CAPITULO 12

TRABAJAR CON FICHEROS

Todos los programas realizados hasta ahora obtenían los datos necesarios para su ejecución de la entrada estándar y visualizaban los resultados en la salida estándar . Por otra parte, una aplicación podrá retener los datos que manipula en su espacio de memoria, sólo mientras esté en ejecución; es decir, cualquier dato introducido se perderá cuando la aplicación finalice.

Por ejemplo, si hemos realizado un programa con la intención de construir una agenda, lo ejecutamos y almacenamos los datos nombre,apellidos y teléfono de cada uno de los componentes de la agenda en una matriz , los datos estarán disponibles mientras el programa esté en ejecución. Si finalizamos la ejecución del programa y lo ejecutamos de nuevo, tendremos que volver a introducir de nuevo todos los datos.

La solución para hacer que los datos persistan de una ejecución para otra es almacenarlos en un fichero en el disco en vez de en una matriz en memoria. Entonces, cada vez que se ejecute la aplicación que trabaja con esos datos, podrá leer del fichero los que necesite y manipularlos . Nosotros procederemos de forma análoga en muchos aspectos de la vida ordinaria; almacenamos los datos en fichas y guardamos el conjunto de fichas en lo que generalmente denominamos fichero o archivo.

Desde el punto de vista informático, un fichero o archivo es una colección de información que almacenamos en un soporte magnético para poderla manipular en cualquier momento. Esta información se almacena como un conjunto de registros, conteniendo todos ellos, generalmente, los mismos campos. Cada campo almacena un dato de un tipo predefinido o de un tipo definido por el usuario. El registro más simple estaría formado por un carácter.

Por ejemplo, si quisiéramos almacenar en un fichero los datos relativos a la agenda de teléfonos a los que nos hemos referido anteriormente, podríamos diseñar cada registro con los campos nombre, dirección y teléfono.

Cada campo almacenará el dato correspondiente. El conjunto de campos descritos forma lo que hemos denominado registro, y el conjunto de todos los registros forman un fichero que almacenaremos, por ejemplo en el disco bajo un nombre.

Por lo tanto, para manipular un fichero que identificamos por un nombre, son tres las operaciones que tenemos que realizar: abrir el fichero,escribir o leer los registros del fichero y cerrar el fichero.

En programación orientada a objetos,hablaremos de objetos más que de registros , y de sus atributos más que de campos.

Podemos agrupar los ficheros en dos tipos: ficheros de la aplicación(son los ficheros .java, .class, etc. que forman la aplicación) y ficheros de datos (son los que proveen de datos a la aplicación). A su vez, Java ofrece dos tipos diferentes de acceso a los ficheros de datos: secuencial y aleatorio.

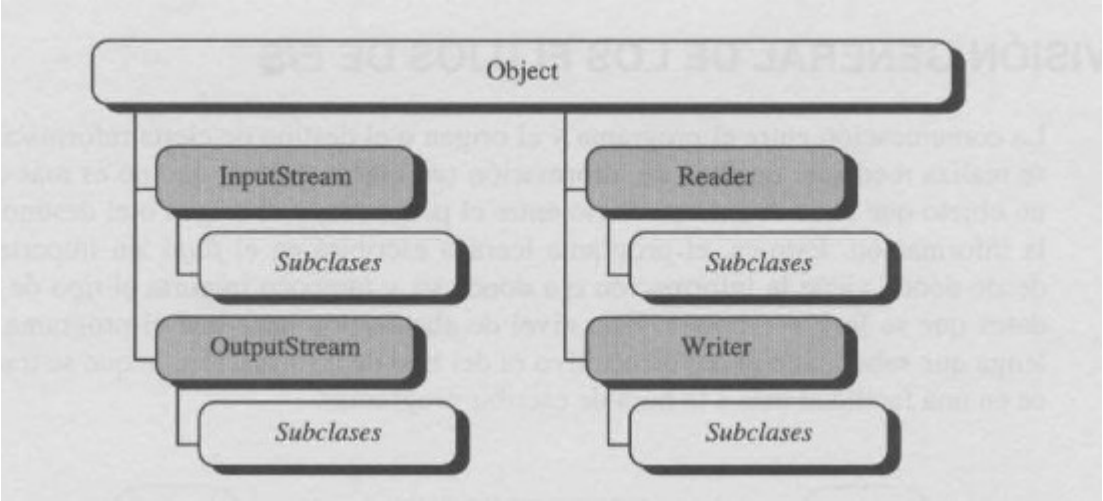
Para dar soporte al trabajo con ficheros, la biblioteca de Java proporciona varias clases de entrada/salida (E/S) que permiten leer y escribir datos a, y desde, ficheros y dispositivos.

