MANEJO DE FICHEROS

UT1

Manejo de Ficheros

Ficheros.

Existen varias definiciones de lo que es un fichero o archivo.

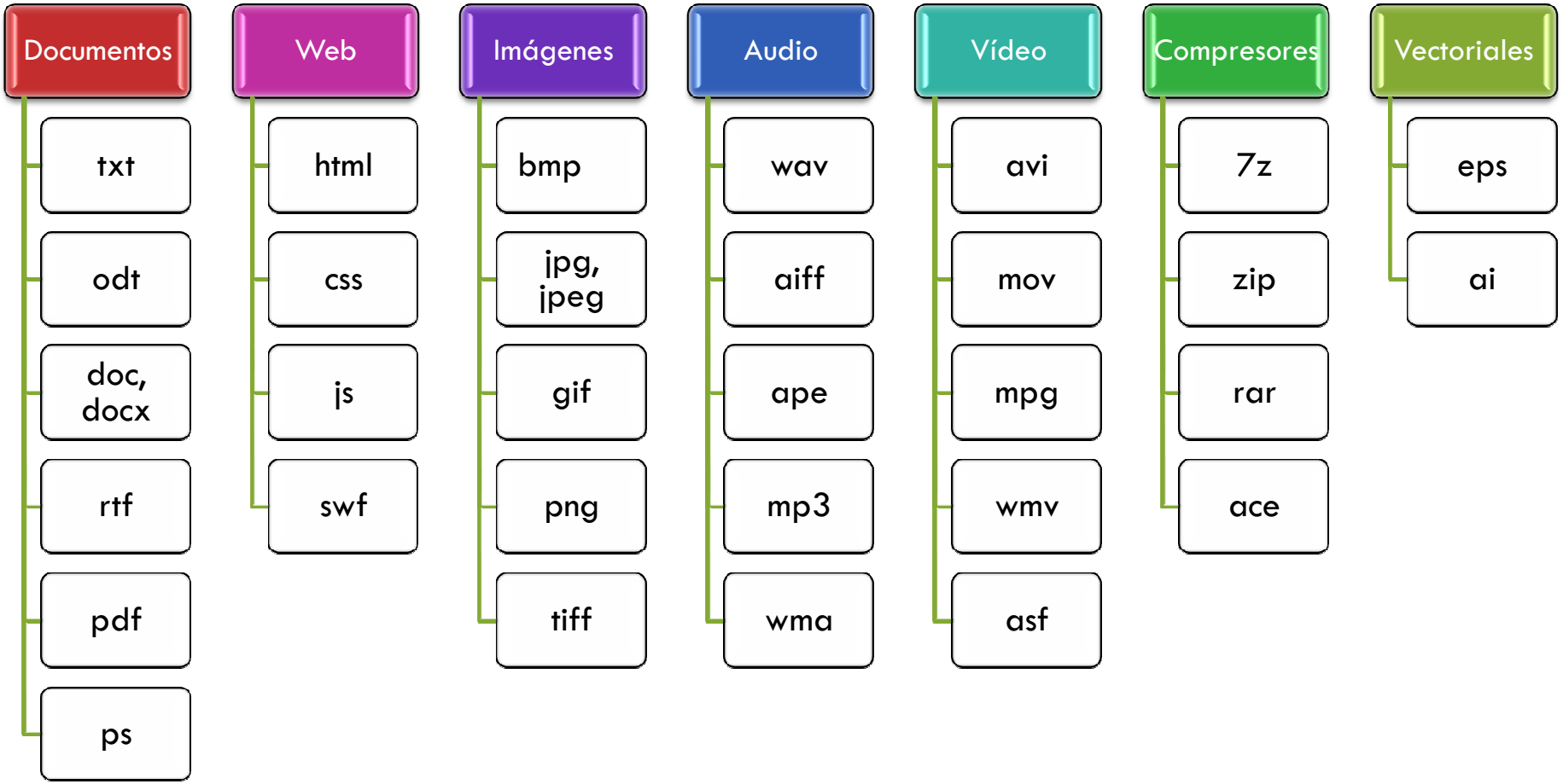
- Un fichero es un conjunto de bits almacenados en un dispositivo.
- Una estructura de información que permite su almacenamiento, una manera de organizar la información en archivos con un nombre y una extensión. La extensión determina el formato que tiene el fichero.

