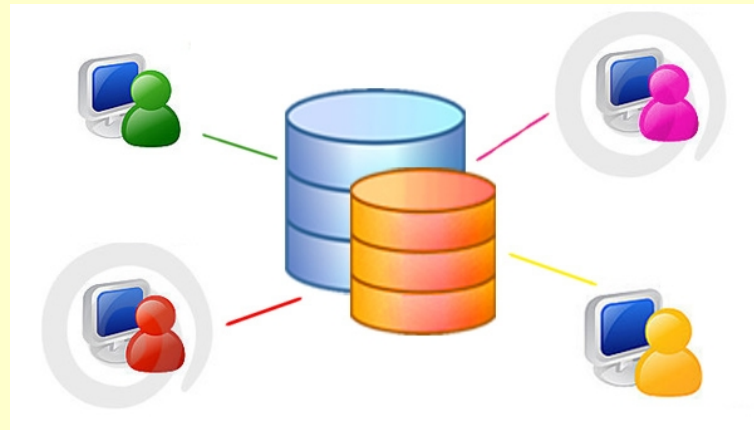


Ficheros y entrada salida en JAVA



INTRODUCCIÓN

- Para la interacción con ficheros, y entrada y salida de datos, Java cuenta con su propia API, agrupada en el paquete `java.io`
- En este tema, repasaremos cuáles son las más importantes para trabajar con ellas

CLASES

- Las clases más importantes son:
 - File
 - FileInputStream, FileOutputStream
 - FileReader, FileWriter
 - ObjectOutputStream, ObjectInputStream
 - BufferedReader, BufferedWriter

File

- La clase File nos permite obtener información sobre los directorios y archivos de nuestro sistema de ficheros
- Nos permitirá crear y eliminar archivos dentro de nuestro sistema de ficheros
- Pero no nos permitirá leer o escribir en ningún archivo o directorio

File

- Los métodos más importantes de la clase File son:
 - `public File (String ruta) //constructor`
 - `boolean exists ()`
 - `boolean isDirectory()`
 - `boolean isFile()`
 - `String[] list () //obtenemos el listado del directorio`
 - `boolean createNewFile ()`
 - `boolean delete()`

Práctica 1

- Crear una carpeta en la raíz de nuestro proyecto Java dentro de Eclipse e introducir algunos ficheros. Hacer después un programa, para que liste el directorio y los ficheros
- Hacer un programa para que desde el directorio actual (“.”), recorra el arbol de directorios y muestre todos los directorios, subdirectorios y sus respectivos archivos

Stream vs Reader

- Cuando necesitemos procesar ficheros binarios (fotos, ejecutables, audio), debemos emplear clases que hereden de InputStream (FileInputStream) **Estamos leyendo bytes**
- Cuando necesitemos procesar ficheros de texto, debemos utilizar clases que hereden de Reader (FileReader), ya que al leer un byte, automáticamente se va a traducir como char, según la codificación del SO. **Leemos caracteres**

FileInputStream

- Es la clase que emplearemos cuando necesitemos leer archivos binarios
- Métodos destacados
 - `FileInputStream (String ruta)`
 - `FileInputStream (File file)`
 - `int read (byte [] b) //lee b.length bytes`
 - `int read () //lee un byte`
 - `getFD() // obtenemos el objeto FileDescriptor`

FileOutputStream

- Es la clase que emplearemos cuando necesitemos escribir archivos binarios
- Métodos destacados
 - `FileOutputStream (String ruta)`
 - `FileOutputStream (File file)`
 - `int write (byte [] b) //escribe b.length bytes`
 - `int read () //escribe un byte`
 - `getFD() // obtenemos el objeto FileDescriptor`

FileReader

- Es la clase que emplearemos cuando necesitemos leer archivos de caracteres
- Métodos destacados
 - `FileReader (String ruta)`
 - `FileReader (File file)`
 - `int read ()` // heredada de `InputStreamReader`

FileWriter

- Es la clase que emplearemos cuando necesitemos escribir archivos de caracteres
- Métodos destacados
 - `FileWriter (String ruta, boolean añadir)`
 - `FileWriter (File file, boolean añadir)`
 - `int write (String cadena int inicio int final) //`
heredada de `OutputStreamWriter`

Buffers

- Aunque FileReader y FileWriter nos permiten leer y escribir en un fichero, existe un método mucho más eficiente y empleado, que consiste en emplear buffer de datos, para eliminar accesos a disco
- Además, nos proporcionan una interfaz más cómoda, con unos métodos más potentes

BufferedReader

- Instanciaremos un objeto de esta clase, pasándole un FileReader

```
BufferedReader br = new  
BufferedReader (objetoFileReader)
```

Y ya, podremos usar métodos de BR, como

- `br.readLine()` //que nos lee una línea de golpe (devuelve null si llegamos al final)

BufferedWriter

- Análogo a la clase anterior, Instanciaremos un objeto de esta clase, pasándole un `FileWriter`

```
BufferedWriter bw = new  
BufferedWriter(objetoFileWriter)
```

Y ya, podremos usar métodos de BW, como

- `bw.write(String cadena)`
- `bw.newLine();` añadimos un intro

Cerrar los ficheros

- Empleemos o no un buffer, es importantísimo que ejecutemos el método `close()` que proporciona cada una de la clases anteriores, para liberar correctamente los recursos. Aquí no podemos esperar al Garbage Collector
- Se recomienda incluir la instrucción `close` dentro de un bloque `finally`, como vemos en el siguiente código:

Cerrar los ficheros

```
try {  
    // Abrimos y creamos el buffer lectura del fichero y creacion de BufferedReader para  
    // poder  
    BufferedReader br = new BuferedReader(new FileReader(new File ("C:\\archivo.txt")));  
    String linea  
    // Leemos el fichero  
    while(null!=(linea=br.readLine()))  
        System.out.println(linea);  
}  
catch(Exception e){  
    e.printStackTrace();  
}finally{ //Se haya producido o no una excepción, finally se ejecutará, de modo que  
    nos //aseguramos de cerrar el fichero  
        try{  
            if( null != br ){  
                br.close();  
            }  
        }catch (Exception e2){  
            e2.printStackTrace();  
        }  
}
```


PRÁCTICA 2

Hacer una clase para almacenar en memoria un fichero de texto.

Concretamente, en un array de Strings

Además, la clase debe aportar otro método, para volcar el contenido en memoria a un fichero de texto.

Serialización

Serializar un objeto en Java, es una capacidad que ofrece el lenguaje para poder escribir Objetos en disco y después recuperarlos cómodamente, así como poder transmitirlos o almacenarlos en una base de datos.

Serializar = escribir objetos

Serialización

Para permitirlo, tan sólo debemos indicar que la clase que queremos serializar, implementen la interfaz **Serializable** (que no tiene ningún método)

Si tenemos una clase, que a su vez, tiene atributos que son una clase, **todas las clases afectadas**, deben implementar **Serializable**

Object Stream

ObjectOutputStream es la clase, que va a permitirnos escribir objetos, mientras que **ObjectInputStream**, es la clase que va a permitirnos leer objetos previamente serializados

Object Stream

Los métodos **ObjectOutputStream** que emplearemos son:

- writeObject (objeto)
- close();

Y de la clase **ObjectInputStream**:

- (Clase)readObject();//hacemos un Casting!
- close();

Object Stream

Nuestros primeros ejemplos, van a consistir en escribir objetos en ficheros.

Ya que no vamos a escribir texto, sino objetos, es decir, bloques de bytes, emplearemos las clases **FileInputStream** y **FileOutputStream** junto con **ObjectOutputStream** y **ObjectInputStream** para lograr nuestra serialización en ficheros

Serializar

ejemplo de código:

```
ObjectOutputStream salida=new  
ObjectOutputStream(new  
FileOutputStream("media3.dat"));  
salida.writeObject (persona); salida.close();  
ObjectInputStream entrada =new  
ObjectInputStream (new  
FileInputStream("media3.dat"));  
persona = (Persona)entrada.readObject();  
entrada.close();
```

Práctica 3

Serliazar ListaPersonas, escribiendo primero y recuperando después, una lista de personas creada previamente.

Una vez recuperado el objeto, llamar a mostrarListaPersonas para comprobar su correcto funcionamiento