

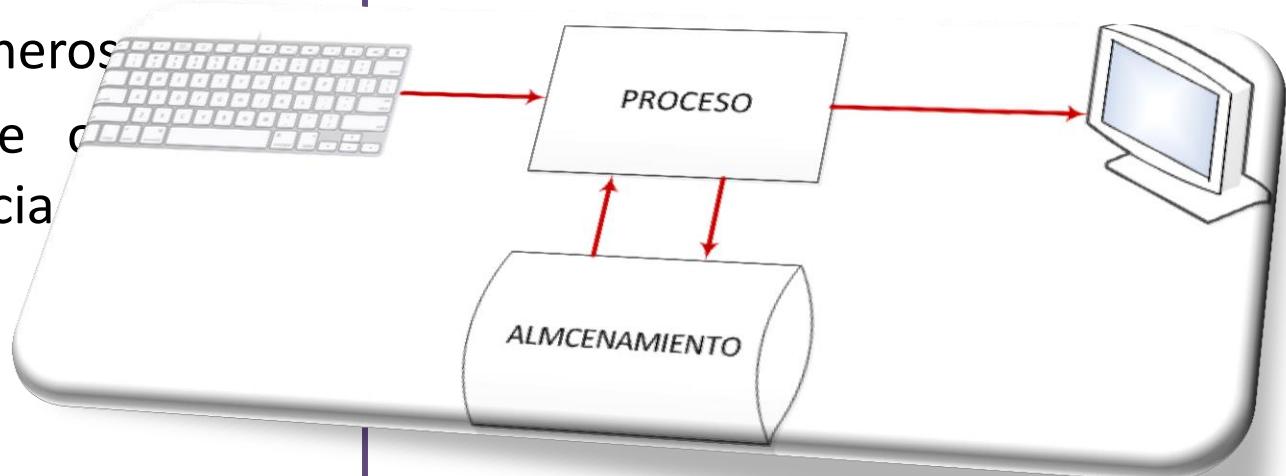
# Desarrollo de Aplicaciones Multiplataforma

## PROGRAMACIÓN

Tema 5  
Lectura y Escritura de Información

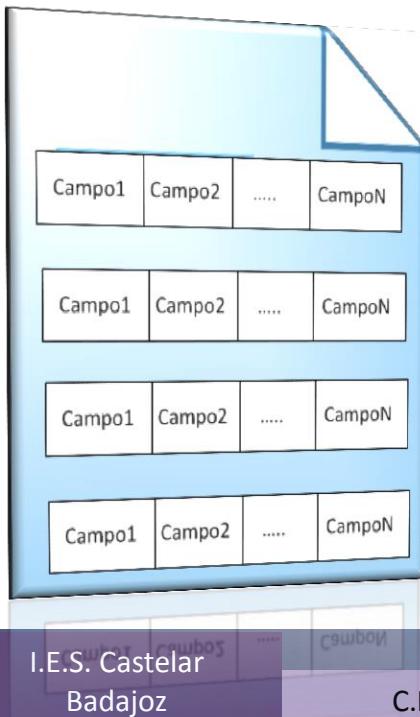
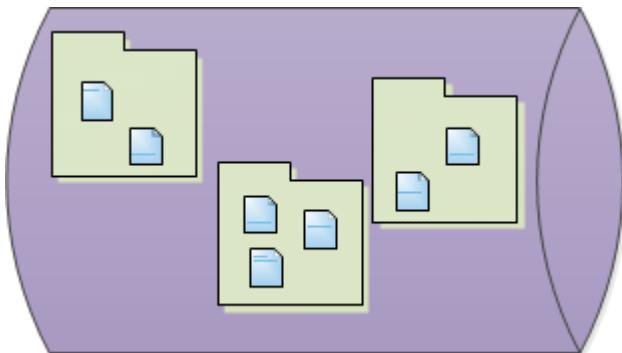
# ÍNDICE

1. Ficheros de datos. Concepto de registro
2. Tipos de flujos: flujos de bytes y flujos de caracteres
3. Flujos predeterminados
4. Flujos de caracteres
5. Flujos de bytes
6. La clase File
7. Operaciones con ficheros
8. Almacenamiento de datos en ficheros. Persistencia de serialización.



## Concepto de fichero.

Un **archivo o fichero informático** es un conjunto de bits almacenado en un dispositivo. Un archivo es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene.



