

Ficheros

Componentes de un directorio

El siguiente programa nos daría la lista de los diferentes directorios y archivos que cuelgan de la unidad C:\

```
public static void main(String[] args) {  
    File f = new File("c:/");  
    String[] archivos = (f.list());  
    for (int i = 0; i < archivos.length; i++)  
    {  
        System.out.println(archivos[i]);  
    }  
}
```

En lugar del directorio raíz
podría ser cualquier directorio

Información de un fichero o directorio

El siguiente programa nos daría la información correspondiente al fichero bd0.sql. También podríamos solicitar la información de un directorio. Haced las comprobaciones necesarias.

```
public static void main(String[] args) {  
    File f=new File("g:/bd07.sql");  
    if(f.exists()){  
        System.out.println("INFORMACIÓN SOBRE EL FICHERO");  
        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("¿Se puede ejecutar?   : "+f.canExecute());  
        System.out.println("Tamaño en bytes      : "+f.length());  
        System.out.println("¿Es directorio?      : "+f.isDirectory());  
        System.out.println("¿Es fichero?         : "+f.isFile());  
        System.out.println("¿Es oculto?          : "+f.isHidden());  
        System.out.println("¿Existe?             : "+f.exists());  
        System.out.println("Directorio padre     : "+f.getParent());  
        System.out.println("Fecha última modif.   : "+new SimpleDateFormat  
            ("dd-MM-yyyy").format(new Date((f.lastModified()))));  
    }  
    else{  
        System.out.println("El fichero no existe");  
    }  
}
```

Diferentes formatos de fecha

El siguiente programa nos daría la fecha de la última modificación en diferentes formatos

```
public static void main(String[] args) {  
    File f=new File("c:\\bd07.sql");  
    if(f.exists()){  
        System.out.println("Fecha última modificación en fomato long      :  
"+f.lastModified());  
        System.out.println("Fecha última modificación en formato fecha americana  : "+new  
Date(f.lastModified()));  
        System.out.println("Fecha última modificación en formato fecha europea corta: "+new  
SimpleDateFormat("dd-MM-yyyy").format(new Date((f.lastModified()))));  
        System.out.println("Fecha última modificación en formato fecha europea larga: "+new  
SimpleDateFormat("dd-MM-yyyy HH:mm:ss").format(new Date((f.lastModified()))));  
    }  
    else{  
        System.out.println("El fichero no existe");  
    }  
}
```

*Comprobar el resultado cuando, en lugar
de un fichero, es un directorio*

Ficheros de texto

El los ficheros de texto se graban caracteres (2 bytes). Su formato es visible por el block de notas.

Grabar en ficheros de texto

Grabar en un fichero de texto carácter a carácter

El programa graba un String de caracteres, carácter a carácter en un fichero de texto utilizando la clase FileWriter, método write.

```
public static void main(String[] args) throws IOException{  
    File f=new File("FicheroTextoS.txt");  
    FileWriter fw=new FileWriter(f);  
    String cadena="Esto es una prueba con FileWriter";//Se crea el string  
    char cararray[]=cadena.toCharArray();//se transforma en array de caracteres  
    for(int i=0;i<cararray.length;i++){  
        fw.write(cararray[i]);//se escribe carácter a carácter  
    }  
    fw.close();  
}
```

Grabar en un fichero de texto un array de caracteres

El programa graba un String de caracteres en un fichero de texto utilizando la clase FileWriter, método write(String), método write(char[]).

```
public static void main(String[] args) throws IOException{  
    File f=new File("FicheroTextoS.txt");  
    FileWriter fw=new FileWriter(f);  
    String cadena="Esto es una prueba con FileWriter";  
    fw.write(cadena);  
}
```

```
fw.append('*');
char[]cad=cadena.toCharArray();
fw.write(cad);
fw.close();
}
```

Grabar en un fichero de texto un array de Strings


El programa graba un array de Strings en un fichero de texto utilizando la clase `FileWriter`, método `write(String)` y método `append(String)`. Write y append son equivalentes en este contexto.

```
public static void main(String[] args) throws IOException{
    File f=new File("FicheroTextoS.txt");
    FileWriter fw=new FileWriter(f,true);//true añade al fichero
    String prov[]={"Burgos","Soria","Palencia","León","Valladolid","Santander"};
    for(int i=0;i<prov.length;i++){
//        fw.append(prov[i]);
        fw.write(prov[i]);
        fw.write(' ');
    }
    fw.close();
}
```

Grabar en un fichero de texto línea a línea diferentes Strings.

