Unidad 11

FICHEROS EN JAVA

Objetivos

- Implementar en Lenguaje Java los conceptos aprendidos en la unidad anterior.
- Conocer las peculiaridades de Java sobre las estructuras de datos ficheros.
- Aplicar en Lenguaje Java los conceptos sobre archivos para el diseño y construcción de aplicaciones.
- Conocer algunas funcionalidades Java para la implementación y uso de ficheros: sintaxis, parámetros, restricciones, etc.
- Saber crear y procesar archivos de texto y archivos binarios.
- Conocer las ventajas e inconvenientes del sistema de archivos en Java.

Contenidos

Introducción

En la unidad anterior hemos estudiado distintos Tipos de ficheros: según su función o su organización.

Otra clasificación puede ser:

- Por la **forma de almacenamiento**:
  - ***Ficheros binarios***: Almacenan la información en el mismo formato que en memoria central **(miArchivo.dat)** .
  - ***Ficheros de texto***: Almacenan la información como secuencias de caracteres (**miArchivo.txt)**.
- **Por su contenido:**
  - ***De datos***: Almacenan datos en sentido estricto.
  - ***De programa***: Contienen instrucciones que actúan sobre los datos (Programas fuentes, obj, exe, etc...)

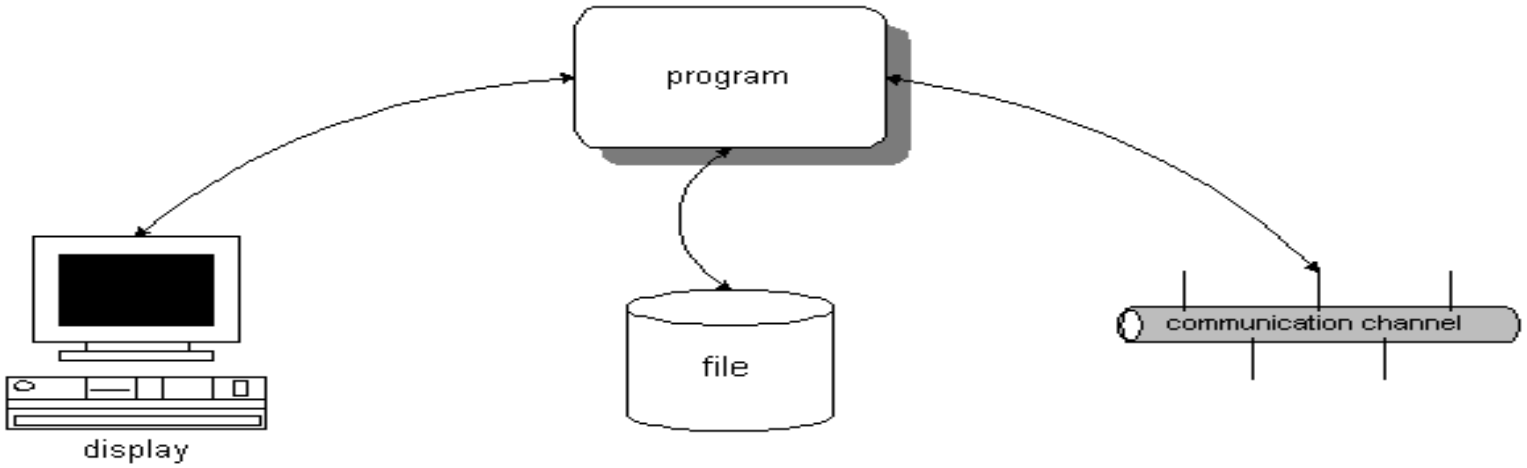
En esta unidad veremos instrucciones en Java para construir ficheros *De programa* que actúen sobre ficheros *De datos*, ya sean *almacenados como de texto o binarios*.

