

# Programación

## UD 10: Lectura y escritura de información

---

# 1.- Aplicaciones de almacenamiento de información en ficheros

# 1.- Aplicaciones de almacenamiento

---

¿Has pensado la **diversidad de ficheros** que existe, según la información que se guarda?

Las **fotos** que haces con tu cámara digital, o con el móvil, se guardan en ficheros. Así, existe una gran cantidad de ficheros de imagen, según el método que usen para guardar la información. Por ejemplo, tenemos los ficheros de extensión: jpg, tiff, fig, bmp, etc.

La **música** que oyes en tu mp3 o en el reproductor de mp3 de tu coche, está almacenada en ficheros que almacenan la información en formato mp3.

Los **sistemas operativos**, como Linux o Windows, están constituidos por un montón de instrucciones e información que se guarda en ficheros.

El propio código fuente de los **lenguajes de programación**, como Java, C, etc. se guarda en ficheros de texto plano la mayoría de veces.

También se guarda en ficheros las **películas** en formato avi, mp4, etc.

Y por supuesto, se usan mucho actualmente los **ficheros XML**, que al fin y al cabo son ficheros de texto plano, pero que siguen una estructura determinada.

---

## 2.- Ficheros de datos. Registros

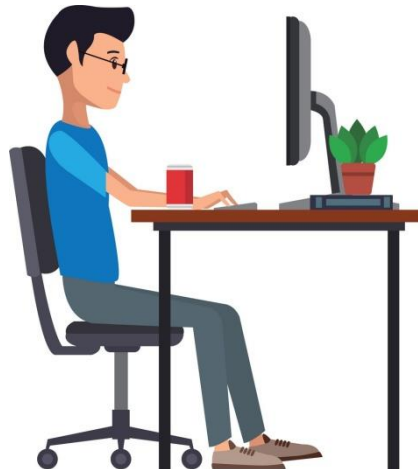
## 2.- Ficheros de datos. Registros

---

En **Java** no se impone una estructura en un fichero, por lo que conceptos como el de registro que si existen en otros lenguajes, en principio **no existen** en los archivos que se crean con Java.

Por tanto, los programadores deben estructurar los ficheros de modo que cumplan con los requerimientos de sus aplicaciones.

Así, **el programador** definirá su registro con el número de bytes que le interesen, moviéndose luego por el fichero teniendo en cuenta ese tamaño que ha definido.



## 2.- Ficheros de datos. Registros

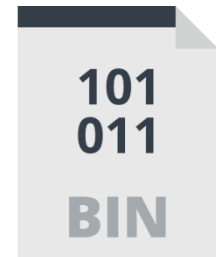
---

Distinguimos dos tipos de ficheros: los de **texto** y los **binarios**.

En los **ficheros de texto** la información se guarda como caracteres. Esos caracteres están codificados en **Unicode**, o en **ASCII** u otras codificaciones de texto.

Los **ficheros binarios** almacenan la información en bytes, codificada en binario, pudiendo ser de cualquier tipo: fotografías, números, letras, archivos ejecutables, etc.

Los archivos binarios guardan una representación de los datos en el fichero. O sea que, cuando se guarda texto no se guarda el texto en sí, sino que se guarda su representación en **código UTF-8**.



---

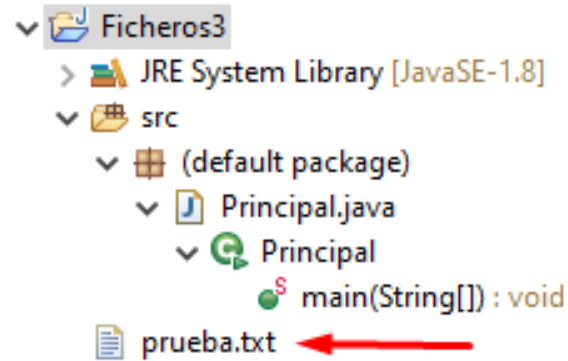
## 3.- Creación y eliminación de ficheros