El programa graba caracteres línea a línea en un fichero de texto utilizando un buffer. La clase `BufferedWriter` deriva de la clase `Writer`. Método `write`

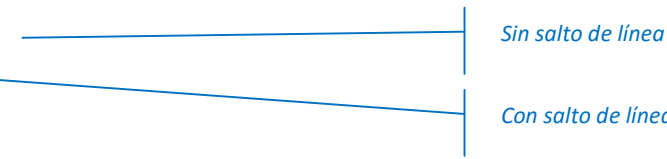
```
public static void main(String[] args) throws IOException {
//    BufferedWriter f=new BufferedWriter(new FileWriter("FicheroTextoB2.txt"));// Crea
    BufferedWriter f=new BufferedWriter(new FileWriter("FicheroTextoB2.txt",true));
    for(int i=0;i<4;i++){
        f.write("fila nº: "+i);
        f.newLine();
    }
    f.close();
}
```



Grabar en un fichero de texto línea a línea diferentes Strings

El programa graba caracteres línea a línea en un fichero de texto utilizando la clase `PrintWriter` método `print(String)` y método `println(String)`. La diferencia entre `print` y `println` es que la última salta de línea. La clase `PrintWriter` pertenece a la clase `FileWriter`.

```
public static void main(String[] args) throws IOException{
    PrintWriter f=new PrintWriter(new FileWriter("FicheroTextoP1.txt"));
    for(int i=10;i<14;i++){
        f.print("fila nº: "+i);
//        f.println("fila nº: "+i);
    }
    f.close();
}
```



Lectura de fichero de texto

Lectura de un fichero de texto carácter a carácter

El programa lee carácter a carácter el contenido de un fichero de texto proporcionando tanto el carácter como su valor ascii.

La clase `FileReader` , método `int read()` devuelve el valor numérico de un carácter ó -1 si no hay nada

```
public static void main(String[] args) throws IOException {
    File f=new File("FicheroTextoE.txt");
    FileReader fr=new FileReader(f);
    int i;
    while((i=fr.read()) !=-1) System.out.println("Caracter: "+(char)i+" Valor ascii:"+i);
    fr.close();
}
```

Lectura de un fichero de texto mediante un array de caracteres

Clase `FileReader`, método `int read(char c[])`: Lee del fichero hasta `c.length` caracteres y los deja en `c[]`. Devuelve el número de caracteres leídos.

```
public static void main(String[] args) throws IOException{
    File f=new File("FicheroTextoE.txt");
    FileReader fr=new FileReader(f);
    int i;
    char b[]=new char[26];
    while((i=fr.read(b)) !=-1){
        System.out.println("nº caracteres :"+i);
        System.out.println(b);
    }
    fr.close();
    char[] c=new char[24];
    fr=new FileReader(f);
    while((i=fr.read(c)) !=-1){
        System.out.println("nº caracteres :"+i);
        System.out.println(c);
    }
    fr.close();
    char[] d=new char[24];
    int cc=0;
    fr=new FileReader(f);
    while((i=fr.read(d)) !=-1){
        System.out.println("nº caracteres :"+i);
        System.out.println(d);
        for(i=0;i<d.length;i++){
            d[i]=32;
        }
    }
    fr.close();
}
```

En caso de no coincidir exactamente el tamaño del array con lo leído da un resultado extraño porque machaca con los n caracteres no leídos los n primeros del buffer leídos en la pasada anterior.

Para evitarlo se limpia el buffer en cada pasada, cargando cada elemento con el valor 32 que se corresponde con el valor ascii de blanco

Lectura de un fichero de texto mediante un array de caracteres con parámetros de desplazamiento y número de caracteres

Clase FileReader, método `int read(char c[],int desplazamiento,int n)`.

Lee del fichero `n` caracteres a partir del desplazamiento y los deja en `c[]`. Devuelve el número de caracteres leídos.

```
public static void main(String[] args) throws IOException{
    File f=new File("FicheroTextoE.txt");
    FileReader fr=new FileReader(f);
    int i;
    char c[]=new char[26];
    while((i=fr.read(c,0,10)) !=-1) System.out.println(c);
    fr.close();
    fr=new FileReader(f);
    char d[]=new char[26];
    for(i=0;i<d.length;i++){
        d[i]=32;
    }
    while((i=fr.read(d,0,10)) !=-1){
        System.out.println(d);
        for(i=0;i<d.length;i++){
            d[i]=32;
        }
    }
    fr.close();
}
```

Para evitar resultados extraños es necesario limpiar el array antes de cada utilización cargando cada elemento con el valor 32 que se corresponde con el valor ascii de blanco

Ficheros Binarios

Los ficheros binarios almacenan secuencias de dígitos binarios que no son directamente legibles por el usuario.

Grabación y lectura en un fichero binario byte a byte

Clase *FileOutoutStream*, método *write(int)* y clase *FileInputStream* método *read()*

```
public static void main(String[] args) throws IOException{
    File f=new File("FichBytes1.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);
    FileInputStream fi=new FileInputStream(f);
    int i;
    for(i=120;i<130;i++){
        fo.write(i);
    }
    fo.close();
    //Visualiza los datos del fichero
    while((i=fi.read())!=-1){
        System.out.println("Caracter :"+(char)i+" Ascii :"+i);
    }
    fi.close();
}
```

Con el parámetro true añade datos al fichero ya existente.

Escribirá los valores ascii que van del 120 al 129.

