Acceso a Datos

UD1. Manejo de Ficheros

1. Introducción

- Los programas usan variables para almacenar **información**: los datos de entrada, los resultados calculados y valores generados a lo largo del cálculo. Toda esa información es **efímera**: cuando se acaba el programa, todo desaparece. Pero, para muchas aplicaciones, es importante poder almacenar datos de manera **permanente**.
- Cuando se desea guardar información más allá del tiempo de ejecución de un programa, lo habitual es organizar esa información en uno o varios **ficheros** almacenados en algún soporte de almacenamiento persistente. Otras posibilidades como el uso de **bases de datos** utilizan archivos como soporte para el almacenamiento de la información.

1. Introducción

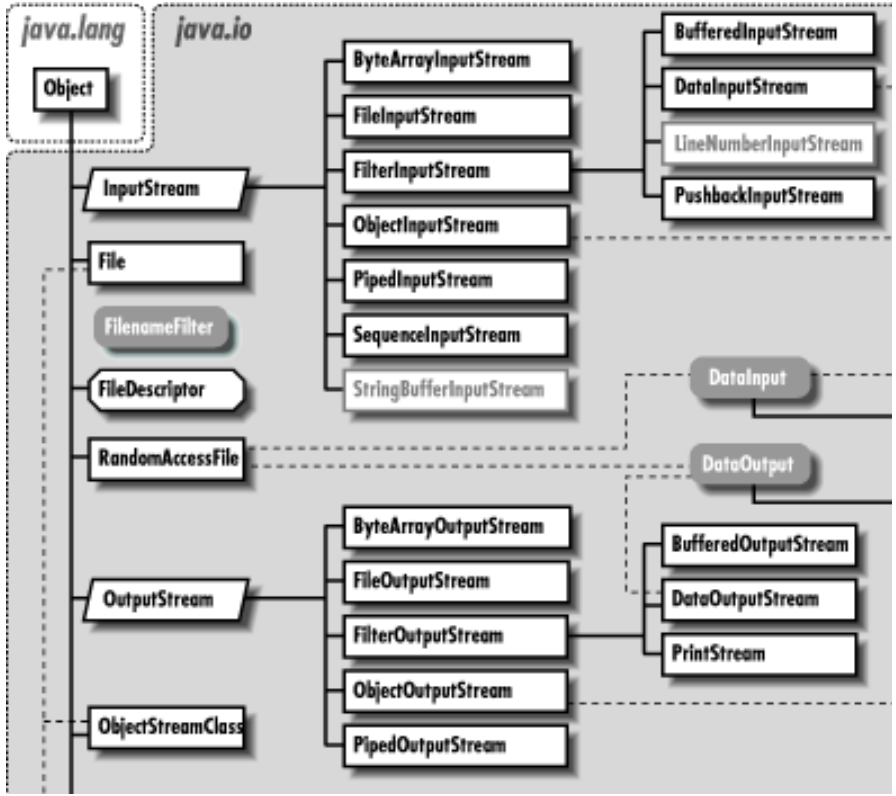
Tipos de ficheros.

- Podemos diferenciar varios tipos de archivos en función de los siguientes criterios:
  - **En función del contenido**: archivos de **texto** (de caracteres) y archivos **binarios** (de bytes).
    - Aunque en el fondo ambos tipos acaben almacenando conjuntos de bits en el archivo, en el caso de los ficheros de texto solo encontraremos caracteres que podrán visualizarse usando cualquier editor de texto. Por el contrario, los ficheros binarios almacenan conjuntos de bytes que pueden representar cualquier cosa: número, imágenes, sonido, etc.
  - **En función del modo de acceso**: archivos **secuenciales** y de a**cceso directo** (o de acceso aleatorio).
    - En el modo secuencial la información del archivo es una secuencia de bytes de forma que para poder acceder al byte i-ésimo se ha de haber accedido previamente a todos los bytes anteriores. Por el contrario, el modo de acceso directo nos permite acceder directamente a la información del byte i-ésimo sin necesidad de recorrerlos todos.

1. Introducción

- En Java, disponemos de varias clases dentro de los paquetes **java.io** y **java.io.file** para manejar los distintos tipos de archivos que existen.

- Veamos las clases más importantes, en primer lugar, del paquete **java.io**.



2. La clase File

- La clase **File** proporciona un conjunto de utilidades relacionadas con los ficheros que nos van a proporcionar información acerca de los mismos:
  - Su nombre
  - Sus atributos
  - Sus directorios
  - Etc.

- Un objeto de la clase **File** puede representar:
  - Un archivo en particular.
  - Una serie de archivos ubicados en un directorio.
  - Un nuevo directo para ser creado.

2. La clase File

- Para crear un objeto de la clase File podemos usar cualquiera de los siguientes constructores:

a) `File(String rutaCompleta)`

```
File fichero= new File("./resources/textFiles/file001.txt");
```

b) `File(String rutaDirectorio, String nombreArchivo)`

```
File fichero2= new File("./resources/textFiles","file001.txt" );
```

2. La clase File

- Dependiendo de la familia SO operativos para los que estemos programando debemos utilizar una nomenclatura u otra para las rutas de ficheros:
  - Windows →

```
File ficheroWindows= new File("C:\\Users\\Admin\\DocumentsandFiles\\file001.txt");
```

  - Unix like (Linux, freeBSD, Mac OS, Android, IOS, etc)

```
File ficheroNix= new File("/Users/Admin/DocumentsandFiles/file001.txt");
```

2. La clase File

Métodos de la clase

Método	Función
`String[] list()`	Devuelve un array de String con los nombres de los ficheros y directorios asociados al objeto File
`File[] listFiles()`	Devuelve un array de objetos File conteniendo los ficheros que estén dentro del directorio representado por el objeto FIle
`String getName()`	Devuelve el nombre del fichero o directorio
`String getPath()`	Devuelve la ruta del fichero o directorio
`String getAbsolutePath()`	Devuelve la ruta absoluta al fichero o directorio
`boolean exists()`	Devuelve true si el fichero o directorio existe
`boolean canWrite()`	Devuelve true si el fichero se puede escribir

2. La clase File

Métodos de la clase

Método	Función
`boolean canRead()`	Devuelve true si el fichero o directorio se puede leer.
`boolean isFile()`	Devuelve true si el objeto File corresponde a un fichero
`boolean isDirectory()`	Devuelve true si el objeto File corresponde con un directorio.
`long length()`	Devuelve el tamaño del fichero en bytes
`boolean mkdir()`	Crea un directorio con el nombre del objeto File (En caso de que no exista)
`boolean renameFileTo(File newName)`	Renombre el fichero representado por el objeto File, asignándole el nombre de este.
`boolean delete()`	Borra el fichero o directorio
`boolean createNewFile()`	Crea un fichero vacío asociado al objeto File (Solo se creará en caso de que no exista)
`String getParent()`	Devuelve el nombre del directorio padre.

2. La clase File

- Estos son tan solo algunos de los métodos de la clase File. Para más información consultar la documentación de Oracle en https://docs.oracle.com/javase/7/docs/api/java/io/File.html

2. La clase File

- Ejemplo de uso de la clase File

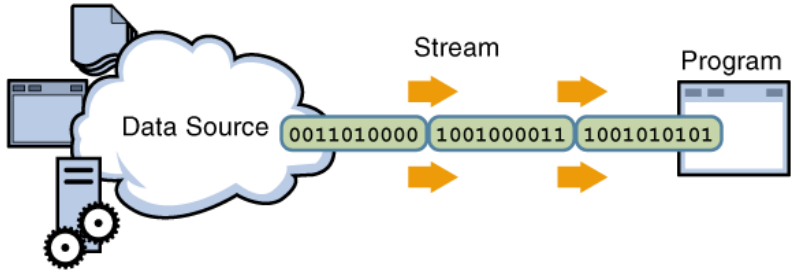
Hemos creado un fichero llamado file001.txt en la carpeta src de nuestro proyecto.

1. Creamos una cadena para la ruta
2. Creamos una cadena para el nombre del fichero
3. Creamos el fichero
4. Mostramos algunas de sus propiedades
5. Comprobamos que el fichero existe
   a. En caso de existir mostramos sus permisos y su tamaño
   b. En caso de no existir se indica por pantalla.

```
public class Main {

    public static void main(String[] args) {

        String ruta = "src/";
        String nombre = "file001.txt";
        File f = new File(ruta+nombre);

        System.out.println("Nombre: "+f.getName());
        System.out.println("Ruta: "+f.getAbsolutePath());
        System.out.println("Directorio padre: "+f.getParent());
        if (f.exists()) {
            System.out.println("¡El fichero existe!");
            System.out.println("Permisos(rwx)=>"+f.canRead()+f.canWrite()+f.canExecute());
            System.out.println("Longitud del fichero: "+f.length()+" bytes");
        }
        else {
            System.out.println("El fichero no existe");
        }



    }

}
```

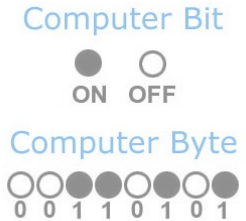
3. Flujos o Streams

- El sistema de I/O de Java dispone de diferentes clases definidas en el paquete java.io
- Emplea la abstracción del flujo (Stream) para tratar la comunicación entre una fuente y un destino.
  - Dicha fuente puede ser:
    - El disco duro.
    - La memoria principal.
    - Una ubicación en red.
    - Otro programa.
- Cualquier programa que necesite obtener o enviar información a cualquier fuente necesita hacer uso de un stream.
- Por defecto, la información de un stream se escribirá y leerá en serie.



3. Flujos o Streams

- Se definen dos tipos de flujos:
  - **Flujos de Bytes (8 bits)** .
    - Realiza operaciones de I/O de bytes.
    - Orientado a operaciones con datos binarios
    - Todas las clases de flujos de bytes descienden de InputStream y OutputStream.
  - **Flujos de Caracteres (16 bits)**
    - Realizan operaciones de I/O de caracteres.
    - Definido en las clases Reader y Writer
    - Soporta caracteres Unicode de 16 bits.



3. Flujos o Streams

Flujos de Bytes (Byte Streams)

- **InputStream** → Representa las clases que producen entradas desde distintas fuentes: un Array de Bytes, un objeto String, un fichero, una tubería, una secuencia de flujos, una conexión de red, etc.
- La siguiente tabla muestra las diferentes clases que heredan de InputStream:

Clase	Función
ByteArrayInputStream	Permite usar el espacio de almacenamiento intermedio de memoria.
StringBufferInputStream	Convierte un String en un InputStream
FileInputStream	Se emplea para leer datos de un fichero.
PipedInputStream	Implementa el concepto de tubería
FilterInputStream	Proporciona funcionalidad adicional a otras streams
SequenceInputStream	Concatena dos o más objetos InputStream

3. Flujos o Streams

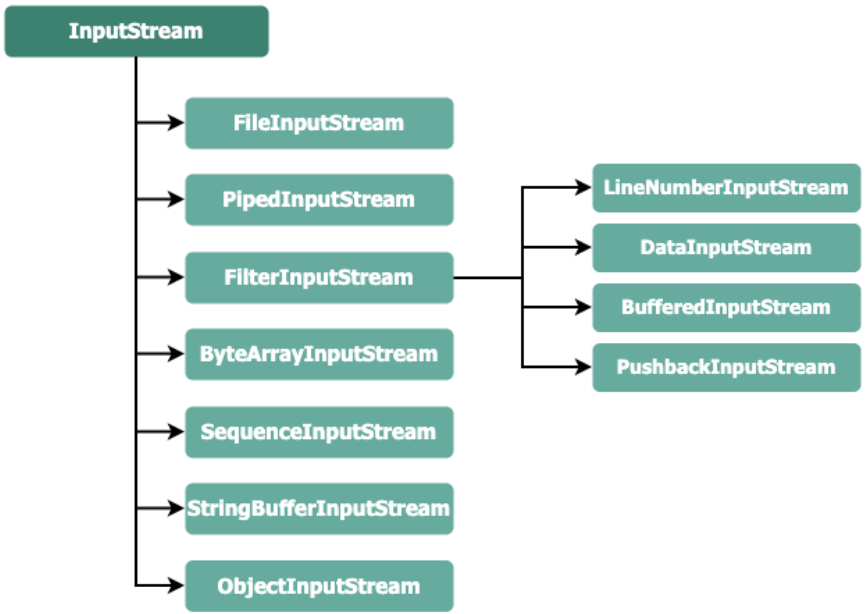
Flujos de Bytes (Byte Streams)

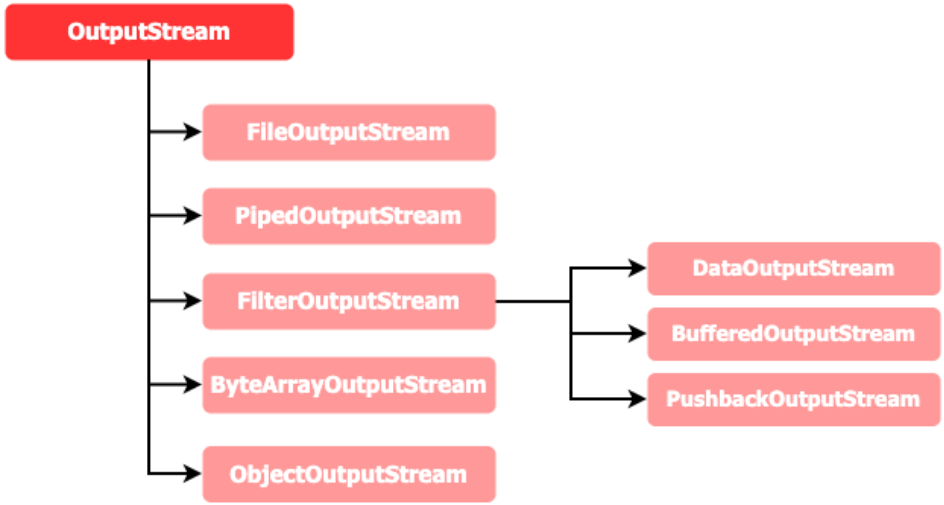
- **OutputStream** → Representa los flujos de salida de un programa.
- La siguiente tabla muestra las diferentes clases que heredan de OutputStream:

Clase	Función
ByteArrayOutputStream	Crea un espacio de almacenamiento en memoria. Todos los datos que se envían a este flujo se almacenan en él.
FileOutputStream	Se emplea para escribir datos en un fichero.
PipedOutPutStream	Cualquier información escrita en un objeto de esta clase deberá ser usada como entrada de un PipedInputStream asociado a él. Implementa el concepto de tubería.
FilterOutputStream	Proporciona funcionalidad adicional a otras OutputStreams

3. Flujos o Streams

Flujos de Bytes (Byte Streams)





3. Flujos o Streams

Flujos de Caracteres (Character Streams)

- Las clases **Reader** y **Writer** manejan flujos de caracteres Unicode.
- Es habitual usar estos flujos en combinación con los flujos que manejan Bytes.
  - Para ello contamos con las llamadas “clases puente”, que convierten de byteStream a characterStream
    - **InputStreamReader** → Convierte un InputStream en un Reader.
    - **OutputStreamWriter** → Convierte un OutputStream en un Writer.

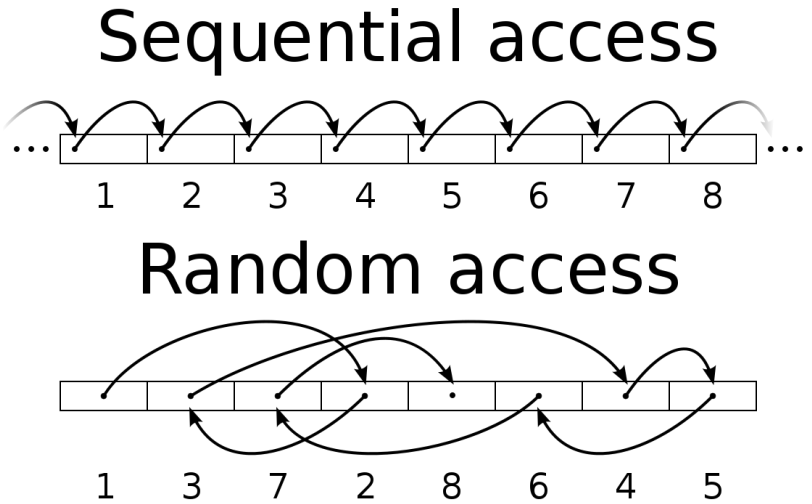
- Las clases de flujos de caracteres más importantes:
  - **FileReader y FileWriter** → Lectura y escritura de caracteres en ficheros.
  - **CharArrayReader y CharArrayWriter** → Lectura y escritura de Array de caracteres.
  - **BufferedReader y BufferedWriter** → Se emplean para utilizar un buffer intermedio entre el programa y el fichero de origen o destino.

4. Formas de acceso a un fichero

- De forma genérica podemos decir que existen dos formas básicas de acceso a un fichero:
  - **Acceso Secuencial.**
    - Los datos se leen y escriben en orden, de forma similar a como se hace en una antigua cinta de video.
    - Para acceder a un dato es necesario leer primero todos los anteriores.
    - Solo se pueden insertar nuevos datos el final del fichero.
  - **Acceso directo o aleatorio.**
    - Permite el acceso directo a un dato en concreto, esto es, se puede acceder a la información en cualquier orden.
    - Los datos se almacenan en registros de tamaño conocido, por tanto podemos inferir la posición de un dato o registro y acceder a él para leerlo o modificarlo.

4. Formas de acceso a un fichero

- En Java, el acceso secuencial puede ser:
  - **Binario** → FileInputStream y FileOutputStream
  - **Caracteres** → FileReader y FileWriter
- Para el acceso aleatorio usaremos RandomAccessFile. (Se estudiará más adelante)



5. Operaciones sobre ficheros

- Las operaciones que podemos realizar sobre cualquier fichero (Independientemente de su tipo):
  - Creación de fichero
  - Apertura del fichero
  - Cierre del fichero
  - Lectura de datos del fichero
  - Escritura de datos en el fichero

- Una vez abierto, las operaciones que podremos realizar se clasifican en:
  - Altas
  - Bajas
  - Modificaciones
  - Consultas
    - Cualquier otra operación cómo búsquedas u ordenaciones se componen de una combinación de las anteriores.

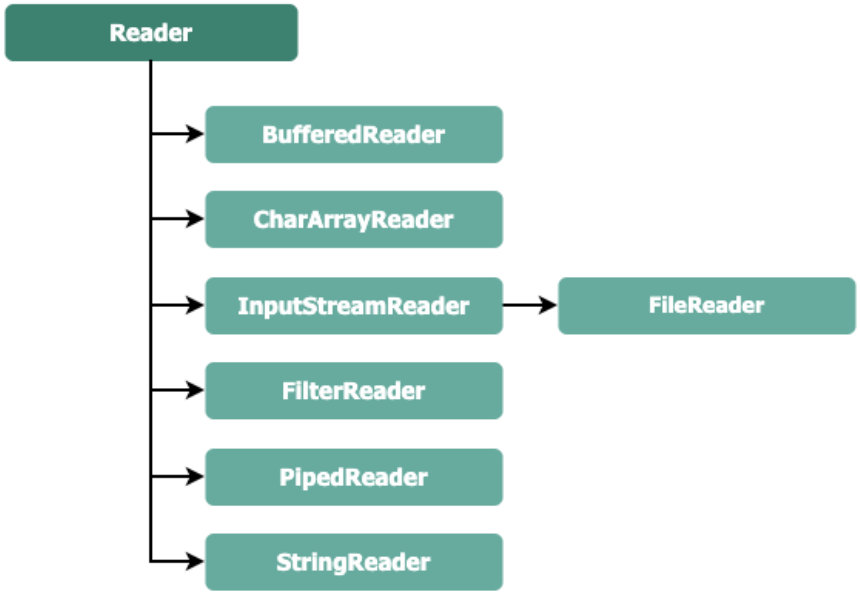
5. Operaciones sobre ficheros

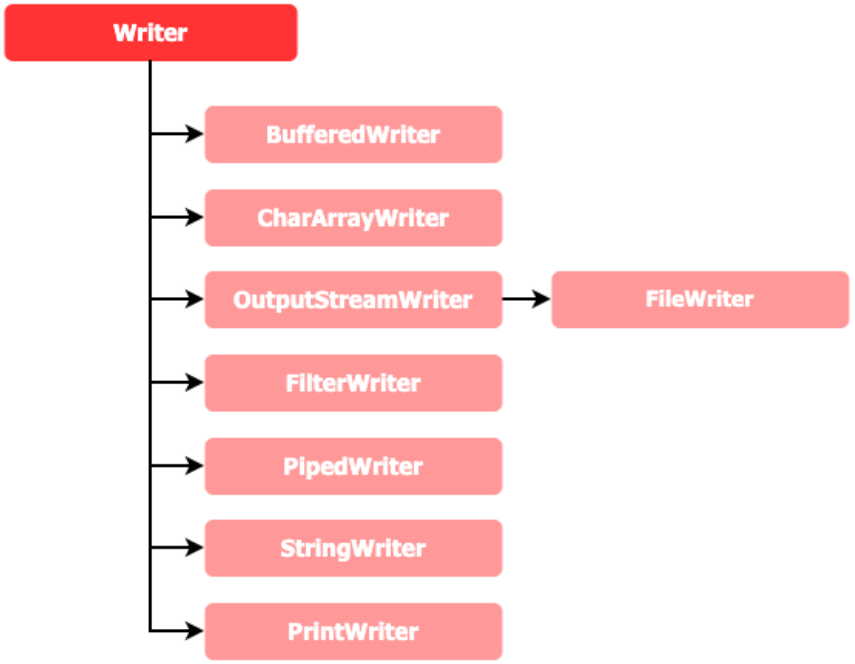
Operaciones sobre ficheros secuenciales

- **Consultas** → Para consultar un determinado registro es necesario comenzar la lectura desde el primer registro y continuar leyendo hasta llegar al deseado.
- **Altas** → Las nuevas altas deben hacerse al final del fichero.
- **Bajas** → Para dar de baja un registro es necesario:
  - Copiar todos los registros menos el que deseamos borrar a un fichero auxiliar
  - Borrar completamente el fichero de origen
  - Volcar el contenido del fichero auxiliar al fichero original.
- **Modificaciones** → Se realiza a través de un fichero auxiliar, de forma similar a como se realizar las bajas.

3. Flujos o Streams

Flujos de Caracteres (Character Streams)





5. Operaciones sobre ficheros

Operaciones sobre ficheros aleatorios

- En los ficheros aleatorios, para poder llevar a cabo cualquier operación, antes es necesario localizar la posición del registro sobre el cual queremos operar.
  - Para posicionarnos en un registro es necesario aplicar una función de conversión, que de forma general estará relacionada con el tamaño del registro y con la clave del mismo.
    - Ejemplo:
      - Contamos con un fichero que contiene información de alumnos 3 campos (ID, nombre, apellido). Usaremos el ID como campo clave del mismo estableciendolo como campo autonumérico; de esta forma, para localizar a un alumno cuyo ID sea *N* deberemos acceder a la posición *tamaño*(N-1)*

5. Operaciones sobre ficheros

Operaciones sobre ficheros aleatorios

- Consulta → Aplicamos la función de conversión para obtener la dirección del registro y lo leemos.
- Alta → Aplicamos la función de conversión para obtener la dirección del registro y escribimos el registro en la posición obtenida.
- Bajas → Aplicamos la función de conversión para obtener la dirección del registro y:
  - Bajas lógicas → Marcamos el registro como “eliminado” mediante un campo booleano, aunque realmente lo mantenemos en la estructura.
  - Bajas físicas → Eliminamos el registro del fichero.
- Modificaciones → Aplicamos la función de conversión para obtener la dirección del registro y modificamos los valores deseados.

5. Operaciones sobre ficheros

Operaciones sobre ficheros aleatorios

- **Consulta** → Aplicamos la función de conversión para obtener la dirección del registro y lo leemos.
- **Alta** → Aplicamos la función de conversión para obtener la dirección del registro y escribimos el registro en la posición obtenida.
- **Bajas** → Aplicamos la función de conversión para obtener la dirección del registro y:
  - Bajas lógicas → Marcamos el registro como “eliminado” mediante un campo booleano, aunque realmente lo mantenemos en la estructura.
  - Bajas físicas → Eliminamos el registro del fichero.
- **Modificaciones** → Aplicamos la función de conversión para obtener la dirección del registro y modificamos los valores deseados.

6. Ficheros de texto en Java

- Para trabajar con ficheros de texto usaremos:
  - **FileReader** para leer.
  - **FileWriter** para escribir.

