TEMA 1 – Manejo de Ficheros

En este tema aprenderemos conceptos básicos que hay que tener en cuenta a la hora de trabajar con ficheros en Java.

# ¿Qué es un fichero?

Un ordenador, lo único que se es capaz de manejar es un conjunto de bits (0, 1). Estos bits se agrupan en la unidad mínima, el byte. Un byte equivale a 8 bits.



Un fichero es una archivo que contendrá un conjunto de caracteres o bytes que se almacenarán en un dispositivo en una ruta y con un nombre y extensión concretos.

En Java se sigue el concepto de un fichero como un conjunto de bytes.

Los ficheros suelen llevar asociado un nombre, una ruta de acceso y una extensión para saber de que tipo de fichero se trata. El conjunto de los 3 elementos debe ser único.

Los ficheros se guardan en dispositivos de almacenamiento fijo como discos duros, pendrives, tarjetas, CD, etc…

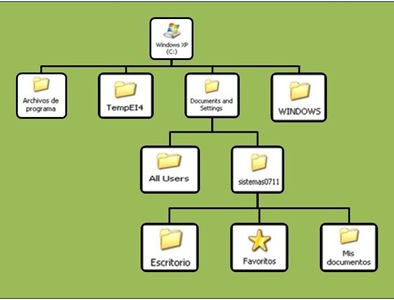
La manera en que se organizan los datos dentro de un fichero depende únicamente del diseñador de ese tipo de archivo.

NOTA: Un archivo y un fichero en el contexto de la informática son exactamente lo mismo.

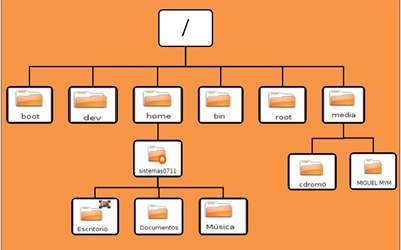
# Sistema de Ficheros

Los ficheros se suelen organizar de forma jerárquica.

* Esta imagen muestra el árbol de archivos en Windows XP:

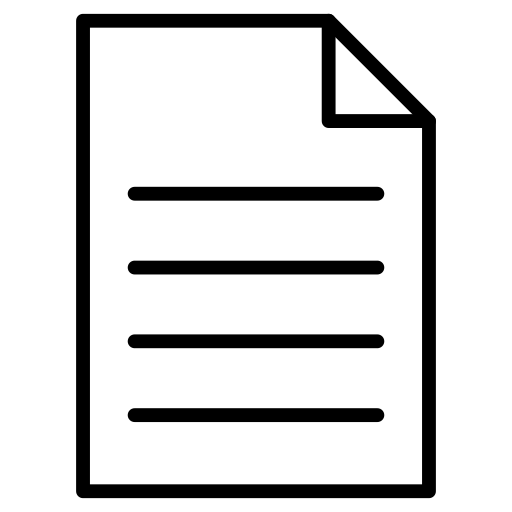
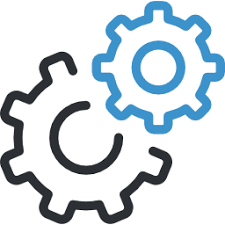


* Este otra imagen muestra el árbol de ficheros en Linux :



# Tipos de ficheros

Existen diferentes tipos de ficheros:

*  Fichero estándar: es un archivo que contiene todo tipo de datos: caracteres, imagen, audio, vídeo, etcétera. Normalmente son ficheros que contienen información de cualquier tipo.
* Directorios o carpetas: son ficheros que albergan más archivos en su interior. Su principal utilidad es mantener un orden o jerarquía en nuestros sistemas.
* Ficheros especiales: son todos esos archivos que usa nuestro sistema operativo y que se utilizan para controlar los dispositivos o periféricos de nuestro ordenador.

En este tema profundizaremos en el tipo de ficheros estándar y en los directorios. Como explicaremos más adelante, este tipo de ficheros nos permitirán realizar diferentes acciones para tratar los ficheros y para mantener un orden y jerarquía con las carpetas.  Podemos destacar dos tipos de ficheros de datos:

* Los ficheros de bytes: también conocidos como ficheros binarios, son archivos que usan los programas para leer o escribir información.
* Los ficheros de caracteres: también conocidos como ficheros de texto, nos permitirán leer o escribir la información que contengan.