## Concepto de registro.

Un registro es una agrupación heterogénea de datos (campo) almacenado en un fichero. Un fichero de datos está compuesto por un conjunto de registros.

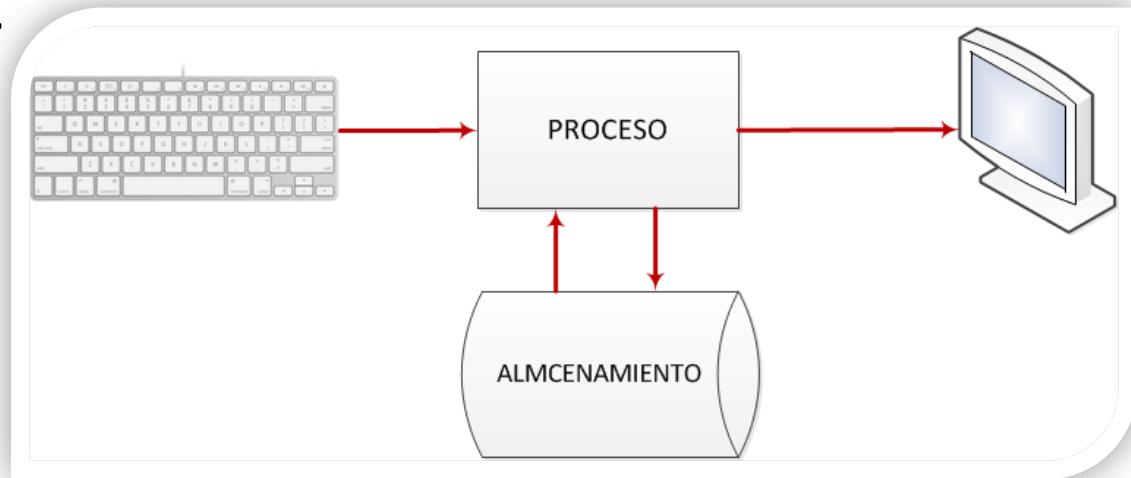
### Concepto de flujo.

La información que se necesita un programa para su función obtiene mediante una entrada de datos de una fuente que puede ser de tipos muy variados, puede de texto, binario, imágenes, etc. Todos los datos fluyen a través del ordenador desde una entrada hacia una salida. Este flujo de datos se denomina también ***stream***. Hay un flujo de entrada (*input stream*) que manda los datos desde el exterior (normalmente el teclado) del ordenador, y un flujo de salida (*output stream*) que dirige los datos hacia los dispositivos de salida (la pantalla o un archivo).

### Tipos de flujos.

Se distinguen dos tipos de flujos, unos que trabajan con bytes y otros trabajan con caracteres. Existen clases conversoras que permiten obtener flujo de bytes a partir de uno de caracteres y viceversa tanto para lectura como para escritura.

El paquete **java.io** que está en la biblioteca entándar de Java tiene todas las clases necesarias para leer y escribir datos en flujos y en el sistema de ficheros. Este paquete tiene una serie de interfaces, clases y excepciones, todas relacionadas con la entrada/salida de datos. La información que necesita un programa para su funcionamiento se obtiene mediante la entrada de datos de una fuente que puede ser de texto, binario, imágenes, etc. Este flujo de datos se denomina también ***stream***. Hay un flujo de entrada (*input stream*) que manda los datos desde el exterior (normalmente el teclado) del ordenador, y un flujo de salida (*output stream*) que dirige los datos hacia los dispositivos de salida (la pantalla o un archivo).



