;
		cad=ES.leerDeTeclado("Escribe un texto:");
		fw.write(cad+"\n");
		fw.close();
		cad="";
		// LECTURA DE NUEVO
		fr=new FileReader("mifichero.txt");
		while(valor!=-1){
			valor=fr.read();
			if (valor!=-1){
				cad=cad+(char)valor;
			}
		}
		System.out.println("El contenido del fichero es:\n"+cad);
		fr.close();
	}
}
```

Las clases “FileReader” y “FileWriter” lo que hacen es leer y escribir carácter a carácter, lo cual hace que el proceso de lectura y escritura sea lento y con continuos accesos al fichero. Para mejorar esta situación nos valdremos de las clases “BufferedReader” y “BufferedWriter” que lo que hacen es almacenar lo que vamos a leer o escribir en un buffer y posteriormente lo lee o escribe en el fichero, así podemos pensar que lo que estamos leyendo o escribiendo son cadenas de texto en vez de carácter a carácter. Para inicializar estas clases tenemos que pasarles al constructor una instancia

de “FileWriter” para el caso de “BufferedWriter” y una instancia de “FileReader” para el caso de “BufferedReader”. En nuestro caso como vemos para escribir en el ejemplo anterior escribimos una cadena completa, por tanto no sería necesario “BufferedWriter”, así que veamos un ejemplo de uso con “BufferedReader” y también se aprovechará para ver como en vez de un String con el nombre del fichero le pasamos un File aprovechando la clase GestorFicheros:

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

public class ejemplo8 {
	public static void main(String[] args) throws IOException{
		// ESCRITURA
		File f=GestorFicheros.abrirFichero("mifichero.txt");
		FileWriter fw=new FileWriter(f);
		String cad="";
		do{
			cad=ES.leerDeTeclado("Escribe (exit para salir):");
			fw.write(cad+"\n");
		}while(!cad.equals("exit"));
		fw.close();
		// LECTURA
		FileReader fr=new FileReader("mifichero.txt");
		BufferedReader br=new BufferedReader(fr);
		System.out.println("El contenido del fichero es:\n");
		while((cad=br.readLine())!=null)
		{
			System.out.println(cad);
		}
		br.close();
		fr.close();
	}
}
```

Si nos fijamos también es necesario cerrar el fujo abierto por el BufferedReader.

Ejercicio Auto-evaluación A8.5: *Agregar a la clase GestorFicheros un método “leerFicheroTxt(File)” que recibe un File y nos muestra el contenido de ese fichero, y también agregar el método “escribirFicheroTxt(File,boolean) que recibe el fichero a escribir y en el boolean recibe true o false dependiendo si queremos sobreescribir el fichero o agregar al final, nos pedirá escribir frases hasta que introduzcamos la frase “exit” la cual no debe incluirse en el fichero.*

8.6 Lectura y escritura de bytes en ficheros.

En este apartado usaremos las clases “FileInputStream” y “FileOutputStream” que nos permiten leer y escribir bytes en ficheros respectivamente. En ambos objetos existen dos

constructores principales que usaremos para abrir ficheros, podemos usar cualquiera de los dos, en uno de ellos pasaremos como parámetro el nombre del archivo, con la ruta absoluta o relativa en caso de que fuese necesario, y en el otro constructor le pasaremos un objeto File.

Para ambas clases la lectura y escritura se realiza mediante un “puntero” que apunta a al byte a leer o escribir, y se mueve secuencialmente.

FileInputStream

Cuando se trata de hacer una apertura de un fichero para la lectura (“FileInputStream”) el constructor lanzará una excepción del tipo “FileNotFoundException” en el caso de que no se encuentre el fichero.

Los principales métodos que veremos de “FileInputStream” son los siguientes:

- int read(): Devuelve en código ASCII el siguiente byte que hay después de donde está situado el puntero del fichero. Devuelve un -1 sino hay ningún byte más que leer.
- int read(byte arrayByte[]): Lee la cantidad de bytes pasada como parámetro, de tal manera que el resultado de la lectura lo obtendremos en el arrayByte. Devuelve el número de bytes leídos o -1 sino hay más que leer.
- void close(): cierra el fichero.