Entrada/salida en Java

- **Todas las operaciones de entrada y salida se hacen mediante librerías externas.**
  - **Las clases para manejo de archivos están en java.io**
- **El manejo de la entrada/salida en Java está basado en el concepto de Stream.**
  - **Un Stream es un flujo de datos entre el programa y el destino u origen de datos**
  - **La entrada/salida basada en Streams soporta la lectura o escritura de datos secuencialmente.**

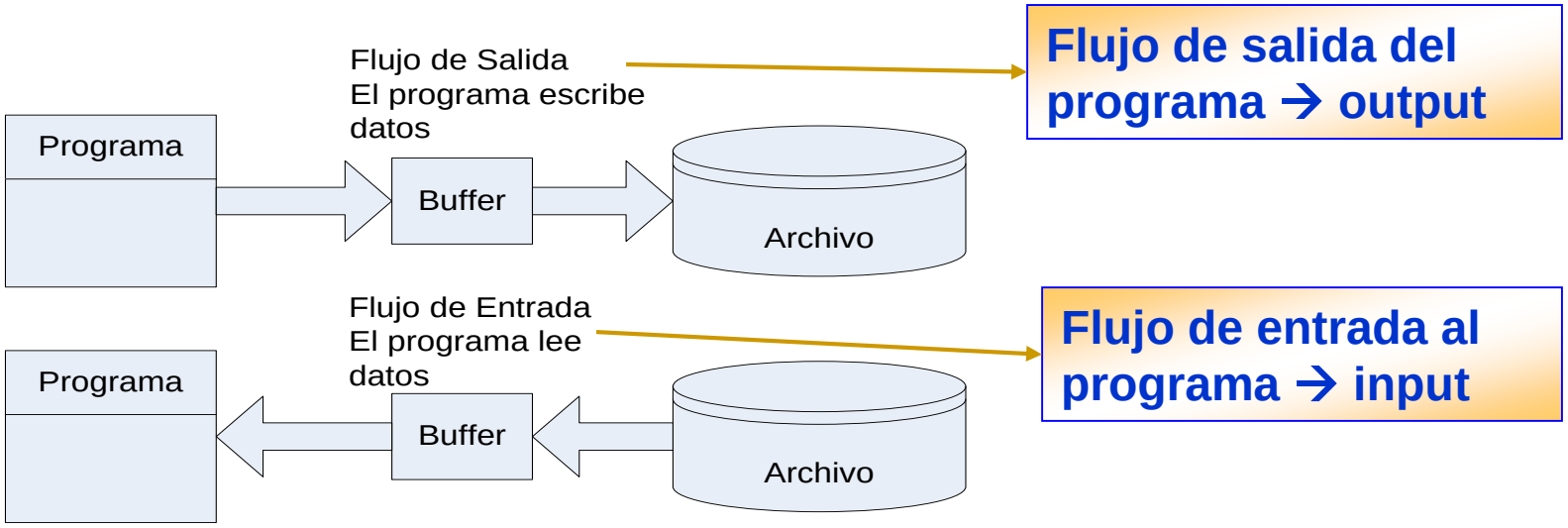
Streams (Flujo, corriente) en Java

- Las operaciones de E/S de Java se concretan como corrientes de datos que salen/entran del programa hacia/desde distintos dispositivos.
- Desde el punto de vista del programa el “dispositivo externo” se entiende como archivo hacia/desde el que fluye información.
- Aunque cada dispositivo es diferente, el programa utiliza un solo concepto de corriente de datos, denominado *Stream* (flujo).



Streams (Flujo, corriente) en Java

- El sistema Java de E/S desliga el concepto de flujo del dispositivo concreto con el que se realiza la transferencia.
- El flujo de datos es una sucesión de bits que pueden representar cualquier cosa: tipos primitivos (**int** o **float**), tipos definidos por el usuario (**enum, class**), caracteres, píxels de una imagen, o notas de una melodía.



Tipos de Streams

- El modo en que se realizará un flujo queda especificado en el momento de abrir el fichero asociado. Puede ser:

  De ***texto y binario***

  - El paquete **java.io** tiene dos jerarquías de clases que dividen los streams según su contenido
  - ***Character Streams***, de ***texto*** .
    - Realizan operaciones de entrada y salida de caracteres
    - La unidad de flujo es 16 bits en UNICODE
    - Utilizados para archivos de texto.
    - se les llama readers o writers.
  - ***Byte Streams*, binarios**
    - Realizan operaciones de entrada y salida de bytes.
    - La unidad de flujo es 8 bits
    - Utilizados para archivos binarios (imagénes, sonidos, etc.).
    - se les llama input streams o output streams.