- Siempre que trabajemos con estas clases debemos realizar una correcta gestión de excepciones ya que pueden producirse
  - **FileNotFoundException** → En caso de no encontrar el fichero.
  - **IOException** → Cuando se produce algún tipo de error de escritura.

6. Ficheros de texto en Java

FileReader

- La siguiente tabla muestra los métodos empleados por la clase FileReader para leer ficheros, en caso de encontrarnos al final del fichero (EOF) todos ellos devuelven -1.

Método	Función
int read()	Lee un carácter y lo devuelve.
int read(char[] buf)	Lee hasta buf.length caracteres de datos. Los caracteres leídos se almacenan en buf
int read(char buf, int desplazamiento, int n)	Lee hasta n caracteres de datos a partir de la posición de desplazamiento.

6. Ficheros de texto en Java

FileReader

- En Java para abrir y leer un fichero de texto debemos
  - Crear una instancia de la clase File
  - Crear un flujo de entrada con la clase FileReader
  - Realizamos las operaciones de lectura pertinentes.
  - Cerramos el fichero con el método .close()

```
//Leemos el fichero de 20 en 20
char[] buffer= new char[20];
while((caracter=fr.read(buffer))!=-1) {
    System.out.print(buffer);
}
```

```
//Creamos el fichero
File f= new File("myFiles/SampleFile.txt");


try {
    //Creamos el stream de entrada
    FileReader fr= new FileReader(f);
    int caracter;

    //Leemos el fichero caracter a caracter
    while((caracter=fr.read())!=-1) {
        System.out.print((char)caracter);
    }
    //Cerramos el fichero
    fr.close();

} catch (FileNotFoundException e) {

    e.printStackTrace();
} catch (IOException e) {

    e.printStackTrace();
}
```

6. Ficheros de texto en Java

FileWriter

- La siguiente tabla muestra los métodos empleados por la clase FileWriter para escribir en ficheros:

Método	Función
void write(int c)	Escribe un carácter
void write(char[] buf)	Escribe un array de caracteres
void write(char buf, int desplazamiento, int n)	Escribe n caracteres de datos a partir de la posición de desplazamiento.
void write(String str)	Escribe una cena de caracteres
void append(char c)	Añade un carácter al final del fichero.

6. Ficheros de texto en Java

FileWriter

- En Java, escribir en un fichero de texto:
  - Crear una instancia de la clase File
  - Crear un flujo de salida con FileWriter
  - Realizamos las operaciones de escritura
  - Cerramos el fichero con el método .close()

```
public static void main(String[] args) {
    //Creamos el fichero
    File f= new File("myFiles/newFile001.txt");
    try {
        //Creamos un stream de salida
        FileWriter fw= new FileWriter(f);
        String cadena="Esta es mi primera escritura en disco.";
        //Escribimos en el fichero
        fw.write(cadena);
        //Cerramos el fichero
        fw.close();
    } catch (IOException e) {

        e.printStackTrace();
    }

}
```

6. Ficheros de texto en Java

FileWriter

En el siguiente ejemplo vamos a escribir todo el contenido de un array dentro de un fichero.

¿Con qué formato crees que se escribirá?

```
File f= new File("myFiles/newFile002.txt");
try {

    FileWriter fw= new FileWriter(f);
    String provincias[]={
        "Albacete", "Avila", "Badajoz",
        "Caceres", "Huelva", "Jaén", "Toledo"
    };
    for (int i = 0; i < provincias.length; i++) {
        fw.write(provincias[i]);
    }

    //Cerramos el fichero
    fw.close();
} catch (IOException e) {

    e.printStackTrace();
}
```

6. Ficheros de texto en Java

FileWriter

- Debemos tener en cuenta que si el fichero ya contenía información anteriormente, al realizar la nuevas escritura sobrescribiremos su contenido.
  - Si nuestra intención es agregar nueva información debemos crear el stream de salida de la siguiente manera:

```
FileWriter fw= new FileWriter(f, true);
```

El segundo parámetro del constructor es el parámetro de append:

true → Añade la información al final del fichero.

false → Borra el contenido anterior para introducir el nuevo.

6. Ficheros de texto en Java

BufferedReader

- Con la clase BufferedReader podemos hacer una gestión más avanzada de la lectura de fichero.
  - Con el método readLine() podremos leer una línea de nuestro documento, especialmente útil si el separador de registros es el carácter \n.
    - readLine() devuelve una String con la cadena leída o null si nos encontramos al final del fichero (EOF).
- Para crear un BufferedReader necesitamos
- partir de la clase FileReader. Esto es debido
- a que la clase que realmente gestiona el
- stream es FileReader. BufferedReader se
- vale de ella para ir haciendo una lectura
- con buffer.

```
public static void main(String[] args) {
    File f= new File("myFiles/newFile002.txt");
    try {
        BufferedReader br= new BufferedReader(new FileReader(f));
        String linea="";
        while((linea=br.readLine())!=null) {
            System.out.println(linea);          }
        br.close();

    } catch (IOException e) {

        e.printStackTrace();
    }
```

6. Ficheros de texto en Java

BufferedWriter

- Del mismo modo contamos con la clase BufferedWriter, que nos permite realizar escrituras línea a línea.
  - El método .write() escribe una línea en nuestro fichero.
  - El método .newLine() escribe un salto de línea.

```
File f= new File("myFiles/newFile003.txt");
try {
    BufferedWriter bw= new BufferedWriter(new FileWriter(f));
    String provincias[]={
            "Albacete", "Avila", "Badajoz",
            "Caceres", "Huelva", "Jaén", "Toledo"
        };
    for (int i = 0; i < provincias.length; i++) {
        bw.write(provincias[i]);
        bw.newLine();
    }

} catch (IOException e) {

    e.printStackTrace();
}
```

6. Ficheros de texto en Java

PrintWriter

- La clase PrintWriter posee los métodos **print(String)** y **println(String)** cuyo comportamiento se asemeja a los de System.out.
- Ambos reciben un String y lo escriben en el fichero. println, además, inserta un salto de línea.
- Para construir un PrintWriter, una vez más, necesitamos un FileWriter.

```
File f= new File("myFiles/newFile004.txt");
try {
    PrintWriter pw= new PrintWriter(new FileWriter(f));
    String provincias[]={
            "Albacete", "Avila", "Badajoz",
            "Caceres", "Huelva", "Jaén", "Toledo"
        };
    for (int i = 0; i < provincias.length; i++) {
        pw.println(provincias[i]);

    }
    pw.close();

} catch (IOException e) {

    e.printStackTrace();
}
```

7. Ficheros binarios en Java

- Los ficheros binarios almacenan secuencias de dígitos binarios que no son legibles por el usuario, a diferencia de lo que ocurría con los ficheros de texto plano.
- Ocupan menos espacio en disco.
- Para trabajar con ficheros binarios usaremos:
  - **FileInputStream** → Para lectura.
  - **FileOutputStream** → Para escritura.

7. Ficheros binarios en Java

FileInputStream

- Esta clase nos permite leer un fichero a través de los siguientes métodos:

Método	Función
int read()	Lee un byte y lo devuelve.
int read(byte[] buf)	Lee hasta buf.length bytes de datos. Los bytes leídos se almacenan en buffer
int read(byte buf, int desplazamiento, int n)	Lee hasta n byte de datos a partir de la posición de desplazamiento.

7. Ficheros binarios en Java

FileOutputStream

- Esta clase nos permite escribir en un fichero a través de los siguientes métodos:

Método	Función
void write(int b)	Escribe un byte
void write(byte[] buf)	Escribe un array de bytes
void write(byte buf, int desplazamiento, int n)	Escribe n bytes de datos a partir de la posición de desplazamiento.

7. Ficheros binarios en Java

FileOutputStream

- Igual que sucede con las clases de escritura de caracteres, FileOutputStream sobrescribre la información ya existente en el fichero.
- Para evitar que esto suceda debemos crear el objeto FileOutputStream en modo append (Añadir).
  - Esto se consigue añadiendo un segundo parámetro al constructor del objeto.

```
FileOutputStream fos = new FileOutputStream(f, true);
```

7. Ficheros binarios en Java

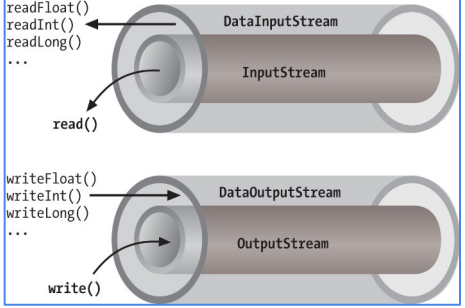
FileInputStream / FileOutputStream

En el siguiente ejemplo podemos ver como creamos un flujo de salida y lo usamos para insertar los 100 primeros números enteros dentro de un fichero.

Posteriormente, creamos un flujo de entrada y procedemos a leerlos.

```
File f= new File("./myFiles/Ejemplo1");
try {
    //Creamos un Stream de salida
    FileOutputStream fos = new FileOutputStream(f);
    //Escribimos el el fichero los 100 primeros números
    for(int i=1;i<=100;i++)
        fos.write(i);
    //Cerramos el stream de salida
    fos.close();
    //Creamos un flujo de entrada
    FileInputStream fis= new FileInputStream(f);
    //Visualizamos los datos del fichero.
    int b=0;
    while((b=fis.read())!=-1) {
        System.out.print(b+ " ");
    }
    //Cerramos el Stream de entrada
    fis.close();
} catch (FileNotFoundException e) {

    e.printStackTrace();
} catch (IOException e) {

    e.printStackTrace();
}
```

7. Ficheros binarios en Java



DataInputStream / DataOutputStream

- Con las clases FileInputStream y FileOutputStream podemos leer y escribir bytes respectivamente.
- Sin embargo, dependiendo del tipo de dato que queramos utilizar su codificación binaria será diferente.
- Es por ello que para la escritura de tipos primitivos emplearemos los métodos:
  - **DataInputStream** → Lectura
  - **DataOutputStream** → Escritura
- Estas clases actúan como un wrap de las clases FileInputStream y FileOutputStream dotandolas de la capacidad de trabajar con diferentes tipos de datos (Más allá de poder leer y escribir bit a bit).

7. Ficheros binarios en Java

DataInputStream / DataOutputStream

- La siguiente tabla muestra los principales métodos de lectura y escritura de las clases DataInputStream y DataOutputStream.

DataInputStream (Lectura)	DataOutputStream (Escritura)
boolean readBoolean();	void writeBoolean(boolean v);
byte readByte()	void writeByte(int v);
int readUnsignedInt()	void writeBytes(String s)
int readUnsignedShort();	void writeShort(short v)
short readShort();	void writeChars(String s)
char readChar();	void writeChar(char c);
int readInt();	void writeInt(int v);
long readLong();	void writeLong(long v);
float readFloat();	void writeFloat(float v);
double readDouble();	void writeDouble(double v);
String readUTF();	void writeUTF(String str);

7. Ficheros binarios en Java

DataInputStream / DataOutputStream

- Para crear un DataInputStream debemos partir de un FileInputStream como muestran los siguientes ejemplos de código.

```
File f= new File("./myFiles/Ejemplo2.dat");
FileInputStream fis= new FileInputStream(f);
DataInputStream dis= new DataInputStream(fis);
```

```
File f= new File("./myFiles/Ejemplo2.dat");
DataInputStream dis= new DataInputStream(new FileInputStream(f));
```

- Del mismo modo, para crear un DataOutputStream debemos partir de un FileOutputStream.

```
File f= new File("./myFiles/Ejemplo2.dat");
FileOutputStream fos = new FileOutputStream(f);
DataOutputStream dos= new DataOutputStream(fos);
```

```
File f= new File("./myFiles/Ejemplo2.dat");
DataOutputStream dos= new DataOutputStream(new FileOutputStream(f));
```

7. Ficheros binarios en Java

DataInputStream / DataOutputStream

- El siguiente ejemplo muestra cómo insertar datos en un fichero binario utilizando DataOutPutStream
  - Como puede verse, dependiendo del tipo de dato que queramos introducir emplearemos un método u otro.
  - Cómo siempre, al terminar la operación **CERRAMOS LOS FLUJOS.**

```
//Creamos el fichero y su flujo de salida
File f= new File("./myFiles/Ejemplo2.dat");
FileOutputStream fos = new FileOutputStream(f);
DataOutputStream dos= new DataOutputStream(fos);
//Definimos dos arrays que escribiremos en el fichero
String[] alumnos= {
"Alberto", "Ana","Josefina","Maribel", "Adriano", "Mariano"
};
int[] calificaciones= {1,2,3,4,5,6};

//Recorremos los arrays y los vamos escribiendo
for (int i = 0; i < alumnos.length; i++) {
    dos.writeUTF(alumnos[i]);
    dos.writeInt(calificaciones[i]);
}
//Creamos el flujo
dos.close();
fos.close();
```

7. Ficheros binarios en Java

DataInputStream/ DataOutputStream

- El siguiente ejemplo muestra cómo leer datos de un fichero binario utilizando DataInputStream
  - Como puede verse, dependiendo del tipo de dato que queramos leer emplearemos un método u otro.
  - Cómo siempre, al terminar la operación **CERRAMOS LOS FLUJOS.**

```
//Creamos el flujo de entrada
File f= new File("./myFiles/Ejemplo2.dat");
DataInputStream dis= new DataInputStream(new FileInputStream(f));
//Creamos un try para capturar el EOF
try {
    //Iniciamos un bucle infinito para recorrer todo el fichero
    do {
        System.out.println(dis.readUTF()+ " ha obtenido un "
    +dis.readInt());
    }while(true);
    /*
     * Cuando el fichero llegue al final (EOF) saltará la expceción
     * EOFException. La capturamos para poder cerrar el lujo una vez
     * recorrido todo el fichero.
     */
}catch (EOFException e) {
    //Cerramos el flujo
    dis.close();
}
```

7. Ficheros binarios en Java

Objetos en ficheros binarios

- Hasta ahora hemos visto que en los ficheros binarios es posible almacenar datos primitivos de forma sencilla.
- Sin embargo, una de las grandes virtudes de Java es la orientación a objetos
    - Imaginemos que contamos con un objeto alumno con los atributos nombre, edad, curso y nota.
    - Con los mecanismos estudiados hasta ahora nos veríamos obligados a introducir cada uno de los atributos de forma individual. Lo cual aumentan en varios órdenes de magnitud la complejidad del código.

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Por suerte, Java permite guardar directamente los objetos en ficheros binarios.
- Para que un objeto pueda ser guardado en un fichero binario, este debe implementar la interface ***serializable.***

```
public class Alumno implements Serializable
```

- Cualquier objeto que implemente esta interface podrá ser leído o escrito a través de
  - **ObjectInputStream** → Lectura de objetos.
  - **ObjectOutputStream** → Escritura de objetos.

- Esto es posible gracias a que cualquier objeto serializable puede ser convertido en una secuencia de bits que posteriormente puede ser restaurada, permitiendo así su recuperación.

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Para crear un ObjectInputStream debemos partir de un FileInputStream como muestran los siguientes ejemplos de código.

```
File f= new File("./myFiles/EjemploObjetos1.dat");
FileInputStream fis= new FileInputStream(f);
ObjectInputStream ois= new ObjectInputStream(fis);
```

```
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectInputStream ois= new ObjectInputStream(new FileInputStream(f));
```

- Del mismo modo, para crear un ObjectOutputStream debemos partir de un FileOutputStream.

```
File f= new File("./myFiles/EjemploObjetos1.dat");
FileOutputStream fos= new FileOutputStream(f);
ObjectOutputStream oos= new ObjectOutputStream(fos);
```

```
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectOutputStream oos= new ObjectOutputStream(new FileOutputStream(f));
```

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Para los siguientes ejemplos usaremos la clase **Alumno**
  - Como se puede ver la clase implementa la interface *serializable*.

```
public class Alumno implements Serializable{


    private static final long serialVersionUID = 1L;
    private String nombre;
    private int calificacion;

    public Alumno(String nombre, int calificacion) {
        super();
        this.nombre = nombre;
        this.calificacion = calificacion;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getCalificacion() {
        return calificacion;
    }

    public void setCalificacion(int calificacion) {
        this.calificacion = calificacion;
    }



}
```

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

El siguiente código muestra un ejemplo de escritura en fichero utilizando la clase ObjectOutputStream

```
//Creamos e inicializamos el flujo
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectOutputStream oos= new ObjectOutputStream(new FileOutputStream(f));

//Definimos un array con los objetos que vamos a insertar
Alumno alumnos[]= {new Alumno("Pepe",10), new Alumno("Maria", 5)};

//Insertamos los objetos
for (int i = 0; i < alumnos.length; i++) {
    oos.writeObject(alumnos[i]);
}
//Cerramos el flujo
oos.close();
```

