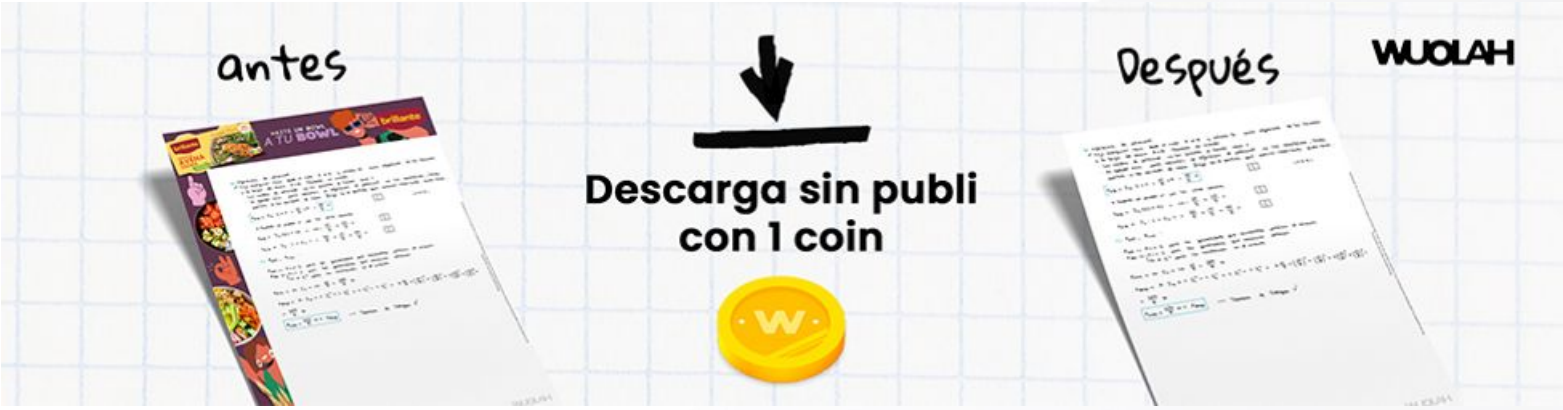
ACCESO-A-DATOS-TEMA-1.pdf

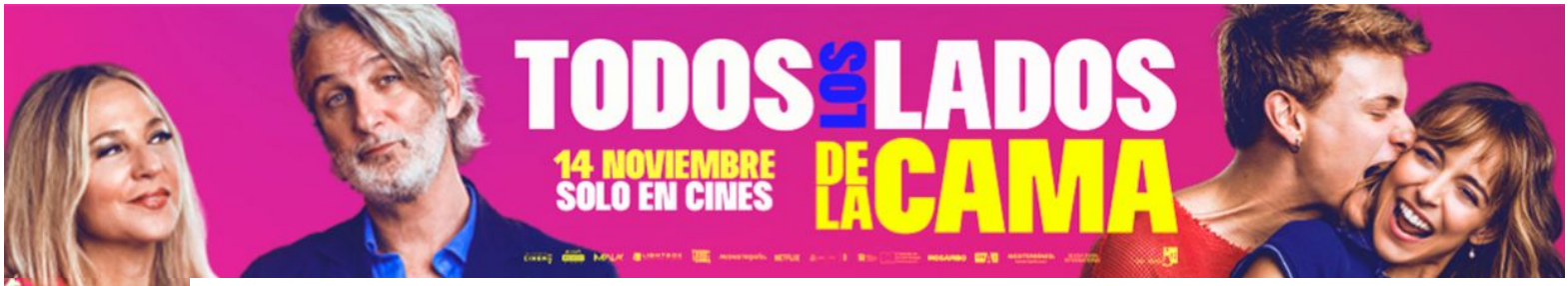
user_4383848

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España





ACCESO A DATOS - TEMA 1
INTRODUCCIÓN AL MANEJO DE FICHEROS

1. DEFINICIÓN Y TIPOS DE FICHEROS:

Es la sucesión de bits que finalmente se almacenan en un determinado dispositivo.
Está compuesto por **nombre** (que identifica al fichero) y **extensión** (que indica la tipología del archivo). Tipos:

De texto (ASCII) → Líneas de texto organizadas en código ASCII. (Archivos de aplicaciones de ofimática, archivos de programas, imágenes, vídeo, etc.)

Binarios → Representada en código binario. Entre otros, tenemos ficheros .bin o .dat.

La clase File (Java.io.File)

La clase File nos será de gran ayuda para **crear, modificar y eliminar** ficheros.

Instanciar objeto clase File → File fichero = new File("ruta");
Crear fichero → fichero.createNewFile();
Borrar fichero → fichero.delete();

Para **mover un fichero**, primero debemos asegurarnos de que existe. Por otra parte, al cargar la ruta del fichero origen «**fileOrigen**» y ejecutar el método **renameTo()**, debemos indicar al método el fichero destino «**fileDestino**». Hay que tener en cuenta que la ruta destino debe ser una ruta operativa y existente.

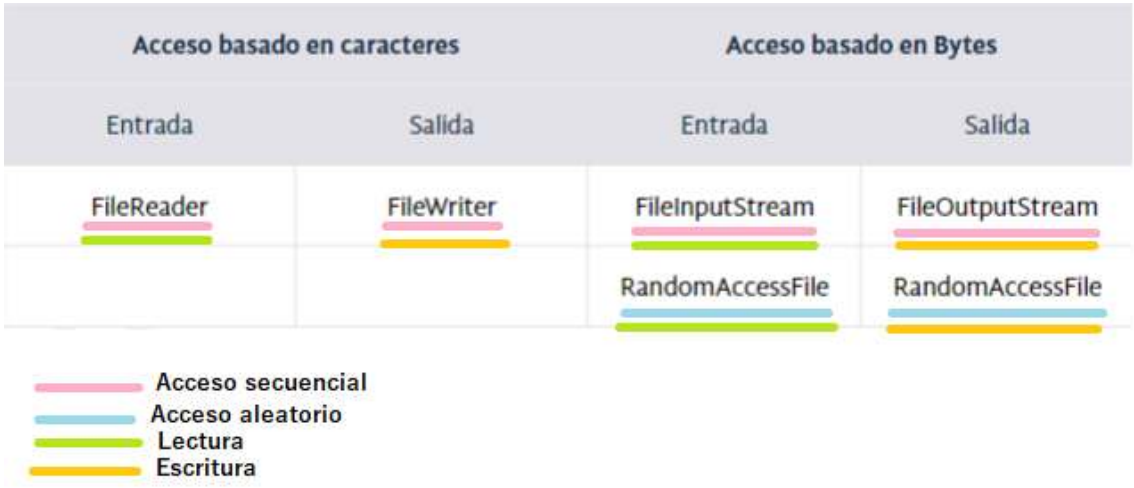
Método	Descripción
createNewFile()	Genera el fichero indicado
delete()	Borra el fichero
mkdirs()	Crea directorio indicado
getName()	Devuelve *String* con nombre del fichero
getPath(), getAbsolutePath()	Devuelve la ruta relativa y absoluta
getParent()	Devuelve el directorio superior
renameTo()	Acepta como parámetro un nuevo File que será la nueva ruta del fichero
exists()	Comprueba si existe el fichero
canWrite(), canRead()	Comprueba si puede ser escrito o leído
listFiles()	Devuelve un *array* con los ficheros del directorio indicado
lastModified()	Devuelve últimas modificaciones

2. FORMAS DE ACCEDER A UN FICHERO:

Acceso secuencial: En el modo secuencial la información de nuestro fichero es una secuencia de caracteres o bytes, de forma que, para acceder a un determinado byte o carácter del fichero, deberíamos de haber pasado previamente por todos los anteriores. Han de ser recorridos byte a byte o carácter a carácter.

Acceso aleatorio o directo: Para acceder a un registro o posición determinada de nuestro fichero. Se recorren estableciendo un puntero en bytes el cual indicará la posición exacta donde vamos a realizar la lectura y/o escritura, y al que se podrá acceder directamente.

3. OPERACIONES DE GESTIÓN DE FICHEROS. CLASES ASOCIADAS:



La **clase RandomAccessFile** además tiene diferentes modos de acceso:

•**"r":**modo lectura (read mode) obtendremos un IOException si utilizamos este modo en métodos de escritura.

• **"rw"**:modo lectura y escritura.

•**"rwd"**:modo de lectura y escritura de forma síncrona. Se escribirán todas las actualizaciones del contenido del fichero.

• **"rws"**: modo de lectura y escritura de forma síncrona. Se escribirán todas las actualizaciones el contenido del fichero pero además, se escribirán los metadatos.

RandomAccessFile posee dos constructores, cuyo segundo parámetro es el mismo: el modo de acceso.
Sin embargo, para el primer parámetro tenemos la versión del constructor en la que podemos introducir el objeto File directamente o, como en el caso de la imagen superior, podemos introducir la ruta del fichero directamente como *String*.

Con el método **seek()** nos posicionamos en el punto que indiquemos del fichero. Acepta como parámetro un objeto de tipo «long».

```
file.seek(pos:8);
```

Si usamos el método **getFilePointer()**, como bien se define, obtendremos como respuesta un tipo long. Este número es exactamente la posición del puntero en Bytes.

```
long filePointer = file.getFilePointer();
```

Lectura y escritura con RandomAccessFile:

Con el método **read()** podremos leer un *byte* directamente de nuestro fichero. Devolverá dicho *byte* a partir de la posición actual del puntero.

Usaremos el método **write()** para escribir un *byte*. Dicho *byte* se escribirá en la posición actual donde se encuentre el puntero. Acepta como parámetro un entero (el *byte* a escribir).

4. OPERACIONES CON BUFFER:

Son un conjunto de clases cuyo fin es leer y escribir información mejorando el rendimiento del sistema en comparación con nuestras clases aprendidas FileInputStream o FileOutputStream.

Buffer: es un espacio determinado y temporal que se aloja en memoria para realizar ciertas operaciones. Se almacena en memoria interna bloques de bytes completos (buffer), y el acceso a memoria es mucho más rápido que a disco
Cada vez que agotamos esa información del buffer y se requiere más, se vuelve a volcar otro bloque de bytes a la memoria (buffer), de esta forma el rendimiento se ve notablemente mejorado.

Las más destacadas son:
BufferedInputStream, BufferedOutputStream, BufferedWriter y BufferedReader