# UD1 - Manejo de ficheros

ACCESO A DATOS

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

# Vamos a aprender a

Desarrollar aplicaciones que gestionan **información almacenada en ficheros**, identificando el campo de aplicación de los mismos y utilizando clases específicas.

# Objetivos de aprendizaje

1. Utilizar clases para la gestión de ficheros y directorios.
2. Valorar las ventajas y los inconvenientes de las distintas formas de acceso.
3. Utilizar clases para recuperar información almacenada en un fichero XML.
4. Utilizar clases para almacenar información en un fichero XML.
5. Utilizar clases para convertir a otro formato información contenida en un fichero XML.
6. Prever y gestionar las excepciones.
7. Probar y documentar las aplicaciones desarrolladas.

# FICHEROS

- Un fichero es un conjunto de datos homogéneos almacenados en un soporte externo permanente (disco duro, CD, pendrive, ...).

- Clasificación de los ficheros
  - Según su contenido.
  - Según su sistema de organización y
  - método de acceso a sus componentes.
  - Según su relación con el programa.

# FICHEROS - CLASIFICACIÓN

- Según su contenido, se pueden distinguir diferentes tipos:
  - **Ficheros de texto:** en los cuales sus componentes o elementos son caracteres dispuestos en líneas.
  - **Ficheros de registros:** los mas clásicos en informática, en los cuales los componentes son registros, los cuales son un conjunto de datos llamados campos, pertenecientes a una misma entidad.
  - **Ficheros de objetos:** donde los componentes del fichero son objetos de una misma clase.

# FICHEROS - CLASIFICACIÓN

- Según su sistema de organización y método de acceso a sus componentes se clasifican en:
  - **Ficheros Secuenciales**: en la que sus componentes se almacenan de forma consecutiva o secuencial, y que para acceder a un componente hay que procesar a todos los componentes que le preceden en dicho fichero
  - **Ficheros Directos/Aleatorios/Relativos**: este tipo de ficheros permiten el acceso a un componente en base a la posición relativa que ocupa dicho componente en el fichero.
  - **Ficheros Indexados**: este tipo de ficheros permiten el acceso a sus componentes en base a una clave que permite diferenciar a cada componente del resto.

# FICHEROS - CLASIFICACIÓN

- Atendiendo a su relación con el programa se clasifican en:
  - **Ficheros de Entrada o lectura**: aportan o envían información al programa.
  - **Ficheros de Salida o escritura**: reciben información desde el programa.
  - **Ficheros de Entrada /Salida**: intercambian información con el programa en ambos sentidos.

# FICHEROS

- Conjunto de bits
- Nombre (único)
- Directorio
- Tipo (extensiones, .java, .doc, .pdf)

# CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE ARCHIVOS

UD1: MANEJO DE FICHEROS

# Clase File en Java

- Sirve para obtener información sobre archivos y directorios
- Se emplea para **crear o eliminar archivos y directorios**
- **Objeto de la clase File** → archivo o directorio
- Constructores:

```
public File(String nombreFichero|directorio);
public File(String directorio, String nombreFichero|directorio);
public File(File directorio, String nombreFichero|directorio);
```

Directorio o path: ruta relativa o absoluta

# Clase File en Java

```
1. public File(String nombreFichero|directorio);
```

```
File f = new File(“archivo.txt");

File f = new File(“documentos/archivo.txt")

File f = new File("c:/documentos/archivo.txt");
```

```
2. public File(String directorio, String nombreFichero|directorio);
```

```
File f = new File(“documentos", “archivo.txt" );

File f = new File("/documentos", “archivo.txt" );
```

```
3. public File(File directorio, String nombreFichero|directorio);
```

```
File ruta = new File(“documentos");
File f = new File(ruta, “archivo.txt" );
```

```
File ruta = new File("/documentos");
File f = new File(ruta, “archivo.txt" );
```

# Clase File en Java

```
import java.io.*;
public class VerDir {
    public static void main(String[] args) {
        System.out.println("Archivos en el directorio actual:");
        File f = new File(".");
        String[] archivos = f.list();
        for (int i = 0; i < archivos.length; i++)
        {
            System.out.println(archivos[i]);
        }
}
```

*Devuelve un array de Strings con los nombres de los ficheros y directorios asociados al objeto* `File`

| NOMBRE | DESCRIPCIÓN | DATO DEVUELTO |
|---|---|---|
| exists | Indica si existe o no el fichero. | boolean |
| isDirectory | Indica si el objeto File es un directorio. | boolean |
| isFile | Indica si el objeto File es un fichero. | boolean |
| isHidden | Indica si el objeto File esta oculto. | boolean |
| getAbsolutePath | Devuelve una cadena con la ruta absoluta del fichero o directorio. | String |
| canRead | Indica si se puede leer. | boolean |
| canWrite | Indica si se puede escribir. | boolean |
| canExecute | Indica si se puede ejecutar. | boolean |
| getName | Devuelve una cadena con el nombre del fichero o directorio. | String |
| getParent | Devuelve una cadena con el directorio padre. | String |
| listFiles | Devuelve un array de File con los directorios hijos. Solo funciona con directorios. | Array de File |
| list | Devuelve un array de String con los directorios hijos. Solo funciona con directorios. | Array de String |
| mkdir | Permite crear el directorio en la ruta indicada. Solo se creara si no existe. | boolean |
| mkdirs | Permite crear el directorio en la ruta indicada, también crea los directorios intermedios. Solo se creara si no existe. | boolean |
| createNewFile | Permite crear el fichero en la ruta indicada. Solo se creara si no existe. Debemos controlar la excepción con IOException. | boolean |

# Clase File en Java

**Actividad 1**:

Realiza un programa Java que muestre la siguiente información de un fichero (VerInf.java):

- Nombre
- Ruta
- Ruta absoluta
- Tamaño
- ¿lectura?
- ¿escritura?
- ¿es un
- directorio?
- ¿es un fichero?