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

El siguiente código muestra un ejemplo de escritura en fichero utilizando la clase ObjectOutputStream

```
//Definimos el stream de entrada
File f= new File("./myFiles/EjemploObjetos1.dat");
ObjectInputStream ois= new ObjectInputStream(new FileInputStream(f));
/*
 * Leemos el fichero hasta que se produzca la EOFException, momento en
 * el que cerramos el flujo
 */
try {
    while(true) {
        Alumno al=(Alumno) ois.readObject();
        System.out.println(al.getNombre()+ " "+ al.getCalificacion());
    }
} catch (ClassNotFoundException e) {

}catch (EOFException e) {
    System.out.print("Fin del archivo");
    ois.close();
}
```

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Problemática con la modificación de ficheros de objetos:
  - Cada vez que se crea un fichero de objetos Java crea una cabecera inicial con metainformación y tras esta se insertan los objetos.
  - Si el fichero se utiliza para añadir nuevos objetos (segunda apertura) se añade una nueva cabecera y se insertan objetos a partir de ella.
  - Esto produce que, cuando vamos a leer el fichero se detecte la segunda cabecera “intercalada” con los objetos y nos salte una StreamCorruptedException.
  - Esta cabecera se agrega cada vez que creamos un nuevo ObjectOutputStream(fichero) por lo que la solución es redefinir esta clase para evitar que la cree.

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Problemática con la modificación de ficheros de objetos:

```
public class MyObjectOutputStream extends ObjectOutputStream {

    protected MyObjectOutputStream() throws IOException, SecurityException {
        super();
    }
    public MyObjectOutputStream(OutputStream out) throws IOException {
        super(out);
    }

    @Override
    protected void writeStreamHeader() throws IOException{
        //Este es el método encargado de crear la cabecera,
        //Lo dejamos vácio para que no haga nada
    }


}
```

7. Ficheros binarios en Java

Objetos en ficheros binarios: ObjectInputStream / ObjectOutputStream

- Problemática con la modificación de ficheros de objetos:
  - De forma que
    - Si el archivo no existe usaremos la clase ObjectOutputStream
    - Si ya existía, usaremos la nueva clase creada por nosotros.

```
File f= new File("./myFiles/Ejemplo5.dat");
ObjectOutputStream os=null;
if(f.exists())
    os= new MyObjectOutputStream(new FileOutputStream(f));
else
    os= new ObjectOutputStream(new FileOutputStream(f));
```

7. Ficheros de acceso aleatorio en Java

La clase RandomAccessFile

- Hasta ahora, todas las operaciones que hemos realizado sobre ficheros se realizaban de forma secuencial.
  - Comenzábamos la lectura por el primer byte o caracter hasta llegar al final del fichero.
- Java dispone de la clase **RandomAccessFile** que dispone métodos para acceder al contenido de un fichero binario de forma aleatoria.

7. Ficheros de acceso aleatorio en Java

La clase RandomAccessFile

- Para crear una instancia de RandomAccessFile usaremos cualquiera de los siguientes constructores:
  - Indicando la ruta hacia el fichero.

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

  - Usando un objeto File para hacer referencia al fichero.

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

- En ambos casos debemos indicar el modo de acceso con el que queremos crear el objeto:
  - r → Abre el fichero en modo de solo lectura.

```
RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/miFichero.dat"), "r");
```

  - rw → Abre el fichero en modo lectura y escritura.

```
RandomAccessFile raf= new RandomAccessFile("./myFiles/miFichero.dat", "rw");
```

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria.

- Usaremos las clases DataInputStream y DataOutputStream.
- La clase **RandomAccessFile** maneja puntero que indica la posición actual en el fichero.
  - Cuando se crea el fichero el puntero se coloca en la posición 0.
    - Las sucesivas llamadas a .write() y .read() ajustan el puntero según la cantidad de bytes escritos.
  - Los métodos más importantes para el manejo de este puntero son:

Método	Función
long getFilePointer()	Devuelve la posición actual del puntero del fichero
void seek (long posición)	Coloca el puntero en una posición determinada
long length()	Devuelve el tamaño del fichero en bytes
int skipBytes (int desp)	Desplaza el puntero desde la posición actual el número de bytes indicados por desp.

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- Gracias al puntero de fichero podemos leer y escribir directamente en cualquier punto del fichero.
  - Para saber dónde debemos situar el puntero debemos conocer el tamaño de los registros que estamos almacenando.

Data Type	Size
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
boolean	1 bit
char	2 bytes

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- En el siguiente ejemplo vamos a escribir registros de tipo empleado, formados por:
  - ID → int (4 Bytes)
  - Apellido → 10 caracteres (2 Byte/char)(2*10=20 Bytes)
  - Departamento → int (4 Bytes)
  - Salario → Double (8 Bytes)
    - TOTAL → 36 Bytes

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

Escritura en fichero

```
//Declaramos los arrays a insertar
String[] apellidos= {"Martinez", "Gil", "Ondaruina"};
int[] dpto= {10,20,30};
double[] salario= {300.2, 123.45, 1256.1};

try {

    //Creamos el fichero
    RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "rw");

    for(int i=0;i<apellidos.length;i++) {
        //Insertamos la información
        raf.writeInt(i+1);
        StringBuffer sb= new StringBuffer(apellidos[i]);
        sb.setLength(10);
        raf.writeUTF(sb.toString());
        raf.writeInt(dpto[i]);
        raf.writeDouble(salario[i]);
    }
    //Cerramos el fichero
    raf.close();
} catch (FileNotFoundException e) {

    e.printStackTrace();
} catch (IOException e) {

    e.printStackTrace();
}
```

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

Lectura de fichero

```
//Creamos el fichero
try {
    RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "r");
    //Declaramos las variables necesarias
    int pos=0;
    int id, dep;
    Double salario;
    char[] apellido= new char[10];
    //Recorremos el fichero
    for(pos=0;pos<raf.length();pos+=36) {
        raf.seek(pos);      //Posicionamos el puntero al comienzo del registro
        id=raf.readInt();
        for(int i=0;i<10;i++) {
            apellido[i]=raf.readChar();
        }
        dep=raf.readInt();
        salario=raf.readDouble();
        System.out.printf("ID: %s, Apellido: %s, Dpto: %d Salario: %.2f %n",
                id, String.valueOf(apellido),dep, salario );


    }
    raf.close(); //Cerramos el fichero
```

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

- En los dos ejemplos anteriores (Escritura y lectura) hemos hecho uso de la clase RandomAccessFile para realizar una escritura y una lectura secuenciales.
  - Sin embargo, con este tipo de ficheros no es necesario recorrer todos los registros para poder acceder a uno concreto.
  - En nuestro ejemplo, conocemos el tamaño de cada registro por tanto, si quisiéramos acceder a uno en concreto podríamos sencillamente calcular su posición y ubicar el puntero en ella.
    - Por ejemplo, para leer los datos del empleado 5 deberíamos situar el puntero en (5*36) la posición 180

7. Ficheros de acceso aleatorio en Java

Lectura y escritura aleatoria

```
//Creamos el fichero
RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "r");
//Declaramos las variables necesarias

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

//Vamos a leer los datos del segundo registro (ID-1)
raf.seek(1*36);//Posicionamos el puntero al comienzo del registro 2
id=raf.readInt();
for(int i=0;i<10;i++) {
    apellido[i]=raf.readChar();
}
dep=raf.readInt();
salario=raf.readDouble();
System.out.printf("ID: %s, Apellido: %s, Dpto: %d Salario: %.2f %n",
        id, String.valueOf(apellido),dep, salario );
raf.close(); //Cerramos el fichero
```

7. Ficheros de acceso aleatorio en Java

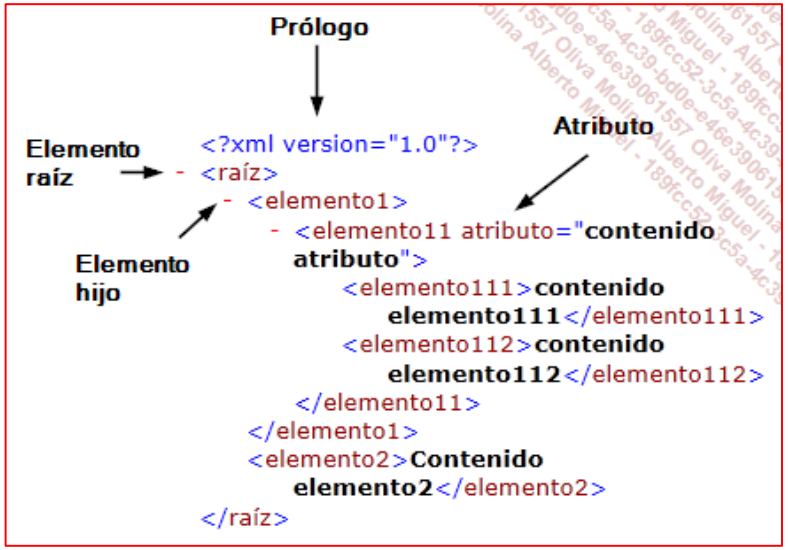
Lectura y escritura aleatoria

- Para mantener la estructura del fichero debemos controlar en qué posición del fichero se insertan cada uno de los registros.
  - Para insertar un nuevo registro aplicamos la función al identificador para cualquier posición.
    - En el siguiente ejemplo se insertará un empleado con identificador 20, debemos calcular

```
RandomAccessFile raf= new RandomAccessFile(new File("./myFiles/misEmpleados.dat"), "rw");
StringBuffer sb=null;
String apellido= "Hidalgo";
Double salario=2000.0;
int id=20;
int dpto=10;

long pos= (id-1)*36; //calculamos la posición
raf.seek(pos);//Nos posicionamos
raf.writeInt(id);//Escribimos el ID
sb=new StringBuffer(apellido);
sb.setLength(10);
raf.writeChars(sb.toString()); //Escribimos el apellido
raf.writeInt(dpto);//Escribimos el dpto
raf.writeDouble(salario); //Escribimos el salario
raf.close();//Cerramos el fichero
```

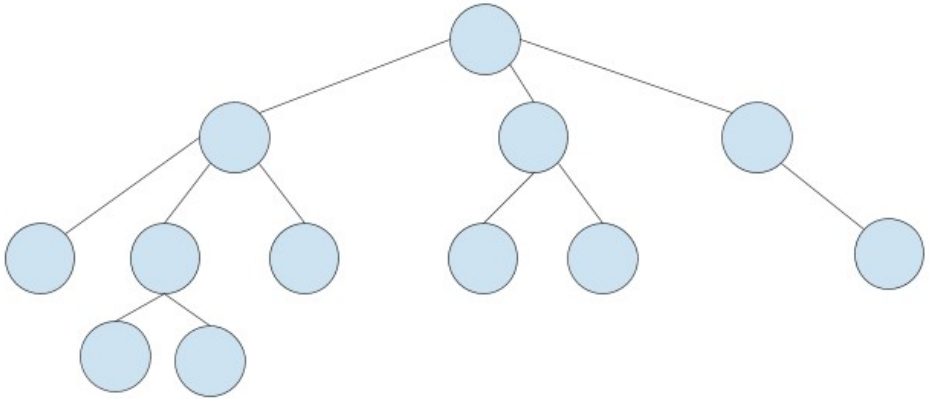
8. Trabajo con ficheros XML

- **XML** es un metalenguaje → Lenguaje para la definición de lenguajes de marcado.
- Nos permite jerarquizar y describir los contenidos dentro del propio documento.
- Los ficheros XML son ficheros donde la información se organiza forma secuencial y en orden jerárquico.
- Un fichero XML sencillo tiene la siguiente estructura:



8. Trabajo con ficheros XML

- En un documento XML la información se organiza de manera jerárquica, de manera que los elementos se relacionan entre sí mediante relaciones de padres, hijos, hermanos, ascendientes, descendientes, etc.



8. Trabajo con ficheros XML

- Los ficheros XML se pueden utilizar para:
  - Proporcionar datos a una BBDD
  - Almacenar información en BBDD especiales.
  - Ficheros de configuración
  - Intercambio de información en entornos web (SOAP)
  - Ejecución de comandos en servidores remotos.
  - Etc.
- Para realizar cualquiera de estas operaciones es necesario un lenguaje de programación que aporte funcionalidad a XML (XML no es un lenguaje Turing completo).
- Para acceder a los ficheros XML, leer su contenido o alterar su estructura se utiliza un **procesador de XML o XML parser.**
- Algunos de los parser más empleados son
  - DOM
  - SAX

8. Trabajo con ficheros XML

Acceso a ficheros con DOM

- Los parser XML que trabajan con DOM almacenan toda la estructura jerarquica del documento en memoria principal.
- Es el estándar que definió W3C para trabajar con ficheros XML
- Este tipo de procesamiento necesita más recursos de memoria y tiempo que SAX.

- Para poder trabajar con DOM en Java vamos a hacer uso los paquetes
  - org.w3c.dom (Contenido en el JSDK)
  - javax.xml.parsers (Del API estándar de Java)
- Que proporcionan dos clases abstractas que debemos extender para trabajar con DOM.
  - **DocumentBuilderFactory**
  - **DocumentBuilder**

8. Trabajo con ficheros XML

Acceso a ficheros con DOM

- DOM no define ningún mecanismo para generar un fichero XML a partir de un árbol DOM.
  - Para ello usaremos el paquete javax.xml.transform. Este paquete permite especificar una fuente y un resultado:
    - Ficheros
    - Flujos de datos,
    - Nodos DOM
    - Etc.

8. Trabajo con ficheros XML

Acceso a ficheros con DOM

- Los programas en Java que utilicen DOM necesitan hacer uso de las siguientes interfaces:
  - **Document** → Es un objeto que representa el documento XML. Permite crear nuevos nodos.
  - **Element** → Representa cada uno de los elementos del documento.
  - **Node** → Representa a cualquier nodo del documento.
  - **NodeList** → Contiene una lista con los nodos hijos de un nodo determinado.
  - **Attr** → Permite acceder a los atributos de un nodo.
  - **Text** → Representa los datos caracter de un nodo.
  - **CharacterData** → Representa los datos caracter presentes en el documento.
  - **DocumentType** → Proporciona información contenida en la etiqueta <!DOCTYPE>

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Escritura

- A continuación vamos a crear un fichero XML de empleados a partir del fichero de empleados que usamos en el ejemplo anterior (Ficheros binarios).
- Lo primero que debemos hacer es importar los paquetes necesarios.

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

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Escritura

- Después crearemos el Document en el cual vamos a insertar a nuestros empleados.

```
/*
 * Crearemos un DocumentBuilder haciendo uso de la factoria
 * DocumentBuilderFactory
 */
DocumentBuilderFactory dbf= DocumentBuilderFactory.newInstance();
DocumentBuilder builder=dbf.newDocumentBuilder();

/*           * Creamos un documento vácio
 *  Nombre --> RegistroEmpleados
 *  Nodo Raíz --> Empleados
 */
DOMImplementation implementacion= builder.getDOMImplementation();
Document registroEmpleados = implementacion.createDocument(null, "empleados", null);
//Asignamos la versión de XML
registroEmpleados.setXmlVersion("1.0");
```

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Escritura

- E iremos creado cada uno de los empleados con sus datos

```
//Creamos un nodo empleado
Element empleado= registroEmpleados.createElement("empleado");
//Lo añadimos como hijo de empleados
registroEmpleados.getDocumentElement().appendChild(empleado);
//Creamos el nodo ID
Element id= registroEmpleados.createElement("id");
//Creamos el nodo texto con el valor de la ID
Text texto= registroEmpleados.createTextNode("01");
//Añadimos el valor al nodo ID
id.appendChild(texto);
//Añadimos el nodo ID a empleado
empleado.appendChild(id);
Element nombre= registroEmpleados.createElement("nombre");
texto= registroEmpleados.createTextNode("Antonio");
nombre.appendChild(texto);
empleado.appendChild(nombre);
Element apellidos= registroEmpleados.createElement("apellido");
texto= registroEmpleados.createTextNode("Morales");
apellidos.appendChild(texto);
empleado.appendChild(apellidos);
```

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Escritura

- Finalmente debemos escribir nuestro Document en disco.

```
/*
 * Finalmente, para guardar el documento en disco debemos:
 *
 * 1. Crear el origen de datos (Nuestro Document)
 * 2. Crear el resultado (El fichero de destino)
 * 3. Crear un TransformerFactory
 * 4. Realizar la transformación
 */

Source origen= new DOMSource(registroEmpleados);
Result resultado= new StreamResult(new File ("Empleados.xml"));
Transformer transformador = TransformerFactory.newInstance().newTransformer();
transformador.transform(origen, resultado);
```

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Escritura

- Del mismo modo que utilizamos la clase transform para enviar nuestro Document a fichero, podemos usarla para mostrarlo por la salida estándar

```
/*
 * Mostramos el resultado por la salida estándar
 */
Result salidaEstnadar = new StreamResult(System.out);
transformador.transform(origen, salidaEstnadar);
```

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Lectura

- Para leer un documento XML haciendo uso de DOM debemos crear una instancia de DocumentBuilderFactory para construir el parser y, a través de él cargar el documento.

```
//Creamos el DocumentBuilder para poder obtener el Document
DocumentBuilderFactory dbf= DocumentBuilderFactory.newInstance();
DocumentBuilder builder=dbf.newDocumentBuilder();
//Leemos el Document desde el fichero
Document registroEmpleados= builder.parse(new File("Empleados.xml"));
//Normalizamos el documento para evitar errores de lectura.
registroEmpleados.getDocumentElement().normalize();
```

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Lectura

- Después creamos una lista como todos los elementos empleado usando la clase NodeList

```
//Mostramos el nombre del elemento raíz
System.out.println("El elemento raíz es "+
registroEmpleados.getDocumentElement().getNodeName());

//Creamos una lista de todos los nodos empleado
NodeList empleados= registroEmpleados.getElementsByTagName("empleado");
//Mostramos el nº de elementos empleado que hemos encontrado
System.out.println("Se han encontrado "+empleados.getLength()+" empleados");
```

8. Trabajo con ficheros XML

Acceso a ficheros con DOM: Lectura

- Finalmente, usando un bucle recorremos el NodeList y mostramos su contenido.

```
//Recorremos la lista.
for(int i=0; i<empleados.getLength();i++) {
    //Obtenemos el primer nodo de la lista
    Node emple= empleados.item(i);
    //En caso de que ese nodo sea un Elemento
    if(emple.getNodeType()==Node.ELEMENT_NODE) {
        //Creamos el elemento empleado y leemos su información
        Element empleado= (Element)emple;
        System.out.print("ID: " + empleado.getElementsByTagName("id").
                item(0).getTextContent() );
        System.out.print("\tNombre: " + empleado.getElementsByTagName("nombre").
                item(0).getTextContent()  );
        System.out.println("\tApellido: " + empleado.getElementsByTagName("apellido").
                item(0).getTextContent()  );
    }
}
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- SAX (Simple API for XML) es un conjunto de clases e interfaces que ofrecen una herramienta para el procesamiento de documentos XML.
- Permite analizar los documentos de forma secuencial (No carga todo el documento en memoria, a diferencia de DOM).
- La API de SAX está totalmente incluida dentro de JRE.
- SAX es más complejo de programar que DOM, sin embargo nos ofrece un menor consumo de memoria, por lo que es más adecuada para el tratamiento de ficheros XML de gran tamaño.

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- La lectura de un XML con SAX produce eventos que ocasionan la llamada a diferentes métodos:
  - **startDocuement()** / **endDocument()** → Encuentra las etiquetas de inicio y fin del documento.
  - **startElement()** / **endElement()** → Encuentra las etiquetas de apertura y cierre de un elemento.
  - **characters()** → Accede al contenido (Caracteres) de un elemento.

```
<?xml version="1.0" encoding="UTF-8"?><!--startDocument()-->
<alumnos><!--startElement()-->
    <alumno><!--startElement() -->
        <nombre><!--startElement()-->
            Juan <!--characters-->
        </nombre><!--endElement()-->
    </alumno><!--endElement() -->
    <alumno><!--startElement() -->
        <nombre><!--startElement()-->
            Marta <!--characters-->
        </nombre><!--endElement()-->
    </alumno><!--endElement() -->
</alumnos><!--endElement() -->
                                    <!--endDocument()-->
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Vamos a ver un pequeño ejemplo en Java.
  - Lo primero que haremos será importar todas las clases e interfaces necesarias.

```
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;
```

- A continuación crearemos un objeto XMLReader (Objeto XML)

```
XMLReader procesadorxml= XMLReaderFactory.createXMLReader();
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Lo siguiente que haremos será indicarle a nuestro XMLReader que objetos poseen los métodos que trataran los eventos que se produzcan en la lectura.
  - **ContentHandler** → Recibe las notificaciones de los eventos que se producen en el documento.
  - **DTDHandler** → Recoge eventos relacionados con el DTD.
  - **ErrorHandler** → define métodos de tratamiento de errores.
  - **EntityResolver** → sus métodos se llaman cada vez que se encuentra una referencia a una entidad.
  - **DefaultHandler** → Esta clase provee la implementación por defecto para todos sus métodos, el programador debe definir aquellos métodos que vayan a ser utilizados por la aplicación. **Esta es la clase de la que extenderemos para poder crear nuestro Parser XML.**

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Por tanto, lo primero que debemos hacer antes de continuar será crearnos una clase que extienda de DefaultHandler para poder manejar los eventos de lectura.

```
public class AlumnosReader extends DefaultHandler{

    public AlumnosReader() {
        super();
    }
    @Override
    public void startDocument() throws SAXException {
        super.startDocument();
        System.out.println("Inicio del documento");
    }
    @Override
    public void endDocument() throws SAXException {
            super.endDocument();
            System.out.println("Fin del documento");
    }
    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
        super.startElement(uri, localName, qName, attributes);
        System.out.printf("\t Inicio del Elemento %s %n", localName);
    }
    @Override
    public void endElement(String uri, String localName, String qName) throws SAXException {
            super.endElement(uri, localName, qName);
            System.out.printf("\t Inicio del Elemento %s %n", localName);
    }
    @Override
    public void characters(char[] ch, int start, int length) throws SAXException {
            super.characters(ch, start, length);
            String car= new String(ch, start, length);
            car.replace("[\t\n]", "");
            System.out.printf("\t Valor del Elemento %s %n", car);
    }
}
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Una vez que tenemos nuestro propio manejador, debemos indicarle a nuestro XML Parser que utilice una instancia del mismo para poder leer el fichero.
- Para indicar al procesador XML que objetos realizarán el tratamiento se utiliza alguno de los siguientes métodos incluidos dentro de los objetos XMLReader:
  - setContentHandler()
  - setDTDHandler()
  - setEntityResolver
  - setErrorResolver()
- Cada uno de ellos trata un evento diferente y está asociada a una interfaz determinada.
- En nuestro caso usaremos un ContentHandler, por tanto:

```
procesadorxml.setContentHandler(new AlumnosReader());
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Finalmente debemos definir la fuente de la que vamos a leer (El fichero) usando un objeto de la clase InputSource.

```
InputSource xmlFile= new InputSource("./alumnos.xml");
```

- Una vez tengamos la fuente creada y el ContentHandler asignado vamos a proceder a leer el fichero simplemente haciendo uso del método Parse de nuestro procesador XML.

```
procesadorxml.parse(xmlFile);
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Código completo de la clase principal.

```
public class ReaderSAX {

    public static void main(String[] args) {
        try {
            XMLReader procesadorxml= XMLReaderFactory.createXMLReader();
            procesadorxml.setContentHandler(new AlumnosReader());
            InputSource xmlFile= new InputSource("./alumnos.xml");
            procesadorxml.parse(xmlFile);

        } catch (SAXException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

8. Trabajo con ficheros XML

Acceso a ficheros con SAX

- Por tanto, a modo de resumen, para poder leer un fichero usando SAX debemos:
  1. Creamos una clase que extienda de DefaultHandler e implementamos los métodos necesarios para el tratamiento.
  2. Creamos una nueva clase que será la encargada de la ejecución principal de nuestra aplicación (MAIN) en la que:
     a. Creamos un XMLReader (Nuestro parser)
     b. Asignamos a nuestro XMLReader un objeto de la clase DefaultHander que acabamos de crear.
     c. Creamos un objeto InputSource indicando el origen de datos.
     d. Usamos el método parse de nuestro XMLReader para leer el fichero.

Dudas y preguntas