Programación Orientada a Objetos

Dra. Maricela Claudia Bravo Contreras

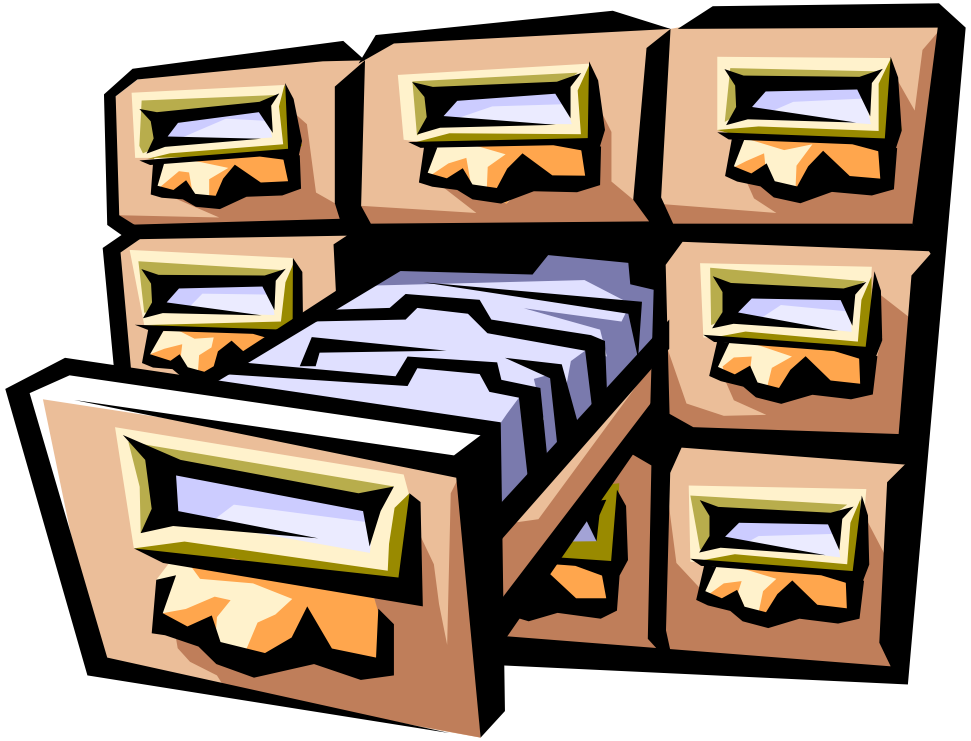
mari_clau_18@hotmail.com

Archivos y Flujos en Java

Programación Orientada a Objetos

Archivos

- Los archivos tienen como finalidad guardar datos de forma permanente.
- Una vez que acaba la aplicación los datos almacenados están disponibles para que otra aplicación pueda recuperarlos para su consulta o modificación.



Archivos

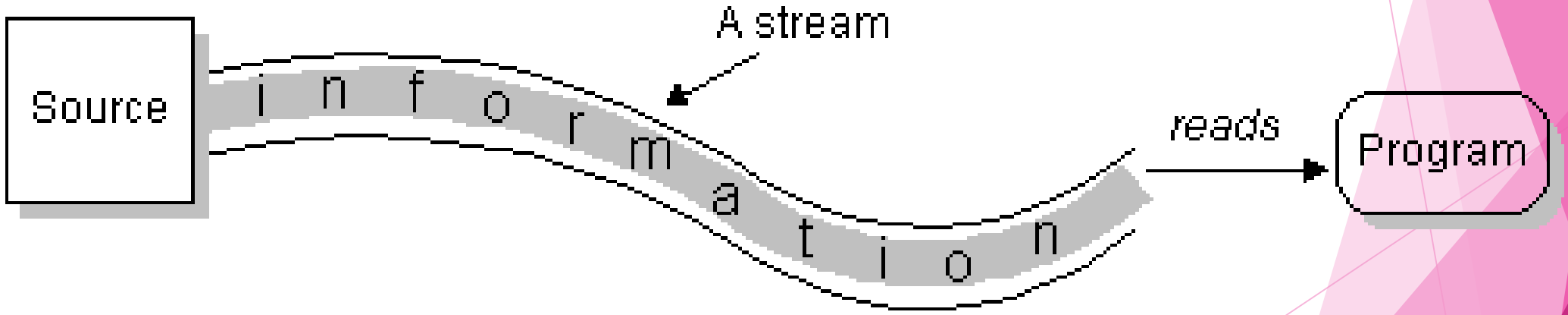
- La organización de un archivo define la forma en la en que se estructuran u organizan los datos.
- Formas de organización fundamentales:
  - **Secuenciales**: los registros se insertan en el archivo en orden de llagada. Las operaciones básicas permitidas son: escribir, añadir al final del archivo y consultar .
  - **Directa o aleatoria**: cuando un registro es directamente accesible mediante la especificación de un índice.

