Resumen ficheros

Contenido

1 Entrada/Salida 2
2 Streams 2
3 Flujos de bytes. InputStream/OutputStream 2
3.1 InputStream 2
3.2 OutputStream 3
4 Flujos de caracteres. Reader/ Writer 4
4.1 Reader 4
4.2 Writer 5
5 Subclases de E/S 5
5.1 InputStreamReader/OutputStreamWriter 5
5.2 DataInputStream/DataOutputStream 5
5.3 ObjectInputStream/ObjectOutputStream 6
5.4 BufferedInputStream / BufferedOutputStream BufferedReader/ BufferedWriter 6
5.5 PrintWriter 7
5.6 PipedInputStream/PipedOutputStream 7
6 Combinación de flujos 7
7 E/S estándar 7
8 BufferedReader 8
9 La clase Scanner 9
10 Salida por pantalla 9

1 Entrada/Salida

Entrada: posibilidad de **introducir datos** hacia un programa.

Salida: capacidad de un programa de **mostrar información al usuario**.

2 Streams

Los **Streams** (flujos) de Java son corrientes de datos **bytes** o **caracteres** para tratar la entrada/salida de un programa.

Java proporciona varias clases diferentes de streams de entrada y salida en el paquete `java.io`.

Los problemas de entrada/salida suelen causar excepciones de tipo `IOException` o de sus derivadas. Con lo que la mayoría de operaciones deben ir inmersas en un bloque `try-catch` o añadir la clausula `throws IOException` correspondiente en la definición de la clase.

Si se utilizan streams que transmiten caracteres Java (tipo char Unicode, de dos bytes…), se habla entonces de un `reader` (si es de lectura) o un `writer` (escritura).

3 Flujos de bytes. InputStream/OutputStream

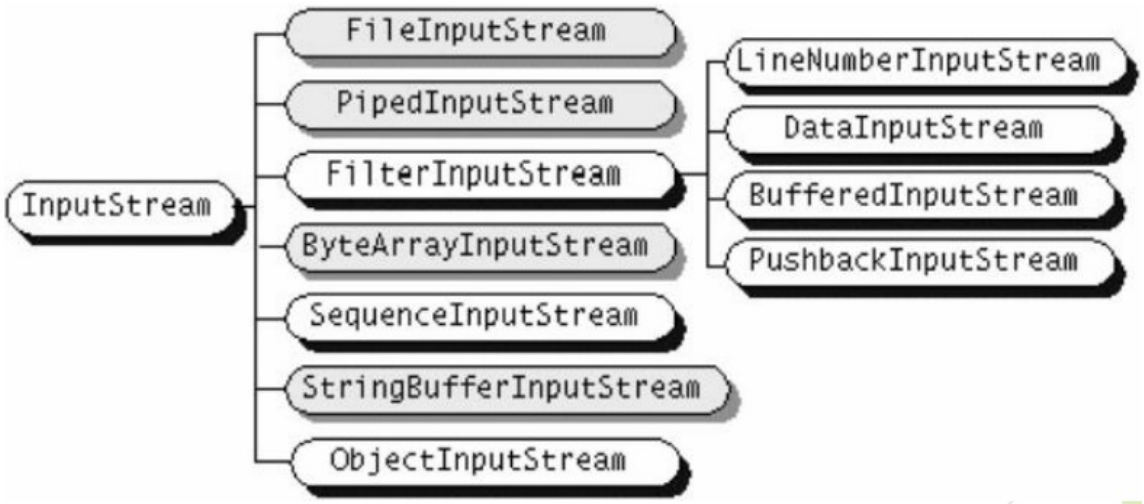
Estas dos clases abstractas, definen las funciones básicas de **lectura y escritura de una secuencia de bytes pura** (sin estructurar).

No representan ni **textos** ni **objetos**, sino datos binarios puros.

Poseen numerosas subclases, de hecho casi todas las clases preparadas para la lectura y la escritura, derivan de estas.

