Ficheros

1. Clase File
2. FileReader y BufferedReader
3. FileWriter y BufferedWriter
4. DataOuputStream y DataInputStream (primitivos)
5. ObjectOutputStream y DataOutputStream (objetos)

Clase File

La clase File se utiliza para representar rutas dentro del sistema de archivos, esto quiere decir que pueden ser tanto archivos como directorios.

Podremos manipular tanto archivos como directorios con sus metodos al igual que comprobar su existencia, tamaño, saber permisos...

Asi podemos instanciar un objeto de la clase file:

```
File f_abs = new File("/Ruta/absoluta/archivo1.txt");    // Ruta absoluta
File f_rel = new File("archivo1.txt")                    // Ruta relativa
```

Metodos

Nombre	Tipo retorno	Descripcion
isDirectory()	boolean	True si es directorio
isFile()	boolean	True si es archivo
exist()	boolean	True si existe
delete()	boolean	Borrar el fichero o directorio
renameTo(File)	boolean	Renombrar el fichero o directorio
canRead()	boolean	True si tenemos permisos de lectura
canWrite()	boolean	True si tenemos permisos de escritura
getPath()	String	Devuelve la ruta relativa del archivo
getAbsolutePath()	String	Devuelve la ruta absoluta del archivo
getName()	String	Devuelve el nombre del archivo
getParent()	String	Devuelve el directorio padre
length()	long	Devuelve el tamaño del archivo en bytes
mkdir()	boolean	Crea el directorio en la ruta en que se ha instanciado
list()	String[]	Devuelve un array con el nombre del contenido de la carpeta
list(String regex)	String[]	Devuelve un array como el anterior pero filtrando con la expresion regular
listFiles()	File[]	Como si gastaramos list() pero en vez de Strings con objetos de tipo File

Metodos como delete() y mkdir() que nos devuelven un booleano, esto nos puede ser util si queremos saber si la operacion se ha ejecutado correctamente. Por ejemplo si hacemos delete a un archivo que no existe no nos dara ningun error a priori, pero nos devolvera false. Esto puede que sea util en algun momento.

FileReader y BufferedReader

Estas dos clases de java se utilizan para leer archivos. Con ambas podemos hacer exactamente lo mismo, lo unico que las diferencia es en terminos de rendimiento y funcionalidad. La unica diferencia entre ellas es que BufferedReader es mas eficiente por que hace menos operaciones de lectura sobre el archivo.

FileReader

Para instanciar un FileReader deberemos pasarle una instancia de la clase File como parametro al constructor. Ojo, el archivo ya tiene que estar creado, si no lanzara una excepcion del tipo ***FileNotFoundException***.

```
File f = new File("archivo.txt");
FileReader reader = new FileReader(f);
```

o tambien

```
FileReader reader = new FileReader(new File("archivo.txt"));
```

Para poder leer el archivo con FileReader deberemos usar su metodo read(). Este metodo nos devuelve un entero con el valor del caracter por lo que deberemos convertirlo a char, si llega al final del archivo nos devolvera -1. Hay que tener en cuenta que si se produce un error de entrada/salida, lanzara una excepcion de tipo ***IOException***.

Con esta informacion se puede deducir como se lee un archivo entero con esta clase:

```
try {
    FileReader reader = new FileReader(new File("archivo.txt"));

    int caracterActual = reader.read();
    while (caracterActual != -1) {
        System.out.print((char) caracterActual);
        caracterActual = reader.read();
    }

    reader.close()       // Cerrar el archivo

} catch (FileNotFoundException e) {
    System.out.println("El archivo no existe");

} catch (IOException e) {
    System.out.println("Ocurrio un error de entrada salida");
}
```

Podemos omitir el cerrar el archivo instanciandolo justo despues del try entre parentesis de la siguiente manera:

```
try (FileReader reader = new FileReader(new File("archivo.txt"))) {
    // Las operaciones que tengamos que hacer

} // Manejo de excepciones
```

Tambien se pueden declarar y asignar varias variables dentro del parentesis

```
try (FileReader reader = new FileReader(new File("archivo.txt");
    Object otraVariable = new Object())) {
    // Las operaciones que tengamos que hacer
```

```
} // Manejo de excepciones
```

De ahora en adelante lo hare asi

BufferedReader

Para poder instanciar un objeto de esta clase lo que se hace es *envolver* una instancia de la clase FileReader.

```
File f = new File("archivo.txt");
FileReader fr = new FileReader(f);
BufferedReader reader = new BufferedReader(fr);
```

o tambien

```
File f = new File("archivo.txt");
BufferedReader reader = new BufferedReader(new FileReader(f));
```

Con una instancia de esta clase podemos leer el archivo como lo leiamos en FileReader pero lo interesante de esto es que nos proporcion un metodo que lee una linea entera, readLine(), que devolvera null cuando no queden mas lineas. Si ocurre un error de entrada/salida nos lanzara la excepcion ***IOException*** al igual que si no existe el archivo lanzara ***FileNotFoundException***.

Asi se lee un archivo con el metodo readLine():

```
File f = new File("archivo.txt");
try (BufferedReader reader = new BufferedReader(new FileReader(f))) {

    String lineaActual = reader.readLine();
    while (lineaActual != null) {
        System.out.println(lineaActual);
        lineaActual = reader.nextLine();
    }

    } catch (FileNotFoundException e) {
        System.out.println("El archivo no existe");

    } catch (IOException e) {
    System.out.println("Ocurrio un error de entrada salida");
    }
```

FileWriter y BufferedWriter

Similares a FileReader y BufferedReader. Estas dos clases de java se utilizan para escribir archivos. Con ambas podemos hacer exactamente lo mismo, lo unico que las diferencia es en terminos de rendimiento y funcionalidad. La unica diferencia entre ellas es que BufferedWriter es mas eficiente por que hace menos operaciones de escritura sobre el archivo.

FileWriter

Para instanciar un objeto de la clase FileWriter deberemos pasarle un objeto de la clase File y un booleano, dependiendo de si queremos sobreescribir el archivo o no. Ademas, si el archivo File no existe, al realizar una operacion de escribir, este se creara automaticamente.

```
File f = new File("mi_archivo");
FileWriter writer = new FileWriter(f, true);    // se añade al final, false sobreescribe
```

o tambien

```
FileWriter writer = new FileWriter(new File("mi_archivo"), true);    // se añade al final, false sobreescribe
```

Con la clase una vez instanciada podemos empezar a escribir en el archivo con el metodo write(). Si queremos introducir un salto de linea deberemos introducir la secuencia de escape ***\n*** en el String que introduzcamos. Hay que tener en cuenta que al escribir nos podra lanzar ***IOException***, pero no FileNotFoundException ya que se crea si no existe:

```
try (FileWriter writer = new FileWriter(new File("mi_archivo"), true)) {
    writer.write("he metido esta linea desde java!\n");
    writer.write("otra linea");

} catch (IOException e) {
    System.out.println("Ocurrió un error al escribir en el archivo: " + e.getMessage());
}
```

BufferedWriter

Para instanciar un objeto de esta clase debemos envolver una instancia de FileWriter:

```
File f = new File("mi_archivo");
FileWriter fw = new FileWriter(f, true);    // se añade al final, false sobreescribe
BufferedWriter writer = new BufferedWriter(fw);
```

o tambien

```
File f = new File("mi_archivo");
BufferedWriter writer = new BufferedWriter(new FileWriter(f, true))
```

Para poder escribir en el archivo se hace de la misma manera que en FileWriter, la unica diferencia es que BufferedWriter nos proporciona un metodo llamado newLine() que inserta una linea, por lo tanto no tenemos que poner la secuencia de escape.

```
File f = new File("mi_archivo");
try (BufferedWriter writer = new BufferedWriter(new FileWriter(f, true))) {
    writer.write("he metido esta linea desde java!");
    writer.newLine();
    writer.write("otra linea");

} catch (IOException e) {
    System.out.println("Ocurrió un error al escribir en el archivo: " + e.getMessage());
}
```