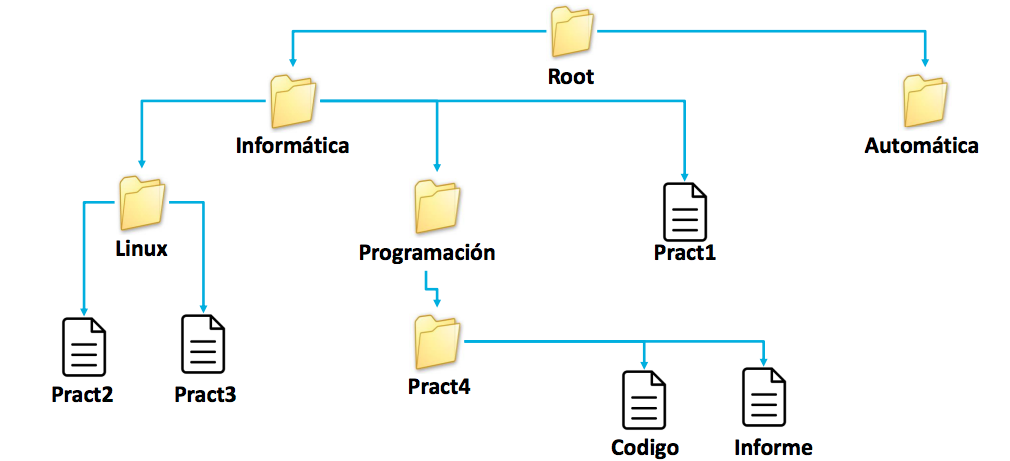
Java puede operar con ficheros de cualquier tipo.

**Rutas**

* Para acceder a un determinado fichero, se utiliza la ruta (path). Una ruta indica la dirección del fichero en el sistema de archivos.
* En una ruta, cada nivel de la jerarquía se representa delimitado por el símbolo /. En Windows, el símbolo separador es \ .
* Además de /, existen 2 elementos especiales en la ruta:
* . Un punto representa al directorio actual.
* .. Dos puntos representan al directorio padre en la jerarquía.
* Existen 2 tipos de rutas:
* Absoluta: Ruta al fichero desde el directorio principal (root). Ej: /home/Documentos/ejemplo.txt
* Relativa: Ruta al fichero desde el directorio actual. Ej: Estando en home, ./Documentos/ejemplo.txt

En Java para trabajar con ficheros, se utiliza la ruta relativa parte de la ubicación del fichero ejecutable. Pero también se permiten rutas absolutas.

**Ejemplo de árbol de directorios**

****

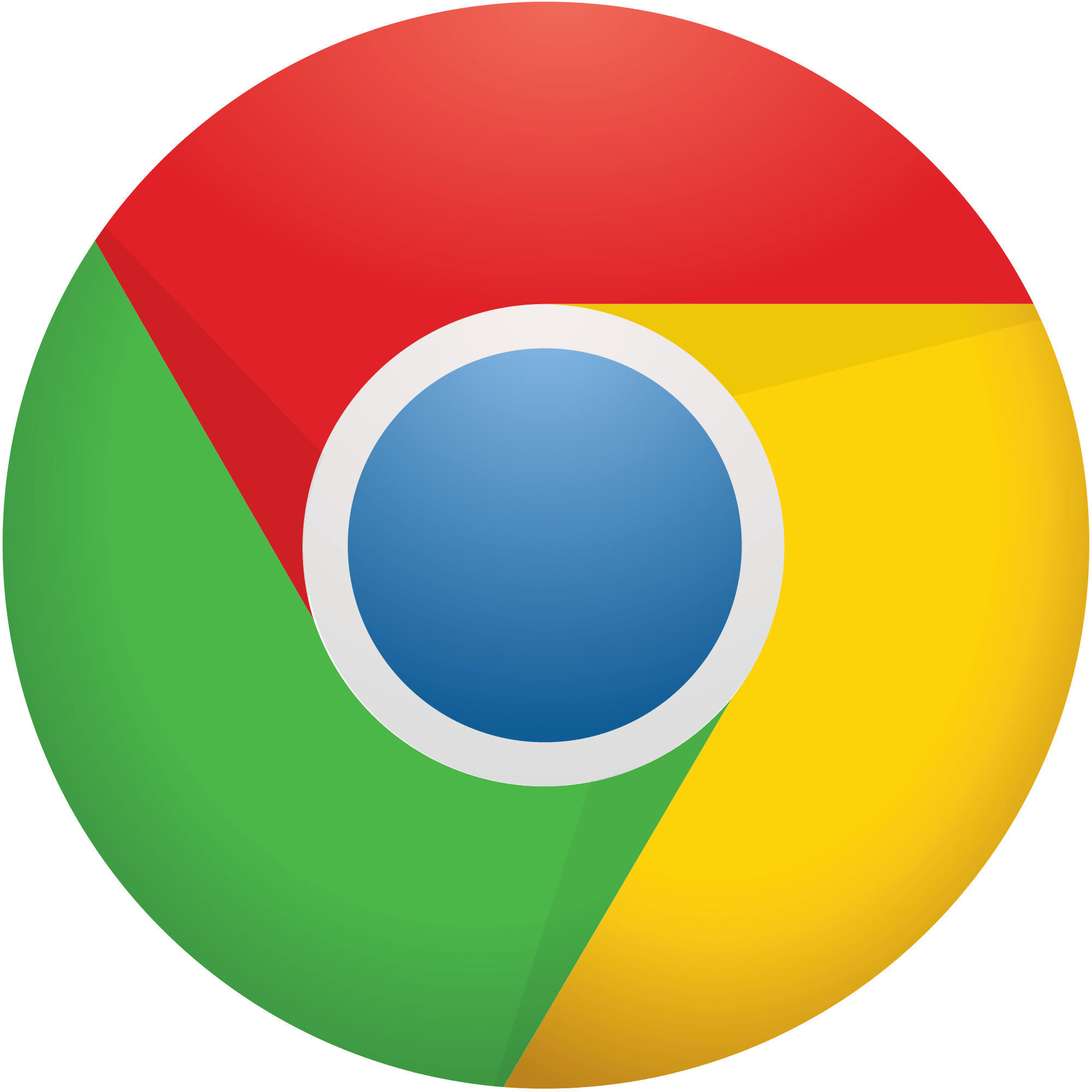
**Extensión de un fichero**

La extensión sirve para asociar un determinado fichero con un programa/s que lo ejecuten o interpreten.

* Indica cómo debe ser interpretado el fichero.



* + doc, docx, odt



* + html, xml, jsp



* + mp3, avi, wma
* Se considera extensión lo existente a partir del último punto del nombre del archivo:
  + HolaMundo.doc -> extensión =doc
  + HolaMundo.doc.mp3 -> extensión= mp3

**Permisos sobre ficheros**

El usuario que crea el fichero tiene derecho a decidir quien y cómo accede a su fichero.

Existen 3 grupos para los que se les puede definir permisos:

* u. Propietario
* g. Grupo
* o. Resto de usuarios

Los permisos que se pueden dar son los siguientes:

* r. Lectura
* w. Escritura
* x. Ejecución

**Codificaciones de caracteres**

Para poder hacer la equivalencia entre byte y símbolo (número, letra, símbolo especial, etc.) se utiliza la tabla ASCII.

La tabla ASCII permite hasta 128 símbolos, [ver tabla ASCII](https://www.ascii-code.com/es).

**Operaciones con ficheros**

* Apertura: El programa abre el fichero y se prepara para leerlo o escribirlo. Suele “reservar” el fichero para sí.
* Cierre: Indica que se ha finalizado con las operaciones sobre el fichero. Libera el fichero.
* Lectura: Lee el fichero o una de sus partes.
* Escritura: Permite escribir en el fichero, ya sea añadiendo datos o sobrescribiendo los ya existentes.
* Ejecución: Similar a la lectura, pero utiliza los datos del fichero para ejecutar un software.
* Creación: Crea un nuevo fichero con un nombre, extensión y ruta determinados.
* Eliminación: Elimina un fichero determinado.

En Java se permiten hacer todas las operaciones con ficheros vistas anteriormente.

Antes de meternos de lleno con cada una de las operaciones, tenemos que hablar de la clase FILE.

**Clase File**

La clase que manipula los ficheros en Java se llama File.

Con esta clase se pueden hacer un gran número de operaciones sobre un fichero y sus propiedades, pero no se permite leer ni escribir.

La clase File permite realizar un gran número de operaciones:

* Apertura y cierre de un fichero.
* Comprobar si el fichero existe.
* Crear un fichero.
* Borrar el fichero.
* Obtener nombre, rutas (absolutas y relativas) y extensión del fichero.
* Decir si el fichero es un fichero o un directorio.
* Obtener el tamaño en bytes del fichero.
* Consultar y cambiar los permisos del fichero.
* Si es un directorio, obtener la lista de ficheros que contiene el mismo.
* Crear nuevos directorios.
* Renombrar fichero.
* Etc.

Consulta todas la operaciones que se pueden realizar en <https://docs.oracle.com/javase/7/docs/api/java/io/File.html>

El resto de clases que manipulan ficheros parten de la existencia de una clase File, por lo que es la base de cualquier operación de manipulación de ficheros.

**Manejo de ficheros en Java.**

Un objeto de la clase File representa un fichero, apunta mediante su ruta a un fichero o directorio del sistema de archivos o también se puede crear un nuevo fichero.

* NO representa el contenido de ningún fichero.

La clase File, nos va a proporcionar una serie de utilidades, evidentemente dependiendo de la plataforma en la que nos encontremos (Linux, Mac o Windows) cambiarán las rutas de los directorios de los archivos, además en los sistemas Linux y Mac habrá que tener en cuenta que en las rutas se distinguen mayúsculas de minúsculas, cosa que no ocurre en los sistemas Windows.

La clase File, puede utilizar cualquiera de los siguientes constructores:

* **File(String path)**
* Linux:

File varFichero = new File (“/directorio/subdirectorio/fichero1.txt”);

* Windows:

File varFichero = new File(“c:\\directorio\\subdirectorio\\fichero1.txt”);

* **File(String path, String name)**
* Linux:

File varFichero= new File(“/directorio/subdirectorio”,”fichero1.txt”);

* **File(File dir, String name)**
* Linux:

File varFichero= new File(new File (“/directorio/subdirectorio”),”fichero1.txt”);

En Java el separador de archivos tanto para Windows como para Linux es el símbolo /, aunque las rutas se escriben en: Linux / y en Windows \\

Al crear el constructor, la variable varFichero es un objeto con los datos del fichero que se encuentra en la ruta pasada por parámetro, si es que existe:

* La ruta puede ser absoluta o relativa.
* Para cerrar un fichero, se usa la siguiente sentencia:

varFichero.close();

Con esta sentencia se cierra el fichero y la variable varFichero pasará a ser nula (null).

**Constantes de la clase File**

La clase File tenemos un par de constantes muy útiles, que nos servirán si usamos nuestra aplicación en varios sistemas operativos. Nos permitirá mostrar el separador entre ficheros entre rutas de cada sistema operativo, por ejemplo, para Windows, se separan las rutas con “;” y los directorios con “\”. y en Unix/Linux se separan con “:” y “/” respectivamente.

Los métodos son:

* pathSeparator
* separator

String path = File.separator + "var"+ File.separator + "temp"

String listOfFiles = ...

String[] filePaths = listOfFiles.split(File.pathSeparator);

Utilizar estas constantes nos evitará duplicar código a la hora de trabajar con ficheros dependiendo si es un sistema Windows o Unix.

**Métodos de la clase File**

A diferencia de otras clases que manejan ficheros, en la clase File no es necesario controlar excepciones en la mayoría de los métodos. Si hay alguna excepción a manejar lo indicaremos.

Estos son los métodos mas conocidos:

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Parámetros | Dato devuelto |
| Exists () | Indica si existe o no el fichero. | Ninguno. | boolean |
| isDirectory () | Indica si el objeto File es un directorio. | Ninguno. | boolean |
| isFile () | Indica si el objeto File es un fichero. | Ninguno. | boolean |
| isHidden () | Indica si el objeto File esta oculto. | Ninguno. | boolean |
| getAbsolutePath () | Devuelve una cadena con la ruta absoluta del fichero o directorio. | Ninguno. | String |
| canRead () | Indica si se puede leer. | Ninguno. | boolean |
| canWrite () | Indica si se puede escribir. | Ninguno. | boolean |
| canExecute () | Indica si se puede ejecutar. | Ninguno. | boolean |
| getName () | Devuelve una cadena con el nombre del fichero o directorio. | Ninguno. | String |
| getParent () | Devuelve una cadena con el directorio padre. | Ninguno. | String |
| listFiles () | Devuelve un array de File con los directorios hijos. Solo funciona con directorios. | Ninguno. | Array de File |
| list () | Devuelve un array de String con los directorios hijos. Solo funciona con directorios. | Ninguno. | Array de String |
| mkdir () | Permite crear el directorio en la ruta indicada. Solo se creara si no existe. | Ninguno. | boolean |
| mkdirs () | Permite crear el directorio en la ruta indicada, también crea los directorios intermedios. Solo se creara si no existe. | Ninguno. | boolean |
| createNewFile () | Permite crear el fichero en la ruta indicada. Solo se creara si no existe. Debemos controlar la excepcion con IOException. | Ninguno. | boolean |

**Creación de un filtro**

Un filtro es un objeto de una clase que implementa el interface *FilenameFilter*, y tiene que redefinir la única función del [interface](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/herencia/interfaces.htm) denominada *accept*. Esta función devuelve un dato de tipo **boolean**. En este caso, la hemos definido de forma que si el nombre del archivo termina con una determinada extensión devuelve **true** en caso contrario devuelve **false**. La función *endsWith* de [la clase *String*](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/string.htm#C%C3%B3mo%20se%20obtiene%20informaci%C3%B3n%20acerca%20del%20string) realiza esta tarea tal como se ve en la porción de código que viene a continuación. La extensión se le pasa al constructor de la clase *Filtro* para inicializar el miembro dato *extension*.

import java.io.\*;

public class Filtro implements FilenameFilter {

String extension;

Filtro(String extension){

this.extension=extension;

}

public boolean accept(File dir, String name){

return name.endsWith(extension);

}

}

Para obtener la lista de archivos con extensión .java en el directorio actual, creamos un objeto de la clase *Filtro* y se lo pasamos a la función *list* miembro de la clase *File*.

listaArchivos=fichero.list(new Filtro(".java"));

for ( int i=0; i<listaArchivos.length; i++ ) {

System.out.println(listaArchivos[i]);

}

**Práctica 1 – Crear un fichero**

Para crear un fichero en JAVA usamos la [clase File](https://docs.oracle.com/javase/7/docs/api/java/io/File.html) de Java, hay que tener en cuenta que podemos generar una [IOException](https://docs.oracle.com/javase/7/docs/api/java/io/IOException.html), por lo que habrá que gestionarla.

En el siguiente ejemplo vamos a crear un fichero llamado **practica1.java**:

import java.io.File;

import java.io.IOException;

public class fileCreate {

public static void main(String[] args) {

try {

File ficheroDatos = new File("prueba1.java"); if (ficheroDatos.createNewFile()) {

System.out.println("Fichero "+ ficheroDatos.getName() +" creado");

}

else {

System.out.println("No se ha podido crear el fichero. Probablemente ya exista.");

}

} catch(IOException error){

System.out.println("Error al crear el fichero.");

error.printStackTrace();

}

}

}

**Práctica 2 – Obtener información de un fichero**

Creamos un objeto fichero de la clase File, pasándole el nombre del archivo, en este caso, el nombre del archivo código fuente anterior practica1.java.

Si este archivo existe, es decir, si la función exists devuelve true, entonces se obtiene información acerca del archivo:

* getName devuelve el nombre del archivo
* getPath devuelve el camino relativo
* getAbsolutePath devuelve el camino absoluto.
* canRead nos indica si el archivo se puede leer.
* canWrite nos indica si el archivo se puede escribir
* length nos devuelve el tamaño del archivo, si dividimos la cantidad devuelta entre 1024 obtenemos el tamaño del archivo en KB.

