DESARROLLO DE APLICACIONES MULTIPLATAFORMA

ACCESO A DATOS

UNIDAD 2. - Manejo de ficheros

Sumario

Caso práctico

Ana está empezando a cursar la Formación en Centros de Trabajo (FCT).

Ya ha tenido unas reuniones con **Juan** y **María**, para saber cómo se trabaja en BK programación. Aunque está haciendo el módulo de FCT en esta empresa, ya sabe que a veces tendrá que salir a otras empresas acompañada de sus tutores para ver los requisitos de los sistemas que la empresa tenga que informatizar y, en ocasiones, **Antonio**, quizás también se apunte para echar una mano.

Ana está nerviosa, también ilusionada, y tiene muchas ganas de conocer de cerca la realidad de lo que ha estudiado en clase.

Ahora verá el uso de los conocimientos adquiridos de diferentes módulos, y buscará respuestas a posibles dudas que se vayan planteando.

En clase les habían explicado la importancia de los ficheros en el acceso a datos. -Es importante repasar los conceptos -piensa Ana.

1.- INTRODUCCIÓN.

Si estás estudiando este módulo, es probable que ya hayas estudiado el de programación, por lo que no te serán desconocidos muchos conceptos que se tratan en este tema.

Ya sabes que cuando apagas el ordenador, los datos de la memoria RAM se pierden. Un ordenador utiliza ficheros para guardar los datos. Piensa que los datos de las canciones que oyes en mp3, las películas que ves en formato avi, o mp4, etc., están, al fin y al cabo, almacenadas en ficheros, los cuales están grabados en un soporte como son los discos duros, DVD, pendrives, etc.

Se llama a los datos que se guardan en ficheros **datos persistentes**, porque persisten más allá de la ejecución de la aplicación que los trata. Los ordenadores almacenan los ficheros en unidades de almacenamiento secundario como discos duros, discos ópticos, etc. En esta unidad veremos, entre otras cosas, cómo hacer con Java las operaciones de crear, actualizar y procesar ficheros.

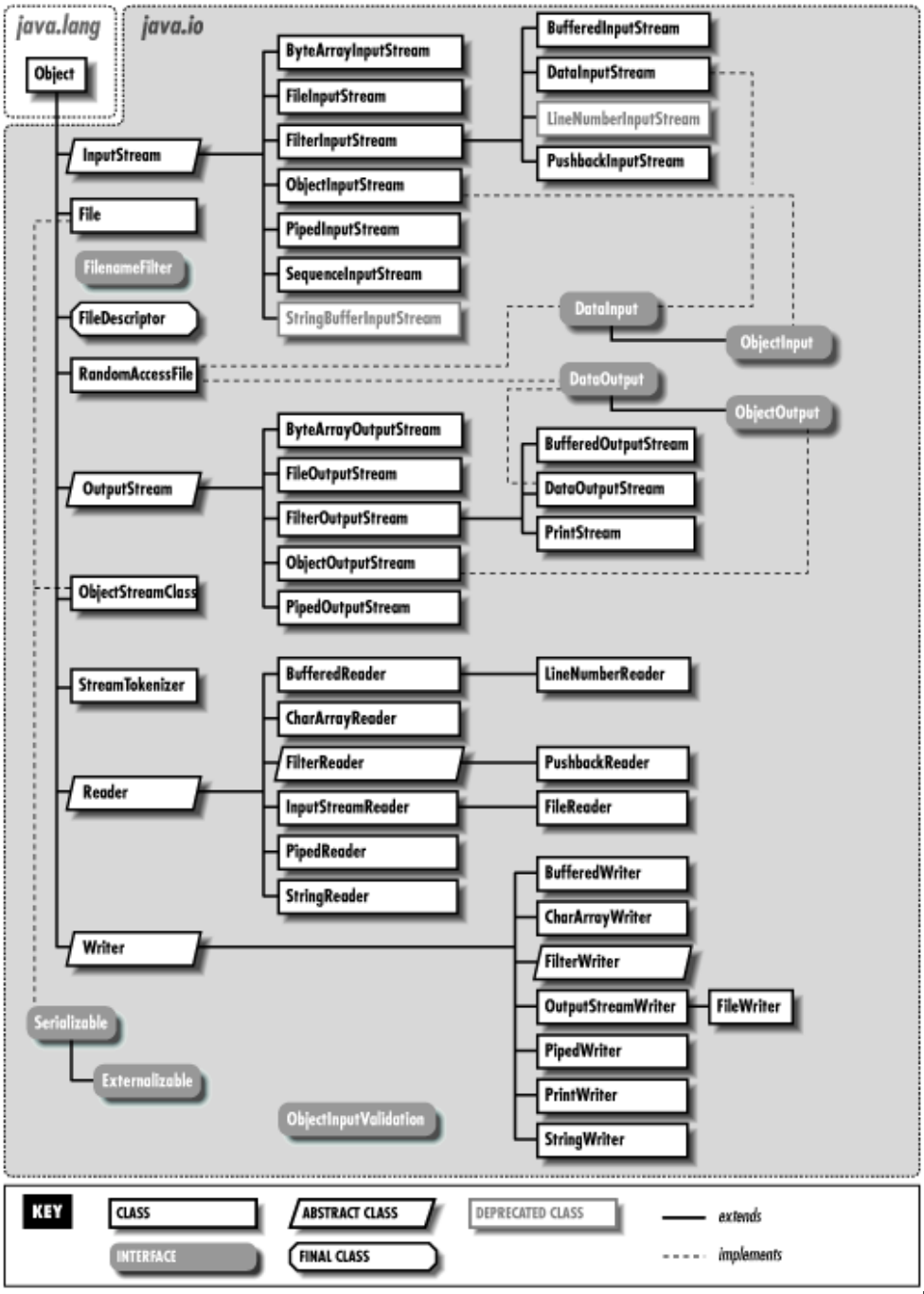
A las operaciones, que constituyen un flujo de información del programa con el exterior, se les conoce como Entrada/Salida (E/S).

Las operaciones de E/S en Java las proporciona el paquete estándar de la API de Java denominado `java.io` que incorpora interfaces, clases y excepciones para acceder a todo tipo de ficheros.

La **librería** `java.io` contiene las clases necesarias para gestionar las operaciones de entrada y salida con Java. Estas clases de E/S las podemos agrupar fundamentalmente en:

- Clases para leer entradas desde un flujo de datos.
- Clases para escribir entradas a un flujo de datos.
- Clases para operar con ficheros en el sistema de ficheros local.
- Clases para gestionar la serialización de objetos.

En la imagen puedes ver las clases de las que se dispone en `java.io`.



