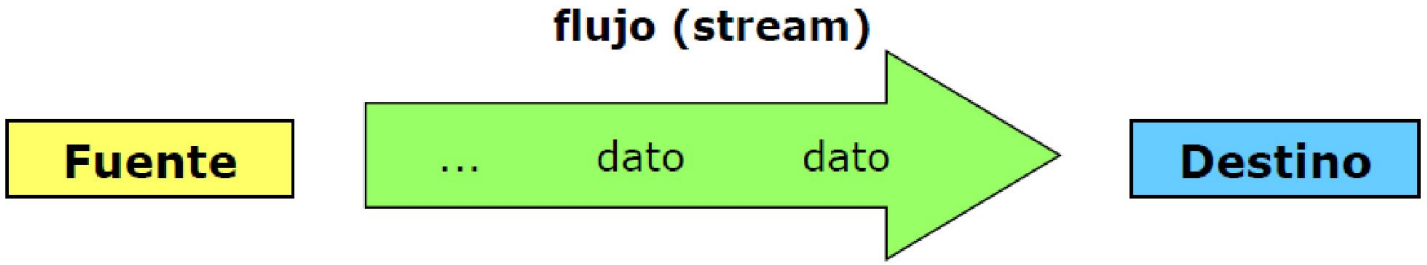
Entrada / Salida en Java

E/S con flujos (streams)

- Un stream es un flujo de una fuente a un destino. Tratar la comunicación de información entre el programa y el exterior (teclado, pantalla, archivos, sockets de red, ...).
- Los flujos son secuencias ordenadas de datos que tienen una fuente (flujos de entrada) o un destino (flujos de salida).



Datos en los Flujos

- **Flujo de Bytes**

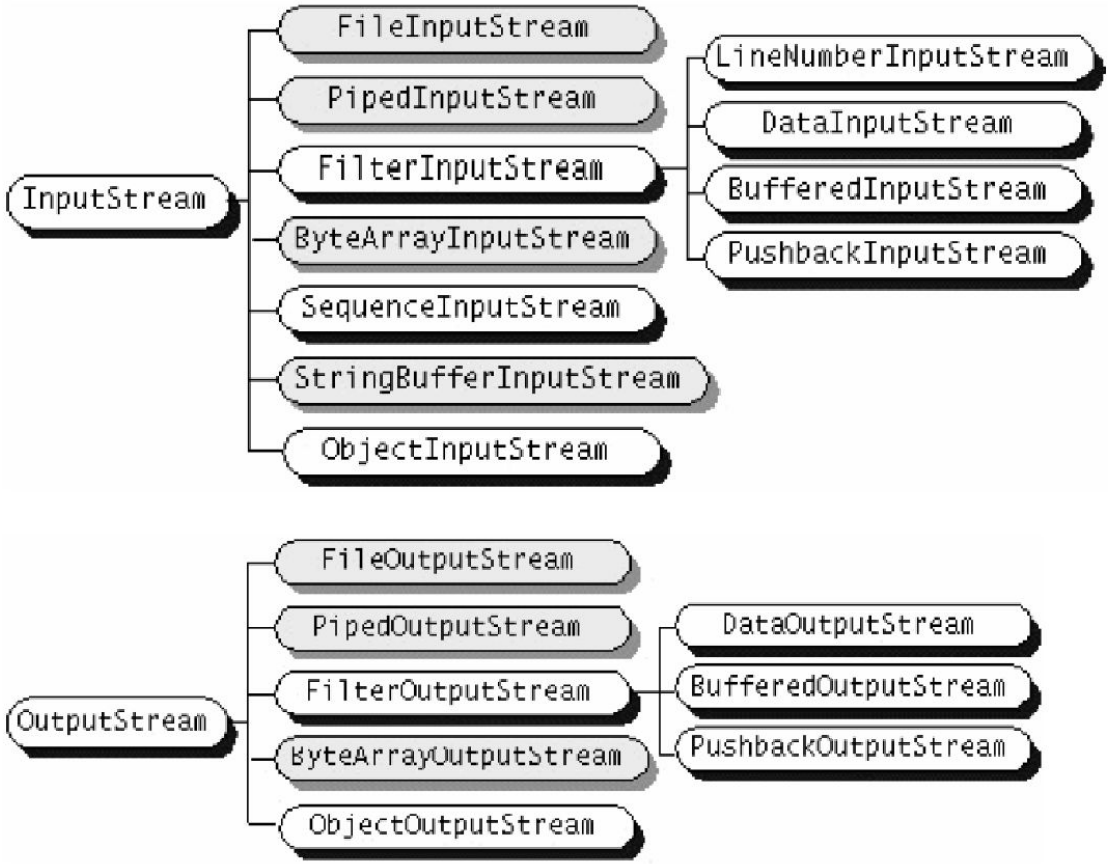
Los flujos de entrada de bytes están implementados por subclases de la clase InputStream y los de salida por subclases de la clase OutputStream.

