



Curso de Java Standard



Ing. Octavio Robleto



octavio.robleto@gmail.com



<https://octaviorobleto.com>



Introducción

Java nos ofrece un paquete completo para gestionar la comunicación (Lectura y Escritura) de bytes.

El paquete IO Input (Entrada) y Output (Salida) nos permite a través de Streams (Flujos) realizar la comunicación con la consola, archivos, etc.

Podemos dividir en dos secciones la comunicación: Orientados a Carácter y Orientados a Bytes.



File

Antes de comenzar con la lectura y escritura de archivos vamos a ver la clase File que se encuentra también en el paquete Java.io que nos proporciona una serie de métodos para obtener información de los directorios y archivos que nos van a ser de mucha utilidad.



Tipo	Método	Descripción
boolean	canRead()	Comprueba si la aplicación puede leer el archivo indicado por esta ruta.
boolean	canWrite()	Comprueba si la aplicación puede modificar el archivo indicado por esta ruta.
boolean	createNewFile()	Crea de forma atómica un archivo nuevo y vacío si y solo si aún no existe un archivo con este nombre.
boolean	delete()	Elimina el archivo o directorio indicado por esta ruta.
boolean	exists()	Comprueba si existe el archivo o directorio indicado por esta ruta.
String	getName()	Devuelve el nombre del archivo o directorio indicado por esta ruta.
String	getAbsolutePath()	Devuelve la cadena de nombre de ruta absoluta de esta ruta.
boolean	isDirectory()	Comprueba si el archivo indicado por esta ruta es un directorio.
boolean	isFile()	Comprueba si el archivo indicado por esta ruta es un archivo normal.
boolean	isHidden()	Comprueba si el archivo nombrado por esta ruta es un archivo oculto.
String[]	list()	Devuelve una matriz de cadenas que nombran los archivos y directorios en el directorio indicado por esta ruta.
File[]	listFiles()	Devuelve una matriz de nombres de ruta abstractos que denotan los archivos en el directorio indicado por esta ruta.
boolean	mkdir()	Crea el directorio nombrado por esta ruta.

Además nos proporciona unas constantes para poder obtener el separador del sistema operativo.

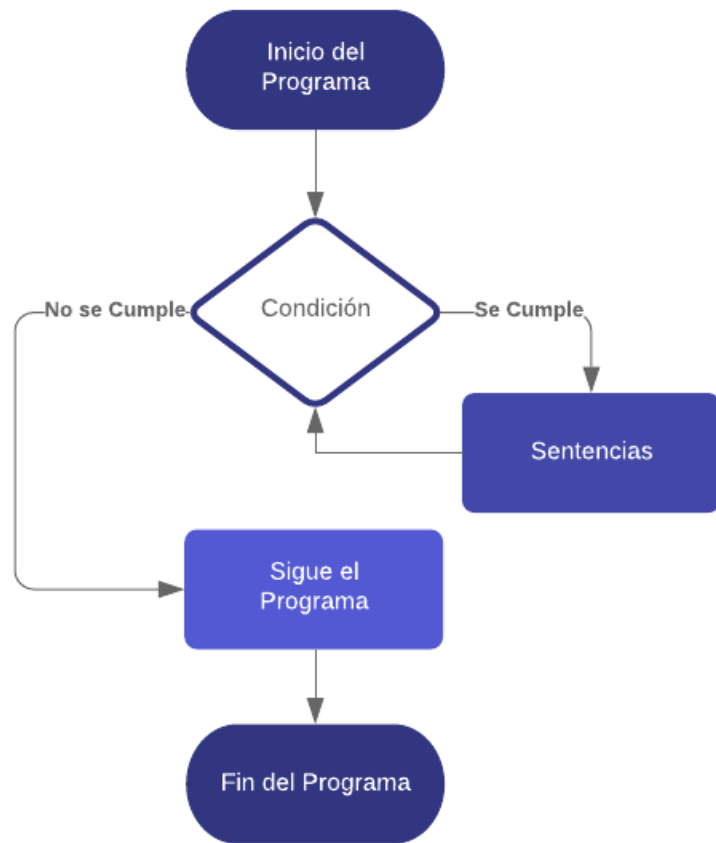
Algoritmo

El algoritmo utilizado por lo general para leer flujos de datos es Instanciar el objeto, recorrerlo mientras (while) exista un elemento que leer y por ultimo cerrar el flujo.

Los flujos de lectura nos proporcionan el método **read** que retorna el byte o carácter a leer.

El algoritmo utilizado por lo general para escribir flujos de datos es Instanciar el objeto, recorrerlo para (for) cada elemento que exista escribir y por ultimo cerrar el flujo.

