

Taller de Lenguajes II

Tema de hoy: Entrada-Salida en JAVA

- Introducción
- Streams estándares: System.in, System.out, System.err
 - Ejemplos
- Clasificación de Streams de E/S
- Streams de caracteres: Readers y Writers
 - Métodos
 - Ejemplos
 - Combinaciones
- Leer y exportar datos en formato CSV

Introducción

- La mayoría de los programas requieren acceder a **datos externos**: procesar datos de entrada y producir datos de salida de acuerdo a dicha entrada. Ej: leer datos de un archivo almacenado en el filesystem o desde la red y escribir en otro archivo o devolver la respuesta en la red.
- Los **datos externos** de un programa se **recuperan** o se **leen** desde un **origen** de entrada o **fuentes** y su **resultado** se **envía** a un **destino** de salida.
- Una **fuentes** o **entrada** y el **destino** o **salida** pueden ser muy **variados**: desde **archivos almacenados** en el filesystem, **datos ingresados** desde el **teclado**, **información mostrada** en el **monitor**, hasta **otros programas**, etc.

Introducción

Entrada-Salida con *streams*

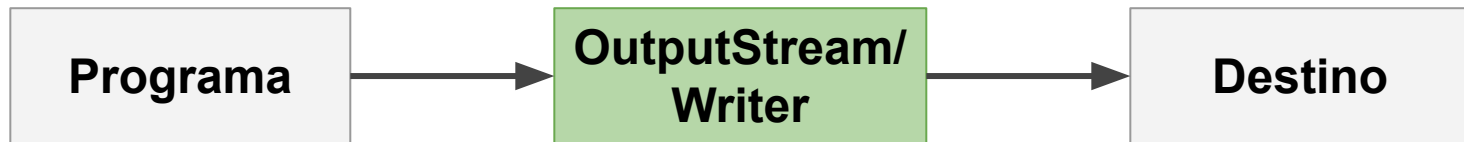
- **Java** define la abstracción llamada ***stream*** como un **flujo de datos** desde el que es posible **leer** o **escribir**. Un ***stream*** siempre está conectado a una **fuente de datos (entrada)** o a un **destino de datos (salida)**.
- Los ***streams*** o **flujos** son una **entidad lógica** que actúan como una **interface** entre los **dispositivos de entrada** y de **salida** y, los **programas**:
 - Independencia del tipo de datos y de los dispositivos.
 - Diversidad de dispositivos: archivos, pantalla, teclado, red.
 - Flexibilidad: es posible combinarlos.
 - Diversidad de formas de comunicación: acceso secuencial, aleatorio, la información que se intercambia puede ser binaria, caracteres, líneas, etc.

Entrada-Salida con *streams*



El **programa** que **lee datos** desde una **fuente** necesita un **InputStream** o un **Reader**. Un **stream de entrada** está **conectado** a una **fuente de datos**.

¿**Cuáles podrían ser fuentes de datos?** el teclado, un archivo en el filesystem, un socket remoto.



El **programa** que **escribe datos** en un **destino** necesita un **OutputStream** o un **Writer**. Un **stream de salida** está **conectado** a un **destino de datos**.

¿**Cuáles podrían ser destinos de datos?** la pantalla, un archivo en el filesystem, un socket local.

Los **streams soportan diferentes tipos de datos**, entre ellos **bytes**, **datos primitivos**, **caracteres** localizados y **objetos**. Algunos streams simplemente pasan los datos, otros los **manipulan** y los **transforman**.

Entrada-Salida con *streams*

- **Java** implementa los **streams** o **flujos de datos** a través de las clases del **paquete java.io**.
- Esencialmente **todos los flujos funcionan igual**, independientemente del dispositivo con el que se conecten.
- El **paquete java.io** se ocupa de la **lectura de datos** sin formato desde una **fuentes** y la **escritura de datos** sin formato en un **destino**. Las **fuentes** y **destinos** de datos típicos son:
 - Archivos del filesystem.
 - Sockets que comunican programas remotos.
 - Buffers en memoria: arreglos.
 - System.in, System.out, System.error

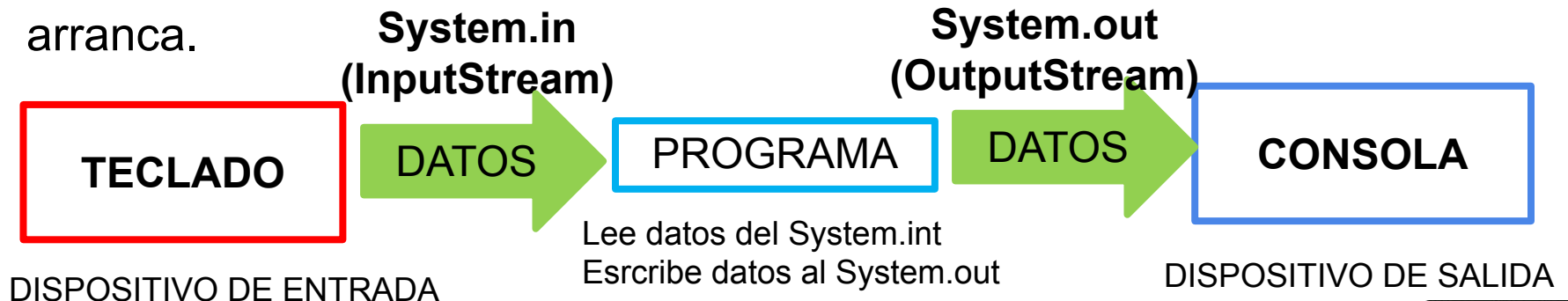
Streams Estándares

Algo conocido....

En **Java** se accede a la **E/S estándar** a través de campos estáticos de la clase **java.lang.System**.

- **System.in** implementa la **entrada estándar**. Es un **InputStream** conectado con el teclado.
- **System.out** implementa la **salida estándar**. Es un **PrintStream** (**OutputStream**) conectado con la consola.
- **System.err** implementa la **salida de error**. Es un **PrintStream** (**OutputStream**) conectado con la consola.

Estos 3 streams están listos para usar, los instancia la JVM cuando arranca.



Streams Estándares

System.in

- Es una instancia de la clase **InputStream**: flujo de bytes de entrada.
- Típicamente conectado al **teclado** para programas de línea de comando.
- Métodos:
 - `read()`: lee un byte de la entrada como un entero
 - `skip(n)`: ignora n bytes de la entrada
 - `available()`: devuelve el número estimado de bytes disponibles para leer de la entrada.

System.out

- Es una instancia de clase **PrintStream (OutputStream)**: flujo de bytes de salida.
- Los datos se escriben en la **consola**. Es usado frecuentemente por programas de línea de comando.
- Métodos para imprimir datos:
 - `print()`, `println()`
 - `flush()`: vacía el buffer de salida y escribe su contenido

System.err

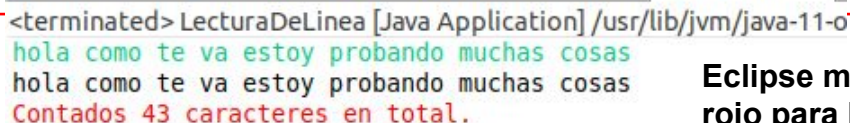
- Es similar al System.out.
- Se usa para enviar mensajes de error a la consola.

Streams Estándares

Ejemplo

```
package taller.entradasalida;
import java.io.*;

class LecturaDeLinea {
    public static void main( String args[] ) throws IOException {
        int c;
        int contador = 0;
        // se lee hasta encontrar el fin de línea
        while( (c = System.in.read()) != '\n' ){
            contador++;
            System.out.print( (char) c );
        }
        System.out.println(); // Se escribe el fin de línea
        System.err.println( "Contados "+ contador +" caracteres en total." );
    }
}
```



The screenshot shows the Eclipse IDE's console window. The title bar includes icons for Problems, Javadoc, Declaration, Search, and Console. The output text is as follows:

```
<terminated> LecturaDeLinea [Java Application] /usr/lib/jvm/java-11-o
hola como te va estoy probando muchas cosas
hola como te va estoy probando muchas cosas
Contados 43 caracteres en total.
```

The first two lines of output are green, while the third line is red, indicating an error.

Eclipse muestra la salida en el **System.err** en rojo para hacer más obvio que es un error.

Clasificación de los Streams de E/S

a) Información que presentan

- Flujos de bytes: clases **InputStream** y **OutputStream**
- Flujos de caracteres: clases **Reader** y **Writer**

Se puede pasar de un flujo de bytes a uno de caracteres con **InputStreamReader** y **OutputStreamWriter**

b) Propósito

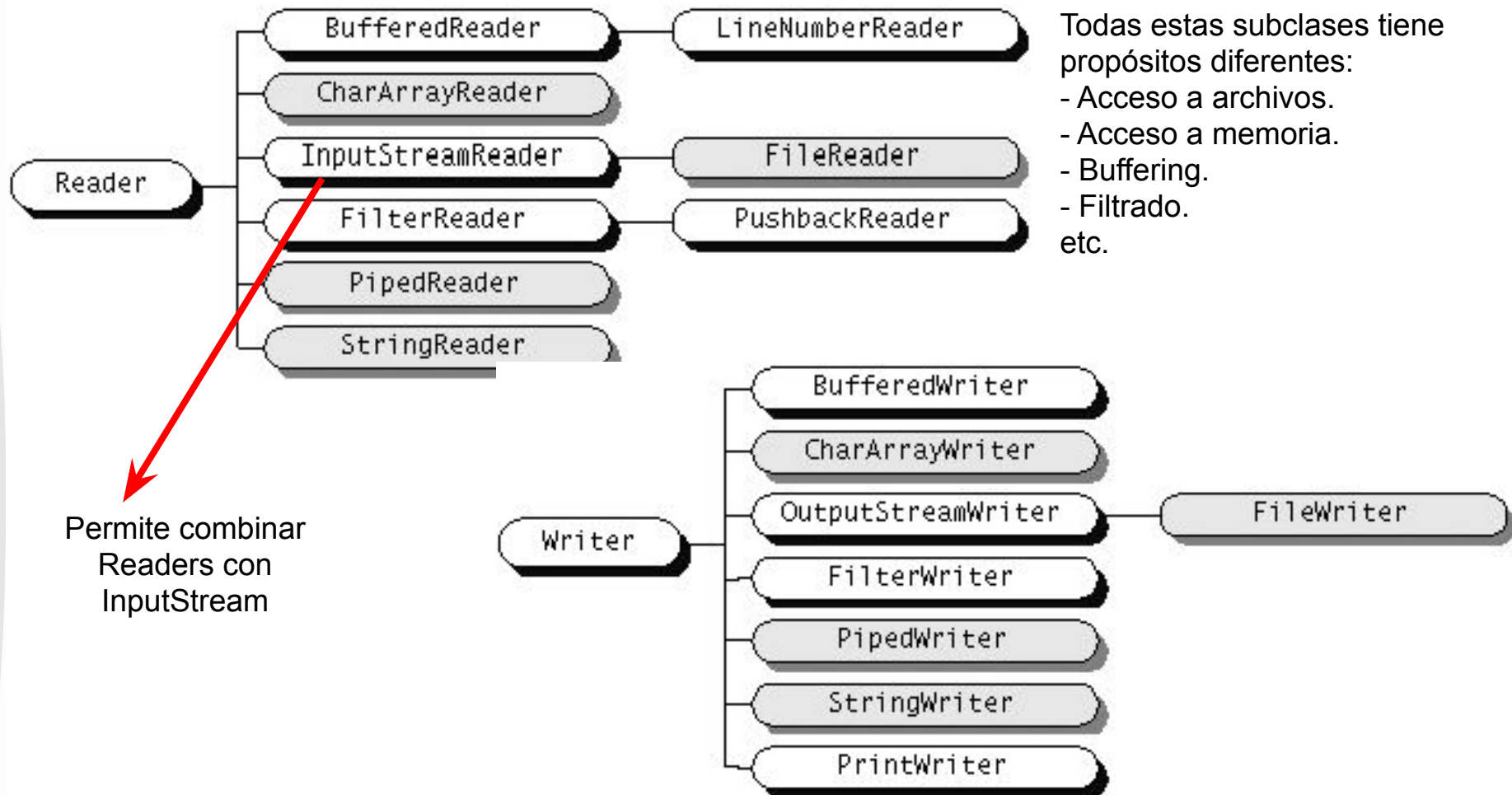
- Entrada: clases **InputStream** y **Reader**
- Salida : clases **OutputStream** y **Writer**
- Entrada/Salida: clase **RandomAccessFile**
- Transformación de datos: realizan algún tipo de procesamiento sobre los datos (p.ej. buffering, conversiones, filtrados): las clases **BufferedReader**, **BufferedWriter**

c) Tipo de Acceso

- Secuencial
- Aleatorio (clase **RandomAccessFile**)

Streams de caracteres

Readers y **Writers** ofrecen un medio para el manejo de **entradas y salidas** de **caracteres**. Dichos flujos usan codificación **Unicode**. Se usan para leer/escribir texto.