Veamos un ejemplo para leer de un fichero:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class ejemplo9 {
	public static void main(String[] args) throws IOException{
		int ch;
		try {
			FileInputStream f=new FileInputStream("mifichero.txt");
			while((ch=f.read())!=-1){
				System.out.print((char)ch);
			}
			f.close();
		} catch (FileNotFoundException e) {
			System.out.println("No se ha encontrado el fichero.");
			e.printStackTrace();
		}

	}
}
```

Si queremos por ejemplo leer varios trozos de cadena sería de la siguiente manera:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class ejemplo10 {
	public static void main(String[] args) throws IOException{
		byte ch[]=new byte[5];
```

```
		int cantidad;
		try {
			FileInputStream f=new FileInputStream("mifichero.txt");
			while((cantidad=f.read(ch))!=-1){
				// Usamos el constructor new String(bytes[],String code)
				// para convertir el array de bytes a un String con la
				// codificación UTF-8
				System.out.println("He leído "+cantidad+" caracteres:"+
						new String(ch, "UTF-8"));
				ch=new byte[5];
			}
			f.close();
		} catch (FileNotFoundException e) {
			System.out.println("No se ha encontrado el fichero.");
			e.printStackTrace();
		}
	}
}
```

FileOutputStream

Cuando creamos un objeto con esta clase lo pueden suceder dos cosas:

- El fichero existe: en tal caso podemos optar por sobrescribir o agregar información al final. Si queremos agregar información al final debemos agregar un segundo parámetro en el constructor, este es el atributo "append", si este parámetro se lo pasamos como "true" agregará información al final del fichero.
  - Sobrescribir: `FileOutputStream fos=new FileOutputStream("mifichero.txt");`
  - Agregar al final :`FileOutputStream fos=new FileOutputStream("mifichero.txt",true);`
- El fichero no existe: en este caso lo crea.

Los métodos más relevantes que usaremos en esta clase son:

- int write(int byte): Escribe el byte que recibe como argumento.
- int write(byte arrayByte[]): Escribe la cantidad de bytes pasada como parámetro, de tal manera que el resultado de la escritura es lo que hemos pasado en el arrayByte.
- void close(): cierra el fichero.

Veamos un ejemplo de escritura:

```
import java.io.FileOutputStream;
import java.io.IOException;

public class ejemplo11 {
	public static void main(String[] args) throws IOException {
		FileOutputStream fos = new FileOutputStream("mifichero.txt", true);
		String cad = "Esta es la cadena que voy \na escribir en el fichero";
		for (int i = 0; i < cad.length(); i++) {
			fos.write(cad.charAt(i));
```

```
            }
            fos.close();
        }
}
```

Ejercicio Auto-evaluación A8.6: *Realizar el ejemplo11 pero de manera que se escriban varios bytes a la vez usando el método "int write(byte arrayByte[])".* ***NOTA****: Hay que usar el método "getBytes()" del String.*

Ejercicio Auto-evaluación A8.7: *Agregar a la clase GestorFicheros un método "leerFicheroByte(File)" que recibe un File y nos muestra el contenido de ese fichero, y también agregar el método "escribirFicheroByte(File,boolean) que recibe el fichero a escribir y en el boolean recibe true o false dependiendo si queremos sobreescribir el fichero o agregar al final, nos pedirá escribir frases hasta que introduzcamos la frase "exit" la cual no debe incluirse en el fichero.*

8.7 Lectura y escritura de tipos básicos en ficheros.

En este apartado vamos a ver como podemos crear ficheros que contienen información a base de escritura de tipos básicos.

Para ello nos valdremos de la clase "DataOutputStream" para la escritura y "DataInputStream" para la lectura.

DataOutputStream

Concretamente usaremos el constructor de "DataOutputStream" que recibe un "FileOutputStream" como parámetro, que será el fichero donde vamos a escribir.

La clase "DataOutputStream" nos proporciona multitud de métodos entre los cuales vamos a destacar los siguientes:

- void writeBoolean(boolean v): Escribe un valor booleano en el fichero.
- void writeInt(int v): Escribe un valor int en el fichero.
- void writeChar(char v): Escribe un valor char en el fichero.
- void writeLong(long v): Escribe un valor long en el fichero.
- void writeDouble(double v): Escribe un valor double en el fichero.
- void writeFloat(float v): Escribe un valor float en el fichero.
- void writeUTF(String v): Escribe un valor String en formato UTF-8 en el fichero.