Flujos (Streams)

- Es una abstracción que representa a un flujo de datos entre un origen y un destino.
- Todo proceso de entrada y salida en Java se hace a través de flujos.
- Entre el origen y el destino debe existir un canal , por el que viajan los datos.
- Cuando se abre un archivo se establece una conexión entre el programa y el dispositivo que contiene ese archivo, por el canal fluirá la secuencia de datos.
- Igual ocurre al intentar escribir en un archivo.

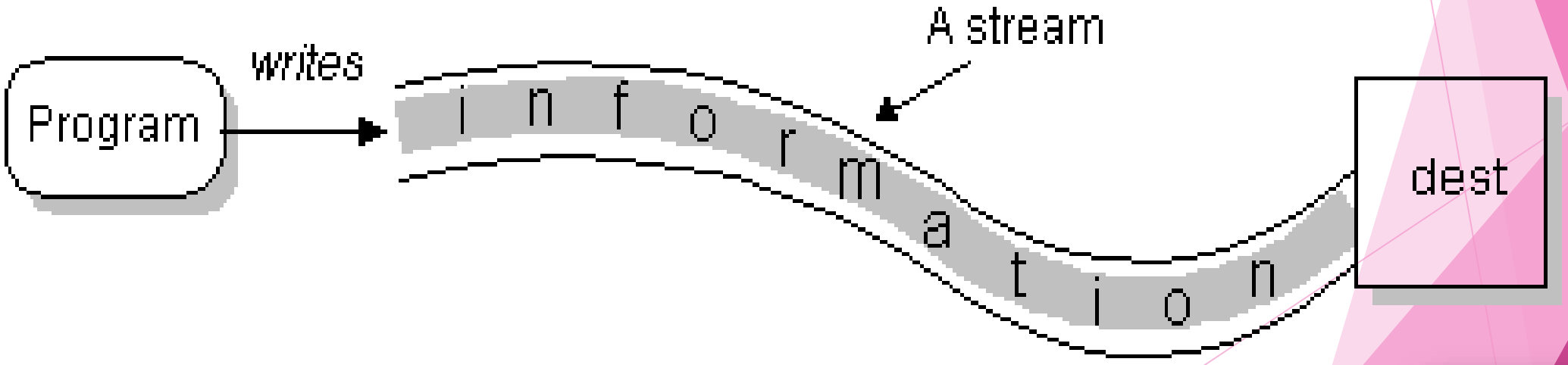
Archivos y Flujos en Java

Para obtener información de una fuente un programa abre un flujo y lee la información secuencialmente.



Archivos y Flujos en Java

De igual forma, un programa puede enviar información a un destino externo abriendo un flujo al destino y escribiendo la información secuencialmente .



Archivos y Flujos

- Java posee una colección de clases stream las cuales soportan estos algoritmos de lectura y escritura.
- Para utilizar las clases stream el programa deberá importar el paquete ***java.io*** donde se encuentran todas las clases necesarias para dar entrada/salida a las aplicaciones.

FileReader y FileWriter

- Los streams para archivos se manejan con los objetos asociados a la clase File(FileReader y FileWriter) para leer y escribir a un archivo en asociación con los métodos: read() y write().

- Un stream para archivos se puede crear a partir de
  - Un String con el nombre del archivo
  - Un objeto tipo File
  - Un objeto tipo FileDescriptor

Archivos y Flujos

```
/* El siguiente programa utiliza un objeto FileReader y uno
FileWriter para copiar el contenido de un archivo denominado
cancion.txt a un archivo denominado salida*/

import java.io.*;

public class Copy
{
    public static void main(String[] args) throws IOException
    {
        File inputFile = new File("cancion.txt");
        File outputFile = new File("salida");

        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
        int c;

        while ((c = in.read()) != -1)
           out.write(c);

        in.close();
        out.close();
    }
}
```

Manipulación de Flujos

Archivos: Entrada - Salida

Clases básicas para el manejo de archivos IO

FileInputStream, para lectura desde un archivo

FileOutputStream, para escritura en un archivo

Ejemplo:

Abrir el archivo "miArch.txt" para lectura

FileInputStream archLectura = new FileInputStream("miArch.txt");

Abrir el archivo “miArch.txt" para escritura

FileOutputStream archEscritura = new FileOutputStream ("miArch.txt");

Desplegar el contenido del archivo

```
public class LeeDatos {
   public static void main(String args[]) {
      try
      {
         FileInputStream archEntrada = new FileInputStream("miArch.txt");
         byte buffer[] = new byte[20];
         int numBytes;
         do
         {
                numBytes = archEntrada.read(buffer);
                System.out.write(buffer, 0, numBytes);
         } while (numBytes == buffer.length);
       } catch (FileNotFoundException e)
         {
          System.err.println("No se encontró el archivo");
       } catch (IOException e)
         {
         System.err.println("Falló la lectura");
       }
   }
}
```

Filtros

- Una vez que se ha creado un stream, se pueden usar filtros
- Los filtros hacen que la lectura/escritura sea más eficiente

- Los filtros más usados son:
- Para los tipos básicos:
    - DataInputStream, DataOutputStream
- Para los objetos:
    - ObjectInputStream, ObjectOutputStream

Escritura de datos a un archivo usando Filtros

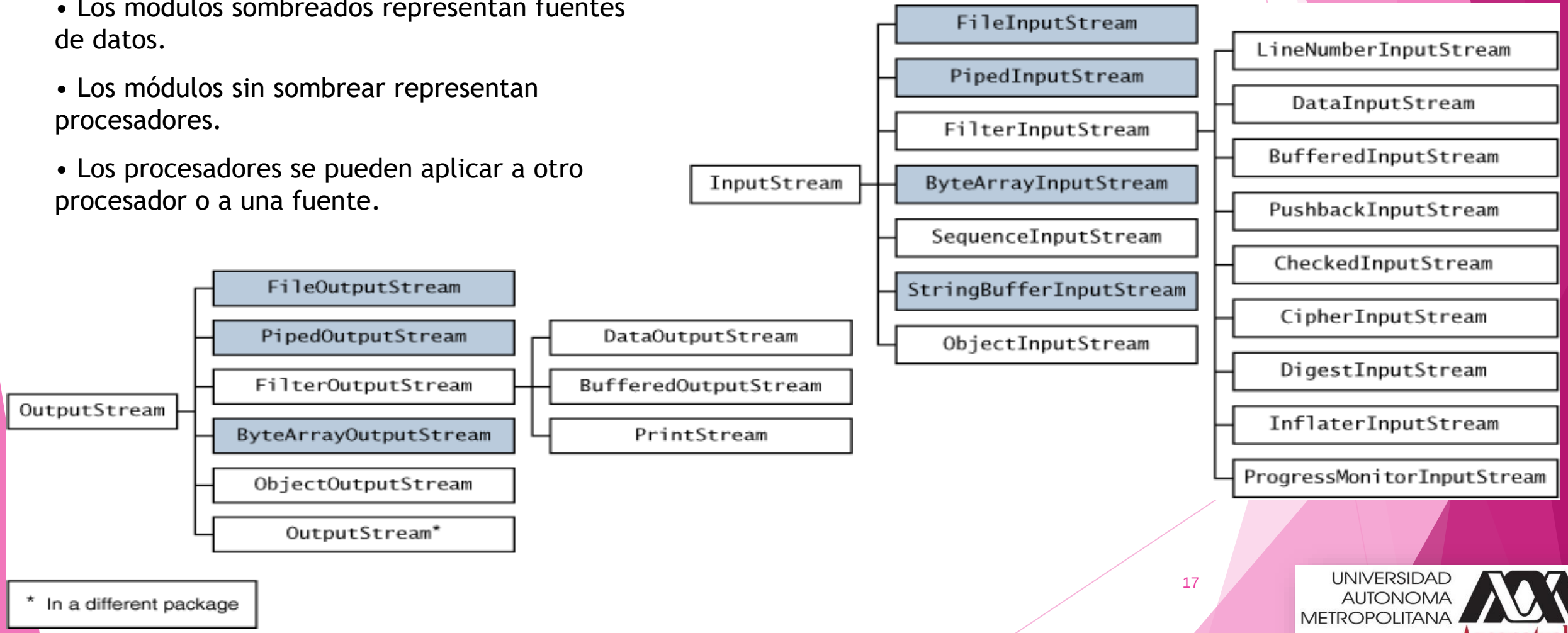
```
import java.io.*;
public class GeneraDatos {
    public static void main(String args[]) {
        try   {
            FileOutputStream fos = new FileOutputStream("stuff.dat");
            DataOutputStream dos = new DataOutputStream(fos);
            dos.writeInt(2);
            dos.writeDouble(2.7182818284590451);
            dos.writeDouble(3.1415926535);
            dos.close(); fos.close();
        }
        catch (FileNotFoundException e) {
             System.err.println(“Archivo no encontrado");
        }
        catch (IOException e) {
            System.err.println(“Falla de lectura o escritura");
        }
    }
}
```