Streams de caracteres

Las subclases de **Reader** y **Writer** implementan **streams de caracteres específicos**:

- Aquellos que **leen** desde o **escriben** en **memoria** (fondo gris): arreglos, archivos, strings, etc.
- Aquellos que realizan algún **procesamiento** o **transformación** (fondo blanco): buffering, filtrado, ect.

Streams de caracteres

Los métodos de **Reader**:

int read()

retorna un carácter Unicode o -1 en caso de error o fin de archivo.

int read(char[] cbuf)

int read(char[] cbuf, int offset, int length)

leen y almacenan el resultado en un arreglo. Los parámetros *offset* y *length* son usados para indicar un subrango en el arreglo destino que necesita ser completado.

void close(): cierra el flujo y libera cualquier recurso asociado a él.

Streams de caracteres

Ejemplo: lectura

```
public class LeerFileReader {  
    public static void main(String[] args) throws FileNotFoundException, IOException {  
        Reader reader = new FileReader("/home/claudia/Documentos/txtfile.txt");  
        int data;  
        try {  
            data = reader.read();  
            while(data != -1){  
                char dataChar = (char) data;  
                System.out.print(dataChar);  
                data = reader.read();  
            }  
        } catch (IOException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        } finally {  
            if (reader!=null) reader.close();  
        }  
    }  
}
```

Streams de caracteres

Los métodos de **Writer**:

void write(int c)

void write(char[] cbuf)

void write(char[] cbuf, int offset, int length)

void write(String string)

void write(String string, int offset, int length)

Escriben en el flujo de datos un carácter, un arreglo de caracteres (o parte de él) o un string (o parte de él).

void close(): cierra el flujo y libera cualquier recurso asociado a él.

void flush(): vacía el flujo.

Streams de caracteres

Ejemplo: escritura

```
public class EscribirFileWriter {  
  
    public static void main(String[] args) throws IOException {  
        Writer writer = new  
            FileWriter("/home/claudia/Documentos/file-output.txt");  
        writer.write("Hola mundo Writer");  
        writer.close();  
    }  
  
}
```

Combinar Reader e InputStreams

- Writers y OutputStreams

La clase InputStreamReader:

Lee **bytes** de un flujo **InputStream** y los **convierte** en **caracteres** UNICODE. Es un **punto** entre flujos de **bytes** y flujos de **caracteres**.

```
Reader reader = new InputStreamReader(inputStream);
```

La clase OutputStreamWriter:

Los **caracteres escritos** en el **OutputStreamWriter** son **codificados a bytes**. Es un **punto** entre flujos de bytes y flujos de caracteres.

```
Writer writer = new OutputStreamWriter(outputStream);
```


Streams de caracteres

Archivos CSV

CSV (Comma-separated values) es un **formato de archivos de texto** popularmente usado para transferencia de datos. Se caracteriza por separar los datos con “,”.

Ejemplo:

Nombre, Materia, Nota

Tomás, Taller2, 9

Analía, Taller1, 10

Susana, Programación3, 8

Es común exportar las planillas Excel a formato CSV.

Streams de caracteres

Lectura de un archivo csv

```
public class LeerArchivoCSV {  
    private static final String SEPARADOR=",";  
    public static void main(String[] args) throws IO Exception {  
        BufferedReader br = null;  
        String linea="";  
        try {  
            br = new BufferedReader(new  
                FileReader("/home/claudia/Documentos/inscriptos OP-OIA.csv"));  
            while ((linea = br.readLine()) != null) {  
                String[] fields = linea.split(SEPARADOR);  
                System.out.println(Arrays.toString(fields));  
                System.out.println("-----");  
            }  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally {  
            if (br != null)  
                br.close();  
        }  
    }  
}
```

```
[Santillán, Thomas hernan, EEST 2 de Berisso, 4º, thomas7777778@gmail.com, "Lunes de 16:30 a 18:00, Miércoles de 18:00 a 20:00, Viernes  
-----  
[Di Renzo, Abril, EEST 5, 5º, adirenzo@eest5.com, Lunes de 16:30 a 18:00, Nunca programé, , me da curiosidad y quiero aprender]  
-----  
[Sosa, Martin Rodrigo, EEST 2 de Berisso, 7º, sosarodrigo861@gmail.com, Viernes de 16:30 a 18:00, Nunca programé, Pseint, Quiero adentrarme  
-----
```

Combinar Readers

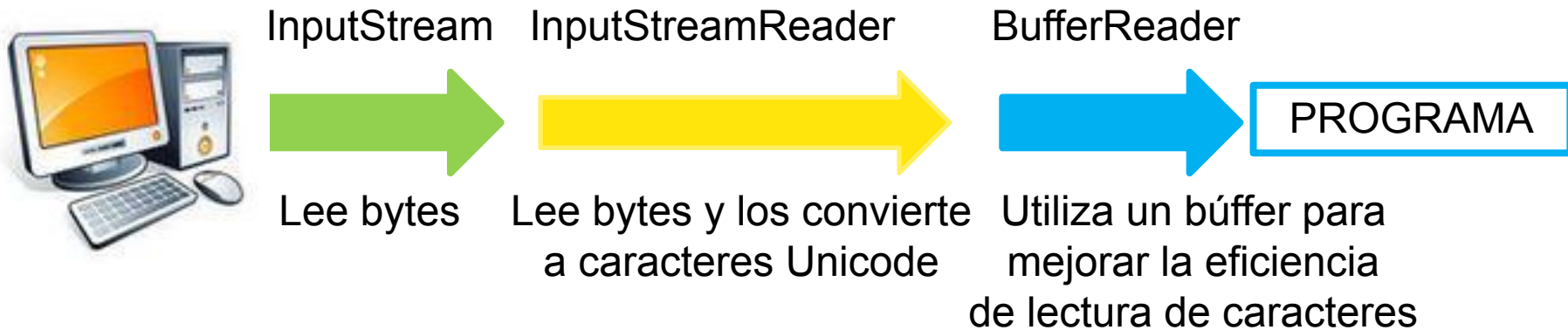
Las clases **BufferedReader** y **BufferedWriter**

Incrementan la eficiencia de la lectura y escritura de los streams de caracteres usando técnicas de *buffering*.

```
package taller.entradasalida;
import java.io.*;
public class Eco {
    public static void main(String[] args) throws IOException {
        BufferedReader entradaEstandar = new BufferedReader(
            new InputStreamReader(System.in));

        String mensaje;
        System.out.println("Introducir una línea de texto:");
        mensaje = entradaEstandar.readLine();
        System.out.println("Introducido: \"" + mensaje + "\"");
    }
}
```

Combinar Readers



Flujo de transformación de datos

Los flujos se pueden combinar para obtener la funcionalidad deseada

Streams de caracteres

Exportar de un archivo csv

```
public class ExportarArchivoCSV {  
    public static void main(String[] args) throws IOException{  
        BufferedReader in = new BufferedReader(new  
            FileReader("/home/claudia/Documentos/txtfile.txt"));  
        BufferedWriter out = new BufferedWriter (new  
            FileWriter("/home/claudia/Documentos/outfile.csv"));  
        String str = "";  
        String strLine;  
        while ((strLine = in.readLine()) != null) {  
            str += strLine+ ",";  
        }  
        System.out.println (str);  
        out.write(str);  
        in.close();  
        out.close();  
    }  
}
```

Nombre del proyecto,Ciencia de datos en la escuela,Directora,Sofía Martin,Objetivo general,Implementar actividades