Tipos de Streams

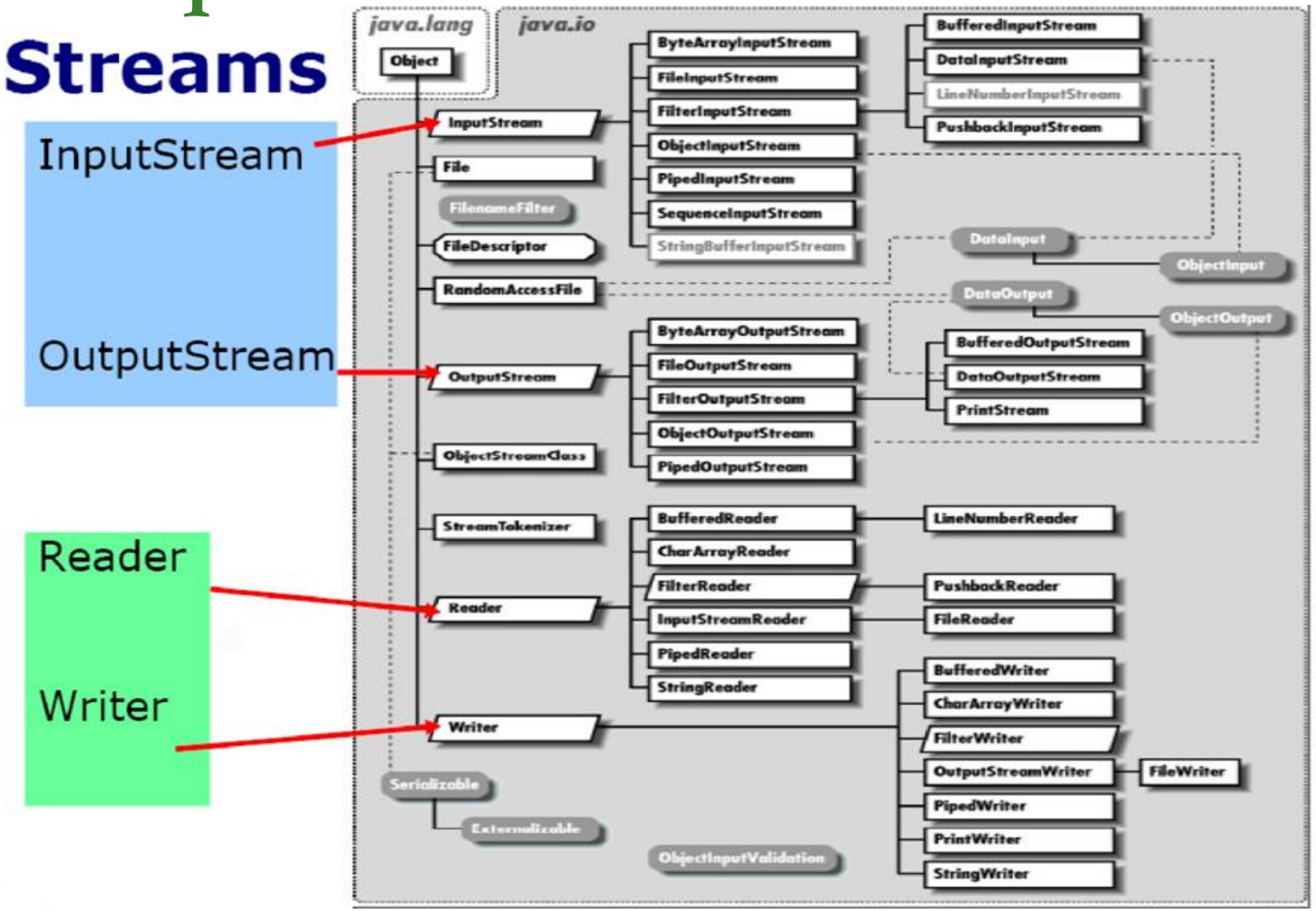
- También se puede considerar una división de las clases de **java.io** por su propósito
- ***Data Sink Streams***
  - Son fuentes u orígenes de datos.
  - Deben estar asociados a archivos que son fuente o destino almacenados en disco.
- ***Processing Streams***
  - Realizan algún tipo de operación de procesamiento de los datos leídos o escritos de una fuente: almacernarlos en buffer, descodificar caracteres, etc.
  - Necesitan estar asociados a un Data Sink Stream.

Estándar Streams en Java

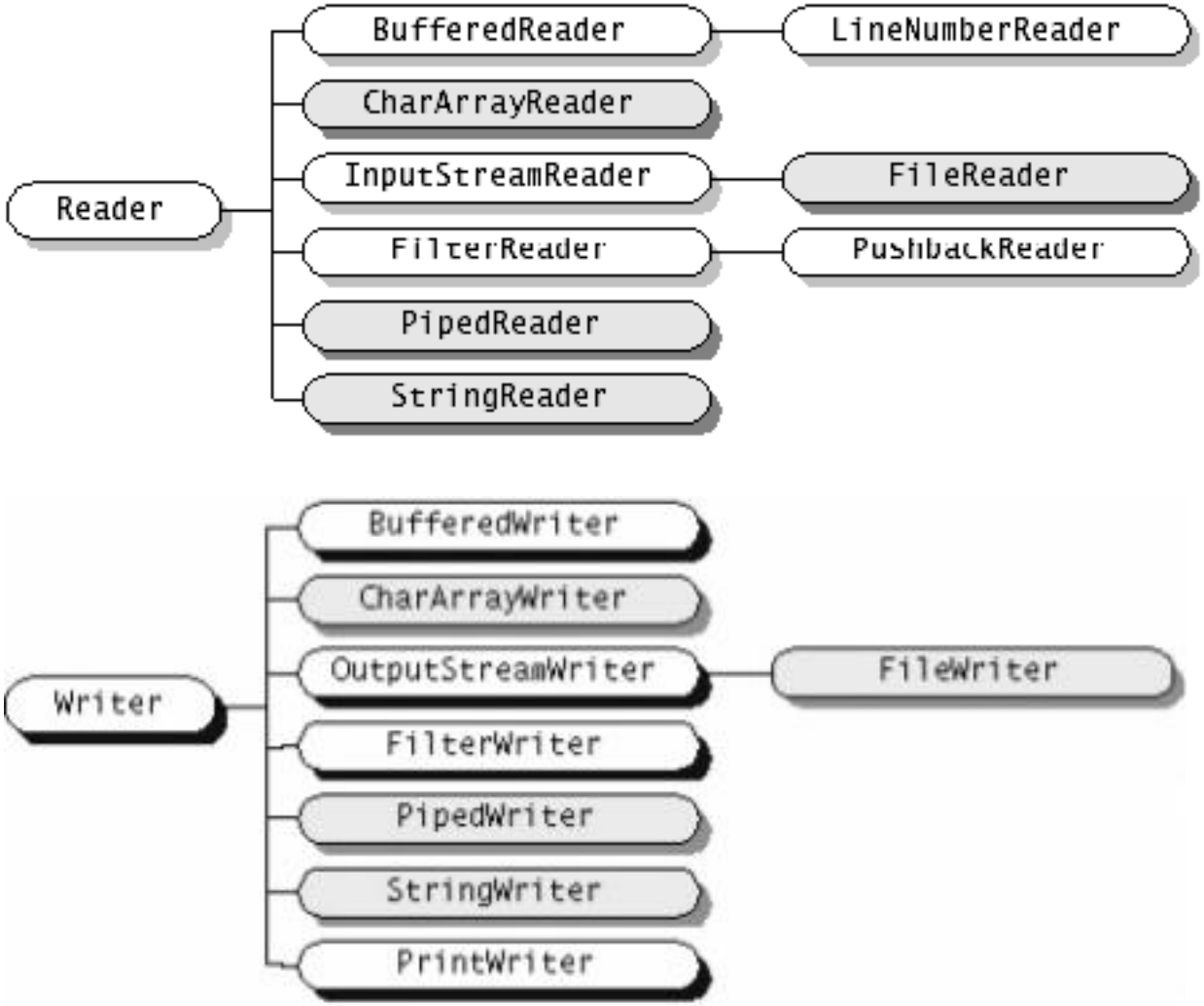
- **Tres estándar streams**
- **Estos objetos son definidos automáticamente y no requieren ser abiertos**
- **System.out y System.err son definidos como objetos PrintStream**