- **Flujo de Caracteres**

Los flujos de entrada de caracteres están implementados por subclases de la clase Reader y los de salida por subclases de la clase Writer.

El paquete **java.io**, es el que contiene todos las clases que representan estos flujos.

Las Clases de Flujo de Byte Básicas



Entradas/Salidas de bytes y su uso lógicamente está orientado a la lectura y escritura de datos binarios.
El tratamiento del flujo de bytes viene gobernado por dos clases abstractas que son InputStream y OutputStream.
Cada una de estas clases abstractas tienen varias subclases concretas que controlan las diferencias entre los distintos dispositivos de I/O que se pueden utilizar. Así mismo, estas dos clases son las que definen los métodos que sus subclases tendrán implementados y, de entre todas, destacan las clases read() y write() que leen y escriben bytes de datos respectivamente.

Métodos de InputStream

Principales métodos:

- Métodos que proveen acceso a los datos en el flujo de entrada:
  - int *read()* //lee un único carácter.
  - int *read*(byte[] buffer) //leen el flujo en un arreglo y devuelve la cantidad de bytes leídos.
  - int *read* (byte[] buffer, int offset, int length) // indica el subrango en el arreglo de destino a ser completado.

Nota: siempre es recomendable leer datos usando *buffer*.

- void close() // cierra el flujo.
- Long skip(long n) //descarta la cantidad especificada de bytes del flujo.

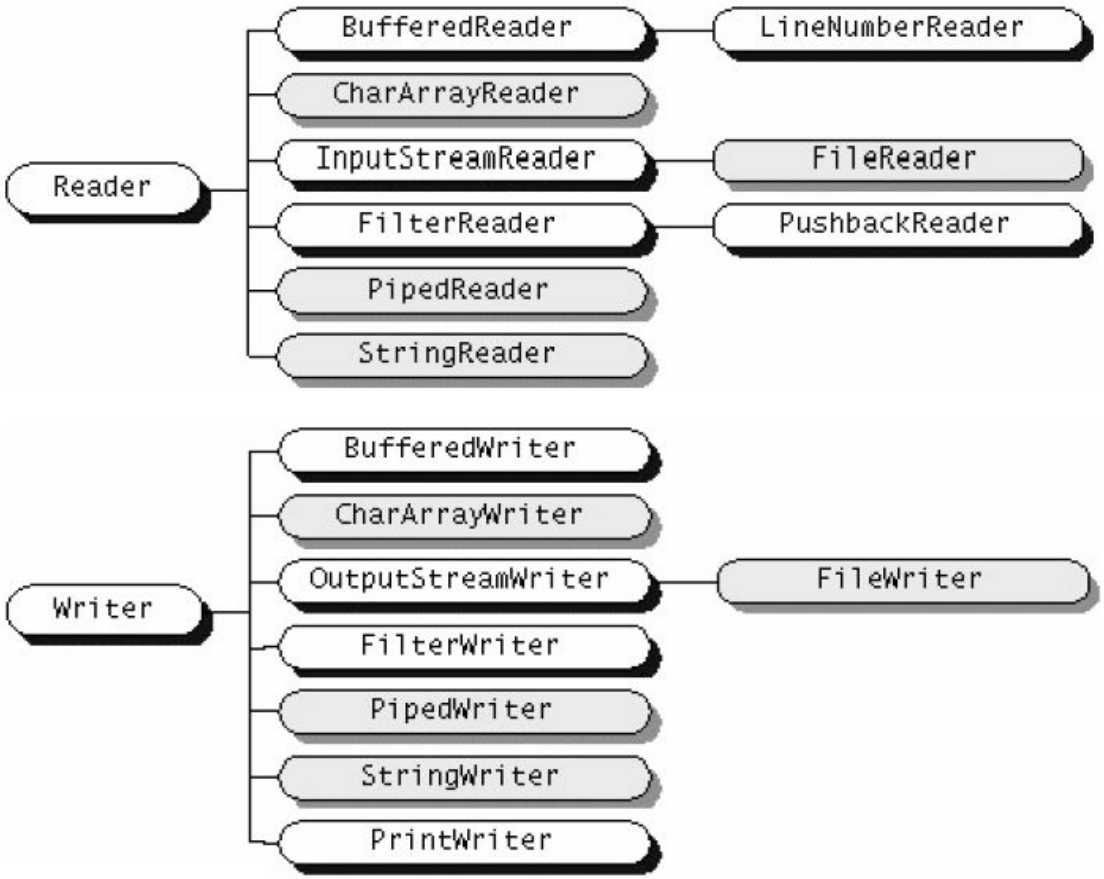
Métodos de OutputStream

Principales métodos:

- Métodos que escriben los datos en el flujo de salida:
  - void write()
  - void write (byte[] buffer)
  - void write (byte[] buffer, int offset, int length)

- void close() // cierra el flujo.
- void flush() //vacía el flujo, de forma que los bytes contenidos en un buffer sean expulsados del flujo.