Veamos un ejemplo de uso:

```
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
public class ejemplo12 {
	public static void main(String[] args) throws IOException {
		FileOutputStream fos=new FileOutputStream("mifichero.txt");
		DataOutputStream dos=new DataOutputStream(fos);
		dos.writeUTF("Ejemplo String");
		dos.writeInt(5);
		dos.writeChar('a');
		dos.writeFloat(3.4f);
		dos.writeDouble(5.4);
		dos.writeLong(47836753);
		if(dos!=null)
		{
			dos.close();
			fos.close();
		}
	}
}
```

Una cosa que nos puede resultar curiosa es que si abrimos el fichero con un editor de texto veremos que aparecen símbolos que no tienen ningún sentido, pero esto es debido a que el fichero contiene los bytes con la información, sólo podremos leerlo usando de nuevo un programa que **capture esos datos en el mismo orden que fueron introducidos.**

DataInputStream

Concretamente usaremos el constructor de "DataInputStream" que recibe un "FileInputStream" como parámetro, que será el fichero donde vamos a escribir, evidentemente tendremos que controlar de nuevo la excepción "FileNotFoundException".

La clase "DataInputStream" nos proporciona multitud de métodos entre los cuales vamos a destacar los siguientes:

- boolean readBoolean(): lee y devuelve un valor booleano en el fichero.
- int readInt(): lee y devuelve un valor int en el fichero.
- char readChar(): lee y devuelve un valor char en el fichero.
- long readLong(): lee y devuelve un valor long en el fichero.
- double readDouble(): lee y devuelve un valor double en el fichero.
- float readFloat(): lee y devuelve un valor float en el fichero.
- String readUTF(): lee y devuelve un valor String en formato UTF-8 en el fichero.

Veamos un ejemplo de uso:

```
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class ejemplo13 {
	public static void main(String[] args) throws IOException {
		FileInputStream fis = null;
		DataInputStream dis = null;
```

```
            fis = new FileInputStream("mifichero.txt");
            dis = new DataInputStream(fis);
            System.out.println("String:" + dis.readUTF() +
                        "\nInt:"+ dis.readInt() +
                        "\nChar:"+ dis.readChar() +
                        "\nFloat:"+ dis.readFloat() +
                        "\nDouble:"+ dis.readDouble() +
                        "\nLong:"+ dis.readLong());
            if (dis != null) {
                  dis.close();
                  fis.close();
            }
      }
}
```

Como se podrá observar es **muy importante hacer la lectura en el mismo orden en el que se ha escrito.**

Ejercicio Auto-evaluación A8.8: *Comprueba que sucede con el ejemplo13 si hacemos una lectura en un orden diferente al establecido. ¿Qué resultados has obtenido?.*

8.8 Lectura y escritura de Objetos en ficheros. Persistencia. Serialización.

La persistencia se refiere a la propiedad de los datos para que estos sobrevivan de alguna manera. Por tanto la persistencia con objetos consiste en almacenar objetos, en nuestro caso en ficheros, para su posterior uso.

Un dato importante a la hora de realizar la persistencia de objetos es que todo aquel objeto que quiera ser almacenado debe **obligatoriamente implementar la interfaz "Serializable" que nos proporciona "java.io.Serializable".** Lo que hacemos es convertir un objeto a una secuencia de bytes para que pueda ser escrito en el fichero.

Para poder escribir los objetos a ficheros usaremos la clase "ObjectOutputStream" y para leerlos usaremos la clase "ObjectInputStream". Para escribir objetos usaremos el método "writeObject()" y para leerlos el método "readObject()".