Lectura de datos de un archivo usando Filtros

```
import java.io.*;
public class LeeDatos {
    public static void main(String args[]) {
        try {
            FileInputStream fis = new FileInputStream("stuff.dat");
            DataInputStream dis = new DataInputStream(fis);
            int n = dis.readInt();
            System.out.println(n);
            for( int i = 0; i < n; i++ ) { System.out.println(dis.readDouble());
            }
            dis.close(); fis.close();
        }
        catch (FileNotFoundException e) {
            System.err.println(" Archivo no encontrado ");
        }
        catch (IOException e) { System.err.println("Falla de lectura o escritura");
        }
    }
}
```

Clases de Streams (Streams orientados a byte)

- Los módulos sombreados representan fuentes de datos.
- Los módulos sin sombrear representan procesadores.
- Los procesadores se pueden aplicar a otro procesador o a una fuente.



Subclases de InputStream

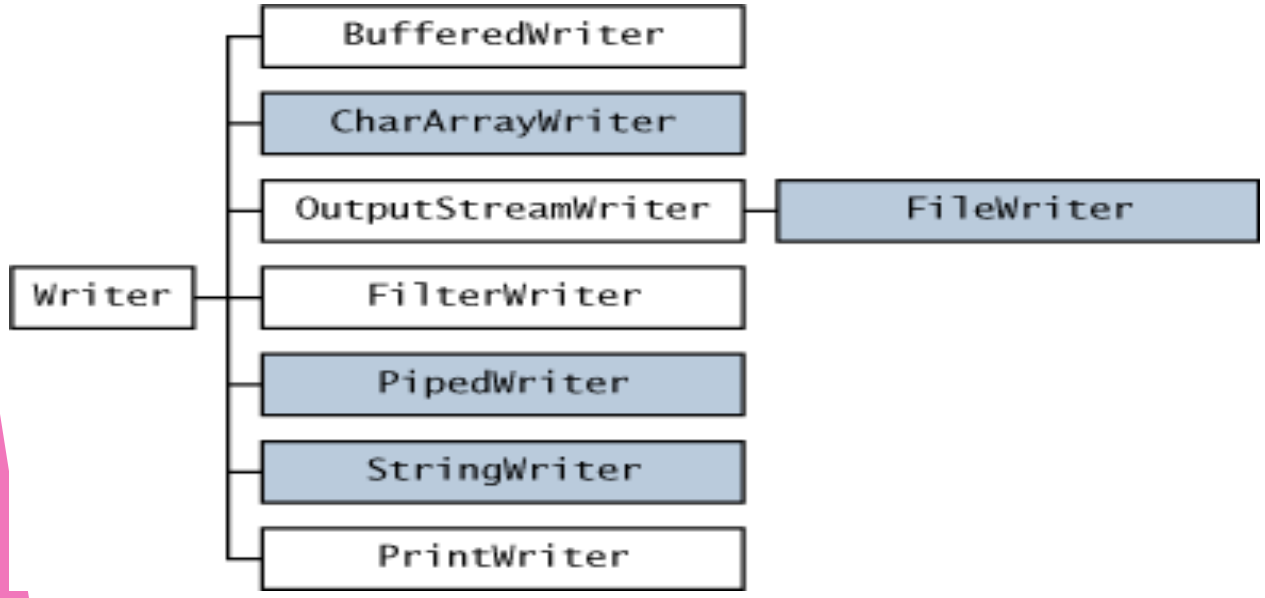
- FileInputStream: lectura de archivos byte a byte
- ObjectInputStream: lectura de archivos con objetos.
- FilterInputStream:
  - BufferedInputStream: lectura con buffer, más eficiente.
  - DataInputStream: lectura de tipos de datos primitivos (int, double, etc.).

