

ACCESO A DATOS
TÉCNICO EN DESARROLLO DE APLICACIONES
MULTIPLATAFORMAS

Introducción al manejo de ficheros

01

ÍNDICE

/ 1. Introducción y contextualización práctica	4
/ 2. Definición y tipos de ficheros	5
2.1. La clase File (Java.io.File)	5
2.2. Otros métodos de utilidad	6
/ 3. Formas de acceder a un fichero	7
/ 4. Caso práctico 1: “Tratamiento de ficheros”	7
/ 5. Operaciones de gestión de ficheros. Clases asociadas	8
5.1. Clases de uso de acceso secuencial	8
5.2. Acceso con bytes	8
5.3. Clases de uso de acceso aleatorio	10
/ 6. Operaciones con buffer	12
/ 7. Caso práctico 2: “Leyendo y escribiendo información”	13
/ 8. Resumen y resolución del caso práctico de la unidad	13
/ 9. Webgrafía	14

OBJETIVOS

Conocer distintas clases para el manejo de ficheros.

Saber elegir entre acceso aleatorio o secuencial dependiendo de la lógica de negocio establecida.

Aprender operaciones básicas de ficheros de acceso secuencial y de acceso aleatorio.

/ 1. Introducción y contextualización práctica

A lo largo de este tema veremos las distintas formas de acceso a ficheros mediante el lenguaje Java.

Como veremos, una práctica muy común será realizar la descarga de uno o varios ficheros con el objetivo de manipularlos, es decir, simplemente coger parte de información y moverla.

También estudiaremos como es la creación, modificación y borrado de ficheros y directorios. Para ello, nos adentraremos en conocer parte de la librería “Java.io” donde podremos realizar las operaciones nombradas previamente.

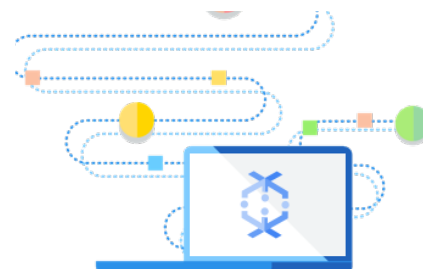


Fig. 1. Manejo de ficheros



Audio Intro. Lectura y escritura fichero

<https://bit.ly/3dwGW6X>





/ 2. Definición y tipos de ficheros

Definimos un fichero como la sucesión de bits que finalmente son almacenados en un determinado dispositivo. Está compuesto por nombre (identifica al fichero) y extensión (indica la tipología del archivo). Podemos organizar los ficheros en 2 grandes familias:

- **De texto (ASCII):** en su interior encontraremos líneas de texto organizadas en código ASCII. Algunos de los ejemplos son: archivos de aplicaciones de ofimática, archivos de programas de los distintos lenguajes de programación, ficheros de imágenes, de video, etc.
- **Binarios:** aquellos ficheros cuya información está representada en código binario. Para poder leer dichos ficheros, antes debemos saber cómo fueron escritos para escoger la misma lógica en su lectura. Algunos son .bin, .dat... entre otros.

2.1. La clase File (Java.io.File)

Ubicada en el paquete "java.io.File", la clase File, nos será de gran ayuda para crear, modificar y eliminar ficheros, además de que nos proporcionará información adicional de los mismos. A continuación, puedes observar como se instancia un objeto de clase File:

```
File fichero = new File( "carpeta/archivo" );
```

Fig. 2. Instancia clase File

Una vez instanciado el objeto podremos también crear dicho fichero, para ello usaremos la siguiente línea:

```
fichero.createNewFile();
```

Fig.3. Creación nuevo fichero

Por último, otra acción básica que deberemos de aprender para el uso y gestión de ficheros en Java es la eliminación de ficheros y directorios, usaremos la siguiente línea:

```
fichero.delete();
```

Fig.4. Eliminación de fichero



2.2. Otros métodos de utilidad

Una vez realizado el aprendizaje de la instanciación, creación y borrado de un fichero, veamos en código sobre cómo mover un fichero:

```
try{
    File fileOrigen =new File( pathname: "C:\\temp\\pruebas1.txt");
    File fileDestino =new File( pathname: "C:\\temp\\pruebas\\pruebas2.txt");

    if(fileOrigen.renameTo( fileDestino)){
        System.out.println("El fichero se movió correctamente!");
    }else{
        System.out.println("El fichero no pudo moverse!");
    }
}

}catch(Exception e){
    e.printStackTrace();
}
```

Fig. 5. Moviendo ficheros.

Para poder mover un fichero, lo primero de lo que hay que asegurarse, es que éste debe de existir. 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 de ser una ruta operativa y existente.

En la tabla siguiente observaremos los métodos más usados de la clase File:

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, el cual 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

Tabla 1. Principales métodos de la clase File



/ 3. Formas de acceder a un fichero

Cuando nos disponemos a acceder al contenido de un fichero es importante tener claro el criterio de acceso que vamos a usar. Disponemos de dos:

- **Acceso secuencial.**
- **Acceso aleatorio o directo.**

Previamente, tendremos que realizar un estudio del aplicativo a desarrollar para saber qué modo de acceso nos conviene. En el **modo secuencial**, como su nombre indica, 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.

Si lo que deseamos es acceder a un registro o posición determinada de nuestro fichero, entonces nuestro método a escoger será el **modo aleatorio**.

La diferencia más notable entre estos dos tipos de acceso es la siguiente: mientras los ficheros de **acceso secuencial** deberán ser recorridos byte a byte o carácter a carácter con el tiempo, procesado de la información, y uso de recursos que esto conllevaría, mientras que, en los **ficheros de acceso aleatorio o directo**, se establecerá 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.

A lo largo de este tema revisaremos las distintas herramientas y clases de las que disponemos para realizar los distintos métodos de acceso. A continuación, mostramos una tabla orientativa sobre algunas de las clases que podremos utilizar:

Acceso basado en caracteres		Acceso basado en Bytes	
Entrada	Salida	Entrada	Salida
FileReader	FileWriter	FileInputStream	FileOutputStream
		RandomAccessFile	RandomAccessFile

Tabla 2. Algunas clases de métodos de acceso.

/ 4. Caso práctico 1: “Tratamiento de ficheros”

Planteamiento: Juan está realizando en su empresa un gestor de archivos simple. Básicamente la idea es un gestor que cree directorios nuevos, nuevos ficheros y pueda moverlos. También le gustaría obtener la fecha de la última modificación de dichos ficheros.

Nudo: Juan ha investigado y ha encontrado la paquetería de Java: “java.io”. Duda porque entiende que es la librería de uso y análisis de ficheros, pero no sabe muy bien qué clase usar.

Desenlace: en este caso, según los requerimientos de Juan deberá escoger la clase “java.io.File”. Esta le permitirá tanto crear ficheros, como saber su fecha de modificación, y a parte podrá moverlos entre directorios.



Para ello, en primer lugar, instanciaremos el objeto File, y justo después procedemos a ver los métodos:

Para crear ficheros utilizaremos:

- `createNewFile()`

Para obtener la fecha de modificación:

- `lastModified()`

Para el movimiento de ficheros usaremos el método:

- `renameTo()`



Fig. 6. Manejo de ficheros

/ 5. Operaciones de gestión de ficheros. Clases asociadas

5.1. Clases de uso de acceso secuencial

Dentro de la familia del acceso secuencial, debemos tener en cuenta la posibilidad de trabajar con bytes o caracteres según nuestra conveniencia. Para ello vamos a exponer algunas de sus diferencias.

5.2. Acceso con bytes

Nos facilitan la lectura o escritura de un fichero como un “stream” de bytes. Para ello nos apoyaremos en la clase `FileInputStream` (lectura entrada de datos) y `FileOutputStream` (escritura salida de datos):

- **`FileInputStream` (lectura):** a continuación, se muestra un ejemplo de uso de esta clase:

```
InputStream entrada = new FileInputStream( "name: "C:\\temp\\pruebas\\pruebas2.txt");
```

Fig. 7. Instancia `FileInputStream`

Para empezar, instanciaremos el objeto de nuestra clase e indicaremos la ruta del fichero a cargar.

```
int data = entrada.read();
```

Fig. 8. Lectura `FileInputStream`

Con el método `read()` estaremos accediendo al primer byte del fichero. Éste devuelve un número entero, por lo tanto lo asignamos a una variable entera.


Una vez obtenido el primer byte, ya podremos trabajar con él. Con este método accedemos al primer Byte del fichero “en bruto” y nos servirá para leer imágenes o archivos binarios. Finalmente, cerramos el fichero y liberamos los recursos del sistema que estaban haciendo uso del “stream” de bytes:

```
entrada.close();
```


Fig. 9. Cierre FileInputStream

A modo aclaración puntualizar que en estos ejemplos se ha suprimido el control de excepciones para mayor limpieza de código. Más adelante veremos cómo manejarlas.

- **FileOutputStream (escritura):** lo explicaremos a través del siguiente vídeo



Vídeo 1. “Escritura ficheros
FileOutputStream”
<https://bit.ly/2U3CBQX>



5.2.1. Acceso con caracteres

El modo de acceso secuencial con caracteres, hace posible la lectura y escritura de un fichero a través de un “stream” de caracteres. Mientras que las clases vistas previamente leen y escriben por medio de Bytes, las que veremos a continuación lo hacen manipulando directamente caracteres. Estas son: FileReader y FileWriter.

- **FileReader (lectura):** básicamente esta clase nos facilitará la lectura de los ficheros de caracteres. Pasamos a ver su utilización:

```
Reader lector = new FileReader( fileName: "C:\\temp\\pruebas\\pruebas2.txt");  
int data = lector.read();  
System.out.println((char)data);  
lector.close();
```

Fig. 10. FileReader

Como podemos observar, la lógica es muy parecida a la clase vista con anterioridad FileInputStream. No obstante, ésta la usábamos para obtener los bytes “en bruto” directamente, mientras que la clase FileReader fue escrita con el propósito de leer flujos o “streams” de caracteres. Para ello, como podemos ver en el ejemplo, justo antes de mostrar el resultado por pantalla, realizamos un casting (char) para poder visualizar ese primer carácter de nuestro fichero “pruebas2.txt”

- **FileWriter (escritura):** con la siguiente clase seremos capaces de escribir ficheros de caracteres:

```
Writer escritorFicheros = new FileWriter( fileName: "C:\\temp\\pruebas\\pruebas2.txt");  
escritorFicheros.write( str: "Esto es un ejemplo de escritura");  
escritorFicheros.close();
```

Fig. 11. FileWriter



En la imagen superior podemos observar como **instanciamos nuestra clase de escritura en la primera línea de código**, en donde, además, indicamos la ubicación. En el caso que no existiera el fichero, se crearía automáticamente. Si el fichero existe, será sobrescrito.

En la segunda línea directamente insertamos los **caracteres que deseamos incluir en nuestro fichero**. Los pasamos como parámetro del método “write()”.

Por último, **cerramos el flujo aplicando el método “close()”**, de esta forma liberaremos los recursos del sistema que han sido ocupados previamente para dicha operación. Una vez cerrado el flujo tendremos la información escrita en nuestro fichero.

5.3. Clases de uso de acceso aleatorio

5.3.1. Instanciación de RandomAccessFile

Utilizaremos la clase **RandomAccessFile** para acceder a ficheros de forma aleatoria. A diferencia de las clases anteriores, permite abrir un fichero en modo lectura y además en modo lectura-escritura.

Esta clase permite acceder a un lugar determinado de un fichero y a su vez, leer información o escribirla. Veamos en detalle las distintas funcionalidades que nos ofrece dicha clase.

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 del contenido del fichero pero además, se escribirán los metadatos.

Veamos ahora cómo instanciar este tipo de objeto y algunos de sus métodos:

```
RandomAccessFile file = new RandomAccessFile("C:\\temp\\pruebas\\pruebas3.txt", "rw");
```

Fig. 12. RandomAccessFile Instanciación

RandomAccessFile posee dos constructores, en ambos constructores el 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, introducir la ruta del fichero directamente como String.

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

Fig.13. Método seek

Con el método `seek()` básicamente nos posicionamos en el punto que indiquemos del fichero. Acepta como parámetro un objeto de tipo “long”

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

Fig. 14. Método `getFilePointer`

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



Audio 1. Método Close automático
<https://bit.ly/3dyeWjm>



5.3.2. Lectura y escritura con `RandomAccessFile`

Continuamos con nuestra clase `RandomAccessFile`, a continuación, veremos la forma de leer y escribir un Byte con esta clase:

```
int unByte = file.read();
```

Fig. 15. Lectura Byte

Con el método `read()`, podremos leer un byte directamente de nuestro fichero. Devolverá dicho byte a partir de la posición actual del puntero. Veámoslo en el siguiente vídeo:



Video 2. `RandomAccessFile` read
<https://bit.ly/2MqSY66>



```
file.write( b: 68 );
```

Fig. 16. Escritura Byte

Usaremos el método `write()` para escribir un Byte. Dicho Byte será escrito en la posición actual donde se encuentre el puntero. Acepta como parámetro un entero (el byte a escribir). En este caso como podemos observar en la imagen, insertaremos la letra “D” en el fichero que corresponde a 68 en la codificación de caracteres ASCII.

Una vez escrito el carácter en el fichero la posición del puntero avanzará una posición.



De la misma forma que hemos realizado una lectura y escritura de un byte, veremos cómo hacerlo de un array de bytes en los siguientes fragmentos:

```
byte[] arrayBytes = new byte[1024];
int inicioPuntero = 0;
int size = 1024;
int bytesLeídos = file.read(arrayBytes, inicioPuntero, size);
```

Fig. 17. Lectura array bytes

Si en lugar de leer un byte, lo que nos interesa realmente es leer una cantidad determinada de bytes, podremos realizar una implementación del estilo a la imagen superior. En primer lugar, crearemos un array de bytes, en concreto 1024. A continuación, definimos la posición del puntero y el tamaño de bytes que queremos extraer de nuestro fichero. Posteriormente, usamos la sentencia `read()`, en la que en este caso, acepta como primer parámetro el array de bytes dónde se va a almacenar la información, y como segundo parámetro, el inicio o posición en la que comenzaremos a leer.

Finalmente, el último parámetro a introducir indica la cantidad de bytes que deseamos extraer. Este método devuelve un número entero que representa la cantidad de bytes que han sido leídos.

/ 6. Operaciones con buffer

En este caso hablaremos de 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`.

Concretamente, nos centraremos en las clases **`BufferedInputStream`**, **`BufferedOutputStream`**, **`BufferedWriter`** y **`BufferedReader`** que tienen una diferencia notable en relación con las estudiadas previamente.

Primero deberemos entender el concepto de Buffer.

Éste, básicamente es un espacio determinado y temporal que se aloja en memoria para realizar ciertas operaciones. En las clases que vimos en apartados anteriores, realizábamos operaciones de lectura o escritura capturando Bytes, sin embargo, con las clases con sufijo `Buffered` el funcionamiento interno será ligeramente distinto ya que se almacena en memoria interna bloques de bytes completos (buffer), y como ya bien sabemos, 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.

A continuación, expondremos un ejemplo de clase **`BufferedInputStream`**:

```
int bufferSize = 4 * 1024;
BufferedInputStream bufferedInputStream = new BufferedInputStream(
    new FileInputStream( "C:\\temp\\pruebas\\pruebas4.txt"),
    bufferSize);
int info = bufferedInputStream.read();
while(info != -1) {
    info = bufferedInputStream.read();
}
bufferedInputStream.close();
```

Fig. 18. `BufferedInputStream`



/ 7. Caso práctico 2: “Leyendo y escribiendo información”

Planteamiento: José tiene la necesidad de acceder un fichero de texto, ya que necesita recorrerlo y coger toda la información del mismo para guardar dicha información en otro fichero más grande que funciona como base de información de muchos ficheros.

Nudo: José se encuentra en la tesitura de elegir entre un modo de acceso secuencial y uno aleatorio. No sabe qué clase elegir y cual le conviene más para su lógica.

Desenlace: en este caso, José deberá elegir un modo secuencial. Así, recorrer todo el fichero y escribir en el otro al mismo tiempo.

Para esto, podría usar las clases «FileReader» y «FileWriter», ya que son clases creadas con el objetivo de leer y escribir este tipo de ficheros.

Para trabajar con estas clases en la resolución del caso, básicamente instanciamos una nueva clase FileReader, con la cual iremos leyendo byte a byte con el método read(). Este método nos devolverá un byte en forma de número entero. En este caso, una vez obtenida la información, procederemos a escribir en el fichero usando ese byte de información mediante el método write(), y pasando como parámetro el byte leído anteriormente.

Tras terminar las operaciones de lectura-escritura (entrada / salida de datos), simplemente tendríamos que tener en cuenta el uso del método .close(), tanto para nuestra clase de lectura como para la de escritura, para, de este modo, liberar recursos.

/ 8. Resumen y resolución del caso práctico de la unidad

En este tema hemos podido analizar la definición de fichero o archivo. Hemos aprendido a diferenciar entre los distintos tipos de ficheros y a cómo gestionarlos con su clase java.io.File.

Realizamos un estudio de los diferentes modos de acceso a los ficheros, y vimos la diferencia entre acceder a ficheros por medio de gestión de Bytes o por medio de modos de acceso orientados a caracteres.

También se han presentado clases de acceso secuencial, y de acceso aleatorio. Hemos podido observar como realizar las implementaciones de dichas clases y también las ventajas e inconvenientes de usar unas u otras. También se han estudiado qué clases pueden mejorar el rendimiento de nuestra máquina y ahorrar recursos.

RESOLUCIÓN DEL CASO PRÁCTICO DE LA UNIDAD.

Irene en este caso podrá elegir entre usar como clase de lectura FileInputStream o FileReader. Esta última, será más recomendable ya que fue diseñada para el manejo de caracteres:

```
import java.io.FileReader;
```

Fig. 20. Importación FileReader

Con dicha clase tendrá la posibilidad de recorrer dicho fichero y contar al mismo tiempo, las apariciones de dicha letra.



Una vez recorrido todo el fichero llegaría el momento de la escritura en disco. Para ello y de la misma forma, tendríamos opción de usar `FileOutputStream` o `FileWriter`. Nos quedaremos igualmente con `FileWriter` por estar desarrollado y orientado a la escritura de caracteres.

```
import java.io.FileWriter;
```

Fig. 21. Importación `FileWriter`

Una vez preparada nuestra clase de escritura. Nuestra compañera Irene ya tiene la cantidad de apariciones almacenada en memoria, por lo tanto, simplemente tendrá que añadirle un guion, la fecha actual del sistema y cerrar el documento.

/ 9. Webgrafía

Oracle web oficial:

<https://docs.oracle.com/javase/8/docs/api/?java/io>

MEDAC