Prácticas de Programación

Tema 7. Entrada/salida con ficheros

7.1 Ficheros

Fichero:

- secuencia de bytes en un dispositivo de almacenamiento: disco duro, CD, DVD, memoria USB, ...
- se puede leer y/o escribir
- se identifica mediante un nombre (***pathname***)
  - `/home/pepe/documentos/un_fichero`

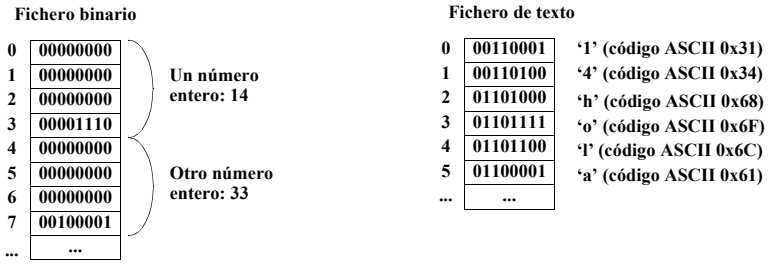
Tipos de ficheros:

- programas: contienen instrucciones
- datos: contienen información, como números (enteros o reales), secuencias de caracteres, ...
- en algunos sistemas operativos (como Linux) también son ficheros los directorios, los dispositivos, las tuberías, ...

Ficheros de texto y binarios

Tipos de ficheros de datos:

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

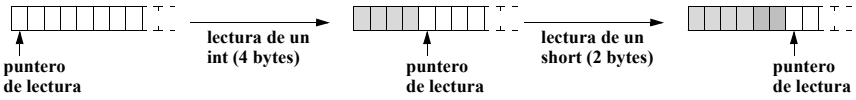


- **Para "entender" los contenidos de un fichero es necesario conocer de antemano el tipo de datos que contiene**

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 sus contenidos**

Ejemplo:

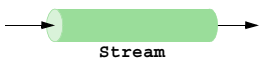


7.2 Flujos de datos (*streams*)

La Entrada/Salida de Java se organiza generalmente mediante objetos llamados `Streams`

Un `Stream` es la generalización de un fichero:

- **secuencia ordenada de datos con un determinado origen y destino**
- **su origen o destino puede ser un fichero, pero también un string o un dispositivo (p.e. el teclado)**



Para poder usar un *stream* primero hay que *abrirle*

- **se abre en el momento de su creación**
- **y hay que *cerrarle* cuando se deja de utilizar**

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

Clasificación de los *streams*

Por el tipo de datos que "transportan":

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

Por el sentido del flujo de datos:

- ***de entrada*****: los datos fluyen desde el dispositivo o fichero hacia el programa**
- ***de salida*****: los datos fluyen desde el programa al dispositivo**

Según su cercanía al dispositivo:

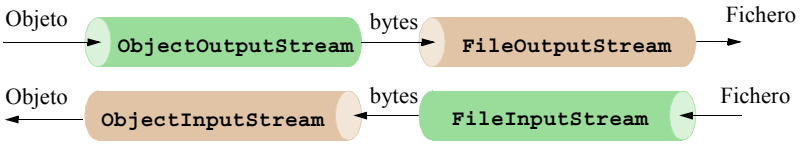
- ***iniciadores*****: son los que directamente vuelcan o recogen los datos del dispositivo**
- ***filtros*****: se sitúan entre un *stream* iniciador y el programa**

Uso de los Streams

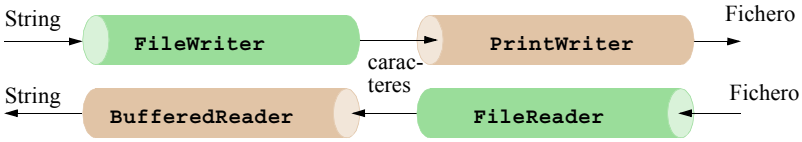
Normalmente se utilizan por parejas

- **formadas por un *stream* iniciador y un filtro**

Binarios

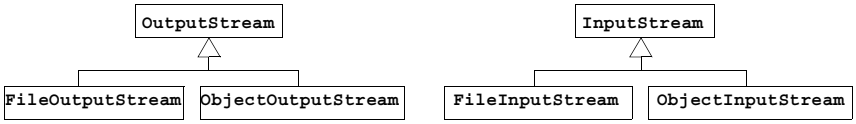


De Texto:



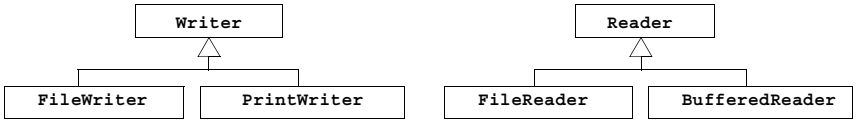
Jerarquía de clases Principales *Streams* binarios

- `OutputStream`**: escritura de ficheros binarios**
  - `FileOutputStream` **(iniciador): escribe bytes en un fichero**
  - `ObjectOutputStream` **(filtro): convierte objetos y variables en arrays de bytes que pueden ser escritos en un** `OutputStream`
- `InputStream`**: lectura de ficheros binarios**
  - `FileInputStream` **(iniciador): lee bytes de un fichero**
  - `ObjectInputStream` **(filtro): convierte en objetos y variables los arrays de bytes leídos de un** `InputStream`



Jerarquía de clases Principales *Streams* de caracteres

- `Writer`: escritura de ficheros de texto
  - `FileWriter` (iniciador): escribe texto en un fichero
  - `PrintWriter` (filtro): permite convertir a texto variables y objetos para escribirlos en un `Writer`
- `Reader`: lectura de ficheros de texto
  - `FileReader` (iniciador): lee texto de un fichero
  - `BufferedReader` (filtro): lee texto (línea a línea) de un `Reader`



Objetos *stream* predefinidos

`System.out`: Salida estándar (consola)
- objeto de la clase `PrintStream` (subclase de `OutputStream`)
  - métodos `print`, `println`, `printf`, ...

`System.err`: Salida de error (consola)
- también es un objeto de la clase `PrintStream`

`System.in`: Entrada estándar (teclado)
- objeto de la clase `InputStream`

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

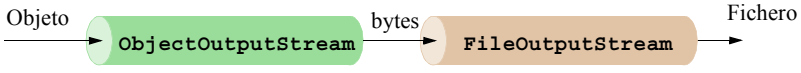
7.3 Salida binaria

Es posible escribir variables y objetos en un fichero binario

Para poder escribir un objeto su clase debe implementar la interfaz `Serializable`, de la manera siguiente:

```
import java.io.*;
public class Nombre implements Serializable
{...}
```

Se usa la pareja de *streams* `FileOutputStream` (iniciador) y `ObjectOutputStream` (filtro)



Clase `FileOutputStream`

Operaciones más habituales:

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

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	`FileOutputStream(String s,`
	`  boolean añade)`
	`  throws FileNotFoundException`

Sincronizar	`void flush()`
Cerrar	`void close()`

Clase `ObjectOutputStream`

Operaciones más habituales:

Descripción	Declaración
Constructor. Requiere un `OutputStream`	`ObjectOutputStream(` `  OutputStream out)`
Escribir un booleano	`void writeBoolean(boolean b)`
Escribir un double	`void writeDouble(double d)`
Escribir un int	`void writeInt(int i)`
Escribir un objeto Se escriben también los objetos a los que el objeto `obj` se refiere (y así recursivamente)	`void writeObject(Object obj)`
Sincronizar (llama a `out.flush()`)	`void flush()`
Cerrar (llama a `out.close()`)	`void close()`

Todos los métodos (incluido el constructor) lanzan `IOException`

- **error al acceder al `OutputStream` (normalmente un fichero)**

Ejemplo de escritura de objeto

Añadir a la clase `Agenda` del problema 2 una operación para grabar una agenda (incluyendo los contactos que contiene), a un fichero:

- **Previamente ha sido necesario declarar las clases que se van a escribir como `Serializable`:**

```
import java.io.*;
public class Contacto implements Serializable {...
```

```
import java.io.*;
public class Agenda implements Serializable {...
```

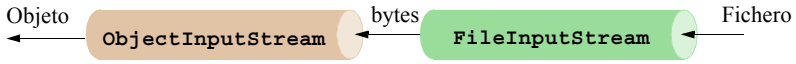
El método a añadir a la clase `Agenda` es:

```
public void salvaAFichero(String nomFich)
                                  throws IOException {
  ObjectOutputStream sal = null;
  try {
    // abre los streams iniciador y filtro
    sal = new ObjectOutputStream(
                    new FileOutputStream(nomFich));
    // graba el objeto actual
    sal.writeObject(this);
  } finally {
    if (sal != null) {
      sal.close(); // cierra los streams
    }
  }
}
```


7.4 Entrada binaria

Es posible leer variables y objetos de un fichero binario que fue creado según lo expuesto en el apartado anterior

Se usa la pareja de *streams* `FileInputStream` (iniciador) y `ObjectInputStream` (filtro)



Clase `FileInputStream`

Operaciones más habituales:

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

Clase `ObjectInputStream`

Descripción	Declaración
Constructor. Requiere un `InputStream`	`ObjectInputStream(InputStream in)`
Leer un booleano	`boolean readBoolean()`
Leer un double	`double readDouble()`
Leer un int	`int readInt()`
Leer un objeto (o un string). Al leer un objeto se leen también los objetos a los que éste se refiere	`Object readObject()`
Cerrar	`void close()`

- **Los métodos lanzan las excepciones:**
  - `IOException`: **problema al acceder al** `InputStream`
  - `EOFException`: **alcanzado el fin de fichero**
  - `ClassNotFoundException`: **sólo producida por** `readObject`

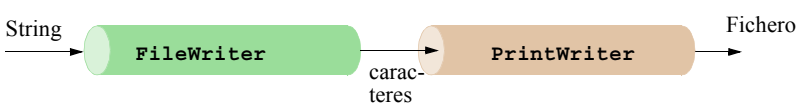
Ejemplo de lectura de objetos

```
static public Agenda leeDeFichero(String nomFich)
       throws IOException, ClassNotFoundException {
  ObjectInputStream ent = null;
  try {
    // abre el fichero
    ent = new ObjectInputStream(
                        new FileInputStream(nomFich));
    // lee el objeto y le retorna
    return (Agenda)ent.readObject();
  } finally {
    if (ent != null) {
      ent.close(); // cierra los streams
    }
  }
}
```


7.5 Salida de texto

Es posible escribir variables y strings en un fichero de texto

Se usa la pareja de *streams* `FileWriter` (iniciador) y `PrintWriter` (filtro)



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 `FileNotFoundException` si el fichero no se puede crear	`FileWriter(String s) throws IOException, FileNotFoundException`
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 s, boolean añade) throws IOException, FileNotFoundException`

Clase `PrintWriter`

Operaciones más habituales:

Descripción	Declaración
Constructor. Requiere un `Writer`	`PrintWriter(Writer writer)`
Constructor. Crea el `FileWriter` internamente	`PrintWriter(String nomFich)`
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	`printf(String formato, Object... args)`
Sincroniza e informa si ha habido un error	`boolean checkError()`
Sincronizar	`void flush()`
Cerrar	`void close()`

- **Los métodos *no lanzan* `IOException`:**
  - **para saber si ha habido un error hay que llamar a `checkError`**

Ejemplo: escritura fichero de texto

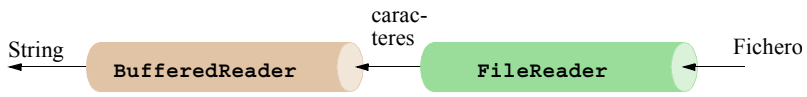
```
static void ejemploEscribeFichTexto(String nomFich,
    int i, double x, String str) throws IOException {
  PrintWriter out = null;
  try {
    // crea los streams
    out = new PrintWriter(nomFich);
    // escribe los datos en el fichero
    out.println("Entero: "+i+"   Real: "+x);
    out.println("String: "+str);
  } finally {
    if (out != null)
      out.close();
  }
}
```

Fichero generado:

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


7.6 Entrada de texto

La lectura de un fichero de texto se realiza con la pareja de *streams* `FileReader` (iniciador) y `BufferedReader` (filtro)



En muchos casos será más práctico utilizar la clase `Scanner` en lugar de `BufferedReader`

- **lo veremos en "Lectura con la clase Scanner" en la página 40**

Clase `FileReader`

Operaciones habituales:

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

Clase `BufferedReader`

Operaciones habituales

Descripción	Declaración
Constructor. Requiere un `Reader`	`BufferedReader(Reader reader)`
Leer un string. Retorna `null` si se ha llegado al final Lanza `IOException` si se produce un error al acceder al `Reader`	`String readLine()` `throws IOException`
Cerrar. Lanza `IOException` si se produce un error al acceder al `Reader`	`void close()` `throws IOException`

La lectura se realiza línea a línea con `readLine()`

- **luego cada línea se procesa (con `split()`, `indexOf()`, ...)**
- **y los trozos se convierten a otros datos con las operaciones `parseInt()`, `parseDouble()`, ...**
- **existe una forma alternativa de procesado: clase `Scanner`**

Ejemplo de lectura de fichero de texto

Leer parejas de texto (una línea) y número (otra línea) y escribirlas en pantalla; ejemplo de fichero de entrada:

```
primero
1.0
segundo
3.0
tercero
    4.5
cuarto
8.0
quinto
34R.5
sexto
1.0
```



```
import java.io.*;

public class LeeStrings {

  public static void main(String[] args)
                                throws IOException {
    String str, num;
    double x;
    BufferedReader ent = null;
    try {
      ent = new BufferedReader(
                        new FileReader("d2.txt"));
      do {
        str=ent.readLine(); // lee una línea
        // si hay más líneas seguimos procesando
        if (str!=null) {
          // escribe la línea de texto
          System.out.println("Texto: "+str);
```

```
            // lee línea y la convierte a número
            num=ent.readLine();
            try {
               x=Double.parseDouble(num);
               System.out.println("Numero:"+x);
            } catch (NumberFormatException e) {
               System.out.println ("Error al "+
                     "leer el numero real: "+num);
            } // try
          } // if
        } while (str!=null);
      } finally {
          if (ent!=null) {
              ent.close();
          }
      } // try
    } // main
} // clase
```


El programa muestra lo siguiente en la pantalla:

```
Texto: primero
Numero: 1.0
Texto: segundo
Numero: 3.0
Texto: tercero
Numero: 4.5
Texto: cuarto
Numero: 8.0
Texto: quinto
Error al leer el número real: 34R.5
Texto: sexto
Numero: 1.0
```


7.7 Entrada/Salida de texto con formato

La clase `PrintWriter` dispone de una operación de salida de texto con formato, llamada `printf`

- **el objeto `System.out` que representa la pantalla, también**
- **está copiada del lenguaje C**
- **el primer parámetro es el string de formato**
- **luego viene un número variable de parámetros**

Ejemplo

```
System.out.printf
   ("%s de %3d 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 %
```


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)
```

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

```
import java.util.Locale;
...
  Locale.setDefault(Locale.ENGLISH);
  ... // usa printf
```


Especificación de anchura y precisión

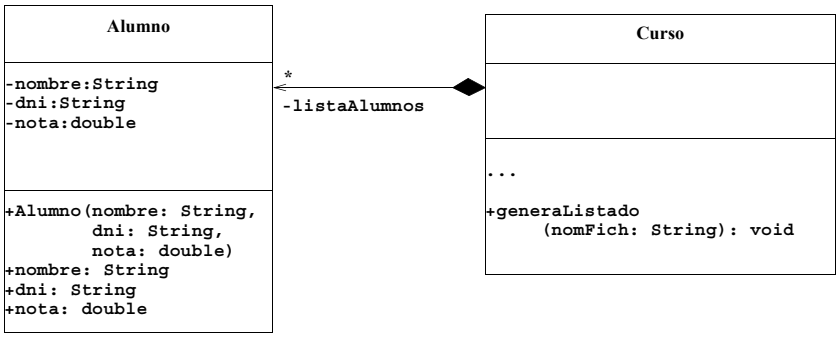
Puede añadirse después del "`%`" (y el carácter de opción si lo hay) la especificación de anchura mínima y/o número de decimales; ejemplos

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

Ejemplo de uso de `printf`

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

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




```
public void generaListado(String nomFich)
                                    throws IOException {
  PrintWriter out = null;
  try {
    // abre el fichero de texto
    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());
    }
  } finally {
    if (out!=null)
      out.close();
  }
}
```


Fichero de texto generado:

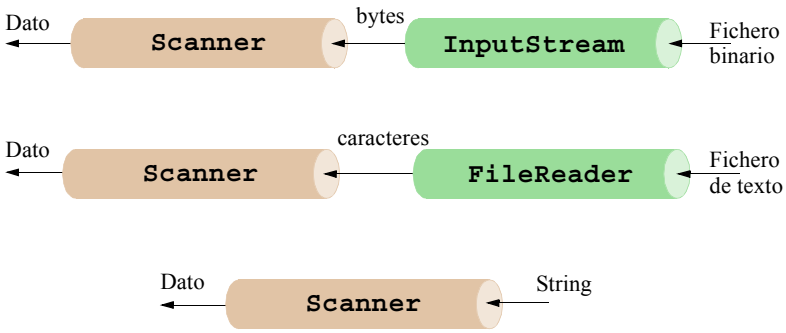
```
Pepe García Hernández       DNI:123456789  Nota: 5.0
Lolo Hernández García       DNI:234567890  Nota: 0.0
Manu López Gómez            DNI:345678901  Nota:10.0
Pepito Gómez López          DNI:456789012  Nota: 7.5
```

Lectura con la 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 (similar a `BufferedReader`)**
- **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 utilizando el método `useDelimiter`**
- **permite reconocer patrones de texto conocidos como "expresiones regulares"**

Algunos usos de la clase `Scanner`



`Scanner` se comporta como un *stream* filtro de caracteres

- **aunque técnicamente no lo es, ya que no extiende a `Reader`**

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()`
Configura el formato de los números. Usar `Locale.ENGLISH` para leer números que utilicen el carácter `'.'` como punto decimal	`Scanner useLocale(` `  Locale locale)`

Descripción	Declaración
Leer una línea	`String nextLine()`
Indica si quedan más líneas por leer	`boolean hasNextLine()`
Leer un booleano	`boolean nextBoolean()`
Indica si es posible leer una palabra que se interprete como un 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 es posible leer una palabra que se interprete como un double	`boolean hasNextDouble()`
Leer un int	`int nextInt()`
Indica si es posible leer una palabra que se interprete como int	`boolean hasNextInt()`
Cambia el delimitador que separa los ítems	`Scanner useDelimiter( String pattern)`

Excepciones que pueden lanzar

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

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

```
public static void main() {
  final String nomFich="datos.txt";
  Scanner in = null;
  try {
    // abre el fichero
    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
      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);
        }
      } // while (in.hasNext())
    } catch (FileNotFoundException e) {
      System.out.println("Error abriendo el fichero "
                                              +nomFich);
    } finally {
      if (in!=null){
        in.close();
      }
    } // try
  } // main
```


- **Para el fichero:**

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

- **La salida producida será:**

```
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 de procesado de la entrada estándar con la clase `Scanner`

Lectura de un entero, una palabra y un texto de la entrada estándar (teclado)

```
Scanner lectura = new Scanner(System.in);
int n = lectura.nextInt();
String palabra=lectura.next();
double d=lectura.nextDouble();

System.out.println("entero: "+n+ " palabra: "+
   palabra+" real: "+d);
```

7.8 Uso de ficheros como tablas

Hasta ahora todos los ficheros se han manejado mediante clases que representan secuencias

En muchas ocasiones es conveniente poder acceder a los datos en cualquier orden, como en una tabla

- **las diferencias con un array son**
  - **tamaño no limitado a priori**
  - **memoria persistente (no volátil)**

Para ello Java dispone de la clase `RandomAccessFile`

- **permite acceso aleatorio al fichero**
- **no es un `Stream`**
- **se puede usar para leer, o leer y escribir a la vez**

Ficheros de acceso aleatorio

En java son ficheros que contienen bytes

Se numeran con un índice que empieza en cero

Existe un índice almacenado en el sistema que se llama el puntero de lectura/escritura

- **las lecturas y escrituras se hacen a partir de él, en secuencia**
- **es posible cambiar el puntero de lectura/escritura**

Si escribimos pasado el final del fichero, su tamaño se amplía

Principales operaciones

Descripción	Declaración
Constructor. El String `fich` indica el nombre del fichero. El String modo será `"r"` para leer y `"rw"` para leer y escribir	`RandomAccessFile (String fich, String modo) throws FileNotFoundException`
Retorna el tamaño actual en bytes del fichero	`long length() throws IOException`
Cambia el tamaño del fichero al indicado, en bytes	`void setLength(long nueva) throws IOException`
Pone el puntero de lectura/escritura a pos	`void seek(long pos) throws IOException`
Retorna el puntero de lectura/escritura	`long getFilePointer() throws IOException`

Principales operaciones

Descripción	Declaración
Intenta leer un array de bytes. Retorna el número de bytes leídos, o -1 si no quedan más	`int read(byte[] b) throws IOException`
Lee repetidamente hasta rellenar el array de bytes completo. Lanza `EOFException` si se acaba el fichero y no se ha podido leer todo	`void readFully(byte[] b) throws IOException`
Lee un double. Lanza `EOFException` si se acaba el fichero	`double readDouble() throws IOException`
Lee un int Lanza `EOFException` si se acaba el fichero	`int readInt() throws IOException`
Lee bytes convirtiéndolos a caracteres hasta encontrar un final de línea	`String readLine() throws IOException`

Principales operaciones

Descripción	Declaración
Escribe un array de bytes	`void write(byte[] b) throws IOException`
Escribe un double	`void writeDouble(double d) throws IOException`
Escribe un int	`void writeInt(int i) throws IOException`
Escribe los caracteres de un string convirtiéndolos primero a bytes (sólo vale para caracteres de 8 bits)	`void writeBytes(String s) throws IOException`
Cerrar el fichero	`void close() throws IOException`