Grabación y lectura en un fichero binario de un array de números

Clase *FileOutoutStream*, método *write(byte[])* y clase *FileInputStream* método *read(byte[])*

```
public static void main(String[] args) throws IOException{
    File f=new File("FichBytes2.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);
    FileInputStream fi=new FileInputStream(f);
    int i;
    byte[] barray_w={10,11,12,13,14,15,16,21,22,65};
    byte[] barray_r=new byte[10];
    fo.write(barray_w);
    fo.close();
    while((i=fi.read(barray_r))!=-1){
        for(i=0;i<barray_r.length;i++){
            System.out.println(barray_r[i]);
        }
        // System.out.println((char)barray_r[i]);
    }
    fi.close();
}
```

Con el parámetro true añade datos al fichero ya existente.

Si lo transformo en valores ascii, sólo es comprensible el 65 que es 'A', el resto son caracteres de control

Grabación y lectura en un fichero binario de un array de letras

Clase FileOutputStream, método write(byte[]) y clase FileInputStream método read(byte[])

```
public static void main(String[] args) throws IOException {
    File f=new File("FichBytes2.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);
    FileInputStream fi=new FileInputStream(f);
    int i;
    byte[] barray_w={'P','r','u','e','b','a',' ','c','o','n',' ','l','e','t','r','a','s',' ','3','3'};
    byte[] barray_r=new byte[20];
    fo.write(barray_w);
    fo.close();
    //Visualiza los datos del fichero
    while((i=fi.read(barray_r))!=-1){
        for(i=0;i<barray_r.length;i++){
            System.out.println((char)barray_r[i]);
        }
    }
    fi.close();
}
```

Grabación y lectura en un fichero binario de un array de números con parámetros de desplazamiento y número de bytes.

Clase FileOutputStream, método write(byte[], int desplazamiento, int número de bytes) y clase FileInputStream método read(byte[], int desplazamiento, int número de bytes)

```
public static void main(String[] args) throws IOException{
    File f=new File("FichBytes2.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);
    FileInputStream fi=new FileInputStream(f);
    int i;
    byte[] barray_w={10,11,12,13,14,15,16,21,22,65};
    byte[] barray_r=new byte[9];
    fo.write(barray_w,0,7);
    fo.close();
    while((i=fi.read(barray_r,2,7))!=-1){
        for(i=0;i<barray_r.length;i++){
            System.out.println(barray_r[i]);
        }
        // System.out.println((char)barray_r[i]);
    }
    fi.close();
}
```

Si lo transformo en valores ascii, sólo es comprensible el 65 que es 'A', el resto son caracteres de control

Grabación y lectura en un fichero binario de un array de letras con parámetros de desplazamiento y número de bytes.

Clase FileOutputStream, método write(byte[], int desplazamiento, int número de bytes) y clase FileInputStream método read(byte[], int desplazamiento, int número de bytes)

```
public static void main(String[] args) throws IOException {
    File f=new File("FichBytes2.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);//Añade
    FileInputStream fi=new FileInputStream(f);
    int i;
    byte[] barray_w={'P','r','u','e','b','a',' ','c','o','n',' ','l','e','t','r','a','s'};
    byte[] barray_r=new byte[7];
    fo.write(barray_w,5,5);
    fo.close();
    while((i=fi.read(barray_r,2,5))!=-1){
        for(i=0;i<barray_r.length;i++){
            System.out.println((char)barray_r[i]);
        }
    }
    fi.close();
}
```

Las clases DataOutputStream y DataInputStream.

Estas clases pertenecientes a las clases FileOutputStream y FileInputStream respectivamente proporcionan métodos para la escritura y lectura de los diferentes tipos de datos primitivos.

Los tipos de datos primitivos y su tamaño en byte es la siguiente:

byte	1 byte.
short	2 bytes.
int	4 bytes.
long	8 bytes.
float	4 bytes.
double	8 bytes
boolean	1 byte.
char	2 bytes.

Grabación y lectura en un fichero binario de variables pertenecientes a los diferentes tipos de datos primitivos

Clase DataOutputStream perteneciente a la clase FileOutputStream y clase DataInputStream perteneciente a la clase FileInputStream con los métodos necesarios para cada tipo de datos primitivo.

```
public static void main(String[] args) throws IOException{
    File f=new File("FicheroData1.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);
    DataOutputStream dou=new DataOutputStream(fo);
    byte by=12;
```



```
short peque=12;
int entero=12;
long largo=12;
float flota=12.12f;
double doble=12.12;
boolean b=true;
char caracter='a';
dou.writeByte(by);
dou.writeShort(peque);
dou.writeInt(entero);
dou.writeLong(largo);
dou.writeFloat(flota);
dou.writeDouble(doble);
dou.writeBoolean(b);
dou.writeChar(caracter);
dou.close();
```

```
FileInputStream fi=new FileInputStream(f);
DataInputStream di=new DataInputStream(fi);
try{
    System.out.println("Byte :"+di.readByte());
    System.out.println("Short :"+di.readShort());
    System.out.println("Int :"+di.readInt());
    System.out.println("Long :"+di.readLong());
    System.out.println("Float :"+di.readFloat());
    System.out.println("Double :"+di.readDouble());
    System.out.println("Boolean :"+di.readBoolean());
    System.out.println("Char :"+di.readChar());
}catch(EOFException eo){}
di.close();
}
```

Dada la diferencia de longitudes en bytes de los diferentes tipos primitivos, la lectura debe hacerse en el mismo orden de variables que la grabación.

Grabación y lectura en un fichero binario de arrays de Strings y arrays de enteros.

clase FileOutputStream ->Clase DataOutputStream -> método write(int). Graba entero.

clase FileOutputStream ->Clase DataOutputStream -> método writeUTF(String). Graba String.

clase FileInputStream -> clase DataInputStream ->método readInt(). Lee entero.

clase FileInputStream -> clase DataInputStream ->método readUTF(). Lee String.

```
public static void main(String[] args) throws IOException{
    File f=new File("FicheroData1.dat");
    FileOutputStream fo=new FileOutputStream(f);
    // FileOutputStream fo=new FileOutputStream(f,true);
    DataOutputStream dou=new DataOutputStream(fo);
    String nombres[]{"Ana","Pedro","María","Juan","Rosario"};
    int edades[]={15,20,18,17,19};
    //Grabación en el fichero
    for(int i=0;i<edades.length;i++){
        dou.writeUTF(nombres[i]);
        dou.writeInt(edades[i]);
    }
```

```
    }  
    dou.close();  
    //Lectura del fichero y salida por pantalla  
    FileInputStream fi=new FileInputStream(f);  
    DataInputStream di=new DataInputStream(fi);  
    String n;  
    int e;  
    try{  
        while(true){  
            n=di.readUTF();  
            e=di.readInt();  
            System.out.println("Nombre: "+n+" edad: "+e);  
        }  
    }catch(EOFException eo){}  
    di.close();  
}
```

Ficheros aleatorios

Hasta ahora todos los accesos que hemos hecho a los ficheros han sido secuenciales. Se empezaba la lectura por el primer byte o carácter y se iban leyendo uno a uno hasta llegar al final del fichero. Lo mismo ocurría para la escritura o grabación.

Java dispone de la clase `RandomAccessFile` con métodos para acceder al contenido de un fichero binario de forma aleatoria y para posicionarse en una posición concreta del mismo. Se usa la clase `RandomAccessFile(String nombre, String modo)`. Los modos posibles son: "r" para lectura, y "rw" para lectura y grabación.

Grabación secuencial en un fichero aleatorio

Clase `RandomAccessFile(String nombre, String modo)`.

Métodos

`writeInt(int)` para grabar un entero.

`writeChars(String)` para grabar una `String`.

`writeDouble(Double)` para grabar un doble.

```
public static void main(String[] args) throws IOException {
    File f=new File("FicheroRandomEmpleados.dat");
    RandomAccessFile rf=new RandomAccessFile(f,"rw");
    String apellidos[]{"Antunez","Perez","Marquez","Jimenez","Rodriguez"};
    int dpto[]={10,20,10,20,30};
    Double salario[]={1200.30,1556.89,2000.34,890.29,1300.34};
    StringBuffer buffer=null;
    int n=apellidos.length;
    for(int i=0;i<n;i++){
        rf.writeInt(i);
        buffer=new StringBuffer(apellidos[i]);
        buffer.setLength(10);
        rf.writeChars(buffer.toString());
        rf.writeInt(dpto[i]);
        rf.writeDouble(salario[i]);
    }
    rf.close();
}
```

StringBuffer: es un String de longitud modificable.

10 caracteres para el apellido

Lectura secuencial de un fichero aleatorio

Clase `RandomAccessFile(String nombre, String modo)`.

Métodos

`readInt()` para leer un entero.

`readChar()` para leer un carácter.

`readDouble()` para leer un docble.

```
public static void main(String[] args) throws IOException {
    File f=new File("FicheroRandomEmpleados.dat");
    RandomAccessFile rf=new RandomAccessFile(f,"r");
    int id,dpto,posicion=0;
    double salario;
```

```
char apellido[]=new char[10];
System.out.println(" ID APELLIDOS   DPTO   SALARIO");
while(posicion<rf.length()){
    rf.seek(posicion);
    id=rf.readInt();
    for(int i=0;i<apellido.length;i++){
        apellido[i]=rf.readChar();
    }
    String Sapellido=new String(apellido);
    dpto=rf.readInt();
    salario=rf.readDouble();

    System.out.println(" "+id+" "+Sapellido+" "+" "+dpto+" "+salario);
    posicion=posicion+36;
}
rf.close();
}
```

Acceso aleatorio a un fichero aleatorio

Métodos

void seek(long posición): Coloca el puntero del fichero en una posición determinada a partir del 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 desplazamiento.