El objetivo es que la información permanezca cuando el ordenador se apague, por ello se almacena en un dispositivo o medio no volátil.

Los ficheros suelen estar formados por un conjunto de líneas o registros , y a su vez, cada registro un conjunto de campos relacionados.

Manejo de Ficheros

Ejemplo de formatos de archivos:



Manejo de Ficheros

Clasificación de los ficheros.

(A) Según el modo de **acceso:**

Acceso Secuencial

- La información se almacena como una secuencia de bytes o caracteres y para acceder al byte i-ésimo hay que haber pasado por los (i-1) anteriores
- La información se procesa en orden, registro a registro

Acceso Aleatorio (Directo)

- Se considera que el fichero está dividido en bloques o registros
- Debe existir un método que permita el acceso a un bloque o registro concreto sin necesidad de recorrer los datos anteriores (ejemplo: el uso de arrays)

Manejo de Ficheros

Clasificación de los ficheros.

(B) Según el tipo de **contenido:**

Ficheros de caracteres (texto)

- Contienen exclusivamente caracteres y pueden ser visualizados por cualquier editor de texto que contenga el SO (notepad, vi…)

Ficheros binarios (bytes)

- No contienen caracteres reconocibles sino que los bytes que contienen son una representación de la información (imágenes, música…)
- Solo podrán ser abiertos por aplicaciones concretas que entiendan cómo están dispuestos los bytes dentro del fichero.

Manejo de Ficheros

Clases asociadas a las operaciones de gestión de ficheros.

Nosotros vamos a manejar ficheros a través del **lenguaje *Java*** y con independencia del modo de acceso java define una clase dentro del paquete ***java.io*** que representa un archivo o directorio. Es la clase **File**.

Un ***objeto*** de la clase *File* representa el nombre de un fichero o directorio que existe en el sistema de ficheros

- Ejemplo de creación del objeto File
- File f=new File("disco\\alumnos.xml");

Los ***métodos*** de *File* permiten obtener toda la información sobre las características del fichero o directorio.

- System.out.println("Nombre:"+ f.getname());
- System.out.println("Dir_padre:"+ f.getparent());
- System.out.println("Ruta relativa:"+ f.getpath());

Constructores para crear un fichero:

i. **File** (**String** DirectorioyFichero)

ii. **File** (**String** Directorio, **String** Fichero)

iii. **File** (**File** Directorio, **String** Fichero)

Manejo de Ficheros

Método	Descripción
getName()	Devuelve el nombre del fichero o directorio
getPath(), getAbsolutePath()	Devuelve camino relativo o absoluto del fichero o directorio
getParent()	Devuelve el nombre del directorio padre o null si no existe
canRead()	Devuelve true si el fichero se puede leer
canWrite()	Devuelve true si el fichero se puede escribir
length()	Devuelve el tamaño del fichero en bytes
createNewFile()	Crea un nuevo fichero vacío, asociado a file si existe un fichero con dicho nombre
delete()	Borra el fichero o directorio
exists()	Devuelve true si el fichero o directorio existe
isDirectory()	Devuelve true si el objeto file corresponde a un directorio
isFile()	Devuelve true si el objeto file corresponde a un fichero
renameTo(File nuevonombre)	Renombra el fichero
mkdir()	Crea un directorio con el nombre indicado en la creación del objeto file

Manejo de Ficheros

Flujos o Streams. Tipos.

El sistema de entrada/salida en Java presenta una gran cantidad de clases que se implementan en el paquete ***java.io*** y usa la abstracción del flujo (*stream*) para tratar la comunicación de información entre una fuente y un destino. **Para realizar la comunicación con los ficheros siempre hay que abrir un stream.**

Tipos de flujos:

Flujos de bytes	Flujos de caracteres
• 8 bits • Orientado a lectura/escritura de datos binarios • De las clases InputStream y OutputStream • Tienen subclases que controlan las diferencias entre los distintos dispositivos	• 16 bits • Orientado a lectura/escritura de caracteres • De las clases Reader y Writer • Objetivo: internacionalización

Manejo de Ficheros

Flujos de Bytes (Byte Streams).

La clase InputStream representa las clases que producen entradas de distintas fuentes (array de bytes, objeto String, fichero, tubería, conexión a internet…).

La clase OutputStream incluye las clases que deciden donde irá la salida (array de bytes, fichero, tubería…)

Flujos de Caracteres (Character Streams).

La clase Reader y Writer manejan flujos de caracteres Unicode. Hay ocasiones en las que hay que usar las clases que manejan bytes en combinación con las clases que manejan caracteres. Para esto hay clases "puente" (convierten streams de bytes a streams de caracteres):

InputStreamReader convierte un InputStream en un Reader (lee bytes y los convierte en caracteres)

OutputStreamReader convierte un OutputStream en un Writer

Manejo de Ficheros

	Clase	Función
InputStream	ByteArrayInputStream	Permite usar un espacio de almacenamiento intermedio de memoria
	StringBufferInputStream	Convierte un String en un InputStream
	FileInputStream	Para leer información de un fichero
	PipedInputStream	Implementa el concepto de "tubería"
	FilterInputStream	Funcionalidad útil a otras clases InputStream
	SequenceInputStream	Convierte dos o más objetos InputStream en un InputStream único
OutputStream	ByteArrayOutputStream	Crea un espacio de almacenamiento intermedio en memoria. Todos los datos que se envían al flujo se ubican en este espacio.
	FileOutputStream	Para enviar información a un fichero
	PipedOutputStream	Implementa el concepto de "tuberia". Cualquier información que se desee escribir aquí será la entrada del PipedInputStream asociado.
	FilterOutputStream	Funcionalidad útil a otras clases OutputStream

Manejo de Ficheros

Clases de flujos de bytes	Clase correspondiente de flujo de caracteres	
InputStream	Reader => InputStreamReader	
OutputStream	Writer => OutputStreamreader	
FileInputStream	***FileReader***	***acceso secuencial**
FileOutputStream	***FileWriter***	***acceso secuencial**
StringBufferInputStream	StringReader	
(no tiene equivalente)	StringWriter	
ByteArrayInputStream	CharArrayReader	
ByteArrayOutputStream	CharArrayWriter	
PipedOutputStream	PipedReader	
PipedOutputStream	PipedWriter	

Para el acceso directo se usa la clase ***RandomAccessFile***

Manejo de Ficheros

Operaciones sobre ficheros.

- **Creación** del fichero:
- **Apertura** del fichero:
- **Cierre** del fichero:
- **Lectura**: Proceso que consiste en transferir información desde el fichero a la memoria principal, a través de variables normalmente.
- **Escritura**: Proceso de transferir información de la memoria (por medio de variables de programa) al fichero.

Las **operaciones sobre un fichero abierto** son:

- ***Altas***: añade un registro al fichero
- ***Bajas***: elimina un registro, que puede ser de forma lógica o física
- ***Modificaciones:*** modifica parte del contenido del registro, y para ello habrá que localizarlo previamente (según sea el método de acceso). El registro se reescribe.
- y ***Consultas***: busca un registro determinado

Manejo de Ficheros

Operaciones sobre ficheros secuenciales

- Altas: al final del último registro
- Bajas: se leen todos los registros y se escriben en un fichero auxiliar a excepción del que queremos dar de baja. Se elimina el fichero original y se renombra el fichero auxiliar con el nombre original
- Modificaciones: proceso similar a las bajas, haciendo uso de un auxiliar, pero modificando el registro
- Consultas: se leen todos los registros desde el principio hasta el registro que se busca.
  - *Los ficheros secuenciales se usan para procesos por lotes (backups…).*
  - *Ventajas: son útiles si el acceso a los registros debe hacerse de forma secuencial.*
  - *Desventajas: no permite acceder directamente a un registro determinado, no permite la actualización pues habría que reescribirlo totalmente y no pueden utilizarse en muchas de las aplicaciones interactivas.*