El nombre del fichero se le pasará al programa desde la línea de comandos.

- Nombre del fichero: **VerInf.java**

# Clase File en Java -- Actividad

```
import java.io.*;
public class VerInf {
    public static void main(String[] args) {
        System.out.println("INFORMACIÓN SOBRE EL FICHERO:");
        File f = new File("VerInf.java");
        if(f.exists()){
            System.out.println("Nombre del fichero  : "+f.getName());
            System.out.println("Ruta               : "+f.getPath());
            System.out.println("Ruta absoluta      : "+f.getAbsolutePath());
            System.out.println("Se puede escribir  : "+f.canRead());
            System.out.println("Se puede leer      : "+f.canWrite());
            System.out.println("Tamaño             : "+f.length());
            System.out.println("Es un directorio   : "+f.isDirectory());
            System.out.println("Es un fichero      : "+f.isFile());
        }
    }
}
```

# Clase File en Java

**Actividad 2**:

- Realiza un programa Java que cree un directorio en el directorio actual, a continuación crea dos ficheros en el directorio.
- Elimina uno de los ficheros y elimina el directorio

➢ Nombre del fichero: **CrearDir.java**

Utiliza estos métodos:

**mkdir()**
File d = new File (“NUEVODIR”);
d.mkdir(); //Crear directorio

**createNewFile()**
File f = new File (d, “fichero1.txt”);
f.createNewFile()- IOException, try/catch

**delete()** Para poder borrar un directorio antes hay que eliminar los ficheros que contiene.

# FLUJOS O STREAMS

UD1: MANEJO DE FICHEROS

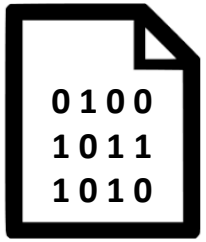
# Flujos o Streams

Sirven para enviar información desde un programa de Java a un sitio remoto (o local).

¿Cómo abordar estos flujos o streams?

- Como flujo de caracteres


- Como flujo de bytes




Si no es necesario leer la información, es más cómodo enviarlo como **archivo binario**

# Flujos o Streams

Sirven para enviar información desde un programa de Java a un sitio remoto (o local).

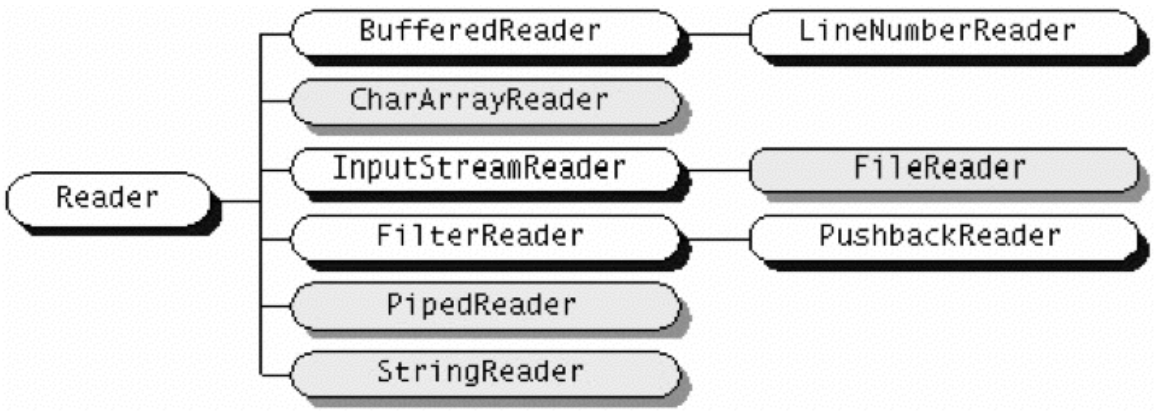
Los programas pueden leer datos de los ficheros del sistema, así como escribir en ellos.