Long getFilePointer(): Devuelve la posición actual del puntero del fichero.

```
public static void main(String[] args) throws IOException {
    File f=new File("FicheroRandomEmpleados.dat");
    RandomAccessFile rf=new RandomAccessFile(f,"r");
    int id,dpto,posicion=0;
    double salario;
    char apellido[]=new char[10];
    int n_reg=0;
    int long_reg=36;
    posicion=n_reg*long_reg;
    if (posicion<rf.length()&& posicion>0){
        System.out.println(" ID APELLIDOS   DPTO   SALARIO");
        rf.seek(posicion);
        id=rf.readInt();
        for(int i=0;i<apellido.length;i++){
            apellido[i]=rf.readChar();
        }
        String Sapellido=new String(apellido);
        dpto=rf.readInt();
        salario=rf.readDouble();
        System.out.println(" "+id+" "+Sapellido+" "+" "+dpto+" "+salario);
        rf.close();
    }
}
```

```
    else{
        System.out.println("El registro solicitado no se encuentra en el fichero");
    }
}
```

Ficheros binarios para almacenar objetos serializables

En lugar de grabar en un fichero los diferentes atributos de los objetos, podemos grabar en un fichero binario objetos completos.

Para poder hacerlo el objeto debe implementar la interfaz Serializable que dispone de una serie de métodos con los que podremos guardar y leer objetos en ficheros binarios.

Los métodos más importantes son los siguientes:

Para leer un objeto del fichero.

void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException

Exception.

Para grabar un objeto en el fichero.

void writeObject(java.io.ObjectOutputStream stream) throws IOException.

Creación de la clase Persona

```
public class Persona implements Serializable{
    private String nombre;
    private int edad;

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

    public void setNombre(String nombre) {this.nombre = nombre;}
    public void setEdad(int edad) {this.edad = edad;}
    public String getNombre() {return nombre;}
    public int getEdad() {return edad;}
}
```

Implementación de la interfaz
Serializable

Grabación de objetos persona. Versión 1.

```
public static void main(String[] args) throws IOException{
    Persona persona;
    File f= new File("FicheroObjetos2.dat");
    FileOutputStream fo=new FileOutputStream(f,true);
    ObjectOutputStream d=new ObjectOutputStream(fo);
    String nombres[]={"Ana","Pedro","María","Juan","Rosario"};
    int edades[]={35,30,38,37,39};
    for(int i=0;i<nombres.length;i++){
        persona=new Persona(nombres[i],edades[i]);
        d.writeObject(persona);
    }
    d.close();
}
```

Lectura de objetos persona

```
public static void main(String[] args) throws IOException, ClassNotFoundException{
    Persona p;
    File f=new File("FicheroObjetos2.dat");
    FileInputStream fi=new FileInputStream(f);
    ObjectInputStream oi=new ObjectInputStream(fi);
    try{
        do{
            p=(Persona)oi.readObject();
            System.out.println("Nombre: "+p.getNombre()+" edad: "+p.getEdad());
        }while(true);
    }catch(EOFException e){
        oi.close();
    }
}
```

StreamCorruptedException

Si grabamos datos en el fichero, lo cerramos y después volvemos a abrirlo para añadir datos `FileInputStream(fichero,true)`, se escribe una nueva cabecera al principio de los nuevos objetos. Luego al leer el fichero se produce el error `StreamCorruptedException`. Para evitarlo debemos crear la clase `MiObjectOutputStream` sobrescribiendo el método `writeStreamHeader()` modificándolo para que no haga nada.

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

public class MiObjectOutputStream extends ObjectOutputStream{
    public MiObjectOutputStream(OutputStream out)throws IOException
    {
        super(out);
    }
    protected MiObjectOutputStream()throws IOException,SecurityException
    {
        super();
    }
    protected void writeStreamHeader()throws IOException
    {
    }
}
```

La grabación de objetos persona utilizará `ObjectOutputStream` la primera vez y `MiObjectOutputStream` las siguientes ocasiones.

Grabación de objetos persona. Versión 2.

```
public static void main(String[] args) throws IOException {
    Persona persona;
    File f = new File("FicheroObjetos.dat");

