UD1: Acceso a ficheros

1. Introducción

Las computadoras utilizan ficheros para guardar los datos, incluso después de que el programa termine su ejecución. Se suele denominar a los datos que se guardan en ficheros datos persistentes, porque existen, persisten más allá de la ejecución de la aplicación. Los ordenadores almacenan los ficheros en unidades de almacenamiento secundario como discos duros, discos ópticos, etc. En esta unidad veremos cómo hacer con Java estas operaciones de crear, actualizar y procesar ficheros.

Java considera a cada archivo como un flujo secuencial de bytes, como se ve a

continuación:

A todas estas operaciones, que constituyen un flujo de información del programa con el exterior, se les conoce como Entrada/Salida (E/S). Distinguimos dos tipos de E/S: la E/S estándar que se realiza con el terminal del usuario y la E/S a través de ficheros, en la que se trabaja con ficheros de disco.

Todas las operaciones de E/S en Java vienen proporcionadas por el paquete estándar del API de Java denominado java.io que incorpora interfaces, clases y excepciones para acceder a todo tipo de ficheros.

Excepciones

Cuando se trabaja con archivos, es normal que pueda haber errores, por ejemplo: podríamos intentar leer un archivo que no existe, o podríamos intentar escribir en un archivo para el que no tenemos permisos de escritura. Para manejar todos estos errores debemos utilizar excepciones. Las dos excepciones más comunes al manejar archivos son:

- FileNotFoundException: Si no se puede encontrar el archivo.
- IOException: Si no se tienen permisos de lectura o escritura o si el archivo está dañado.

2. Utilización del sistema de archivos

En este apartado veremos cómo crear y borrar directorios, poder filtrar archivos, es decir,

buscar sólo aquellos que tengan determinada característica, por ejemplo, que su extensión sea: .txt.

Los ficheros o archivos y directorios del disco se representan de forma lógica en las aplicaciones Java como objetos de la clase File.

La clase File no se utiliza para transferir datos entre la aplicación y el disco, sino para obtener información sobre los ficheros y directorios de éste e incluso para la creación y eliminación de los mismos.

2.1 Creación de un objeto File

Existen diversas formas de crear un objeto File en función de cómo se indique la localización del fichero o directorio que va a representar. Por ejemplo, el constructor *File(String path)* permite construir un objeto File a partir de su dirección absoluta o relativa al directorio actual:

```
f = new File ("datos.txt");
File f2 = new File ("misubdirectorio");
```

La creación del objeto no implica que exista el fichero o directorio indicado en la ruta. Si éste no existe, las anteriores instrucciones no provocarán ninguna excepción, aunque tampoco será creado de forma implícita.

La clase File también permite crear un objeto hijo dentro de un directorio padre, para esto se puede usar los siguientes constructores:

```
File(File padre, String hijo);
```

Para crear físicamente el fichero o directorio indicado en el constructor de File habrá que recurrir a los siguientes métodos de la clase:

- boolean createNewFile(). Crea el fichero cuyo nombre ha sido especificado en el constructor. Devuelve *true* si se ha podido crear el fichero, mientras que el resultado será *false* si ya existía y por tanto no ha sido creado. Por ejemplo, si el fichero datos.txt del ejemplo anterior no existe, se podría ejecutar la siguiente instrucción para crearlo:

  ```
  f.createNewFile();
  ```

  La llamada a createNewFile() puede provocar una excepción IOException que habrá que capturar.

  Si un objeto File hace referencia a un fichero no existente y no es creado de forma explícita invocando a *createNewFile(),* la creación del mismo se llevará a cabo implícitamente cuando se vaya a utilizar el objeto File para construir un objeto Writer u OutputStream, a fin de realizar una operación de escritura sobre el fichero.

- boolean mkdir(). Crea el directorio cuyo nombre ha sido especificado en el constructor. Si el directorio no existía y se ha podido crear, el método devolverá *true,* de lo contrario el resultado será *false.*

  Por ejemplo, para crear físicamente el subdirectorio referenciado por el objeto File f2 sería:

  f2.mkdir();

La clase File también proporciona un constructor que permite crear un objeto File asociado a un fichero, indicando a través de otro objeto File el directorio donde se encuentra dicho fichero:

File (File dir, String nombre_fichero)

En este caso, el directorio especificado en *dir* debe existir, de lo contrario se producirá una excepción al intentar utilizar el fichero para realizar cualquier operación sobre el mismo.

El siguiente bloque de instrucciones crearía un fichero llamado info.txt en el interior del directorio *datos:*

```
File f=new File ("datos");
f.mkdir(); //si datos no existiese y esta instrucción
           //no estuviera, la llamada a createNewFile()
           //provocaría una excepción.
File fichero=new File (f, "info.txt");
fichero.createNewFile();
```

2.2 Información sobre un fichero/directorio

Una vez creado el objeto File asociado al fichero o directorio, podemos obtener información del mismo aplicándole los siguientes métodos de la clase File:

- boolean canRead(). Indica si se puede o no leer el fichero.
- boolean canWrite(). Indica si se puede o no escribir en el fichero.
- boolean exists(). Indica si existe o no el fichero o directorio indicado en la ruta.
- boolean isFile(). Indica si el objeto File hace referencia o no a un fichero.
- boolean isDirectory(). Indica si el objeto File hace referencia o no a un directorio.
- String getName(). Devuelve el nombre del fichero sin el path.
- String getAbsolutePath(). Devuelve el path absoluto completo.

2.3 Eliminación y renombrado

Para eliminar físicamente un fichero o directorio la clase File proporciona el método *delete*() con el siguiente formato:

- boolean delete(). Elimina el fichero o directorio especificado por el objeto. Si el

elemento ha podido ser eliminado el método devolverá *true,* si no devolverá *false.* Un directorio sólo podrá ser eliminado si está vacío; si no lo está, la llamada a *delete*() sobre el objeto File devolverá *false,* aunque no provocará ninguna excepción.

Si lo que queremos es renombrar un fichero o directorio existente, utilizaríamos el siguiente método:

- boolean renameTo(File nuevo). Renombra el fichero o directorio, asignándole el nombre del objeto File especificado. La llamada a este método devolverá *true* si se ha podido renombrar el elemento, mientras que el resultado será *false* si esto no ha sido posible. En el caso de un directorio, no es necesario que esté vacío para poderlo renombrar.

El siguiente ejemplo renombra un fichero existente:

```
File f=new File ("fichero.txt");
f.createNewFile(); //para poderlo renombrar
                          //debe existir
File f2=new File ("nuevo_nombre.txt );
f.renameTo (f2) ; //renombra "fichero.txt" a
                       //"nuevo_nombre.txt"
```

2.4. Listar directorios

Para listar directorios se utiliza el método list() de la clase File. Dicho método devuelve un array de cadenas de caracteres con los nombres de los ficheros y directorios dentro del fichero asociado al objeto File

El siguiente ejemplo lista el contenido un directorio:

```
File d= new File(directorio);
String[] contenido_d=d.list();
```

Ejercicios con la Clase File

1. Crea un método de Java que cree la siguiente estructura de ficheros y directorios en el directorio desde el escritorio de tu ordenador:

```
d
        d1
                f11
                f12
        d2
                d21
                f21
                d22
                        f222
        d3
                d31
```

2. Crea un método de Java que dado como entrada el directorio raíz del ejercicio 1, liste los ficheros y directorios del mismo, mostrando la misma información que se ve en el ejercicio 1.

3. Modifica el método del ejercicio 2 para que pueda listar cualquier estructura de ficheros y directorios que contenga el directorio raíz pasado como parámetro. Pruébalo con varias estructuras de directorios y ficheros.

4. Escribir un método que muestre los nombres de los archivos de un directorio, que se pasará como argumento cuya extensión coincida con la que se pase como segundo argumento.

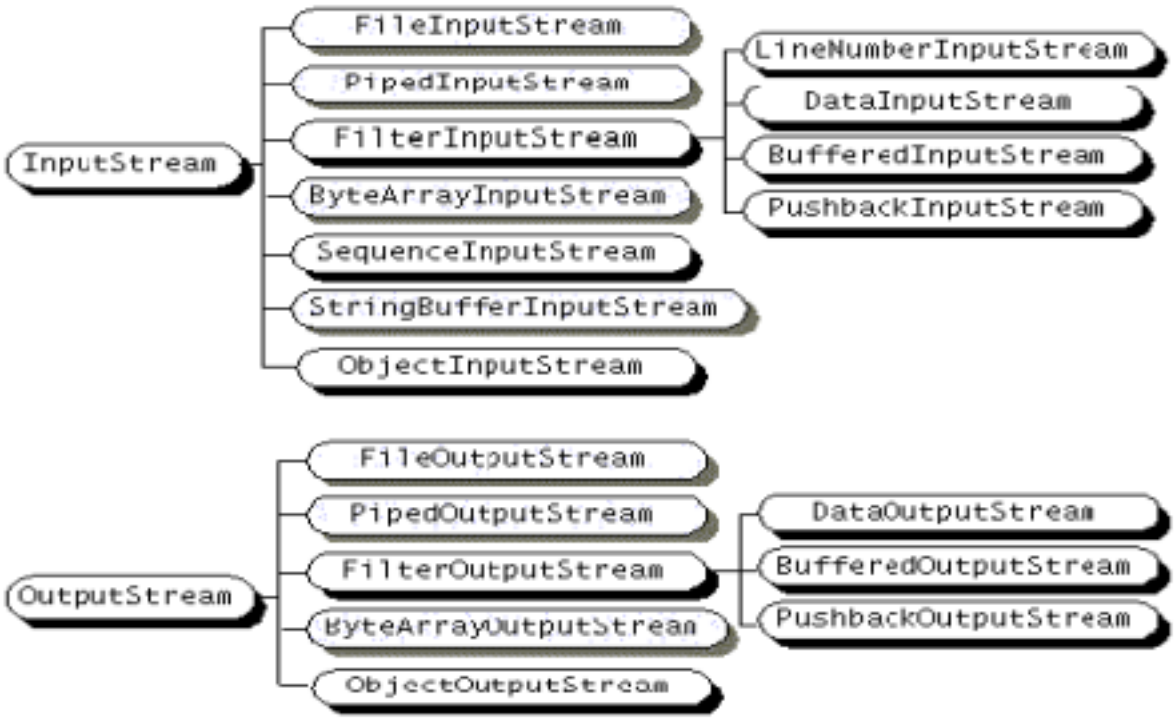
5. Escribe un método que borre todos los ficheros que tienen la extensión txt dentro de un directorio que se pasará como parámetro.

3. Clases relativas a los flujos

Existen dos tipos de flujos, flujos de bytes y flujos de caracteres:

- Los flujos de caracteres (16 bits) se usan para manipular datos legibles por humanos (por ejemplo un fichero de texto). Vienen determinados por dos clases abstractas: Reader y Writer. Dichas clases manejan flujos de caracteres Unicode. De ellas derivan subclases concretas que implementan los métodos definidos destacados los métodos read() y write() que, en este caso, leen y escriben caracteres de datos respectivamente.
- Los flujos de bytes (8 bits) se usan para manipular datos binarios, legibles sólo por la máquina (por ejemplo un fichero de programa). Su uso está orientado a la lectura y escritura de datos binarios. El tratamiento del flujo de bytes viene determinado por dos clases abstractas que son InputStream y OutputStream. Estas dos clases son las que definen los métodos que sus subclases tendrán implementados y, de entre todos, destacan read() y write() que leen y escriben bytes de datos respectivamente.

Las clases del paquete java.io se pueden ver en la ilustración. Destacamos las clases relativas a flujos:
• BufferedInputStream: permite leer datos a través de un flujo con un buffer intermedio.
• BufferedOutputStream: implementa los métodos para escribir en un flujo a través de un buffer.
• FileInputStream: permite leer bytes de un fichero.
• FileOutputStream: permite escribir bytes en un fichero o descriptor.



4. Manejo de ficheros

A la hora de manipular un fichero es muy importante seguir los siguientes pasos:

1) Abrir o crear el fichero.
2) Realizar las operaciones que necesitemos sobre el fichero.
3) Cerrar el fichero.

También es importante controlar las excepciones para evitar que se produzcan fallos en tiempo de ejecución.

4.1. Escritura en ficheros de texto con la clase FileWriter

En primer lugar, para escribir en un fichero de texto, hay que construir un objeto de tipo FileWriter que nos permita tener acceso al fichero en modo escritura. Para ello, podemos utilizar uno de los siguientes constructores:

- *FileWriter (String path)*
- *FileWriter (File fichero)*
- *FileWriter (String path, boolean append)*
- *FileWriter (File fichero, boolean append)*

Donde el parámetro *append* permite indicar si los datos que se van a escribir se añadirán a los ya existentes *(true)* o sobreescribirá a estos *(false)*. Si se utiliza uno de los dos primeros constructores, los datos escritos en el fichero sustituirán a los existentes (equivalente a append=false).

La clase FileWriter proporciona un método *write()* que permite escribir en el fichero la cadena de caracteres pasada como parámetro.

Sobre el control de excepciones: hay que tener en cuenta que la clase FileWriter lanza la excepción IOException si, por ejemplo, el nombre del fichero existe pero es directorio en vez de un fichero regular, o no existe pero no se puede crear o no se puede abrir por cualquier razón.

4.2. Escritura en ficheros de texto con la clase PrintWriter

La utilización de la clase PrintWriter posibilita que la escritura sobre un fichero se realice de la misma forma que la escritura en pantalla. La diferencia está en que en el caso de la consola o pantalla el objeto PrintWriter asociado se encontraba en System.out, mientras que para un fichero de texto habrá que construir el objeto PrintWriter a partir de un objeto FileWriter:

```
FileWriter fw = new FileWriter("datos.txt");

PrintWriter out = new PrintWriter(fw) ;
```

Desde la versión Java 5 también es posible crear un objeto PrintWriter a partir de un objeto File o incluso de la ruta del fichero, por lo que las dos instrucciones

anteriores podrían reducirse a una sola:

```
PrintWriter out = new PrintWriter ("datos.txt");
```

Una vez creado el objeto PrintWriter, podemos utilizar los métodos *print(), println()* y *printf()* para escribir en el fichero. El siguiente bloque de instrucciones almacenaría en un fichero el grupo de nombres existentes en un array:

```
import java.io.*;

public class Grabación {
    public static void main (String [] args) throws IOException
        {
            //array de nombres
        String [] nombres= {"ana", "rosa", "jorge", "manuel"} ;

            FileWriter fw = new FileWriter ("datos.txt");

            PrintWriter out=new PrintWriter (fw);

            for(int i=0;i<nombres.length;i++)
            {
                    out.println (nombres[i]);
            }
            out.flush();
            out.close();
        }
}
```

Obsérvese la utilización de los métodos *flush()* y *close()* de PrintWriter después de la escritura en el fichero. La llamada al método *flush()* garantiza que todos los datos enviados a través del buffer de salida han sido escritos en el fichero, mientras que *close()* cierra la conexión con el fichero y libera los recursos utilizados por ésta.

4.3. Lectura de un fichero con la clase FileReader

Para poder recuperar información de un fichero de texto es necesario primeramente crear un objeto FileReader asociado al mismo. Un objeto FileReader representa un fichero de texto "abierto" para la lectura de datos. Este objeto es capaz de adaptar la información recuperada del fichero a las características de una aplicación Java, transformando los bytes almacenados en el mismo en caracteres unicode.

Se puede construir un objeto FileReader a partir de un objeto File existente o bien proporcionando directamente la ruta del fichero, a partir de los siguientes constructores:

- *FileReader (String path)*
- *FileReader (File fichero)*

La clase FileReader proporciona el método *read()* para la lectura de la información almacenada en un fichero, resultando su uso bastante engorroso ya que los

caracteres son recuperados como tipo byte, debiendo ser posteriormente convertidos a char. Dicho método lee los caracteres del fichero uno a uno hasta encontrar el carácter -1, que indica el fin del fichero.

Sobre el control de excepciones: hay que tener en cuenta que la clase FileReader lanza la excepción FileNotFoundException: si el fichero no existe, es un directorio en vez de un fichero regular o no se puede abrir por cualquier otra razón.

4.4. Lectura de ficheros con la clase BufferedReader

La clase BufferedReader se utiliza para crear un *buffer* que haga de memoria auxiliar.

La idea es que cuando una aplicación necesita leer datos de un fichero, tiene que estar esperando a que el disco en el que está el fichero le proporcione la información. Un dispositivo cualquiera de memoria masiva, por muy rápido que sea, es mucho más lento que la CPU del ordenador.

Así que, es fundamental reducir el número de accesos al fichero a fin de mejorar la eficiencia de la aplicación, y para ello se asocia al fichero una memoria intermedia, el *buffer*, de modo que cuando se necesita leer un byte del archivo, en realidad se traen hasta el *buffer* asociado al flujo, ya que es una memoria mucho más rápida que cualquier otro dispositivo de memoria masiva.

Para crear un objeto BufferedReader, en primer lugar hay que crear un objeto FileReader, que se usará como entrada de su constructor, como se puede ver a continuación:

BufferedReader (Reader entrada)

Una vez creado el objeto, puede utilizarse el método *readLine()* para leer líneas de texto del fichero de forma similar a como se leían del teclado. En el caso de un fichero, y dado que éste puede estar formado por más de una línea, será necesario utilizar un bucle *while* para recuperar todas las líneas de texto del mismo de forma secuencial. El método *readLine()* apunta a la siguiente línea de texto después de recuperar la línea actual. Cuando no existan más líneas para leer, la llamada a *readLine()* devolverá *null.*

Si lo que se quiere es leer carácter a carácter en vez de línea a línea, deberíamos utilizar el método *read()* en lugar de *readLine().* El método *read()* devuelve un entero que representa el código unicode del carácter leído, siendo el resultado -1 si no hay más caracteres para leer.

Ejercicios con flujo de caracteres

1. Escribe un programa en el que escribas "Hola mundo" en un fichero
2. Modifica el programa anterior para escribir en el fichero ya creado
3. Lee un fichero
4. Lee un fichero que cuenta con varias líneas
5. Realiza un programa que cuente el número de palabras de un fichero
6. Escribe un programa que, dado un fichero como entrada, cree un nuevo fichero en el que se copie el contenido del primer fichero eliminando los espacios en blanco y convirtiendo las minúsculas en mayúsculas.

5. Lectura y escritura de flujo de bytes

Las clases PrintWriter y BufferedReader están pensadas para escribir y recuperar, respectivamente, cadenas de caracteres en un fichero. En el caso de flujos de bytes tenemos las clases FileOutputStream y FileInputStream.

5.1 Creación de un objeto FileOutputStream

Este objeto es el equivalente al **FileWriter** utilizado en la escritura de cadenas de caracteres. Como en el caso de éste, se puede crear un objeto FileOutputStream que permita añadir información al fichero o sobrescribirla haciendo uso de los constructores:

FileOutputStream (File fichero, boolean append)
FileOutputStream (String path, boolean append)

La clase FileOutputStream es una subclase de OutputStream, la cual representa un stream o flujo de salida para la escritura de bytes.

5.2 Creación de un objeto FileInputStream

Es el equivalente al **FileReader** utilizado en la lectura de cadenas de caracteres. Se puede crear un objeto de esta clase a partir de un objeto File o de la cadena que representa la ruta del fichero.

FileInputStream es una subclase de InputStream que, al igual que OutputStream, está orientada al tratamiento de bytes. En este caso, a la lectura de bytes.

5.3. Escritura de datos primitivos en fichero con Java

Para almacenar datos en un fichero en forma de tipos básicos Java, el paquete java.io proporciona las clases FileOutputStream y DataOutputStream.

A partir del objeto FileOutputStream se puede crear un objeto DataOutputStream para realizar la escritura de los datos. El constructor utilizado sería:

DataOutputStream (OutputStream)

La clase DataOutputStream proporciona métodos para escribir datos en un fichero en cada uno de los ocho tipos primitivos Java. Estos métodos tienen el formato:

void writeXxx(xxx dato)

Siendo xxx el nombre del tipo primitivo Java.

El siguiente programa escribe en un fichero el contenido de un array de enteros:

```
import java.io.*;
public class GrabaArray {
    public static void main (String [] args)throws IOException
    {
        FíleOutputStream fw=
        new FileOutputStream (“datos.mbd”n,false);
        DataOutputStream ds=new DataOutputStream(fw); //array de enteros
        int [] m={5,10,3,6};
        for(int i=0 ; i<m.length;i ++){
            ds.writeInt(m[i]);
    }
```

```
                //cierra el stream

                ds.close();

        }

}
```

5.4. Lectura de tipos primitivos de un fichero

Para realizar la lectura de los datos almacenados en un fichero mediante DataOutputStream**,** el paquete java.io proporciona la clase DataInputStream que a su vez se apoya en FíleInputStream**.**

A partir del objeto FileInputStream se puede crear un DataInputStream para realizar la lectura de los datos.

Esta clase dispone de los métodos *xxx readXxx()* para recuperar los datos almacenados en el tipo indicado, siendo *xxx* el nombre del tipo primitivo.

El siguiente programa recupera todos los datos almacenados en el fichero del programa anterior y los muestra por pantalla:

```
import java.io.*;
public class VocadoDatos {

        public static void main (String [] args) throws IOException {

                DataInputStream ds=new DataInputStream(

                   new FileInputStream ("datos.mbd"));

                try{
                        //bucle infinito
                        for (;;){
                                System.out.println(ds.readInt());
                }
                }
                catch(EOFException e){

                        // aquí cerramos
                }
        }
}
```

Como se desprende del ejemplo anterior, el método ***readInt()*** devuelve el entero actual y tras la lectura de éste se posiciona en el siguiente dato entero del fichero. Cuando se llegue

al final del fichero se producirá una excepción **EOFExcepction** al invocar al método ***readInt(),*** produciéndose una salida del bucle infinito y finalizando la aplicación.

Ejercicios con flujo de bytes

1. Realiza un programa que realice una copia de una imagen que se encuentre en tu escritorio.
2. Escribe un programa que cree un archivo que contiene parejas de números enteros separados por blanco en cada línea. La introducción de datos finalizará al pulsar la tecla INTRO.
   Ejemplo de estructura del documento:
   12 14
   25 12

   Nota: Los números en el documento deberán representarse mediante un tipo primitivo de Java (int), no mediante caracteres.
   Nota 2: Puedes utilizar un Scanner para introducir los números por teclado.
   Nota 3: La comprobación de si se ha pulsado la tecla INTRO se deberá hacer cuando introduzcas dos números, dado que los números van en parejas. Puedes introducir por teclado los números de uno en uno o de dos en dos.

3. Se tiene un fichero que contiene dos números enteros en cada línea. Se pide hacer un programa que lea el fichero y realice los siguientes cálculos:
   - La media aritmética de los números que están en la primera columna.
   - La media ponderada de los números de la primera columna empleando como pesos los de la segunda.

7. Acceso aleatorio a ficheros

A veces no necesitamos leer un fichero de principio a fin, sino acceder al fichero como si fuera una base de datos, donde se accede a un registro concreto del fichero. Java proporciona la clase RandomAccessFile para este tipo de entrada/ salida.

La clase RandomAccessFile permite utilizar un fichero de acceso aleatorio en el que el programador define el formato de los registros.

```
RandomAccessFile objFile = new RandomAccessFile( ruta, modo );
```

Donde ruta es la dirección física en el sistema de archivos y modo puede ser:

1. "r" para sólo lectura.

2. "rw" para lectura y escritura.

La clase RandomAccessFile implementa los interfaces DataInput y DataOutput. Para abrir un archivo en modo lectura haríamos:

```
RandomAccessFile in = new RandomAccessFile("input.dat", "r");
```

Para abrirlo en modo lectura y escritura:

```
RandomAccessFile inOut = new RandomAccessFile("input.dat", "rw");
```

Esta clase permite leer y escribir sobre el fichero, no se necesitan dos clases diferentes.

Hay que especificar el modo de acceso al construir un objeto de esta clase: sólo lectura o lectura/escritura.

Dispone de métodos específicos de desplazamiento como seek y skipBytes para poder moverse de un registro a otro del fichero, o posicionarse directamente en una posición concreta del fichero.

Crear fichero:

Debido a la existencia de dos constructores, tenemos dos posibilidades de abrir un fichero de acceso aleatorio:

Mediante el nombre del fichero:

```
miFichero = new RandomAccessFile(String nombre, String modo);
```

Mediante un objeto file:

```
miFichero = new RandomAccessFile(File fichero, String modo);
```

El parámetro modo determina si se tiene acceso de sólo lectura (r) o bien de lectura/ escritura (rw).

Por ejemplo, se puede abrir un fichero de sólo lectura:

```
RandomAccessFile miRAFile;
miRAFile = new RandomAccessFile("/usr/bin/pepe.txt","rw");
```

Acceder a la información:

Con un objeto RandomAccessFile se tiene acceso a todas las operaciones read() y write(), además de los métodos writeXXX y readXXX de las clases DataInputStream y DataOutputStream.

Se dispone de muchos métodos para ubicarse dentro de un fichero:

```
long getFilePointer() //Devuelve la posición actual del puntero del fichero
void seek(long pos) // Sitúa el puntero del fichero en una posición
                    // determinada. La posición se da como un
                    // desplazamiento en bytes desde el
                    // comienzo del fichero. La posición 0 marca el comienzo
                    // de ese fichero

long length() // Devuelve la longitud del fichero. La posición length() marca el
              // final de ese fichero.

int skipBytes(int desplazamiento) // Desplaza el puntero desde la posición
                                  // actual, el número de bytes indicado por
                                  // desplazamiento
```

Actualizar la información:

Con los ficheros de acceso aleatorio se puede añadir información a un fichero aleatorio existente.

Aquí nos iríamos al final del fichero:

```
miFR = new RandomAccessFile("c:\\aleatorios\prueba.dat","rw");

miFR.seek(miFR.length());
```

Cualquier write() que hagamos a partir de este punto del código añadirá información al fichero.

Ejercicios de ficheros de acceso aleatorio

1. Escribe un programa que, dado un fichero como entrada, modifique el fichero eliminando los espacios en blanco y convirtiendo las minúsculas en mayúsculas.
2. Escribe un programa que guarde en un fichero tantos enteros como el usuario quiera, una vez el usuario termine se deberá cambiar el todos los enteros con valor 5 por el valor 0.

```
package me.victor;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.RandomAccessFile;

/**
 * Hello world!
 *
 */
public class App {

    public static void main(String[] args) {
        escribir("datos.dat");
        //leer("datos.dat");
        accesoDirecto("datos.dat");

    }

    public static void accesoDirecto(String path) {
        try {
            try {
                RandomAccessFile rafe = new
RandomAccessFile(path, "rw");
                while (true) {
                    //el primer byte siempre apunta a
la primera posición

                    //rafe.skipBytes(4); // cuando
lea algo, el puntero se coloca al final de lo que
hayamos leido

System.out.println(rafe.readInt());
                    Double num =
rafe.readDouble(); //este número lo vamos a
modificar.
                    System.out.println("--" + num +
"--");
```

```
                    rafe.seek(rafe.getFilePointer() -
8);
                    rafe.writeDouble(num + 1);
                    rafe.skipBytes(8);

System.out.println(rafe.readChar());


                    /*leemos, volvemos al inicio de
la posición leída,
                y ahora la modificamos añadiendole
uno, siempre
                tenemos que tener en cuenta que para
trabajar con
                lectura y escrotura al mismo tiempo
tenemos que
                reposicionarnos a done estábamos
antes de hacer la escritura.
                     */

 /*Tenemos que tener en cuenta que al leer
                 un tipo de dato u otro, este ocupa
un número distinto de bytes,
                 por lo que tenemos que volver atrás
un número distinto de posiciones
                 (bytes).*/
                }
            } catch (EOFException e) {
                System.out.println(e);
            }
        } catch (IOException e) {
            System.out.println(e);
        }

    }
```

```
public static void escribir(String path) {
        try {
            FileOutputStream fos = new
FileOutputStream(path);
            DataOutputStream dos = new
DataOutputStream(fos);
            dos.writeInt(1);
            dos.writeDouble(2.10);
            dos.writeChar('a');

            dos.writeInt(2);
            dos.writeDouble(3.99);
            dos.writeChar('b');

            dos.writeInt(3);
            dos.writeDouble(4.444);
            dos.writeChar('c');

            dos.flush();
            dos.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
public static void leer(String path) {
        try (DataInputStream dis = new
         DataInputStream(new FileInputStream(path))) {
            try {
                for (;;) {
                    System.out.print(dis.readInt());

System.out.print(dis.readDouble());
                    System.out.print(dis.readChar());
                    System.out.println();
                }
            } catch (EOFException e) {
                System.out.println(e);
            }
        } catch (IOException e) {
            System.out.println(e);
        }
    }}
```

```
package ejemplo_acceso_a_datos;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        escribirLinea(new File("dato1.txt"),
"klajsdhalkjshdlkasjh");
        escribirLinea(new File("dato1.txt"),
"klajsdhalkjshdlkasjh");
        escribirLinea(new File("dato1.txt"),
"klajsdhalkjshdlkasjh");
        escribirLinea(new File("dato1.txt"),
"klajsdhalkjshdlkasjh");
        escribirLinea(new File("dato1.txt"), "pepe");

        leer(new File ("dato1.txt"));

        }//main end


public static void escribirLinea (File f, String
linea) {
    try {
        //1.Abrir
        FileWriter fw=new FileWriter(f, true);
        PrintWriter pw = new PrintWriter(fw, true);

        //2. Operar
        pw.write(linea);

        //3.Cerrar Importante,
        //si vamos a escribir con printwriter,
        //no podemos hace flush con el filewriter,
        //o el fielflush, porque ahi no hay nada que
        flushear o ejecutar,
        //esta todo en el otro lado.
```

```
            pw.flush();
            pw.close();
        } catch (IOException e) {
            System.out.println(e);
        }

}//method end


public static void leer (File f) {
        try {
            //1.Abrir
            FileReader fr = new FileReader(f);

            /*2. Operar, va a ser devuelto un int,
             * por eso ponemos el caracter como un
entero,
             * despues tnemos que hcaer la conversion */
            int car = fr.read();

            while (car!=-1) {
                System.out.print((char)car);
                car=fr.read();
            }

            //3.Cerrar Importante,
            fr.close();
        } catch (IOException e) {
            System.out.println(e);
        }

}//method end

}//class end
```

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import
javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class CrearXML {

    public static void main(String[] args) {
        try {
            // 1. Crear una instancia de
DocumentBuilderFactory
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();

            // 2. Crear un DocumentBuilder
            DocumentBuilder dBuilder =
dbFactory.newDocumentBuilder();

            // 3. Crear un nuevo documento (vacío)
            Document doc = dBuilder.newDocument();

            // 4. Crear el elemento raíz (root
element)
            Element rootElement =
doc.createElement("biblioteca");
            doc.appendChild(rootElement);

            // 5. Crear elementos hijo y añadirlos al
documento
            Element libro =
doc.createElement("libro");
            rootElement.appendChild(libro);

            // Añadir atributos al libro
```

```
            libro.setAttribute("id", "1");

            // Añadir el elemento "titulo"
            Element titulo =
doc.createElement("titulo");
            titulo.appendChild(doc.createTextNode("El
Quijote"));
            libro.appendChild(titulo);

            // Añadir el elemento "autor"
            Element autor =
doc.createElement("autor");

autor.appendChild(doc.createTextNode("Miguel de
Cervantes"));
            libro.appendChild(autor);

            // Añadir el elemento "precio"
            Element precio =
doc.createElement("precio");

precio.appendChild(doc.createTextNode("19.99"));
            libro.appendChild(precio);

            // 6. Escribir el contenido del XML en un
archivo
            TransformerFactory transformerFactory =
TransformerFactory.newInstance();
            Transformer transformer =
transformerFactory.newTransformer();

            // Opciones de salida (formato)

transformer.setOutputProperty(OutputKeys.INDENT,
"yes");  // Indentación del XML
            transformer.setOutputProperty("{http://
xml.apache.org/xslt}indent-amount", "4");

            DOMSource source = new DOMSource(doc);
            StreamResult result = new
StreamResult(new File("biblioteca.xml"));

            // Transformar el documento a archivo
```

```
            transformer.transform(source, result);

            System.out.println("Archivo XML creado
exitosamente!");

        } catch (ParserConfigurationException |
TransformerException e) {
            e.printStackTrace();
        }
    }
}
```