2.- CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS Y DIRECTORIOS.

Caso práctico

Ana le comenta a **Antonio** -Por lo que nos han comentado, vamos a tener que utilizar bastante los ficheros. ¿Cómo te manejas tú con ellos?

-Pues tan bien como tú -responde **Antonio**, -yo también estudié el módulo de Programación. ¿Es que no recuerdas que en el módulo estudiamos un tema sobre ficheros?

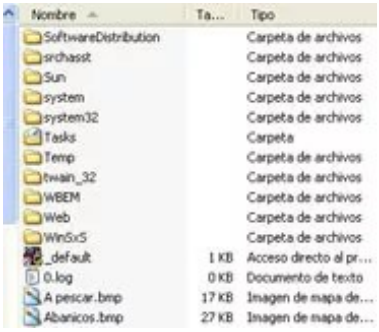
-Sí, pero es que, como había tantos métodos para listar, renombrar archivos, etc., ya casi no me acuerdo; y eso que hace poco que lo estudiamos -contesta **Ana**.

Antonio intenta tranquilizar a **Ana** y le dice que no se preocupe, que en cuanto se les presente la ocasión de tener que programar con ficheros, seguro que no tienen problema y refrescan los conceptos que aprendieron en su día.

En efecto, tal y como dicen Ana y Antonio, hay bastantes métodos involucrados en las clases que en Java nos permiten manipular ficheros y carpetas o directorios.

Vamos a ver la clase `File` que nos permite hacer unas cuantas operaciones con ficheros, también veremos cómo filtrar ficheros, o sea, obtener aquellos con una característica determinada, como puede ser que tengan la extensión **.odt**, o la que nos interese, y por último, en este apartado también veremos como crear y eliminar ficheros y directorios.

2.1.- Clase File.



¿Para qué sirve esta clase, qué nos permite? La clase `File` proporciona una representación abstracta de ficheros y directorios.

Esta clase, permite examinar y manipular archivos y directorios, independientemente de la

plataforma en la que se esté trabajando: Linux, Windows, etc.

Las instancias de la clase `File` representan nombres de archivo, no los archivos en sí mismos.

El archivo correspondiente a un nombre puede ser que no exista, por esta razón habrá que controlar las posibles **excepciones**.

Un objeto de clase `File` permite examinar el nombre del archivo, descomponerlo en su rama de directorios o crear el archivo si no existe mediante el método `createNewFile()`;

Para archivos que existen, a través del objeto `File`, un programa puede examinar los atributos del archivo, cambiar su nombre, borrarlo o cambiar sus permisos. Dado un objeto `File`, podemos hacer las siguientes operaciones con él:

- **Renombrar** el archivo, con el método `renameTo()`. El objeto `File` dejará de referirse al archivo renombrado, ya que el `String` con el nombre del archivo en el objeto `File` no cambia.
- **Borrar** el archivo, con el método `delete()`. También, con `deleteOnExit()` se borra cuando finaliza la ejecución de la máquina virtual Java.
- **Crear** un nuevo fichero con un nombre único. El método estático `createTempFile()` crea un fichero temporal y devuelve un objeto `File` que apunta a él. Es útil para crear archivos temporales, que luego se borran, asegurándonos tener un nombre de archivo no repetido.
- **Establecer** la fecha y la hora de modificación del archivo con `setLastModified()`. Por ejemplo, se podría hacer: `new File("prueba.txt").setLastModified(new Date().getTime());` para establecerle la fecha actual al fichero que se le pasa como parámetro, en este caso prueba.txt.
- **Crear** un directorio, mediante el método `mkdir()`. También existe `mkdirs()`, que crea los directorios superiores si no existen.
- **Listar** el contenido de un directorio. Los métodos `list()` y `listFiles()` listan el contenido de un directorio. `list()` devuelve un vector de `String` con los nombres de los archivos, `listFiles()` devuelve un vector de objetos `File`.
- **Listar** los nombres de archivo de la raíz del sistema de archivos, mediante el método estático `listRoots()`.

2.1.1.- Ejemplos de la Clase File.

Para crear un objeto File, se puede utilizar cualquiera de los tres constructores siguientes:

- File(String directorioyfichero): en Linux: new File(“/directorio/fichero.txt”);
  en plataformas Microsoft Windows: new File("C:\\directorio\\fichero.txt");
- File(String directorio, String nombrefichero): new File(“directorio", "fichero.txt");
- File(File directorio, String fichero): new File(new File(“directorio"), "fichero.txt");

En Linux se utiliza como prefijo de una ruta absoluta “/". En Microsoft Windows, el prefijo de un nombre de ruta consiste en la letra de la unidad seguida de “:” y, posiblemente, seguida por "\\" si la ruta es absoluta.

El siguiente ejemplo muestra la lista de ficheros en el directorio actual. Se utiliza el método list() que devuelve un array de Strings con los nombres de los ficheros y directorios contenidos en el directorio asociado al objeto File. Para indicar que estamos en el directorio actual creamos un objeto File y le pasamos el parámetro “.”:

VerDir (0.01 MB)

```
package file;

import java.io.*;

public class VerDir {
        public static void main(String[] args) {
                //String dir = "."; //Directorio actual
                String dir ="C:\\TMP\\DAM\\AD";
                File f = new File(dir);
                String[] archivos = f.list();
                System.out.printf("Ficheros en el directorio actual: %d %n",
archivos.length);
                for (int i = 0; i < archivos.length; i++) {
                        File f2 = new File(f, archivos[i]);
                        System.out.printf("Nombre: %s, es fichero?: %b, es
directorio?: %b %n", archivos[i], f2.isFile(),
                                        f2.isDirectory());
                }
        }
}
```

El siguiente ejemplo muestra información del fichero seleccionado;

VerInf (0.01 MB)

```
package file;

import java.io.*;
public class VerInf {
public static void main(String[] args) {
  System.out.println("INFORMACIÓN SOBRE EL FICHERO:");
  File f = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\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 leer      : "+f.canRead());
        System.out.println("Se puede escribir  : "+f.canWrite());
```

```
        System.out.println("Tamaño              : "+f.length());
                System.out.println("Es un directorio    : "+f.isDirectory());
                System.out.println("Es un fichero       : "+f.isFile());
                System.out.println("Nombre del directorio padre:
"+f.getParent());
  }
 }
}
```

Por último, este ejemplo crea un directorio (de nombre NUEVODIR) en el directorio actual, a continuación crea dos ficheros vacíos en dicho directorio y uno de ellos lo renombra. En este caso para crear los ficheros se definen 2 parámetros en el objeto File: File(File directorio, String nombrefich), en el primero indicamos el directorio donde se creará el fichero y en el segundo indicamos el nombre del fichero:

CrearDir (0.01 MB)

```
package file;

import java.io.*;
public class CrearDir {
 public static void main(String[] args) {
   File d = new File("NUEVODIR"); //directorio que creo a partir del actual
   File f1 = new File(d,"FICHERO1.TXT");
   File f2 = new File(d,"FICHERO2.TXT");

   d.mkdir();//CREAR DIRECTORIO

   try {
    if (f1.createNewFile())
       System.out.println("FICHERO1 creado correctamente...");
    else
       System.out.println("No se ha podido crear FICHERO1...");

    if (f2.createNewFile())
       System.out.println("FICHERO2 creado correctamente...");
    else
       System.out.println("No se ha podido crear FICHERO2...");
   } catch (IOException ioe) {ioe.printStackTrace();}

   f1.renameTo(new File(d,"FICHERO1NUEVO"));//renombro FICHERO1

   try {
        File f3 = new File("NUEVODIR/FICHERO3.TXT");
          f3.createNewFile();//crea FICHERO3 en NUEVODIR
   } catch (IOException ioe) {ioe.printStackTrace();}
 }
}
```

Para saber más

En este enlace puedes ver ejemplos para obtener las propiedades de los ficheros, usando la clase `File`:

Ejemplo creando carpetas o directorios en Java

2.1.2.- Creación y eliminación de ficheros y directorios.

Cuando queramos **crear un fichero**, podemos proceder del siguiente modo:

```
try {
   // Creamos el objeto que encapsula el fichero
   File fichero = new File("c:\\prueba\\miFichero.txt");
   // A partir del objeto File creamos el fichero físicamente
   if (fichero.createNewFile())
       System.out.println("El fichero se ha creado correctamente");
   else
       System.out.println("No ha podido ser creado el fichero");
} catch (Exception ioe) {
        ioe.getMessage();
}
```

Código de crear un fichero.

Para **borrar un fichero** podemos usar la clase `File`, comprobando previamente si existe el fichero, del siguiente modo:

```
File fichero = new File( "C:\\prueba", "agenda.txt");
if (fichero.exists())
    fichero.delete();
```

Para **crear directorios**, podríamos hacer:

```
try {
// Declaración de variables
String directorio = "C:\\prueba";
String varios = "carpeta1/carpeta2/carpeta3";

// Crear un directorio
boolean exito = (new File(directorio)).mkdir();
if (exito)
    System.out.println("Directorio: " + directorio + " creado");
    // Crear varios directorios
    exito = (new File(varios)).mkdirs();
if (exito)
    System.out.println("Directorios: " + varios + " creados");
 }catch (Exception e){
        System.err.println("Error: " + e.getMessage());
}
```

Código de crear un directorio.

Para **borrar un directorio** con Java, tendremos que borrar cada uno de los ficheros y directorios que éste contenga. Al poder almacenar otros directorios, se podría recorrer recursivamente el directorio para ir borrando todos los ficheros.

Se puede listar el contenido del directorio e ir borrando con:

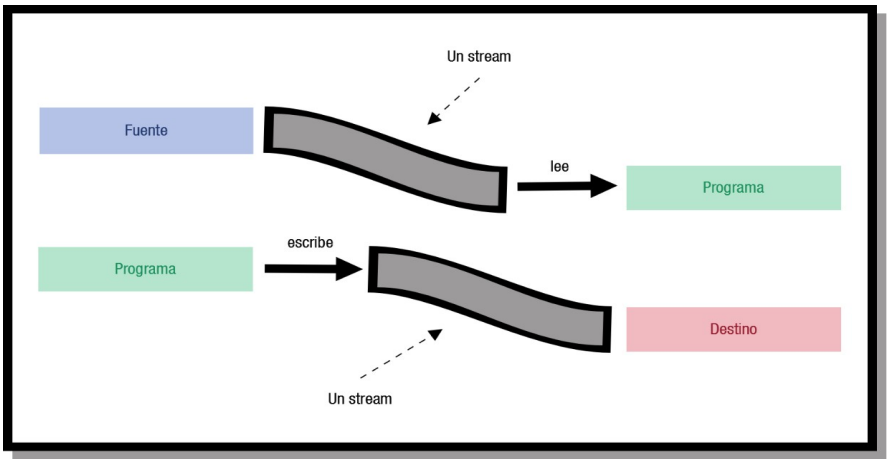
```
File[] ficheros = directorio.listFiles();
```

Si el elemento es un directorio, lo sabemos mediante el método `isDirectory()`.

3.- FLUJOS.

Caso práctico

Ana y **Antonio** saben que van a tener que ayudar a **Juan** y a **María** en labores de programación de ficheros en Java. Así que, además, de la clase `File`, van a necesitar utilizar otros conceptos relacionados con la entrada y salida: los flujos o `streams`. Ana recuerda que hay dos tipos de flujos: flujos de caracteres y flujos de `bytes`.



Un programa en Java, que necesita realizar una operación de entrada/salida (en adelante E/S), lo hace a través de un flujo o `stream`.

Un **flujo** es una **abstracción de todo aquello que produce o consume información**.

La vinculación de este flujo al dispositivo físico la hace el sistema de entrada y salida de Java.

Las clases y métodos de E/S que necesitamos emplear son las mismas independientemente del dispositivo con el que estemos actuando. Luego, el núcleo de Java sabrá si tiene que tratar con el teclado, el monitor, un sistema de archivos o un socket de red; liberando al programador de tener que saber con quién está interactuando.

Java define dos tipos de flujos en el paquete `java.io`:

- **Byte streams** (8 bits): proporciona lo necesario para la gestión de entradas y salidas de bytes y su uso está orientado a la lectura y escritura de datos binarios. El tratamiento del flujo de bytes viene determinado por dos clases abstractas que son `InputStream` y `OutputStream` Estas dos clases definen los métodos que sus subclases tendrán implementados y, de entre todos, destacan `read()`y `write()` que leen y escriben bytes de datos respectivamente.

- **Character streams** (16 bits): de manera similar a los flujos de bytes, los flujos de caracteres están determinados por dos clases abstractas, en este caso: `Reader` y `Writer`. Dichas clases manejan flujos de caracteres Unicode. Y también de ellas derivan subclases concretas que implementan los métodos definidos en ellas siendo los más destacados los métodos `read()` y `write()` que leen y escriben caracteres de datos respectivamente.

3.1.- Flujos basados en bytes.

Para el tratamiento de los flujos de bytes, hemos dicho que Java tiene dos clases abstractas que son `InputStream` y `OutputStream`.

Los archivos binarios guardan una representación de los datos en el archivo, es decir, cuando guardamos texto no guardan el texto en si, sino que guardan su representación en un código llamado .

Las clases principales que heredan de `OutputStream`, para la escritura de ficheros binarios son:

- `FileOutputStream`: escribe bytes en un fichero. Si el archivo existe, cuando vayamos a escribir sobre él, se borrará. Por tanto, si queremos añadir los datos al final de éste, habrá que usar el constructor `FileOutputStream(String filePath, boolean append)`, poniendo `append` a `true`.
- `ObjectOutputStream`: convierte objetos y variables en vectores de bytes que pueden ser escritos en un `OutputStream`.
- `DataOutputStream`, que da formato a los tipos primitivos y objetos `String`, convirtiéndolos en un flujo de forma que cualquier `DataInputStream`, de cualquier máquina, los pueda leer. Todos los métodos empiezan por "`write`", como `writeByte()`, `writefloat()`, etc.

De **InputStream**, para lectura de ficheros binarios, destacamos:

- `FileInputStream`: lee bytes de un fichero.
- `ObjectInputStream`: convierte en objetos y variables los vectores de bytes leídos de un InputStream.

En el siguiente ejemplo puedes ver cómo se crea un flujo de salida y otro de entrada a un archivo binario mediante las clase `FileInputStream y FileOutputStream`:

EscribirFichBytes

```
package streamsBinarios;

import java.io.*;
public class EscribirFichBytes {
  public static void main(String[] args) throws IOException {
   File fichero = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\streamsBinarios\\
fichBytes.dat");//declara fichero
   //crea flujo de salida hacia el fichero
   FileOutputStream fileout = new FileOutputStream(fichero,true);
   //crea flujo de entrada
   FileInputStream filein = new FileInputStream(fichero);
   int i;

   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
  }
}
```

En los siguientes ejemplos se puede ver cómo se escribe y se lee un archivo binario con `DataOutputStream y DataInputStream`

EscribirFichData

```
package streamsBinarios;


import java.io.*;
public class EscribirFichData {
  public static void main(String[] args) throws IOException {
          File fichero = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\
streamsBinarios\\fichData.dat");
          DataOutputStream dataOS = new DataOutputStream(new
FileOutputStream(fichero));

          String nombres[] = {"Ana","Luis
Miguel","Alicia","Pedro","Manuel","Andrés"};

          int edades[] = {14,15,13,15,16,12};

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

LeerFichData

```
package streamsBinarios;

import java.io.*;
public class LeerFichData {
  public static void main(String[] args) throws IOException {
        File fichero = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\
streamsBinarios\\fichData.dat");
        DataInputStream dataIS = new DataInputStream(new
FileInputStream(fichero));

        String n;
        int e;

   try {
    while (true) {
        n = dataIS.readUTF(); //recupera el nombre
        e = dataIS.readInt(); //recupera la edad
        System.out.println("Nombre: " + n + ", edad: " + e);
    }
    }catch (EOFException eo) {}

   dataIS.close();  //cerrar stream
  }
}
```

3.1.1.- Objetos en ficheros binarios.

LeerFichObjectJava nos permite guardar objetos en ficheros binarios; para poder hacerlo, el objeto tiene que implementar la interfaz **Serializable** que dispone de una serie de métodos con los que podremos guardar y leer objetos en ficheros binarios. Los más importantes a utilizar son:

- `Object readObject()  throws IOException, ClassNotFound Exception:` para leer un objeto del ObjectInputStream
- `void writeObject(Object obj) throws IOException:` para escribir el objeto especificado en el ObjectOutputStream

La serialización de objetos de Java permite tomar cualquier objeto que implemente la interfaz **Serializable** y convertirlo en una secuencia de bits, que puede ser posteriormente restaurada para regenerar el objeto original.

Para leer y escribir objetos serializables a un stream se utilizan las clases Java `ObjectlnputStream y ObjectOutputStream` respectivamente. A continuación se muestra la clase `Persona` que implementa la interfaz Serializable y, que utilizaremos para escribir y leer objetos en un fichero binario.

Persona

```
package objetosBinarios;


import java.io.Serializable;
@SuppressWarnings("serial")
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 nom){nombre=nom;}
        public void setEdad(int ed){edad=ed;}

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

El siguiente ejemplo escribe objetos Persona en un fichero. Necesitamos crear un flujo de salida a disco con FileOutputStream y a continuación se crea el flujo de salida ObjectOutputStream, que es el que procesa los datos y se ha de vincular al fichero de FileOutputStream:

EscribirFichObject

```
package objetosBinarios;

import java.io.*;

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

   Persona persona;//defino variable persona

   File fichero = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\objetosBinarios\\
fichPersona.dat");//declara el fichero
   FileOutputStream fileout = new FileOutputStream(fichero,true);  //crea el
flujo de salida
    //conecta el flujo de bytes al flujo de datos
   ObjectOutputStream dataOS = new ObjectOutputStream(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};
   System.out.println("GRABO LOS DATOS DE PERSONA.");
   for (int i=0;i<edades.length; i++){ //recorro los arrays
      persona= new Persona(nombres[i],edades[i]); //creo la persona
          dataOS.writeObject(persona); //escribo la persona en el fichero
          System.out.println("GRABO LOS DATOS DE PERSONA.");
   }
   dataOS.close();  //cerrar stream de salida
   }
}
```

Para leer objetos Persona del fichero necesitamos el flujo de entrada a disco FilelnputStream y a continuación crear el flujo de entrada ObjectlnputStream que es el que procesa los datos y se ha de

vincular al fichero de FilelnputStream:

LeerFichObject

```
package objetosBinarios;

import java.io.*;

public class LeerFichObject {
        public static void main(String[] args) throws IOException,
ClassNotFoundException {
                Persona persona; // defino la variable persona
                File fichero = new File ("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\
objetosBinarios\\fichPersona.dat");
                ObjectInputStream dataIS = new ObjectInputStream(new
FileInputStream(fichero));

                int i = 1;
                try {
                        while (true) { // lectura del fichero
                                persona = (Persona) dataIS.readObject(); // leer
una Persona
                                System.out.print(i + "=>");
                                i++;
                                System.out.printf("Nombre: %s, edad: %d %n",
persona.getNombre(),persona.getEdad());
                        }
                } catch (EOFException eo) {
                        System.out.println("FIN DE LECTURA.");
                } catch (StreamCorruptedException x) {
                }

                dataIS.close(); // cerrar stream de entrada
        }
}
```

3.2.- Flujos basados en caracteres.

Para los flujos de caracteres, Java dispone de dos clases abstractas: `Reader` y `Writer`.

Si se usan sólo `FileInputStream`, `FileOuputStream`, `FileReader` o `FileWriter`, cada vez que se efectúa una lectura o escritura, se hace físicamente en el disco duro. Si se leen o escriben pocos caracteres cada vez, el proceso se hace costoso y lento por los muchos accesos a disco duro.

Los `BufferedReader`, `BufferedInputStream`, `BufferedWriter` y `BufferedOutputStream` añaden un buffer intermedio. Cuando se lee o escribe, esta clase controla los accesos a disco. Así, si vamos escribiendo, se guardarán los datos hasta que haya bastantes datos como para hacer una escritura eficiente.

Al leer, la clase leerá más datos de los que se hayan pedido. En las siguientes lecturas nos dará lo que tiene almacenado, hasta que necesite leer otra vez físicamente. Esta forma de trabajar hace los accesos a disco más eficientes y el programa se ejecuta más rápido.

En un programa Java para crear o abrir un fichero se invoca a la clase **File** y a continuación, se crea el flujo de entrada hacia el fichero con la clase **FileReader**. Después se realizan las operaciones de lectura o escritura y cuando terminemos de usarlo lo cerraremos mediante el método close(). El siguiente ejemplo lee cada uno de los caracteres del fichero de texto de nombre LeerFichTexto.java y los muestra en pantalla, los métodos read() pueden lanzar la excepción **IOException**, por ello en main() se ha añadido throws IOException ya que no se incluye el manejador **try-catch**:

LeerFichTexto

```
package streamsTexto;

import java.io.*;

public class LeerFichTexto {
  public static void main(String[] args) throws IOException {
    File fichero = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\streamsTexto\\
fichero1.txt"); //declarar fichero
    FileReader fic = new FileReader(fichero); //crear el flujo de entrada

    int i;
    while ((i = fic.read()) != -1) //se va leyendo un carácter hasta que lea -1
        System.out.print((char) i); //cast a char

    //para leer de 20 en 20 utilizamos el metodo read pasándole el buffer de
caracteres
    /*  char b[] = new char [20];
        while ((i = fic.read(b)) != -1)
                System.out.println(b);*/

        //System.out.println( (char) i + "==>"+ i);

    fic.close(); //cerrar fichero

  }
}
```

En el ejemplo anterior, la expresión ((char) i) convierte el valor entero recuperado por el método read() a carácter, es decir, hacemos un cast a char. Se llega al final del fichero cuando el método read() devuelve -1.

El siguiente ejemplo escribe 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:

EscribirFichTexto

```
package streamsTexto;

import java.io.*;
```

```
public class EscribirFichTexto {
  public static void main(String[] args) throws IOException {
   File fichero = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\streamsTexto\\
fichtexto.txt");//declara fichero
   FileWriter fic = new FileWriter(fichero); //crear el flujo de salida. Si
existe borra lo que había FileWriter(fichero, true)
   String cadena ="Esto es una prueba con FileWriter";
   char[] cad = cadena.toCharArray();//convierte un String en array de
caracteres

   for(int i=0; i<cad.length; i++)
           fic.write(cad[i]);  //se va escribiendo un carácter
   fic.append('*'); //añado al final un *


   fic.write("\n");
   fic.write(cad);//escribir un array de caracteres
   String c="\n*esto es lo ultimo*";
   fic.write(c);//escribir un String


   String prov[] = {"Albacete","Avila","Badajoz"};
   fic.write("\n");
   for(int i=0; i<prov.length; i++) {
              fic.write(prov[i]);
              fic.write("\n");
   }

   fic.close();    //cerrar fichero

  }
}
```

En vez de escribir los caracteres uno a uno, también podemos escribir todo el array: fic.write(cad);

3.2.1.- Búferes en ficheros de texto

FileReader no contiene métodos que nos permitan leer líneas completas, pero BuíferedReader sí; dispone del método readLine() que lee una línea del fichero y la devuelve, o devuelve null si no hay nada que leer o llegamos al final del fichero. También dispone del método read() para leer un carácter. Para construir un BufferedReader necesitamos la clase FileReader:

BufferedReader fichero = new BufferedReader(new FileReader(NombreFichero));

El siguiente ejemplo lee el fichero LeerFichTexto.java línea por línea y las va visualizando en pantalla, en este caso las instrucciones se han agrupado dentro de un bloque try-catch:

LeerFichTextoBuf

```
package streamsTexto;
```

```
import java.io.*;
public class LeerFichTextoBuf {
  public static void main(String[] args) {
  try{
          File fic = new File("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\streamsTexto\\
fichtexto.txt");//declara fichero
      BufferedReader fichero = new BufferedReader(new FileReader(fic));

      String linea;
      while((linea = fichero.readLine())!=null)
        System.out.println(linea);
      fichero.close();
        }
        catch (FileNotFoundException fn ){
               System.out.println("No se encuentra el fichero");}
        catch (IOException io) {
               System.out.println("Error de E/S ");}

  }
}
```

La clase BufferedWriter también deriva de la clase Writer. Esta clase añade un buffer para realizar una escritura eficiente de caracteres. Para construir un BufferedWriter necesitamos la clase FileWriter:

BufferedWriter fichero = new BufferedWriter (new FileWriter (NombreFichero));

El siguiente ejemplo escribe 10 filas de caracteres en un fichero de texto y después de escribir cada fila salta una línea con el método newLine():

EscribirFichTextoBuf

```
package streamsTexto;

import java.io.*;
public class EscribirFichTextoBuf {
  public static void main(String[] args) {
  try{
      BufferedWriter fichero = new BufferedWriter (new FileWriter("C:\\TMP\\
DAM\\AD\\AD\\UD2\\src\\streamsTexto\\fichtexto1.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 ");}
  }
}
```

Por último, la clase PrintWriter, que también deriva de Writer, posee los métodos print(String) y println(String) (idénticos a los de System.out) para escribir en un fichero. Ambos reciben un String y lo imprimen en un fichero, el segundo método salta de línea. Para construir un PrintWriter necesitamos la clase FileWriter:

PrintWriter fichero = new PrintWriter (new FileWriter(NombreFichero));

El ejemplo anterior usando la clase PrintWriter y el método println() quedaría:

EscribirFichTextoPrint

```
package streamsTexto;

import java.io.*;
public class EscribirFichTextoPrint {
  public static void main(String[] args) {
  try{
      PrintWriter fichero = new PrintWriter (new FileWriter("C:\\TMP\\DAM\\AD\\
AD\\UD2\\src\\streamsTexto\\fichtexto2.txt"));
      for (int i=1; i<11; i++){
            fichero.println("Fila numero: "+i); //escribe una línea
      }
      fichero.close();
      }
        catch (FileNotFoundException fn ){
               System.out.println("No se encuentra el fichero");}
        catch (IOException io) {
               System.out.println("Error de E/S ");}
  }
}
```

4.- FORMAS DE ACCESO A UN FICHERO.

Caso práctico

El momento de programar ha llegado, y **Antonio** se pregunta qué opción será mejor para el acceso a los ficheros: si **acceso secuencial o aleatorio**. ¿Qué uso se le va a dar a estos ficheros?, ¿para qué van a servir cuando la aplicación informática esté en funcionamiento? Esa es la cuestión clave, piensa **Antonio**.

Hemos visto que en Java puedes utilizar **dos tipos de ficheros (de texto o binarios) y dos tipos de acceso a los ficheros (secuencial o aleatorio)**. Si bien, y según la literatura que consultemos, a veces se distingue una tercera forma de acceso denominada concatenación, tuberías o pipes.

- **Acceso aleatorio:** los archivos de acceso aleatorio, al igual que lo que sucede usualmente con la memoria (RAM=Random Access Memory), permiten acceder a los datos en forma no secuencial, desordenada. Esto implica que el archivo debe estar disponible en su totalidad al momento de ser accedido, algo que no siempre es posible.
- **Acceso secuencial:** En este caso los datos se leen de manera secuencial, desde el comienzo del archivo hasta el final (el cual muchas veces no se conoce a priori). Este es el caso de la lectura del teclado o la escritura en una consola de texto, no se sabe cuándo el operador terminará de escribir.

Citas para pensar

> El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad.
>
> Víctor Hugo.

4.1.- Operaciones básicas sobre ficheros de acceso secuencial.

Como **operaciones más comunes en ficheros de acceso secuencial**, tenemos el acceso para:

- Crear un fichero o abrirlo para grabar datos.
- Leer datos del fichero.

- Borrar información de un fichero.
- Copiar datos de un fichero a otro.
- Búsqueda de información en un fichero.
- Cerrar un fichero.

4.2.- Operaciones básicas sobre ficheros de acceso aleatorio.

A menudo, no necesitas leer un fichero de principio a fin, sino simplemente acceder al fichero como si fuera una base de datos, donde se salta de un registro a otro; cada uno en diferentes partes del fichero. Java proporciona una clase `RandomAccessFile` para este tipo de entrada/salida.

Esta clase:

- Permite leer y escribir sobre el fichero, no es necesario dos clases diferentes.
- Necesita que le especifiquemos el modo de acceso al construir un objeto de esta clase: sólo lectura o bien lectura y escritura.
- Posee métodos específicos de desplazamiento como `seek(long posicion)` o `skipBytes(int desplazamiento)` para poder movernos de un registro a otro del fichero, o posicionarnos directamente en una posición concreta del fichero.

Por esas características que presenta la clase, un archivo de acceso directo tiene sus registros de un tamaño fijo o predeterminado de antemano.

La clase posee dos constructores:

- `RandomAccessFile(File file, String mode).`
- `RandomAccessFile(String name, String mode).`

En el primer caso se pasa un objeto `File` como primer parámetro, mientras que en el segundo caso es un String. El modo es: "r" si se abre en modo lectura o "rw" si se abre en modo lectura y escritura.

El ejemplo que se muestra a continuación inserta datos de empleados en un fichero aleatorio. Por cada empleado también se insertará un identificador que coincidirá con el índice +1 con el que se recorren los arrays. La longitud del registro de cada empleado es la misma (36 bytes) y los tipos que se insertan y su tamaño en bytes es el siguiente:

- Se inserta en primer lugar un entero, que es el identificador, ocupa 4 bytes.
- A continuación una cadena de 10 caracteres, es el apellido, cada carácter Unicode ocupa 2 bytes, por tanto el apellido ocupa 20 bytes.

- Un tipo entero que es el departamento, ocupa 4 bytes.
- Un tipo Double que es el salario, ocupa 8 bytes.

Tamaño de otros tipos: short (2 bytes), byte (1 byte), long (8 bytes), boolean (1bit), float (4 bytes), etc.
El fichero se abre en modo "rw" para lectura y escritura:

EscribirFichAleatorio

```
package accesoAleatorio;

import java.io.*;
public class EscribirFichAleatorio {
  public static void main(String[] args) throws IOException {
   File fichero = new File ("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\accesoAleatorio\\
AleatorioEmple.dat");
   //declara el fichero de acceso aleatorio
   RandomAccessFile file = new RandomAccessFile(fichero, "rw");
   //arrays con los datos
   String apellido[] = {"FERNANDEZ","GIL","LOPEZ","RAMOS",
                        "SEVILLA","CASILLA", "REY"};//apellidos
   int dep[] = {10, 20, 10, 10, 30, 30, 20};       //departamentos
   Double salario[]={1000.45, 2400.60, 3000.0, 1500.56,
                     2200.0, 1435.87, 2000.0};//salarios

   StringBuffer buffer = null;//buffer para almacenar apellido
   int n=apellido.length;//numero de elementos del array

   for (int i=0;i<n; i++){ //recorro los arrays
         file.writeInt(i+1); //uso i+1 para identificar empleado
     buffer = new StringBuffer( apellido[i] );
     buffer.setLength(10); //10 caracteres para el apellido
     file.writeChars(buffer.toString());//insertar apellido
         file.writeInt(dep[i]);       //insertar departamento
         file.writeDouble(salario[i]);//insertar salario
   }
   file.close();  //cerrar fichero
   }
}
```

El siguiente ejemplo toma el fichero anterior y visualiza todos los registros. El posicionamiento para empezar a recorrer los registros empieza en 0, para recuperar los siguientes registros hay que sumar 36 (tamaño del registro) a la variable utilizada para el posicionamiento:

LeerFichAleatorio

```
package accesoAleatorio;

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

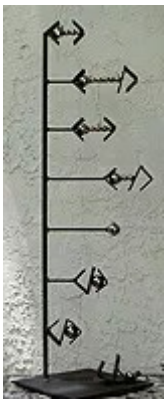
```
   File fichero = new File ("C:\\TMP\\DAM\\AD\\AD\\UD2\\src\\accesoAleatorio\\
AleatorioEmple.dat");
   //declara el fichero de acceso aleatorio
   RandomAccessFile file = new RandomAccessFile(fichero, "r");
   //
   int  id, dep, posicion;
   Double salario;
   char apellido[] = new char[10], aux;

   posicion = 0;  //para situarnos al principio

        while (file.getFilePointer() != file.length()) {
        file.seek(posicion); //nos posicionamos en posicion
        id = file.readInt();   // obtengo id de empleado

      //recorro uno a uno los caracteres del apellido
      for (int i = 0; i < apellido.length; i++) {
         aux = file.readChar();
         apellido[i] = aux;    //los voy guardando en el array
      }

      //convierto a String el array
      String apellidos = new String(apellido);
        dep = file.readInt();        //obtengo dep
        salario = file.readDouble(); //obtengo salario

        if(id >0)
        System.out.printf("ID: %s, Apellido: %s, Departamento: %d, Salario: %.2f
%n",
                id,   apellidos.trim(), dep, salario);

      //me posiciono para el sig empleado, cada empleado ocupa 36 bytes
      posicion= posicion + 36;

   }//fin bucle while

   file.close();  //cerrar fichero
   }
}
```

Para consultar un empleado determinado no es necesario recorrer todos los registros del fichero, conociendo su identificador podemos acceder a la posición que ocupa dentro del mismo y obtener sus datos

5.- TRABAJO CON FICHEROS XML: ANALIZADORES SINTÁCTICOS (parser) Y VINCULACIÓN (binding).

Caso práctico

María y **Juan** están trabajando en un proyecto común, una aplicación para "Farmacia Manolín de Benidorm". Hay una parte de la aplicación que no hicieron ellos, sino el hermano del farmacéutico, que tiene ciertos conocimientos, por su interés autodidacta de la informática. Parece ser que Arturín, el hermano de Manolín, utilizó ficheros XML para guardar información de la aplicación. Juan y María están contemplando la posibilidad de aprovechar lo que hay desarrollado, estudiando la aplicación y viendo si podrían analizar los ficheros XML que ya existen para obtener la información que desean.



Probablemente hayas estudiado ya XML, bien porque hayas cursado el módulo **Lenguajes de marcas y sistemas de gestión de información**, o bien porque lo conozcas por tu cuenta. Si no conoces XML, te recomendamos que te familiarices con él, hay múltiples tutoriales y cursos en Internet.

El metalenguaje XML se crea para evitar problemas de interoperabilidad entre plataformas y redes. Con él se consigue un soporte estándar para el intercambio de datos: no sólo los datos están en un formato estándar sino también la forma de acceder a ellos. Y entre las ventajas de su uso destacamos:

- **Facilita el intercambio de información** entre distintas aplicaciones ya que se basa en estándares aceptados.
- **Proporciona una visión estructurada de la información**, lo que permite su posterior tratamiento de forma local.

5.1.- Conceptos previos.

¿Cómo se trabaja con datos XML desde el punto de vista del desarrollador de aplicaciones?

Una aplicación que consume información XML debe:

- Leer un fichero de texto codificado según dicho estándar.
- Cargar la información en memoria y, desde allí...
- Procesar esos datos para obtener unos resultados (que posiblemente también almacenará de forma persistente en otro fichero XML).

El proceso anterior se enmarca dentro de lo que se conoce en informática como **"parsing"** o **análisis léxico-sintáctico**, y los programas que lo llevan a cabo se denominan "parsers" o

analizadores (léxico-sintácticos). Más específicamente podemos decir que el "parsing XML" es el proceso mediante el cual se lee y se analiza un documento XML para comprobar que está bien formado para, posteriormente, pasar el contenido de ese documento a una aplicación cliente que necesite consumir dicha información.

Algunos de los procesadores más empleados son: **DOM**: Modelo de Objetos de Documento y **SAX**: API Simple para XML.

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 (que son aquellos que no tienen descendientes).
SAX: un procesador que utilice este planteamiento lee un fichero XML de forma secuencial y produce una secuencia de eventos (comienzo/fin del documento, comienzo/fin de una etiqueta, etcétera) en función de los resultados de la lectura. Cada evento invoca a un método definido por el programador.

5.2.- Ficheros XML con DOM

Para poder trabajar con DOM en Java necesitamos las clases e interfaces que componen el paquete *org.w3c.dom* (contenido en el JSDK) y el paquete *javax.xml.parsers* del API estándar de Java que proporciona un par de clases abstractas que toda implementación DOM para Java debe extender.

Los programas Java que utilicen DOM necesitan estas interfaces:

- **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.
- **Node.** Representa a cualquier nodo del documento.
- **NodeList**. Contiene una lista con los nodos hijos de un nodo.
- **Attr**. Permite acceder a los atributos de un nodo.
- **Text.** Son los datos carácter de un elemento.
- **CharacterData.** Representa a los datos carácter presentes en el documento.
- **DocumentType.** Proporciona información contenida en la etiqueta <!DOCTYPE>.

5.2.1.- Creación de un fichero XML I

A continuación vamos a crear un fichero XML, a partir del fichero aleatorio de empleados creado en el epígrafe anterior. Lo primero que hemos de hacer es importar los paquetes necesarios:

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

import javax.xml.transform.dom.*;
import javax.xml.transíorm.stream.*;
import java.io.*;

A continuación creamos una instancia de *DocumentBuilderFactory* para construir el parser (se debe encerrar entre try-cath porque se puede producir la excepción *ParserConfigurationException*:

DocumentBuilderFactory factory = DocumentBuilderFactory.newlnstance();
Try{
DocumentBuilder builder = factory.newDocumentBuilder();

Creamos un documento vacío de nombre *document* con el nodo raíz de nombre *Empleados* y asignamos la versión del XML:

DOMImplementation implementation = builder.getDOMImplementation();
Document document = implementation.createDocument(nuil, "Empleados", nuil);
document.setXmlVersion("1.0") ; // asignamos la versión de nuestro XML

El siguiente paso sería recorrer el fichero con los datos de los empleados y por cada registro crear un nodo empleado con 4 hijos (id, apellido, dep y salario). Cada nodo hijo tendrá su valor (por ejemplo: 1, FERNANDEZ, 10, 1000.45). Para crear un elemento usamos el método createElement(String) llevando como parámetro el nombre que se pone entre las etiquetas menor que y mayor que. El siguiente código crea y añade el nodo <empleado> al documento:

Element raiz = document.createElement("empleado"); // creamos el nodo empleado
document.getDocumentElement().appendChild(raiz); //lo pegamos a la raiz del doc

A continuación se añaden los hijos de ese nodo (raiz), estos se añaden en la función *CrearElemento():*

CrearElemento("id",Integer.toStríng(id), raiz, document); //añadir ID
CrearElemento("apellido",apellidos,trím(), raíz, document); //añadir APELLIDO
CrearEIemento("dep",Integer.toStríng(dep), raíz, document); //añadir DEP
CrearElemento("salario",Double.toString(salario), raiz, document); //Añadir SALARIO

Como se puede ver la función recibe el nombre del nodo hijo (id, apellido, dep o salario) y sus textos o valores que tienen que estar en formato String (1, FERNANDEZ, 10,1000.45), el nodo al que se va a añadir (raiz) y el documento (document). Para crear el nodo hijo (<id> o <apellido> o <dep> o <salario>) se escribe:

Element elem = document.createElement(datoEmple); //creamos un hijo

Para añadir su valor o su texto se usa el método *createTextNode(String*):

Text text = document.createTextNode(valor); //damos valor

A continuación se añade el nodo hijo a la raíz (empleado) y su texto o valor al nodo hijo:

raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
elem.appendChild(text); //pegamos el valor

Al final se generaría algo similar a esto por cada empleado:

<empleado><id>1</id><apellido>FERNANDEZ</apellido><dep>10</dep><salario>1000.45< /salario></empleado>

5.2.2.- Creación de un fichero XML II

La función es la siguiente:

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
}

En los últimos pasos 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")); //fíchero XML

Se obtiene un TransfomerFactory:

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

Se realiza la transformación del documento a fichero.

transformer.transform(source, result);

Para mostrar el documento por pantalla, podemos especificar como resultado el canal de salida System.out:

Result console = new StreamResult(System.out) ; transformer.transform(source, console);

El código completo es el siguiente:

CrearEmpleadoXml

```
package xmlfiles;

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 class CrearEmpleadoXml {
 public static void main(String args[]) throws IOException{
   File fichero = new File ("C:\\TMP\\accesoAleatorio\\AleatorioEmple.dat");
   RandomAccessFile file = new RandomAccessFile(fichero, "r");
```

```
   int  id, dep, posicion=0; //para situarnos al principio del fichero
   Double salario;
   char apellido[] = new char[10], aux;

   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

   try{
     DocumentBuilder builder = factory.newDocumentBuilder();
     DOMImplementation implementation = builder.getDOMImplementation();
     Document document = implementation.createDocument(null, "Empleados", null);
     document.setXmlVersion("1.0");

     for(;;) {
         file.seek(posicion); //nos posicionamos
         id=file.readInt();   // obtengo id de empleado
     for (int i = 0; i < apellido.length; i++) {
         aux = file.readChar();
         apellido[i] = aux;
     }
     String apellidos = new String(apellido);
     dep = file.readInt();
         salario = file.readDouble();

         if(id>0) { //id validos a partir de 1
           Element raiz = document.createElement("empleado"); //nodo empleado
       document.getDocumentElement().appendChild(raiz);
       //añadir ID
       CrearElemento("id",Integer.toString(id), raiz, document);
       //Apellido
       CrearElemento("apellido",apellidos.trim(), raiz, document);
       //añadir DEP
       CrearElemento("dep",Integer.toString(dep), raiz, document);
       //añadir salario
       CrearElemento("salario",Double.toString(salario), raiz, document);
         }
         posicion= posicion + 36; // me posiciono para el sig empleado
         if (file.getFilePointer() == file.length()) break;

     }//fin del bucle infinito for que recorre el fichero

     Source source = new DOMSource(document);
     Result result =
            new StreamResult(new java.io.File("C:\\TMP\\Empleados.xml"));
     Transformer transformer =
            TransformerFactory.newInstance().newTransformer();
     transformer.transform(source, result);

    }catch(Exception e){ System.err.println("Error: "+e); }

    file.close();  //cerrar fichero
 }//fin de main

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

```
    elem.appendChild(text); //pegamos el valor
 }
}//fin de la clase
```

5.2.3.- Lectura de documento XML

Para leer un documento XML, creamos una instancia de *DocumentBuilderFactory* para construir el parser y cargamos el documento con el método *parse()*:

Document document = builder.parse(new File("Empleados.xml"));

Obtenemos la lista de nodos con nombre empleado de todo el documento:

NodeList empleados = document.getElementsByTagName("empleado");

Se realiza un bucle para recorrer esta lista de nodos. Por cada nodo se obtienen sus etiquetas y sus valores llamando a la función *getNodo()*. El código es el siguiente:

LecturaEmpleadoXml

```
package xmlfiles;

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

public class LecturaEmpleadoXml {
         public static void main(String[] args) {

          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

          try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.parse(new File("C:\\TMP\\
Empleados.xml"));
            document.getDocumentElement().normalize();

            System.out.printf("Elemento raiz: %s %n",
document.getDocumentElement().getNodeName());
            //crea una lista con todos los nodos empleado
            NodeList empleados = document.getElementsByTagName("empleado");
            System.out.printf("Nodos empleado a recorrer: %d %n",
empleados.getLength());
            //recorrer la lista
            for (int i = 0; i < empleados.getLength(); i ++) {
                Node emple = empleados.item(i); //obtener un nodo empleado
                if (emple.getNodeType() == Node.ELEMENT_NODE) {//tipo de nodo
                        //obtener los elementos del nodo
                        Element elemento = (Element) emple;
                        System.out.printf("ID = %s %n",
elemento.getElementsByTagName("id").item(0).getTextContent());
                        System.out.printf(" * Apellido = %s %n",
elemento.getElementsByTagName("apellido").item(0).getTextContent());
```

```
                        System.out.printf(" * Departamento = %s %n",
elemento.getElementsByTagName("dep").item(0).getTextContent());
                        System.out.printf(" * Salario = %s %n",
elemento.getElementsByTagName("salario").item(0).getTextContent());
                    }
                }
              } catch (Exception e)
                {e.printStackTrace();}

            }//fin de main
}//fin de la clase
```

Anexo I.- Código de crear un fichero.

volver

```
try {
         // Creamos el objeto que encapsula el fichero
         File fichero = new File("c:\\prufba\\miFichero.txt");
         // A partir del objeto File creamos el fichero físicamente
         if (fichero.createNewFile())
                  System.out.println("El fichero se ha creado correctamente");
         else
                  System.out.println("No ha podido ser creado el fichero");
} catch (Exception ioe) {
         ioe.getMessage();
}
```

Anexo II.- Código de crear un directorio.

volver

```
try {
// Declaración de variables
        String directorio = "C:\\prueba";
        String varios = "carpeta1/carpeta2/carpeta3";

// Crear un directorio
        boolean exito = (new File(directorio)).mkdir();
        if (exito)
           System.out.println("Directorio: " + directorio + " creado");
// Crear varios directorios
       exito = (new File(varios)).mkdirs();
       if (exito)
           System.out.println("Directorios: " + varios + " creados");
}catch (Exception e){
       System.err.println("Error: " + e.getMessage());
}
```