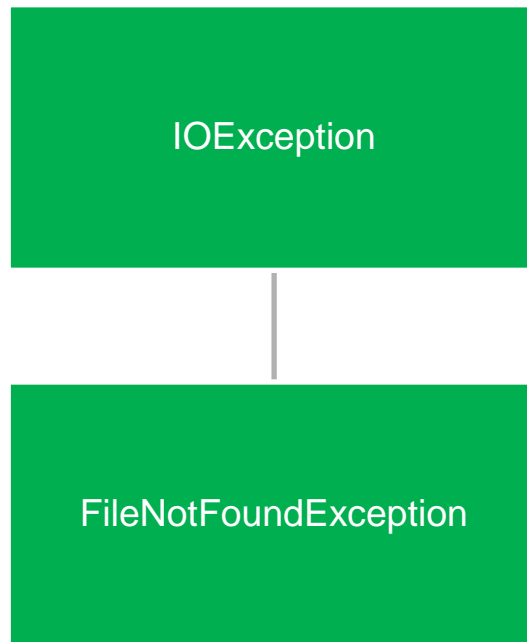
Los flujos de escritura nos proporcionan el método **write** que escribe el dato u objeto a escribir.



Excepciones

La excepción **IOException** se encargara de lanzar errores cuando exista un problema al tratar de leer un fichero, bien sea porque se encuentra bloqueado o porque se interrumpió la comunicación.

La excepción **FileNotFoundException** se encargara de lanzar un error si no encuentra el fichero especificado en la ruta especificada, se encuentra generalmente en los constructores.

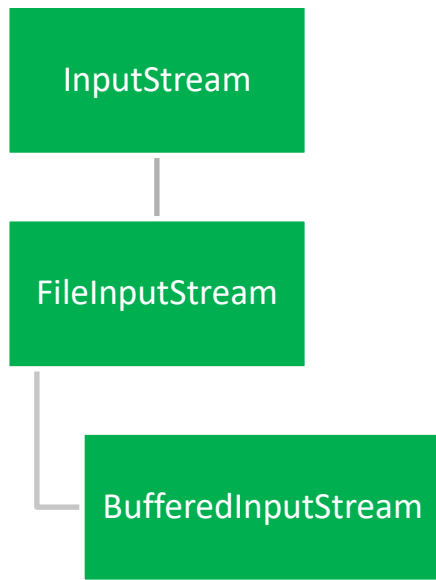


InputStream

La clase abstracta **InputStream** representa un flujo de bytes nos proporcionan los métodos para acceder a la información.

La clase **FileInputStream** reescribe todos los métodos de la clase **InputStream** proporcionando una implementación mas específica y adecuada.

Esta clase lee la información en byte lo que no es legible para el ser humano y se utiliza en la mayoría de los casos para leer imágenes, audios, videos, etc.



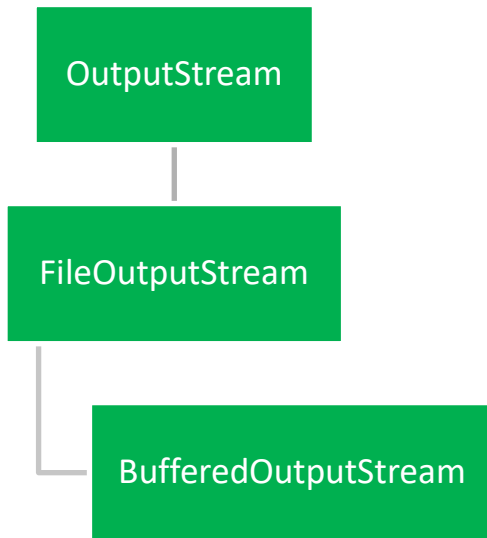
Tipo	Método	Descripción
int	available()	Retorna el numero estimado de bytes del fichero (Tamaño del fichero)
void	close()	Cierra este flujo de entrada y libera los recursos del sistema asociados con el flujo.
int	read()	Lee el siguiente byte de datos del flujo de entrada.
byte[]	readAllBytes()	Lee todos los bytes restantes del flujo de entrada.

OutputStream

La clase abstracta **OutputStream** representa un flujo de bytes nos proporcionan los métodos para acceder a la información.

La clase **FileOutputStream** reescribe todos los métodos de la clase OutputStream proporcionando una implementación mas específica y adecuada.

Esta clase escribe la información en byte y se utiliza en la mayoría de los casos para leer imágenes, audios, videos, etc.



Tipo	Método	Descripción
int	available()	Retorna el numero estimado de bytes del fichero (Tamaño del fichero)
void	close()	Cierra este flujo de entrada y libera los recursos del sistema asociados con el flujo.
void	write(byte[] b)	Escribe el arreglo de bytes en el flujo de salida.
void	write(int b)	Escribe el byte especificado del flujo de salida.

InputStream y OutputStream

```
try (FileInputStream archivoBinario = new FileInputStream(fichero)) {  
    byte[] archivoBytes = archivoBinario.readAllBytes();  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```



```
try (FileOutputStream archivoBinario = new FileOutputStream(fichero)) {  
    archivoBinario.write(bytes);  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```



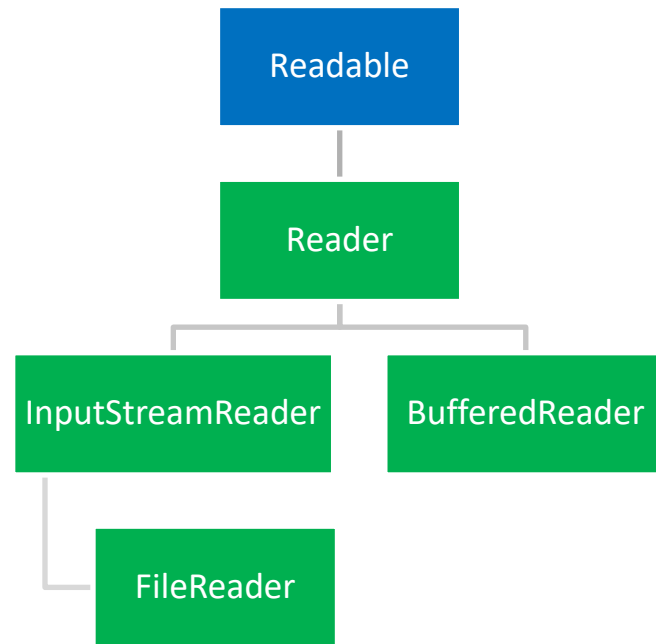
En este caso no nos interesa recorrer cada byte del archivo por lo que aprovechamos el método `readAllBytes` que retorna un arreglo con todo lo que contiene y de la misma forma escribimos el archivo.

Readable

La interfaz **Readable** nos proporciona el método de lectura .

La clase abstracta **Reader** Proporciona una implementación esquelética de la interfaz **Readable** y simplemente agrega implementaciones para los métodos **read** y **close**.

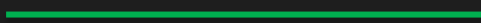
La clase **InputStreamReader** es un puente entre los flujos de bytes y los flujos de caracteres: lee bytes y los decodifica en caracteres en su representación ASCII a través de un numero entero que proporciona la clase **FileReader**.



Tipo	Método	Descripción
int	getEncoding()	Devuelve el nombre de la codificación de caracteres que utiliza esta secuencia.
void	close()	Cierra este flujo de entrada y libera los recursos del sistema asociados con el flujo.
int	read()	Lee el siguiente byte de datos del flujo de entrada.

FileReader

```
try (FileReader leerFichero = new FileReader(archivoLectura)) {  
    int c;  
  
    while ((c = leerFichero.read()) != -1) {  
        char character = (char) c;  
        System.out.print(character);  
    }  
  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```



De Azure a LinkedIn, de SQL Server a Minecraft... Microsoft «recurre a Java más de lo que la gente pueda imaginar», a la hora de desarrollar varios de sus productos y servicios más populares. Los propios servidores internos de la compañía mantienen en funcionamiento medio millón de máquinas virtuales Java.

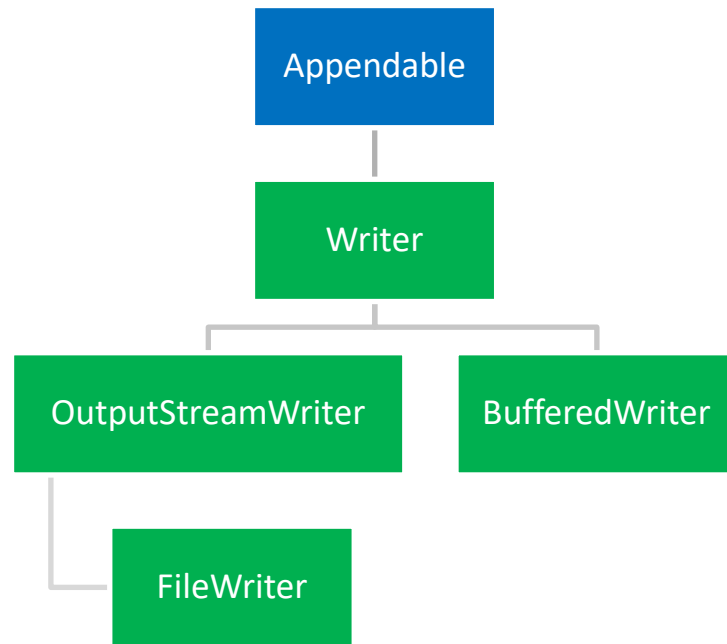
Pero ahora, además de usarlo va a aportar su granito de arena al desarrollo del lenguaje lanzando su propia compilación de OpenJDK, que incorpora todas aquellas «correcciones y mejoras que consideramos importantes para nuestros clientes y nuestros usuarios internos».

Readable

La interfaz **Appendable** nos proporciona el método de escritura.

La clase abstracta **Writer** Proporciona una implementación esquelética de la interfaz **Appendable** y simplemente agrega implementaciones para los métodos **write** y **close**.

La clase **OutputStreamWriter** es un puente entre los flujos de bytes, para escribir los datos como caracteres nos proporcionan la clase **FileWriter**.



Tipo	Método	Descripción
void	close()	Cierra este flujo de entrada y libera los recursos del sistema asociados con el flujo.
void	write(int b)	Escribe el byte especificado del flujo de salida.

FileWriter