Ejemplo: Tabla de datos persistente

```
import fundamentos.*;
import java.io.*;
/**
 * Clase que contiene los datos de un libro
 */
public class Libro {
  // constantes estáticas
  public static final int maxCaracteresTítulo=100;
  public static final int tamañoEnBytes =
        maxCaracteresTitulo+1  // título + \n
        +Integer.SIZE/8        // publicado
        +Double.SIZE/8         // precio
        +1;                    // tamaño par
  // atributos privados
  private String título;
  private int publicado; // año de publicación
  private double precio; // en euros
```

```
/**
 * Constructor al que se le pasan los datos del
 * libro
 */
public Libro(String titulo, int publicado,
             double precio) {
  // asegurarse de que el titulo no supera
  // maxCaracteresTítulo
  if (titulo.length()>maxCaracteresTítulo) {
    this.titulo=
          titulo.substring(0,maxCaracteresTítulo);
  } else {
    this.titulo=titulo;
  }
  this.publicado=publicado;
  this.precio=precio;
}
```



```
/**
 * Lee de fichero
 */
public static Libro leeDeFichero(
       RandomAccessFile fich) throws IOException {
  // lee los tres datos, por orden
  int publi=fich.readInt();
  double prec=fich.readDouble();
  String tit=fich.readLine().trim();

  // crea y retorna el libro
  return new Libro(tit,publi,prec);
}
```



```
/**
 * Escribe en el fichero
 */
public void escribeEnFichero(
       RandomAccessFile fich) throws IOException {
  // escribe los tres datos, por orden
  fich.writeInt(publicado);
  fich.writeDouble(precio);
  fich.writeBytes(titulo+'\n');
}

métodos observadores, toString, ...

} // clase Libro
```

```
import java.io.*;
import fundamentos.*;
/**
 * Tabla de libros persistente almacenada en
 * un fichero de acceso aleatorio
 */
public class TablaLibros {
  // atributos privados
  private RandomAccessFile fich;

  /**
   *  Constructor al que se le pasa el nombre
   *  del fichero
   */
  public TablaLibros(String nombreFichero)
                  throws FileNotFoundException {
    fich = new RandomAccessFile(nombreFichero,"rw");
  }
```



```
  /**
   * Obtener el elemento de la tabla que esta en
   * "índice"
   */
  public Libro obten(int índice)
                                  throws IOException {
    // posiciona el contador de lectura/escritura
    long pos=indice*Libro.tamañoEnBytes;
    fich.seek(pos);

    // lee y retorna el libro
    return Libro.leeDeFichero(fich);
  }
```



```
  /** Escribir un libro en la posición "índice"
   *  de la tabla */
  public void almacena(int índice, Libro l)
                                   throws IOException {
    // posiciona el contador de lectura/escritura
    long pos=indice*Libro.tamañoEnBytes;
    fich.seek(pos);

    // escribe el libro
    l.escribeEnFichero(fich);
  }

  /** Cerrar la tabla */
  public void cerrar() throws IOException {
    fich.close();
  }
} // clase TablaLibros
```

```
// ejemplo de uso de TablaLibros
TablaLibros t = null;
try {
  t = new TablaLibros("random.dat");
  Libro libro1 = new Libro("Java", 2006, 15.0);
  Libro libro2 = new Libro("1984", 1949, 25.0);
  t.almacena(0,libro1);
  t.almacena(1,libro2);
  Libro l1= t.obten(0);
  Libro l2= t.obten(1);
} finally {
  if (t != null) {
    t.cerrar();
  }
}
```


Uso de objetos en ficheros de acceso aleatorio

Para escribir objetos en ficheros de acceso aleatorio es preciso convertirlos a un array de bytes

- **a este proceso se le llama "*serialización*"**

Para leer el objeto hay que "deserializarlo"

La clase `ByteArrayOutputStream` nos ayuda en este proceso

Tener cuidado pues la serialización incluye objetos a los que se hace referencia

- **esto puede hacer que el tamaño del objeto serializado varíe mucho**
- **para acceso aleatorio es conveniente tener tamaños fijos**

Ejemplo de serialización

```
// serializa un libro y le escribe en un fichero

RandomAccessFile fich = null;
try {
  // abre el fichero de acceso aleatorio
  fich = new RandomAccessFile (nomFich, "rw");
  // pone el puntero al principio
  fich.seek(0L);

  // serializa el libro convirtiéndolo a una
  // secuencia de bytes
  ByteArrayOutputStream bos =
                        new ByteArrayOutputStream();
  ObjectOutputStream out =
                        new ObjectOutputStream(bos);
  out.writeObject(libro);
  out.close();
```

```
    // obtiene los bytes del libro serializado
    byte[] buf = bos.toByteArray();

    // escribe los bytes en el fichero
    fich.write(buf);

  } finally {
    if (fich!=null) {
      fich.close();
    }
  }
```