### 3. FLUJOS PREDETERMINADOS

#### 3.2. CLASES PRINCIPALES DE TRABAJO CON FLUJOS

	Flujos con bytes	Flujos con caracteres
Entrada de datos	<b>InputStream</b> <ul style="list-style-type: none"> <li>ByteArrayInputStream</li> <li>FileInputStream</li> <li>FilterInputStream</li> <li>BufferedInputStream</li> <li>DateInputStream</li> <li>LineNumberInputStream</li> <li>PushbackInputStream</li> <li>ObjectInputStream</li> <li>PipedInputStream</li> <li>SequenceInputStream</li> <li>StringBufferInputStream</li> </ul>	<b>Reader</b> <ul style="list-style-type: none"> <li>BufferedReader</li> <li>LineNumberReader</li> <li>CharArrayReader</li> <li>FilterReader</li> <li>PushBackReader</li> <li>InputStreamReader</li> <li>FileReader</li> <li>PipedReader</li> <li>StringReader</li> </ul>
Salida de datos	<b>OutputStream</b> <ul style="list-style-type: none"> <li>ByteArrayOutputStream</li> <li>FileOutputStream</li> <li>FilterOutputStream</li> <li>BufferedOutputStream</li> <li>DataOutputStream</li> <li>PrintStream</li> <li>ObjectOutputStream</li> <li>PipedOutputStream</li> </ul>	<b>Writer</b> <ul style="list-style-type: none"> <li>BufferedWriter</li> <li>CharArrayWriter</li> <li>FilterWriter</li> <li>OutputStreamWriter</li> <li>FileWriter</li> <li>PipedWriter</li> <li>PrintWriter</li> <li>StringWriter</li> </ul>