VISIÓN GENERAL DE LOS FLUJOS DE E/S

La comunicación entre el programa y el origen o el destino de cierta información, se realiza mediante un flujo de información que no es más que un objeto que hace de intermediario entre el programa, y el origen o el destino de la información. Esto es, el `programa leerá o escribirá en el flujo sin importarle desde dónde viene la información o a dónde va y tampoco importa el tipo de los datos que se leen o escriben. Este nivel de abstracción hace que el programa no tenga que saber nada ni del dispositivo ni del tipo de información, lo que se traduce en una facilidad más a la hora de escribir programas.

Entonces, para que un programa pueda obtener información desde un fichero tiene que abrir un flujo y leer la información en él almacenada. Análogamente, para que un programa puede enviar información a un fichero tiene que abrir un flujo y escribir la información en el mismo.

El paquete java.io de la biblioteca estándar de Java, contiene una colección de clases que soportan estos algoritmos para leer y escribir. Estas clases se dividen en dos grupos distintos, según se muestra en la figura siguiente. El grupo de la izquierda ha sido diseñado para trabajar con datos de tipo byte y el de la derecha con datos de tipo char. Ambos grupos presentan clases análogas que tienen interfaces casi idénticas, por lo que se utilizan de la misma manera.



Java implementa la jerarquía de clases para la E/S sin incluir demasiadas capacidades dentro de una clase porque rara vez se necesitan muchas de ellas al mismo tiempo. En cambio, sí se pueden obtener todas esas capacidades superponiendo una clase sobre otra.

Sin embargo, a menudo es más conveniente agrupar las clases según su finalidad en vez de por el tipo de datos que leen o escriben. Desde este punto de vista distinguimos flujos que simplemente permiten leer y escribir datos y flujos que, además , procesan la información leída o escrita.

FLUJOS QUE NO PROCESAN LOS DATOS DE E/S

La tabla siguiente lista las subclases que permiten definir flujos para leer o escribir información en un medio sin realizan ningún proceso añadido:

Medio	*Flujo de caracteres*	*Flujo de bytes*
Memoria	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
	StringReader StringWriter	StringBufferInputStream
Fichero	FileReader FileWriter	FileInputStream FileOutputStream
Tubería	PipedReader PipedWriter	PipedInputStream PipedOutputStream

Un programa que cree un flujo de alguna de estas clases podrá leer o escribir información en algunos de los medio especificados: una matriz en memoria, un fichero en el disco o una tubería. Una tubería es un flujo que permite comunicar dos subprocesos para transferencia de información entre uno y otro. Veremos esto con más detalle en el capítulo dedicado a hilos.