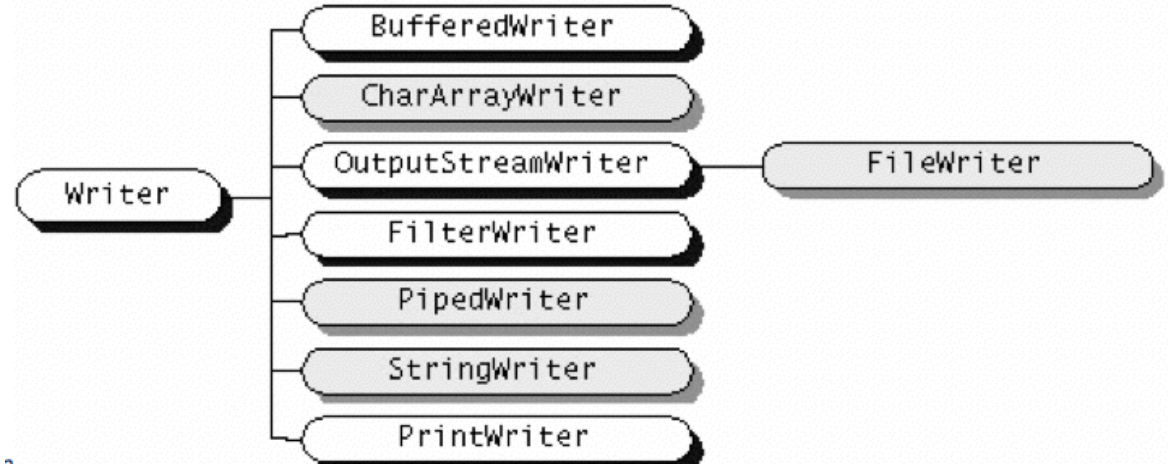
Java aporta en su paquete **java.io** varias clases para estas tareas:

- **File**: Ficheros, tanto directorios como ficheros finales
- **Reader**: lectura de caracteres
- **Writer**: escritura de caracteres
- **InputStream**: lectura de bytes
- **OutputStream**: escritura de bytes

# Reader/Writer

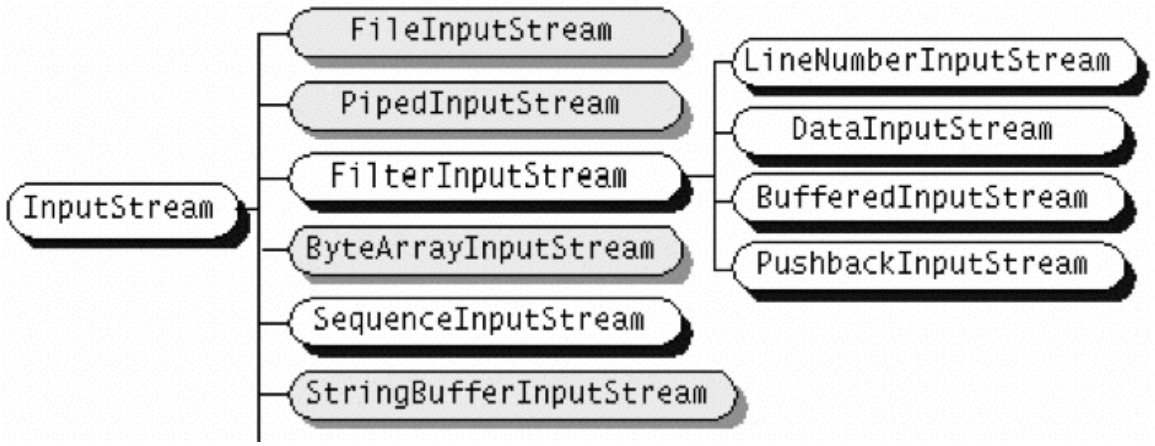
Realizan operaciones de **entrada/salida** de caracteres. Éstas clases existen para poder manejar caracteres *UNICODE*

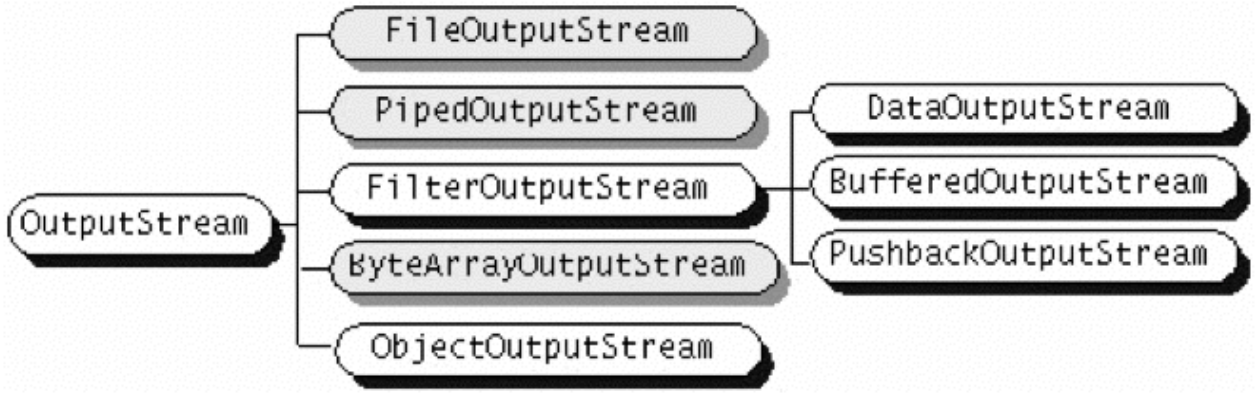


# InputStream/OutputStream

Realizan operaciones de entrada/salida de bytes y su uso está orientado a la **lectura/escritura** de datos binarios



# InputStreamReader/InputStreamWriter

Se puede pasar de un flujo de bytes a uno de caracteres con

- **InputStreamReader**
- **OutputStreamWriter**

# FORMAS DE ACCESO A UN ARCHIVO

UD1: MANEJO DE FICHEROS

# Formas de acceso a un archivo

**Acceso secuencial**:

Los datos se leen y escriben en orden. Para acceder a un dato que está en la mitad del fichero hay que leer los datos anteriores.