Ejemplo de deserialización

```
// recupera del fichero un libro serializado

RandomAccessFile fich = null;
try {
  // abre el fichero de acceso aleatorio
  fich = new RandomAccessFile (nomFich, "r");

  // pone el puntero al principio
  fich.seek(0L);

  // Lee un array de bytes del fichero
  byte[] bytes = new byte[(int) fich.length()];
  fich.readFully(bytes);
```



```
    // Deserializa el array de bytes
    ObjectInputStream in = new ObjectInputStream(
                       new ByteArrayInputStream(bytes));
    libro=(Libro) in.readObject();
    in.close();

  } finally {
    if (fich!=null) {
      fich.close();
    }
  }
```

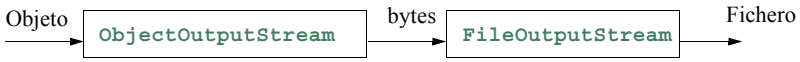
7.9 Resumen

Tipos de acceso a ficheros:

- **lectura de *ficheros binarios***
- **escritura de *ficheros binarios***
- **lectura de *ficheros de texto* línea a línea**
- **escritura de *ficheros de texto***
- **lectura de *ficheros de texto* palabra a palabra (clase `Scanner`)**
- **lectura de *ficheros binarios de acceso aleatorio***
- **escritura de *ficheros binario de acceso aleatorio***

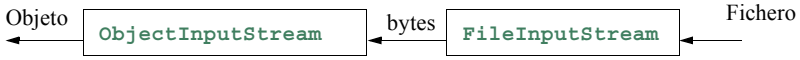
Ficheros Binarios

Escritura



- **Operaciones: `writeBoolean`, `writeDouble`, `writeInt`, `writeObject`**

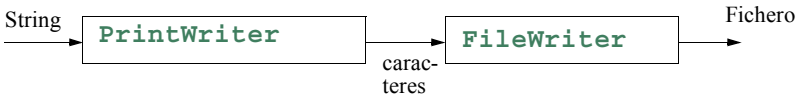
Lectura



- **Operaciones `readBoolean`, `readDouble`, `readInt`, `readObject`**

Ficheros de texto

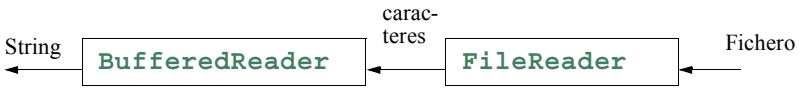
Escritura



- **Operaciones: `print`, `println`, `printf`**

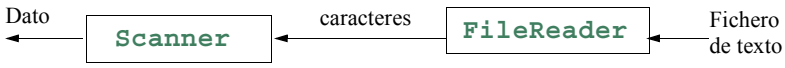
Ficheros de texto (cont.)

Lectura **(línea a línea)**



- **Operaciones:** `readLine`

Lectura **(palabra a palabra)**



- **Operaciones:** `nextDouble`, `hasNextDouble`, `nextInt`, `hasNextInt`, `nextBoolean`, `hasNextBoolean`, `next`, `hasNext`
- **la clase** `Scanner` **también puede usarse como filtro de ficheros binarios o de strings**

Ficheros de acceso aleatorio

Escritura

- **Operaciones:** `write`, `writeDouble`, `writeInt`, `writeBytes`
- **Posicionamiento en el fichero:** `seek`

Lectura

- **Operaciones** `read`, `readFully`, `readDouble`, `readInt`, `readLine`
- **Posicionamiento en el fichero:** `seek`

7.10 Bibliografía

- **King, Kim N. “Java programming: from the beginning”. W. W. Norton & Company, cop. 2000**
- **The Java Turorials. Basic I/O. http://java.sun.com/docs/books/tutorial/essential/io/index.html**
- **Francisco Gutiérrez, Francisco Durán, Ernesto Pimentel. “Programación Orientada a Objetos con Java”. Paraninfo, 2007.**
- **Ken Arnold, James Gosling, David Holmes, “El lenguaje de programación Java”, 3ª edición. Addison-Wesley, 2000.**
- **Eitel, Harvey M. y Deitel, Paul J., “Cómo programar en Java”, quinta edición. Pearson Educación, México, 2004.**