
Java
Streams


Java Streams
dropWhile()
takeWhile()
Java

E/S en Java

- Los programas de Java realizan la **E/S a través de flujos (stream).**
- Por tanto, se pueden aplicar **las mismas clases y métodos** E/S **a cualquier** tipo de **dispositivo.**
- **Java implementa los flujos en las clases incluidas en el paquete java.io**

E/S en Java

El sistema de E/S de Java es extenso, con numerosas clases, interfaces y métodos.

Java define dos sistemas completos de E/S:

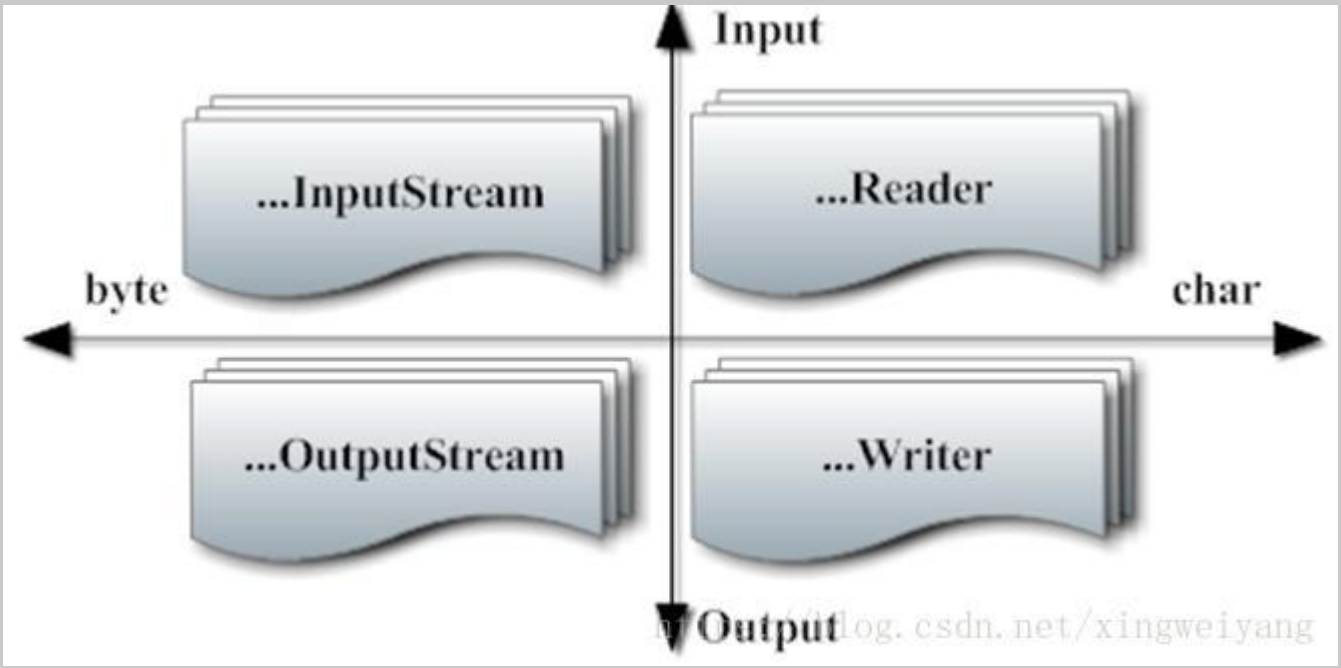
- Para bytes
- Para caracteres

FLUJOS DE BYTES Y DE CARACTERES

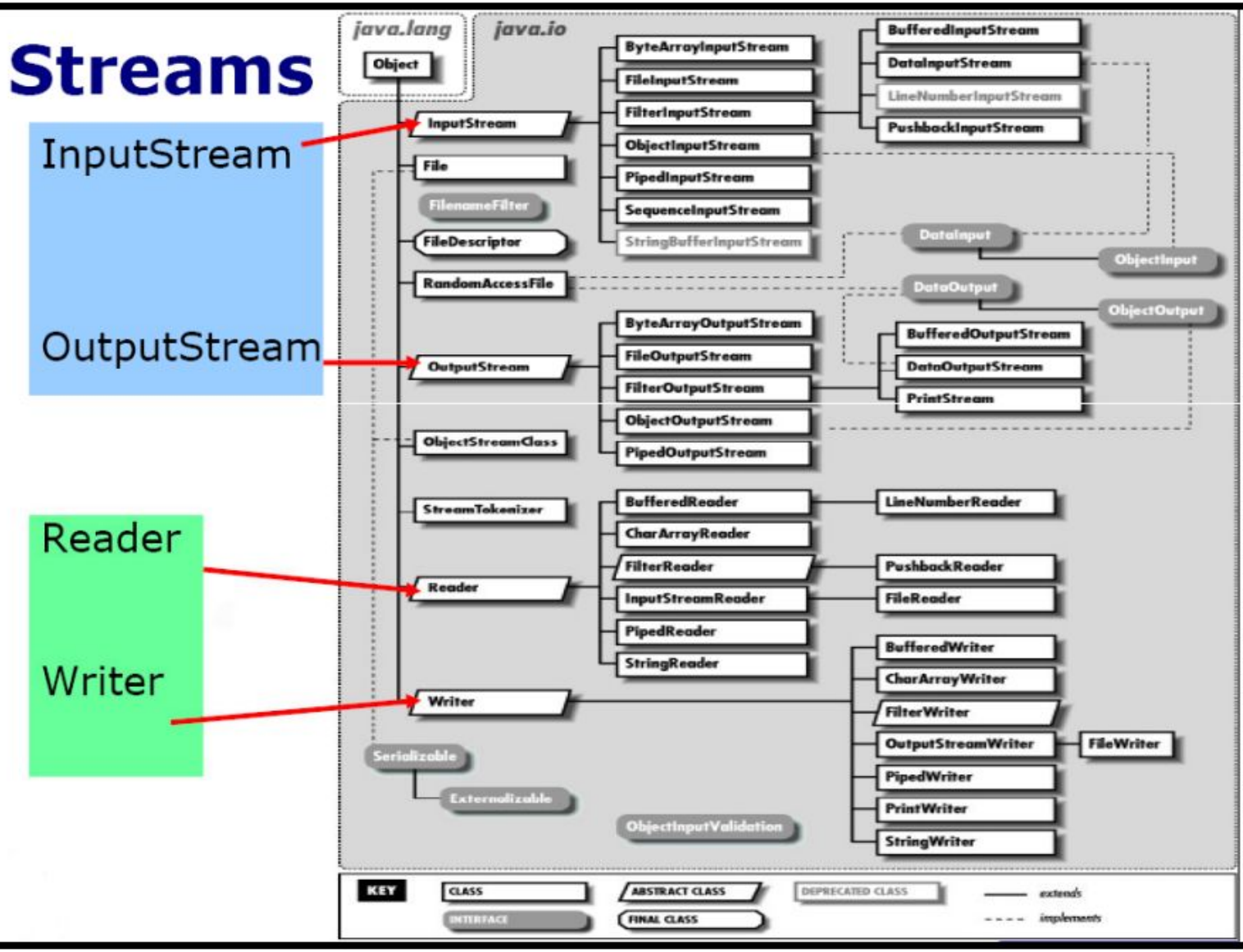
En Java hay 2 formas de recibir o enviar datos:

- **Flujo de caracteres** : permiten procesar la E/S de caracteres. Usan UNICODE. Ficheros de texto.

- **Flujo bytes:** permiten procesar la E/S de bytes. Uso por ejemplo para leer y escribir datos binarios, Útiles para trabajar con archivos, enviar paquetes de datos a través de una red,..


Input
...InputStream
...Reader
byte
char
...OutputStream
...Writer
Output
http://blog.csdn.net/xingweiyang

Jerarquía de la clase Stream



Buffered Streams

- Un unbuffered I/O significa que cada solicitud de lectura o escritura es gestionado directamente por el sistema operativo subyacente (ineficiente)
- Para reducir esta sobrecarga, Java implementa los buffered I/O streams
  - Con los buffered input streams se leen datos desde un area de memoria conocida como buffer; la API nativa se invoca sólo cuando el buffer está vacío
  - Para los buffered output streams la API se invoca cuando el buffre está lleno

Creación de Buffered Streams

- Un programa puede convertir un unbuffered stream en un buffered stream usando envolventes. Ejemplo:

```
respuesta = respuesta+"\n";
OutputStream os = p.getOutputStream();

/* ALTERNATIVA 1 - Trabajamos con bytes */
//os.write(respuesta.getBytes());
//os.flush(); // vacía el buffer de salida. Usar uno entre flush y close
//os.close();

/* ALTERNATIVA 2 - Trabajamos con bytes en buffer*/
/*BufferedOutputStream bs = new BufferedOutputStream(os);
bs.write(respuesta.getBytes());
bs.flush(); // vacía el buffer de salida. Usar uno entre flush y close
//bs.close();
*/
/* ALTERNATIVA 3 - Trabajamos con cadenas  en buffer*/
OutputStreamWriter ow = new OutputStreamWriter(os);
BufferedWriter bw = new BufferedWriter (ow);
bw.write(respuesta);
bw.flush();
```

Estándar Streams en Java

- Tres estándar streams
  - Estándar Input, accedido a través de *System.in*
  - Estándar Output, accedido a través de *System.out*
  - Estándar Error, accedido a través de *System.err*
- Estos objetos son definidos automáticamente y no requieren ser abiertos
- System.out y System.err son definidos como objetos PrintStream

Data Streams

- Data Streams soportan I/O binaria de valores de tipos de datos primitivos (boolean, char, byte, short, int, long, float, y double) así como valores String
- Todos los data streams implementan las interfaces *DataInput* o *DataOutput*
- Las implementaciones más utilizadas de esas interfaces son *DataInputStream* y *DataOutputStream*

CLASES DE FLUJOS DE BYTES

- Se definen con dos jerarquías de clases:
  - **InputStream** (características flujos entrada)
  - **OutputStream** (comportamiento flujos de salida)

A partir de estas dos clases se crean otras subclases concretas de distinta funcionalidad y que se encargan de los detalles de leer y escribir en distintos dispositivos (ejem. Archivos en disco)

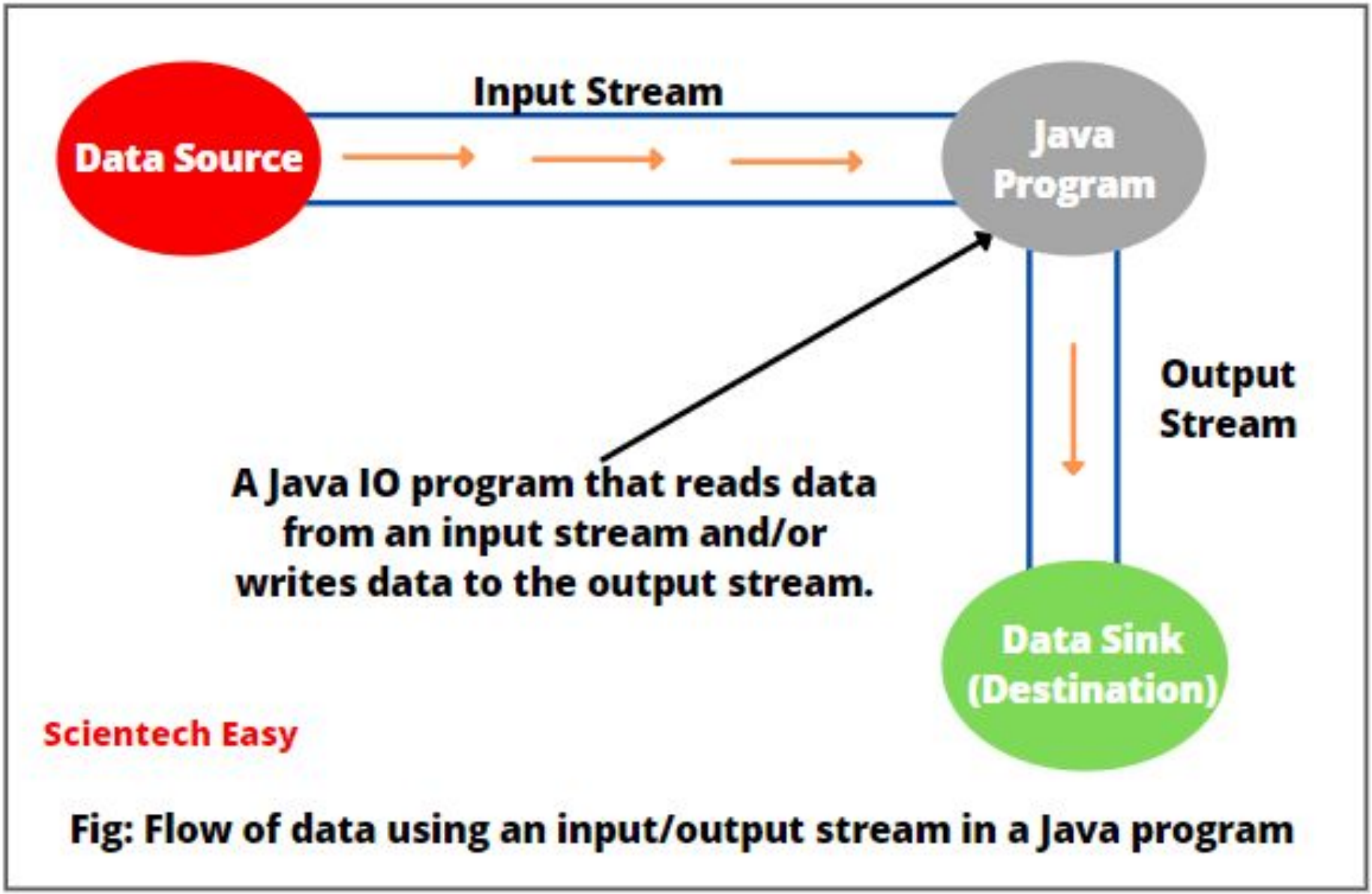
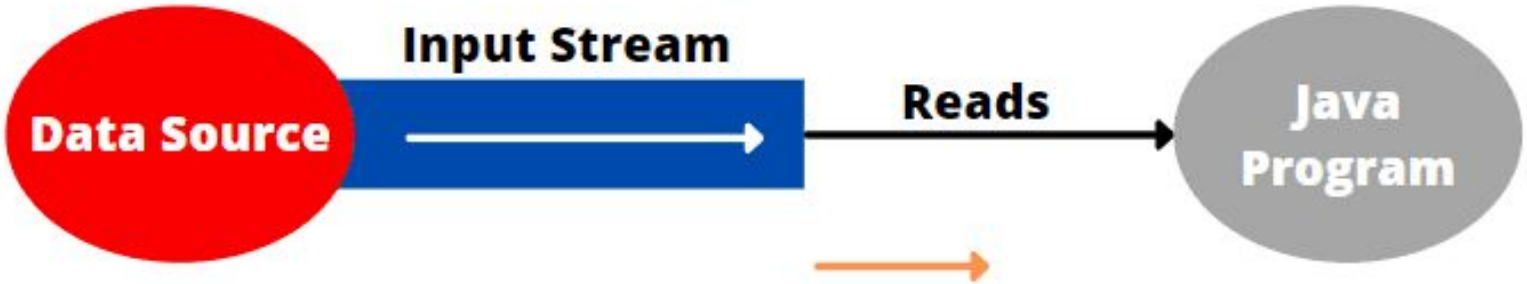
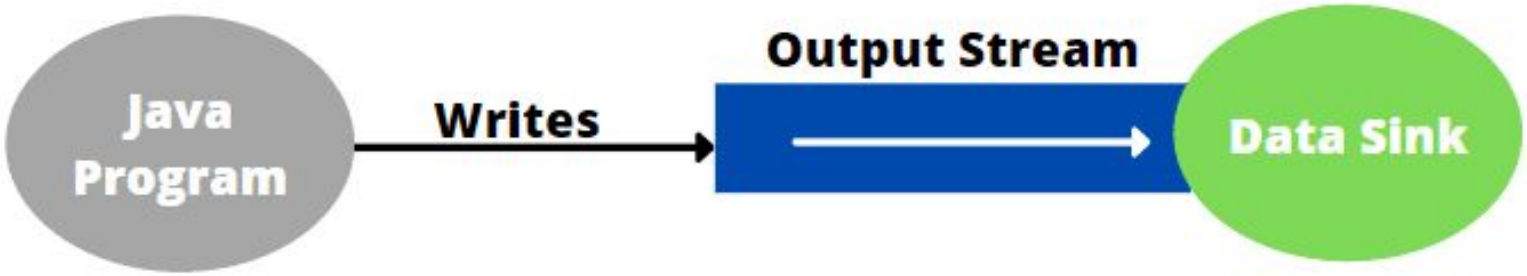


Fig: Flow of data using an input/output stream in a Java program



(a) Reading data from data source into a Java program



(b) Writing data to a data sink (destination)

Scientech Easy

Fig: Flow of data using input and output streams

CLASES DE FLUJOS DE BYTES

Clase de flujo de bytes	Significado
BufferedInputStream **BufferedOutputStream**	Flujo de entrada y salida en búfer
ByteArrayInputStream **ByteArrayOutputStream**	Flujo de entrada y salida en una matriz de bytes
DataInputStream **DataOutputStream**	Flujo de entrada y salida que contiene métodos para leer los tipos de datos estándar de Java
FileInputStream **FileOutputStream**	Flujo de entrada y salida que lee y escribe desde un archivo
FilterInputStream **FilterOutputStream**	Implementa InputStream y OutputStream
InputStream **OutputStream**	Clase abstracta que describe flujos de entrada y salida
ObjectdInputStream **ObjectOutputStream**	Flujo de entrada y salida de objetos
PipeddInputStream **PipedOutputStream**	Conducción de entrada y salida en búfer
PrintStream	Flujo de salida que contiene print() y println()
PushbackInputStream	Flujo de entrada que permite devolver bytes de flujo
SecuenceInputStream	Flujo de entrada que combina dos o más flujos de entrada que se leen secuencialmente, uno tras otro

CLASES DE FLUJOS DE BYTES

Clases InputStream y OutputStream: Clases abstractas que describe flujos de entrada y salida.

MÉTODOS de la clase InputStream	
Método	Descripción
int available()	Devuelve el número de bytes de entrada disponibles para lectura.
void close()	Cierra el origen de entrada. Los posteriores intentos de lectura generan IOException.
void mark(int numBytes)	Añade una marca al punto actual de flujo de entrada que se valida hasta que se lean numBytes.
boolean markSupported()	Devuelve true si el flujo de invocación admite mark()/reset().
int read()	Devuelve una representación entera del siguiente byte de entrada disponible. Devuelve -1 al llegar al final del flujo.
int read (byte buffer[])	Intenta leer hasta los bytes de buffer.length en búfer y devuelve el número real de bytes leídos correctamente. Se devuelve -1 al llegar al final del flujo.
int read(byte buffer[], int desplazamiento, int numBytes)	Intenta leer hasta los bytes de numBytes en búfer comenzando en búfer[desplazamiento] y devuelve el número de bytes leidos correctamente. Devuelve -1 al llegar al final del flujo.
void reset()	Restablece el puntero de entrada a la marca definida previamente.
long skip(long numBytes)	Ignora los bytes de entrada indicados por numBytes y devuelve el número de bytes ignorados

CLASES DE FLUJOS DE BYTES

Clases InputStream y OutputStream: Clases abstractas que describe flujos de entrada y salida.

MÉTODOS de la clase OutputStream	
Método	**Descripción**
void flush()	Envía a su destino la salida que se haya guardado en búfer. Es decir, vacía el búfer de salida.
void close()	Cierra el origen de salida. Los posteriores intentos de escritura generan IOException.
void write(int b)	Escribe un solo byte en un flujo de salida. El parámetro es int, lo que le permite invocar write() con expresiones sin tener que convertirlas de nuevo a byte.
void write(byte buffer [])	Escribe una matriz completa de bytes en un flujo de salida.
void write(byte buffer[], int desplazamiento, int numBytes)	Escribe un subintervalo de numBytes bytes desde el búfer de matriz, empezando por búfer[desplazamiento]

Control del flujo de una operación I/O

Crear un objeto stream y asociarlo con la fuente de datos

Dar al objeto stream la funcionalidad deseada a través del encadenamiento del stream

while (hay más información)

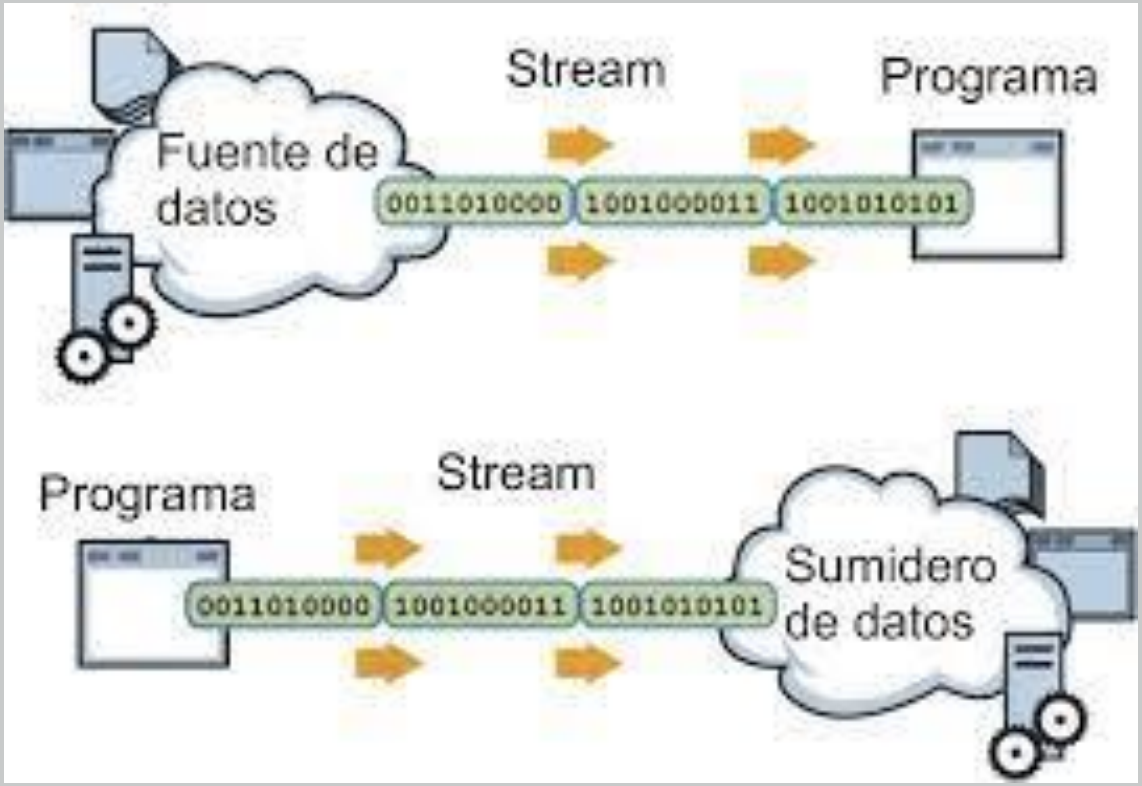
    leer (escribir) siguiente dato desde (a) el stream

cerrar el stream

FLUJOS DE BYTES Y DE CARACTERES

CLASES DE FLUJOS DE CARACTERES

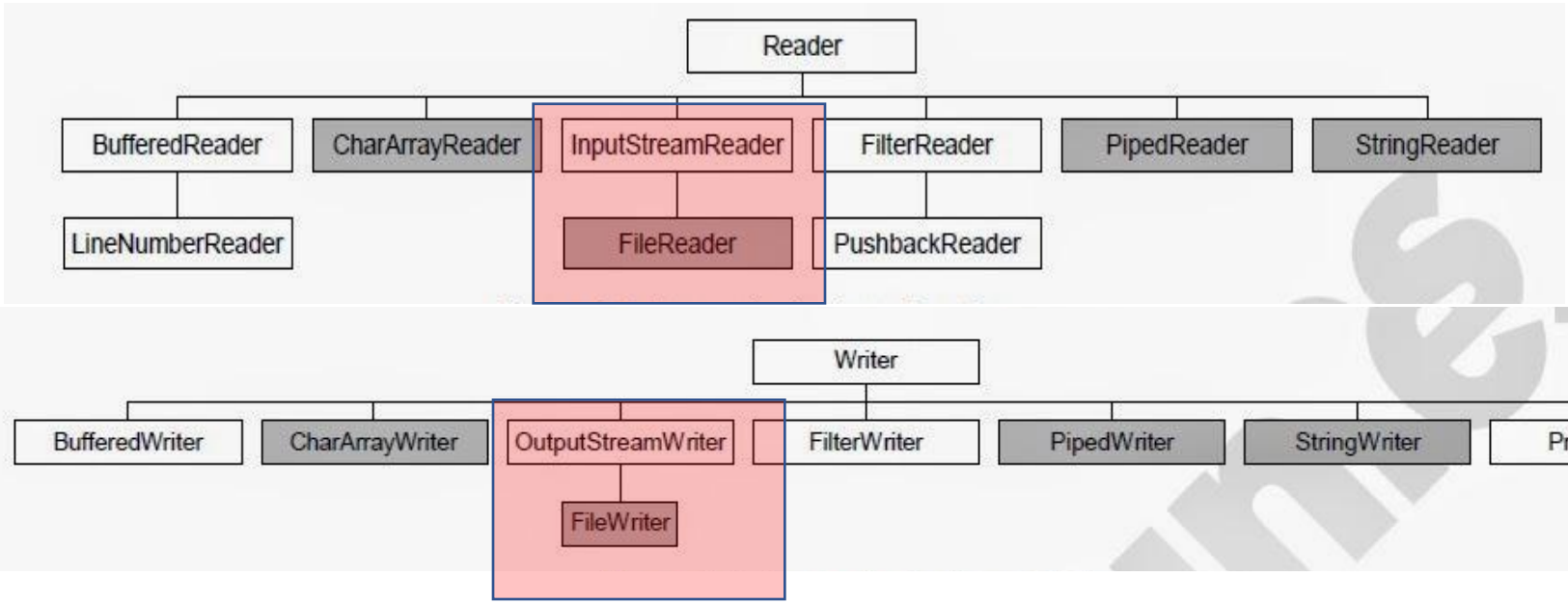
- Se definen con dos jerarquías de clases:
  - **Reader** (características flujos entrada)
  - **Writer** (comportamiento flujos de salida)



Ficheros:Acceso con Flujo de Caracteres

CLASES DE FLUJOS DE CARACTERES

- Se definen con dos jerarquías de clases:
  - **Reader** (características flujos entrada)
  - **Writer** (comportamiento flujos de salida)



FICHEROS DE FLUJO DE CARACTERES

Ventaja: operan directamente con caracteres Unicode. Por tanto, para almacenar un texto Unicode es la mejor opción.

Por lo general, para realizar E/S de archivos basada en caracteres , se usan las clases:

- FileReader
- FileWriter

F.F. CARACTERES – Lectura de Ficheros

Abrimos el fichero de lectura y leeremos la información de forma secuencial.

CONSTRUCTORES DE LA CLASE FileReader:

FileReader nombre = new FileReader(“fichero.extension”);
(Controlar la excepción FileNotFoundException)

MÉTODOS

int read( ): devuelve en ASCII el byte leído. Nos devuelve **-1** si no tiene más información el fichero. En algún caso puede que falle el -1 y debemos preguntar por 65535.

int skip(long): salta n caracteres.

String readLine( ): devuelve toda una línea del fichero.

close() : cierra el stream

F.F. CARACTERES - FILEREADER

Ejemplo:

Leer el fichero de texto pruebaTexto.txt que has creado previamente en el Escritorio.

Lee el texto hasta que alcance final de fichero.

F.F. CARACTERES - FILEWRITER

Abrimos el fichero de escritura, y guardaremos la información carácter a carácter.

CONSTRUCTORES :
FileWriter nombre = new FileWriter(“fichero.extension”);
FileWriter nombre = new FileWriter(“fichero.extension”, boolean a);
a= true añade datos al fichero que ya existe
a= false sobreescribe el fichero

MÉTODOS
void write(caracter): escribe el carácter en el fichero.

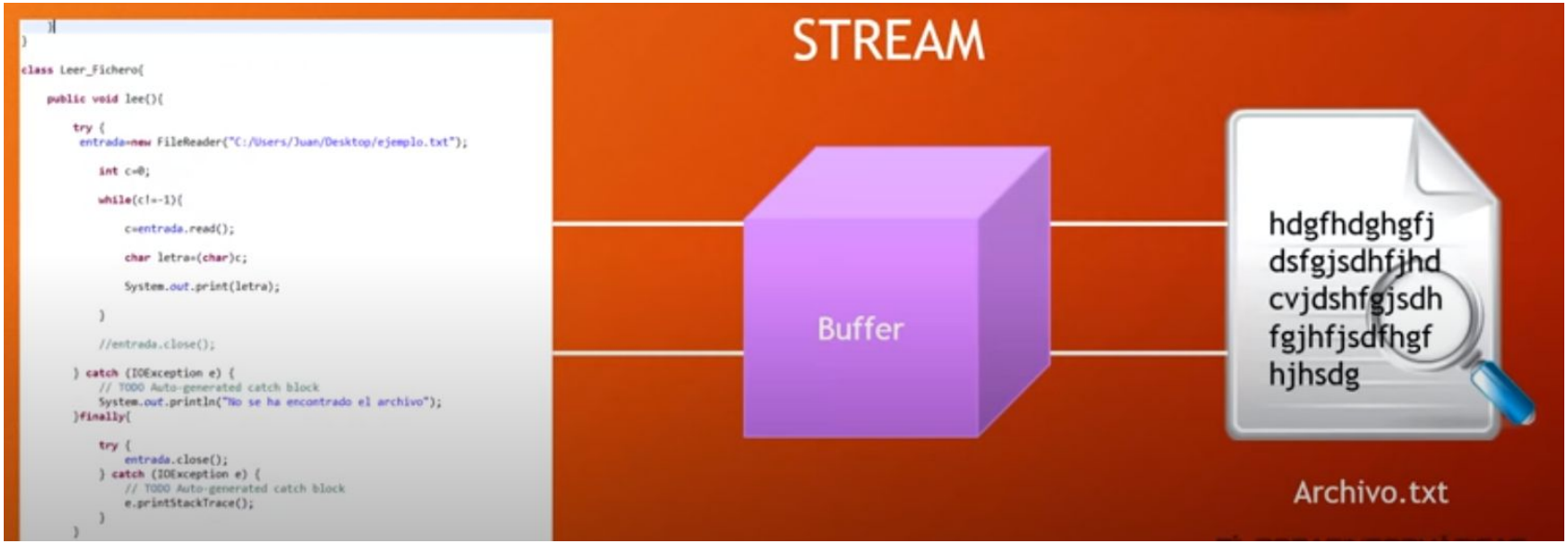
PARA CERRAR EL CANAL
void close( ), cierra el fichero.

F.F. CARACTERES - FILEREADER

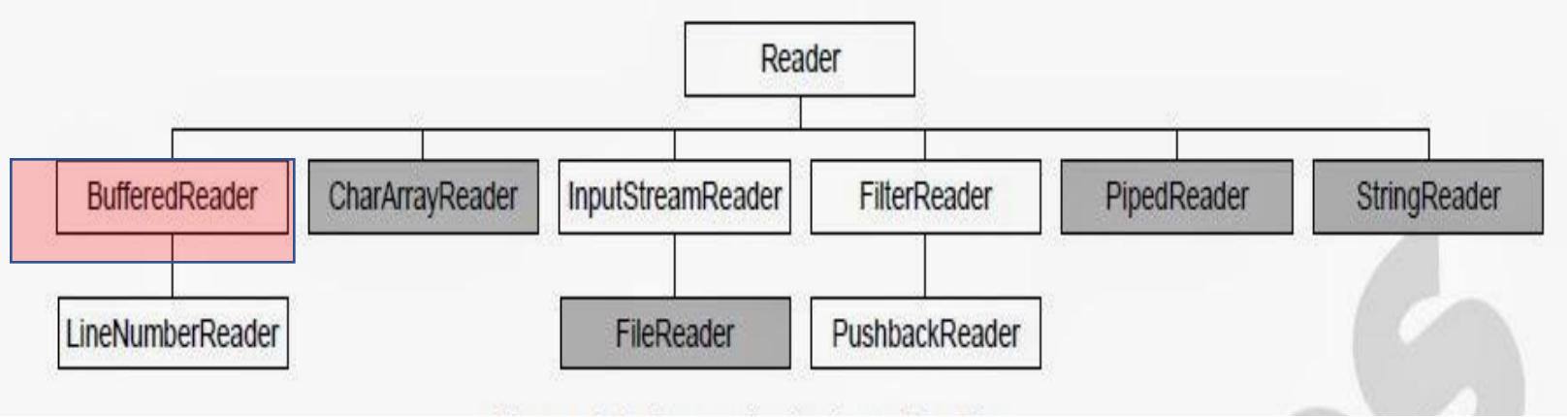
Ejemplo:

Sobreescribir y añadir un texto capturado por teclado, en un fichero de texto pruebaTexto.txt y almacénalo en el Escritorio.

F.F. CARACTERES – FILEREADER
Con Buffer



F.F. CARACTERES – FILEREADER
Con Buffer



Método: readLine();

F.F. CARACTERES – FILEREADER
Con Buffer

Si cada registro está almacenado en una fila del fichero podemos leer todos los datos de una línea y no carácter a carácter.

Debemos utilizar **BufferedReader**

Recordar uso de BufferedReader

```
import java.io.*;

BufferedReader <nombre_objeto> =new
BufferedReader(objeto clase FileReader);

<variable>=<nombre_objeto>.readLine();
```

Preguntaremos por distinto de null, en la lectura del fichero.

F.F. CARACTERES - BufferedReader

Abrimos el fichero, y guardaremos la información carácter a carácter.

CONSTRUCTORES :

BufferedReader(objeto Reader);
BufferedReader(objeto Reader, int tamaño);

MÉTODOS

Int read( ): leer un carácter del fichero.
String readLine(): leer una línea de texto.

PARA CERRAR EL CANAL

void close( ), cierra el fichero.

F.F. CARACTERES - FILEREADER

Ejemplo:

Haz un programa que lea un archivo de texto utilizando buffer de entrada.

(*) Ten en cuenta:

- readLine() te devuelve el texto que encuentra hasta un \n o \r.
- Cuando acaba el fichero retorna un null;

CLASES DE FLUJOS DE CARACTERES FILEWRITER