Los métodos más importantes son `read` (leer) y `write` (escribir), que sirven para leer un byte del dispositivo de entrada o escribir un byte respectivamente.

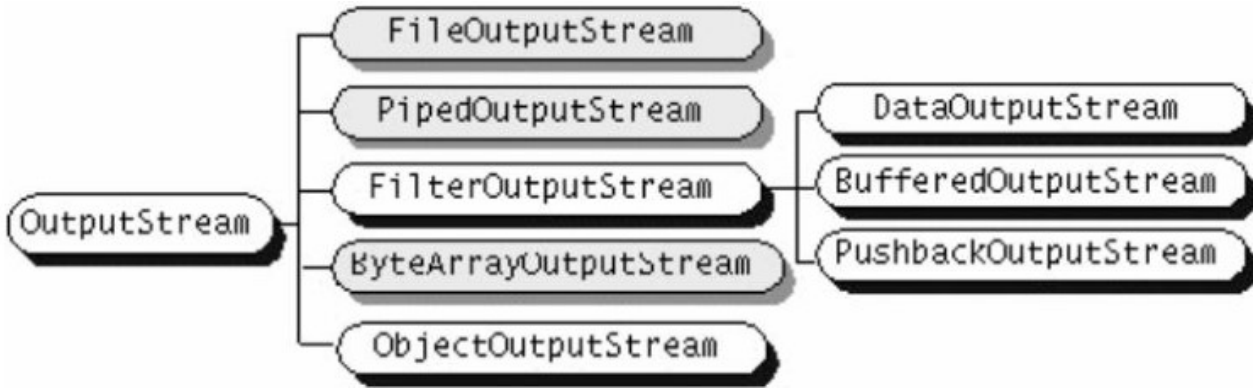
3.1 InputStream



Métodos de InputStream

Método	Uso
int available()	Devuelve el número de bytes de entrada.
void close()	Cierra la corriente de entrada. Cualquier acceso posterior generaría una IOException.
void mark(int bytes)	Marca la posición actual en el flujo de datos de entrada. Cuando se lea el número de bytes indicado, la marca se elimina.
boolean markSupported()	Devuelve verdadero si en la corriente de entrada es posible marcar mediante el método mark.
int read()	Lee el siguiente byte de la corriente de entrada y lo almacena en formato de entero. Devuelve -1 si estamos al final del fichero
int read(byte[] buffer)	Lee de la corriente de entrada hasta llenar el array buffer.
void reset()	Coloca el puntero de lectura en la posición marcada con mark.
long skip()	Se salta de la lectura el número de bytes indicados

3.2 OutputStream



Métodos de OutputStream

Método	Uso
void close()	Cierra la corriente de salida. Cualquier acceso posterior generaría una IOException.
void flush()	Vacía los buffers de salida de la corriente de datos.
void write(int byte)	Escribe un byte en la corriente de salida.
void write(byte[] bufer)	Escribe todo el array de bytes en la corriente de Salida.
void write(byte[] buffer, int posinicial, int numbytes)	Escribe el array de bytes en la salida, pero empezando por la posición inicial y sólo la cantidad indicada por numBytes.

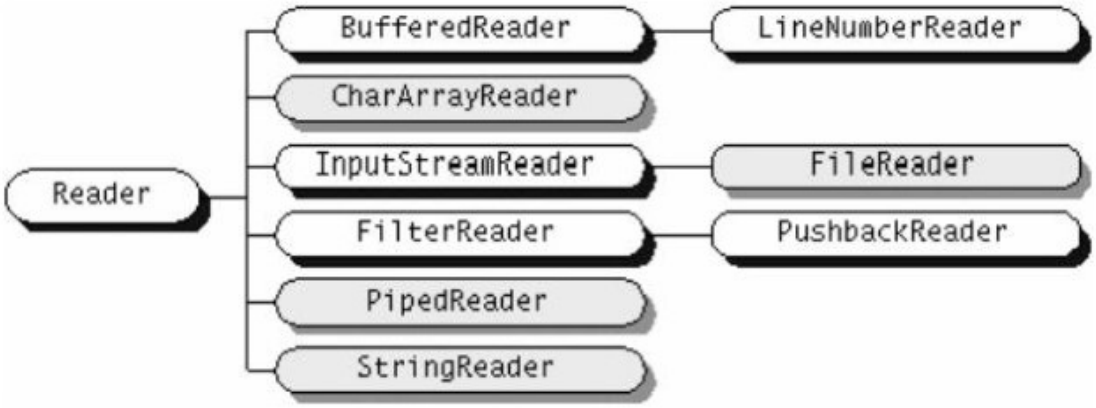
`EjemploCopiaFicheros`

4 Flujos de caracteres. Reader/ Writer

Son clases abstractas que definen las funciones básicas de escritura y lectura basada en texto Unicode.

Se dice que estas clases pertenecen a la jerarquía de lectura/escritura orientada a caracteres.

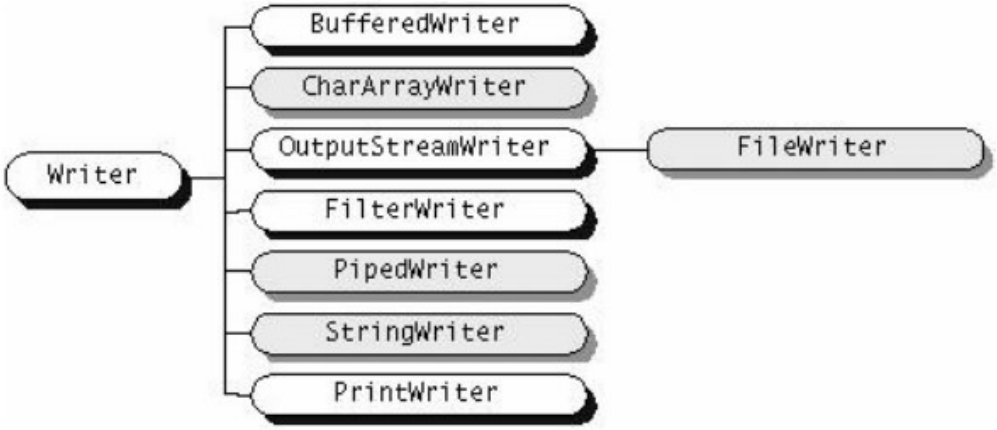
4.1 Reader



Métodos de Reader

Método	Uso
void close()	Cierra la corriente de entrada. Cualquier acceso posterior generaría una IOException.
void mark(int bytes)	Marca la posición actual en el flujo de datos de entrada. Cuando se lea el número de bytes indicado, la marca se elimina.
boolean markSupported()	Devuelve verdadero si en el flujo de entrada es posible marcar mediante el método mark.
int read()	Lee el siguiente byte del flujo de entrada y lo almacena en formato de entero. Devuelve -1 si estamos al final del fichero
int read(byte[] buffer)	Lee bytes del flujo de entrada y los almacena en el buffer. Lee hasta llenar el buffer.
int read (byte[] buffer, int posinicio, int despl)	Lee bytes del flujo de entrada y los almacena en el buffer. La lectura la almacena en el array pero a partir de la posición indicada, el número máximo de bytes leídos es el tercer parámetro.
boolean ready()	Devuelve verdadero si del flujo de entrada está listo.
void reset()	Coloca el puntero de lectura en la posición marcada con mark.
void skip()	Se salta de la lectura el número de bytes indicados

4.2 Writer



Métodos de Writer

Método	Uso
void close()	Cierra el flujo de salida. Cualquier acceso posterior generaría una IOException.
void flush()	Vacía los buffers de salida del flujo de datos.
void write(int byte)	Escribe un byte en el flujo de salida.
void write(byte[] bufer)	Escribe todo el array de bytes en el flujo de salida.
void write(byte[] buffer, int posinicial, int numbytes)	Escribe el array de bytes en la salida, pero empezando por la posición inicial y sólo la cantidad indicada por numBytes.
void write(String texto)	Escribe los caracteres en el String en el flujo de salida.
void write(String buffer, int posInicial, int numBytes)	Escribe el String en la salida, pero empezando por la posición inicial y sólo la cantidad indicada por numBytes.

5 Subclases de E/S

5.1 InputStreamReader/OutputStreamWriter

Son clases que sirven para adaptar la entrada y la salida. La razón es que los flujos básicos de E/S son de tipo `Stream` y estas clases consiguen adaptarlos a flujos `Reader/Writer`.

Puesto que derivan de las clases `Reader` y `Writer`, ofrecen los mismos métodos que éstas.

Poseen un constructor que permite crear objetos `InputStreamReader` pasando como parámetro un flujo de tipo `InputStream` y objetos `OutputStreamWriter` partiendo de objetos `OutputStream`.

5.2 DataInputStream/DataOutputStream

Leen flujos de datos de entrada en forma de byte, pero adaptándola a los tipos simples de datos (`int`, `short`, `byte`,..., `String`).

Ambas clases construyen objetos a partir de flujos `InputStream` y `OutputStream` respectivamente.

En cuanto a los métodos de `OutputStreamWriter` la idea es la misma que la de la tabla posterior, los métodos son: **`writeBoolean`**, **`writeByte`**, **`writeBytes`**(para Strings), **`writeFloat`**, **`writeShort`**, **`writeUTF`**, **`writeInt`**, **`writeLong`**. Todos poseen un argumento que son los datos a escribir (cuyo tipo debe coincidir con la función).

Métodos de DataInputStream

Método	Uso
boolean readBoolean()	Lee un valor booleano del flujo de entrada. Puede provocar excepciones de tipo IOException o excepciones de tipo EOFException. Esta última se produce cuando se ha alcanzado el final del archivo y es una excepción derivada de la anterior, por lo que si se capturan ambas, ésta debe ir en un catch anterior (de otro modo, el flujo del programa entraría siempre en la IOException).
byte readByte()	Idéntica a la anterior, pero obtiene un byte. Las excepciones que produce son las mismas.
…readChar(), readShort(), readLong(), readFloat(), readDouble()	Como las anteriores, pero devolviendo el tipo de datos adecuado.
String readLine()	Lee de la entrada caracteres hasta llegar a un salto de línea o al fin del fichero y el resultado se obtiene en forma de String.
String readUTF()	Lee un String en formato UTF (codificación norteamericana). Además de las excepciones comentadas antes, puede ocurrir una excepción del tipo UTFDataFormatException (derivada de IOException) si el formato del texto no está en UTF.

5.3 ObjectInputStream/ObjectOutputStream

Son filtros de secuencia que permiten leer y escribir objetos de un flujo de datos orientado a **`bytes`**. Sólo tienen sentido si los datos almacenados son **objetos**. Tienen los mismos métodos que `DataInputStream`, pero aportan un nuevo método de lectura:

El método `readObject` devuelve un objeto **`Object`** de los datos de la entrada. En caso de que no haya un objeto o no sea serializable, da lugar a excepciones. Las excepciones pueden ser:

- `ClassNotFoundExcepcion`
- `InvalidClassExcepcion`
- `StreamCorruptedException`
- `OptionalDataException`
- `IOException` genérica.

La clase `ObjectOutputStream` posee el método de escritura de objetos `writeObject` al que se le pasa el objeto a escribir. Este método podría dar lugar en caso de fallo a excepciones `IOException`, `NotSerializableException` o `InvalidClassException`.

5.4 BufferedInputStream / BufferedOutputStream BufferedReader/ BufferedWriter

La palabra buffered hace referencia a la capacidad de almacenamiento temporal en la lectura y escritura.

Los datos se almacenan en una **memoria temporal** antes de ser realmente leídos o escritos.