### 3. FLUJOS PREDETERMINADOS

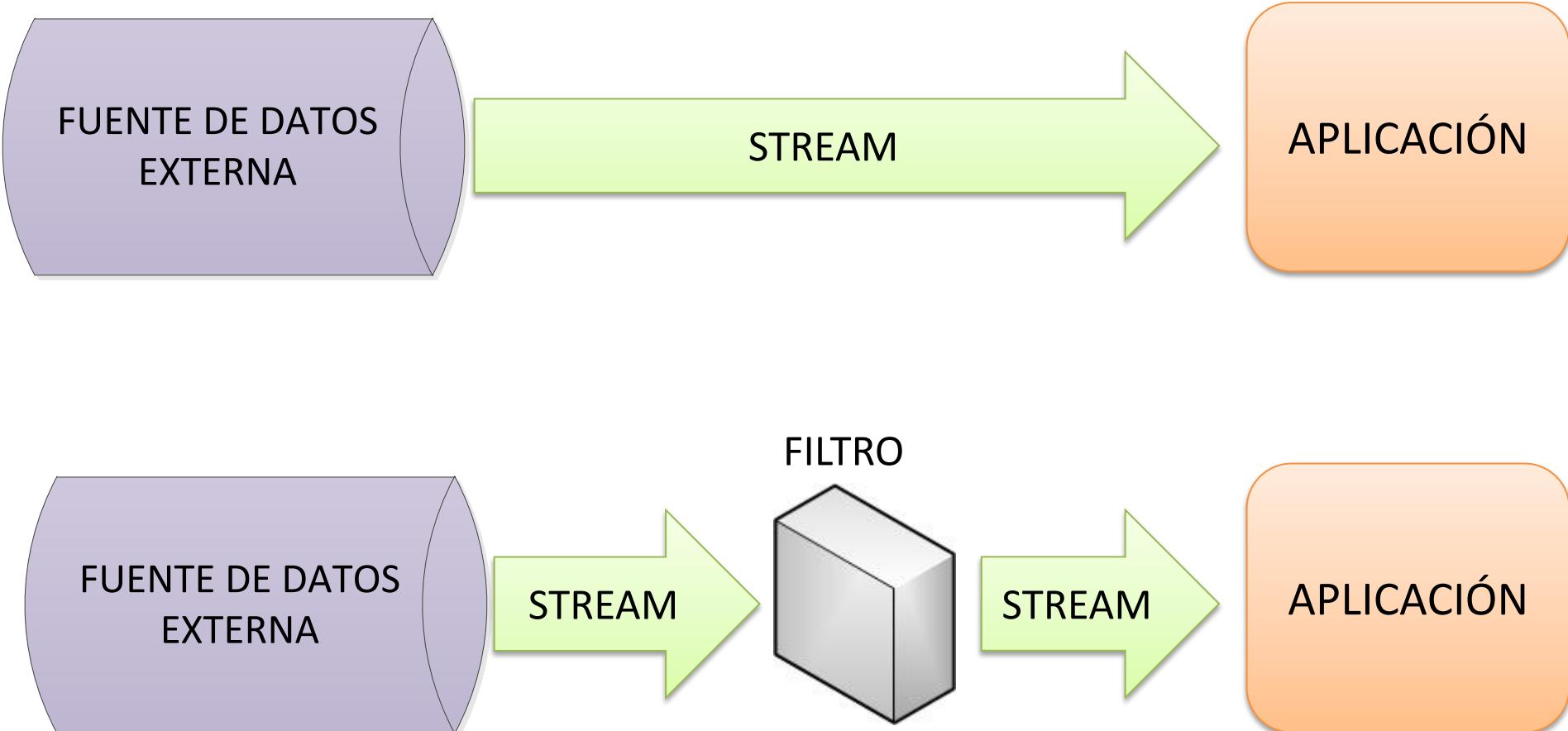
#### 3.3. ALGORITMOS

##### Lectura

1. Abrir flujo desde un archivo
2. Mientras haya datos
  1. Leer datos
3. Cerrar flujo

##### Escritura

1. Abrir flujo hacia un archivo
2. Mientras haya datos
  1. Escribir datos
3. Cerrar flujo



## 4. FLUJOS DE CARACTERES

### 4.1. LA CLASE FILEWRITER

Permite realizar escrituras de caracteres en un archivo.

```
import java.io.*  
public class EscrituraCaracterArchivo {  
    public static void main(String[] args) {  
        String nombreArchivo = "ejemplo1.txt";  
        FileWriter archivo;  
        try {  
            archivo = new FileWriter(nombreArchivo);  
            for (char letra='A'; letra <='z'; letra++) {  
                archivo.write(letra);  
            }  
            archivo.close();  
        } catch (IOException e) {  
            System.out.println("Se ha producido un error al escribir en el archivo.");  
        }  
    }  
}
```

```
File f = new File("ejemplo1.txt");  
...  
archivo = new FileWriterReader(f);
```

## 4. FLUJOS DE CARACTERES

### 4.1. LA CLASE FILEWRITER

Los constructores de la clase FileWriter son:

Constructor	Descripción
FileWriter (File file)	Crea el objeto FileWriter a partir de un objeto File. Si no existe lo crea y si existe borra el contenido.
FilerWriter(File file, boolean append)	Crea el objeto FileWriter a partir de un objeto File. Si no existe lo crea y si existe situa el puntero para añadir datos.
FilerWriter(FileDescriptor fd)	Crea el objeto FileWriter asociado a un descriptor. Se utiliza principalmente para el acceso a dispositivos y socket.
FilerWriter(String fileName)	Crea el objeto FileWriter a partir del nombre de un fichero. Si no existe lo crea y si existe borra el contenido.
FilerWriter(String FileName, boolean append)	Crea el objeto FileWriter a partir del nombre de un fichero. Si no existe lo crea y si existe situa el puntero para añadir datos.

*Investigue los diversos métodos asociado a esta clase.*

## 4. FLUJOS DE CARACTERES

### 4.2. LA CLASE FILEREADER

Permite realizar lecturas de caracteres desde un archivo.

```
import java.io.*  
public class LecturaCaracterArchivo {  
    public static void main(String[] args) {  
        String nombreArchivo = "ejemplo1.txt";  
        FileReader archivo;  
        char letra;  
        try {  
            archivo = new FileReader(nombreArchivo);  
            letra = archivo.read();  
            While (letra != -1) {  
                System.out.print((char) letra);  
                letra = archivo.read(); }  
            archivo.close();  
        } catch (IOException e) {  
            System.out.println("Se ha producido un error al leer del archivo.");  
        } } }
```

```
File f = new File("ejemplo1.txt");  
...  
archivo = new FileReader(f);
```

Los constructores de la clase FileWriter son:

Constructor	Descripción
FileReader (File file)	Crea el objeto FileReader a partir de un objeto File.
FileReaderWriter(FileDescriptor fd)	Crea el objeto FileReader asociado a un descriptor. Se utiliza principalmente para el acceso a dispositivos y socket.
FileReader(String fileName)	Crea el objeto FilerReader a partir del nombre de un fichero.

*Investigue los diversos métodos asociado a esta clase.*

## 4. FLUJOS DE CARACTERES

### 4.3. LA CLASE PRINTERWRITER

A continuación se muestra como redirigir el la salida hacia la pantalla utilizando la clase PrintWriter.

```
import java.io.*  
public class EscrituraPantalla {  
    public static void main(String[] args) throws IOException{  
        PrintWriter pantalla = new PrintWriter(System.out);  
        char [] array = {'M', 'o', 'r', 'e', 'n', 'o'};  
        String str = new String ("Juan Carlos");  
        pantalla.write(str);  
        pantalla.print (" ");  
        pantalla.write(array, 0, 6);  
        pantalla.println();  
        pantalla.flush();  
    }  
}
```

## 4. FLUJOS DE CARACTERES

### 4.4. LAS CLASES FILEREADER Y BUFFEREDREADER

La clase BufferedReader representa un filtro que se puede conectar a un flujo de E/S permitiendo leer una línea completa de una sola vez

```
import java.io.*  
public class LeerArchivo {  
    public static void main(String[] args) {  
        FileReader archivo;  
        BufferedReader filtro;  
        String cadena;  
        try {  
            archivo = new FileReader("ejemplo1.txt");  
            filtro = new BufferedReader(archivo);  
            cadena = filtro.readLine();  
            while (cadena != null) {  
                System.out.println(cadena);  
                cadena = filtro.readLine();  
            }  
            filtro.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

A continuación se muestra el uso de estas clases para redirigir el flujo de entrada de teclado

```
import java.io.*  
public class LeerTeclado {  
    public static void main(String[] args) throws IOException {  
        InputStreamReader entrada;  
        BufferedReader teclado;  
        String cadena;  
        try {  
            entrada = new InputStreamReader(System.in);  
            teclado = new BufferedReader(entrada);  
            System.out.print("Nombre: ");  
            cadena = filtro.readLine();  
            System.out.println("Nombre: " + cadena);  
        }  
    }  
}
```

## 5. FLUJOS DE BYTES

### 5.1. LA CLASE DATA FILEINPUTSTREAM

Gran parte de la entrada y salida de los programas se realiza basándose en ficheros y para ello java.io aporta dos clases. Una para ficheros de entrada, `FileInputStream`, y otra para ficheros de salida, `FileOutputStream`. `FileOutputStream` crea un fichero, salvo que exista y sea de sólo lectura.

Estas clases aportan tres constructores cada una, dependiendo de la información aportada para identificar el fichero a abrir:

- Un constructor que toma como argumento un `String` que es el nombre del fichero que se va a abrir.
- Un constructor que toma un objeto de tipo `File` que se refiere al fichero (ver la clase `File`).
- Un constructor que toma un objeto de tipo `FileDescriptor`, que constituye un valor dependiente del sistema de ficheros y que describe un fichero abierto.

En el siguiente ejemplo se representa realiza la copia de un archivo.

```
import java.io.*;
Public class FileIO {
    public static void main(String [] args) throws IOException {
        FileInputStream in = new FileInputStream("entrada.txt");
        FileOutputStream out = new FileOutputStream("salida.txt");
        int n=0,c;
        System.out.print ("\nCopiando ...");
        while( (c = in.read()) != -1) {
            out.write(c);
            n++;
        }
        in.close();  out.close();
        System.out.print ("\nSe han copiado "+n+" caracteres\n");
    }
}
```

## 5. FLUJOS DE BYTES

### 5.2. LA CLASE DATAOUTPUTSTREAM

La clase *DataOutputStream* es útil para escribir datos del tipo primitivo de una forma portable. Esta clase tiene un sólo constructor que toma un objeto de la clase *OutputStream* o sus derivadas como parámetro.

Se crea un objeto de la clase *DataOutputStream* vinculándolo a un objeto *FileOutputStream* para escribir en un archivo en disco denominado fic.txt.

**FileOutputStream fileOut=new FileOutputStream("fic.txt");**

**DataOutputStream salida=new DataOutputStream(fileOut);**

o en una sola línea

**DataOutputStream entrada=new DataOutputStream(new FileOutputStream("fic.txt"));**

<b>void writeBoolean(boolean v);</b>	<b>void writeByte(int v);</b>	<b>void writeBytes(String s);</b>
<b>void writeShort(int v);</b>	<b>void writeChars(String s);</b>	<b>void writeChar(int v);</b>
<b>void writeInt(int v);</b>	<b>void writeLong(long v);</b>	<b>void writeFloat(float v);</b>
<b>void writeDouble(double v)</b>		

## 5. FLUJOS DE BYTES

### 5.2. LA CLASE DATAOUTPUTSTREAM

...

