Grado Superior
Programación

Ficheros

COLEGIO litterator

Curso 2023/24

Profesor
Jonathan Carrero

Java

Conceptos generales

Definición

Antes de comenzar, mencionar que la lectura y escritura se profundizará el año que viene en la asignatura de Acceso a Datos.

Es una **secuencia de bytes** en un dispositivo de almacenamiento tales como disco duro, CD, DVD, memoria USB, etc.

Se puede **leer** y/o **escribir**.

Se identifica mediante un nombre conocido como `pathname`:

```
/home/jonathan/documents/fichero.txt
```

Definición

La lectura y escritura de ficheros es muy importante porque permite a nuestros programas dialogar e **intercambiar** información con el usuario.

La entrada/salida se realiza mediante el empleo de **flujo** de datos.

Un flujo es una **secuencia ordenada** de datos que se transmite desde una fuente hasta un destino. En Java se denomina `Stream`.

El origen o el destino pueden ser un fichero, un `String`, un dispositivo (teclado, altavoz, pantalla...), etc.

Flujo de datos

La entrada/salida se organiza generalmente mediante **objetos** llamados `Streams`.

Un `stream` es la generalización de un fichero, es decir: su secuencia es ordenada y su origen o destino pueden ser un fichero, un `String`, otro dispositivo, etc.

Para poder usar un `stream` primero hay que **abrirlo**. Esto se hace en el momento de su creación y se **cierra** cuando se deja de utilizar.

Las clases relacionadas con streams se encuentran **definidas** en el paquete `java.io`, abreviatura de Input/Output.

Clasificación de los streams

Por el **tipo** de datos que transportan:

- Binarios: de bytes.
- Caracteres: de texto.

Por el **sentido** del flujo de datos:

- Entrada: los datos fluyen desde el fichero hacia el programa.
- Salida: los datos fluyen desde el programa hacia el fichero.

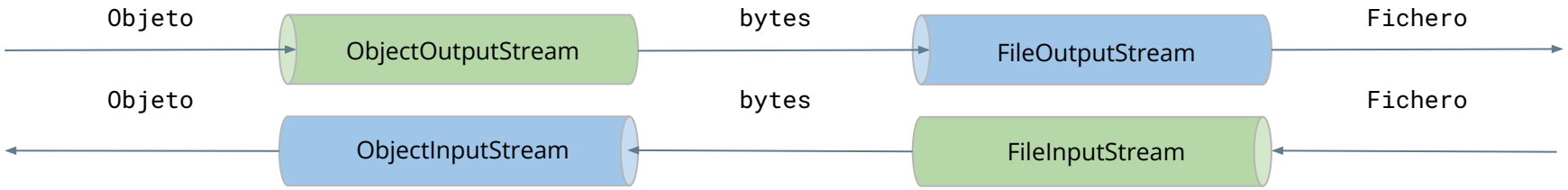
Según su **cercanía** al dispositivo:

- Iniciadores: directamente vuelcan o recogen los datos del dispositivo.
- Filtros: se sitúan entre un stream iniciador y el programa.

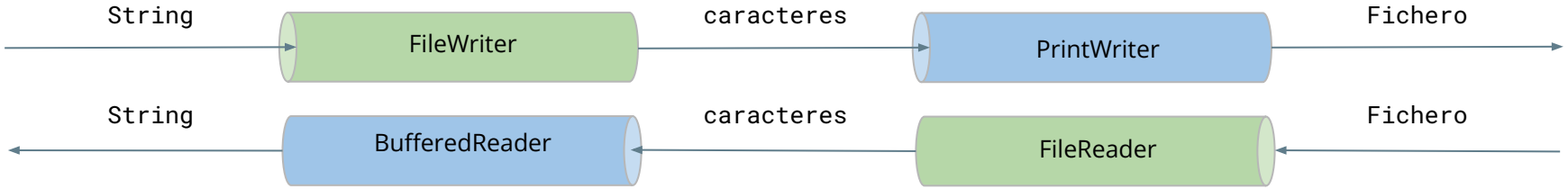
Uso de los streams

Para la leer y escribir de fichero se suele usar un `stream` iniciador y un filtro. Esto nos permite hablar “la misma lengua” que el fichero y comunicarnos con él correctamente.

