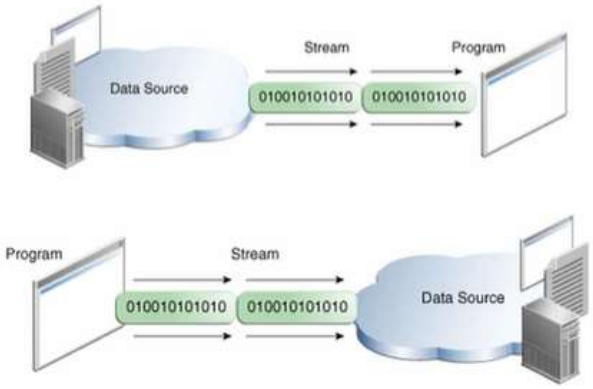
FLUJO

- Canal de comunicación de las operaciones de entrada/salida
- Literalmente, es un flujo (producción/consumo de información).

Nos da independencia:

- Entrada desde el teclado
- Salida hacia el monitor
- Lectura de un fichero
- Envío de datos por red
- ...

FLUJO

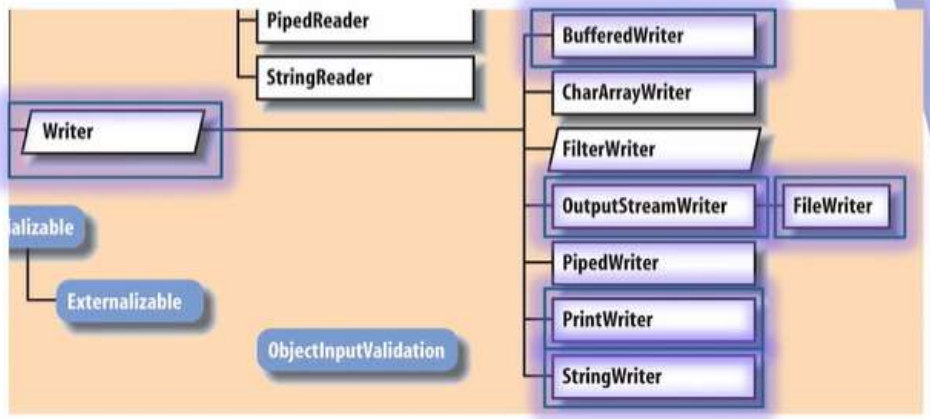


FLUJOS DE SALIDA

- Patrón básico de uso de flujos de salida:

```
Abrir el flujo
Mientras hay datos que escribir
    Escribir datos en el flujo
Cerrar el flujo
```

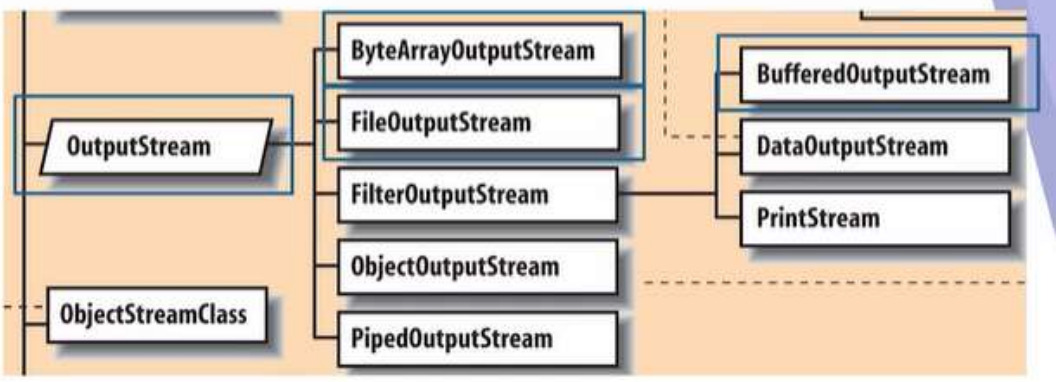
FLUJOS DE SALIDA DE CARACTERES



FLUJOS DE SALIDA DE CARACTERES

- ***Writer***: clase abstracta, padre de la mayoría de los flujos de caracteres.
- ***FileWriter***: flujo que permite escribir en un fichero, caracter a caracter.
- ***BufferedWriter***: flujo que permite escribir líneas de texto.
- ***StringWriter***: flujo que permite escribir *en memoria*, obteniendo lo escrito en un String
- ***OutputStreamWriter***: flujo que permite transformar un OutputStream en un Writer.
- ***PrintWriter***: flujo que permite escribir tipos básicos Java.

FLUJOS DE SALIDA DE BYTES

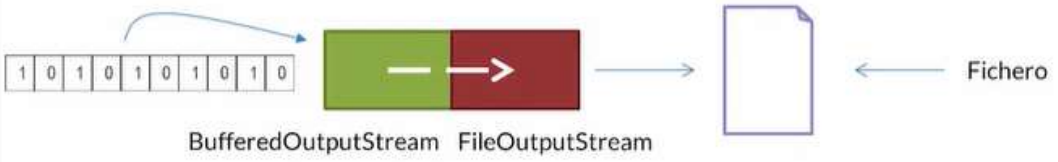


FLUJOS DE SALIDA DE BYTES

- ***OutputStream***: clase abstracta, padre de la mayoría de los flujos de bytes.
- ***FileOutputStream***: flujo que permite escribir en un fichero, byte a byte.
- ***BufferedOutputStream***: flujo que permite escribir grupos (buffers) de bytes.
- ***ByteArrayOutputStream***: flujo que permite escribir *en memoria*, obteniendo lo escrito en un array de bytes.