Veamos un ejemplo de como escribir objetos persona en un fichero. Primero la clase persona debe implementar la interfaz Serializable:

```
import java.io.Serializable;

public class Persona implements Serializable{
      private String nombre;
      private int edad;
      public Persona(String n,int e){
            nombre=n;
            edad=e;
      }
      public String getNombre() {
```

```
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public void mostrar(){
		System.out.println("Nombre:"+nombre+"\tedad:"+edad);
	}
}
```

Ahora veamos un ejemplo de como escribimos 3 personas en un fichero:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class ejemplo14 {
	public static void main(String[] args) throws IOException {
		FileOutputStream fos=new FileOutputStream("personas.dat");
		ObjectOutputStream oos=new ObjectOutputStream(fos);
		Persona p=new Persona("Andres",30);
		oos.writeObject(p);
		oos.writeObject(new Persona("Maria",28));
		oos.writeObject(new Persona("Paco",25));
		oos.flush();
		if(oos!=null)
		{
			oos.close();
			fos.close();
		}
	}
}
```

Veamos ahora como podemos recuperar de nuevo esas 3 personas y usarlas:

```
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class ejemplo15 {
	public static void main(String[] args) throws IOException {
		FileInputStream fis = null;
		ObjectInputStream ois = null;
		try {
			fis = new FileInputStream("personas.dat");
			ois = new ObjectInputStream(fis);
			Persona p;
```

```
                while(true){
                        p=(Persona)ois.readObject();
                        p.mostrar();
                }
            } catch (EOFException e) {
                System.out.println("Se han leído todos los registros...");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }finally{
                if(ois!=null)
                {
                        ois.close();
                        fis.close();
                }
            }
        }
}
```

En este último ejemplo podemos ver que para leer varios registros tenemos que usar un “while(true)”, es un while que se ejecuta continuamente, este while para cuando se produce la excepción “EOFException”, EOF viene de “EndOfFile” (fin de fichero), por eso ponemos el mensaje en el que indicamos que se han leído todos los registos. Así mismo también hemos de capturar la excepción “ClassNotFoundException” porque cuando hacemos el casting a “Persona” sino se encuentra la clase que coincida se producirá esta excepción.

También se ha usado este ejemplo para mostrar un posible uso del “finally”.

8.9 Ejercicios Propuestos

1. En el ejercicio A8.1 y A8.2 cuando en la frase que tecleamos introducimos una ñ o acentos al imprimirlos aparecen con símbolos diferentes, esto es debido a la codificación. Investiga como solucionar el problema y modifica la clase "ES.java" para solucionar el problema.

2. Teniendo en cuenta las clases FileWriter y FileReader, crear una clase llamada "CopiaTXT" que al ejecutarla nos pide un nombre de fichero de texto origen y un nombre de fichero destino, y realiza una copia exacta del origen en el fichero destino.

3. Teniendo en cuenta que las clases FileOutputStream y FileInputStream no solo sirven para leer o escribir ficheros de texto, realizar una clase llamda "CopiaTODO" que al ejecutarla nos pide un nombre de fichero de origen y un nombre de fichero destino, y realiza una copia exacta del origen en el fichero destino. ***NOTA**: Tener en cuenta que el fichero origen puede ser cualquier tipo de fichero, una imagen, un PDF, etc...*

4. Teniendo en cuenta lo visto con las clases "DataOutputStream" y "DataInputStream" crear una clase llamada AgendaTelefonos que escribe en un fichero llamado "agenda.dat" un nombre de una persona y su número de teléfono. La clase AgendaTelefonos va a tener un método "insertar" que nos pide el nombre y el número de teléfono y a continuación lo agrega en el archivo "agenda.dat". También tendremos un método llamado "listar" que abre el fichero "agenda.dat" y nos lista por pantalla todos los usuarios de la agenda con su respectivo teléfono.

5. Teniendo en cuenta lo visto con las clases "ObjectOutputStream" y "ObjectInputStream" crear una clase PuntosDeInteres que va a proporcionar métodos para leer puntos GPS de un fichero de texto. Primero debemos tener una clase PuntoGPS que va a contener un nombre, una longitud (lleva decimales), una latitud (lleva decimales) y un tipo que va a ser "Restaurante", "Gasolinera" o "Museo". La clase PuntosDeInteres va a tener un método guardarPunto(PuntoGPS) que va a guardar en un fichero llamado "puntos.dat" el PuntoGPS que se le pasa como parámetro. La clase PuntosDeInteres además va a tener un método "listarPuntos()" que va a listar todos los PuntosGPS que contenga el fichero "puntos.dat".

8.10 Solución a los ejercicios de Auto-evaluación

Solución Auto-evaluación A8.1:

```
import java.io.IOException;

public class ES {
	public static String leerDeTeclado(){
		String cadena="";
		char c=' ';
		do{
			try {
				c=(char)System.in.read();
			} catch (IOException e) {
				e.printStackTrace();
			}
			// NO INCLUÍMOS EL SALTO DE LÍNEA NI EL RETORNO DE CARRO
			if ((int)c!=13 && (int)c!=10){
				cadena=cadena+c;
			}
		}while(c!='\n');
		return cadena;
	}

	public static void main(String[] args) {
		String prueba;
		prueba=leerDeTeclado();
		System.out.println(prueba);
	}
}
```

Solución Auto-evaluación A8.2:

```
import java.io.IOException;

public class ES {
	public static String leerDeTeclado(){
		String cadena="";
		char c=' ';
		do{
			try {
				c=(char)System.in.read();
			} catch (IOException e) {
				e.printStackTrace();
			}
			if ((int)c!=13 && (int)c!=10){
				cadena=cadena+c;
			}
		}while(c!='\n');
		return cadena;
	}

	public static String leerDeTeclado(String msg){
		System.out.print(msg);
```

```
		return leerDeTeclado();
	}

	public static int leerNum(String msg){
		String num;
		int n=-1;
		num=leerDeTeclado(msg);
		try{
			n=Integer.parseInt(num);
		}catch (NumberFormatException e){
			System.err.println("Error!, no es un número!");
		}
		return n;
	}

	public static double leerNumReal(String msg){
		String num;
		double n=-1;
		num=leerDeTeclado(msg);
		try{
			n=Double.parseDouble(num);
		}catch (NumberFormatException e){
			System.err.println("Error!, no es un número!");
		}
		return n;
	}
	public static void main(String[] args) {
		String prueba;
		int num;
		prueba=leerDeTeclado("Escribe una cadena:");
		System.out.println(prueba);
		num=leerNum("Escribe un número:");
		System.out.println("El número introducido es:"+num);
	}
}
```

Solución Auto-evaluación A8.3:

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

public class GestorFicheros {

	public static File abrirFichero(String ruta)
	{
		File f = new File(ruta);

		if(f.exists()==false)
		{
			System.err.println("El fichero no existe.\n");
		}
		else
		{
			System.out.println("El fichero "+f.getName()+
```

```
                        " ha sido abierto correctamente");
            }
            return f;
      }

      public static void getInfo(File fichero)
      {
            System.out.println("Información de '"+fichero.getName()+"'");
            System.out.println("La ruta es: "+fichero.getAbsolutePath());
            System.out.println("Tamaño: "+fichero.length()+" bytes.");
            System.out.println("Permite LEER:"+fichero.canRead()+
                              "  ESCRIBIR:"+fichero.canWrite()+
                              "  EJECUTAR:"+fichero.canExecute());
      }

      public static void getList(File directorio)
      {
            int i=0;
            File lista[];
            lista=directorio.listFiles();
            System.out.println("Contenido del directorio '"+
                                    directorio.getPath()+"':");
            for(i=0;i<lista.length;i++)
            {
                  System.out.println(lista[i].getName()+"\t"+lista[i].length()+
                        " bytes\t"+"R:"+lista[i].canRead()+"\tW:"+
                              lista[i].canWrite()+"\tX:"+lista[i].canExecute());
            }
      }
      public static void main(String[] args) {
            File fichero=
                  GestorFicheros.abrirFichero("mifichero.txt");
            GestorFicheros.getInfo(fichero);
      }
}
```

Solución Auto-evaluación A8.4:

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

public class GestorFicheros {

      // AQUÍ DEBERÍAN ESTAR LOS MÉTODOS abrirFichero y getInfo

      public static void getList(File directorio)
      {
            int i=0;
            File lista[];
            lista=directorio.listFiles();
            System.out.println("Contenido del directorio '"+
                                    directorio.getPath()+"':");
            for(i=0;i<lista.length;i++)
            {
```