La escritura de datos se hará a partir del último dato escrito.

- `FileReader` y `FileWriter`
- `FileInputStream` y `FileOutputStream`

# Formas de acceso a un archivo

**Acceso secuencial**:

Los ficheros secuenciales se utilizan en aplicaciones de proceso por lotes, como por ejemplo en el respaldo de los datos o backups.

Ventajas
- Rápida capacidad de acceso al siguiente registro
- Aprovechan mejor la utilización del espacio
- Son fáciles de utilizar

Desventaja:
- No se puede acceder directamente a un registro determinado

# Formas de acceso a un archivo

**Acceso directo o aleatorio**:

Permite acceder directamente a un dato sin necesidad de leer los anteriores y se puede acceder a la información en cualquier orden.

- `RandomAccessFile`

# Formas de acceso a un archivo

Ventaja:

- Rápido acceso a una posición determinada

Inconvenientes:

- Ocupa más espacio que un archivo secuencial debido al uso de índices
- Desaprovecha el espacio destinado al fichero ya que pueden existir huecos entre registros
- A veces, necesidad de hardware más sofisticado

# OPERACIONES SOBRE UN FICHERO

UD1: MANEJO DE FICHEROS

# Operaciones sobre un fichero

- Creación del fichero
- Apertura del fichero
- Cierre del fichero
- Lectura de los datos del fichero
- Escritura de datos en el fichero
- Altas
- Bajas
- Modificaciones

# Operaciones sobre un fichero

**Operaciones sobre archivos secuenciales**

- **Altas**: al final del último registro insertado
- **Bajas**: leer y escribir todos los registros en un fichero auxiliar excepto el que queremos dar de baja
- **Modificaciones**: localizar el registro a modificar, realizar la modificación y utilizar un fichero auxiliar que incluya el cambio

# CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS

UD1: MANEJO DE FICHEROS

# Tipos de ficheros

- **Texto:** Almacenan caracteres alfanuméricos ASCII, UNICODE, UTF8
  - `FileReader` (*`FileNotFoundException`*, el fichero no existe o no es válido)
  - `FileWriter` (*`IOException`*, disco lleno o protegido contra escritura. Si no existe el fichero lo crea)

- **Binario:** Almacenan secuencias de bits
  - `FileInputStream` (*`FileNotFoundException`*)
  - `FileOutputStream` (*`FileNotFoundException`*)

# Tipos de ficheros

**Métodos de lectura**: devuelve el número de caracteres leídos o -1 si ha llegado al final

FileReader

- int read(): lee un carácter y lo devuelve

# CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS

**FileReader**

En un programa Java (1) para crear o abrir un fichero se invoca a la **clase File** y a continuación, (2) se crea el flujo de entrada hacia el fichero con la clase **FileReader**. Después se realizan (3) las operaciones de lectura (o escritura) y cuando terminemos de usarlo lo (4) cerraremos mediante el método **close()**

# CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS

```
import java.io.*;
public class LeerFichTexto {
    public static void main(String[] args) throws IOException {
1       File fichero = new File(".\\ruta\\fichero.txt");
        //declarar fichero
2       FileReader fic = new FileReader(fichero); //crear el flujo de entrada
        int i;
3       while ((i = fic.read()) != -1) //se va leyendo un carácter
            System.out.println((char) i);
4       fic.close(); //cerrar flujo de entrada
    }
}
```

# CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS

**Actividad 3**

- Crea un fichero de texto con algún editor de textos y después realiza un programa Java que visualice su contenido.

➢ Nombre del fichero: **LeerFichTexto.java**

# FileReader

**Métodos de lectura**: devuelve el número de caracteres leídos o -1 si ha llegado al final

`FileReader`

**int read()**: lee un carácter y lo devuelve,devuelve un número que si le hacemos un casting a char este será legible por nosotros.

```
while ((i = fic.read()) != -1) //se va leyendo un caracter
   System.out.println((char) i);
```

# FileReader

**Actividad**

Encuentra cuáles son los **métodos de lectura** que proporciona la clase **FileReader** y explica el funcionamiento de cada uno de ellos

# FileReader

**int read()**: lee un carácter y lo devuelve,devuelve un número que si le hacemos un casting a char este será legible por nosotros.

```
while ((i = fic.read()) != -1)
    System.out.println((char)i);
```

**int read(char[] buffer):** lee hasta buff.length caracteres. Los caracteres leídos del fichero se van almacenando en buff

```
char buffer[] =new char [20];
//pinta buffer.length caracteres
while ((fic.read(b)) != -1)
    System.out.println(b);
```

# FileReader

**int read(char[] buffer, int offset, int n)**: lee n caracteres del buffer comenzado por buffer[offset] y devuelve el número de caracteres leído

```
File fis = new File("ejemplo");
FileReader isr = new FileReader(fis);
char[] cbuf = new char[10];
// lee los datos en el buffer
i = isr.read(cbuf, 2, 5);
for(char c:cbuf)
{
    // caracteres vacios
    if(((int)c)==0)
        c='-';
    System.out.println(c);
}
```

# FileReader

```
import java.io.*;
public class actividad2 {
    public static void main(String[] args) {
        if (args.length!=1) {
            System.out.println("HAY QUE INTRODUCIR UN ARGUMENTO....") ;
        }else{
            try {
                String entrada=args[0];
                File fichero = new File(entrada);//"ejemplofichtexto.txt";
                FileReader fic = new FileReader(fichero);
                int i;
                //pinta uno a uno los caracteres
                while ((i = fic.read()) != -1)
                System.out.println((char) i);
```

# FileReader

**Actividad 3-1**

Modifica el programa anterior para que lea los caracteres 20 en 20.

Crea un fichero de texto con algún editor de textos y después realiza un programa Java que visualice su contenido. Cambia el programa Java para que el nombre del fichero se acepte al ejecutar desde la línea de comandos.

- Nombre del fichero: **LeerFichTexto2.java**

# BufferedReader

FileReader **no** contiene métodos que permitan **leer líneas completas.** Necesitamos la clase **BufferedReader**

**String readLine()**: lee una línea del fichero y la devuelve, o devuelve nulo si no hay nada que leer o hemos llegado al final del fichero

Para construir un BufferedReader necesitamos la clase FileReader:

```
BufferedReader f = new BufferedReader(new FileReader(nombrefic));
```

# BufferedReader

**Actividad 3-2**

Basándote en la actividad 3, crea un programa que lea un fichero línea por línea y que las vaya visualizando en pantalla.

Utiliza el método `readLine()` para ello.

Intenta agrupar las instrucciones dentro de un bloque **try-catch**

**Excepciones: FileNotFoundException** y **IOException**

- Nombre del fichero: **LeerFichTexto3.java**

# BufferedReader

```
try {
        FileReader fr = new FileReader("fichero.txt");
        BufferedReader br = new BufferedReader(fr);

        String linea;

        while((linea = br.readLine())!= null)
            System.out.println(linea);

        br.close();
}
catch (FileNotFoundException fn) {
            System.out.println("Error de lectura");
}

catch (IOException io){
            System.out.println("Error de E/S");
}
```

# FileWriter

Si el fichero no existe lo crea

- IOException
  - protegido contra escritura
  - no existe y no se puede crear
  - no se puede abrir por cualquier otra razón

Para escribir, usaremos el método ***write*** de **FileWriter**, este método puede usar como parámetro un String con lo que queremos escribir o un número que se corresponderá un carácter de la tabla ASCII

# FileWriter

**Métodos de escritura**: pueden lanzar `IOException`

- **void write(int c)**: escribe un carácter
- **void write (char[] buf)**: escribe un array de caracteres
- **void write (char[] buf,int desplazamiento, int n)**: escribe n caracteres de datos en buf y comieza por buf[desplazamiento]
- **void write(String str)**: escribe una cadena de caracteres
- **append(char c)**: añade un carácter a un fichero

# FileWriter

Hay que tener en cuenta que si el fichero existe cuando vayamos a escribir caracteres todo lo que tenía almacenado anteriormente se eliminará.

Si queremos añadir caracteres al final utilizaremos la clase `FileWriter` de la siguiente manera:

```
FileWriter f = new FileWriter (fichero, true)
```

*Si es true escribirá al final en lugar de al principio*

**Opción A)**

```
FileWriter f = new FileWriter (fichero, true)
```

**Opción B)**

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

# FileWriter

## FileWriter

```
public FileWriter(String fileName,
          boolean append)
           throws IOException
```

Constructs a FileWriter object given a file name with a boolean indicating whether or not to append the data written.

**Parameters:**

`fileName` - String The system-dependent filename.

`append` - boolean if `true`, then data will be written to the end of the file rather than the beginning.

**Throws:**

`IOException` - if the named file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason

# Escritura

1. Declarar el fichero mediante la clase File
2. Crear flujo de salida mediante FileWriter
3. Escribir con el método write
4. Cerrar flujo de salida con el método close() (FileWriter)

# Escritura

**Actividad 4**

Crea un programa que escriba caracteres en un fichero de nombre *FichTexto.txt* (si no existe lo crea). Los caracteres se escriben uno a uno y se obtienen de un String

Nota: tenemos que convertir el String en un array de caracteres mediante el método `toCharArray()`

```
void write (char[] buf)
```

- Nombre del fichero: **EscribirFichTexto.java**

# Escritura

**Actividad 5**

Modifica el ejercicio anterior para que en el fichero se escriban cadenas de caracteres obtenidas de un array de Strings. Las cadenas se irán grabando una a una sin saltos de línea.

```
String prov[]={“gipuzkoa”,”bizkaia”,”araba”}

void write(String str)
```

- Nombre del fichero: **EscribirCadenaFich.java**

# BufferedWriter

La clase `BufferedWriter` añade un buffer para realizar la escritura eficiente de caracteres.

Tiene el método **newLine()** para poder crear nuevas líneas.