Streams de caracteres

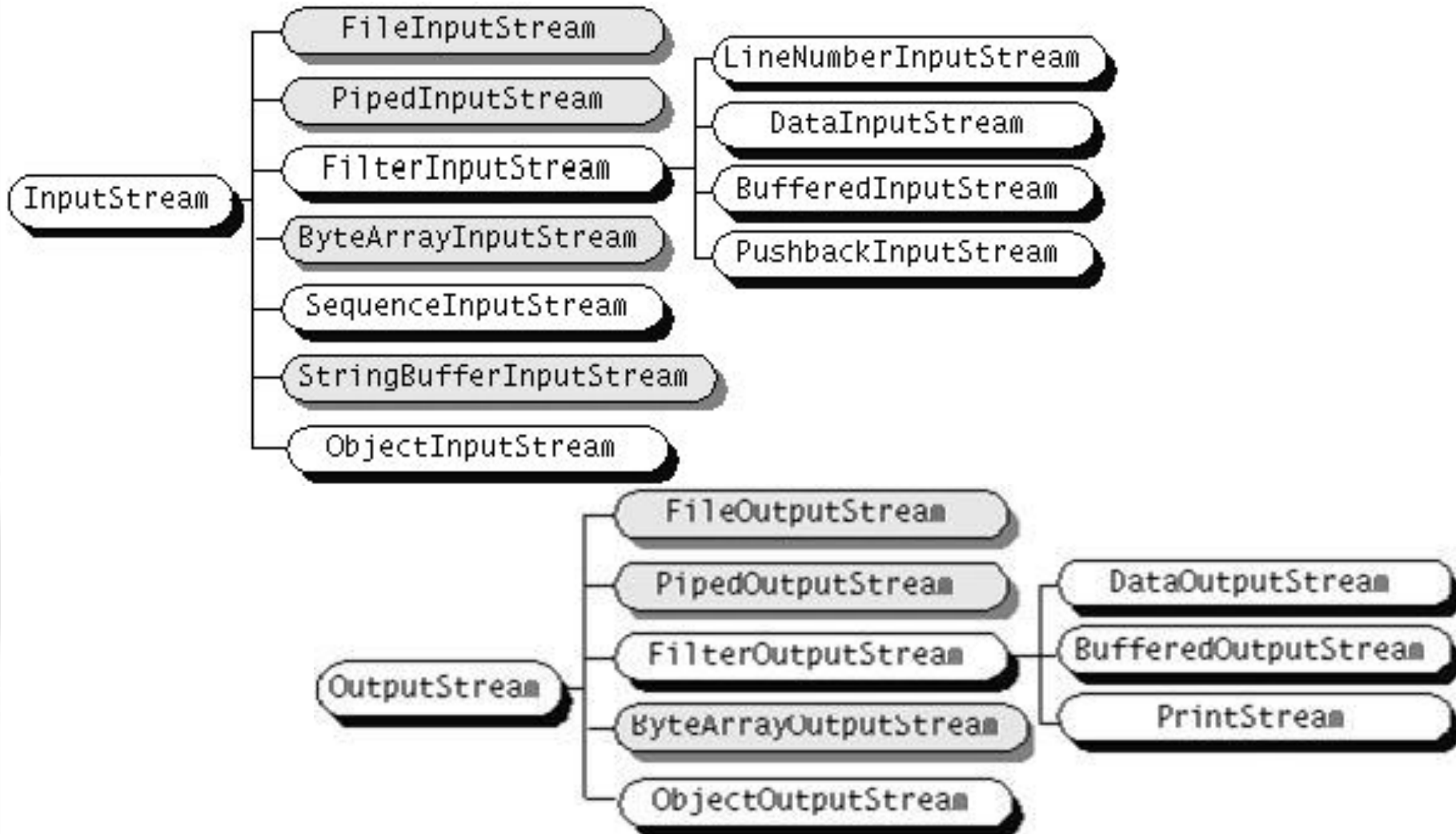
Exportar de un archivo csv

```
public class EscribirArchivoCSV2 {  
    public static void main(String[] args) throws IOException {  
        List<List<String>> filas = Arrays.asList(  
            Arrays.asList("Tomás", "23", "Taller2"),  
            Arrays.asList("Analía", "21", "Taller1"),  
            Arrays.asList("Susana", "25", "Programación3") );  
        FileWriter csvWriter = new  
FileWriter("/home/claudia/Documentos/salida.csv");  
        csvWriter.append("Nombre");  
        csvWriter.append(",");  
        csvWriter.append("Edad");  
        csvWriter.append(",");  
        csvWriter.append("Materia");  
        csvWriter.append('\n');  
        for (List<String> datos_fila : filas) {  
            csvWriter.append(String.join(",", datos_fila));  
            csvWriter.append('\n');  
        }  
        csvWriter.close();  
    }  
}
```

Nombre	Edad	Materia
Tomás	23	Taller2
Analía	21	Taller1
Susana	25	Programación3

Streams de bytes

Ofrecen un medio adecuado para el manejo de **entradas y salidas de bytes**. Su uso está orientado a **lectura y escritura de datos binarios**.



Streams de bytes

Las subclases de **InputStream** y **OutputStream** implementan streams específicos:

- Aquellos que leen desde o escriben en lugares de memoria (fondo gris).
- Aquellos que realizan algún procesamiento (fondo blanco).
- **ObjectInputStream** y **ObjectOutputStream** son utilizados para persistir objetos.
- Los métodos son similares a los de las clases **Reader** y **Writer** pero para elementos del tipo byte.

Streams de bytes

Combinar InputStreamReader y FileInputStream

```
package entradasalida;
import java.io.*;
public class LeeLineaDeArchivo {
    public static void main(String[] args) throws IOException{
        FileInputStream fstream = new
        FileInputStream("/home/claudia/Documentos/file-output.txt");
        BufferedReader br = new BufferedReader(new InputStreamReader(fstream));
        String strLine;
        while ((strLine = br.readLine()) != null) {
            //Se procesa la línea
            System.out.println (strLine);
        }
        fstream.close();
    }
}
```