```
double[] precios={1350, 400, 890, 6200, 8730};  
int[] unidades={5, 7, 12, 8, 30};  
String[] descripciones={"paquetes de papel", "lápices", "bolígrafos",  
"carteras", "mesas"};
```

```
DataOutputStream salida=new DataOutputStream(new  
FileOutputStream("pedido.txt"));  
for (int i=0; i<precios.length; i++) {  
    salida.writeChars(descripciones[i]);  
    salida.writeChar('\n');  
    salida.writeInt(unidades[i]);  
    salida.writeChar('\t');  
    salida.writeDouble(precios[i]);  
}  
salida.close();
```

...

## 5. FLUJOS DE BYTES

### 5.3. LA CLASE DATA INPUTSTREAM

La clase *DataInputStream* es útil para leer datos del tipo primitivo de una forma portable. Esta clase tiene un sólo constructor que toma un objeto de la clase *InputStream* o sus derivadas como parámetro.

Se crea un objeto de la clase *DataInputStream* vinculándolo a un objeto *FileInputStream* para leer desde un archivo en disco denominado fic.txt.

**FileInputStream fileIn=new FileInputStream("fic.txt");**

**DataInputStream entrada=new DataInputStream(fileIn));**

o en una sola línea

**DataInputStream entrada=new DataInputStream(new FileInputStream("fic.txt"));**

<b>boolean readBoolean()</b>	<b>byte readByte()</b>	<b>int readUnsignedByte()</b>
<b>short readShort()</b>	<b>int readUnsignedShort()</b>	<b>char readChar()</b>
<b>int readInt()</b>	<b>String readLine()</b>	<b>long readLong()</b>
<b>float readFloat()</b>	<b>double readDouble()</b>	

## 5. FLUJOS DE BYTES

### 5.3. LA CLASE DATAINPUTSTREAM

...

```
double precio; int unidad;
String descripcion; double total=0.0;
DataInputStream entrada=new DataInputStream(new
FileInputStream("pedido.txt"));
try {
    while ((descripcion=entrada.readLine())!=null) {
        unidad=entrada.readInt();
        entrada.readChar(); //lee el carácter tabulador
        precio=entrada.readDouble();
        System.out.println("has pedido "+unidad+" "+descripcion+" a "+precio+
pts.");
        total=total+unidad*precio;
    }
}catch (EOFException e) {}
System.out.println("por un TOTAL de: "+total+" pts.");
entrada.close();
...
```

## 5. FLUJOS DE BYTES

### 5.4. LA CLASE RANDOMACCESSFILE

Permite abrir un fichero con lectura y/o escritura.

#### Constructores:

RandomAccessFile(File objeto\_fichero, String modo)

RandomAccessFile(String nombre, String modo)

modo: "r" (read) y "rw"(read-write)

#### Métodos:

Creación	Escritura	Lectura
void seek(long posición) long getFilePointer() int skipBytes(int desplazamiento) long length() ...	writeInt(entero) writeDouble(doble) writeBytes(cadena) writeUTF(String) ...	readInt() readDouble() readUTF() readFloat() readShort() ...

## 5. FLUJOS DE BYTES

### 5.5. LA CLASE FILE

Proporciona información acerca de los archivos, de sus atributos, de los directorios, etc.

#### Constructores:

`File(String nombre)`

`File(String directorio, String nombre)`

`File(File directorio, String nombre)`

#### Métodos:

Métodos		
<code>String getName()</code>	<code>boolean canRead</code>	<code>boolean mkdir()</code>
<code>String getPath()</code>	<code>boolean isFile()</code>	<code>boolean mkdirs()</code>
<code>String getAbsolutePath()</code>	<code>boolean isDirectory()</code>	<code>boolean renameTo(File dest);</code>
<code>boolean exists()</code>	<code>boolean isAbsolute()</code>	<code>boolean delete()</code>
<code>boolean canWrite()</code>	<code>long lastModified()</code>	<code>String[] list()</code>
	<code>long length()</code>	<code>String[] list(FilenameFilter filter)</code>

## Definición.

- La serialización de objetos permite convertir cualquier objeto **que implemente la interfaz Serializable** en una secuencia de bits que puede ser utilizada posteriormente para reconstruir el objeto original.
- La secuencia de bits puede guardarse en un fichero o puede enviarse a otra máquina virtual para reconstruir el objeto posteriormente.
- La posibilidad de guardar un objeto de forma que pueda existir incluso cuando la aplicación haya finalizado se conoce como persistencia.
- Si los objetos mantienen referencias a otros objetos. Estos objetos también deben ser serializables.
- La interfaz Serializable no define ningún método.
- La serialización se introdujo en Java para soportar la Invocación remota de Métodos (RMI) que permite a una aplicación enviar mensajes a un objeto remoto (que se está ejecutando en otra MV). También es necesaria en el caso de los JavaBeans

## 5. FLUJOS DE BYTES

### 5.6. SERIALIZACIÓN DE LOS OBJETOS

Los objetos `ObjectInputStream` y  `ObjectOutputStream` suministran un mecanismo para el almacenamiento persistente de objetos. La clase `ObjectInputStream` permite leer objetos previamente serializados y estos objetos, para ser serializados, deben implementar la interfaz **Serializable** o **Externalizable**.

Para leer un objeto desde un flujo se utiliza el método `readObject()`. La escritura se realiza con el método `writeObject()`.

Los pasos para crear un objeto serializable son los siguientes:

Crear una clase que implemente la interfaz `Serializable`.

Esa clase debe tener un método para serializarse a si misma, mediante el método `writeObject()` de la clase  `ObjectOutputStream`.

La lectura de un objeto serializado se realiza con el método `readObject()` de la clase `ObjectInputStream`.

## 5. FLUJOS DE BYTES

### 5.6. SERIALIZACIÓN DE LOS OBJETOS

Los objetos `ObjectInputStream` y  `ObjectOutputStream` suministran un mecanismo para el almacenamiento persistente de objetos. La clase `ObjectInputStream` permite leer objetos previamente serializados y estos objetos, para ser serializados, deben implementar la interfaz **Serializable** o **Externalizable**.

Para leer un objeto desde un flujo se utiliza el método `readObject()`. La escritura se realiza con el método `writeObject()`.

Los pasos para crear un objeto serializable son los siguientes:

Crear una clase que implemente la interfaz `Serializable`.

Esa clase debe tener un método para serializarse a si misma, mediante el método `writeObject()` de la clase  `ObjectOutputStream`.

La lectura de un objeto serializado se realiza con el método `readObject()` de la clase `ObjectInputStream`.

El siguiente ejemplo implementa una clase serializable.

