



FICHEROS DE TEXTO

CLASE FILE

La clase File nos permite gestionar y administrar un fichero o directorio a través de sus metadatos.

Podremos crearlo, borrarlo, cambiarle el nombre o cualquiera de sus propiedades.

Pero NO permite leer ni escribir en ellos.

Ejemplo de uso

Si queremos abrir el fichero `c:\docs\archivo.txt`, haremos lo siguiente:

```
File fichero = new File ("c:\\docs\\archivo.txt");
```

Observa que las barras van dobles si usamos la notación de Windows.

Si el archivo fuera de Linux, pondríamos

```
File fichero = new File ("/docs/archivo.txt");
```

Rutas absolutas y rutas relativas

Es muy importante saber distinguir entre una ruta absoluta y una ruta relativa en un sistema de ficheros, ya que nos interesará en la mayoría de los casos trabajar directamente con rutas relativas.

- Una ruta absoluta contiene el path completo de un archivo.
- Una ruta relativa contiene el path desde la posición en la que yo me encuentro.

Métodos de la clase File

Constructores:

- `File(pathArchivo: String)`
- `File(directorioPadre: String, archivo: String)`
- `File(directorioPadre: File, archivo: String)`

Métodos de la clase File

Métodos sobre sus atributos:

- `exists(): boolean`
- `canRead(): boolean`
- `canWrite(): boolean`
- `isDirectory(): boolean`
- `isFile(): boolean`
- `isAbsolute(): boolean`
- `isHidden(): boolean`

Métodos de la clase File

Métodos sobre la ruta del fichero o directorio:

- `getAbsolutePath(): String`
- `getCanonicalPath(): String`
- `getName(): String`
- `getPath(): String`
- `getParent(): String`

Métodos de la clase File

Métodos sobre metadatos y de manipulación

- `lastModified(): long`
- `length(): long (en bytes)`
- `listFile(): File[]`
- `delete(): boolean`
- `renameTo(dest: File): boolean`
- `mkdir(): boolean`
- `makedirs(): boolean`

Ejemplo de uso:

LISTING 12.12 TestFileClass.java

```
1  public class TestFileClass {
2      public static void main(String[] args) {
3          java.io.File file = new java.io.File("image/us.gif");
4          System.out.println("Does it exist? " + file.exists());
5          System.out.println("The file has " + file.length() + " bytes");
6          System.out.println("Can it be read? " + file.canRead());
7          System.out.println("Can it be written? " + file.canWrite());
8          System.out.println("Is it a directory? " + file.isDirectory());
9          System.out.println("Is it a file? " + file.isFile());
10         System.out.println("Is it absolute? " + file.isAbsolute());
11         System.out.println("Is it hidden? " + file.isHidden());
12         System.out.println("Absolute path is " +
13             file.getAbsolutePath());
14         System.out.println("Last modified on " +
15             new java.util.Date(file.lastModified()));
16     }
17 }
```

create a File
exists()
length()
canRead()
canWrite()
isDirectory()
isFile()
isAbsolute()
isHidden()

getAbsolutePath()

lastModified()

Escribir en un fichero

Clase PrintWriter

Para escribir en un fichero tendremos que crear algún objeto que cree un flujo de salida hacia dicho fichero.

Una de las posibilidades de la clase PrintWriter.

Se inicializa con el fichero sobre el que queremos escribir y se utiliza exactamente igual que el print, con todas sus variantes.

Ejemplo de uso de la clase PrintWriter

LISTING 12.13 WriteData.java

```
1 public class WriteData {
2     public static void main(String[] args) throws IOException {
3         java.io.File file = new java.io.File("scores.txt");
4         if (file.exists()) {
5             System.out.println("File already exists");
6             System.exit(1);
7         }
8
9         // Create a file
10        java.io.PrintWriter output = new java.io.PrintWriter(file);
11
12        // Write formatted output to the file
13        output.print("John T Smith ");
14        output.println(90);
15        output.print("Eric K Jones ");
16        output.println(85);
17
18        // Close the file
19        output.close();
20    }
21 }
```

create PrintWriter

print data

John T Smith 90
Eric K Jones 85

scores.txt

close file

Leer de un fichero

Clase Scanner

Para leer de un fichero usaremos la clase Scanner, pero cambiando la entrada estándar System.in por el acceso a un fichero.

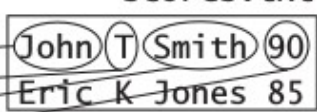
Se usa igual que la hemos usado hasta ahora, con los métodos `nextLine()`, `next()`, `nextInt()`, `nextDouble()`...

Sabremos si hemos llegado al final de un fichero usando el método `hasNext():boolean`.

Ejemplo de uso:

LISTING 12.15 ReadData.java

```
1 import java.util.Scanner;
2
3 public class ReadData {
4     public static void main(String[] args) throws Exception {
5         // Create a File instance
6         java.io.File file = new java.io.File("scores.txt");
7
8         // Create a Scanner for the file
9         Scanner input = new Scanner(file);
10
11        // Read data from a file
12        while (input.hasNext()) {
13            String firstName = input.next();
14            String mi = input.next();
15            String lastName = input.next();
16            int score = input.nextInt();
17            System.out.println(
18                firstName + " " + mi + " " + lastName + " " + score);
19        }
20
21        // Close the file
22        input.close();
23    }
24 }
```



The diagram shows a box labeled "scores.txt" containing two lines of text: "John T Smith 90" and "Eric K Jones 85". Arrows point from specific parts of these lines to the corresponding scanner methods in the code above. The first line "John T Smith 90" has arrows pointing to `input.next()` (for "John"), `input.next()` (for "T"), `input.next()` (for "Smith"), and `input.nextInt()` (for "90"). The second line "Eric K Jones 85" has arrows pointing to `input.next()` (for "Eric"), `input.next()` (for "K"), `input.next()` (for "Jones"), and `input.nextInt()` (for "85").