# 3.- Creación y eliminación de ficheros

## Creacion de ficheros

```
public static void main(String[] args){  
  
    //Creación del objeto File, pero aun no existe en el sistema de archivos.  
    File Archivo = new File("prueba.txt");  
    try {  
        //Se crea fisicamente en el sistema de archivos  
        Archivo.createNewFile();  
    }catch (IOException e) {  
        System.out.println(e.getMessage());  
    }  
  
}
```

Si no se especifica ninguna ruta, el archivo se creará en la carpeta del proyecto:





# 3.- Creación y eliminación de ficheros

---

## El método `createNewFile`

El método que utilizamos para la creación de ficheros devuelve `TRUE` si el fichero se ha creado y `FALSE` si no se creado **porque ya existía**, por lo que podríamos completar el código anterior del siguiente modo:

```
public static void main(String[] args){  
  
    //Creación del objeto File, pero aun no existe en el sistema de archivos.  
    File Archivo = new File("prueba.txt");  
    try {  
        if (Archivo.createNewFile()) {  
            System.out.println("El archivo ha sido creado");  
        }else {  
            System.out.println("El archivo ya existia");  
        }  
    } catch (IOException e) {  
        System.out.println(e.getMessage());  
    }  
}
```

1ª ejecución: `<terminated> Principal (25) [Java Application] C:\Program Files\JAVA\`  
El archivo ha sido creado

2ª ejecución: `<terminated> Principal (25) [Java Application] C:\Program Files\JAVA\`  
El archivo ya existia

# 3.- Creación y eliminación de ficheros

---

## El método delete

El método delete devuelve TRUE si el fichero se pudo borrar con éxito y FALSE si no se pudo borrar **porque alguien ya lo ha borrado o se ha cambiado su ubicación o permanece abierto.**

```
//Creación del objeto File, pero aun no existe en el sistema de archivos.  
File Archivo = new File("prueba.txt");  
try {  
    if (Archivo.createNewFile()) {  
        System.out.println("El archivo ha sido creado");  
    }else {  
        System.out.println("El archivo ya existia");  
    }  
} catch (IOException e) {  
    System.out.println(e.getMessage());  
}  
  
boolean resultado_borrar = Archivo.delete();  
if (resultado_borrar) {  
    System.out.println("El archivo fue borrado");  
}else {  
    System.out.println("El archivo no pudo ser borrado");  
}
```

---

## 4.- Lectura y escritura de ficheros.

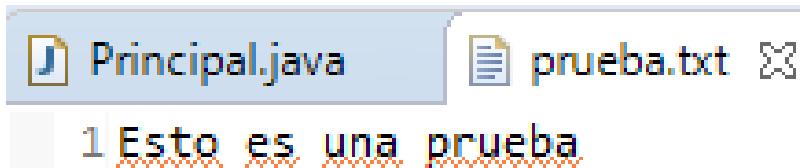
# 4.- Lectura y escritura de ficheros

La clase `FileWriter` para escribir en ficheros

```
//Creación del objeto File, pero aun no existe en el sistema de archivos.
File Archivo = new File("prueba.txt");
try {
    Archivo.createNewFile();
} catch (IOException e) {
    System.out.println(e.getMessage());
}

//FileWriter - clase que me permite escribir en un archivo a través de su método write
try {
    FileWriter file = new FileWriter(Archivo);
    file.write("Esto es una prueba\n");
    file.close();
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

Es muy importante cerrar el fichero con el método **close**, ya que sino, NO veremos los datos escritos en el fichero.




# 4.- Lectura y escritura de ficheros

La clase `FileWriter` para escribir varias líneas en un fichero

```
//FileWriter - clase que me permite escribir en un archivo a través de su método write
try {
    FileWriter file = new FileWriter(Archivo);
    for (int i=0; i < 10; i++) {
        file.write("Esto es una prueba" + i + "\n");
    }
    file.close();
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