Las Clases de Flujo de Caracteres



Este es un modo que Java nos proporciona para manejar caracteres, pero al nivel más bajo todas las operaciones de I/O son orientadas a byte. Al igual que la anterior el flujo de caracteres también viene gobernado por dos clases abstractas: Reader y Writer. Dichas clases manejan flujos de caracteres Unicode. Y también de ellas derivan subclases concretas que implementan los métodos definidos en ellas siendo los más destacados los métodos read() y write() que, en este caso, leen y escriben caracteres de datos respectivamente.

Los Métodos Reader

Principales métodos de lectura son:

- int *read()* //lee un único carácter.
- int *read*(byte[] buffer) //leen el flujo en un arreglo y devuelve la cantidad de bytes leídos.
- int *read* (byte[] buffer, int offset, int length) //indica el subrango en el arreglo de destino a ser completado.

Nota: siempre es recomendable leer datos usando buffer.

- void close() // cierra el flujo.
- Long skip(long n) //descarta la cantidad especificada de bytes del flujo.

Los Métodos Writer

Los métodos de escritura más usuales son:

- void write (int b) // escribe en el flujo de salida los ocho bits de menor orden del argumento b.
- void write(byte [ ] b)// escribe en un flujo de salida todos los bytes del array b.
- void write (byte [ ] b, int offset, int length) // escribe length bytes desde el array b con un desplazamiento de offset, en el flujo de salida.
- void close() // cierra el flujo.
- void flush() //vacía el flujo, de forma que los bytes contenidos en un buffer sean expulsados del flujo.

Flujos estándar

En Java se accede a la E/S estándar a través de tres variables públicas estáticas de la clase ***java.lang.System***

- ***System.in*** *es un objeto InputStreqm que referencia al teclado del usuario.*
  - Métodos:
    - *read() //*permite leer un byte de la entrada como entero.
    - *skip(n) //*ignora n bytes de la entrada.
    - *available() //*número de bytes disponibles para leer en la entrada.
- ***System.out*** es un objeto *PrintStream* que referencia a la ventana de comandos que lanzo la aplicación de tecnología Java.
  - Métodos para impresión de datos:
    - *print(), println()*
    - *flush() //vacía el buffer de salida escribiendo su contenido.*
- ***System.err es un objeto PrintStream que referencia a la ventana de comandos que lanzó la aplicación de tecnología Java.***

*Es posible cam*biar esta asignación de flujos usando los métodos static: System.setOut, System.setIn y System.setErr. (Por ejemplo se puede cambiar el flujo estándar de error a un

Ejemplo Flujo estándar

```
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 +" bytes en total." );
      }
}
```

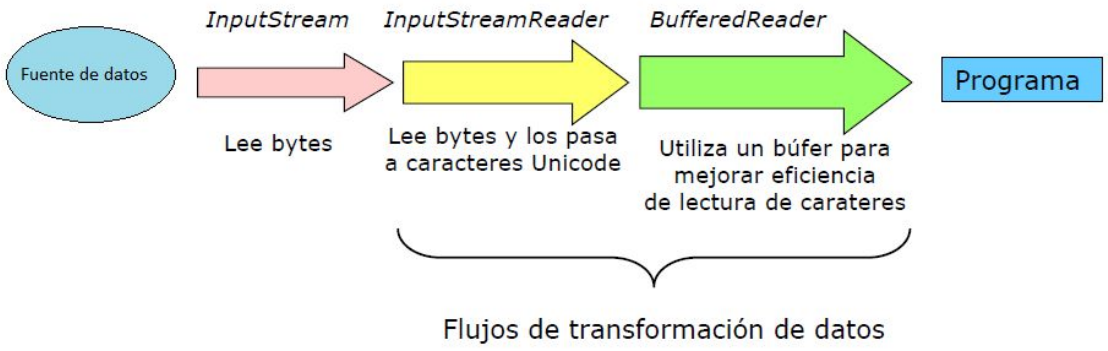
Los Buffer

- Las operaciones de E/S son costosas.
- El uso de buffer incrementa la eficiencia de las operaciones de E/S.
- Los datos se almacenan en una memoria temporal antes de ser realmente leídos o escritos.
- Existen clases que manejan los buffer: BufferedReader /Bufferedwriter para flujos de caracteres y BufferedInputStream /BufferedOutputStream para flujos de bytes.
- Ejemplo: lectura de una línea como cadena ingresada desde el teclado.

```
InputStreamReader entrada = new InputStreamReader(System.in);
BufferedReader teclado = new BufferedReader (entrada);
String cadena = teclado.readLine();
```

Encadenamiento de flujos de E/S

- Un programa, raramente usa un único objeto de flujo. Generalmente encadena una serie de flujos para procesar los datos.
- Ejemplo:



La Entrada/Salida de Archivos

Clase *File*

- Se encuentra en el paquete java.io
- Nos proporciona métodos para acceder al contenido del archivo.
- Provee métodos para operaciones a nivel de sistema de archivos .
- Un objeto File representa un archivo o un directorio

La Entrada/Salida de Archivos (continuación)

Clase *File*

- **Constructores**
  - File(String ruta)
  - File(String ruta, String nombre)
  - File(File directorio, String nombre)
- **Métodos**
  - canRead() comprueba si el fichero se puede leer
  - canWrite() comprueba si el fichero se puede escribir
  - delete() borra dicho fichero
  - getPath() devuelve la ruta del fichero
  - mkdir() crea un directorio con la ruta del objeto que lo recibe
  - isDirectory() comprueba si dicho fichero es un directorio
- **Constructores de otras clases**
  - FileReader(File fichero)
  - FileWriter(File fichero)

La Entrada/Salida de Archivos (continuación)

La entrada de archivos puede ser:

- Usando la clase FileReader para leer caracteres.
- Usando la clase BufferedReader para usar el método readLine.
  Nota: Recordar que BufferedReader se crea asociado a un flujo de entrada ser de bytes (Input/OutputStream) o de caracteres (Reader/Writer).

La salida de archivos puede ser:

- Usando la clase FileWriter para escribir caracteres.
- Usando la clase PrintWriter para usar los métodos print y println.

Clase *Scanner*

- Scanner es una clase en el paquete **java.util** utilizada para obtener la entrada de los tipos primitivos como int, double, etc. y también String.

- Es **la forma más fácil de leer datos** en un programa Java, aunque no es muy eficiente. En versiones nuevas, esto no sería tan relevante, gracias a que Scanner, al igual que PrintWriter, usan buffers internos.

Las características más importantes son:

- Un Scanner divide su entrada en tokens utilizando un patrón delimitador, que por defecto coincide con los espacios en blanco.

- Una operación de Scanner puede bloquear la espera de entrada.

- Un Scanner no es seguro para uso multiproceso sin sincronización externa.

Scanner

- Para crear un objeto de clase Scanner, normalmente pasamos el objeto predefinido **System.in**, que representa el flujo de entrada estándar. Podemos pasar un objeto de clase **File** si queremos leer la entrada de un archivo. La clase Scanner tiene ocho constructores, por ejemplo:
  - **Scanner (new File (String ruta))**; //Construye un nuevo escáner que produce valores escaneados del archivo especificado.
  - **Scanner(new File (String ruta), String charsetName)**; //construye un nuevo escáner que produce valores escaneados del archivo especificado.

- Para leer valores numéricos de un determinado tipo de datos XYZ, la función que se utilizará es nextXYZ(). Por ejemplo, para leer un valor de tipo *int*, podemos usar **nextInt()**.

- Para leer cadenas (strings), usamos **nextLine()**.

- Para leer un solo carácter, se usa **next().charAt(0)**. La función next()devuelve el siguiente token/palabra en la entrada como cadena y la función charAt (0) devuelve el primer carácter de esa cadena.

- El método **hasNextLine(),** este método determina si el archivo contiene líneas que permiten ser leídas, retorna un valor booleano (true/false)

Ejemplos

https://github.com/paradigmas-de-programacion/workspace/tree/master/basicas-entrada-salida

Bibliografía

- Programación Java (SL-275-SE6). Sun. Microsystems
- https://www.discoduroderoer.es/entrada-y-salida-de-datos-en-java/
- https://java.ciberaula.com/articulo/java_io
- https://www.tutorialspoint.com/java/io/java_io_filereader.htm
- https://sites.google.com/site/prograunlam/descargas-1
- https://www.javatpoint.com/java-bufferedoutputstream-class
- https://www.tutorialspoint.com/java/number_parseint.htm