

- **Acceso secuencial:** los datos o registros se leen y se escriben en orden, del mismo modo que se hace en una antigua cinta de vídeo. Si se quiere acceder a un dato o un registro que está hacia la mitad del fichero es necesario leer antes todos los anteriores. La escritura de datos se hará a partir del último dato escrito, no es posible hacer inserciones entre los datos que ya hay escritos.
- **Acceso directo o aleatorio:** permite acceder directamente a un dato o registro sin necesidad de leer los anteriores y se puede acceder a la información en cualquier orden. Los datos están almacenados en registros de tamaño conocido, nos podemos mover de un registro a otro de forma aleatoria para leerlos o modificarlos.

En Java el acceso secuencial más común en ficheros puede ser binario o a caracteres. Para el acceso binario: se usan las clases **FileInputStream** y **FileOutputStream**; para el acceso a caracteres (texto) se usan las clases **FileReader** y **FileWriter**. En el acceso aleatorio se utiliza la clase **RandomAccessFile**.

## 1.5. OPERACIONES SOBRE FICHEROS

Las operaciones básicas que se realizan sobre cualquier fichero independientemente de la forma de acceso al mismo son las siguientes:

- **Creación del fichero.** El fichero se crea en el disco con un nombre que después se debe utilizar para acceder a él. La creación es un proceso que se realiza una vez.
- **Apertura del fichero.** Para que un programa pueda operar con un fichero, la primera operación que tiene que realizar es la apertura del mismo. El programa utilizará algún método para identificar el fichero con el que quiere trabajar, por ejemplo, asignar a una variable el descriptor del fichero.
- **Cierre del fichero.** El fichero se debe cerrar cuando el programa no lo vaya a utilizar. Normalmente suele ser la última instrucción del programa.
- **Lectura de los datos del fichero.** Este proceso consiste en transferir información del fichero a la memoria principal, normalmente a través de alguna variable o variables de nuestro programa en las que se depositarán los datos extraídos del fichero.
- **Escritura de datos en el fichero.** En este caso el proceso consiste en transferir información de la memoria (por medio de las variables del programa) al fichero.

Normalmente las operaciones típicas que se realizan sobre un fichero una vez abierto son las siguientes:

- **Altas:** consiste en añadir un nuevo registro al fichero.
- **Bajas:** consiste en eliminar del fichero un registro ya existente. La eliminación puede ser lógica, cambiando el valor de algún campo del registro que usemos para controlar dicha situación; o física, eliminando físicamente el registro del fichero. El borrado físico consiste muchas veces en reescribir de nuevo el fichero en otro fichero sin los datos que se desean eliminar y luego renombrarlo al fichero original.
- **Modificaciones:** consiste en cambiar parte del contenido de un registro. Antes de realizar la modificación será necesario localizar el registro a modificar dentro del fichero; y una vez localizado se realizan los cambios y se reescribe el registro.
- **Consultas:** consiste en buscar en el fichero un registro determinado.

### 1.5.1. Operaciones sobre ficheros secuenciales

En los ficheros secuenciales los registros se insertan en orden cronológico, es decir, un registro se inserta a continuación del último insertado. Si hay que añadir nuevos registros estos se añaden a partir del final del fichero.

Veamos cómo se realizan las operaciones típicas:

- **Consultas:** para consultar un determinado registro es necesario empezar la lectura desde el primer registro, y continuar leyendo secuencialmente hasta localizar el registro buscado. Por ejemplo, si el registro a buscar es el 90 dentro del fichero, será necesario leer secuencialmente los 89 que le preceden.
- **Altas:** en un fichero secuencial las altas se realizan al final del último registro insertado, es decir, solo se permite añadir datos al final del fichero.
- **Bajas:** para dar de baja un registro de un fichero es necesario leer todos los registros uno a uno y escribirlos en un fichero auxiliar, salvo el que deseamos dar de baja. Una vez reescritos hemos de borrar el fichero inicial y renombrar el fichero auxiliar dándole el nombre del fichero original.
- **Modificaciones:** consiste en localizar el registro a modificar, efectuar la modificación y reescribir el fichero inicial en otro fichero auxiliar que incluya el registro modificado. El proceso es similar a las bajas.

Los ficheros secuenciales se usan típicamente en aplicaciones de proceso por lotes como, por ejemplo, en el respaldo de los datos o backup, y son óptimos en dichas aplicaciones si se procesan todos los registros. La **ventaja** de estos ficheros es la rápida capacidad de acceso al siguiente registro (son rápidos cuando se accede a los registros de forma secuencial) y que aprovechan mejor la utilización del espacio. También son sencillos de usar y aplicar.