Nombre del flujo	Acceso (Dispositivo asociado)
Input	Accedido a través de System.in (teclado)
Output	Accedido a través de System.out (pantalla)
Error	Accedido a través de System.err (normalmente la pantalla)

Jerarquía de la clase Stream



Jerarquía de Character Stream



Jerarquía de la clase Byte Stream

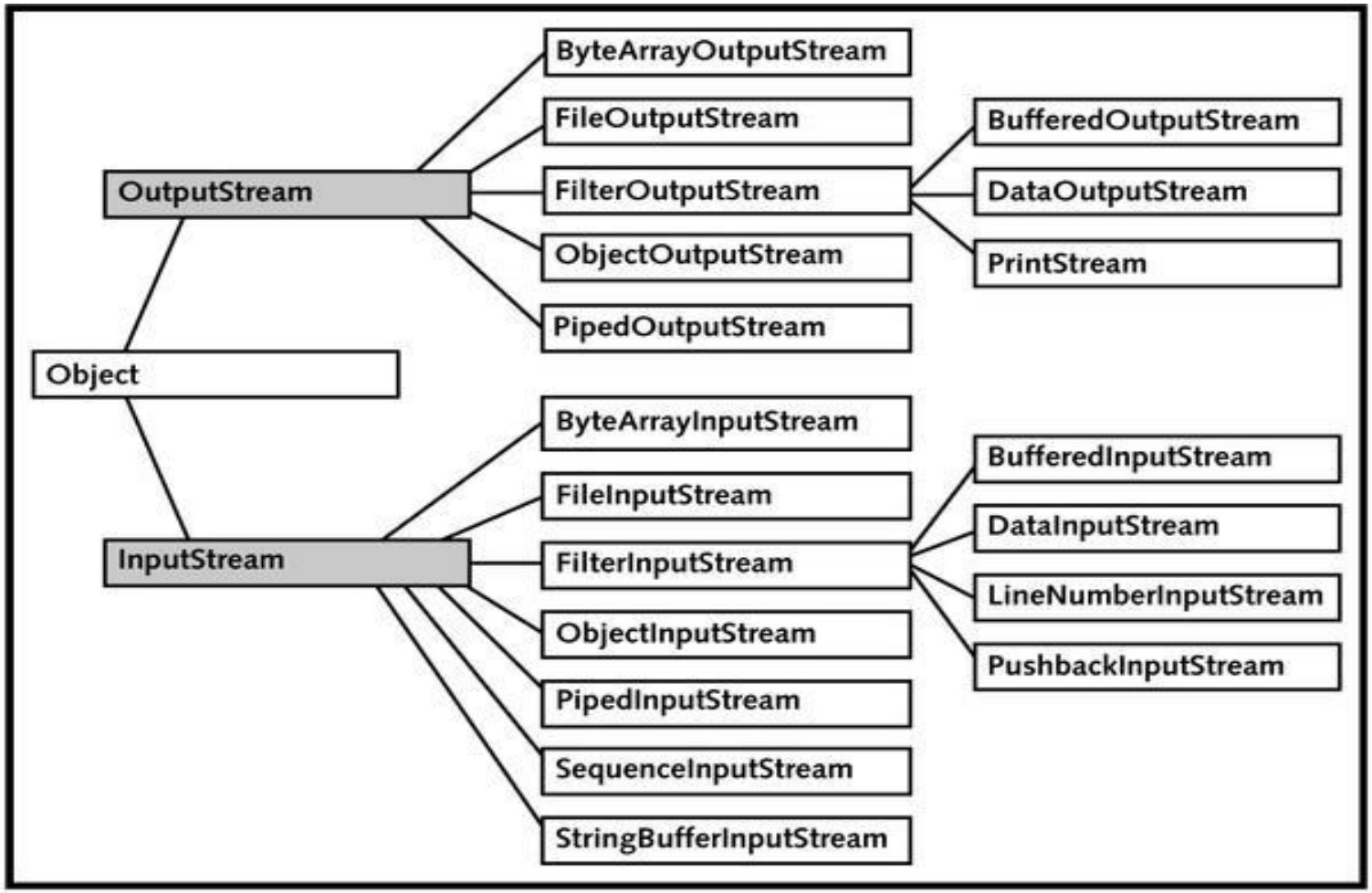


Figure 8-1 Byte-oriented stream classes

Jerarquía Character y Byte Stream

Todas las clases tienen:

Constructores: abren el archivo

métodos: write, read, close

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

- **Character Streams**
  - **Para caracteres Unicode**
  - **Clases raíz de Character Streams (ambas abstractas):**
    - **Clase Reader:** Se usa para leer datos de una fuente
    - **Clase Writer:** Se usa para escribir datos en un destino
- **Byte Streams**
  - **Para datos binarios**
  - **Clases raíz de Byte Streams (ambas son abstractas):**
    - **Clase InputStream:** is used to read data from a source.
    - **Clase OutputStream:** used for writing data to a destination.

FileReader/FileWriter

- **Subclases de Reader y Writer (texto)**
- **Leen o escriben: 1 carácter, un array de caracteres, Strings.**
- **Son fuente o destino de datos (Data Sink Streams).**
- **Lanzan IOException**
- **FileReader:**
  - **Constructores**
  - **Métodos; read, skip, ready, close**
- **FileWriter**
  - **Si al crear un nuevo flujo de salida el archivo destino ya existe, se pierden los datos que tuviera, si no existe se crea un archivo nuevo.**
  - **Constructores**
  - **Métodos: write, flush, close**

BufferedReader/BufferedWriter

- **Subclases de Reader y Writer (texto)**
- **Se usan para evitar que cada lectura o escritura acceda directamente a un archivo.**
- **Son Streams de procesamiento (Processing Streams), por tanto, necesitan estar asociados a un Data Sink Stream.**
- **Constructores: ¡OJO! reciben como parámetros un Reader o Writer o alguno de sus descendientes.**
- **Métodos; readxx, writexx, close**
- **Ejemplo**

Fuente o destino asociado

Data Sink

```
FileWriter flS=new FileWriter("./src/Agenda.txt");
BufferedWriter fS=new BufferedWriter(flS);
```

processing

FileInputStream/FileOuputStream*

- Subclases de InputStream y OuputStream (binario)
- Leen o escriben 1 byte o un array de bytes
- Son fuente o destino de datos (Data Sink Streams).
- Lanzan IOException
- FileInputStream:
  - Constructores
  - Métodos; read, skip, close
- FileOuputStream
  - Si al crear un nuevo flujo de salida el archivo destino ya existe, se pierden los datos que tuviera, si no existe se crea un archivo nuevo.
  - Constructores
  - Métodos: write, flush, close

* Son las más usuales en ficheros binarios

DataInputStream/DataOuputStream

- **Subclases de InputStream y OuputStream (binario)**
- **Leen o escriben tipos primitivos directamente bytes**
- **Son Streams de procesamiento (Processing Streams), por tanto, necesitan estar asociados a un Data Sink Stream.**
- **Constructores: ¡OJO! reciben como parámetros un InputStream o OutputStream o alguno de sus descendientes.**
- **Métodos; readxx, writexx, ready, close**
- **Ejemplo**

```
DataInputStream miFlujo= new DataInputStream(new
FileInputStream("nombreFichero.ext"));
DataOutputStream miFlujo= new DataOutputStream(new
FileOutputStream("nombreFichero.ext"));
```

InputStreamReader/OuputStreamWriter

- **Subclases de Reader y Writer**
- **Sirven de unión entre Character Streams y Byte Streams**
- **Los caracteres escritos en un OutputStreamWriter son transformados en bytes y los bytes leídos de un InputStreamReader son transformados en caracteres**
- **Son Streams de procesamiento (Processing Streams), por tanto, necesitan estar asociados a un Data Sink Stream.**
- **Constructores: ¡OJO! reciben como parámetros un InputStream o OutputStream o alguno de sus descendientes.**
- **Métodos; readxx, writexx, ready, close**
- **Ejemplo**

```
BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
```