Se trata de cuatro clases que trabajan con métodos distintos pero que suelen trabajar con los mismos flujos de entrada, que podrán ser de bytes (`Input/OutputStream`) o de caracteres (`Reader/Writer`).

La clase `BufferedReader` aporta el método `readLine` que permite leer caracteres **hasta** la presencia de **`null`** o del **salto de línea**.

5.5 PrintWriter

Clase pensada para secuencias de datos orientados a la impresión de textos.

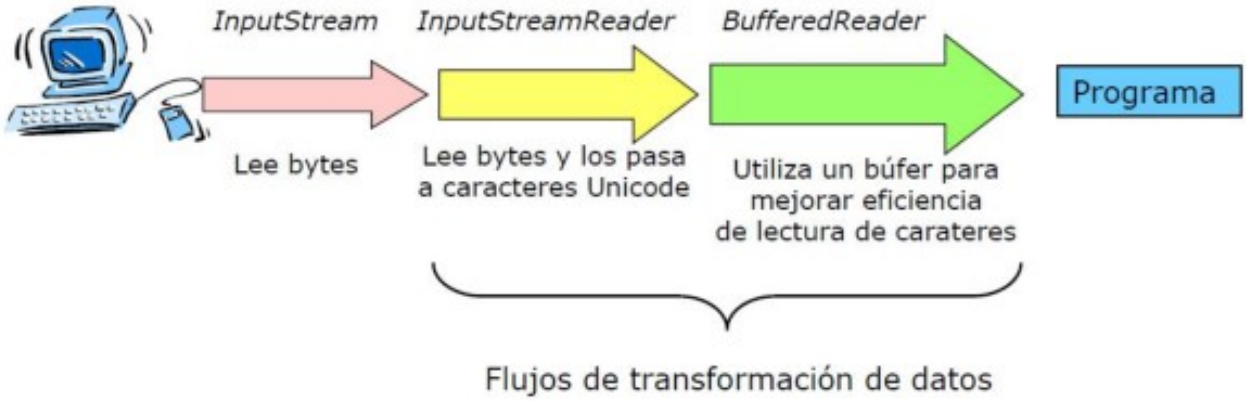
Es una clase escritora de caracteres en flujos de salida, que posee los métodos print y println, que otorgan gran potencia a la escritura.

5.6 PipedInputStream/PipedOutputStream

`PipedInputStream` tiene como origen de datos: aplicación que se ejecuta concurrentemente en otro hilo. Y `PipedOutputStream` tiene como dstino de datos: aplicación que se ejecuta concurrentemente en otro hilo.

6 Combinación de flujos

Los flujos se pueden combinar para mejorar la funcionalidad de un programa.



7 E/S estándar

La clase `java.lang.System` tiene los atributos públicos y estáticos `in, out`, y `err`, que representan respectivamente los streams estándar de entrada, salida y error.
`System.in` es del tipo `java.io.InputStream` y `System.out` y `System.err` son del tipo `java.io.PrintStream`.

- `System.in` : Objeto `InputStream`, conectado al teclado.
  - Instancia de la clase `InputStream`: flujo de bytes de entrada.
  - Metodos
    - `read()` permite **leer un byte** de la entrada como **entero** (valor entre 0 y 255)
    - `skip(n )` ignora n bytes de la entrada
    - `available()` número de bytes disponibles para leer en la entrada

- `System.out` : Objeto `OutputStream`, resultados normales en monitor.
  - Instancia de la clase `PrintStream`: flujo de bytes de salida
  - Metodos para impresión de datos
    - `print(),println()`
    - `flush()` vacía el buffer de salida escribiendo su contenido

- `System.err`: Objeto `OutputStream`, para mensajes de error – monitor.
  - Funcionamiento similar a `System.out`
  - Se utiliza para enviar mensajes de error (por ejemplo a un fichero de log o a la consola)

Ejemplo de uso de flujos estándar: `EjemploSystemIn`

El hecho de que las clases `InputStream` y `OutputStream` usen el tipo `byte` para la lectura, complica mucho su uso.

El método `read()` lee sólo un byte de la entrada estándar (del teclado), y lo normal es escribir textos largos; por lo que el método `read` no es el apropiado. El método `read` puede recibir un argumento que es un array de `bytes` que almacenará cada carácter leído y devolverá el número de caracteres leído.

Ejemplo de lectura utilizando un array de bytes: `EjemploSysInConArrayBytes`

Hay otras clases que facilitan el trabajo con textos: `InputStreamReader` y `OutputStreamWriter`.

Se utilizan para convertir secuencias de `byte` en secuencias de caracteres según una determinada configuración regional.

Permiten construir objetos de este tipo a partir de objetos `InputStream` u `OutputStream`. Puesto que son clases derivadas de `Reader` y `Writer` el problema está solucionado.

El constructor de la clase `InputStreamReader` requiere un objeto `InputStream` y, opcionalmente, una cadena que indique el código que se utilizará para mostrar caracteres (por ejemplo “ISO-8914-1” es el código Latín 1, el utilizado en la configuración regional). Sin usar este segundo parámetro se construye según la codificación actual (es lo normal).

Lo que hemos creado de esa forma es un objeto convertidor. De esa forma podemos utilizar la función `read` orientada a caracteres Unicode que permite leer caracteres extendidos.

`EjemploInputStreamReader`

8 BufferedReader

El uso del método `read` con un array de caracteres sigue siendo un poco enrevesado, además hay que tener en cuenta que el teclado es un dispositivo con buffer de lectura. -> `BufferedReader`.

La razón es que esta clase posee el método `readLine()` que permite leer una línea de texto en forma de `String`, que es más fácil de manipular.

Esta clase usa un constructor que acepta objetos `Reader` (y por lo tanto `InputStreamReader`¸ ya que desciende de ésta) y, opcionalmente, el número de caracteres a leer.

`EjemploBufferedReader`

`EjemploBufferedReader2`

`EjemploBufferedReader3`

Si queremos trabajar con otros tipos de datos en lugar de cadenas, hay que hacer conversión de tipos.

- `byte num=Byte.parseByte(cad);`
- `short num=Short.parseShort(cad);`
- `int num=Integer.parseInt(cad);`
- `long num=Long.parseLong(cad);`
- `float num=Float.parseFloat(cad);`
- `double num=Double.parseDouble(cad);`

9 La clase Scanner

Para poder usarla:

```
import java.util.Scanner;
```

Primero hay que crear un objeto:

```
Scanner teclado = new Scanner(System.in);
```

El funcionamiento de la clase `Scanner` lo podemos resumir diciendo que cuando se introducen caracteres por teclado, el objeto `Scanner` toma toda la cadena introducida y la divide en elementos llamados tokens.

El carácter predeterminado que sirve de separador de tokens es el espacio en blanco.

Cuando en un programa se leen por teclado datos numéricos y datos de tipo carácter o `String` debemos tener en cuenta que al introducir los datos y pulsar intro estamos también introduciendo en el buffer de entrada el intro.

Métodos de `Scanner`

Método	Uso
nextXxx()	Devuelve el siguiente token como un tipo básico. Xxx es el tipo. Por ejemplo, nextInt() para leer un entero, nextDouble para leer un double, etc.
next()	Devuelve el siguiente token como un String.
nextLine()	Devuelve la línea entera como un String. Elimina el final \n del buffer.
hasNext()	Devuelve un boolean. Indica si existe o no un siguiente token para leer.
hasNextXxx()	Devuelve un boolean. Indica si existe o no un siguiente token del tipo especificado en Xxx, por ejemplo hasNextDouble().
useDelimiter(String)	Establece un nuevo delimitador de token.

Problemas con nextLine(): **EjemploScanner**

Solución: **EjemploScanner2**

10 Salida por pantalla

- `System.out.print();`
- `System.out.println();`
- `System.out.printf();` // Permite aplicar formato a la salida por pantalla. Ver Anexo I