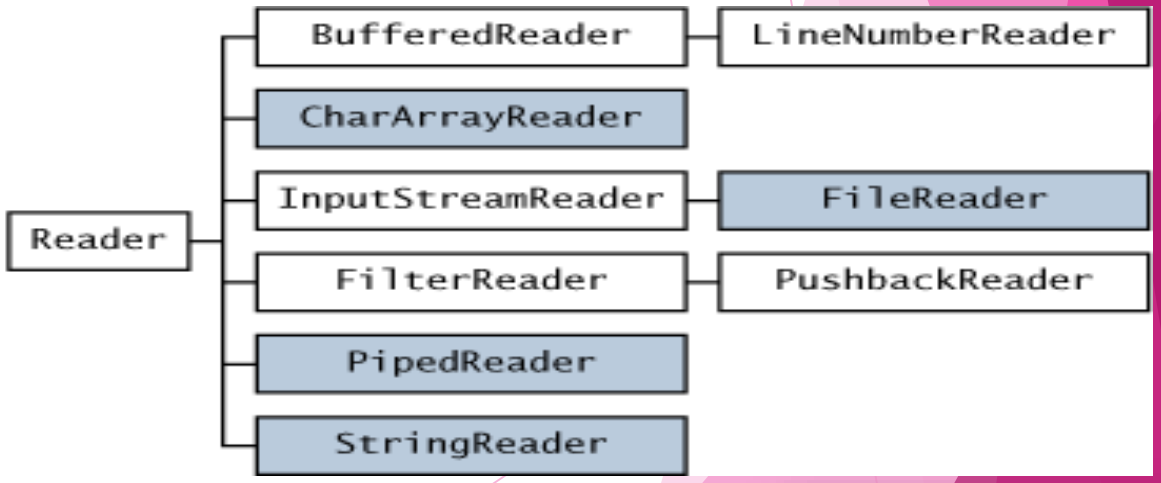
Subclases de OutputStream

- FileOutputStream: escritura de archivos byte a byte
- ObjectOutputStream: escritura de archivos con objetos.
- FilterOutputStream:
  - BufferedOutputStream: escritura con buffer, más eficiente.
  - DataOutputStream: escritura de tipos de datos primitivos (int, double, etc.).