RandomAccessFile

- **Subclase de Object**
- **Sirve para leer y escribir directamente en un fichero. Es decir, se puede leer o escribir en cualquier parte del archivo.**
- **Para posicionarse se utiliza un puntero que debe colocarse en la posición a partir de la que queremos leer o escribir.**
- **El puntero se desplaza los bytes accedidos.**
- **Si el puntero está al final y después se escribe añade datos, pero si está en medio sobreescribe.**
- **Si en la lectura se llega al final se lanza EOFException.**
- **Si hay algún error durante el proceso se lanza IOException.**

RandomAccessFile

- **Constructores:**

```
public RandomAccessFile(String nombreArchivo,
   String modo)
```

  - **Crea un stream "de fichero de acceso aleatorio", para leer de o escribir en el fichero especificado por nombreArchivo.**
  - **El argumento `modo` especifica el modo de acceso:**
    - `r` **si sólo se desea leer.**
    - `rw` **si se desea leer y escribir. En este caso si el fichero no existe lo crea.**

RandomAccessFile

- **Métodos: readxx, writexx, close**
- **Otros:**
  - `long getFilePointer ()` **: Indica la posición del puntero del archivo, en bytes desde el principio del fichero.**
  - `void seek (long pos)` **: coloca el puntero a tantos bytes del inicio como indique** `pos`
  - `void skipBytes (int cont)` **: mueve el puntero** `cont` **bytes. Si** `cont` **es positivo se mueve hacia el final y si es negativo hacia el principio.**
  - `long lenght ()` **: devuelve el tamaña en bytes del fichero.**

Object Streams

- **Si se desea escribir un objeto en un fichero:**
  - **Pasar a String y escribirlo.**
  - **Escribir sus atributos por separado.**
  - **Escribir el objeto completo en un Stream, lo adecuado es usar los object streams.**
- **Para que el objeto pueda ser escrito usando object streams hay que serializarlo, lo que significa que la clase del objeto debe implementar la interfaz serializable (¡Ojo!, esta clase no tiene métodos).**
- **Los valores almacenados son además de los atributos los necesarios para identificar el objeto.**
- **Clases básicas:**
  - **ObjectInputStream.**
  - **ObjectOutputStream**

¡Cuidado con los Object Streams!

- **Cuidado al escribir un Objetct Streams**
- Cuando a **ObjectOutputStream** le damos un objeto para escribir tiene algunos problemas**.**
- **Cuidado con las cabeceras añadidas cada vez que se accede el fichero para escribir.** Al instanciarlo escribe unos bytes de cabecera en el fichero
- Para más información mira **“Leer y Escribir Objetos Java en un Fichero.doc”.**

Evita estos errores

- **No intentes utilizar directamente Streams de procesamiento (Processing Streams). Deben estar asociados a una fuente u origen de datos.**
- **Cuidado al grabar Strings en archivos de acceso directo. Recuerda que los Strings son de longitud variable.**
- **Hay que Serializar las clases si se desea guardar objetos como tales.**
- **No intentes combinar ObjectStream y acceso directo (esto no es posible en Java).**

- **Recuerda que la utilización del constructor abre el archivo**

Recomendaciones

- **Recuerda que cuando se pasa el path de un archivo como parámetro debe escribirse doble barra \\ para señalar directorio, ya que \ es un carácter especial (\n,\v,…).**
- **Recuerda leer detenidamente la documentación de las funciones sobre archivos que uses, leer, escribir, cerrar y presta especial atención a las excepciones que lanzan.**
- **Si se desea almacenar objetos en archivo de texto, es normal convertir el objeto en String y almacenar éste. En este caso y para que la lectura sea sencilla, al construir el String se puede utilizar un carácter separador entre cada atributo del objeto, de manera que al leer el String del archivo posteriormente, este carácter nos pueda servir para conocer qué parte del String corresponde a cada atributo.**

“Tan poco hecho...Tanto por hacer.”
Cecil Rhodes, Estadista inglés
(1853-1902)