En determinadas ocasiones, es interesante conocer el tamaño del archivo después de escribir, para lo que utilizaremos el método **`Archivo.length()`** que devolverá el tamaño del archivo en bytes .



```
1 Esto es una prueba0
2 Esto es una prueba1
3 Esto es una prueba2
4 Esto es una prueba3
5 Esto es una prueba4
6 Esto es una prueba5
7 Esto es una prueba6
8 Esto es una prueba7
9 Esto es una prueba8
10 Esto es una prueba9
```

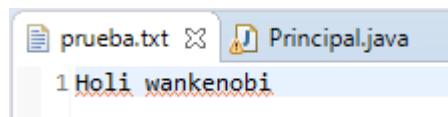
# 4.- Lectura y escritura de ficheros

La clase `FileWriter` para añadir líneas a un fichero ya creado

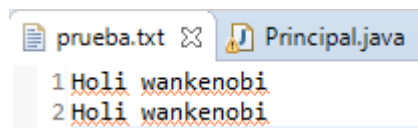
Si le pasamos un segundo parámetro a “true” en la creación de un objeto `FileWriter`, los datos siempre se añadirán al final del fichero, de modo que NO machacamos el contenido de anteriores ejecuciones.

```
try {  
    FileWriter file = new FileWriter(Archivo, true);  
    file.write("Holi wankenobi\n");  
    file.close();  
} catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

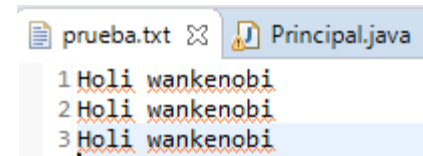
1ª Ejecución



2ª Ejecución



3ª Ejecución



# 4.- Lectura y escritura de ficheros

---

La clase Scanner para leer de ficheros

Utilizaremos el método hasNext() para leer hasta final de fichero

```
try {  
    Scanner sc = new Scanner(Archivo);  
    while (sc.hasNext()) {  
        String linea = sc.nextLine();  
        System.out.println(linea);  
    }  
    sc.close();  
}  
catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

<terminated> Principal (25) [Java .

Esto es una prueba0  
Esto es una prueba1  
Esto es una prueba2  
Esto es una prueba3  
Esto es una prueba4  
Esto es una prueba5  
Esto es una prueba6  
Esto es una prueba7  
Esto es una prueba8  
Esto es una prueba9

---

## 5.- Serialización de objetos



# 5.- Serialización de objetos

---

La **serialización** consiste en transformar un objeto en una secuencia o serie de **bytes** de tal manera que represente el **estado** de dicho objeto.

Una vez tenemos serializado un objeto, se puede enviar a un **fichero**.

La **persistencia** se consigue al tener el objeto seriado y almacenado en un fichero, porque sería posible recomponer el objeto.

El **estado de un objeto** es básicamente el estado de cada uno de los campos. Imaginemos que un campo es a su vez otro objeto, en ese caso debería de ser serializado para serializar el primer objeto.

# 5.- Serialización de objetos

---

Para poder serializar un objeto de una clase es necesario que implemente la interfaz `java.io.Serializable`.

Dicha interfaz no define ningún método, el objetivo es marcar las clases que vamos a convertir en secuencias de bytes.

**Ejemplo:**

```
public class Amigo implements Serializable {  
    //atributos y métodos de la clase  
}
```

El objeto Amigo se ha marcado como serializable, ahora Java se encargará de realizar la serialización de forma automática.

# 5.- Serialización de objetos

---

Imaginemos la clase “Amigo” que guarda el nombre y el teléfono:

```
public class Amigo implements Serializable{  
  
    private String nombre;  
    private long telefono;  
  
    public Amigo(String nombre, long telefono) {  
        this.nombre = nombre;  
        this.telefono = telefono;  
    }  
  
    public void datos_amigo() {  
        System.out.println(nombre + " -> " + telefono);  
    }  
  
}
```

Desde la clase “Principal” crearemos 2 objetos de la clase “Amigo”, los escribiremos en un fichero (serializar) y luego leeremos del mismo fichero los objetos (deserializar).