Manejo de Ficheros

Operaciones sobre ficheros aleatorios

- Puede usarse direccionamiento relativo o absoluto. En disco, se hace relativo, por lo que el programa puede ser independiente de la dirección absoluta en el disco. Absoluta: nº de pista + nº de sector
- Para posicionarnos en un registro, normalmente se usa una función de conversión (*hash*), asociada al tamaño del registro y la clave del mismo, o los campos que identifican el registro. Puede ser que al aplicar la función, nos devuelve la posición ocupada por otro, por lo que hay que buscar otra posición libre (zona de excedentes).
- Altas: **Necesitamos saber su clave**, aplicar la función de conversión para obtener la dirección y escribir el registro.
- Bajas: se suelen hacer de forma lógica (reescribir campo switch). Se localiza mediante su campo clave y se reescribe este campo.
- Modificaciones: hay que localizar el registro mediante clave, modificar el dato y reescribir en la misma posición.
- Consultas: necesitamos su clave, obtener la dirección mediante la función de conversión y leer el registro. Hay que comprobar que el registro buscado está en esa posición y si no, habrá que buscarla en la zona de excedentes.
  - *Ventajas: rápido acceso*
  - *Desventajas: hay que establecer la relación entre la posición que ocupa y su contenido (directa por función o en excedentes). Y se desaprovecha el espacio en disco, pues quedan huecos.*

Manejo de Ficheros

Clases para la gestión de flujos de datos (desde/hacia ficheros).

1. **Ficheros de texto.**

Clases *FileReader y FileWriter*

Siempre hay que hacerlo dentro de un bloque **try-catch** (posibles excepciones: FileNotFoundException, IOException)

Métodos de FileReader:

devuelven el nº de caracteres leídos o -1 si llega al final

- ***int read()*** lee un carácter y lo devuelve
- ***int read(char[] buf)*** lee hasta buf.lenght caracteres de una matriz de caracteres (buf). Los caracteres leídos se almacenan en buf.
- ***int read(char[] buf, int desplazamiento, int n)*** lee hasta n caracteres de datos de la matriz buf comenzando por buf[desplazamiento].

Métodos de FileWriter:

- ***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 en la matriz buf comenzando por buf[desplazamiento]
- ***void write(String str)*** escribe una cadena de caracteres
- ***append (char c)*** añade un carácter a un fichero

Manejo de Ficheros

FileReader y FileWriter

FileReader no contiene métodos que nos permitan leer líneas completas, pero **BufferedReader** si. Lo mismo ocurre con la escritura.

Pero para construir un BufferedReader necesitamos la clase FileReader.

BufferedReader dispone del método *readline()* para leer línea a línea (devolverá null cuando no ha nada más que leer)

BufferedWriter también deriva de la clase Writer, necesitamos a FileWriter. BufferedWriter añade un buffer para la escritura, y para el avance de línea hace uso del método *newLine()*.

La clase **PrintWrite** deriva de Writer y dispone de los métodos print(String) y println(String), similares a System.out, para escribir en un fichero.

Manejo de Ficheros

Clases para la gestión de flujos de datos (desde/hacia ficheros).

2. **Ficheros binarios.**

Almacenan secuencias de dígitos binarios que no son legibles directamente por el usuario. Tienen la ventaja de que ocupan menos espacio en disco.

Clases *FileInputStream y FileOutputStream*

Métodos de FileInputStream:

devuelven el nº de bytes leídos o -1 si llega al final

- ***int read()*** lee un byte y lo devuelve
- ***int read(byte[] b)*** lee hasta b.lenght bytes de una matriz de bytes.
- ***int read(byte[] b, int desplazamiento, int n)*** lee hasta n bytes de la matriz b comenzando por b[desplazamiento].