```
                System.out.println(lista[i].getName()+"\t"+lista[i].length()+
                        " bytes\t"+"R:"+lista[i].canRead()+"\tW:"+
                                lista[i].canWrite()+"\tX:"+lista[i].canExecute());
            }
        }
        public static void main(String[] args) {
            File fichero=
                GestorFicheros.abrirFichero("/Users/andres/Desktop");
            GestorFicheros.getList(fichero);
        }
}
```

Solución Auto-evaluación A8.5:

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

public class GestorFicheros {

    // AQUÍ DEBERÍAN ESTAR LOS MÉTODOS abrirFichero, getInfo y getList

    public static void leerFicheroTxt(File f)
    {
        try {
            String s="";
            System.out.println("********* Contenido de "+f.getName());
            FileReader fr=new FileReader(f);
            BufferedReader br=new BufferedReader(fr);
            while((s=br.readLine())!=null)
            {
                System.out.println(s);
            }
            br.close();
            fr.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void escribirFicheroTxt(File f, boolean agregar)
    {
        try {
            String s;
            FileWriter fw=new FileWriter(f,agregar);
            System.out.println("Escribiendo en fichero "+f.getName()+
                    " Agregando al final:"+agregar+" ('exit' para salir):");
            while(!(s=ES.leerDeTeclado()).equals("exit"))
            {
```

```
                        fw.write(s+"\n");
                  }
                  fw.close();
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }
      public static void main(String[] args) {
            File fichero=
                  GestorFicheros.abrirFichero("mifichero.txt");
            escribirFicheroTxt(fichero, true);
            leerFicheroTxt(fichero);
      }
}
```

Solución Auto-evaluación A8.6:

```
import java.io.FileOutputStream;
import java.io.IOException;

public class ejemplo11b {
      public static void main(String[] args) throws IOException {
            FileOutputStream fos = new FileOutputStream("mifichero.txt", true);
            String cad = "Esta es la cadena que voy \na escribir en el fichero";
            byte ch[]=cad.getBytes();
            fos.write(ch);
            fos.close();
      }
}
```

Solución Auto-evaluación A8.7:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;

public class GestorFicheros {

      // AQUÍ DEBERÍAN ESTAR LOS MÉTODOS abrirFichero, getInfo, getList,
      // leerFicheroTxt y escribirFicheroTxt

      public static void leerFicheroByte(File f)
      {
            try {
                  System.out.println("********* Contenido de "+f.getName());
                  FileInputStream fis=new FileInputStream(f);
                  byte ch[]=new byte[8];
                  while((fis.read(ch))!=-1){
                        System.out.print(new String(ch, "UTF-8"));
                        ch=new byte[8];
```

```
                }
                fis.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void escribirFicheroByte(File f, boolean agregar)
        {
            try {
                String s;
                FileOutputStream fos=new FileOutputStream(f,agregar);
                System.out.println("Escribiendo en fichero "+f.getName()+
                        " Agregando al final:"+agregar+" ('exit' para salir):");
                while(!(s=ES.leerDeTeclado()).equals("exit")){
                    fos.write((s+"\n").getBytes());
                }
                fos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        public static void main(String[] args) {
            File f=GestorFicheros.abrirFichero("ejemplo_.txt");
            GestorFicheros.escribirFicheroByte(f, true);
            GestorFicheros.leerFicheroByte(f);
        }
}
```

Solución Auto-evaluación A8.8:

```
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class ejemplo13 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = null;
        DataInputStream dis = null;
        fis = new FileInputStream("mifichero.txt");
        dis = new DataInputStream(fis);
        System.out.println("String:" + dis.readUTF() +
                "\nDouble:"+ dis.readDouble() +
                "\nInt:"+ dis.readInt() +
                "\nFloat:"+ dis.readFloat() +
                "\nChar:"+ dis.readChar() +
                "\nLong:"+ dis.readLong());
        if (dis != null) {
            dis.close();
            fis.close();
        }
```

```
        }
}
```

Lo que sucede es que se obtienen valores incorrectos como estos:

```
String:Ejemplo String
Double:1.0613127865E-313
Int:-1717944299
Float:-1.5881867E-23
Char:?
Long:47836753
```