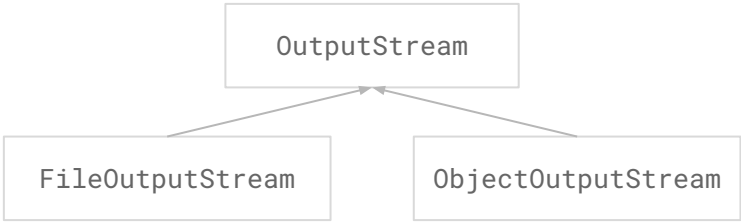
Binario



Texto

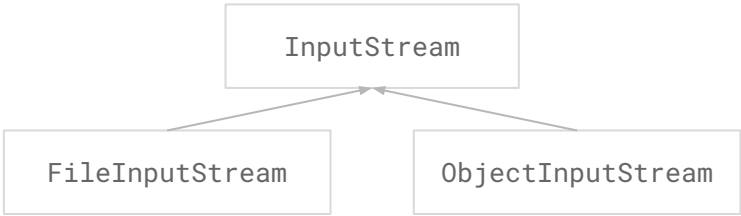


Jerarquía de clases streams binarios



`OutputStream`: escritura de ficheros binarios.

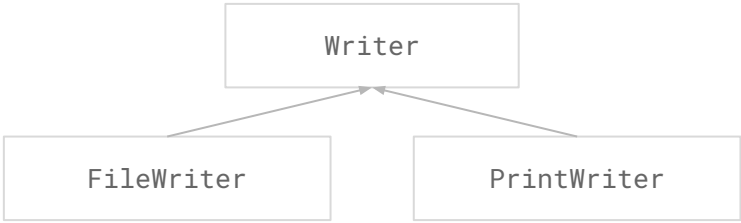
- **`FileOutputStream`**: escribe bytes en un fichero.
- **`ObjectOutputStream`**: convierte objetos y variables en arrays de bytes que pueden ser escritos en un `OutputStream`.



`InputStream`: lectura de ficheros binarios.

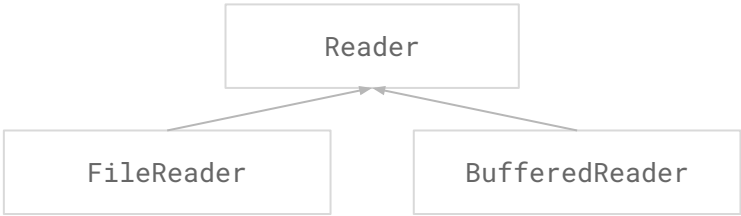
- **`FileInputstream`**: lee bytes de un fichero.
- **`ObjectInputStream`**: convierte en objetos y variables los arrays de bytes leídos de un `InputStream`.

Jerarquía de clases streams de caracteres



`Writer`: escritura de ficheros de texto.

- **`FileWriter`**: escribe texto en un fichero.
- **`PrintWriter`**: permite convertir a texto variables y objetos para escribirlos en un `Writer`.



`Reader`: lectura de ficheros de texto.

- **`FileReader`**: lee texto de un fichero.
- **`BufferedReader`**: lee texto (línea a línea) de un `Reader`.

Objetos stream predefinidos

- **`System.out`**: salida estándar por consola. Es un objeto de la clase `PrintStream` (subclase de `OutputStream`)

- **`System.err`**: salida de error por consola. También es un objeto de la clase `PrintStream`.

- **`System.in`**: entrada estándar por teclado. Es un objeto de la clase `InputStream`.

Java

Ficheros binarios

Clases para el manejo de flujo de bytes

Miscelánea de clases

`FileOutputStream`: permite escribir bytes hacia un fichero binario (ejemplo página 222).

`BufferedOutputStream`: permite escribir información a otro `OutputStream` utilizando un buffer interno que mejora el rendimiento (ejemplo página 222).

`DataOutputStream`: define algunos métodos que permiten escribir datos de tipo primitivo desde un `OutputStream` (ejemplo página 222).

`ByteArrayOutputStream`: permite usar un array de bytes como un `OutputStream` (ejemplo página 223).

Miscelánea de clases

`PrintStream`: permite escribir los datos de un stream en otro automáticamente y sin necesidad de invocar al método `flush()` (ejemplo página 223).

`FileInputStream`: permite leer bytes desde un fichero binario (ejemplo página 223).

`BufferedInputStream`: permite leer información de otro `InputStream` utilizando un buffer que mejora el rendimiento (ejemplo página 224).

`DataInputStream`: define métodos que permiten leer un tipo de datos primitivo desde un `InputStream` (ejemplo página 224).

Miscelánea de clases

`ByteArrayInputStream`: permite usar un array de bytes como `InputStream` (ejemplo página 224).

`SequenceInputStream`: flujo de entrada que puede combinar varios `InputStream` en uno (ejemplo página 225).

Java

Ficheros de texto

Clases para el manejo de flujo de caracteres

Miscelánea de clases

`FileWriter`: permite escribir caracteres hacia un fichero de texto (ejemplo página 226).

`BufferedWriter`: permite escribir información de otro `Writer` utilizando un buffer interno que mejora el rendimiento (ejemplo en página 226).

`PrintWriter`: es la implementación de la clase `Writer`. Permite escribir una cadena de caracteres en cualquier `OutputStream` (ejemplo en página 227).

`StringWriter`: flujo de caracteres que recoge su salida de un buffer y que puede utilizarse para construir un `String` (ejemplo en página 227).

Miscelánea de clases

`FileReader`: flujo de caracteres para la lectura de ficheros de texto (ejemplo página 228).

`BufferedReader`: permite leer información de otro `Reader` utilizando un buffer interno que mejora el rendimiento (ejemplo página 228).

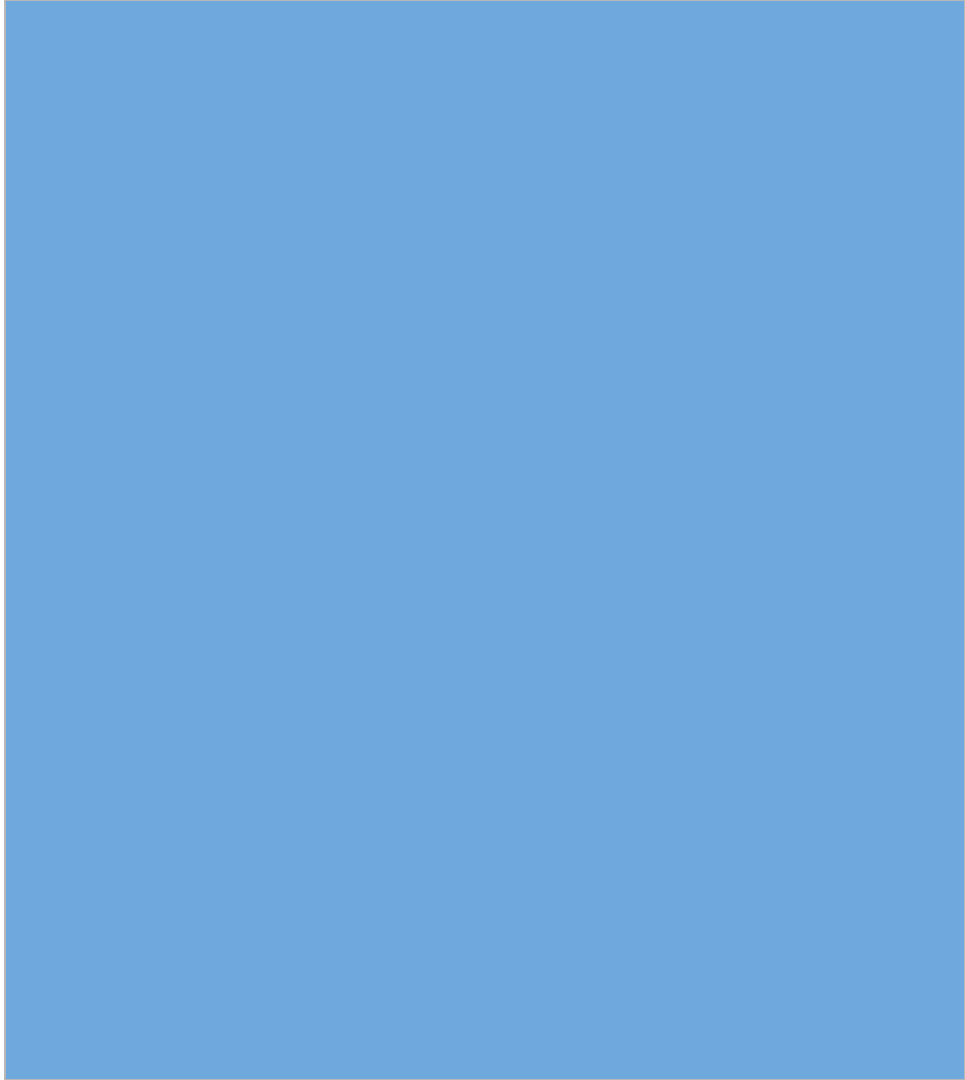
`LineNumberReader`: tipo especial de `BufferedReader` que lleva un conteo del número de línea que se está leyendo en cada momento (ejemplo página 228).

`CharArrayReader`: recoge de un buffer los datos provenientes de un array de caracteres (ejemplo página 228).

`StringReader`: recoge de un buffer los datos provenientes de un `String` (ejemplo página 229).

Java

Lectura y escritura en ficheros de texto con la clase Scanner



Lectura con la clase Scanner

La clase `Scanner` (paquete `java.util`) permite leer número y texto de un fichero de texto y de otras fuentes.

Permite la lectura de ficheros línea a línea (muy similar a `BufferedReader`).

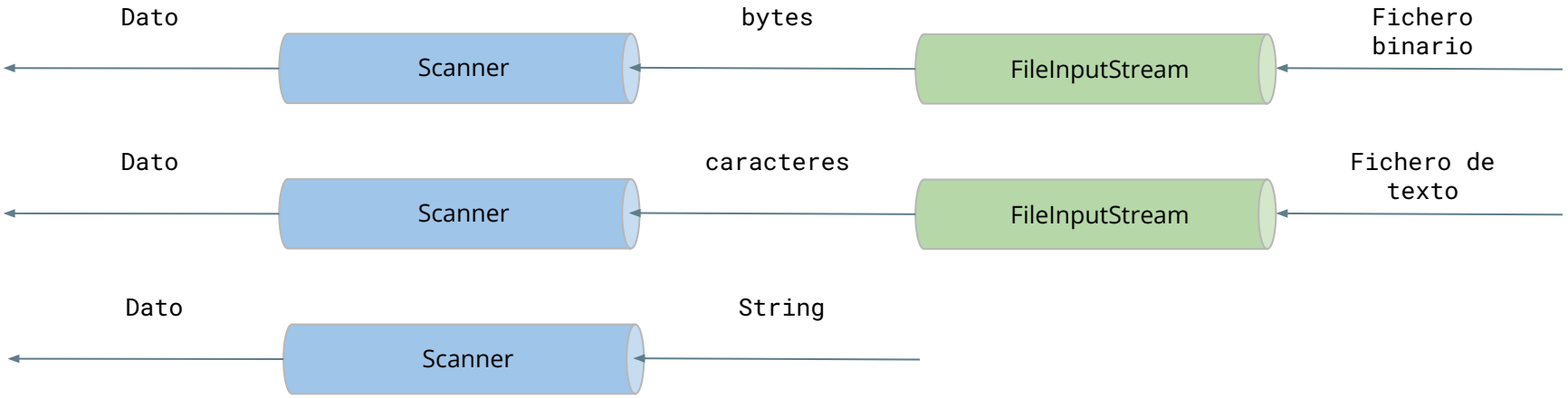
Permite la lectura sencilla de números y palabras separados por algún **separador** específico. El separador por defecto es cualquier espacio en blanco (espacio, salto de línea, tabulador, etc.).

Permite reconocer patrones de texto conocidos como **expresiones regulares**.

Tarea: ¿qué son las expresiones regulares?

Lectura con la clase Scanner

Además, la clase Scanner puede leer de un `InputStream` y de un `FileReader`.



Principales operaciones de la clase Scanner

Descripción	Declaración
Constructor. Requiere un `InputStream`.	`Scanner(InputStream source)`
Constructor. Requiere un objeto que implemente `Readable` (por ejemplo un `FileReader`).	`Scanner(Readable source)`
Constructor. Requiere un `String`.	`Scanner(String source)`
Cerrar.	`void close()`
Leer una línea.	`String nextLine()`
Indica si quedan líneas por leer.	`boolean hasNextLine()`
Lee un booleano.	`boolean nextBoolean()`

Principales operaciones de la clase Scanner

Descripción	Declaración
Indica si es posible leer una palabra que se interprete como un `booleano/double/int`.	`boolean hasNextBoolean(), hasNextDouble(), hasNextInt()`
Leer una palabra.	`String next()`
Indica si quedan más palabras o datos por leer.	`boolean hasNext()`
Leer un `double`.	`double nextDouble()`
Leer un `int`.	`int nextInt()`
Cambia el delimitador que separa los ítems.	`Scanner useDelimiter(String pattern)`

Excepciones de la clase Scanner

El uso de estas operaciones puede generar algunas excepciones tales como:

- `NoSuchElementException`: no quedan más palabras.
- `IllegalStateException`: el scanner está cerrado.
- `InputMismatchException`: el dato leído no es del tipo esperado.

Tarea: haz un esquema de las clases vistas para el manejo de ficheros binarios y de ficheros de caracteres. Además, enumera las principales operaciones para todas ellas.

Clase Scanner, ejemplo

```
public static void main() {
	final String nomFich = "datos.txt";
	Scanner in = null;

	try {
		// Abre el fichero
		in = new Scanner(new FileReader(nomFich));
		// Lee el fichero palabra a palabra
		while (in.hasNext()) {
			// Lee primera palabra
			String palabra = in.next();
			System.out.println("Palabra:" + palabra);
			// Lee números
			while (in.hasNextDouble()) {
				// Lee un double
				double d = in.nextDouble();
				System.out.println("Número: " + d);
			}
		}
```

Clase Scanner, ejemplo

```
	} catch (FileNotFoundException e) {
		System.out.println("Error abriendo el fichero " + nomFich);
	} finally {
		if (in != null)
			in.close();
	}
}
```


¿Qué muestra por pantalla?

Grado Superior
Programación

Ficheros

Curso 2023/24

Profesor
Jonathan Carrero