    String nombres[] = {"Ana", "Pedro", "María", "Juan", "Rosario"};
    int edades[] = {35, 30, 38, 37, 39};
    if (f.exists()) {
        FileOutputStream fo = new FileOutputStream(f, true);
        MiObjectOutputStream d = new MiObjectOutputStream(fo);
        for (int i = 0; i < nombres.length; i++) {
            persona = new Persona(nombres[i], edades[i]);
            d.writeObject(persona);
        }
        d.close();
    } else {
        FileOutputStream fo = new FileOutputStream(f, true);
        ObjectOutputStream d = new ObjectOutputStream(fo);
        for (int i = 0; i < nombres.length; i++) {
            persona = new Persona(nombres[i], edades[i]);
            d.writeObject(persona);
        }
        d.close();
    }
}
```

FICHEROS XML

Para leer los ficheros XML y acceder a su contenido y estructura, se utiliza un procesador de XML o parser. El procesador lee los documentos y proporciona acceso a su contenido y estructura. Los dos procesadores que vamos a ver son DOM y SAX. Son independientes del lenguaje de programación y existen versiones particulares para Java, VisualBasic, C, etc.

DOM (Modelo de Objetos de Documento)

Almacena toda la estructura del documento en memoria en forma de árbol con nodos padre, nodos hijo y nodos finales (sin descendientes). Con el árbol creado se van recorriendo los nodos y determina a qué tipos pertenecen. Tiene su origen en el W3C. Necesita más recursos de memoria y tiempo que SAX.

SAX (API Simple para XML)

Lee un fichero XML de forma secuencial y produce una secuencia de eventos (comienzo y fin de documento, comienzo y fin de etiqueta, etc.) dependiendo de los resultados de la lectura. Cada evento invoca a un método de finido por el programador. Prácticamente no consume memoria. En contra tiene que no proporciona una visión global del documento.

DOM .Utilización práctica.

Necesitamos las clases e interfaces del paquete org.w3c.dom contenido en el JSDK y el paquete javax.xml.parser del API estándar de Java que proporciona un par de clases abstractas que toda implementación DOM para Java debe extender. Las dos clase fundamentales son DocumentBuilderFactory y DocumentBuilder.

Para generar un fichero XML a partir de un árbol DOM se usa el paquete javax.xml.transform que permite especificar una fuente y un resultado.

Para utilizar DOM con Java se necesitan las siguientes interfaces (hay más) para nuestros ejemplos.

Document: Es un objeto que equivale a un ejemplar de un documento XML. Permite crear nuevos nodos en el documento.

Element: Cada elemento del documento XML tiene un equivalente en un objeto de este tipo. Expone propiedades y métodos para manipular los elementos del documento y sus atributos.

Node: Representa cualquier nodo del documento.

NodeList: Contiene una lista con los nodos hijo de un nodo.

Text: son los datos carácter de un elemento.

Creación de un fichero XML con DOM

Vamos a utilizar como entrada el fichero de empleados creado como fichero aleatorio en un ejercicio anterior.

Necesitaremos las siguientes clases e interfaces.

```
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.*;

public static void main(String[] args)throws IOException {
    File fichero=new File("../e_ficherosAleatorios/ficheroRandomEmpleados.dat");
    RandomAccessFile ficheroR=new RandomAccessFile(fichero,"r");

    int id,dep,posicion=0;
    Double salario;
    char apellido[]=new char[10],aux;

    DocumentBuilderFactory factoria=DocumentBuilderFactory.newInstance();
    try{
        DocumentBuilder constructor=factoria.newDocumentBuilder();
        DOMImplementation implementacion=constructor.getDOMImplementation();
        Document documento=implementacion.createDocument(null, "Empleados", null);
        documento.setXmlVersion("1.0");

        do {
            ficheroR.seek(posicion);
            id=ficheroR.readInt();
            for(int i=0;i<apellido.length;i++){
                aux=ficheroR.readChar();
                apellido[i]=aux;
            }
        }
```



```
String apellidoS=new String(apellido);
dep=ficheroR.readInt();
salario=ficheroR.readDouble();

if (id>=0){
    Element raiz=documento.createElement("empleado");
    documento.getDocumentElement().appendChild(raiz);
    CrearElemento("id",Integer.toString(id),raiz,documento);
    CrearElemento("apellido",apellidoS.trim(),raiz,documento);
    CrearElemento("dep",Integer.toString(dep),raiz,documento);
    CrearElemento("salario",Double.toString(salario),raiz,documento);
}
posicion=posicion+36;
}while(ficheroR.getFilePointer()<ficheroR.length());
Source fuente=new DOMSource(documento);
// Salida a fichero
Result resultado=new StreamResult(new java.io.File("Empleados.xml"));
Transformer transformador=TransformerFactory.newInstance().newTransformer();
transformador.transform(fuente, resultado);
// Salida por pantalla
Result console=new StreamResult(System.out);
transformador.transform(fuente, console);
}catch(Exception e){System.err.println("Error: "+e);}
ficheroR.close();
}

static void CrearElemento(String datoEmple,String valor,
    Element raiz,Document documento){
    Element elemento=documento.createElement(datoEmple);
    Text texto=documento.createTextNode(valor);
    raiz.appendChild(elemento);
    elemento.appendChild(texto);
}
```

Lectura de un fichero XML con DOM

*Utilizamos como entrada el fichero Empleados.xml creado en el apartado anterior.
Necesitaremos las siguientes clases e interfaces.*

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

public static void main(String[] args) {
    DocumentBuilderFactory factoria=DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder constructor=factoria.newDocumentBuilder();
        Document documento=constructor.parse(new File("Empleados.xml"));
        documento.getDocumentElement().normalize();
        System.out.println("Elemento raiz: "+
            documento.getDocumentElement().getNodeName());
        NodeList empleados=documento.getElementsByTagName("empleado");
        for(int i=0;i<empleados.getLength();i++){
            Node empleado=empleados.item(i);
```

```

        if (empleado.getNodeType()==Node.ELEMENT_NODE){
            Element elemento=(Element)empleado;
            System.out.println("ID: " + getNodo("id",elemento));
            System.out.println("Apellido: " + getNodo("apellido",elemento));
            System.out.println("Departamento: " + getNodo("dep",elemento));
            System.out.println("Salario: " + getNodo("salario",elemento));
        }
    }
}
} catch (Exception e){e.printStackTrace();}
}
private static String getNodo(String etiqueta,Element elemento)
{
    NodeList nodo=elemento.getElementsByTagName(etiqueta).item(0).getChildNodes();
    Node valornodo=(Node) nodo.item(0);
    return valornodo.getNodeValue();
}
}