Métodos de FileOutputStream:

- ***void write(int b)*** escribe un byte
- ***void write(byte[] b***) escribe b.lenght bytes
- ***void write(byte[] b, int desplazamiento, int n***) escribe n bytes a partir de la matriz de bytes de entrada y comenzando por b[desplazamiento]

- Al igual que con los ficheros de caracteres, es posible añadir bytes al final de un fichero, colocando el segundo parámetro de **FileOutputStream** a *true*.

Manejo de Ficheros

Las **clase DataInputStream y DataOutputStream** permiten leer y escribir datos primitivos (int, float, long…), de modo independiente en cada máquina.

Los métodos para abrir un objeto DataInputStream o DataOutputStream son los mismos que para FileInputSTream y FileOutputStream. Y además de read() y write() proporcionan:

Métodos lectura	Métodos escritura
boolean readBoolean()	void writeBoolean(boolean v)
byte readByte()	void writeByte(int v)
int readUnisgnedByte()	void writeBytes(String s)
int readUnsignedShort()	void writeShort(int v)
short readShort()	void writeChars(String s)
char readChar()	void writeChar(int v)
int readInt()	void writeInt(int v)
long readLong()	void writeLong(long v)
float readFloat()	void writeFloat(float v)
double readDouble()	void writeDouble(double v)
String readUTF()	void writeUTF(String str)

Manejo de Ficheros

Objetos SERIALIZABLES.

Java permite guardar objetos en ficheros binarios y para ello debe implementar la interfaz ***Serializable*** que dispone de una serie de métodos para leer y guardar objetos en ficheros binarios.

Ejemplo: objeto tipo empleado con los atributos (nombre, dirección, salario, departamento…)

Esta característica se mantiene incluso a través de la red, por lo que podemos crear un objeto en un ordenador que corra por ejemplo bajo Windows 95/98, serializarlo y enviarlo a través de la red a una estación de trabajo que corra bajo UNIX donde será correctamente reconstruido. No tenemos que preocuparnos de las **diferentes representaciones de datos en los distintos ordenadores.**

Métodos de Serializable:

- ***void readObject(java.io.ObjectInputSTream stream) throws IOException, ClassNotFound Exception:*** para leer un objeto.
- ***void writeObject(jObjectOutputSTream stream) throws IOException Exception:*** para escribir un objeto.

- *Actividad guiada Objetos*

Manejo de Ficheros

Clases para la gestión de flujos de datos (desde/hacia ficheros).

3. **Ficheros de acceso aleatorio.**

Java dispone de la clase ***RandomAccessFile*** con métodos para acceder al contenido de un fichero binario de forma aleatoria (no secuencial) y para posicionarnos en una posición concreta. Puede avanzar y retroceder dentro de un fichero, por lo que NO forma parte de la jerarquía InputStream/OutputStream.

La creación del fichero se puede hacer de dos maneras:

- escribiendo el nombre del fichero:

    fichero=new RandomAccessfile(String nombre, String modoAcceso)

- con un objeto File:

    fichero=new RandomAccessFile(File f, String modoAcceso)

Modo de acceso puede ser: r (solo lectura)

rw (lectura y escritura)

Manejo de Ficheros

Y una vez abierto el fichero, pueden usarse los métodos read() y write() de las clases DataInputStream y DataOutputStream.

La clase RandomAccessFile maneja un puntero que indica la posición actual del fichero (al crearse es 0, y cada read()-write() modifica el valor). *Es importante saber cuánto ocupa cada tipo de datos, para un correcto manejo del puntero.*

Métodos de RandomAccessFile:

- ***long getFilePointer():*** posición actual del puntero del fichero
- ***void seek(long posicion):*** coloca el puntero del fichero en una posición determinada desde el comienzo.
- **long length():** devuelve el tamaño del fichero en bytes. La posición length() marcará el final del fichero.
- **int skipBytes(int desplazamiento):** desplaza ealpuntero desde la posición actual el nº de bytes indicados en desplazamiento.

Manejo de Ficheros

Las variables de tipo primitivo contiene un solo valor del tamaño y formato apropiado de su tipo: un número, un carácter, o un valor booleano.

La tabla siguiente lista los tipos de datos primitivos soportados por Java.

Tipo	Tamaño
Byte	8 bits
Short	16 bits
Int	32 bits
Long	64 bits
Float	32 bits
Double	64 bits
char	16 bits
boolean	1 byte

Manejo de Ficheros

Ejemplo: Inserta datos de alumnos en un fichero de acceso aleatorio.

```
import java.io.*;

public class ficheroaleatorio {
    public static void main(String[] args) throws IOException {
                File f= new File("C:/Users/Trabajo/Documents/borrar6.dat");
                RandomAccessFile alea=new RandomAccessFile(f,"rw");
                String nombre[]={"David","Eduardo","Joseph","Miguel","Luis"};
                int edad[]={19,20,18,21,21};

                StringBuffer buf=null;

                        /*Un objeto String representa una cadena alfanumérica de un valor constante
                        * que no puede ser cambiada después de haber sido creada.
                        * Un objeto StringBuffer representa una cadena cuyo tamaño puede variar
                        * */

                for (int i=0; i<nombre.length; i++){
                alea.writeInt(i+1);// se pone un identificador
                buf=new StringBuffer(nombre[i]);
                buf.setLength(10);  // establece un tamaño fijo para el nombre
                alea.writeChars(buf.toString());
                alea.write(edad[i]);
                }
                alea.close();
    }
}
```

Manejo de Ficheros

Para recorrer el fichero, calculamos tamaño del registro: 4 + 20 + 4 (28 bytes)

Y se utilizará una variable para recorrerlo en bytes.

```
import java.io.*;
public class LeeFicheroAleatorio {
    public static void main(String[] args) throws IOException {
                File f= new File("C:/Users/Trabajo/Documents/borrar6.dat");
                RandomAccessFile alea=new RandomAccessFile(f,"r");
                int id, vedad, pos=0;
                char cnombre[]= new char[10], aux;

                try{
                        while (true){
                                alea.seek(pos);
                                id=alea.readInt();
                                for (int i=0; i<cnombre.length; i++) {
                                        aux=alea.readChar();
                                        cnombre[i]=aux;
                                }
                                String vnombre=new String(cnombre);
                                vedad=alea.readInt();
                                System.out.println("Id: "+id+" Nombre: "+vnombre+" Edad: "+vedad);
                                pos=pos+28;                     //se avanza según tamaño del registro
                        }
                } catch (EOFException e) {}
                alea.close();
}}
```

Manejo de Ficheros

FICHEROS XML.

¿Te acuerdas de XML?

El código debería ser "portable"… pues los datos también

Un documento XML tiene dos estructuras, *una lógica y otra física*.

- **Físicamente**, el documento está compuesto por unidades llamadas **entidades**. Una entidad puede hacer referencia a otra entidad, causando que esta se incluya en el documento. Cada documento comienza con una entidad documento, también llamada **raíz**.

- **Lógicamente**, el documento está compuesto de declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento, todos los cuales están indicados por una marca explícita. Cada documento XML contiene uno o más elementos, cuyos limites están delimitados por etiquetas de comienzo y de final o, en el caso de elementos vacíos, por una etiqueta de elemento vacío. Cada elemento tiene un tipo, identificado por un nombre, denominado identificador genérico, y puede tener un conjunto de especificaciones de atributos. Cada especificación de atributo tiene un nombre y un valor.

Manejo de Ficheros

FICHEROS XML.

XML permite jerarquizar y estructurar la información así como describir los contenidos dentro del propio documento. La información está organizada de forma SECUENCIAL.

XML ofrece mecanismos más versátiles de mostrar datos, flexibilidad y posibilidad de comunicar sistemas heterogéneos.

DOM (Document Object Model): API que sea soportada por todos los procesadores de XML y HTML. La idea detrás de esta API es que podamos representar (a través de javascripts o JavaApplets) documentos XML en los navegadores Web, pero de una forma más sofisticada que los documentos HTML, ya que XML no solo proporciona una sintaxis, sino también una semántica.

Para leer ficheros XML se utiliza un procesador de XML o **parser**. Algunos de estos procesadores:

- DOM: Modelo de Objetos de Documentos
- SAX: API Simple para XML

Son independientes del lenguaje de programación y existen versiones para Java, C…

Manejo de Ficheros

Ejemplo de fichero XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <Empleados>
    <empleado>
          <nombre>Catalina</nombre>
           <apellidos>No Me Acuerdo</apellidos>
           <dni>44444444E</dni>
     </empleado>
</Empleados>
```

Manejo de Ficheros

DOM:

Un procesador XML que utilice este planteamiento almacena toda la estructura del documento en memoria en forma de árbol con nodos padre, nodos hijo y nodos finales. Una vez creado el árbol se van recorriendo los diferentes nodos y se analiza a qué tipo particular pertenecen. Con este tipo de procesamiento se requiere más recursos de memoria y tiempo.

Actividad ejemplo XML_DOM

SAX:

Un procesador XML que utilice este planteamiento lee un fichero XML de forma secuencial y produce una secuencia de eventos en función de los resultados de la lectura. Cada evento invoca a un método definido por el programador. No consume memoria pero tiene el inconveniente que impide tener una visión global del documento por el que navegar.

Manejo de Ficheros

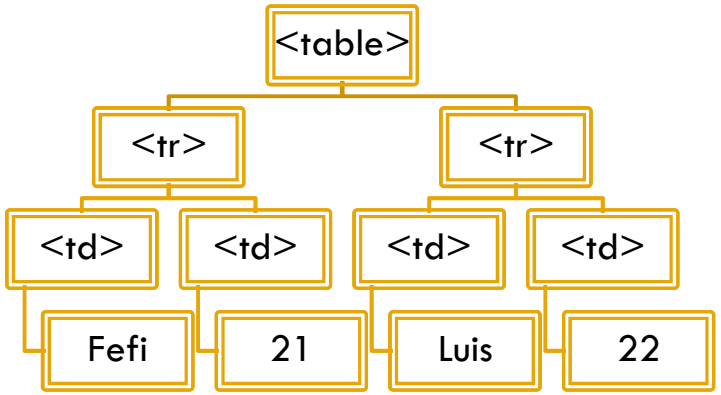
DOM:

Algunas de las interfaces utilizadas son:

- **Document:** objeto que equivale a un ejemplar de un documento XML. Permite crear nuevos nodos en el documento.
- **Element:** equivalente para cada elemento del documento XML. Expone propiedades y métodos para manipular los elementos del documento.
- **Node**: Representa cualquier nodo del documento.
- **NodeList**: Contiene la lista con los nodos hijos de un nodo.

Ejemplo de representación lógica de tabla-HTML con Modelo de Objetos de Documento (tipo árbol).

```
<table>
    <tr> <td>Fefi</td> <td>21</td> </tr>
    <tr> <td>Luis</td> <td>22</td> </tr>
</table>
```



Manejo de Ficheros

Serialización de objetos XML.

XStream es una librería Java que permite serializar objetos Java a XML y realizar su posterior recuperación. Se caracteriza por su eficiencia y simplicidad de uso, generando documentos XML legibles.

Como en el acceso a ficheros XML, se verá con una actividad guiada, **XML_Serializable.doc**

Conversión de ficheros XML a otro formato.

XSL (*Extensible Stylesheet Language*) es una familia de recomendaciones del w3 para definir hojas de estilo en lenguaje XML. Una hoja de estilo XSL describe el proceso e presentación a través de un conjunto de elementos XML.

En la siguiente actividad guiada "**ConversionXML.doc**", con un fichero XML con datos y otro XSL con la presentación, se puede generar un HTML usando Java.