Parte I: Programación en un lenguaje orientado a objetos

1. Introducción a los lenguajes de programación
2. Datos y expresiones
3. Estructuras algorítmicas
4. Datos compuestos
5. Tratamiento de errores
6. *Entrada/salida con ficheros*
   - Escritura de texto con formato. Ficheros. Flujos de datos. Escritura de ficheros de texto. Lectura de ficheros de texto.
7. Herencia y Polimorfismo

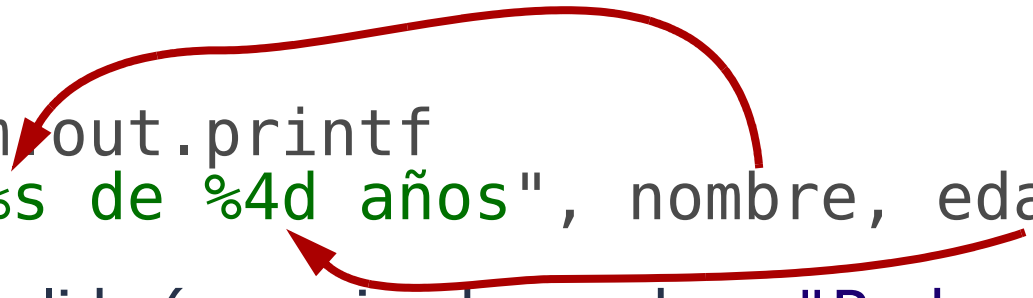
1. Escritura de texto con formato

Las clases que permiten salida de texto disponen de una operación de salida de texto con formato, llamada `printf`

- por ejemplo, el objeto `System.out` que representa la pantalla
- está copiada del lenguaje C
- el primer parámetro es el string de formato, que indica cómo se va a formatear la salida
- luego viene un número variable de parámetros

Ejemplo

```
System.out.printf
    ("%s de %4d años", nombre, edad);
```



Produce la salida (suponiendo nombre="Pedro", edad=18)

```
Pedro de   18 años
```

String de formato

Contiene caracteres que se muestran tal cual

- y especificaciones de formato que se sustituyen por los sucesivos parámetros

Especificaciones de formato más habituales:

```
%d      enteros
%c      caracteres
%s      string
%f      float y double, coma fija
%e      float y double, notación exponencial
%g      float y double, exponencial o coma fija
%n      salto de línea en el formato del sist. operat.
%%      el carácter %
```

String de formato

Puede lanzarse `IllegalFormatException` si el formato no corresponde al parámetro

Después del carácter `%` se puede poner un carácter de opciones:

```
-       alinear a la izquierda
0       rellenar con ceros (números sólo)
+       poner signo siempre (números sólo)
```

Especificación de anchura y precisión

Puede añadirse después del "%" (y del carácter de opción si lo hay) la especificación de anchura mínima y (si corresponde) el número de decimales; ejemplos

Parámetros de printf()	Salida
("Pi= %4.0f %n",Math.PI)	Pi= 3
("Pi= %4.2f %n",Math.PI)	Pi= 3.14
("Pi= %12.4f %n",Math.PI)	Pi= 3.1416
("Pi= %12.8f %n",Math.PI)	Pi= 3.14159265
("I= %8d %n",18	I= 18
("I= %4d %n",18)	I= 18
("I= %04d %n",18)	I= 0018

Separador de cifras decimales

El sistema usa como separador de cifras decimales

- El '.' si está configurado en inglés
- La ',' si está configurado en español

Para forzar la utilización del punto como separador de las cifras decimales:

```
import java.util.Locale;
...
  Locale.setDefault(Locale.ENGLISH);
```

Y para la coma:

```
  Locale.setDefault(new Locale("es"));
```

2. Ficheros

Fichero:

- secuencia de bytes en un dispositivo de almacenamiento: disco duro, DVD, memoria USB, ...
- se puede leer y/o escribir
- se identifica mediante un nombre
  - absoluto (*pathname*)
    `/home/pepe/documentos/un_fichero`
  - o relativo (a la carpeta del proyecto con el `main`)
    `un_fichero`

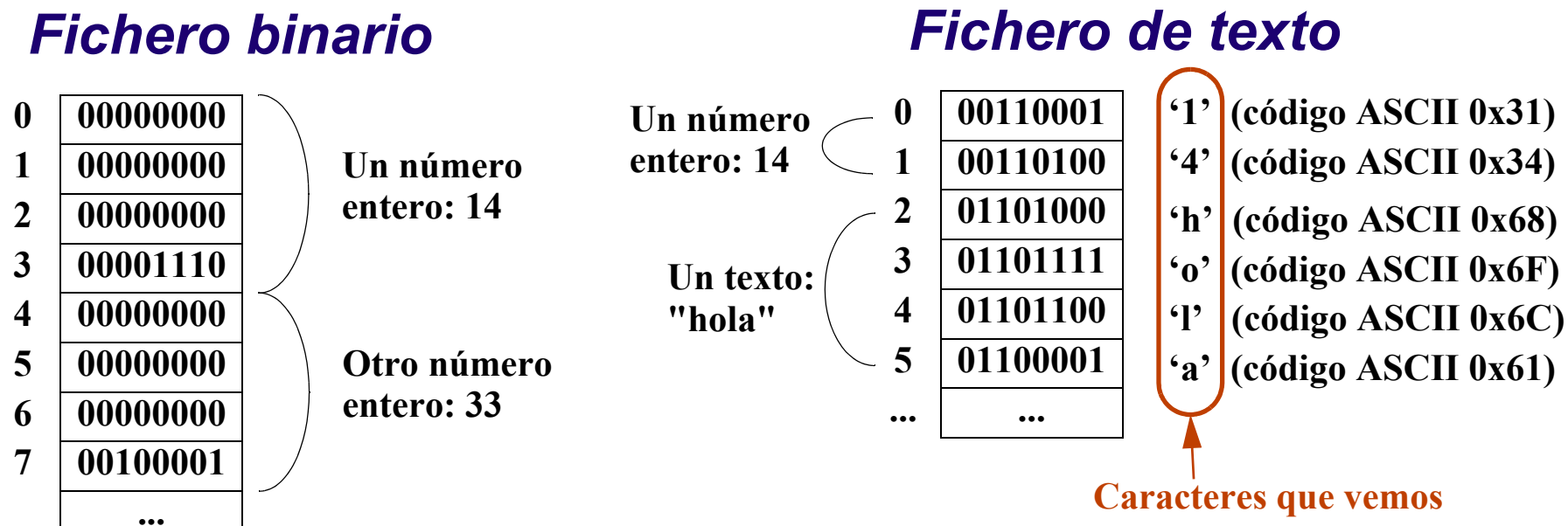
Tipos de ficheros:

- ***programas***: contienen instrucciones
- ***datos***: contienen información, como números (enteros o reales), secuencias de caracteres, ...

Ficheros de texto y binarios

Tipos de ficheros de datos:

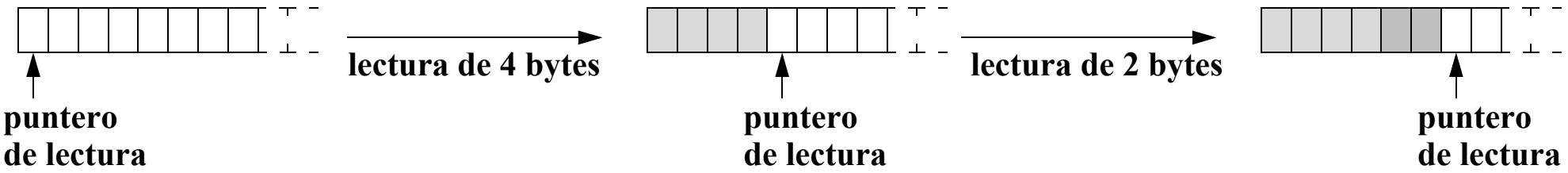
- ***de bytes*** (binarios): pensados para ser leídos por un programa
- ***de caracteres*** (de texto): pueden ser leídos y/o creados por una persona



Punteros de lectura y escritura

- Indican el próximo byte a leer o a escribir
- Gestionados automáticamente por el sistema operativo
- Comienzan apuntando al primer byte del fichero
- Van avanzando por el fichero según se van leyendo/escribiendo sus contenidos

Ejemplo:



3. Flujos de datos (I/O Streams)

La Entrada/Salida de Java se organiza generalmente mediante objetos llamados ***I/O Streams***

Un ***I/O Stream*** es una secuencia ordenada de datos con un determinado origen o destino

Hay dos tipos de streams:

- ***de bytes*** (binarios)
- ***de caracteres*** (de texto)

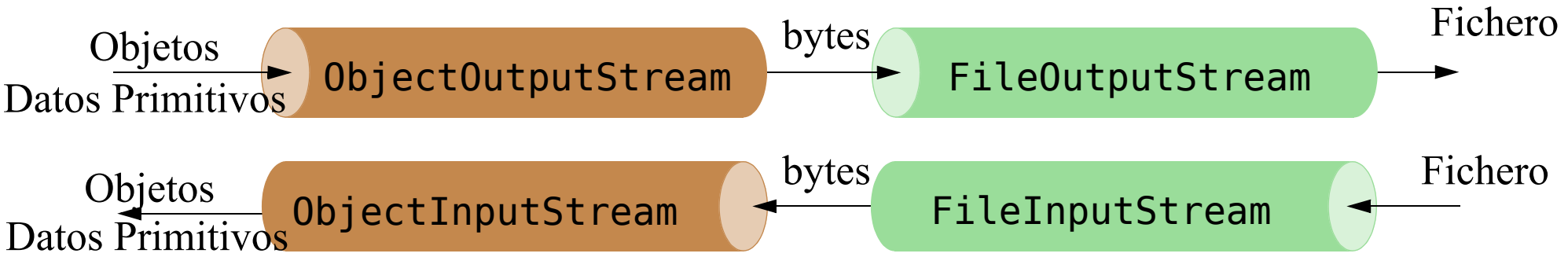
En caso de error, muchos de los métodos de los streams lanzan `IOException` y otras excepciones

Casi todos están definidos en `java.io`

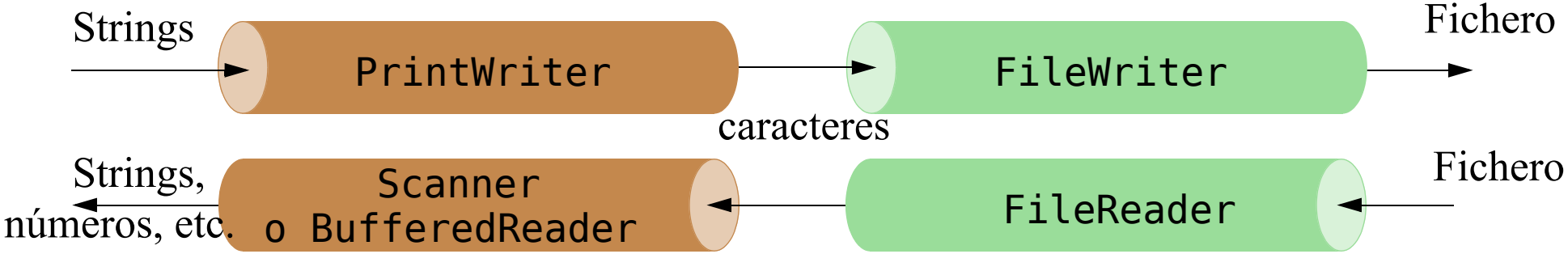
Uso de Streams para acceder a ficheros

Normalmente se utilizan encadenados por parejas:

Binarios



De Texto:



Objetos predefinidos

`System.out` es un objeto de la clase `OutputStream` que representa la pantalla

- métodos `print, println, printf, ...`

`System.in` es un objeto de la clase `InputStream` que representa el teclado

Deberían ser de las clases `PrintWriter` y `BufferedReader`

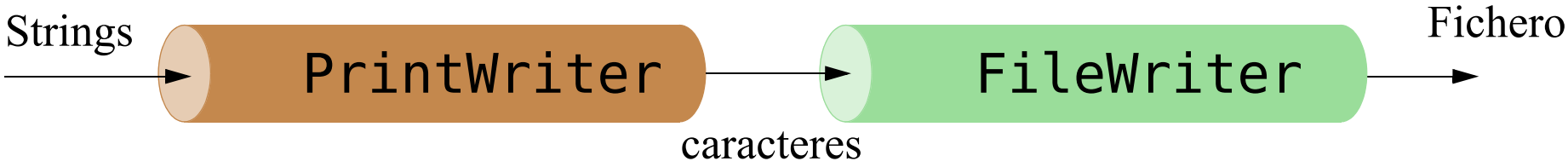
- pero los *streams* de caracteres no existían en las primeras versiones de Java
- siguen siendo *streams* binarios por compatibilidad con versiones antiguas

4. Escritura de ficheros de texto

Es posible escribir datos primitivos y strings en un fichero de texto

Se usa la pareja de *streams:*

- `FileWriter`: escritura básica de caracteres en un fichero
- `PrintWriter`: proporciona funciones de escritura más cómodas



Clase `FileWriter`

Operaciones más habituales:

Descripción	Declaración
Constructor. Requiere el nombre del fichero. Lo crea si no existe. Si existe se borran sus contenidos. Lanza `IOException` si el fichero no se puede crear	`FileWriter(String nombreFichero)` `  throws IOException`

Constructor igual que el anterior, salvo en que cuando `añade` es `true` no se borran los contenidos, sino que los datos se añaden al final del fichero	`FileWriter`
	`  (String nombreFichero,`
	`  boolean añade)`
	`  throws IOException`

Clase `PrintWriter`

Operaciones más habituales:

Descripción	Declaración
Constructor. Requiere un `Writer` (usaremos un `FileWriter`)	`PrintWriter(Writer writer)`
Escribir un string	`void print(String str)`
Escribir un string con retorno de línea	`void println(String str)`
Escribe los argumentos con el formato deseado	`void printf(String formato, Object... args)`
Cerrar	`void close()`

Ejemplo sencillo: patrón de uso de los streams (anterior a Java 7)

```
PrintWriter out = null;
try {
  // crea los streams y los conecta
  out = new PrintWriter(new FileWriter(nomFich));

  // escribe los datos en el fichero
  out.println("Hola");
  ...
} catch (IOException e){
  ...
} finally {
  if (out != null) {
    // cierra los streams
    out.close();
  }
}
```

Patrón de uso de los streams a partir de Java 7

Crea los streams dentro del `try`, entre paréntesis

- El `finally` no se pone: el fichero se cierra *automáticamente*, haya error o no

```
// Crea los streams y los conecta en el propio try
try (PrintWriter out = new PrintWriter(
        new FileWriter(nomFich)))
{
  // escribe los datos en el fichero
  out.println("Hola");
  ...
} catch (IOException e){
  ...
}
```

Ejemplo: método que escribe en un fichero de texto

```
public static void escribeFichTexto
   (String nomFich,
    int i, double x, String str)
{
   // crea los streams y los conecta, en el try
   try (PrintWriter out = new PrintWriter(
           new FileWriter(nomFich))
   {
      // escribe los datos en el fichero
      out.println("Entero: "+i+"   Real: "+x);
      out.println("String: "+str);
   } catch (IOException e){
      System.out.println("Error al abrir "+nomFich);
   }
}
```

Ejemplo: método que escribe en un fichero de texto (cont.)

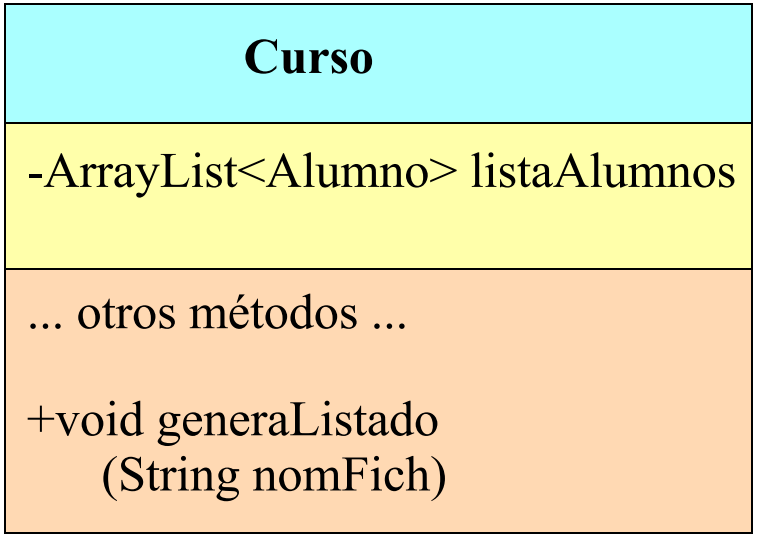
Ejemplo de fichero generado:

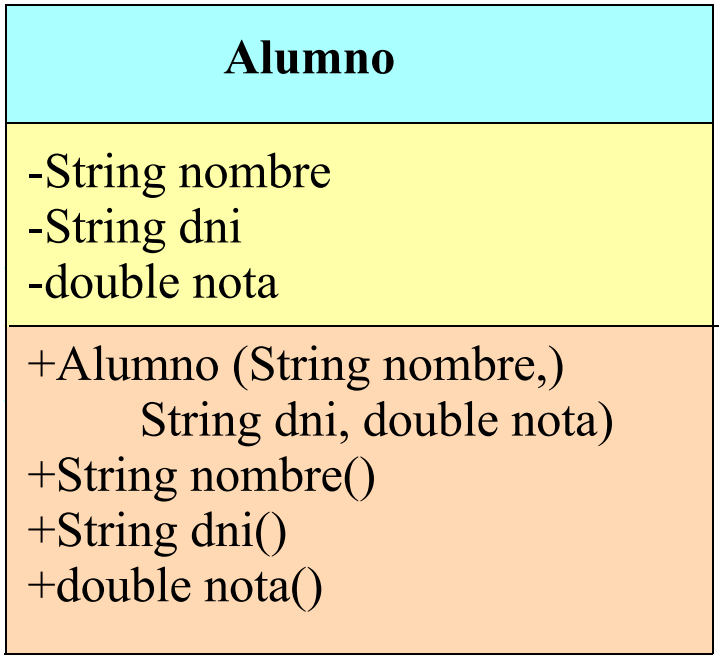
```
Entero: 11   Real: 22.2
String: hola
```

Ejemplo: escritura de fichero de texto con formato (método `printf`)

Añadir el método `generaListado` a la clase `Curso`:

- Escribe en un fichero de texto los datos de todos los alumnos del curso





Ejemplo: escritura de fichero de texto con formato (método `printf`) (cont.)

```
/**
 * Genera un listado de los alumnos del curso en un
 * fichero de texto
 * @param nomFich nombre del fichero generado
 */
```

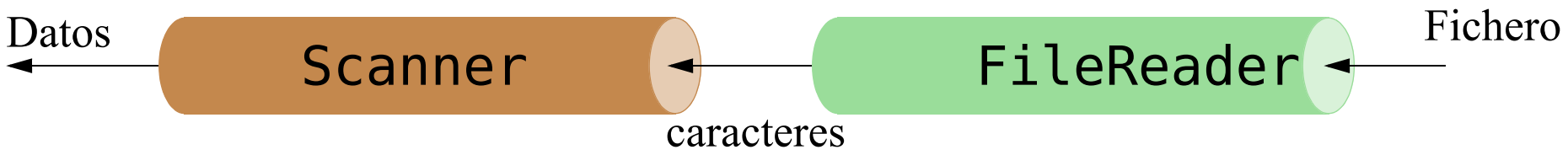
Ejemplo: escritura de fichero de texto con formato (método `printf`) (cont.)

```
public void generaListado(String nomFich){
  try (PrintWriter out=new PrintWriter
          (new FileWriter(nomFich)))
  {
    // escribe el listado alumno por alumno
    for(Alumno a: listaAlumnos) {
      // nombre con 25 carac. justificado a la izq.
      // nota con 4 carac. totales con un decimal
      out.printf("%-25s  DNI:%s  Nota:%4.1f%n",
                 a.nombre(),a.dni(), a.nota());
    }
  } catch (IOException e){
    System.out.println("Error al abrir "+nomFich));
  }
}
```