FLUJOS HACIA OTROS FLUJOS

- Solo *FileOutputStream* tiene un constructor que acepta una ruta (entre otras opciones).
- El resto reciben en sus constructores *OutputStream*. ¿Por qué?
- Podemos construir flujos que escriben en flujos (encadenados).



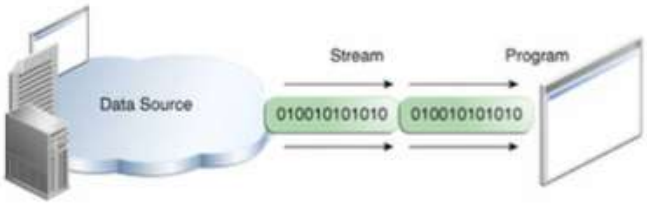
Ejemplo sencillo de escribir en un fichero binario: Escribe bytes, solo byte a byte, escribe 100 números (los mayores de 255 no porque se pasa de byte)

```
OutputStream fOut = null;

try {
    fOut = new FileOutputStream("primero.dat");
    for(int i = 0; i < 100; i++) {
        fOut.write(i);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (fOut != null)
        try {
            fOut.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

Igual que los de salida, vemos los de entrada, que pueden ser también de caracteres o de Bytes.

FLUJOS DE ENTRADA

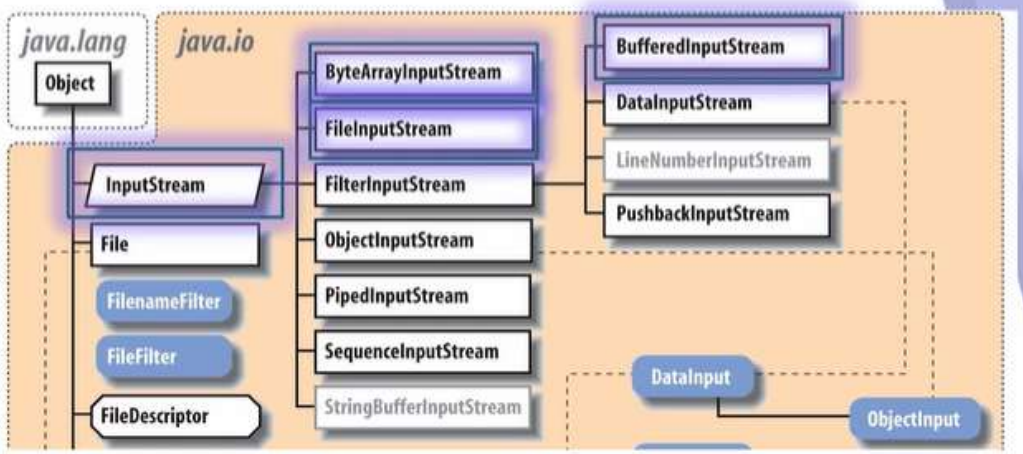


FLUJOS DE ENTRADA

- Patrón básico de uso de flujos de entrada:

```
Abrir el flujo
Mientras hay datos que leer
   Leer datos del flujo
   Procesarlos
Cerrar el flujo
```

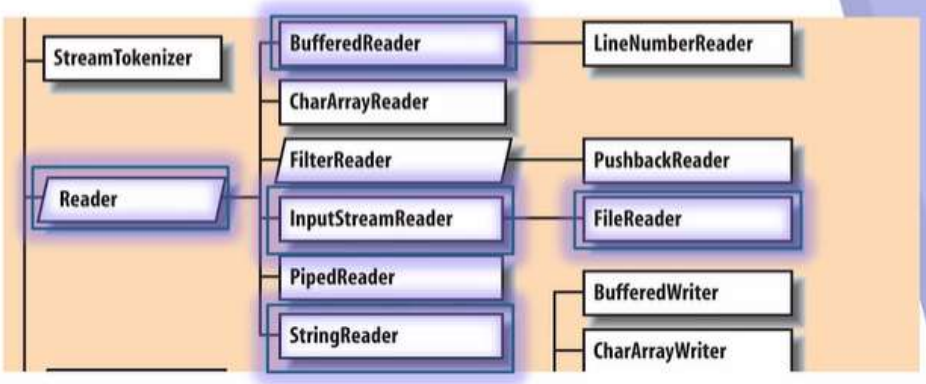
FLUJOS DE ENTRADA DE BYTES



FLUJOS DE ENTRADA DE BYTES

- ***InputStream***: clase abstracta, padre de la mayoría de los flujos de bytes.
- ***FileInputStream***: flujo que permite leer de un fichero, byte a byte.
- ***BufferedInputStream***: flujo que permite leer grupos (buffers) de bytes.
- ***ByteArrayInputStream***: flujo que permite leer *de memoria* (de un array de bytes).

FLUJOS DE ENTRADA DE CARACTERES



FLUJOS DE ENTRADA DE CARACTERES

- ***Reader***: clase abstracta, padre de la mayoría de los flujos de caracteres.
- ***FileReader***: flujo que permite leer de un fichero, caracter a caracter.
- ***BufferedReader***: flujo que permite leer líneas de texto.
- ***StringReader***: flujo que permite leer desde la memoria.
- ***InputStreamReader***: flujo que permite transformar un InputStream en un Reader.

Ejemplo: Leer el archivo dat que, en el ejemplo, habíamos escrito:

```
public static void main(String[] args) {

    FileInputStream fIn = null;

    try {
        fIn = new FileInputStream("primero.dat");
        int c;
        while ((c = fIn.read()) != -1)
            System.out.println(c);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (fIn != null)
            try {
                fIn.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
```