# 5.- Serialización de objetos

Para **serializar**, creamos un objeto de tipo File (como antes) y ahora, además:

- **FileOutputStream**: crea un flujo de datos para escribir en File.
- **ObjectOutputStream**: nos permite escribir objetos en ese flujo de datos.

```
public static void main(String[] args) {
    File f = new File("C:\\Users\\franp\\Desktop\\amigos.txt");
    try {
        FileOutputStream fs = new FileOutputStream(f);
        ObjectOutputStream oos = new ObjectOutputStream(fs);

        Amigo a = new Amigo("Paco Perez", 655643140);
        oos.writeObject(a);
        Amigo a2 = new Amigo("Juan Perez", 666641123);
        oos.writeObject(a2);
        oos.close();
        fs.close();
    }
    catch(IOException e) {
        e.printStackTrace();
    }
}
```

amigos.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
-í [sr [Amigo][µ·N-^ J [telefonoL [nombret [Ljava/lang/String;xp ' [R]t
Paco Perezsq ~ '%ãt
Juan Perez
```

# 5.- Serialización de objetos

---

Para **deserializar**, creamos los siguientes objetos

- **FileInputStream**: crea un flujo de datos para leer de File.
- **ObjectInputStream**: nos permite leer objetos en ese flujo de datos.

```
try {  
    FileInputStream fis = new FileInputStream(f);  
    ObjectInputStream ois = new ObjectInputStream(fis);  
  
    Amigo a = (Amigo)ois.readObject();  
    a.datos_amigo();  
  
    Amigo a2 = (Amigo)ois.readObject();  
    a2.datos_amigo();  
  
    ois.close();  
    fis.close();  
}  
catch(Exception e) {  
    System.out.println("Excepción: " + e.getMessage());  
}
```

```
<terminated> Principal (24) [Java Application]
```

```
Paco Perez -> 655643140
```

```
Juan Perez -> 666641123
```

---

## 6.- Utilización de los sistemas de ficheros

# 6.- Utilización de los sistemas de ficheros

---

## Creación de un directorio con mkdir

```
public static void main(String[] args) {  
    File carpeta = new File("C:\\Users\\franp\\Desktop\\micarpeta");  
    boolean exito = carpeta.mkdir();  
    if (exito) {  
        System.out.println("Directorio " + carpeta + " creado!");  
    }else {  
        System.out.println("Directorio " + carpeta + " no creado!");  
    }  
}
```

## Agregar archivos al directorio recién creado con createNewFile

```
File Archivo = new File("C:\\Users\\franp\\Desktop\\micarpeta\\fichero.txt");  
File Archivo2 = new File("C:\\Users\\franp\\Desktop\\micarpeta\\fichero2.txt");  
try {  
    Archivo.createNewFile();  
    Archivo2.createNewFile();  
} catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```

# 6.- Utilización de los sistemas de ficheros

---

Listar los ficheros de un directorio con list

```
String[] ficheros = carpeta.list();  
for (int i=0; i< ficheros.length; i++) {  
    System.out.println(ficheros[i]);  
}
```

---

```
Directorio C:\Users\franp\Desktop\micarpeta creado!  
fichero.txt  
fichero2.txt
```

Renombrar archivo o carpeta con RenameTo

```
File nueva_carpeta = new File("C:\\Users\\franp\\Desktop\\nueva_carpeta");  
boolean resultado = carpeta.renameTo(nueva_carpeta);  
if (resultado) {  
    System.out.println("Carpeta renombrada con éxito");  
}else {  
    System.out.println("No se pudo renombrar la carpeta");  
}
```