La **desventaja** es que no se puede acceder directamente a un registro determinado, hay que leer antes todos los anteriores; es decir, no soporta acceso aleatorio. Otra desventaja es el proceso de actualización, la mayoría de los ficheros secuenciales no pueden ser actualizados, habrá que reescribirlos totalmente. Para las aplicaciones interactivas que incluyen peticiones o actualizaciones de registros individuales, los ficheros secuenciales ofrecen un rendimiento pobre.

### 1.5.2. Operaciones sobre ficheros aleatorios

Las operaciones en ficheros aleatorios son las vistas anteriormente, pero teniendo en cuenta que para acceder a un registro hay que localizar la posición o dirección donde se encuentra. Los ficheros de acceso aleatorio en disco manipulan direcciones relativas en lugar de direcciones absolutas (número de pista y número de sector en el disco), lo que hace al programa independiente de la dirección absoluta del fichero en el disco.

Normalmente para posicionarnos en un registro es necesario aplicar una función de conversión, que usualmente tiene que ver con el tamaño del registro y con la clave del mismo (la clave es el campo o campos que identifica de forma única a un registro). Por ejemplo, disponemos de un fichero de empleados con tres campos: identificador, apellido y salario. Usamos el identificador como campo clave del mismo, y le damos el valor 1 para el primer empleado, 2 para el segundo empleado y así sucesivamente; entonces, para localizar al empleado con identificador X necesitamos acceder a la posición  $tamaño*(X-1)$  para acceder a los datos de dicho empleado.

Puede ocurrir que al aplicar la función al campo clave nos devuelva una posición ocupada por otro registro, en ese caso, habría que buscar una nueva posición libre en el fichero para ubicar dicho registro o utilizar una **zona de excedentes** dentro del mismo para ir ubicando estos registros.

Veamos cómo se realizan las operaciones típicas:

- **Consultas:** para consultar un determinado registro necesitamos saber su clave, aplicar la función de conversión a la clave para obtener la dirección y leer el registro ubicado en esa posición. Habría que comprobar si el registro buscado está en esta posición, si no está, se buscaría en la zona de excedentes.
- **Altas:** para insertar un registro necesitamos saber su clave, aplicar la función de conversión a la clave para obtener la dirección y escribir el registro en la posición devuelta. Si la posición está ocupada por otro registro, en ese caso el registro se insertaría en la zona de excedentes.
- **Bajas:** las bajas suelen realizarse de forma lógica, es decir, se suele utilizar un campo del registro a modo de switch que tenga el valor 1 cuando el registro existe y le damos el valor 0 para darle de baja, físicamente el registro no desaparece del disco. Habría que localizar el registro a dar de baja a partir de su campo clave y reescribir en este campo el valor 0.
- **Modificaciones:** para modificar un registro hay que localizarlo, necesitamos saber su clave para aplicar la función de conversión y así obtener la dirección, modificar los datos que nos interesen y reescribir el registro en esa posición.

Una de las principales **ventajas** de los ficheros aleatorios es el rápido acceso a una posición determinada para leer o escribir un registro. El gran **inconveniente** es establecer la relación entre la posición que ocupa el registro y su contenido; ya que a veces al aplicar la función de conversión para obtener la posición se obtienen posiciones ocupadas y hay que recurrir a la zona de excedentes. Otro inconveniente es que se puede desaprovechar parte del espacio destinado al fichero, ya que se pueden producir huecos (posiciones no ocupadas) entre un registro y otro.

## 1.6. CLASES PARA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS

En Java podemos utilizar dos tipos de ficheros: de texto o binarios; y el acceso a los mismos se puede realizar de forma secuencial o aleatoria. Los ficheros de texto están compuestos de caracteres legibles, mientras que los binarios pueden almacenar cualquier tipo de dato (*int*, *float*, *boolean*, etc.).

### 1.6.1. Ficheros de texto

Los ficheros de texto, los que normalmente se generan con un editor, almacenan caracteres alfanuméricos en un formato estándar (ASCII, UNICODE, UTF8, etc.) Para trabajar con ellos usaremos las clases **FileReader** para leer caracteres y **FileWriter** para escribir los caracteres en el fichero. Cuando trabajamos con ficheros, cada vez que leemos o escribimos en uno debemos hacerlo dentro de un manejador de excepciones **try-catch**. Al usar la clase **FileReader** se puede generar la excepción **FileNotFoundException** (porque el nombre del fichero no exista o no sea válido) y al usar la clase **FileWriter** la excepción **IOException** (el disco está lleno o protegido contra escritura).

Los métodos que proporciona la clase **FileReader** para lectura son los siguientes, estos métodos devuelven el número de caracteres leídos o -1 si se ha llegado al final del fichero:

Método	Función
int read()	Lee un carácter y lo devuelve
int read(char[] buf)	Lee hasta <i>buf.length</i> caracteres de datos de una matriz de caracteres ( <i>buf</i> ). Los caracteres leídos del fichero se van almacenando en <i>buf</i>
int read(char[] buf, int desplazamiento, int n)	Lee hasta <i>n</i> caracteres de datos de la matriz <i>buf</i> comenzando por <i>buf[desplazamiento]</i> y devuelve el número leído de caracteres

En un programa Java para crear o abrir un fichero se invoca a la clase **File** y a continuación se crea el flujo de entrada hacia el fichero con la clase **FileReader**. Después se realizan las operaciones de lectura o escritura y cuando terminemos de usarlo lo cerraremos mediante el método **close()**.

El siguiente ejemplo lee cada uno de los caracteres del fichero de texto de nombre *LeerFichTexto.java* (localizado en la carpeta *C:\EJERCICIOS\UNI1*) y los muestra en pantalla, los métodos **read()** pueden lanzar la excepción **IOException**, por ello en **main()** se ha añadido **throws IOException** ya que no se incluye el manejador **try-catch**:

```
import java.io.*;
public class LeerFichTexto {
    public static void main(String[] args) throws IOException {
        //declarar fichero
        File fichero =
            new File("C:\\EJERCICIOS\\UNI1\\LeerFichTexto.java");
        //crear el flujo de entrada hacia el fichero
        FileReader fic = new FileReader(fichero);
        int i;
        while ((i = fic.read()) != -1) //se va leyendo un carácter
            System.out.println((char) i);
        fic.close(); //cerrar fichero
    }
}
```

En el ejemplo, la expresión *((char) i)* convierte el valor entero recuperado por el método **read()** a carácter, es decir, hacemos un *cast a char*. Se llega al final del fichero cuando el método **read()** devuelve -1. También se puede declarar el fichero de la siguiente manera:

```
FileReader fic =
    new FileReader("C:\\EJERCICIOS\\UNI1\\LeerFichTexto.java");
```

Para ir leyendo de 20 en 20 caracteres escribimos:

```
char b[] = new char[20];
while ((i = fic.read(b)) != -1) System.out.println(b);
```

**ACTIVIDAD 1.2**

Crea un fichero de texto con algún editor de textos y después realiza un programa Java que visualice su contenido. Cambia el programa Java para que el nombre del fichero se acepte al ejecutar el programa desde la línea de comandos.

Los métodos que proporciona la clase **FileWriter** para escritura son:

Método	Función
<code>void write(int c)</code>	Escribe un carácter.
<code>void write(char[] buf)</code>	Escribe un array de caracteres.
<code>void write(char[] buf, int desplazamiento, int n)</code>	Escribe n caracteres de datos en la matriz <i>buf</i> comenzando por <i>buf[desplazamiento]</i> .
<code>void write(String str)</code>	Escribe una cadena de caracteres.
<code>void append(char c)</code>	Añade un carácter a un fichero.

Estos métodos también pueden lanzar la excepción **IOException**. Igual que antes declaramos el fichero mediante la clase **File** y a continuación se crea el flujo de salida hacia el fichero con la clase **FileWriter**. El siguiente ejemplo escribe caracteres en un fichero de nombre *FichTexto.txt* (si no existe lo crea). Los caracteres se escriben uno a uno y se obtienen de un *String* que se convierte en array de caracteres:

```
import java.io.*;
public class EscribirFichTexto {
    public static void main(String[] args) throws IOException {
        File fichero = new
        File("C:\\EJERCICIOS\\UNI1\\FichTexto.txt"); //declarar fichero
        //crear flujo de salida
        FileWriter fic = new FileWriter(fichero);

        String cadena ="Esto es una prueba con FileWriter";
        //convierte la cadena en array de caracteres para extraerlos 1 a 1
        char[] cad = cadena.toCharArray();
        for(int i=0; i<cad.length; i++)
            fic.write(cad[i]); //se va escribiendo un carácter

        fic.append('*'); //se añade al final un *
        fic.close(); //cerrar fichero
    }
}
```

En vez de escribir los caracteres uno a uno, también podemos escribir todo el array usando **fic.write(cad)**. El siguiente ejemplo escribe cadenas de caracteres que se obtienen de un array de *String*, las cadenas se irán insertando en el fichero una a continuación de la otra sin saltos de linea:

```
String prov[] =
    {"Albacete", "Avila", "Badajoz", "Cáceres", "Huelva", "Jaén",
     "Madrid", "Segovia", "Soria", "Toledo", "Valladolid", "Zamora"};

for(int i=0; i<prov.length; i++) fic.write(prov[i]);
```

Hay que tener en cuenta que si el fichero existe cuando vayamos a escribir caracteres sobre él, todo lo que tenía almacenado anteriormente se borrará. Si queremos añadir caracteres al final, usaremos la clase **FileWriter** de la siguiente manera, colocando en el segundo parámetro del constructor el valor **true**:

```
FileWriter fic = new FileWriter(fichero, true);
```

**FileReader** no contiene métodos que nos permitan leer líneas completas, pero **BufferedReader** sí; dispone del método **readLine()** que lee una línea del fichero y la devuelve, o devuelve **null** si no hay nada que leer o llegamos al final del fichero. También dispone del método **read()** para leer un carácter. Para construir un **BufferedReader** necesitamos la clase **FileReader**:

```
BufferedReader fichero = new  
    BufferedReader (new FileReader(NombreFichero));
```

El siguiente ejemplo lee el fichero *LeerFichTexto.java* línea por línea y las va visualizando en pantalla, en este caso, las instrucciones se han agrupado dentro de un bloque **try-catch**:

```
import java.io.*;  
public class LeerFichTextoBuf {  
    public static void main(String[] args) {  
        try{  
            BufferedReader fichero = new BufferedReader(  
                new FileReader("LeerFichTexto.java"));  
            String linea;  
            while((linea = fichero.readLine())!=null)  
                System.out.println(linea);  
  
            fichero.close();  
        }  
        catch (FileNotFoundException fn ){  
            System.out.println("No se encuentra el fichero");}  
        catch (IOException io) {  
            System.out.println("Error de E/S ");}  
    }  
}
```

La clase **BufferedWriter** también deriva de la clase **Writer**. Esta clase añade un buffer para realizar una escritura eficiente de caracteres. Para construir un **BufferedWriter** necesitamos la clase **FileWriter**:

```
BufferedWriter fichero = new  
    BufferedWriter(new FileWriter(NombreFichero));
```

El siguiente ejemplo escribe 10 filas de caracteres en un fichero de texto y después de escribir cada fila salta una línea con el método **newLine()**:

```
import java.io.*;  
public class EscribirFichTextoBuf {  
    public static void main(String[] args) {  
        try{  
            BufferedWriter fichero = new BufferedWriter
```

```
        (new FileWriter("FichTexto.txt"));
    for (int i=1; i<11; i++) {
        fichero.write("Fila numero: "+i); //escribe una línea
        fichero.newLine(); //escribe un salto de línea
    }
    fichero.close();
}
catch (FileNotFoundException fn ){
    System.out.println("No se encuentra el fichero");
}
catch (IOException io) {
    System.out.println("Error de E/S ");
}
```

La clase **PrintWriter**, que también deriva de **Writer**, posee los métodos ***print(String)*** y ***println(String)*** (idénticos a los de *System.out*) para escribir en un fichero. Ambos reciben un *String* y lo escriben en un fichero, el segundo método, además, produce un salto de línea. Para construir un **PrintWriter** necesitamos la clase **FileWriter**:

```
PrintWriter fichero = new  
PrintWriter(new FileWriter(NombreFichero));
```

El ejemplo anterior usando la clase **PrintWriter** y el método *println()* quedaría así:

```
PrintWriter fichero = new PrintWriter  
                    (new FileWriter("FichTexto.txt"));  
for(int i=1; i<11; i++)  
    fichero.println("Fila numero: "+i);  
fichero.close();
```

## 1.6.2. Ficheros binarios

Los ficheros binarios almacenan secuencias de dígitos binarios que no son legibles directamente por el usuario como ocurría con los ficheros de texto. Tienen la ventaja de que ocupan menos espacio en disco. En Java, las dos clases que nos permiten trabajar con ficheros son **FileInputStream** (para entrada) y **FileOutputStream** (para salida), estas trabajan con flujos de bytes y crean un enlace entre el flujo de bytes y el fichero.

Los métodos que proporciona la clase **FileInputStream** para lectura son similares a los vistos para la clase **FileReader**, estos métodos devuelven el número de bytes leídos o -1 si se ha llegado al final del fichero:

Método	Función
int read()	Lee un byte y lo devuelve
int read(byte[] b)	Lee hasta $b.length$ bytes de datos de una matriz de bytes
int read(byte[] b, int desplazamiento, int n)	Lee hasta $n$ bytes de la matriz $b$ comenzando por $b[desplazamiento]$ y devuelve el número leído de bytes

Los métodos que proporciona la clase **FileOutputStream** para escritura son: