

Gestión de ficheros

Introducción

- Un fichero o archivo es un conjunto de bits almacenado en un dispositivo, como por ejemplo, un disco duro.
- Los ficheros tienen un nombre y una extensión.
- El nombre debe ser único en el directorio donde se encuentre el fichero.
- La extensión no es más que una convención que nos permite saber el tipo de fichero.
- Un fichero puede constar de líneas o registros o cualquier cosa; cada registro puede estar formado por campos relacionados; cada línea puede tener el texto de un impreso
- La manera en que se agrupan los datos en el fichero depende de la persona que lo diseñe.

Clases asociadas a la gestión de ficheros

- El paquete `java.io` contiene las clases para manejar entrada/salida en Java.
- La primera clase a estudiar es `FILE`
- `FILE` encapsula toda la funcionalidad necesaria para gestionar un sistema de archivos:
 - Manipulación y consulta de la estructura del sistema de archivos
 - Manipulación propiedades de los elementos del sistema de archivos
 - Gestión de permisos (dependiendo del SO)
- Cada instancia de `FILE` representa una ruta. Puede ser ruta a un fichero o a un directorio.
- La ruta de la instancia se mantiene siempre; no hay forma de modificarla
- Constructores:
 - `File (String rutaAlFichero)`
 - `File (String directorio, String nombreFichero)`
 - `File (File directorio, String nombreFichero)`

Métodos más importantes de File

- `String[] list()` → Devuelve array de `String` con los nombres de ficheros y directorios asociados al objeto `File`
- `File[] listFiles` → Devuelve array de objetos `File` conteniendo los ficheros que estén dentro del directorio representado por el objeto `File`.
- `String getName()` → Nombre del fichero
- `String getPath()` → Ruta relativa
- `String getAbsolutePath()` → Ruta absoluta
- `Boolean exists()` → Devuelve *true* si el fichero existe
- `Boolean canWrite()` → Devuelve *true* si el fichero se puede escribir
- `Boolean canRead()` → Devuelve *true* si el fichero se puede leer
- `Boolean isFile()` → Devuelve *true* si el objeto `File` corresponde a un fichero normal
- `Boolean isDirectory()` → Devuelve *true* si el objeto `File` corresponde a un directorio

- `long length()` → Devuelve el tamaño del fichero en bytes
- `Boolean mkdir()` → Crea un directorio con el nombre indicado en la creación del objeto `File`. Solo se creará si no existe.
- `Boolean renameTo (File nuevoNombre)` → Renombrar fichero
- `Boolean delete()`
- `Boolean createNewFile()` → Crea un nuevo fichero, vacío, asociado a `File` si y solo si no existe un fichero con ese nombre.
- `String getParent()` → Devuelve el nombre del directorio padre o `null` si no existe.

Ejemplo

Muestra la lista de ficheros en el directorio actual

```
import java.io.*;
public class VerDir{
    public static void main (String[] args) {
        String dir = "."; // directorio actual
        File f = new File(dir);
        String[] archivos = f.list();
        System.out.printf("Ficheros en el directorio actual: %d %n", archivos.length);
        for (int i=0; i<archivos.length; i++){
            File f2 = new File(f, archivos[i]);
            System.out.printf("Nombre: %s, es fichero?: %b, es directorio?:%b %n", archivos[i],
                               f2.isFile(), f2.isDirectory());
        }
    }
}
```

Problemas

- Ejercicio 1: Cambia la ruta del ejemplo anterior. Utiliza una ruta absoluta a tu carpeta de descargas, por ejemplo.
- Ejercicio 2: Ahora haz los cambios necesarios para que el programa anterior muestre los ficheros del directorio introducido desde línea de comandos al ejecutar el programa
- Ejercicio 3: Realiza un programa Java que utilice el método `listFiles()` para mostrar la lista de ficheros de un directorio que se pasará al programa desde los argumentos del *main*
- Ejercicio 4: Añade al programa anterior las instrucciones necesarias para que envíe un mensaje de error en caso de que el directorio pasado como argumento no exista.
- Ejercicio 5: Realiza un programa Java que muestre la siguiente información de un fichero cualquiera: Nombre, ruta relativa, ruta absoluta, permisos y tamaño.

Ejemplo

Veamos otro ejemplo para la creación/eliminación de archivos

```
import java.io.*;
public class CrearDir{
    public static void main(String[] args){
        File d=new File("NuevoDir");
        File f1= new File(d,"Fichero1.txt");
        File f2= new File (d,"Fichero2.txt");
        d.mkdir();
        try {
            if (f1.createNewFile())
                System.out.println("Fichero1 creado correctamente");
            else
                System.out.println("No se ha podido crear Fichero1");
            if (f2.createNewFile())
                System.out.println("Fichero2 creado correctamente");
            else
                System.out.println("No se ha podido crear Fichero2");
        } catch (IOException ioe) {ioe.printStackTrace();}
```



```
f1.renameTo( new File(d,"Fichero1Nuevo"));  
try {  
    File f3 = new File ("NuevoDir/Fichero3.txt");  
    f3.createNewFile();  
}catch (IOException ioe) {ioe.printStackTrace();}
```

- El método `createNewFile()` puede lanzar la excepción `IOException`, por ello hemos utilizado el bloque `try-catch`

Problemas

- Ejercicio 6: Copia el ejemplo anterior y ejecútalo
- Ejercicio 7: Realiza un programa que elimine el directorio creado en el punto anterior. Para ello habrás de eliminar todos los archivos que se encuentren dentro del directorio.