File fichero=new File("practica1.java");

If ( fichero.exists() ) {

System.out.println("Nombre del archivo "+fichero.getName());

System.out.println("Camino "+fichero.getPath());

System.out.println("Camino absoluto "+fichero.getAbsolutePath());

System.out.println("Se puede escribir "+fichero.canRead());

System.out.println("Se puede leer "+fichero.canWrite());

System.out.println("Tamaño "+fichero.length());

}

**Práctica3** – Obtener lista de archivos del directorio actual

Para ello se crea un nuevo objeto de la clase *File.*

Para obtener la lista de los archivos que contiene este directorio se llama a la función miembro *list*, la cual nos devuelve un array de strings.

fichero=new File(".");

String[] listaArchivos=fichero.list();

For ( int i=0; i<listaArchivos.length; i++ )

{

System.out.println(listaArchivos[i]);

}

**Práctica 4 – Eliminar un archivo**

Para eliminar un fichero podemos usar la clase [clase File](https://docs.oracle.com/javase/7/docs/api/java/io/File.html) con su método **delete()**.

El método delete devolverá cierto si ha podido borrar el fichero y falso si no ha podido, probablemente en ese caso el fichero no existía.

En el ejemplo siguiente borramos el fichero practica1.java y avisamos al usuario si se ha podido borrar o no:

import java.io.File;

public class fileDelete {

public static void main(String[] args) {

File ficheroDatos = new File("datos.txt");

Boolean ficheroBorrado = ficheroDatos.delete();

if(ficheroBorrado){

System.out.println("Se ha borrado el fichero.");

}else{

System.out.println("No se ha borrado el fichero."); }

}

}

**Práctica 5 – Crear un directorio**

Para crear un directorio, la clase [clase File](https://docs.oracle.com/javase/7/docs/api/java/io/File.html) dispone del método **mkdir()**. En el siguiente ejemplo creamos el directorio **/home/sonia/midirectorio/**:

import java.io.File;

public class fileDirectory {

public static void main(String[] args) {

File ficheroDatos = new File("/home/profe/directorioTemporal/"); if (ficheroDatos.mkdir()) {

System.out.println("Directorio "+ ficheroDatos.getName() +" creado"); }

else {

System.out.println("No se ha podido crear el directorio. Probablemente ya exista.");

}

}

}

**Práctica 6 – Eliminar un directorio**

Es exactamente igual que eliminar un fichero pero poniendo la ruta del directorio en lugar de la de un fichero. Por ejemplo para eliminar el directorio /home/sonia/midirectorio/ escribiremos:

import java.io.File;

public class fileDelete {

public static void main(String[] args) {

File directorioDatos = new File("/home/usuario/directorioTemporal/"); Boolean directorioBorrado = directorioDatos.delete();

if(directorioBorrado){

System.out.println("Se ha borrado el directorio."); }else{

System.out.println("No se ha borrado el directorio."); }

}

}

**Práctica 7 – Escribir en un fichero**

Usaremos la clase FileWriter de Java para escribir en un fichero. En el siguiente ejemplo vamos a escribir información en el fichero **practica1.java**:

import java.io.FileWriter;

import java.io.IOException;

public class fileWrite {

public static void main(String[] args) {

try{

FileWriter escritorDeFichero = new FileWriter("practica1.java");

escritorDeFichero.write("Información.");

escritorDeFichero.close();

System.out.println("Fichero escrito con éxito.");

}catch (IOException error){

System.out.println("Error escribir en el fichero."); error.printStackTrace();

}

}

}

## Sobreescribir o añadir al final (overwrite or append):

Con FileWriter podemos escribir un fichero borrando todo lo que tenía en él utilizando el constructor FileWriter("namefile") o añadir información al final de un fichero que ya tiene algo escrito, con el método FileWriter("namefile, True).

En el ejemplo anterior si queremos escribir información a continuación de la que ya hay debemos substituir la línea:

FileWriter escritorDeFichero = new FileWriter("practica1.java");

por esta otra:

FileWriter escritorDeFichero = new FileWriter("practica1.java",true);

**Práctica 8 – Leer un fichero**

Para leer de fichero utilizaremos la clase [Scanner](https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html) de Java. Cuando leemos, si no existe el fichero, podemos provocar una excepción FileNotFoundException que debemos controlar.

El siguiente código lee la información del fichero practica1.java y la muestra por consola:

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class fileRead {

public static void main(String[] args) {

try {

File ficheroDatos = new File("practica1.java");

Scanner lectorDatos = new Scanner(ficheroDatos);

while (lectorDatos.hasNextLine()){

String datos = lectorDatos.nextLine();

System.out.println(datos);

}

lectorDatos.close();

}catch(FileNotFoundException error){

System.out.println("Error al abrir el fichero para lectura."); error.printStackTrace();

}

}

}