```
package serializacion;
```

```
import java.io.*;
```

```
public class Coche implements Serializable {
```

```
    String matricula;
```

```
    int anioMatriculacion;
```

```
    public Coche() {
```

```
}
```

```
    public Coche (String matricula, int anioMatriculacion) {
```

```
        this.matricula = matricula;
```

```
        this.anioMatriculacion=anioMatriculacion;
```

```
}
```

## 5. FLUJOS DE BYTES

### 5.6. SERIALIZACIÓN DE LOS OBJETOS

```
public void saveObject() {  
    try {  
        FileOutputStream fichero = new FileOutputStream("coches.obj");  
        ObjectOutputStream oos = new ObjectOutputStream(fichero);  
        oos.writeObject(this);  
        oos.flush();  
        oos.close();  
    } catch (IOException e) {  
        System.out.println("Error al serializar el objeto.");  
    }  
}
```

## 5. FLUJOS DE BYTES

### 5.7. LECTURA Y ESCRITURA DE OBJETOS

```
package serializacion;

import java.io.*;

public class Concesionario {
    public static void main(String[] args ) {
        Coche coche = new Coche ("GHV2312",2011);
        coche.saveObject();
        Coche cocheNuevo = readCoche() ;
        System.out.println("Matricula: " + cocheNuevo.matricula +
                           " Matriculado: " + cocheNuevo.anioMatriculacion);
    }
}
```

## 5. FLUJOS DE BYTES

### 5.7. LECTURA Y ESCRITURA DE OBJETOS

```
public static Coche readCoche() {  
    Coche cocheNuevo = null;  
    try {  
        FileInputStream fichero = new FileInputStream("coches.obj");  
        ObjectInputStream ois = new ObjectInputStream(fichero);  
        cocheNuevo = (Coche) ois.readObject();  
        ois.close();  
        fichero.close();  
    } catch (Exception e) {  
        System.out.println("Error al serializar el objeto.");  
    }  
    return cocheNuevo;  
}  
}
```

# REFERENCIAS BIBLIOGRÁFICAS

## ❖ Programación.

- Autor. Juan Carlos Moreno
- Editorial. Ra-Ma

## ❖ Programación en Java2.

- Autor. Jesús Sánchez Allende y otros
- Editorial. Schaum

# Desarrollo de Aplicaciones Multiplataforma

## PROGRAMACIÓN

Tema 5  
Lectura y Escritura de Información