```
try (FileWriter escribirFichero = new FileWriter(archivoEscritura)) {  
    for (int i = 0; i < PARRAFO.length(); i++) {  
        escribirFichero.write(PARRAFO.charAt(i));  
    }  
}  
catch (FileNotFoundException e) {  
    e.printStackTrace();  
}  
catch (IOException e) {  
    e.printStackTrace();  
}
```

Búfer

Es un espacio de memoria, en el que se almacenan datos de manera temporal, normalmente para un único uso; su principal uso es para evitar que el programa o recurso que los requiere se quede sin datos durante una transferencia de datos.



BufferedInputStream y BufferedOutputStream

```
try (BufferedInputStream archivoBinario = new BufferedInputStream(new FileInputStream(fichero))) {  
    byte[] bytesArchivo = archivoBinario.readAllBytes();  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```



```
try (BufferedOutputStream archivoBinario = new BufferedOutputStream(new FileOutputStream(fichero))) {  
    archivoBinario.write(bytes);  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```



En este caso no nos interesa recorrer cada byte del archivo por lo que aprovechamos el método `readAllBytes` que retorna un arreglo con todo lo que contiene y de la misma forma escribimos el archivo.

BufferedReader

Lee texto de un flujo de entrada de caracteres, almacenando caracteres en búfer para proporcionar una lectura eficiente de caracteres

Tipo	Método	Descripción
void	close()	Cierra este flujo de entrada y libera los recursos del sistema asociados con el flujo.
Int	read()	Lee el siguiente byte de datos del flujo de entrada.
String	readLine()	Lee una línea de texto.
boolean	ready()	Indica si el flujo está listo para leerse.

```
try (BufferedReader leerFichero = new BufferedReader(new FileReader(archivoLectura))) {
    String mensaje = null;

    if (leerFichero.ready()) {
        while ((mensaje = leerFichero.readLine()) != null) {
            System.out.println(mensaje);
        }
    }

} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

De Azure a LinkedIn, de SQL Server a Minecraft... Microsoft «recurre a Java más de lo que la gente pueda imaginar», a la hora de desarrollar varios de sus productos y servicios más populares. Los propios servidores internos de la compañía mantienen en funcionamiento medio millón de máquinas virtuales Java.

Pero ahora, además de usarlo va a aportar su granito de arena al desarrollo del lenguaje lanzando su propia compilación de OpenJDK, que incorpora todas aquellas «correcciones y mejoras que consideramos importantes para nuestros clientes y nuestros usuarios internos».

BufferedWriter

Escribe texto de un flujo de entrada de caracteres, almacenando caracteres en búfer para proporcionar una escritura eficiente de caracteres

Tipo	Método	Descripción
void	close()	Cierra este flujo de entrada y libera los recursos del sistema asociados con el flujo.
void	write(int b)	Escribe el byte especificado del flujo de salida.
void	write(String s)	Escribe una cadena de caracteres
void	newLine()	Escribe un salto de línea.

```
try (BufferedWriter escribirFichero = new BufferedWriter(new FileWriter(archivoEscritura))) {  
  
    for (String linea : PARRAFO) {  
        escribirFichero.write(linea);  
        escribirFichero.newLine();  
    }  
  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```