Para construir un `BufferedWriter` necesitamos la clase `FileWriter`:

```
BufferedWriter f = new BufferedWriter(new FileWriter(nombrefic));
```

```java
import java.io.*;
public class EscribirFichTextoBuf {
    public static void main(String[] args) {
        try{
            BufferedWriter fichero = new BufferedWriter(new FileWriter("FichTexto.txt"));
            for (int i=1; i<11; i++){
                fichero.write("Fila numero: "+i); //escribe una lÌnea
                fichero.newLine(); //escribe un salto de línea
            }
        fichero.close();
        }
        catch (FileNotFoundException fn ){
            System.out.println("No se encuentra el fichero");}
        catch (IOException io) {
            System.out.println("Error de E/S ");}
        }
}
```

# PrintWriter

Es una subclase de `FileWriter` que permite escribir con formato de texto, tanto en la salida estándar como en ficheros.

Métodos de PrintWriter de escritura en un fichero:

- `print(String s)`
- `println(String s)`

```
PrintWriter p = new PrintWriter(new FileWriter(nombrefic));
```

```
PrintWriter fichero = new PrintWriter(new FileWriter("FichTexto.txt"));
for (int i=1; i<11; i++){
    fichero.println("Fila numero: "+i);
}
```

Ejemplo anterior

# FICHEROS BINARIOS

UD1: MANEJO DE FICHEROS

# Ficheros binarios

Almacenan secuencias de dígitos binarios

Clases para trabajar con flujos de bytes:

- `FileInputStream (entrada)`
- `FileOutputStream (salida)`

# Ficheros binarios - FileInputStream

## FileInputStream

Los métodos de `FileInputStream` para lectura son similares a los de la clase FileReader, éstos métodos devuelven el número de bytes leídos o -1 si se ha llegado al final del fichero

- **int read():** lee un byte y lo devuelve
- **int read(byte[] b)**: lee hasta b.length bytes
- **int read(byte[] b, int desplazamiento, int n)**: lee n bytes del buffer comenzado por b[desplazamiento] y devuelve el número de bytes leídos

```
File fichero = new File(".\\src\\FichBytes.dat")
//crea flujo de salida hacia el fichero
FileOutputStream fileout = new FileOutputStream(fichero);
```

# Ficheros binarios - FileOutputStream

**FileOutputStream**

- **void write(int b)**: escribe un byte
- **void write(byte[] b)**: escribe b.length bytes
- **void write(byte[] b, int desplazamiento, int n)**: escribe n bytes del buffer comenzado por b[desplazamiento]

Para añadir bytes al final del fichero:

```
FileOutputStream f = new FileOutputStream(fichero, true)
```

```
import java.io.*;
public class EscribirLeerFichBytes {
    public static void main(String[] args) throws IOException {
        File fichero = new File(".\\src\\FichBytes.dat");
        //crea flujo de salida hacia el fichero
        FileOutputStream fileout = new FileOutputStream(fichero);
        //crea flujo de entrada
        FileInputStream filein = new FileInputStream(fichero);
        int i;
        //Escribir los datos del fichero
        for (i=1; i<100; i++)
            fileout.write(i); //escribe datos en el flujo de salida
        fileout.close(); //cerrar stream de salida

        //visualizar los datos del fichero
        while ((i = filein.read()) != -1) //lee datos del flujo de entrada
            System.out.println(i);
        filein.close(); //cerrar stream de entrada
    }
}
```

# Ficheros binarios - DataInputStream

**DataInputStream**

Clase para **leer** **datos de tipos primitivos**: int, float, long…

- `boolean readBoolean();`
- `byte readByte();`
- `int readUnsignedByte();`
- `int readUnsignedShort();`
- `short readShort();`
- `char readChar();`
- `int readInt();`
- `long readLong();`
- `float readFloat();`
- `double readDouble();`
- `string readUTF();`

# Ficheros binarios - DataOutputStream

**DataOutputStream**

Clase para **escribir** **datos de tipos primitivos**: int, float, long…

- `void writeBoolean(boolean v);`
- `void writeByte(int v);`
- `void writeBytes(String s);`
- `void writeShort(int v);`
- `void writeChars(String s);`
- `void writeChar(int v);`
- `void writeInt(int v);`
- `void writeLong(long v);`
- `void writeFloat(float v);`
- `void writeDouble(double v);`
- `void writeUTF(String str);`

# Ficheros binarios – DataInputStream/DataOutputStream

```
File f = new File (“fichero.dat”;)
FileInputStream fin = new FileInputStream(f);
DataInputStream dataOS = new DataInputStream(fin);
```

```
File f = new File (“fichero.dat”;)
FileOutputStream fout = new FileOutputStream(f);
DataOutputStream dataOS = new DataOutputStream(fout);
```

Inserta datos en el fichero FichData.dat. Los datos los toma de dos arrays, uno contiene los nombres de una serie de personas y el otro de sus edades. Se recorren los arrays y se van escribiendo en el fichero el nombre (con writeUTF(String)) y la edad (con writeInt(int)).

```
import java.io.*;
public class EscribirFichData {
    public static void main(String[] args) throws IOException {

        File fichero = new File("FichData.dat");
        FileOutputStream fileout = new FileOutputStream(fichero);
        DataOutputStream dataOS = new DataOutputStream(fileout);

        String nombres[] = {"Ana","Luis, Miguel","Alicia","Pedro","Manuel","Andrés",
        "Julio","Antonio","María Jesús"};
        int edades[] = {14,15,13,15,16,12,16,14,13};

        for (int i=0;i<edades.length; i++){
            dataOS.writeUTF(nombres[i]); //inserta nombre
            dataOS.writeInt(edades[i]); //inserta edad
        }
        dataOS.close(); //cerrar stream
    }
}
```

# Ficheros binarios

**Actividad 6**

Visualiza los datos grabados en el fichero anterior. Se deben recuperar los datos en el mismo orden en el que se insertaron, es decir, primero obtendremos el nombre y luego la edad.

- Nombre de la actividad: **LeerFichData.java**

# Ficheros binarios – Objetos Serializables

Se han visto cómo se guardan los tipos de datos primitivos en un fichero, pero, por ejemplo, **si tenemos un objeto de tipo empleado con varios atributos (nombre, dirección, salario, etc.) y queremos guardarlo en un fichero**, tendríamos que guardar el atributo que forma parte del objeto por separado. Supone un problema cuando tenemos muchos objetos (empleados).

- Java nos permite guardad **objetos** en **ficheros binarios**. Para poder hacerlo el objeto tiene que implementar la interfaz **Serializable**.

- **Serializable** dispone de una serie de métodos con los que podremos guardar y leer objetos en ficheros binarios.

**void readObject ()**: para leer un objeto del ObjetctInputStream.

- IOException, ClassNotFound

**void writeObject (java.io.ObjectOuputStream stream)**: para escribir el objeto en ObjectOutputStream

- IOException

# Clases para la **lectura** de Objetos

`File`

- **`FileInputStream`**
- **`ObjectInputStream`**
- *`readObject()`*

Al acabar el fichero se produce un `java.io.EOFException`

```
File fichero = new File ("FichPersona.dat");
//Flujo de entrada
FileInputStream filein = new FileInputStream(fichero);
//Conecta el flujo de bytes al flujo de datos
ObjectInputStream dataIS = new ObjectInputStream(filein);
```

# Clases para la **escritura** de Objetos

File

- **FileOutputStream**
- **ObjectOutputStream**
- *writeObject(obj)*

```
File fichero = new File ("FichPersona.dat");
//Flujo de salida
FileOutputStream fileout = new FileOutputStream(fichero);
//Conecta el flujo de bytes al flujo de datos
ObjectOutputStream dataOS = new ObjectOutputStream(fileout);
```

# Ficheros binarios – Objetos Serializables

```
import java.io.Serializable;
public class Persona implements Serializable {
	private String nombre;
	private int edad;

	public Persona(String nombre,int edad) {
		this.nombre=nombre;
		this.edad=edad;
	}

	public Persona() {
		this.nombre=null;
	}
	public void setNombre (String nombre) {this.nombre=nombre;}
	public void setEdad (int edad) {this.edad=edad;}

	public String getNombre() {return nombre; } //devuelve nombre
	public int getEdad() {return edad; } //devuelve edad
}//fin Persona
```

Clase Persona que implementa la interfaz Serializable.

- Tiene dos atributos: nombre y edad
- Dos métodos get (obtener valor)
- Dos métodos set (dar valor)

# Ficheros binarios

**Actividad 7**

Crea un programa que escriba objetos Persona en un fichero binario y después los lea y los imprima por pantalla.

*Nota*: EscribirFichObject.java es parecido (no igual) a EscribirFichData.java hasta el “for”.

- Nombre de las actividades:
  - **Persona.java**
  - **EscribirFichObject.java**
  - **LeerFichObject.java**

# Ficheros binarios

**Actividad 8**

Crea un programa Java que cree un fichero binario para guardar datos de departamentos. Dale el nombre de “Departamentos.dat”.

Introduce varios departamentos. Los datos por cada departamento son los siguientes:

- Número de departamento: entero (tamaño: 4)
- Nombre: String (tamaño: 20)
- Localidad: String (tamaño: 20)

- **Departamentos.java**

```java
import java.io.Serializable;
class Departamento implements Serializable {
    private String nombre;
    private String loc;
    private int dep;
    public Departamento(String nombre, int dep, String loc) {
        this.nombre = nombre;
        this.dep = dep;
        this.loc = loc;
    }
    public Departamento() {
        this.nombre = null;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    public int getDep() {
        return dep;
    }
    public void setDep(int dep) {
        this.dep = dep;
    }
}
```

```
public static void Listado() throws IOException {
    Departamento dep;
    // declara el fichero
    File fichero = new File("Departamentos.dat");
    FileInputStream filein = new FileInputStream(fichero);
    // conecta el flujo de bytes al flujo de datos
    ObjectInputStream dataIS = new ObjectInputStream(filein);
    System.out.println("LISTADO DE DEPARTAMENTOS:");
    System.out.println("=====================================================");
    try {
        while (true) { // lectura del fichero
            dep = (Departamento) dataIS.readObject();
            System.out.println("Departamento: "+dep.getDep()+ ", Nombre: " + dep.getNombre() + ", Localidad:
                " + dep.getLoc());
        }
    } catch (Exception e) {
        //Se produce EOFException al finalizar la lectura
        if(e.toString()== "java.io.EOFException")
            System.err.println("Final de Lectura...");
        else System.err.println("Error al leer fichero: "+e.toString());
    }
    System.out.println("=====================================================");
    dataIS.close(); // cerrar stream de entrada
```

```
public static void main(String[] args) throws IOException {
        Departamento dep;
        // declara el fichero
        File fichero = new File("Departamentos.dat");
        FileOutputStream fileout = new FileOutputStream(fichero);
        // conecta el flujo de bytes al flujo de datos
        ObjectOutputStream dataOS = new ObjectOutputStream(fileout);
        String nombres[] = { "INFORMÁTICA", "MÁRKETING", "CONTABILIDAD", "VENTAS", "COMPRAS",      "PERSONAL",
        "RECURSOS", "ADMINISTRACIÓN", "ECONOMÍA" };
        int num[] = { 10, 15, 20, 25, 30, 35, 40, 45, 50 };
        String loc[] = { "MADRID", "SEVILLA", "LEÓN", "TOLEDO", "GUADALAJARA", "CUENCA", "OVIEDO", "BILBAO",
        "VALENCIA" };
        for (int i = 0; i < num.length; i++) { // recorro los arrays
                dep = new Departamento(nombres[i], num[i], loc[i]);
                dataOS.writeObject(dep); // escribo la persona en el fichero
        }
        dataOS.close(); // cerrar stream de salida
}//fin main()
```

# Modificar objetos (acceso secuencial)

# Modificaciones

1. Localizar el registro a modificar

```
if (dep.getDep() == depmodif)
```

2.Realizar la modificación

```
dep.setNombre(nombrenuevo);
dep.setLoc(locnueva);
departamentoexiste = 1;
```

3.Insertarlo en el fichero auxiliar

```
Departamento dep2 = new Departamento(dep.getNombre(),
dep.getDep(), dep.getLoc());
dataOS.writeObject(dep2);
```

```java
import java.io.Serializable;
class Departamento implements Serializable {
    private String nombre;
    private String loc;
    private int dep;
    public Departamento(String nombre, int dep, String loc) {
        this.nombre = nombre;
        this.dep = dep;
        this.loc = loc;
    }
    public Departamento() {
        this.nombre = null;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    public int getDep() {
        return dep;
    }
    public void setDep(int dep) {
        this.dep = dep;
    }
}
```

```
public static void main(String[] args) throws IOException {
        Departamento dep;
        File fichero = new File("Departamentos.dat");
        FileInputStream filein = new FileInputStream(fichero);
        ObjectInputStream dataIS = new ObjectInputStream(filein);
        // aqui estaran los nuevos datos
        File ficheroaux = new File("DepartamentosAux.dat");
        FileOutputStream fileout = new FileOutputStream(ficheroaux);
        ObjectOutputStream dataOS = new ObjectOutputStream(fileout);
        // recuperar argumentos de main
        String numerodep = args[0]; // num. departamento
        String nombrenuevo = args[1]; // nombre
        String locnueva = args[2]; // localidad
        int depmodif = Integer.parseInt(numerodep);
        int departamentoexiste = 0;
        try {
                while (true) { // lectura del fichero
                dep = (Departamento) dataIS.readObject();
                        if (dep.getDep() == depmodif) {
                                System.out.println("Datos ANTIGUOS DEL DEPARTAMENTO " + depmodif);
                                System.out.println("Nombre: " + dep.getNombre() + ", Localidad : " + dep.getLoc());
                                Dep.setNombre(nombrenuevo);
                                dep.setLoc(locnueva);
                                departamentoexiste = 1;
                        }
                Departamento dep2 = new Departamento(dep.getNombre(),
                        dep.getDep(), dep.getLoc());
                dataOS.writeObject(dep2); // inserto en fichero auxiliar
                }
        } catch (Exception e) {
// Se produce EOFException al finalizar la lectura
```

Si ejecutamos desde la línea de comandos e introducimos parámetros

Lectura y modificación

El objeto actualizado se escribe en el nuevo fichero

```
if (departamentoexiste > 0) {
        CrearNuevoDep();
        ListadoNuevo();
}
else
{
        System.out.println("====================================");
        System.out.println("DEPARTAMENTO A MODIFICAR NO EXISTE");
        System.out.println("====================================");
}}
```

```
public static void CrearNuevoDep() throws IOException {
    Departamento dep;
    // Leo auxiliar e inserto en Departamentos
    File fichero = new File("DepartamentosAux.dat");
    FileInputStream filein = new FileInputStream(fichero);
    ObjectInputStream dataIS = new ObjectInputStream(filein);
    // aqui estaran los nuevos datos
    File ficheroaux = new File("Departamentos.dat");
    FileOutputStream fileout = new FileOutputStream(ficheroaux);
    ObjectOutputStream dataOS = new ObjectOutputStream(fileout);
    try {
        while (true) { // lectura del fichero
            dep = (Departamento) dataIS.readObject();
            // Leo fichero auxiliar
            Departamento dep2 = new Departamento(dep.getNombre(), dep.getDep(),
            dep.getLoc());
            // inserto en nuevo fichero de Departamentos
            dataOS.writeObject(dep2);
        }
    }
    catch (Exception e) {
    // Se produce EOFException al finalizar la lectura
    dataIS.close(); // cerrar stream de entrada
    dataOS.close(); // cerrar stream de SALIDA
    }// fin Crear Nuevo Dep
```