DataInputStream y DataOutputStream

Para poder usar estas estas clases debemos conocer lo que es ***FileInputStream*** y ***FileOutputStream***. Estas dos clases lo que nos permiten es leer y escribir bits en un archivo binario (es un File pero no legible). Se declaran asi:

```
File f1 = new File("archivo_leer");
FileInputStream fis = new FileInputStream(f1);

File f2 = new File("archivo_escribir");
FileOutputStream fos = new FileOutputStream(f2, true);  // sobreescribe
```

Ademas ***FileInputStream*** nos proporciona un metodo que nos dice el numero de bytes estimado que quedan por leer, este es **available**. Esto nos sera util para poder leer todo el archivo.

```
File f1 = new File("archivo_leer");
FileInputStream fis = new FileInputStream(f1);

while (fis.available() > 0) {
    // lo que tengamos que leer
}
```

Por que debemos saber esto, por que ***DataInputStream*** y ***DataOutputStream*** lo que nos ofrece es un nivel de abstraccion mayor. A grandes rasgos en vez de escribir bits directamente escribiremos el tipo primitivo dependiendo del metodo que usemos, aunque estos tambien se guardaran en bits, en formato UTF-8.

Tambien hay que decir que estos estan dirigidos a leer y escribir tipos primitivos: int, long, boolean, double... y Strings (que no es un tipo primitivo, pero tambien nos lo permite).

Tambien es necesario saber que para instanciar cualquiera de estos dos objetos tenemos que envolver el FileInputStream o FileOutputStream en su respectivo. Es decir:

```
File f1 = new File("archivo_leer");
FileInputStream fis = new FileInputStream(f1);
DataInputStream reader = new DataInputStream(fis);

File f2 = new File("archivo_escribir");
FileOutputStream fos = new FileOutputStream(f2, true);  // sobreescribe
DataOutputStream writer = new DataOutputStream(fos);
```

o tambien

```
File f1 = new File("archivo_leer");
DataInputStream reader = new DataInputStream(new FileInputStream(f1));

File f2 = new File("archivo_escribir");
DataOutputStream writer = new DataOutputStream(new FileOutputStream(f2, true));
```

Antes de saber como leer y escribir con estas clases hay que tener claro que esto es para leer y escribir informacion estructurada. Por ejemplo, nosotros creamos un archivo binario en el que guardamos, con ***DataOutputStream***, nombre(String), edad(int) y si es mayor de edad(boolean), vamos guardando en ese orden: String, int, boolean, String, int, boolean... Para luego cuando recuperemos, con ***DataInputStream***, poder leerlos con ese orden.

DataInputStream

Los metodos que nos proporciona esta clase son:

Metodo	Descripcion
readBoolean()	Devuelve un boolean
readByte()	Devuelve un byte
readChar()	Devuelve un char
readShort()	Devuelve un short
readInt()	Devuelve un int

readLong()	Devuelve un long
readFloat()	Devuelve un float
readDouble()	Devuelve un double
readUTF()	Devuelve un String codificado en UTF-8

Aqui un ejemplo de como usarlo:

```
File f1 = new File("archivo");
try (DataOutputStream reader = new DataInputStream(new FileInputStream(f1))) {
    String nombre = reader.readUTF("juanda");
    int edad = reader.readInt(23);
    boolean esMayor = reader.readBoolean(true);
    // do something

} catch (FileNotFoundException e) {
    System.out.println("No existe el archivo");

} catch (IOException e) {
    System.out.println("Ocurrio un error de entrada/salida");
}
```

o en un bucle:

```
File f1 = new File("archivo");
try (FileInputStream fis = new FileInputStream(f1);
     DataInputStream reader = new DataInputStream(fis)) {

    while (fis.available() > 0) {
        String nombre = reader.readUTF();
        int edad = reader.readInt();
        boolean esMayor = reader.readBoolean();
        // hacer algo con los datos leídos
    }

} catch (FileNotFoundException e) {
    System.out.println("No existe el archivo");
} catch (IOException e) {
    System.out.println("Ocurrió un error de entrada/salida");
}
```

DataOutputStream

Los metodos que nos proporciona esta clase son:

Metodo	Descripcion
writeBoolean(boolean)	Escribe un boolean
writeByte(byte)	Escribe un byte
writeChar(char)	Escribe un char
writeShort(short)	Escribe un short

writeInt(int)	Escribe un int
writeLong(long)	Escribe un long
writeFloat(float)	Escribe un float
writeDouble(double)	Escribe un double
writeUTF(String)	Escribe un String codificado en UTF-8

Aqui un ejemplo de como usarlo:

```
File f = new File("archivo");
try (DataOutputStream writer = new DataOutputStream(new FileOutputStream(f, true))) {
    writer.writeUTF("juanda");
    writer.writeInt(23);
    writer.writeBoolean(true);

} catch (IOException e) {
    System.out.println("Ocurrio un error de entrada/salida");
}
```

ObjectInputStream y ObjectOutputStream

Estas dos clases son muy similares a las dos anteriores, solo que en vez de ir dirigidas a tipos primitivos van dirigidas a los objetos que nosotros creemos. Como en las dos clases anteriores tambien hay que *envolver* FileInputStream y FileOutputStream en su respectivo, ya que escribira los bits, pero nos dara el nivel de abstraccion en el cual nosotros solo nos preocupamos de los objetos.

Se instancian de la siguiente manera:

```
File f1 = new File("file1");
FileOutputStream fos = new FileOutputStream(f1);
ObjectOutputStream writer = new ObjectOutputStream(fos);

File f2 = new File("file2");
FileInputStream fis = new FileInputStream(f2);
ObjectInputStream reader = new ObjectInputStream(fis);
```

o tambien

```
File f1 = new File("file1");
ObjectOutputStream writer = new ObjectOutputStream(new FileOutputStream(f1));

File f2 = new File("file2");
ObjectInputStream reader = new ObjectInputStream(new FileInputStream(f2));
```

Para indicar al compilador que la clase que tenemos se va a poder serializar (convertir un objeto en una secuencia de bytes, que son los que se guardaran) simplemente tenemos que implementar la interfaz funcional (interfaz que no tiene ningun metodo) ***Serializable*** del paquete ***java.io***. Con una clase persona quedaria asi:

```
import java.io.Serializable;

public class Persona implements Serializable {
    public String nombre;
    public int edad;
```

```
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

ObjectOutputStream

Para poder guardar en un fichero un objeto, teniendo declarada la clase, que usaremos Persona, se hace de la siguiente manera, con el metodo writeObject:

```
File f = new File("persona");

try (ObjectOutputStream writer = new ObjectOutputStream(new FileOutputStream(f))) {
    Persona yo = new Persona("juanda", 23);      // Creamos la persona
    writer.writeObject(yo);                      // La guardamos

} catch (IOException e) {
    System.out.println("Ocurrio un error de entrada/salida");
}
```

ObjectInputStream

Para poder leer de un fichero un objeto se hace de la siguiente manera, utilizando el metodo readObject (que lanzara ***ClassNotFoundException*** si no se ha declarado la clase) y casteando al tipo de objeto que es:

```
File f = new File("persona");

try (ObjectInputStream reader = new ObjectInputStream(new FileInputStream(f))) {
    Persona p = (Persona) reader.readObject();      // Se lee objeto y se castea
    System.out.println("Nombre: " + p.nombre + ", edad: " + p.edad) // Nombre: juanda, edad: 23

} catch (FileNotFoundException e) {
    System.out.println("No existe el archivo");

} catch (IOException e) {
    System.out.println("Ocurrio un error de entrada/salida");

} catch (ClassNotFoundException e) {
    System.out.println("La clase no se encuentra");
}
```

o en bucle:

```
File f = new File("persona");

try (FileInputStream fis = new FileInputStream(f);
     ObjectInputStream reader = new ObjectInputStream(fis)) {

        while (fis.available() > 0) {
            Persona p = (Persona) reader.readObject();      // Se lee objeto y se castea
            // hacer algo con los datos leidos
        }
```

```
} catch (FileNotFoundException e) {
    System.out.println("No existe el archivo");

} catch (IOException e) {
    System.out.println("Ocurrio un error de entrada/salida");

} catch (ClassNotFoundException e) {
    System.out.println("La clase no se encuentra");
}
```