

# 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
makedirs	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

---

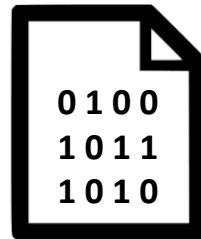
Sirven para enviar información desde un programa de Java a un sitio remoto (o local).

¿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).

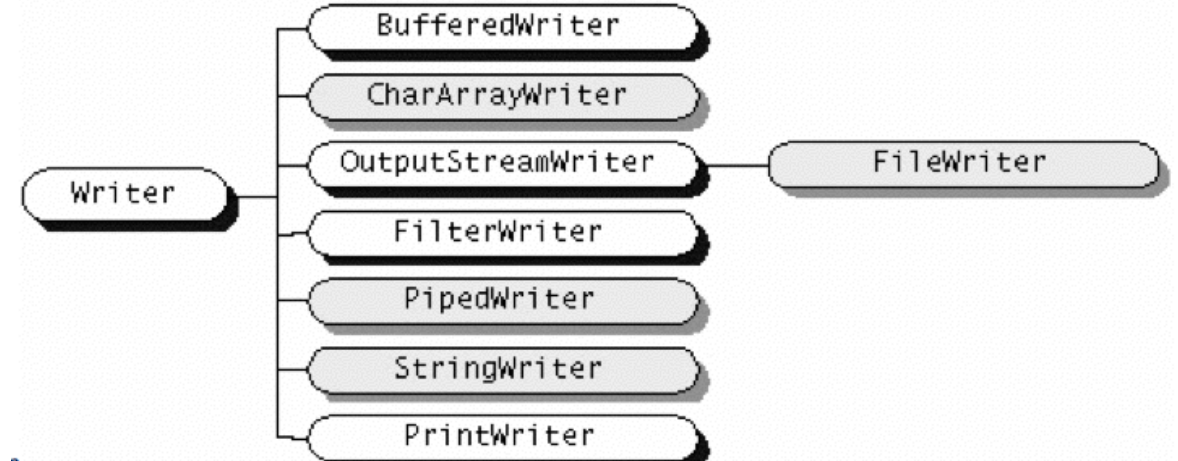
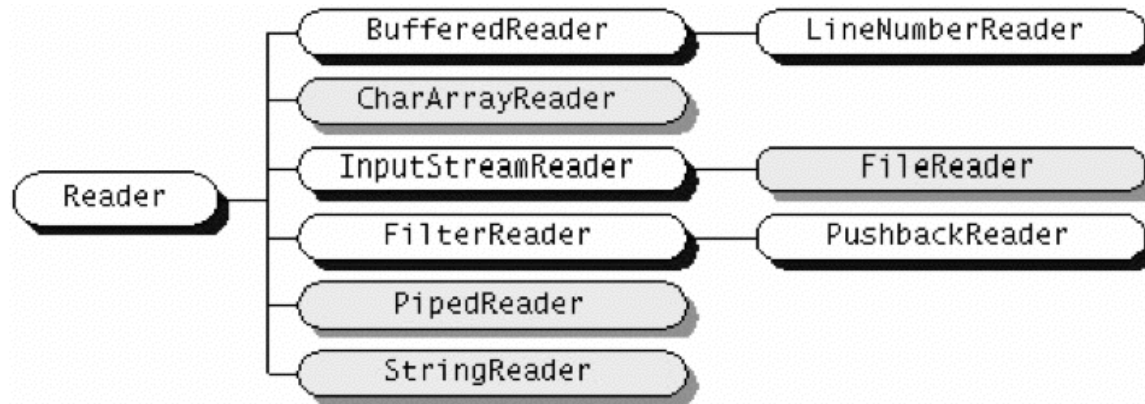
Los programas pueden leer datos de los ficheros del sistema, así como escribir en ellos.

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

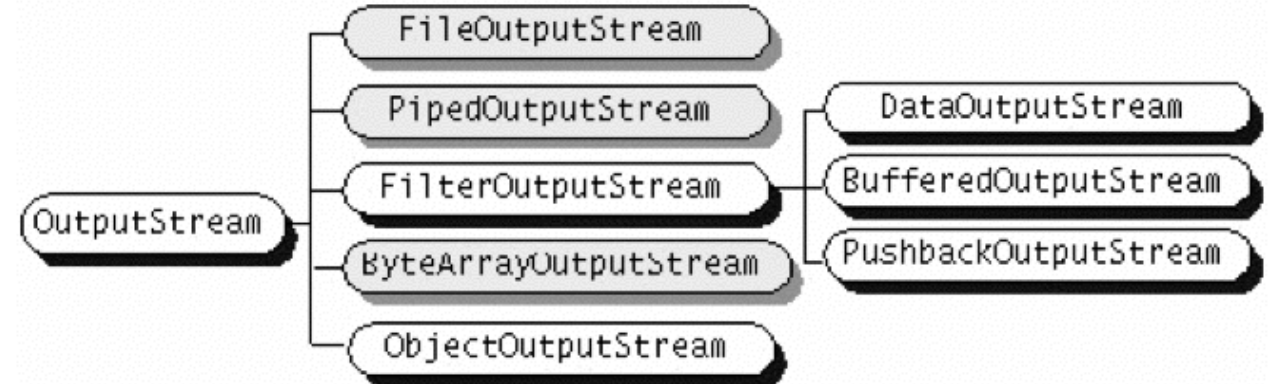
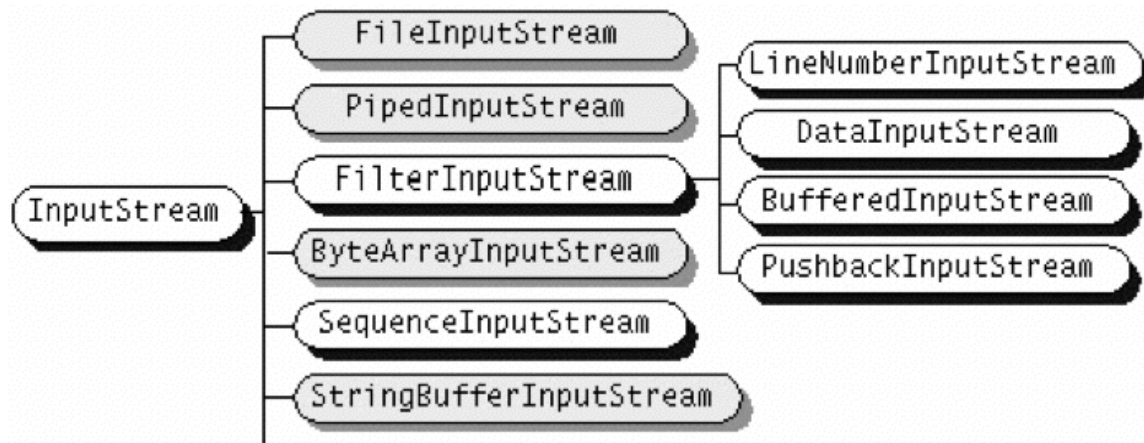
# Reader/Writer

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/OutputStreamWriter

---

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 comienza 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));
```

```
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 línea
                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 entrada hacia el fichero
FileInputStream filein = new FileInputStream(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)
```

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

```
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 guardar **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 ObjectInputStream.

- IOException, ClassNotFoundException

**void writeObject (java.io.ObjectOutputStream 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 bytes)
- Nombre: String (tamaño: 20 bytes → 1 char = 1 byte)
- Localidad: String (tamaño: 20 bytes → 1 char = 1 byte)

➤ **Departamentos.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;  
    // 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 el departamento en el fichero  
    }  
    dataOS.close(); // cerrar stream de salida  
} //fin main()
```

```
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
}
```

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);
```

```
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);
    // aquí estarán 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
```



# FICHEROS DE ACCESO ALEATORIO

---

UD1: MANEJO DE FICHEROS

# Operaciones sobre ficheros aleatorios

---

Para acceder a un registro hay que localizar su posición.

Se utilizan **direcciones relativas** en vez de direcciones absolutas (número de pista y sector en el disco).

Por lo tanto el programa será independiente de la dirección absoluta del fichero.

# Operaciones sobre ficheros aleatorios

---

## **Función de conversión**

**Clave:** el campo que identifica de forma unívoca a un registro

Para posicionarnos en un registro es necesario aplicar una función de conversión y normalmente tiene que ver con la clave del registro.

# Operaciones sobre ficheros aleatorios

---

## Ejemplo

Un fichero de empleados con tres campos: identificador, apellido y salario

1 Errasti 20000  
2 Gorosabel 40000  
3 Ugarte 35000

Utilizamos el identificador como clave

Para localizar el empleado con identificador X tenemos que acceder a la posición (X-1)

# Operaciones sobre ficheros aleatorios

---

## **Problemas**

Al aplicar la función al campo clave puede ocurrir que accedamos a un registro que está ocupado por otro registro.

En este caso deberíamos buscar una nueva posición libre en el fichero o utilizar la zona de excedentes .

# Operaciones sobre ficheros aleatorios

---

Siempre necesitaremos saber la clave para aplicar la función de conversión a la clave y obtener el registro en el que queremos realizar las operaciones

- **Altas:** si la posición está ocupada, zona de excedentes
- **Bajas:** se utiliza un campo del registro a modo de *switch* que tenga el valor 1 cuando el registro exista y 0 para darlo de baja. Físicamente el registro seguirá existiendo
- **Modificaciones:** localizar el registro a modificar, realizar la modificación y reescribir el registro en esa posición

# Operaciones sobre ficheros aleatorios

---

## **RandomAccessFile**

Incluido el path



```
RandomAccessFile(String nombreFichero, String modoAcceso)
```

```
RandomAccessFile(File objetoFile, String modoAcceso)
```

El argumento `modoAcceso` puede ser:

- `r`: lectura. El fichero debe existir (lanza la excepción `IOException`)
- `rw`: modo lectura y escritura. Si el fichero no existe, se crea

La clase **RandomAccessFile** maneja un **puntero** que indica la posición actual en el fichero.

- puntero = 0 cuando se crea el fichero
- puntero = número de bytes leídos o escritos

# Operaciones sobre ficheros aleatorios

---

## **RandomAccessFile**

Una vez abierto el fichero utilizando `RandomAccessFile`, se pueden usar los métodos `readXXX` y `writeXXX` de las clases `DataInputStream` y `DataOutputStream`.

La clase `RandomAccessFile` maneja un puntero que indica la posición actual en el fichero.

- Cuando el fichero se crea, el puntero al fichero se coloca en 0 (apuntando al principio).
- Las llamadas a los métodos `read()` y `write()` ajustan el puntero según los bytes que se lean o escriban



# Operaciones sobre ficheros aleatorios

---

## Métodos de `RandomAccessFile` más relevantes

- **`long getFilePointer()`**: devuelve la posición actual del puntero del fichero
- **`void seek(long position)`**: coloca el puntero del fichero en una posición determinada desde el comienzo del mismo
- **`long length()`**: devuelve el tamaño del fichero en bytes. La posición *length* marca el final del fichero
- **`int skipBytes(int desplazamiento)`**: desplaza el puntero desde la posición actual el número de bytes indicados en el desplazamiento

# Operaciones sobre ficheros aleatorios

---

## **Actividad – RandomAccessFile**

Crea un programa que inserta datos de empleados en un fichero aleatorio (empleados.dat).

Datos a insertar: *apellido*, *departamento* y *salario*. Se obtienen de varios arrays que **se llenan en el código directamente**. Los datos se van introduciendo de forma secuencial (no es necesario utilizar seek()). El archivo se abre en modo lectura y escritura (rw).

Por cada empleado se insertará un identificador (mayor que 0) que coincidirá con el índice + 1 con el que se recorren los arrays. Por ejemplo, id=4, índice=3+1

El registro de cada empleado es de 36 bytes:

- Identificador, entero de 4 bytes
- Apellido, cadena de 10 caracteres, 20 bytes (2 por cada, al ser caracteres UNICODE)
- Departamento, entero de 4 bytes
- Salario, doble de 8 bytes

➤ **EscribirFichAleatorio**

# Operaciones sobre ficheros aleatorios

---

## **Actividad – RandomAccessFile**

Aprovechando el fichero creado en el ejercicio anterior, crea un programa que visualice todos los registros.

El posicionamiento para empezar a recorrer los registros empieza en 0, y para recuperar los siguientes registros hay que sumar 36 (tamaño del registro) a la variable utilizada para el posicionamiento.

➤ **LeerFichAleatorio**

# Operaciones sobre ficheros aleatorios

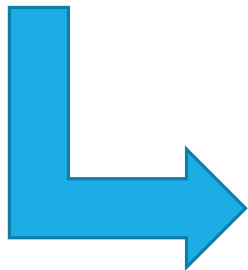
---

## **Actividad 11 – RandomAccessFile**

Crea un programa en Java que reciba un identificador de empleado desde la línea de comandos y visualice sus datos. Como alternativa, crea una variable que guarde un identificador concreto.

Si el empleado no existe debe visualizar el mensaje indicándolo. Primero debéis crear el fichero empelados.dat

### ➤ **Actividad11.java**



- 1.Leer el identificador que entra por consola
- 2.Calcular la posición
- 3.Posicionar el puntero
- 4.Leer el objeto

```
int posicion = (num - 1) * 36; //calcula la posición
if (posicion >= 0) {
    int dep, id;
    Double sal;
    char ape[] = new char[10], aux;
    try {
        file.seek(posicion); // ME POSICIONO
        id = file.readInt(); // obtengo identificador
        if (id > 0) { // si id es 0 es que no hay hueco
            for (int i = 0; i < ape.length; i++) {
                aux = file.readChar();
                ape[i] = aux;
            }
            String apellidoS = new String(ape); // obtengo apellido
            dep = file.readInt(); // obtengo dep
            sal = file.readDouble(); // obtengo salario

            // CÓDIGO PARA MOSTRAR LOS DATOS - SYSTEM.OUT.PRINTF...
        } else {
            System.out.println("***NO EXISTE-HUECO**");
        }
    } catch (java.io.EOFException e) { // SI EL POSICIONAMIENTO FALLA
        System.out.println("***EMPLEADO NO EXISTE***");
    }
}
```

# Operaciones sobre ficheros aleatorios

---

## Añadir registros al final

### 1. Conocer el tamaño del fichero

- `long length()` : devuelve el tamaño del fichero en bytes. La posición *length* marca el final del fichero

### 2. Posicionar el puntero al final para empezar a escribir

- `void seek(long position)` : coloca el puntero del fichero en una posición determinada desde el comienzo del mismo

```
long position = file.length()  
file.seek(position)
```

# Operaciones sobre ficheros aleatorios

---

## Añadir un registro

**1. Calculamos la posición:** debemos conocer la posición y el número de bytes del registro para aplicar la función de conversión.

- `(identificador-1) * 36bytes`

**2. Posicionamos el puntero:** `seek(posicion)`

**3. Escribimos** cada uno de los campos con sus métodos correspondientes.

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

# Operaciones sobre ficheros aleatorios

---

## Modificar un registro

### 1. Abrir el fichero como RW

### 2. Calculamos la posición y posesionamos el puntero:

- `(identificador-1)*36bytes`
- `seek(posicion)`

### 3. Realizar modificaciones

- Dependiendo del campo que queramos modificar deberemos calcular su posición
- Para modificar el **departamento**, debo posicionar el puntero en

`posición + id + apellido = posición + 4 + 20`



# Operaciones sobre ficheros aleatorios

---

## **Actividad 12**

Crea un programa Java que reciba desde la línea de comandos un identificador de empleado y un importe. Se debe sumar al salario del empleado el importe tecleado. El programa debe visualizar el apellido, el salario antiguo y el nuevo.

Si el identificador no existe se visualizará un mensaje indicándolo.

➤ **Actividad12.java**

# Ficheros XML

---

# Ficheros XML

---

- Los ficheros XML son ficheros de texto escritos en el lenguaje XML(extensible Markup Language) donde la información está organizada de forma secuencial y en orden jerárquico.
- Se usan etiquetas `<?xml version="1">` para delimitar las marcas del documento.
- Cada marca tiene un nombre y puede tener 0 o más atributos

```
<?xml version="1.0"?>
<listadealumnos>
  <alumno>
    <nombre>Juan</nombre>
    <edad>19</edad>
  </alumno>
  <alumno>
    <nombre>Maria</nombre>
    <edad>20</edad>
  </alumno>
</listadealumnos>
```

# Ficheros XML

---

## ¿Para qué es útil un fichero XML?

Un programa informático puede estar escrito en Java, Visual Basic y cualquier otro lenguaje. En esencia, todos los programas procesan información, entendiéndose por información “dato + significado”.

Para el caso que estamos viendo, el dato en el ejemplo sería “**Juan**” y el significado es un “**nombre de alumno**”. Por lo tanto un documento escrito en XML tendría la información que necesitan los programas para procesar.

XML se plantea como un **lenguaje estándar para el intercambio de información entre diferentes programas** de una manera segura, fiable y libre

# Ficheros XML

---

- Roberto nació el 15.10.2012 en la ciudad de Madrid con un peso de 3.1 kg y una estatura de 45 cm.
- Ana nació el 11.09.1976 en la ciudad de Sevilla con un peso de 3 Kg y una estatura de 40 cm

Analizando el texto, nos encontramos que hay datos como “Madrid” y su correspondiente significado, que es una “ciudad” y otros más en un formato humano, tan sólo entendible por personas, no por los programas.

Por tanto, podemos convertir el texto tanto en una base de datos “tradicional” como en un archivo o documento XML para que los programas lo puedan procesar

# Ficheros XML

---

```
<Datos-Nacimiento>
  <Persona>
    <Nombre>Roberto</Nombre>
    <Fecha>15.10.2012</Fecha>
    <Ciudad>Madrid</Ciudad>
    <Peso>3.1Kg</Peso>
    <Estatura>45cm</Estatura>
  </Persona>
  <Persona>
    <Nombre>Ana</Nombre>
    <Fecha>11.09.2012</Fecha>
    <Ciudad>Sevilla</Ciudad>
    <Peso>3Kg</Peso>
    <Estatura>40cm</Estatura>
  </Persona>
</Datos-Nacimiento>
```

# Procesadores

---

Para leer los ficheros XML y acceder a su contenido y estructura necesitamos un **procesador** XML o un ***parser***

El procesador lee los documentos y proporciona acceso a su contenido y estructura

- **DOM**: Modelo de Objetos de Documento
- **SAX**: API Simple para XML



# DOM

---

- Define un estándar para acceder y manipular los documentos.
- DOM XML presenta un documento XML como una estructura de árbol.
- Entender el DOM es una necesidad para cualquier persona que trabaje con HTML o XML.

# DOM

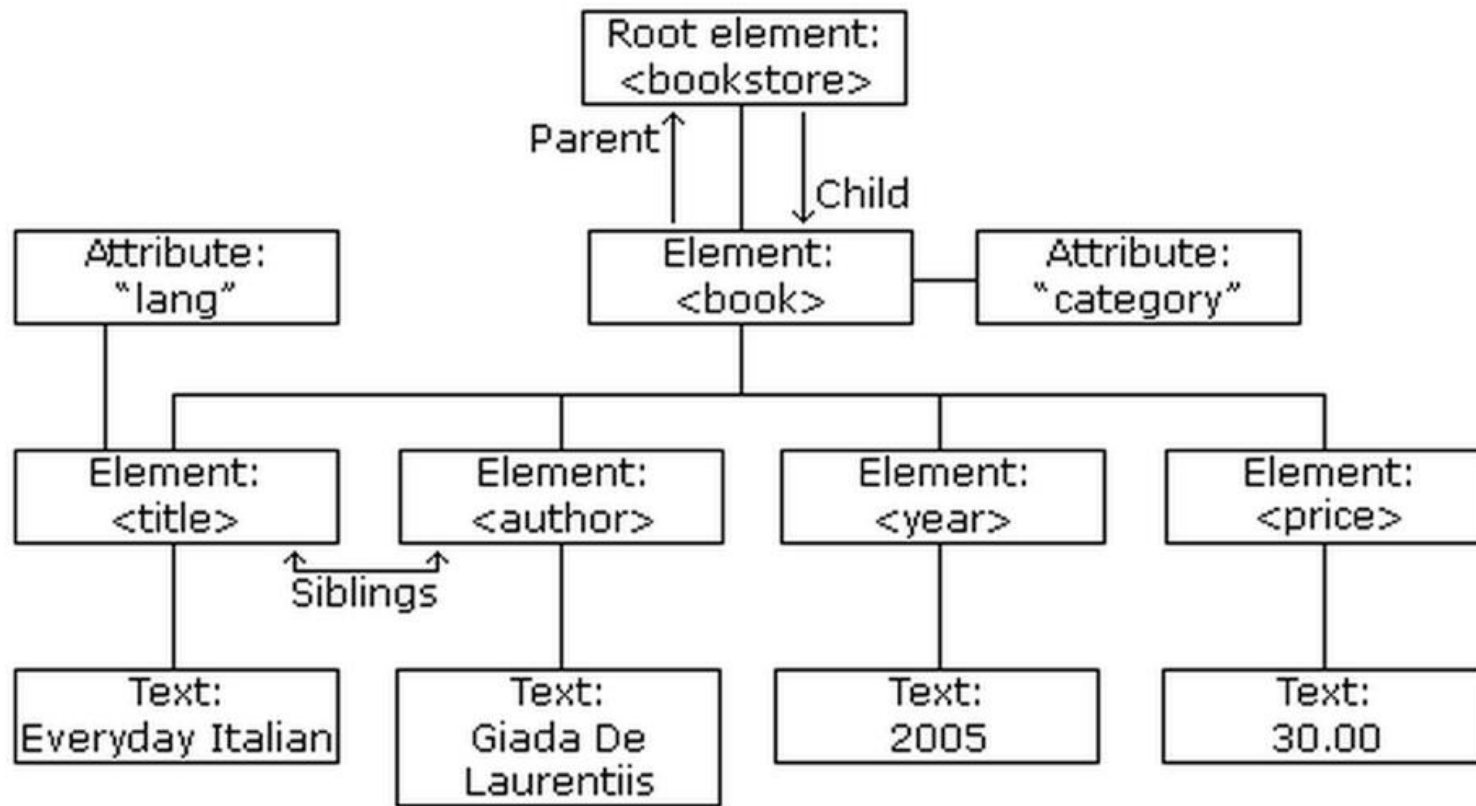
---

DOM: El API "Document Object Model" (**DOM**), definido por el grupo de trabajo **DOM** de la W3C, es un conjunto de interfaces para construir una representación de objeto, en forma de árbol, de un documento XML analizado. Una vez que hemos construido el **DOM**, podemos manipularlo con métodos **DOM** como **insert** y **remove**, igual que manipularíamos cualquier otra estructura de datos en forma de árbol.

Almacena toda la estructura del documento en memoria en forma de árbol con nodos padre, nodos hijo y nodos finales (los que no tienen descendiente).

# DOM

## Estructura de árbol



```
<bookstore>
  <book>
    <title>Every day Italian</title>
    <author>Giada de Lauretis</author>
    <year>2005</year>
    <time>30.00</time>
  </book>
</bookstore>
```

# Acceso a ficheros XML con DOM

---

Para poder trabajar con DOM en Java necesitamos utilizar las clases e interfaces de los paquetes:

- `org.w3c.dom`
- `javax.xml.parser`
- `javax.xml.transform`

Clases fundamentales:

- `DocumentBuilderFactory`
- `DocumentBuilder`

# Creación de un fichero XML

---

Vamos a **crear** un fichero XML a partir del fichero aleatorio de empleados

El fichero empleado contiene los siguientes datos:

- Un identificador (entero)
- Apellido (cadena de 20 bytes)
- Número de departamento (entero)
- Salario (Double)



```
<empleado>  
  <id>3</id>  
  <apellido>Gorosabel</apellido>  
  <dep>12</dep>  
  <salario>30000</salariio>  
</empleado>
```

# Pasos para la creación de un fichero XML

---

## **Paso 1:** Importar los paquetes necesarios

```
import org.w3c.dom.*;  
import javax.xml.parsers.*;  
import javax.xml.transform.*;  
import javax.xml.transform.dom.*;  
import javax.xml.transform.stream.*;  
import java.io.*;
```

# Pasos para la creación de un fichero XML

---

## Paso 2: Construir el parser

Se crea una instancia **DocumentBuilderFactory** para construir el parser, que luego es usado para crear el objeto `DocumentBuilder` que construye el *parser*

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
  
try{  
    DocumentBuilder builder = factory.newDocumentBuilder();  
    ...  
}
```

# Pasos para la creación de un fichero XML

---

## Paso 3: Crear un documento

Se crea un documento vacío con el nombre *document* y el nodo raíz de nombre *Empleados*.  
Asignamos la versión del XML (Versión: 1.0).

```
DOMImplementation implementation = builder.getDOMImplementation();  
Document document = implementation.createDocument(null, "Empleados", null);  
document.setXmlVersion("1.0");
```



# Pasos para la creación de un fichero XML

---

**Paso 4:** Recorrer el fichero empleados y crear un nodo empleado por cada registro

El nodo empleado tendrá 4 hijos:

- id
- apellido
- dep
- salario

Cada nodo hijo tendrá su valor, por ejemplo: 1, Rodriguez, 2, 35000

# Pasos para la creación de un fichero XML

---

//Crea y añade el nodo empleado al documento

```
Element raiz = document.createElement("empleado"); //nodo empleado
```

```
document.getDocumentElement().appendChild(raiz); //lo añade a la raíz del documento
```

//se añaden los hijos al nodo raiz

```
CrearElemento("id",1, raiz, document);
```

```
CrearElemento("apellido","Rodriguez", raiz, document);
```

```
CrearElemento("dep",2, raiz, document);
```

```
CrearElemento("salario",35000, raiz, document);
```

# Pasos para la creación de un fichero XML

---

```
static void CrearElemento(String datoEmple, String valor,
                          Element raiz, Document document)
{
    Element elem = document.createElement(datoEmple); //creamos hijo
    Text text = document.createTextNode(valor); //damos valor
    raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
    elem.appendChild(text); //pegamos el valor
}
```

Por cada empleado generaríamos:

```
<empleado>
  <id>1</id>
  <apellido>Rodriguez</apellido>
  <dep>12</dep>
  <salario>35000</salario>
</empleado>
```

# Pasos para la creación de un fichero XML

---

Se crea la fuente XML a partir del documento

```
Source source = new DOMSource(document);
```

Se crea el resultado en el fichero Empleados.xml

```
Result result = new StreamResult(new java.io.File("Empleados.xml"));
```

Se obtiene un TransformerFactory

```
Transformer transformer = TransformerFactory.newInstance().newTransformer();
```

Se realiza la transformación de documento a fichero

```
transformer.transform(source, result);
```

Para mostrar el documento por pantalla podemos especificar como resultado el canal `system.out`

```
Result console= new StreamResult(System.out);
```

```
transformer.transform(source, console)
```

# Pasos para la creación de un fichero XML

---

El código completo está escrito en el siguiente documento:

`CrearEmpleadoXML_DOM.java`

# XStream

---

## Serialización de objetos a XML

- Fácil de usar.
- No requiere modificaciones en los objetos Java que se van a serializar.
- Integración sencilla con otros APIs para XML

Método de escritura:

```
xstream.toXML(Object o, new FileOutputStream("Fichero.xml")) ;
```

Método de lectura:

```
xstream.fromXML(new FileInputStream("Fichero.xml")) ;
```

# XStream

---

Para poder utilizar Xstream debemos descargar los JAR desde el siguiente sitio web:

<http://x-stream.github.io/download.html> → xstream-distribution-1.4.X-bin.zip

Necesitaremos:

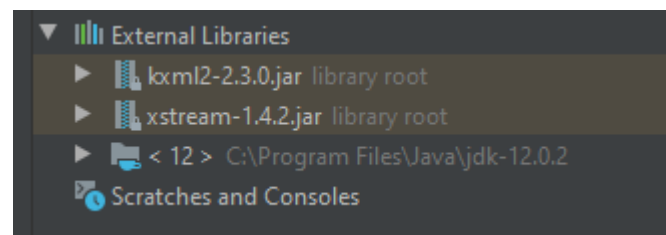
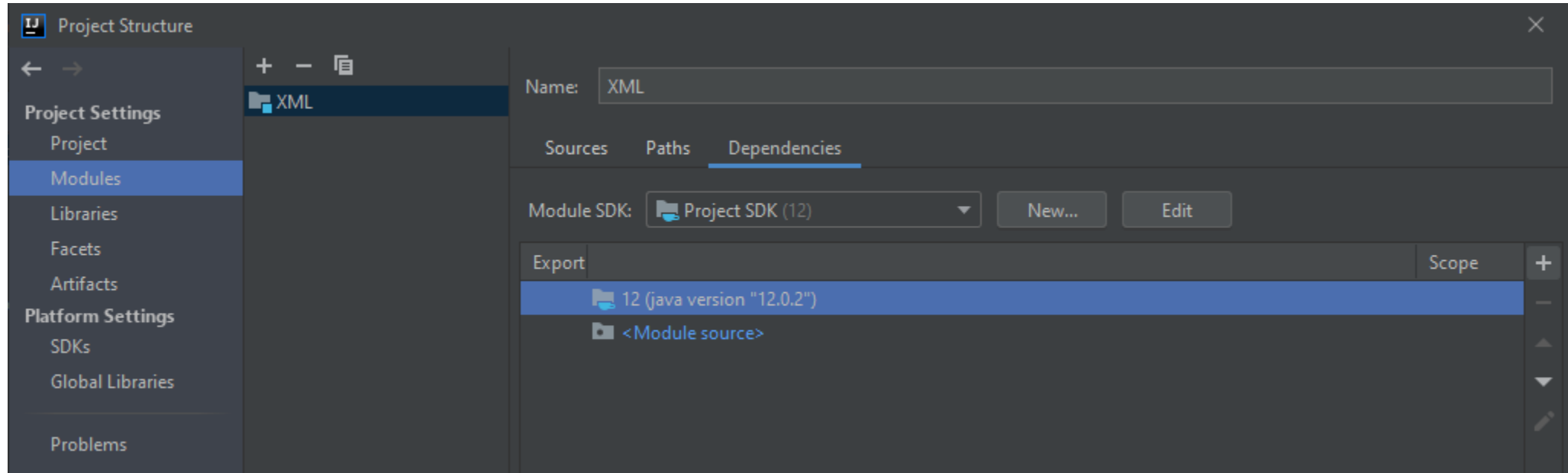
- **xstream-1.4.X.jar (dentro de lib)**
- **kxml2-2.3.0.jar (lib/xstream)**

Debemos añadirlos al proyecto o los definimos en el CLASSPATH

File -> Project Structure

Project Settings -> Modules -> Dependencies -> + -> Jar or directories

IntelliJ





## Añadir JAR a Eclipse

Para añadir un jar externo a nuestro proyecto con Eclipse seguimos los siguientes pasos:

- Con el botón derecho del ratón sacamos el menú sobre el proyecto y elegimos "propiedades"
- En la ventana que se abre, en el arbolito de la izquierda elegimos "java build path".
- En el lado derecho de la ventana, elegimos la pestaña "Libraries"
- Le damos al botón "add external jar"
- Buscamos el jar que queremos añadir y lo seleccionamos.
- Aceptar, aceptar, aceptar y listo, ese jar ya forma parte de nuestro proyecto Eclipse y podemos usarlo en nuestro programa.

# Xstream – Ejemplo

---

Para el ejemplo, utilizaremos el archivo **FichPersona.dat**, creado en ejercicios anteriores utilizando la clase **Persona**.

- Creamos una lista de objetos Persona
- Convertimos la lista de objetos en un fichero de datos XML
- Necesitaremos la clase Persona y la clase ListaPersona en la que se define una lista de objetos Persona que pasaremos al fichero XML

El proceso consiste en recorrer el fichero FichPersonas.dat para crear una lista de personas que después se insertarán en el fichero Personas.xml.

# XStream

---

## **Archivos (carpeta Xstream):**

- Persona.java
- ListaPersonas.java
- EscribirPersonas.java

# XStream

---

## Fichero XML generado:

```
<ListaPersonasMunicipio>
  <DatosPersona>
    <nombre>Ana</nombre>
    <edad>14</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Pedro</nombre>
    <edad>15</edad>
  </DatosPersona>
  ...
</ListaPersonasMunicipio>
```

```
import java.util.ArrayList;
import java.util.List;

public class ListaPersonas {

    private List<Persona> lista = new ArrayList<Persona> ();

    public ListaPersonas(){}

    public void add(Persona per) {
        lista.add(per);
    }

    public List<Persona> getListaPersonas() {
        return lista;
    }
}
```

```

public class EscribirPersonas {
    public static void main(String[] args) throws IOException , ClassNotFoundException {
        File fichero = new File("FichPersona.dat");
        FileInputStream filein = new FileInputStream(fichero); //crea el flujo de entrada
        //conecta el flujo de bytes al flujo de datos
        ObjectInputStream dataIS = new ObjectInputStream(filein);
        System.out.println("Comienza el proceso de creación del fichero a XML ...");
        //Creamos un objeto Lista de Personas
        ListaPersonas listaper = new ListaPersonas();
        try {
            while (true) { //lectura del fichero
                Persona persona= (Persona) dataIS.readObject(); //leer una Persona
                listaper.add(persona); //añadir persona a la lista
            }
        } catch (EOFException eo) {}
        dataIS.close(); //cerrar stream de entrada
        try {
            XStream xstream = new XStream();
            //cambiar de nombre a las etiquetas XML
            xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
            xstream.alias("DatosPersona", Persona.class);
            //quitar etiqueta lista (atributo de la clase ListaPersonas)
            xstream.addImplicitCollection(ListaPersonas.class, "lista");
            //Insertar los objetos en el XML
            xstream.toXML(listaper, new FileOutputStream("Personas.xml"));
            System.out.println("Creado fichero XML....");
        } catch (Exception e)
        {e.printStackTrace();}
    } // fin main
} //fin EscribirPersonas

```

EscribirPersonas.java

```

public class EscribirPersonas {
    public static void main(String[] args) throws IOException , ClassNotFoundException {
        File fichero = new File("FichPersona.dat");
        FileInputStream filein = new FileInputStream(fichero); //crea el flujo de entrada
        //conecta el flujo de bytes al flujo de datos
        ObjectInputStream dataIS = new ObjectInputStream(filein);
        System.out.println("Comienza el proceso de creación del fichero a XML ...");
        //Creamos un objeto Lista de Personas
        ListaPersonas listaper = new ListaPersonas();
        try {
            while (true) { //lectura del fichero
                Persona persona= (Persona) dataIS.readObject(); //leer una Persona
                listaper.add(persona); //añadir persona a la lista
            }
        } catch (EOFException eo) {}
        dataIS.close(); //cerrar stream de entrada

        try {
            XStream xstream = new XStream();
            //cambiar de nombre a las etiquetas XML
            xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
            xstream.alias("DatosPersona", Persona.class);
            //quitar etiqueta lista (atributo de la clase ListaPersonas)
            xstream.addImplicitCollection(ListaPersonas.class, "lista");
            //Insrtar los objetos en el XML
            xstream.toXML(listaper, new FileOutputStream("Personas.xml"));
            System.out.println("Creado fichero XML....");
        } catch (Exception e) {
            {e.printStackTrace();}
        } // fin main
    } //fin EscribirPersonas
}

```

Gestión fichero:  
ficheros binarios –  
objetos serializables

XStream

# XStream

---

```
import com.thoughtworks.xstream.XStream;
```

1. Creamos una instancia de la clase XStream

```
XStream xstream = new XStream();
```

2. En general las etiquetas XML se corresponden con el nombre de los atributos de la clase  
En este caso, se utiliza el método “alias” para poder cambiar las etiquetas XML.

→ **alias(String alias, Class clase)**

```
xstream.alias("ListaPersonasMunicipio", ListaPersonas.class)
```

3. También se ha dado un alias a la clase persona

```
xstream.alias("DatosPersona", Persona.class);
```



# XStream

---

4. El método `aliasField(String alias, Class clase, String nombrecampo)` permite crear un alias para un nombre de campo. Por ejemplo, si queremos cambiar el nombre de los campos `nombre` y `edad` (de la clase `Persona`), crearíamos los siguientes alias:

```
xstream.aliasField("Nombre alumno", Persona.class, "nombre");  
xstream.aliasField("Edad alumno", Persona.class, "edad")
```



En el fichero XML se crearían con las etiquetas:

<Nombre alumno> en vez de <nombre>

<Edad alumno> en vez de <edad>

# XStream

---

5. Para que no aparezca el atributo *lista* de la clase *ListaPersonas* en el XML generado, se ha utilizado el método **addImplicitCollection(Class clase, String nombredecampo)**

```
xstream.addImplicitCollection(ListaPersonas.class, "lista");
```

6. Para generar el fichero Personas.xml a partir de la lista de objetos, se utiliza el método **toXML(Objeto objeto, OutputStream out)**

```
xstream.toXML(listaper, new FileOutputStream("Personas.xml"));
```

# XStream

---

## Actividad

¿Cómo se realizaría la lectura del fichero XML generado en el ejercicio anterior?

➤ **LeerPersonas.java**

Es necesario que exista el archivo XML generado en el ejercicio anterior

# XStream

---

Para leer archivos XML se deben utilizar los métodos que se han utilizado para la escritura:

- **alias()**
- **addImplicitCollection()**

Para obtener el objeto con la lista de personas (*deserializar* el objeto a partir del fichero):

- **fromXML (InputStream input) → devuelve Object:**

```
ListaPersonas listadoTodas = (ListaPersonas) xstream.fromXML("Personas.xml");
```

```
import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.thoughtworks.xstream.XStream;
```

```
public class LeerPersonas {
    public static void main(String[] args) throws IOException {
```

```
        XStream xstream = new XStream();

        xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
        xstream.alias("DatosPersona", Persona.class);
        xstream.addImplicitCollection(ListaPersonas.class, "lista");
```

```
        FileInputStream fichero = new FileInputStream("Personas.xml");
```

```
        ListaPersonas listadoTodas = (ListaPersonas) xstream.fromXML(fichero);
```

```
        System.out.println("Número de personas: " + listadoTodas.getListaPersonas().size());
```

```
        List<Persona> listaPersonas = new ArrayList<Persona>();
        listaPersonas = listadoTodas.getListaPersonas();
```

```
        Iterator iterador = listaPersonas.listIterator();
```

```
        while(iterador.hasNext()){
            Persona p = (Persona) iterador.next();
            System.out.printf("Nombre: %s, edad: %d %n", p.getNombre(), p.getEdad());
        }
```

```
        System.out.println("Fin de listado ....");
```

```
    } //fin main
```

```
} //fin LeerPersonas
```

# XStream

---

## Actividad

Convertir el fichero *Empleados.dat* a XML

Clases que se necesitarán

- Empleado.java
- EscribirObjetosEmpleado.java
- ListaEmpleados.java
- EscribirEmpleados.java (crea el XML)

# Manejo de Ficheros Secuenciales de Objetos en Java

# Manejo de Ficheros Secuenciales de Objetos en Java

---

1. Importar una serie de clases del paquete `java.io`

```
import java.io.* ;
```

2. La clase tipo de los objetos componentes del fichero debe implementar el interfaz *Serializable*, para lo cual será necesario indicarlo en la cabecera de definición de la clase:

```
public class NombreDeLaClase implements Serializable
```

3. El método *main*, y cualquier otro método que maneje algún fichero en un programa, lanza la excepción en caso de producirse algún error en el manejo de dichos ficheros (especificarlo en la cabecera del método)

```
public static void main (String[] args) throws Exception
```



## Ejemplo 1

Un distribuidor de coches quiere informatizar una parte de la gestión de su negocio para lo cual desea que se cree inicialmente un **fichero** que contenga los datos de todos los **automóviles** que el concesionario tiene para su venta y asignarle a dicho fichero el nombre de “Deposito.dat”.

La nave donde tienen almacenados los coches para la venta tiene capacidad para contener un máximo de 100 coches.

Un `Automóvil` contiene los atributos de un coche:

- `marca`: cadena de caracteres.
- `potencia`: Entero positivo.
- `precio`: Entero positivo.

Como métodos públicos se deberán definir:

- Los dos **constructores**: por defecto y recibiendo los datos de los atributos.
- Los **consultores** (*get*) y **modificadores** (*set*) para cada atributo.
- Un **método** que muestre el *valor* de los atributos.

```
import java.io.*;
```

```
public class Automovil implements Serializable {
```

```
    private String marca;  
    private int potencia;  
    private int precio;
```

```
    public Automovil() {  
    }
```

```
    public Automovil(String marca, int potencia, int precio) {  
        this.marca = marca;  
        this.potencia = potencia;  
        this.precio = precio;  
    }
```

```
    public String getMarca() {  
        return marca;  
    }
```

```
    public int getPotencia() {  
        return potencia;  
    }
```

```
    public int getprecio() {  
        return precio;  
    }
```

```
    public void setMarca(String marca) {  
        this.marca = marca;  
    }
```

```
    public void setPotencia(int potencia) {  
        this.potencia = potencia;  
    }
```

```
    public void setPrecio(int precio) {  
        this.precio = precio;  
    }
```

```
    public void mostrar() {  
        System.out.println();  
        System.out.println(" Marca: " + marca);  
        System.out.println(" Potencia: " + potencia);  
        System.out.println(" Precio:" + precio);  
    }  
}
```

# Creación de un archivo secuencial en Java

---

1. Crear un objeto de la clase `FileOutputStream` asociado al nombre del fichero físico:

```
FileOutputStream NombreObjetoFicheroFisico =  
    new FileOutputStream("NombreDelFichero.DAT");
```



```
FileOutputStream foAutomoviles = new FileOutputStream("DEPOSITO.DAT");
```

# Creación de un archivo secuencial en Java

---

2. Crear un objeto de la clase **ObjectOutputStream** asociado al objeto anterior, es decir al fichero físico:

```
ObjectOutputStream NombreObjetoFicheroLogico =  
    new ObjectOutputStream(NombreObjetoFicheroFisico);
```



```
ObjectOutputStream ooAutomoviles = new ObjectOutputStream(foAutomoviles);
```

# Creación de un archivo secuencial en Java

---

3. Para escribir en el fichero cada uno de sus componentes que son objetos de una clase concreta deberemos utilizar:

```
NombreObjetoFicheroLogico.writeObject(VariableObjeto);
```



```
Automovil coche= new Automovil(marca, potencia, precio);  
ooAutomoviles.writeObject(coche);
```



```
ooAutomoviles.writeObject(new Automovil(marca, potencia, precio));
```

# Creación de un archivo secuencial en Java

---

4. Opcionalmente, al finalizar la escritura de los diferentes componentes del fichero , se puede grabar un objeto con valor *null* como marca de fin de fichero, para que en los procesos de lectura de dicho fichero quede establecido el momento de parada en el proceso de inspección o recorrido del fichero:

```
NombreObjetoFicheroLogico.writeObject(null);
```



```
ooAutomoviles.writeObject(null);
```

# Creación de un archivo secuencial en Java

---

5. Al finalizar el proceso de creación se debe cerrar el flujo de salida:

```
NombreObjetoFicheroLogico.close() ;
```



```
ooAutomoviles.close() ;
```

# Esquema de cómo crear un fichero en Java

---

{

Crear un objeto de la clase `FileOutputStream` asociado al nombre del fichero fisico.

```
FileOutputStream NombreObjetoFicheroFisico =  
    new FileOutputStream("NombreFisicoDelFichero.DAT");
```

Crear un objeto de la clase `ObjectOutputStream` asociado al objeto que identifica al fichero fisico.

```
ObjectOutputStream NombreObjetoFicheroLogico =  
    new ObjectOutputStream(NombreObjetoFicheroFisico);
```

Repetir {

Solicitar los datos

Introducir los datos

Construir el Objeto con los datos

Escribir el objeto en el fichero

```
NombreObjetoFicheroLogico.writeObject(VariableObjeto);
```

Preguntar si hay más datos y responder

} Mientras ( Hay mas datos )

Escribir en el fichero la marca de Fin de Fichero

```
NombreObjetoFicheroLogico.writeObject(null);
```

Cerrar el stream

```
NombreObjetoFicheroLogico.close();
```

}

esquema\_crear\_fichero.pdf



# Creación de un fichero secuencial

---

**Un distribuidor de coches quiere informatizar una parte de la gestión de su negocio para lo cual desea que se cree inicialmente un fichero que contenga los datos de todos los automóviles que el concesionario tiene para su venta y asignarle a dicho fichero el nombre de “Deposito.DAT”.**



- Documento pdf
- ejemplo1\_fichero\_secuencial.java

# Inspección de un fichero secuencial en Java

---

1. Crear un objeto de la clase **FileInputStream** asociado al nombre del fichero físico:

```
FileInputStream NombreObjetoFicheroFisico = new  
    FileInputStream("NombreFisicoDelFichero.DAT");
```

2. Crear un objeto de la clase **ObjectInputStream** asociado al objeto anterior, es decir al fichero fisico:

```
ObjectInputStream NombreObjetoFicheroLogico = new  
    ObjectInputStream(NombreObjetoFicheroFisico);
```

# Inspección de un fichero secuencial en Java

---

3. Para leer del fichero cada uno de sus componentes, que son objetos de la misma clase de cuando se ha creado dicho fichero, deberemos utilizar:

```
ClaseDelObjeto VariableObjeto =  
    (ClaseDelObjeto) NombreObjetoFicheroLogico.readObject() ;
```

Será necesario realizar un casting para manejar el objeto de la clase correspondiente.

Si en la creación del fichero se grabó un objeto null como marca de fin de fichero, la lectura de dicho objeto marcará la parada del proceso de inspección o recorrido del fichero. En caso contrario, se finalizará con una excepción al intentar realizar una lectura en el final del fichero:

```
while (VariableObjeto != null)
```

# Inspección de un fichero secuencial en Java


---

4. Al finalizar el proceso de inspección se debe cerrar el fichero:


```
NombreObjetoFicheroLogico.close() ;
```

# Esquema básico de inspección de un fichero

---




Crear un objeto de la clase `FileInputStream` asociado al nombre del fichero físico.



Crear un objeto de la clase `ObjectInputStream` asociado al objeto anterior, es decir al fichero físico.




Leer el primer objeto del fichero



Mientras no es el fin del fichero hacer  
    Tratamiento del objeto en curso

...

    Leer el siguiente objeto del fichero  
fin\_Mientras



Cerrar el flujo de entrada

# Esquema básico de inspección de un fichero en Java

→ Crear un objeto de la clase `FileInputStream` asociado al nombre del fichero físico.

```
FileInputStream NombreObjetoFicheroFisico =  
    new FileInputStream("NombreFisicoDelFichero.DAT");
```

→ Crear un objeto de la clase `ObjectInputStream` asociado al objeto anterior, es decir al fichero físico.

```
ObjectInputStream NombreObjetoFicheroLogico =  
    new ObjectInputStream(NombreObjetoFicheroFisico);
```

→ Leer el primer objeto del fichero

```
ClaseDelObjeto VariableObjeto =  
    (ClaseDelObjeto) NombreObjetoFicheroLogico.readObject();
```

→ Mientras no es el fin del fichero hacer

```
while (VariableObjeto != null)  
{  
    Tratamiento del objeto en curso  
    ...  
    Leer el siguiente objeto del fichero  
    VariableObjeto =  
        (ClaseDelObjeto) NombreObjetoFicheroLogico.readObject();  
}
```

fin\_Mientras

→ Cerrar el flujo de entrada

```
NombreObjetoFicheroLogico.close();
```

# Inspección de un fichero secuencial - ejemplos

---

## Ejemplo 2 : **ejemplo2\_inspeccion\_fsecuencial.java**

El distribuidor de coches quiere que a partir del fichero que se le ha creado en el programa anterior se le desarrolle un nuevo programa que le pueda dar la información de todos los coches que tiene en el deposito y cuyo precio sea inferior o igual a una cantidad introducida previamente por teclado.

## Ejemplo 3: **ejemplo3\_inspeccion\_fsecuencial.java**

El distribuidor de coches ahora quiere que a partir del fichero que se le ha creado en el primer programa se le desarrolle un nuevo programa que le pueda dar la información de todos los coches de una determinada marca cuya potencia sea superior a un determinado numero de caballos.

# Inspección de un fichero secuencial - ejemplos

---

## Ejemplo 4 : **ejemplo4\_inspeccion\_fsecuencial.java**

El distribuidor de coches ahora nos pide que a partir del fichero que se le ha creado en el primer programa se le diseñe un nuevo programa que le pueda dar la información de todos los coches con una determinada potencia que no superen el precio medio de los que hay en el deposito de esa misma potencia.



# Ejecutar desde la línea de comandos

---

El parámetro **args** es un array de Strings que debe aparecer obligatoriamente como argumento del método **main** en un programa Java

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

El array args del método main es el encargado de **recoger y almacenar los valores que se introducen por consola**

```
C:\> java ordenar
```

```
C:\> java ordenar 4 6 3 7 1
```

# Ejecutar desde la línea de comandos

---

```
public class LineaComandos {  
    public static void main(String[] args) {  
        for(int i = 0; i < args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```

args.length = contiene el **número de valores** enviados al programa