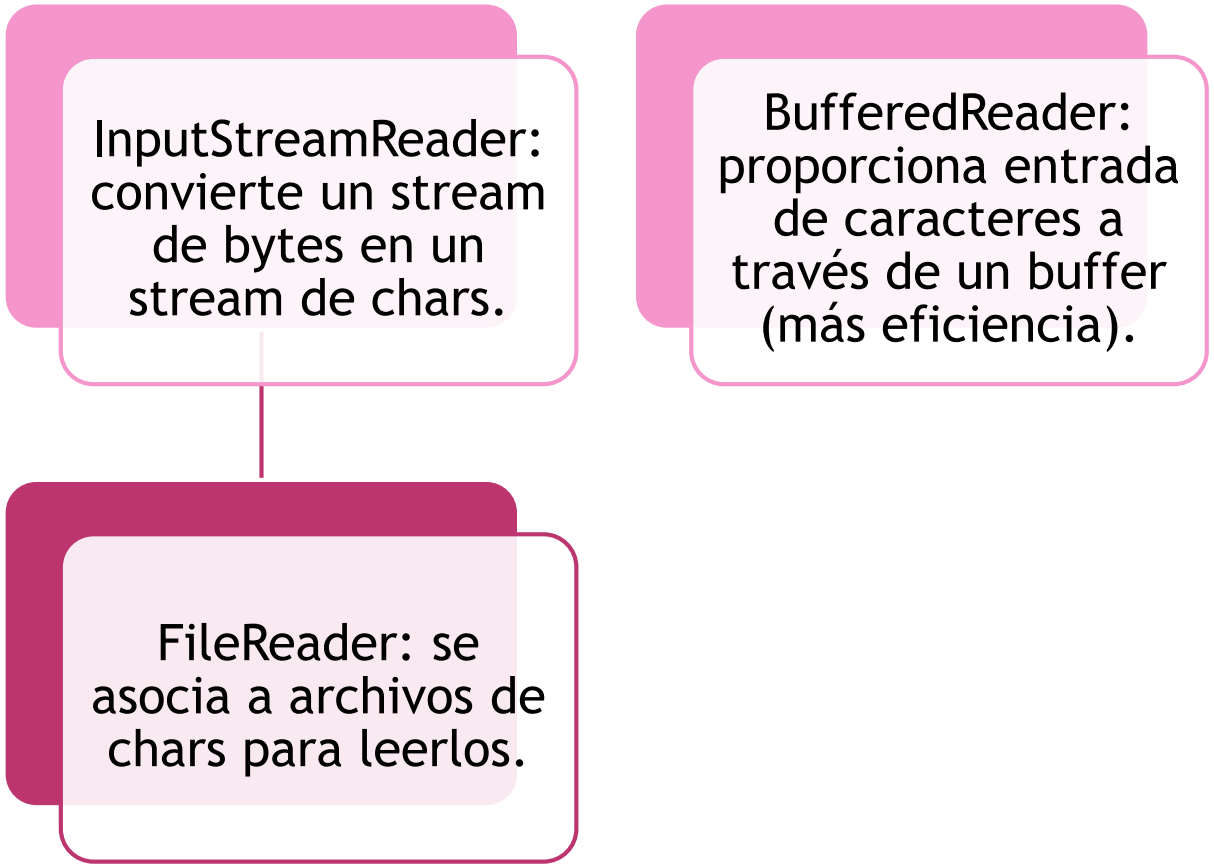
Clases de Streams (Streams orientados a caracter)

- Soportan UNICODE (16 bits para un char).
- Módulos sombreados son fuentes, y sin sombrear son procesadores.