```

SAX . Utilización práctica

SAX(API Simple para XML) es un conjunto de clases e interfaces que ofrecen una herramienta para el procesamiento de documentos XML. Analiza los elementos de forma secuencial. Esto supone poco consumo de memoria pero imposibilidad de tener una visión global del documento.

La lectura de un documento XML produce eventos que ocasiona la llamada a métodos. Los eventos son encontrar la etiqueta de inicio y fin del documento (startDocument() y endDocument()), la etiqueta de inicio y fin de elemento (startElement() y endElement()), etc.

Lectura de un fichero XML con SAX

Utilizamos como entrada el fichero Empleados.xml. Necesitamos las siguientes clases e interfaces.

```

import java.io.*;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;

    public static void main(String[] args) throws FileNotFoundException, IOException,
    SAXException
    {
        XMLReader procesadorXML=XMLReaderFactory.createXMLReader();
        GestionContenido gestor=new GestionContenido();
        procesadorXML.setContentHandler(gestor);
        InputSource ficheroXML=new InputSource("Empleados.xml");
        procesadorXML.parse(ficheroXML);
    }
class GestionContenido extends DefaultHandler{
    public GestionContenido(){
        super();
    }
}

```

```
public void startDocument(){
    System.out.println("Comienzo del Documento XML");
}
public void endDocument(){
    System.out.println("Final del Documento XML");
}
public void startElement(String uri,String nombre,
    String nombreC,Attributes atts) {
    System.out.println("\tPrincipio Elemento: " + nombre);
}
public void endElement(String uri,String nombre,String nombreC){
    System.out.println("\tFin Elemento: " + nombre);
}
public void characters(char[] ch,int inicio,int longitud)
    throws SAXException{
    String car=new String(ch,inicio,longitud);
    car=car.replaceAll("[\t\n]", "");
    if (!car.equals("")){
        System.out.println("\tCaracteres: "+car);
    }
}
}
```

Serialización de objetos a XML utilizando la librería XStream.

Hay que descargar los JAR desde el sitio web <http://xstream.codehaus.org/download.html>. Para el ejemplo se descarga el fichero Binary distribution (xstream-distribution-1.4.2-bin.zip), y después de descomprimirlo buscar el JAR xstream-1.4.2.jar que está en la carpeta lib. También necesitamos el fichero kxml2-2.3.0.jar que se puede descargar desde el apartado Optional Dependencies. Una vez que tenemos los dos ficheros los definimos en el CLASSPATH, por ejemplo, supongamos que tenemos los JAR en la carpeta D:\uni1\xstream
SET CLASSPATH=.; D:\uni1\xstream\kxml2-2.3.0.jar; D:\uni1\xstream\xstream-1.4.2.jar
Vamos a utilizar el mismo fichero

Creación de la clase Persona

```
public class Persona implements Serializable{
    private String nombre;
    private int edad;
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public void setNombre(String nombre) {this.nombre = nombre;}
    public void setEdad(int edad) {this.edad = edad;}
    public String getNombre() {return nombre;}
    public int getEdad() {return edad;}
}
```

Creación de la clase ListaPersonas

```
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;
}
}
```

Creación del fichero binario con objetos persona (ya visto en apartado anterior)

```
public static void main(String[] args) throws IOException{
    Persona persona;
    File fichero= new File("ObjetosPersona.dat");
    FileOutputStream flujoS=new FileOutputStream(fichero,true);
    ObjectOutputStream flujoSO=new ObjectOutputStream(flujoS);
    String nombres[]={ "Ana", "Pedro", "María", "Juan", "Rosario"};
    int edades[]={35,30,38,37,39};
    for(int i=0;i<edades.length;i++){
        persona=new Persona(nombres[i],edades[i]);
        flujoSO.writeObject(persona);
    }
    flujoSO.close();
}
```

Lectura del fichero de objetos persona (ya visto en apartado anterior)

```
public static void main(String[] args) throws IOException, ClassNotFoundException{
    Persona p;
    File fichero=new File("ObjetosPersona.dat");
    FileInputStream flujoE=new FileInputStream(fichero);
    ObjectInputStream flujoEO=new ObjectInputStream(flujoE);
    try{
        while(true){
            p=(Persona)flujoEO.readObject();
            System.out.println("Nombre: "+p.getNombre()+" edad: "+p.getEdad());
        }
    }catch(EOFException e){
        flujoEO.close();
    }
}
```

Serialización del fichero de objetos persona a XML

```
public static void main(String[] args) throws IOException,ClassNotFoundException {
    File fichero=new File("ObjetosPersona.dat");
    FileInputStream flujoE=new FileInputStream(fichero);
    ObjectInputStream flujoEO=new ObjectInputStream(flujoE);
    System.out.println("Comienza el proceso de creacion del fichero a XML...");
    ListaPersonas l_persona=new ListaPersonas();
```

```
try {
    while(true){
        Persona persona=(Persona) flujoEO.readObject();
        l_persona.add(persona);
    }
}catch(EOFException eo){}
flujoEO.close();
System.out.println(l_persona);
try {
    XStream flujoXS=new XStream();
    flujoXS.alias("ListaPersonasMunicipio",ListaPersonas.class);
    flujoXS.alias("DatosPersona",Persona.class);
    flujoXS.addImplicitCollection(ListaPersonas.class,"lista");
    flujoXS.toXML(l_persona,new FileOutputStream("Personas.xml"));
    System.out.println("Creado fichero XML..");
}catch (Exception e)
{e.printStackTrace();}
}
```

Lectura del fichero XML serializado con Notepad++ para comprobación.

```
<ListaPersonasMunicipio>
<DatosPersona>
  <nombre>Ana</nombre>
  <edad>35</edad>
</DatosPersona>
<DatosPersona>
  <nombre>Pedro</nombre>
  <edad>30</edad>
</DatosPersona>
<DatosPersona>
  <nombre>María</nombre>
  <edad>38</edad>
</DatosPersona>
<DatosPersona>
  <nombre>Juan</nombre>
  <edad>37</edad>
</DatosPersona>
<DatosPersona>
  <nombre>Rosario</nombre>
  <edad>39</edad>
</DatosPersona>
</ListaPersonasMunicipio>
```

Lectura del fichero XML serializado

```
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");
ListaPersonas listadoTodas=(ListaPersonas)
    xstream.fromXML(new FileInputStream("Personas.xml"));
System.out.println("Numero 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.println("Nombre: " + p.getNombre() +
        ", edad: "+ p.getEdad());
}
System.out.println("Fin de listado...");
}
```

Conversión de ficheros XML a otro formato

XSL (Extensible Stylesheet Language) es toda una familia de recomendaciones del World Wide Web Consortium (<http://www.w3.org/Style/XSL/>) para expresar hojas de estilo en lenguaje XML. Una hoja de estilo XSL describe el proceso de presentación a partir de un conjunto de elementos XML.

En el siguiente ejemplo se verá cómo a partir de un fichero XML que contiene datos y otro XSL que contiene la presentación de datos se puede generar un fichero HTML usando Java. Los ficheros son los siguientes.

Alumnos.xml

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

AlumnosPlantilla.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match='/>
    <html><xsl:apply-templates/></html>
</xsl:template>
```

```
<xsl:template match='listadealumnos'>
  <head><title>LISTADO DE ALUMNOS</title></head>
  <body>
    <h1>LISTA DE ALUMNOS</h1>
    <table border='2'>
      <tr><th>Nombre</th><th>Edad</th></tr>
      <xsl:apply-templates select='alumno' />
    </table>
  </body>
</xsl:template>
<xsl:template match='alumno'>
  <tr><xsl:apply-templates /></tr>
</xsl:template>
<xsl:template match='nombre|edad'>
  <td><xsl:apply-templates /></td>
</xsl:template>
</xsl:stylesheet>
```

Partiendo de un fichero XML y un fichero XSL conseguimos un fichero HTML

Para ello necesitamos las siguientes clases e interfaces

```
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.*;

public static void main(String[] args) throws IOException{
  String hojaEstilo="Alumnosplantilla.xsl";
  String datosAlumnos="Alumnos.xml";
  File paginaHTML=new File("Alumnos.html");
  FileOutputStream flujoSalida=new FileOutputStream(paginaHTML);
  Source estilo=new StreamSource(hojaEstilo);
  Source datos=new StreamSource(datosAlumnos);
  Result resultado= new StreamResult(flujoSalida);
  try{
    Transformer transformador=
      TransformerFactory.newInstance().newTransformer(estilo);
    transformador.transform(datos, resultado);
  }catch(Exception e){System.err.println("Error:"+e); }
  flujoSalida.close();
}
```

Lectura del fichero HTML con Notepad++ para comprobación.

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>LISTADO DE ALUMNOS</title>
</head>
<body>
```

```
<h1>LISTA DE ALUMNOS</h1>
<table border="2">
<tr>
<th>Nombre</th><th>Edad</th>
</tr>
<tr>

<td>Juan</td>
<td>19</td>

</tr>
<tr>

<td>Maria</td>
<td>20</td>

</tr>
</table>
</body>
</html>
```