5. Lectura de ficheros de texto

La lectura de un fichero de texto se realiza con la pareja de *streams:*

- `FileReader`: permite leer caracteres de un fichero
- `Scanner`: permite convertir grupos de caracteres en strings o datos primitivos (definida en `java.util`)



Clase `FileReader`

Operaciones habituales:

Descripción	Declaración
Constructor. Requiere el nombre del fichero. Si no existe lanza `FileNotFoundException`	`FileReader(String nombreFich)` `  throws FileNotFoundException`

Clase `Scanner`

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

- permite la lectura del texto línea a línea
- permite la lectura sencilla de números y palabras separadas por el separador especificado
  - el separador por defecto es cualquier tipo de espacio en blanco (espacio, salto de línea, tabulador, etc.)
  - puede utilizarse otro separador cambiándolo con el método `useDelimiter`
- permite reconocer patrones de texto conocidos como "*expresiones regulares*"

Principales operaciones de la clase `Scanner`

Descripción	Declaración
Constructor. Requiere un objeto que implemente `Readable` (por ejemplo un `FileReader`)	`Scanner(Readable source)`
Constructor. Requiere un String	`Scanner(String source)`
Cerrar	`void close()`
Configura el formato de los números. Usar `Locale.ENGLISH` para leer números que utilicen el carácter `'.'` como punto decimal Usar `new Locale("es")` para leer números que utilicen el carácter `','` como punto decimal	`Scanner useLocale (Locale locale)`

Principales operaciones de la clase `Scanner` (cont.)

Descripción	Declaración
Leer una línea	`String nextLine()`
Indica si quedan más líneas por leer	`boolean hasNextLine()`
Leer un booleano	`boolean nextBoolean()`
Indica si la siguiente palabra es un valor booleano	`boolean hasNextBoolean()`
Leer una palabra	`String next()`
Indica si quedan más palabras o datos por leer	`boolean hasNext()`
Leer un double	`double nextDouble()`
Indica si la siguiente palabra es un double	`boolean hasNextDouble()`
Leer un int	`int nextInt()`
Indica si la siguiente palabra es un int	`boolean hasNextInt()`

Principales operaciones de la clase `Scanner` (cont.)

Descripción	Declaración
Cambia el delimitador que separa los items	`Scanner useDelimiter (String pattern)`

Excepciones que pueden lanzar

- `NoSuchElementException`: se ha intentado leer, pero no quedan más palabras o datos
- `IllegalStateException`: el *scanner* está cerrado
- `InputMismatchException`: el dato leído no es del tipo esperado

Ejemplo: lectura línea a línea con la clase `Scanner`

```
try (Scanner in= new Scanner
     (new FileReader(nomFich)))
{
   // lee el fichero línea a línea
   while (in.hasNextLine()) {
      // lee la siguiente línea y la muestra
      System.out.println(in.nextLine());
   }
} catch (FileNotFoundException e) {
   System.out.println("Error abriendo " + nomFich);
}
```

Ejemplo: procesado de fichero de texto con la clase `Scanner`

- Para el fichero (`datos.txt`):

```
azul 1.0 3.5 7.7
rojo 2
verde 10.0 11.1
```

- Se desea obtener la siguiente salida por consola:

```
Palabra:azul
Número:1.0
Número:3.5
Número:7.7
Palabra:rojo
Número:2.0
Palabra:verde
Número:10.0
Número:11.1
```

Ejemplo: procesado de fichero de texto con la clase `Scanner` (cont.)

```
public static void main() {
  final String nomFich="datos.txt";
  try (Scanner in= new Scanner
       (new FileReader(nomFich)))
  {
    // configura el formato de números
    in.useLocale(Locale.ENGLISH);
    // lee el fichero palabra a palabra
    while (in.hasNext()) {
      // lee primera palabra y la escribe
      String palabra = in.next();
      System.out.println("Palabra:"+palabra);
```

Ejemplo: procesado de fichero de texto con la clase `Scanner` (cont.)

```
        // lee números
        while (in.hasNextDouble()) {
          // lee un double
          double d = in.nextDouble();
          System.out.println("Número:"+d);
        }
      } // fin while (in.hasNext())
    } catch (FileNotFoundException e) {
      System.out.println("Error abriendo " + nomFich);
    }
} // fin main
```