

¿Qué es un Fichero?

En el contexto de la informática, un fichero o archivo es un conjunto de datos almacenados en un medio físico (un disco o memoria).

Un fichero se identifica por:

- La ruta completa donde está almacenado o donde se pretende almacenar (si es nuevo).
- El nombre completo incluyendo la extensión (si la tiene)

¿Qué es un fichero de texto?

La información que se registra en un fichero siempre son bytes. Sin embargo, estos bytes a veces son legibles por un humano y otras veces no. Cuando los bytes almacenados corresponden únicamente a caracteres alfanuméricos (además de signos de puntuación, separaciones, etc.) se dice que se trata de un fichero de texto.

Este puede ser abierto y leído desde cualquier editor de textos.

¿Cómo puedo trabajar con ficheros en Java?

En Java existe una clase llamada **File**. Un objeto de esta clase representa un fichero real. Puedo crear un objeto de tipo **File** así:

```
File file = new File("c:/temporal/nombreFichero.txt");
```

En este ejemplo, mi fichero se llama “nombreFichero.txt” y está ubicado en la carpeta “c:/temporal”

No es obligatorio que el fichero exista realmente para que se pueda crear un **File**. Es decir, lo normal será una de estas dos opciones:

- a) El fichero ya existe → Me creo un **File** para leer o escribir en él.
- b) El fichero no existe aún → Me creo un **File** para crear el fichero por primera vez y luego escribir en él.

Si el fichero no existe, ¿cómo lo creamos? Utilizamos el método **createNewFile()**

```
try {  
    Boolean ficheroNuevo = file.createNewFile();  
    if (ficheroNuevo) {  
        // el fichero se ha creado nuevo  
    } else {  
        // el fichero ya existía, no se ha creado  
    }  
} catch (IOException e) {  
    System.out.println("Ha habido un error al crear el fichero: " + e.getMessage());  
    e.printStackTrace();  
}
```

IMPORTANTE:

- El fichero no tiene por qué existir, pero sí debe existir la ruta donde lo queremos crear. Si no, se generará un error `IOException`
- Si ya existe un fichero con el mismo nombre en el mismo sitio, el método no hará nada. Devolverá `false`. En caso contrario, devolverá `true`
- Es obligatorio tratar la excepción `IOException` al llamar el método. O la capturamos para hacer algo con ella, o la lanzamos hacia arriba en la pila de ejecución.

La clase `File` tiene otros métodos útiles para conocer información de los ficheros. Por ejemplo:

- `getAbsolutePath()` → Devuelve una cadena con la ruta + nombre del fichero completo.
- `canExecute()` `canRead()` `canWrite()` → Devuelven booleanos indicando si se puede ejecutar, leer o escribir en el fichero
- `getTotalSpace()` `getFreeSpace()` → Devuelven enteros con el tamaño total y el libre del fichero.

¿Cómo puedo trabajar con directorios o carpetas en Java?

Se trabaja también con la clase `File`. En Java, una carpeta es como si fuera un fichero más. A la hora de crear el `File` de una carpeta, indicamos la ruta y nombre de esa carpeta.

La clase `File` tiene métodos específicos para trabajar con carpetas:

- `isFile()` / `isDirectory()` → Devuelven un booleano indicando el tipo de file
- `mkdir()` / `mkdirs()` → Crea la carpeta de la ruta si no existe
- `getParentFile()` → Devuelve un `File` con la carpeta padre donde está el `File` actual.

¿Cómo puedo escribir en un fichero?

Lo podemos hacer utilizando la clase `FileWriter`. Nos creamos un `FileWriter` a partir de un objeto `File`. Así:

```
File file = new File("c:/temporal/nombreFichero.txt");
FileWriter writer = new FileWriter(file);
```

Luego puedo utilizar el método `write()` para escribir. A este método sólo hay que pasarle una cadena. Si quieres que haya un salto de línea, tendrás que añadir `"\n"`

Ten en cuenta que `FileWriter` sobrescribirá el fichero si ya existía (borrando su contenido previo). Si no quieres que ocurra esto y que se escriba a continuación de lo que ya hubiera, tendrás que crear el `FileWriter` pasando un segundo parámetro con valor `true`:

```
FileWriter writer = new FileWriter(file, true);
```

Es importante cerrar siempre el `FileWriter` cuando hayamos terminado con el método `close()`

Además, todos estos métodos pueden lanzar `IOException`, por lo que habrá que controlarla capturándola o propagándola.

Ejemplo:

```
try {  
    File file = new File("c:/temporal/nombreFichero.txt");  
    FileWriter writer = new FileWriter(file);  
  
    writer.write("Esto va en la primera línea\n");  
    writer.write("Esto va en la segunda línea\n");  
  
    writer.close();  
} catch (IOException e) {  
    System.out.println("Ha habido un errores al escribir: " + e.getMessage());  
}
```

Resumen para escribir:

1. Obtener objeto `File`
2. Obtener objeto `FileWriter` usando el `File` anterior
3. Escribir todo lo que sea necesario con `write()` o con `append()`
4. Cerrar el `FileWriter` con `close()`

NOTA: Si el fichero no existe, cuando intentemos escribir en él, se creará automáticamente. Pero la ruta de carpetas que lo va a contener sí tiene que existir.

¿Cómo puedo leer un fichero?

Lo podemos hacer utilizando la clase `Scanner` que ya conocemos. Hasta ahora hemos usado el `Scanner` para leer desde el teclado. Pero podemos utilizarlos igual para leer de un fichero. A la hora de crear el `Scanner`, en lugar de indicar `System.in`, indicamos el fichero que queremos leer:

```
File file = new File("c:/temporal/nombreFichero.txt");  
Scanner scanner = new Scanner(file);
```

Tendremos que controlar una posible excepción: `FileNotFoundException`

Una vez que tenemos el `Scanner`, podemos utilizar el método `nextLine()` para ir leyendo línea a línea todo el fichero. Pero... ¿cómo sabemos cuándo hemos terminado? Nuestro viejo amigo `Scanner` tiene un método llamado `hasNext()` que devuelve un boolean indicando si todavía hay algo por leer. Por tanto, bastaría con un bucle `while`:

```
while(scanner.hasNext()) {  
    String linea = scanner.nextLine();  
    System.out.println(linea);  
}  
scanner.close();
```