FLUJOS QUE PROCESAN LOS DATOS DE E/S

La tabla siguiente lista las subclases que permiten definir flujos para leer o escribir información en un medio,además de realizar alguna operación como añadir un buffer,un filtro,realizar una conversión,etc:

Operación	*Flujo de caracteres*	*Flujo de bytes*
Establecer un buffer	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Establecer un filtro	FilterReader FilterWriter	FilterInputStream FilterOutputStream
Conversión (bytes - cars.)	InputStreamReader OutputStreamWriter	
Concatenación		SequenceInputStream
Seriación		ObjectInputStream ObjectOutputStream
Conversión (datos)		DataInputStream DataOutputStream
Contar	LineNumberReader	LineNumberInputStream
Mirar anticipa-damente	PushbackReader	PushbackInputStream
Escribir	PrintWriter	PrintStream

Un programa que cree un flujo de alguna de estas clases podrá leer o escribir información además de ejecutar la operación para la que ha sido diseñado. Por ejemplo, un flujo de la clase PushbackReader (derivada de FilterReader que a su vez se deriva de Reader) es útil cuando, por ejemplo, un analizador necesita mirar el siguiente carácter en la entrada con el fin de determinar qué hacer a continuación;para ello, el analizador leerá el carácter y después lo devolverá a la entrada para que pueda ser leído por el código que tenga que ejecutarse a continuación.

Alguno de los métodos que proporciona la clase PushbackReader son:

Método	*Significado*
close	Cierra el flujo.
read	Lee un único carácter, o bien una matriz de caracteres.
unread	Devuelve a la entrada un único carácter, o bien todo o parte de una matriz de caracteres.
ready	Devuelve **true** si se puede leer del flujo porque hay caracteres disponibles; en otro caso devuelve **false**. Este método, heredado de la clase **Reader**, permite realizar operaciones análogas al método **available** de la clase **InputStream**.

ABRIENDO FICHEROS PARA ACCESO SECUENCIAL

El tipo de acceso más simple a un fichero de datos es el secuencial. Un fichero abierto para acceso secuencial es un fichero que puede almacenar registros de cualquier longitud, incluso de un sólo byte. Cuando la información se escribe registro a registro, éstos son colocados uno a continuación de otro, y cuando se lee, se empieza por el primer registro, y se continúa al siguiente hasta alcanzar el final.

Este tipo de acceso generalmente se utiliza con ficheros de texto en los que se escribe toda la información desde el principio hasta el final y se lee de la misma forma. En cambio, los ficheros de texto no son los más apropiados para almacenar grandes series de números, por que cada número es almacenado como una secuencia de bytes en lugar de los cuatro requeridos par aun entero. De ahí que a continuación se expongan distintos tipos de flujos: de bytes y de caracteres para el tratamientos de texto, y de dato spara el tratamiento de números.

FLUJOS DE BYTES

Los datos pueden ser escritos o leídos de un fichero byte a byte utilizando flujos de las clases FileOutputStream y FileInputStream.

FileOutputStream

Un flujo de la clase FileOutputStream permite escribir bytes en un fichero. Además de los métodos que esta clase hereda de OutputStream,la clase proporciona los constructores siguientes

```
FileOutputStream(String nombre)
FileOutputStream(String nombre, boolean añadir)
FileOutputStream(File fichero)
```

El primer constructor abre un flujo de salida hacia el fichero especificado por nombre, mientras que el segundo hace lo mismo,pero con la posibilidad de añadir datos a un fichero existente (añadir = true); el tercero lo hace a partir de un objeto File. Un ejemplo aclarará los conceptos expuestos.

Clase File

El ejemplo anterior utiliza un String para referirse al fichero, pero también podría haber utilizado un objeto de la clase File . Un objeto de esta clase representa el nombre de un fichero o de un directoria que puede existir en el sistema de ficheros de la máquina; por lo tanto,sus métodos permitirán interrogar al sistema sobre todas las características de ese fichero o directorio.

Métodos de la clase File:

Método	*Significado*
getName	Devuelve el nombre del fichero especificado por el objeto **File** que recibe este mensaje.
getParent	Devuelve el directorio padre.
getPath	Devuelve la ruta relativa del fichero.
getAbsolutePath	Devuelve la ruta absoluta del fichero.
exists	Devuelve **true** si el nombre especificado por el objeto **File** que recibe este mensaje existe.
canWriter	Devuelve **true** si se puede escribir en el fichero o directorio especificado por el objeto **File**.

Método	*Significado*
canRead	Devuelve **true** si se puede leer desde el fichero o directorio especificado por el objeto **File**.
isFile	Devuelve **true** si se trata de un fichero válido.
isDirectory	Devuelve **true** si se trata de un directorio válido.
isHidden	Devuelve **true** si se trata de un fichero o directorio oculto.
length	Devuelve el tamaño del fichero (cuando se trate de un directorio, el valor devuelto es cero).
list	Devuelve una matriz de objetos **String** que almacena los nombres de los ficheros y directorios que hay en el directorio especificado por el objeto **File**.
mkdir	Crea el directorio especificado por el objeto **File**.
mkdirs	Crea el directorio especificado por el objeto **File** incluyendo los directorios que no existan en la ruta especificada.
delete	Borra el fichero o directorio especificado por el objeto **File**. Cuando se trate de un directorio, éste debe de estar vacío.
deleteOnExit	Igual que **delete**, pero cuando la máquina virtual termina.
createTempFile	Crea el fichero vacío especificado por los argumentos pasados, en el directorio temporal del sistema.
renameTo	Renombra el fichero especificado por el objeto **File** que recibe este mensaje, con el nombre especificado por el objeto **File** pasado como argumento.
setReadOnly	Marcar el fichero o directorio especificado por el objeto **File** de sólo lectura.
toString	Devuelve la ruta especificada cuando se creó el objeto **File**.

FLUJO DE CARACTERES

Una vez que sabemos trabajar con flujos de bytes, hacerlo con flujos de caracteres es prácticamente lo mismo. Esto nos será util cuando necesitamos trabajar con texto representado por un conjunto de caracteres ASCII o Unicode. Las clases que definen estos flujos son subclases de Reader, como FileWriter y FileReader.

FILEWRITER

Un flujo de la clase FileWriter permite escribir caracteres (char) en un fichero.

FILEREADER

Un flujo de la clase FileReader permite leer caracteres desde un fichero.

FLUJO DE DATOS

Seguramente, en alguna ocasión desearemos escribir en un fichero datos de tipos primitivos(boolean, byte, double, float, long, int y short) para posteriormente recuperarlos como tal. Para estos casos, el paquete java.io proporciona las clases DataInputStream y DataOutputStream, las cuales permiten leer y escribir, respectivamente,datos de cualquier tipo

primitivo. Entonces, ¿ por qué no se han analizado previamente? Pues, simplemente por que no pueden utilizarse con los dispositivos ASCII de E/S estándar. Un flujo DataInputStream sólo puede leer datos almacenados en un fichero a través de un flujo DataOutputStream.

Observa que los flujos de estas clases actúan como filtros; esto es, los datos obtenidos del origen o enviados al destino son transformados mediante alguna operación; en este caso, sufren una conversación a un formato portable cuando son almacenados y viceversa cuando son recuperados. El procedimiento para utilizar un filtro es básicamente así:

- Se crea un flujo asociado con un origen o destino de los datos.
- Se asocia un filtro con el flujo anterior.
- Finalmente, el programa leerá o escribirá datos a través de ese filtro.

DATAOUTPUTSTREAM

Un flujo de la clase DataOutputStream, derivada indirectamente de OutputStream, permite a una aplicación escribir en un flujo de salida subordinado,datos de cualquier tipo primitivo.

Todo los métodos proporcionados por esta clase están definidos en la interfaz DataOutput implementada por la misma.

Los métodos más utilizados de esta clase se resumen en la tabla siguiente:

Método	*Descripción*
writeBoolean	Escribe un valor de tipo **boolean**.
writeByte	Escribe un valor de tipo **byte**.
writeBytes	Escribe un **String** como una secuencia de bytes.
writeChar	Escribe un valor de tipo **char**.
writeChars	Escribe un **String** como una secuencia de caracteres.
writeShort	Escribe un valor de tipo **short**.
writeInt	Escribe un valor de tipo **int**.
writeLong	Escribe un valor de tipo **long**.
writeFloat	Escribe un valor de tipo **float**.
writeDouble	Escribe un valor de tipo **double**.
writeUFT	Escribe una cadena de caracteres en formato UTF-8; los dos primeros bytes especifican el número de bytes de datos escritos a continuación.

DataInputStream

Un flujo de la clase DataInputStream, derivada indirectamente de InputStream, permite a una aplicación leer de un flujo de entrada subordinado,datos, de cualquier tipo primitivo escritos por un flujo de la clase DataOutputStream.

Todos los métodos proporcionados por esta clase están definidos en la interfaz DataInput implementados por la misma.

Los métodos más utilizados de esta clase se resumen en la tabla siguiente:

Método	*Descripción*
readBoolean	Devuelve un valor de tipo **boolean**.
readByte	Devuelve un valor de tipo **byte**.
readShort	Devuelve un valor de tipo **short**.
readChar	Devuelve un valor de tipo **char**.
readInt	Devuelve un valor de tipo **int**.
readLong	Devuelve un valor de tipo **long**.
readFloat	Devuelve un valor de tipo **float**.
readDouble	Devuelve un valor de tipo **double**.
readUFT	Devuelve una cadena de caracteres en formato UTF-8; los dos primeros bytes especifican el número de bytes de datos que serán leídos a continuación.

UN EJEMPLO DE ACCESO SECUENCIAL

Después de la teoría expuesta ahora acerca del trabajo con ficheros, habrá observado que la metodología de trabajo se repite. Es decir , para escribir datos en fichero:

- Definimos un flujo hacia el fichero en el que deseamos escribir datos.
- Leemos los datos del dispositivo de entrada o de otro fichero y los escribimos en nuestro fichero. Este proceso se hace normalmente registro a registro. Para ello, utilizaremos los métodos proporcionados por la interfaz del flujo.
- Cerramos el flujo.

Para leer datos de un fichero existente:

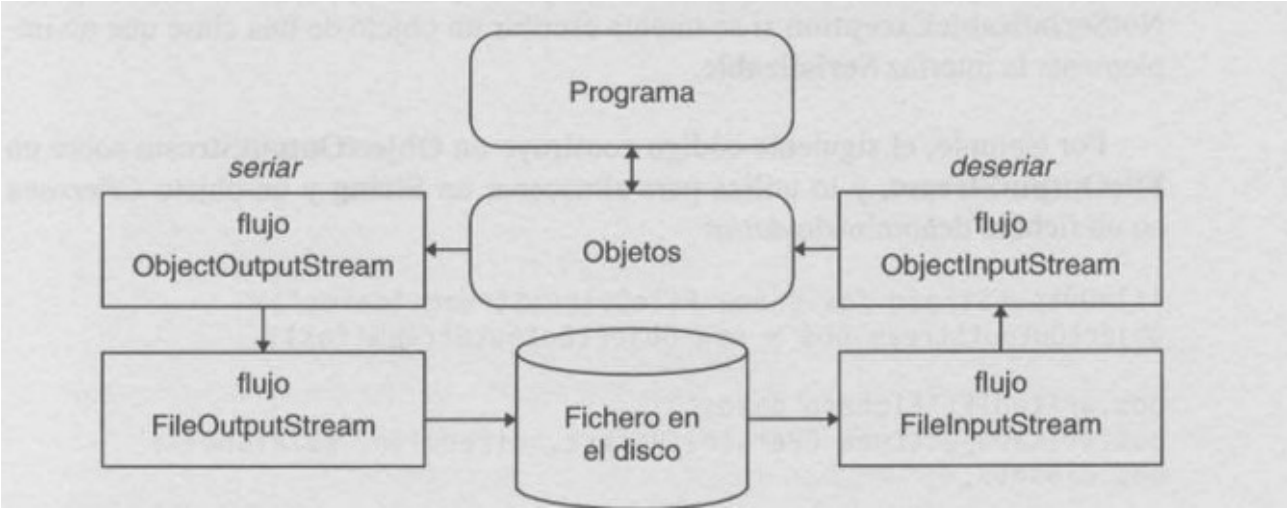
- Abrimos un flujo desde el fichero del cual queremos leer los datos.
- Leemos los datos del fichero y los almacenamos en variable de nuestro programa con el fin de trabajar con ellos. Este proceso se hace normalmente registro a registro. Para ello,utilizaremos los métodos proporcionados por la interfaz del flujo.
- Cerramos el flujo.

Esto pone de manifiesto que un fichero no es más que un medio permanente de almacenamiento de datos , dejando esos datos disponibles para cualquier programa que necesite manipularlos. Lógicamente, los datos serán recuperados del fichero con el mismo formato con el que fueron escritos, de lo contrario los resultados serán inesperados. Es decir, si en el ejercicio siguiente los datos son guardados en el orden: una cadena,otra cadena y un long, tendrán que ser recuperados en este orden y con este mismo formato. Sería un error recuperar primero un long después una cadena y finalmente la otra cadena, o recuperar primero una cadena, después un float la otra cadena; etc.

SERIACIÓN DE OBJETOS

En un desarrollo orientado a objetos debemos pensar en objetos; por lo tanto, ese grupo de datos al que nos hemos referido no lo trataremos aisladamente, más bien se corresponderá con los atributos , de un objeto,por lo que nos conducirá a escribir y leer objetos a y desde un fichero.

Normalmente la operación de enviar una serie de objetos a un fichero en disco para hacerlos persistentes recibe el nombre de **seriación**, y la operación de leer o recuperar su estado del fichero para reconstruirlos en memoria recibe el nombre de **deseriación**. Para realizar estas operaciones de unna forma automática, el paquete java.io proporciona las clases ObjectOutputStream y ObjectInputStream. Ambas clases dan lugar a flujos que procesan sus datos; en este caso, se trata de convertir el estado de un objeto (los atributos excepto las variables estáticas), incluyendo la clase del objeto y el prototipo de la misma, en una secuencia de bytes y viceversa. Por esta razón lps flujos que canalicen esos bytes a y desde el fichero.



Para poder seriar los objetos de una clase, ésta debe de implementar la interfaz **Serializable.** Se trata de una interfaz vacía; esto es, sin ningún método; su propósito es simplemente identificar clases cuyos objetos se pueden seriar.

ESCRIBIR OBJETOS EN UN FICHERO

Un flujo de la clase ObjectOutputStream permite enviar datos de tipos primitivos y objetos hacia un flujo OutputStream o derivado; concretamente, cuando se trate de almacenarlos en un fichero, utilizaremos un flujo FileOutputStream. Posteriormente, esos objetos podrán ser reconstruidos a través de un flujo Object-InputStream.

Para escribir un objeto en un flujo ObjectOutputStream utilizaremos el método writeObject. Los objetos pueden incluir Strings y matrices, y el almacenamiento de los mismos puede combinarse con datos de tipos primitivos, ya que esta clase implementa la interfaz DataOutput. Este método lanzará la excepción NotSerializableException si se intenta escribir un objeto de una clase que no implementa la interfaz Serializable.

LEER OBJETOS DESDE UN FICHERO

Un flujo de la clase ObjectInputStream permite recuperar datos de tipos primitivos y objetos desde un flujo InputStream o derivado; concretamente,cuando se trate de datos de tipos primitivos y objetos almacenados en un fichero, utilizaremos un flujo FileInputStream. La clase ObjectInputStream implementa la interfaz DataInput para permitir leer también datos de tipos primitivos.

Para leer un objeto desde un flujo ObjectInputStream utilizaremos el método readObject. Si se almacenaron objetos y datos de tipos primitivos, deben ser recuperados en el mismo orden.

SERIAR OBJETOS QUE REFERENCIAN A OBJETOS

Cuando en un fichero se escribe un objeto que hace referencia a otros objetos, entonces todos los objetos accesibles desde el primero deben de ser escritos en el mismo proceso para mantener así la relación existente entre todos ellos. Este proceso es llevado a cabo automáticamente por el método writeObject, que escribe el objeto especificado, recorriendo sus referencias a otros objetos recursivamente, escribiendo así todos ellos.

Análogamente , si el objeto recuperado del flujo por el método readObject hace referencia a otros objetos, readObject recorrerça sus referencias a otros objetos recursivamente, para recuperar todos ellos manteniendo la relacion que existía entre ellos cuando fueron escritos.

ABRIENDO FICHEROS PARA ACCESO ALEATORIO

Hasta este punto, hemos trabajado con ficheros de acuerdo con el siguiente esquema:abrir el fichero,leer o escribir hasta el final del mismo, y cerrar el fichero.
Pero no hemos leído o escruto a partir de una determinada posición dentro del fichero.Esto es particularmente importante cuando necesitamos modificar algunos de los valores contenidos en el fichero o cuando necesitemos extraer una parte concreta dentro del fichero.

El paquete java.io contiene la clase RandomAccessFile la que proporciona las capacidades que permiten este tipo de acceso directo. Además, un flujo de esta clase permite realizar tanto operación de lectura como de escritura sobre el fichero vinculado con el mismo. Esta clase se deriva directamente de Object, e implementa las interfaces DataInput y DataOutput.

Un fichero accedido aleatoriamente es comparable a una matriz. En una matriz para acceder a uno de sus elementos utilizamos un índice. En un fichero accedido aleatoriamente el índice es sustituido por un puntero de lectura o escritura(L/E). Dicho puntero es situado automáticamente al principio del fichero cuando este se abre para leer y/o escribir. Por lo tanto, una operación de lectura o de escritura comienza en la posición donde esté el puntero dentro del fichero; finalmente, su posición coincidirá justo a continuación del último byte leído o escrito.

LA CLASE RandomAccessFile

Un lujo de esta clase permite acceder directamente a cualquier posición dentro del fichero vinculado con él.

La clase RandomAccessFile proporciona dos constructores:

```
RandomAccessFile(String nombre-fichero, String modo)
RandomAccessFile(File objeto-File, String modo)
```

El primer constructor abre un flujo vinculado con el fichero especificado por *nombre-fichero*, mientras que el segundo hace lo mismo, pero a partir de un objeto **File**. El argumento *modo* puede ser:

Modo	*Significado*
r	*read*. Sólo se permiten realizar operaciones de lectura.
rw	*read/write*. Se pueden realizar operaciones de lectura y de escritura sobre el fichero.

La clase RandomAccessFile provee, además de los métodos de las interfaces DataInput y DataOutput,los métodos getFilePointer,length y seek que se definen de la forma siguiente:

```
public long getFilePointer() throws IOException
```

Este método devuelve la posición actual en bytes del puntero de L/E en el fichero.Piense que en el puntero de L/E análogamente a como lo hace cuando piensa en el indice de una matriz. Este puntero marca siempre la posición donde se iniciará la siguiente operación de lectura o de escritura en el fichero.

```
public long length() throws IOException
```

Este otro método devuelve la longitud del fichero en bytes.

```
public void seek(long pos) throws IOException
```

Y este otro método, mueve el puntero de L/E a una nueva localización desplazada *pos* bytes del principio del fichero. No se permiten desplazamientos ne-

gativos. El desplazamiento requerido puede ir más allá del final del fichero; esta acción no cambia la longitud del fichero; la longitud del fichero sólo cambiará si a continuación, realizamos una operación de escritura.

Según lo expuesto, las dos líneas de código siguientes sitúan el puntero de L/E, la primera *desp* bytes antes del final del fichero y la segunda *desp* bytes después de la posición actual.

```
listaTeléfonos.seek(listaTeléfonos.length() - desp);
listaTeléfonos.seek(listaTeléfonos.getFilePointer() + desp);
```

Con esta clase no tenemos posibilidad de seriar objetos. Los datos deben guardarse uno a uno utilizando el método adecuado de la clase según su tipo. Por ejemplo, las siguientes líneas de código escriben en el fichero "datos" a partir de la posición *desp*, los atributos *nombre*, *dirección* y *teléfono* relativos a un objeto *CPersona*:

```
CPersona objeto;
// ...
RandomAccessFile fes = new RandomAccessFile("datos", "rw");
fes.seek(desp);
fes.writeUTF(objeto.obtenerNombre());
fes.writeUTF(objeto.obtenerDirección());
fes.writeLong(objeto.obtenerTeléfono());
```

Si para nuestros propósitos, pensamos en los atributos *nombre*, *dirección* y *teléfono* como si de un registro se tratara, ¿cuál es el tamaño en bytes de ese registro? Si escribimos más registros ¿todos tienen el mismo tamaño? Evidentemente no; el tamaño de cada registro dependerá del número de caracteres almacenados en los **String** *nombre* y *dirección* (*teléfono* es un dato de tamaño fijo, 8 bytes, puesto que se trata de un **long**) ¿A cuento de qué viene esta exposición?

Al principio de este apartado dijimos que el acceso aleatorio a ficheros es particularmente importante cuando necesitemos modificar algunos de los valores contenidos en el fichero, o bien cuando necesitemos extraer una parte concreta dentro del fichero. Esto puede resultar bastante complicado si las unidades de grabación que hemos denominado registros no son todas iguales, ya que intervienen los factores de: posición donde comienza un registro y longitud del registro. Tenga presente que cuando necesite reemplazar el registro *n* de un fichero por otro, no debe sobrepasarse el número de bytes que actualmente tiene. Todo esto es viable llevando la cuenta en una matriz de la posición de inicio de cada uno de los registros y de cada uno de los campos si fuera preciso (esta información se almacenaría en un fichero índice para su utilización posterior), pero resulta mucho más fácil si todos los registros tienen la misma longitud.

UTILIZACIÓN DE DISPOSITIVOS ESTÁNDAR

La salida de un programa puede también ser enviada a un dispositivo de salida que no sea el disco o la pantalla; por ejemplo, a una impresora conectada al puerto paralelo. Como Java no tiene definido un flujo estándar para el puerto paralelo, la solución es definir uno y vincularlo a dicho dispositivo.

Una forma de realizar lo expuesto es crear un flujo hacia el dispositivo *LPT1*, *LPT2*, o *PRN* y escribir en ese flujo (los nombres indicados son los establecidos por Windows para nombrar a la impresora; en UNIX la primera impresora tiene asociado el nombre */dev/lp0*, la segunda */dev/lp1*, etc.). Las siguientes líneas de código muestran cómo realizar esto:

```
// Crear un flujo hacia la impresora
FileWriter flujoS = new FileWriter("LPT1");
flujoS.write("Esta línea se escribe en la impresora\r\n");
flujoS.write("\r\n"); // saltar una línea
Long n = new Long(123456789);
flujoS.write("Valor: " + n.toString() + "\r\n");
flujoS.write("\f");   // saltar a la página siguiente
flujoS.close();       // cerrar el flujo
```

El flujo creado es de la clase **FileWriter**, pero se podría haber creado de otra clase que permita definir flujos de salida, como **FileOutputStream**, **DataOutputStream, RandomAccessFile**, etc. Se puede observar que para obtener datos impresos legibles se envían cadenas de caracteres a la impresora, ya que se trata de un dispositivo ASCII. En general podemos enviar datos de tipo **char** o **byte**.