Subclases de Reader



Ejemplo: Lectura de teclado

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class lecturaTeclado
{
	public static void main(String args[]) throws IOException
	{
		BufferedReader stdin = new BufferedReader( new InputStreamReader(System.in));
		System.out.print("Introduce un texto:");
		String texto = stdin.readLine();
		System.out.println(texto);
	}
}
```

Subclases de Writer

- OutputStreamWriter: convierte un stream de bytes en un stream de chars.
  - FileWriter: se asocia a archivos de caracteres para modificarlos.
- BufferedWriter: proporciona salida de caracteres a través de un buffer (más eficiente).
- PrintWriter: métodos *print()* y *println()* para distintos tipos de datos.

¿Qué son los puertos?

- Los puertos son representados por valores enteros positivos de 16 bits.
- Algunos puertos están reservados para soportar servicios preestablecidos:
  - FTP 21/TCP
  - Telnet 23/TCP
  - SMTP 25/TCP
  - HTTP 80/TCP
- Los procesos o servicios de usuarios generalmente usan números de puertos >= 1024.

Sockets

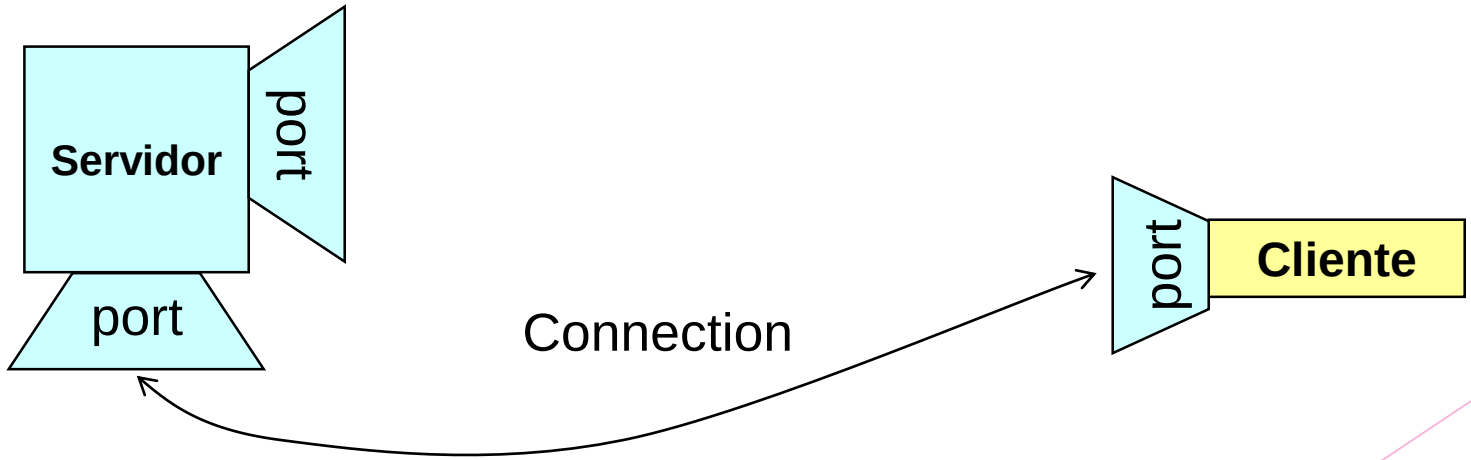
- Los sockets proporcionan una interfaz para la programación de redes en la capa de transporte.
- Las comunicaciones de redes utilizando Sockets es muy similar al manejo de I/O en archivos.
  - El manejo de sockets es tratado como el manejo de archivos.
  - Los flujos utilizados en operaciones de I/O de archivos también son aplicables a I/O basado en sockets.
- La comunicación basada en Sockets es independiente del lenguaje de programación.
  - Esto es, que un programa de socket escrito en Java también se puede comunicar con un programa escrito en Java o con un programa de socket no escrito en Java.

Comunicación entre Sockets

- Un servidor es un programa que se ejecuta en una computadora específica y tiene un socket que se asocia con un puerto específico.
- El servidor se mantiene en espera escuchando al socket para cuando un cliente realiza una petición de conexión.

Comunicación entre Sockets

- Si todo sale bien, el servidor acepta la conexión. Después de la aceptación, el servidor obtiene un nuevo socket asociado a un puerto diferente. Necesita un nuevo socket (consecuentemente un número de puerto diferente), de tal forma que puede continuar escuchando al socket original para solicitudes de conexión mientras que atiende al cliente conectado.



Clases de Java para crear Sockets

- Un socket es un endpoint de un enlace de comunicación bi-direccional entre dos programas ejecutándose en la red.
- Un socket se asocia a un número de puerto de tal forma que la capa de TCP puede identificar la aplicación a la cual están destinados los datos.
- El paquete de Java .net proporciona dos clases:
  - Socket – para implementar un cliente
  - ServerSocket – para implementar un servidor.

Servidor de chat

```
public class ServidorChat
{
	private static int port = 1001;
	public static void main (String[] args) throws IOException
	{
		ServerSocket server = null;
		try{
			server = new ServerSocket(port);
		 } catch (IOException e)
		{
			System.err.println("No puede escuchar en el puerto: " + port);
			System.err.println(e); System.exit(1);
		}
		Socket cliente = null;
		try{
			cliente = server.accept();
		}catch (IOException e)
		{
			System.err.println("Falló la aceptación de la conexión");
			System.err.println(e);
			System.exit(1);
```

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

public class ClienteChat
{
	private static int port = 1001;
	private static String host = "localhost";
	public static void main (String[] args) throws IOException
	{
		Socket servidor;
		PrintWriter out = null;
		try
		{
			servidor = new Socket(host, port);
			out = new PrintWriter(servidor.getOutputStream(), true);
		}
		catch (UnknownHostException e)
		{
			System.err.println(e);
			System.exit(1);
		}
```

Dudas O comentarios???

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco