M6.UF1.A1
GESTIÓN Y PERSISTENCIA EN FICHEROS

Eduard Lara

INDICE

1. INTRODUCCION

- Un fichero o archivo es un conjunto de bits almacenado en un dispositivo. Los ficheros tienen un nombre y se ubican en directorios, el nombre del fichero debe ser único en ese directorio.
- Por convención cuentan con diferentes extensiones y suelen ser de 3 letras (pdf, doc, gif, etc) que permiten saber el tipo de fichero
- Un fichero esta formado por un conjunto de registros o líneas y cada registro por un conjunto de campos relacionados. La manera en que se agrupan los datos en un fichero depende de quien lo diseñe.

2. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS

Operaciones para el manejo habitual sobre archivos/directorios:

- Creación
- Borrado
- Operar con él (lectura/escritura, inserción, borrado, etc)
- Cierre

Clase java.io.File

Permite referenciar un archivo o directorio

A través de esta clase podemos manipular los atributos de los elementos referenciados

2. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS

DIRECTORIOS:

CONSTRUCCION DEL OBJETO	MÉTODOS	DESCRIPCION
File nombre_directorio= new File (String pathname)	Boolean mkdri()	Crea directorio identificado por el pathname
	Boolean mkdirs()	Crea directorio identificado por el pathname, incluyendo los directorios necesarios de la ruta
	Boolean isDirectory()	Investiga si es un directorio
	Boolean exists()	Investiga si existe
	String[] Lists()	Lista los archivos de un directorio
	File[] listRoots()	Lista cada uno de los sistemas de archivos disponibles
	Boolean delete()	Borra directorio

ARCHIVOS:

CONSTRUCCION DEL OBJETO	MÉTODOS	DESCRIPCION
File nombre_fichero= New File(String parent, String child) New File(String pathname) New File(File parent, String child)	Boolean createNewFile()	Crea archivo vacío identificado por el pathname
	Boolean isFile()	Investiga si es un fichero
	Boolean exists()	Investiga si existe
	Boolean canRead()	Investiga si la aplicación puede leer el archivo
	Boolean canWrite()	Investiga si la aplicación puede escribir el archivo
	Boolean delete()	Borra archivo

2. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS

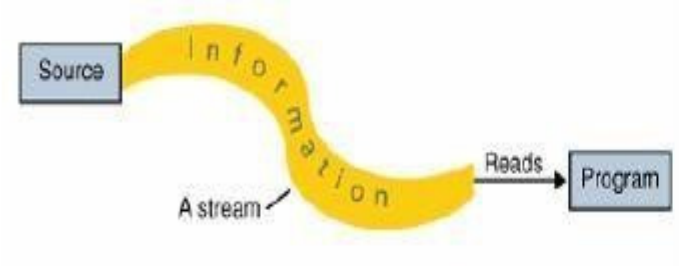
ACTIVIDAD OPERACIONES CON FICHEROS

Crea una aplicación Java que cree dos directorios en el workspace del proyecto denominados: DirectorioA, DirectorioB.

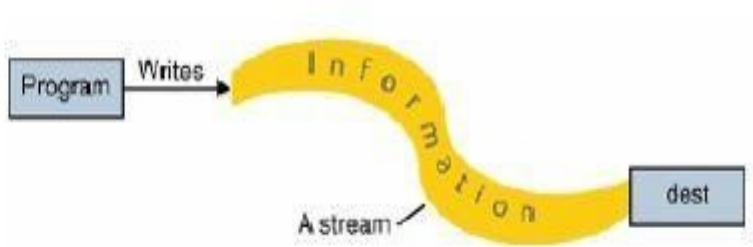
En el DirectorioA, crea dos ficheros denominados Fichero1.txt; Fichero2.txt Lista por consola el contenido de ambos directorios DirectorioA, DirectorioB Borra el DirectorioB
Borra el DirectorioA y su contenido

3. FLUJOS O STREAMS

- El sistema de entrada/salida de Java dispone de distintos tipos de clases dentro del paquete java.io
- Usa la abstracción de flujo (stream) para tratar la comunicación de información entre una fuente y un destino
- STREAMS: canales, flujos de datos o «tuberías»



Entrada



Salida

3. FLUJOS O STREAMS

Existen dos tipos de flujos:

- **Flujos de bytes (8bits):** realizan las operaciones de entradas y salidas de bytes y su uso está orientado a la lectura/escritura de datos binarios. Todas las clases de flujos de bytes descienden de las **clases InputStream, OutputStream**.
  - FileInputStream, FileOutputStream: manipulan flujos de bytes provenientes o dirigidos a ficheros.
  - DataInputStream, DataOutputStream: permiten leer/escribir datos de tipo primitivo (int, double, float, long, etc)
- **Flujos de caracteres:** realizan las operaciones de entradas y salidas de caracteres Unicode. La razón de ser de estas clases es la internacionalización; la antigua jerarquía de flujos de E/S solo soporta flujos de 8 bits, no manejando caracteres Unicode de 16 bits que se utilizaba con fines de internacionalización. Todas las clases de flujos de caracteres desciendes de las **clases Reader y Writer**
  - FileReader, FileWriter: acceso a ficheros lectura/escritura
  - BufferedReader, BufferedWriter: se utilizan para evitar que cada lectura/escritura acceda directamente al fichero, utilizando un buffer intermedio entre la memoria y el stream

4. FORMAS DE ACCESO A UN FICHERO

Hay dos formas de acceso a la información almacenada en un fichero:

Acceso Secuencial: los datos o registros se leen y se escriben en orden (como una cinta de video). Si se quiere acceder a un dato o registro que está en 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 hay ya escritos.

Flujos de bytes-> clases FileInputStream, FileOutputStream, DataInputStream, DataOutputStream

Flujos de caracteres -> clases FileReader, FileWriter, BufferedReader, BufferedWriter

Acceso Aleatorio: permite acceder directamente a un dato o registro si 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.

Clase RandomAccesFile

5. OPERACIONES SOBRE FICHEROS

- **Creación del fichero:** el fichero se crea en disco con un nombre que después se debe utilizar para acceder a él.
- **Apertura del fichero:** para que el programa pueda operar con el fichero, la primero que tiene que hacer es la apertura del mismo.
- **Cierre del fichero:** el fichero se debe cerrar cuando el programa no lo vaya a utilizar
- **Lectura de los datos del fichero:** este proceso consiste en transferir información del fichero a la memoria principal, normalmente a través de variable/variables del programa, donde se depositarán los datos extraídos del fichero.
- **Escritura de los datos en el fichero:** este proceso consiste en transferir información de la memoria (por medio de variables del programa) al fichero.

5. OPERACIONES SOBRE FICHEROS

Una vez abierto el fichero, las operaciones típicas son:

- **Alta:** se añade un nuevo registro al fichero
- **Baja:** se elimina un registro existente del fichero
- **Modificación:** se modifica parte del contenido de un registro existente del fichero.
- **Consulta:** buscar un registro determinado en el fichero.

Acceso secuencial

Alta: el registro se inserta a continuación del último insertado

Baja: 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 se desea dar de baja. Una vez re- escritos hemos de borrar el fichero inicial y renombrar el fichero auxiliar dándole el nombre del fichero inicial.

Modificación: consiste en localizar el registro a modificar (leyendo todos los registros anteriores), realizar la modificación y reescribir el fichero inicial en otro fichero auxiliar que incluya el registro modificado (similar a la operación de dar de Baja)

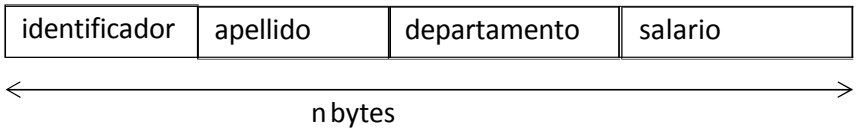
Consulta: es necesario empezar la lectura desde el primer resgistro y continuar leyendo secuencialmente hasta localizar el registro buscado

5. OPERACIONES SOBRE FICHEROS

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.

Para posicionarnos en un registro es necesario tener presente el tamaño del registro y la clave que identifica de forma única al registro. A esto se le llama función de conversión

Por ejemplo un fichero de empleados con 4 campos: identificador, apellido, departamento y salario



Función de conversión:
posicion = (identificador-1)*n

identificador = 1, 2…..

Para localizar al empleado con identificador X, necesitaremos acceder a la posición= (X-1)*n

5. OPERACIONES SOBRE FICHEROS

Una vez abierto el fichero, las operaciones típicas son:

- **Alta:** se añade un nuevo registro al fichero
- **Baja:** se elimina un registro existente del fichero
- **Modificación:** se modifica parte del contenido de un registro existente del fichero.
- **Consulta:** buscar un registro determinado en el fichero.

Acceso aleatorio

Alta: para dar de alta un registro necesitamos saber su clave, aplicar la función de conversión para obtener su dirección y escribir el registro en ella.

Baja: las bajas suelen realizarse de forma lógica, es decir, se suele utilizar un campo del registro a modo de switch, por ejemplo que tenga un valor de 0 o -1 para darle de baja y cualquier otro valor cuando el registro exista. Habría que localizar el registro a dar de baja a partir de su campo clave y reescribir en este campo el valor de 0 o -1.

Modificación: para modificar un registro necesitamos saber su clave, aplicar la función de conversión para obtener su dirección y modificar los datos que nos interesen y reescribir el registro en esa posición.

Consulta: para consultar un registro necesitamos saber su clave, aplicar la función de conversión para obtener su dirección, comprobar que y leer el registro ubicado en esa posición.

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

FICHEROS DE TEXTO:

Almacenan caracteres alfanuméricos en un formato estándar (ASCII, UNICODE; UTF8, etc).

Clases:

- java.io.FileReader
- java.io.FileWriter
- java.io BufferedReader
- java.io.BufferedWriter
- Java.io.PrintWriter

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

Clase java.io.FileReader

FileReader

```
public FileReader(File file)
            throws FileNotFoundException
```

Creates a new `FileReader`, given the `File` to read from.

Parameters:

`file` - the `File` to read from

Throws:

`FileNotFoundException` - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

`int`	`read()` Reads a single character.
`int`	`read(char[] cbuf)` Reads characters into an array.
`abstract int`	`read(char[] cbuf, int off, int len)` Reads characters into a portion of an array.

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

Clase java.io.FileWriter

FileWriter

```
public FileWriter(File file)
           throws IOException
```

Constructs a FileWriter object given a File object.

Parameters:

`file` - a File object to write to.

Throws:

`IOException` - if the file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason

`Writer`	`append(char c)` Appends the specified character to this writer.
`Writer`	`append(CharSequence csq)` Appends the specified character sequence to this writer.
`void`	`write(char[] cbuf)` Writes an array of characters.
`abstract void`	`write(char[] cbuf, int off, int len)` Writes a portion of an array of characters.
`void`	`write(int c)` Writes a single character.
`void`	`write(String str)` Writes a string.
`void`	`write(String str, int off, int len)` Writes a portion of a string.

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
24      static String[] apellido = {"Fernández", "Gil", "López", "Ramos", "Sevilla", "Casillas", "Rey"};
25      static int[] departamento = {10,20,10,10,30,30,20};
26      static double[] salario={1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0};
27
28      public static void main(String[] args) {
29          File file = new File("datos.dat");
            PrintWriter printWriter=null;
31          try{
32             //apertura del stream (thorws FileNotFoundException):
33              printWriter=new PrintWriter(new FileWriter(file));
34              for (int i=0;i<apellido.length;i++){
35                  printWriter.println((i+1)+" "+apellido[i]+" "+departamento[i]+" "+salario[i]);
36              }
37          } catch (FileNotFoundException ex) {
38              System.err.println("Fichero no existe "+ex);
39          } catch (IOException ex) {
40              System.err.println("No es posible escribir en el fichero "+ex);
41          }finally{
42              //Se cierra el stream y se liberan los recursos de sistema asociados a él.
                if (printWriter!=null) printWriter.close();
44
45          }
```

Escritura de texto en fichero

Se insertan los datos de empleados (id, apellido, departamento y salario).

La clase PrintWriter también deriva de la clase Writer. Dispone de los métodos print(String) y println(String) para escribir en un fichero

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
47          BufferedReader bufferedReader=null;
48          try {
49              //apertura del stream (thorws FileNotFoundException):
50              bufferedReader=new BufferedReader(new FileReader(file));
51              //lectura de lineas (thorws IOException):
52              String linea;
53              while ((linea=bufferedReader.readLine())!=null){
54                      System.out.println(linea);
55              }
56          } catch (FileNotFoundException ex) {
57              System.err.println("Fichero no existe "+ex);
58          } catch (IOException ex) {
59              System.err.println("No es posible leer fichero "+ex);
60          }finally{
61              try {
62                  //Se cierra el stream y se liberan los recursos de sistema asociados a él.
63                  bufferedReader.close();
64              } catch (IOException ex) {
65                  System.err.println("No es posible cerrar el fichero "+ex);
66              }
67          }
```

Lectura de texto desde fichero

Se leen los datos del fichero línea por línea

```
run:
1 Fernández 10 1000.45
2 Gil 20 2400.6
3 López 10 3000.0
4 Ramos 10 1500.56
5 Sevilla 30 2200.0
6 Casillas 30 1435.87
7 Rey 20 2000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

ACTIVIDAD FICHERO TEXTO ACCESO SECUENCIAL

A partir del programa anterior, añade las siguientes funcionalidades:

a) Reciba un identificador de empleado desde la línea de comandos y visualice sus datos. Si el empleado no existe, debes informar al usuario.
b) Inserte en el fichero el empleado con idEmpleado 20, apellido Perez, departamento 10 y salario 1000.56
c) Elimine en el fichero el empleado con idEmpleado 2.
d) Modifique en el fichero el salario del empleado con idEmpleado=4, a 3000.0

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

FICHEROS BINARIOS

Almacenan secuencias de dígitos binarios que no son legibles directamente por el usuario como ocurre con los ficheros de texto.
Ocupan menos espacio en disco

Clases:

- java.io.FileInputStream
- java.io.FileOutputStream
- java.io.DataInputStream
- java.io.DataOutputStream

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
22      static String[] apellido = {"Fernández", "Gil", "López", "Ramos", "Sevilla", "Casillas", "Rey"};
23      static int[] departamento = {10,20,10,10,30,30,20};
24      static double[] salario={1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0};
25
26      public static void main(String[] args) {
27          File file = new File("datos.dat");
28          FileOutputStream fileOut=null;
29          DataOutputStream dataOut=null;
30          try {
31          //apertura del stream de salida (thorws FileNotFoundException):
32          fileOut = new FileOutputStream(file);
33          dataOut= new DataOutputStream (fileOut);
34          for (int i=0; i<apellido.length;i++){
35             //escritura de bytes (thorws IOException):
36             dataOut.writeInt(i+1);
37             dataOut.writeUTF(apellido[i]);
38             dataOut.writeInt(departamento[i]);
39             dataOut.writeDouble(salario[i]);}
40          } catch (FileNotFoundException ex) {
41              System.err.println("Fichero no existe "+ex);
42          }catch (IOException ex) {
43              System.err.println("Error I/O "+ex);
44          }finally{
45              try{
46              fileOut.close();
47              dataOut.close();
48              }catch (IOException ex) {
49                  System.err.println("No es posible cerrar el fichero "+ex);
50              }
```

Escritura de bytes en fichero

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
53          FileInputStream fileIn=null;
54          DataInputStream dataIn=null;
55          try {
56          //apertura del stream de entrada (thorws FileNotFoundException):
57          fileIn = new FileInputStream(file);
58          dataIn= new DataInputStream (fileIn);
59
60          try{
61              while (true){
62                  System.out.println(dataIn.readInt()+ " "
63                          +dataIn.readUTF()+" "+dataIn.readInt()+" "+dataIn.readDouble());
64              }
65          }catch (EOFException ex){System.out.println("Fin de fichero detectado");}
66
67          } catch (FileNotFoundException ex) {
68              System.err.println("Fichero no existe "+ex);
69          }catch (IOException ex) {
70              System.err.println("Error I/O "+ex);
71          }finally{
72              try{
73              fileIn.close();
74              dataIn.close();
75              }catch (IOException ex) {
76                  System.err.println("No es posible cerrar el fichero "+ex);
77              }
78          }
```

Lectura de bytes desde fichero

```
run:
1 Fernández 10 1000.45
2 Gil 20 2400.6
3 López 10 3000.0
4 Ramos 10 1500.56
5 Sevilla 30 2200.0
6 Casillas 30 1435.87
7 Rey 20 2000.0
Fin de fichero detectado
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

ACTIVIDAD FICHERO BINARIO ACCESO SECUENCIAL

A partir del programa anterior, añade las siguientes funcionalidades:

a) Reciba un identificador de empleado desde la línea de comandos y visualice sus datos. Si el empleado no existe, debes informar al usuario.
b) Inserte en el fichero el empleado con idEmpleado 20, apellido Perez, departamento 10 y salario 1000.56
c) Elimine en el fichero el empleado con idEmpleado 2.
d) Modifique en el fichero el salario del empleado con idEmpleado=4, a 3000.0

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

OBJETOS SERIALIZABLES

Hemos visto como se guardan los tipos de datos primitivos en un fichero.
¿Y si tenemos el objeto Empleado con varios atributos (id, apellido, departamento, salario y queremos guardarlo en un fichero?
Java permite guardar objetos en ficheros binarios-> para ello la clase debe implementar la **interface Serializable**.
La interface Serializable dispone de los métodos: Para leer el objeto desde el fichero:

- void readObject (ObjectInputStream stream) throws IOException, ClassNotFoundException.

Para escribir el objeto en el fichero:

- void writeObject (ObjectOutputStream stream) throws IOException

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
13  public class Empleado implements Serializable{
14
15      private int id;
16      private String apellido;
17      private int departamento;
18      private double sueldo;
19
20      public Empleado(int id, String apellido, int departamento, double sueldo) {
21          this.id = id;
22          this.apellido = apellido;
23          this.departamento = departamento;
24          this.sueldo = sueldo;
25      }
26
27      public int getId() {return id;}
28      public String getApellido() {return apellido;}
29      public int getDepartamento() {return departamento;}
30      public double getSueldo() {return sueldo;}
31
32      public void setId(int id) {this.id = id;}
33      public void setApellido(String apellido) {this.apellido = apellido;}
34      public void setDepartamento(int departamento) {this.departamento = departamento;}
        public void setSueldo(double sueldo) {this.sueldo = sueldo;}
36  }
```

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
24      static String[] apellido = {"Fernández", "Gil", "López", "Ramos", "Sevilla", "Casillas", "Rey"};
25      static int[] departamento = {10,20,10,10,30,30,20};
26      static double[] salario={1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0};
27
28      public static void main(String[] args) {
29          File file = new File("datos.dat");
30          FileOutputStream fileOut=null;
31          ObjectOutputStream dataOut=null;
32          try {
33              //apertura del stream de salida (thorws FileNotFoundException):
34              fileOut = new FileOutputStream(file);
35              dataOut= new ObjectOutputStream (fileOut);
36              for (int i=0; i<apellido.length;i++){
37                  Empleado empleado = new Empleado ((i+1), apellido[i], departamento[i], salario[i]);
38                  //escritura de objetos (thorws IOException):
39                  dataOut.writeObject(empleado);
40              }
41          } catch (FileNotFoundException ex) {
42              System.err.println("Fichero no existe "+ex);
43          }catch (IOException ex) {
44              System.err.println("Error I/O "+ex);
45          }finally{
46              try{
47              fileOut.close();
48              dataOut.close();
49              }catch (IOException ex) {
50                  System.err.println("No es posible cerrar el fichero "+ex);
51              }
52          }
```

Escritura de objetos (bytes) en fichero

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

```
55          FileInputStream fileIn=null;
56          ObjectInputStream dataIn=null;
57          try {
58              //apertura del stream de entrada (thorws FileNotFoundException):
59              fileIn = new FileInputStream(file);
60              dataIn= new ObjectInputStream (fileIn);
61              try{
62                  while (true){
63                  //lectura de objetos (thorws ClassNotFoundException):
64                  Empleado empleado = (Empleado) dataIn.readObject();
65                  System.out.println(empleado.getId()+ " "
66                          +empleado.getApellido()+" "+empleado.getDepartamento()+
67                          " "+empleado.getSueldo());
68                  }
69              }catch (EOFException ex){System.out.println("Fin de fichero detectado");}
70
71          } catch (FileNotFoundException ex) {
72              System.err.println("Fichero no existe "+ex);
73          }catch (IOException ex) {
74              System.err.println("Error I/O "+ex);
75          }catch (ClassNotFoundException ex) {
76              System.err.println("Error clase no encontrada "+ex);
77          }finally{
78              try{
79              fileIn.close();
80              dataIn.close();
81              }catch (IOException ex) {
82                  System.err.println("No es posible cerrar el fichero "+ex);
83              }
```

Lectura de objetos (bytes) desde fichero

6. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO SECUENCIAL

ACTIVIDAD FICHERO BINARIO (OBJETOS) ACCESO SECUENCIAL

A partir del programa anterior, añade las siguientes funcionalidades:

a) Reciba un identificador de empleado desde la línea de comandos y visualice sus datos. Si el empleado no existe, debes informar al usuario.
b) Inserte en el fichero el empleado con idEmpleado 20, apellido Perez, departamento 10 y salario 1000.56
c) Elimine en el fichero el empleado con idEmpleado 2.
d) Modifique en el fichero el salario del empleado con idEmpleado=4, a 3000.0

7. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO ALEATORIO

Java dispone de la clase RandomAccessFile

Constructors

Constructor and Description
`RandomAccessFile(File file, String mode)` Creates a random access file stream to read from, and optionally to write to, the file specified by the `File` argument.
`RandomAccessFile(String name, String mode)` Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Throws:

`IllegalArgumentException` - if the mode argument is not equal to one of `"r"`, `"rw"`, `"rws"`, or `"rwd"`

`FileNotFoundException` - if the mode is `"r"` but the given file object does not denote an existing regular file, or if the mode begins with `"rw"` but the given file object does not denote an existing, writable regular file and a new regular file of that name cannot be created, or if some other error occurs while opening or creating the file

`SecurityException` - if a security manager exists and its `checkRead` method denies read access to the file or the mode is "rw" and the security manager's `checkWrite` method denies write access to the file

El argumento mode puede ser:

Value	Meaning
`"r"`	Open for reading only. Invoking any of the `write` methods of the resulting object will cause an `IOException` to be thrown.
`"rw"`	Open for reading and writing. If the file does not already exist then an attempt will be made to create it.

7. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO ALEATORIO

Una vez abierto el fichero pueden usarse los métodos read() y write() de las clases DataInputStream y DataOutputStream vistas antes.

La clase RandomAccessFile maneja un puntero que indica la posición actual en el fichero.

- Cuando el fichero se crea el puntero se coloca a 0, apuntando al principio del mismo.
- Las sucesivas llamadas a los métodos read() y write() ajustan el puntero según la cantidad de bytes leídos o escritos.

Los métodos más importantes son:

long **getFilePointer**(): devuelve la posición actual del puntero del fichero

void **seek**(long pos): coloca el puntero del fichero en la posición determinada por el argumento *pos*

long **length**(): devuelve el tamaño del fichero en bytes (final del fichero)

int **skipBytes**(int n): desplaza el puntero desde la posición actual el número de bytes indicados en el argumento *n*

7. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO ALEATORIO

Tamaño de los tipos de datos:

- *int-> 4bytes*
- *carácter unicode (char)->2bytes*
- *double-> 8bytes*
- *short->2bytes*
- *byte->1byte*
- *long->8bytes*
- *boolean->1bit*
- *float->8bytes*

7. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO ALEATORIO

```
20      static String[] apellido = {"Fernández", "Gil", "López", "Ramos", "Sevilla", "Casillas", "Rey"};
21      static int[] departamento = {10,20,10,10,30,30,20};
22      static double[] salario={1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0};
23
24      public static void main(String[] args)  {
25          File file = new File("datos.dat");
26          RandomAccessFile randomFile=null;
27          try {
28              randomFile = new RandomAccessFile(file, "rw");
29              StringBuffer buffer = null;
30              for (int i=0; i<apellido.length;i++){
31               randomFile.writeInt(i+1); //apertura del stream de entrada (thorws IOException):
32               buffer = new StringBuffer(apellido[i]);
33               buffer.setLength(10); //max 10 caracteres para el apellido
34               randomFile.writeChars(buffer.toString());
35               randomFile.writeInt(departamento[i]);
36               randomFile.writeDouble(salario[i]);
37              }
38
```

Se insertan los datos de empleados (id, apellido, departamento y salario). La longitud de cada registro será de 36 bytes:

- id: int (4bytes)
- apellido: cadena de 10 caracteres (2bytes*10=20bytes)
- departamento: int (4bytes)
- salario: double (8bytes)

7. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO ALEATORIO

```
39              int posicion =0;
                int idEmpleado, departamento;
                double salario;
42              char[] apellidoCharSequence = new char[10];
43
44              randomFile.seek(posicion);//nos situamos al principio del fichero
45
46              while(randomFile.getFilePointer()<randomFile.length()){
47                  randomFile.seek(posicion);//actualiza posición puntero
48                  idEmpleado=randomFile.readInt();
49                  for (int i=0; i<apellidoCharSequence.length;i++){
50                      apellidoCharSequence[i]=randomFile.readChar();
51                  }
52                  String apellidoS = new String (apellidoCharSequence);
53                  departamento = randomFile.readInt();
54                  salario = randomFile.readDouble();
55                  System.out.println(idEmpleado+" "+apellidoS+" "+departamento+" "+salario);
56                  posicion= posicion +36 ;
57              }
58          } catch (FileNotFoundException ex) {
59              System.err.println("Fichero no existe "+ex);
60          } catch (IOException ex) {
61            System.err.println("Error I/O "+ex);
62          } finally{
63              try {
64              randomFile.close();
65              } catch (IOException ex) {
66                  System.err.println("No es posible cerrar el fichero "+ex);
67              }
```

7. CLASES PARA LA GESTIÓN DE FLUJOS DE DATOS: ACCESO ALEATORIO

ACTIVIDAD ACCESO ALEATORIO

A partir del programa anterior, añade las siguientes funcionalidades:

a) Reciba un identificador de empleado desde la línea de comandos y visualice sus datos. Si el empleado no existe, debes informar al usuario.
b) Inserte en el fichero el empleado con idEmpleado 20, apellido Perez, departamento 10 y salario 1000.56
c) Elimine en el fichero el empleado con idEmpleado 2.
d) Modifique en el fichero el salario del empleado con idEmpleado=4, a 3000.0

8. CRITERIOS DE ELECCIÓN DEL TIPO DE FICHERO

Cuando trabajemos con ficheros, las decisiones que hay que tomar es cómo se almacenarán los datos (en bytes o en caracteres), y el tipo de acceso (secuencial o aleatorio).

En muchos casos, sea cual sea la elección tomada el problema se puede resolver igualmente, la única diferencia es la complejidad del programa que se tendrá que implementar y el número de veces que el fichero deberá ser leído.

Siempre que se use el acceso aleatorio, implica que el fichero está orientado a bytes.

	Ventajas	Inconvenientes
Acceso secuencial	Rápida capacidad de acceso al siguiente registro. Aprovechan mejor la utilización de espacio. Sencillos de utilizar	No se puede acceder directamente a un registro determinado, hay que leer antes todos los anteriores Los ficheros no pueden ser actualizados, sino que deben re-escribirse totalmente
Acceso aleatorio	Rápido acceso a una posición determinada para leer o modificar un registro	Establecer la relación entre la posición que ocupa el registro y su contenido. Se puede desaprovechar parte del espacio destinado al fichero ya que se pueden producir huecos (posiciones ocupadas) entre un registro y otro

8. CRITERIOS DE ELECCIÓN DEL TIPO DE FICHERO

Fichero para almacenar el Ranking de las 10 máximas puntuaciones de un juego

- Fichero de texto acceso secuencial

Fichero que almacena los datos de los adversarios que aparecen en el juego:

- Fichero de bytes (objetos) acceso aleatorio

Atributos de un objeto Adversario:

String Nombre, int nivel int puntos int numVidasMax int ataque int defensa

9. ACCESO FICHEROS XML CON DOM

- XML es un metalenguaje. Es decir, define lenguajes. Nos permite jerarquizar y estructurar la información.
- Los ficheros XML se pueden utilizar para proporcionar datos a una aplicación. Tienen la gran ventaja de ser archivos de texto lo que permite exportar y recuperar datos de una plataforma a otra (solo es necesario conocer la codificación de los caracteres)
- Para leer los ficheros XML se utiliza un procesador XML o “parser”. Los procesadores más utilizados son: DOM y SAX.
- Los procesadores son independientes del lenguaje de programación y existen versiones particulares para Java, VisalBasic, C, etc.

9. ACCESO FICHEROS XML CON DOM

- DOM: Modelo de Objetos de Documento
- Se trata de una API para manipulación de documentos XML y HTML
- Almacena toda la estructura de un documento en memoria en forma de árbol; con nodos padre, nodos hijo y nodos finales (que no tienen descendientes)
- Requiere una buena cantidad de recursos de memoria y tiempo sobre todo si los ficheros XML a procesar son bastante grandes y complejos

9. ACCESO FICHEROS XML CON DOM

- Para poder trabajar con DOM en Java necesitamos las clases e interfaces que componen el paquete **org.w3c.dom** y el paquete **javax.xml.parsers**.
- Las dos clases más importantes son **DocumentBuilderFactory** y **DocumentBuilder**.
- Los programas Java que utilicen DOM necesitan estas y otras interfaces:
  - **Document** → Es un objete que equivale a un ejemplar de un documento XML. Permite crear nuevos nodos en el documento
  - **Element** → Cada elemento del documento XML tiene un equivalente en un objeto de este tipo. Expone propiedades y métodos para manipular los elementos del documento y sus atributos
  - **NodeList** → Contiene una lista con los nodos hijos de un nodo
  - **Attr** → Permite acceder a los atributos de un nodo.
  - **Text** → Son los datos carácter de un elemento

9. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 1

```
public static void main (String args[]) throws IOException {
    File fichero = new File ("AleatorioEmpleado.dat");
    RandomAccessFile file = new RandomAccessFile(fichero, "r");
    int id, dep, posicion=0;
    Double salario;
    char apellido[] = new char[10], aux;

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        DOMImplementation implementation = builder.getDOMImplementation();
        Document document = implementation.createDocument (null,"Empleados", nu
        document.setXmlVersion("1.0");


        for ( ; ;){
            file.seek(posicion);
            id = file.readInt();
            for (int i = 0; i < apellido.length ; i++) {
                aux =file.readChar();
                apellido[i] = aux;
            }
            String apellidos = new String (apellido);
            dep = file.readInt();
            salario = file.readDouble();
            if (id>0) {
                Element raiz = document.createElement ("empleado");
                document.getDocumentElement().appendChild(raiz);
                CrearElemento ("id", Integer.toString(id), raiz, document);
                CrearElemento ("apellido",apellidos.trim(), raiz, document);
                CrearElemento ("dep", Integer.toString(dep), raiz, document);
                CrearElemento ("salario", Double.toString(salario),raiz, docume
            }
            posicion = posicion + 36;
            if (file.getFilePointer() == file.length()) break;
        }

        Source source = new DOMSource (document);
        Result result = new StreamResult (new java.io.File ("Empleados.xml"));
        Transformer transformer = TransformerFactory.newInstance().newTransformer();

        transformer.transform (source, result);
    } catch (Exception e ) {
        System.err.println ("Error: " + e);
    }
    file.close();
}

static void CrearElemento (String datoEmpleado, String valor, Element raiz, Document document) {
    Element elem = document.createElement (datoEmpleado);
    Text text = document.createTextNode(valor);
    raiz.appendChild (elem);
    elem.appendChild (text);
}
```

9. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 1

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    DOMImplementation implementation = builder.getDOMImplementation();
    Document document = implementation.createDocument (null,"Empleados", null);
    document.setXmlVersion("1.0");
```

Se crea una instancia de DocumentBuilderFactory para construir el parser. Se debe encerrar entre try-catch porque se puede producir la excepción ParserConfigurationException.

Creamos un documento vacío de nombre *document* con el nodo raíz de nombre *Empleados* y asignamos la versión XML. La interfaz DOMImplementation permite crear objetos Document con nodo raíz*

El método CrearElemento genera los 4 nodos-hijo (<id>, <apellido>,<dep> y <salario>) de cada nodo <Empleado>. Primero genera el elemento (**Element**), después el texto o valor (**Text**) y los asocia con la raiz o nodo padre. Recibe como parámetros sus textos o valores que deben estar en formato String, el nodo al que se va añadir (*raiz*) y el documento (*document*). Luego lo vemos con más detalle. Para crear un elemento usamos el método **createElement (String)** llevando como parámetro el nombre que se pone entre las etiquetas < y >.

```
static void CrearElemento (String datoEmpleado, String valor, Element raiz, Document document) {
    Element elem = document.createElement (datoEmpleado);
    Text text = document.createTextNode(valor);
    raiz.appendChild (elem);
    elem.appendChild (text);
}
```

9. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 1

La especificación DOM no define ningún mecanismo para generar un fichero XML a partir de un árbol DOM. Para eso usaremos el paquete **javax.xml.transform** que permite especificar una fuente y un resultado. La fuente y el resultado pueden ser ficheros, flujos de datos o nodos DOM entre otros. Primero crearemos la fuente (**Source**) XML a partir del documento (**document**); después se crea el resultado (**Result**) en el fichero *Empleados.xml.* A continuación, se obtiene un **TransformerFactory** y se realiza la transformación del documento al fichero

```
    Source source = new DOMSource (document);
    Result result = new StreamResult (new java.io.File ("Empleados.xml"));
    Transformer transformer = TransformerFactory.newInstance().newTransformer();
    transformer.transform (source, result);
} catch (Exception e ) {
    System.err.println ("Error: " + e);
}
file.close();
```

9. ACCESO FICHEROS XML CON DOM

Escritura de archivos – Ejemplo 2

```
try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.newDocument();

    // Definimos el elemento raíz del documento
    Element eRaiz = doc.createElement("concesionario");
    doc.appendChild(eRaiz);

    // Definimos el nodo que contendrá los elementos
    Element eCoche = doc.createElement("coche");
    eRaiz.appendChild(eCoche);

    // Atributo para el nodo coche
    Attr attr = doc.createAttribute("id");
    attr.setValue("1");
    eCoche.setAttributeNode(attr);

    // Definimos cada uno de los elementos y le asignamos un valor
    Element eMarca = doc.createElement("marca");
    eMarca.appendChild(doc.createTextNode("Renault"));
    eCoche.appendChild(eMarca);
```

```
    Element eModelo = doc.createElement("modelo");
    eModelo.appendChild(doc.createTextNode("Megano"));
    eCoche.appendChild(eModelo);

    Element eCilindrada = doc.createElement("cilindrada");
    eCilindrada.appendChild(doc.createTextNode("1.5"));
    eCoche.appendChild(eCilindrada);

    // Clases necesarias finalizar la creación del archivo XML
    TransformerFactory transformerFactory = TransformerFactory.newInstance();
    Transformer transformer = transformerFactory.newTransformer();
    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(new File("concesionario.xml"));

    transformer.transform(source, result);
} catch(Exception e) {
      e.printStackTrace();
}
```

concesionario.xml

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <concesionario>
3      <coche id="1">
4          <marca>Renault</marca>
5          <modelo>Megano</modelo>
6          <cilindrada>1.5</cilindrada>
7      </coche>
8  </concesionario>
```

9. ACCESO FICHEROS XML CON DOM

Lectura de archivos - Ejemplo 1

```java
public static void  main(String args[]) {
    File file = new File("concesionario.xml");
    Document doc=null;
    try {
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        doc = dBuilder.parse(file);
    } catch(Exception e) {
        e.printStackTrace();
    }
    doc.getDocumentElement().normalize();
    NodeList nList = doc.getElementsByTagName("coche");
    System.out.println("Número de coches: " + nList.getLength());

    for(int temp = 0; temp < nList.getLength(); temp++) {
        Node nNode = nList.item(temp);
        if(nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;
            System.out.println("\nCoche id: " + eElement.getAttribute("id"));
            System.out.println("Marca: " + eElement.getElementsByTagName("marca").item(0).getTextContent());
            System.out.println("Modelo: " + eElement.getElementsByTagName("modelo").item(0).getTextContent());
            System.out.println("Cilindrada: " + eElement.getElementsByTagName("cilindrada").item(0).getTextContent());
        }
    }
}
```

concesionario.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<concesionario>
  <coche id="1">
    <marca>Renault</marca>
    <modelo>Megane</modelo>
    <cilindrada>1.5</cilindrada>
  </coche>
  <coche id="2">
    <marca>Seat</marca>
    <modelo>León</modelo>
    <cilindrada>1.6</cilindrada>
  </coche>

  <coche id="3">
    <marca>Suzuki</marca>
    <modelo>Vitara</modelo>
    <cilindrada>1.9</cilindrada>
  </coche>
</concesionario>
```

9. ACCESO FICHEROS XML CON DOM

Lectura de archivos - Ejemplo 1

- A la hora de leer un archivo XML a través de DOM debemos instanciar una serie de clases antes de poder tratar el fichero. Primero utilizaremos la conocida clase File para cargar nuestro fichero.
- Posteriormente y ya dentro de un try/catch (para tratar los excepciones) parsearemos el fichero con estas clases: Utilizaremos una instancia de DocumentBuilderFactory para poder construir el parser con DocumentBuilder y Document, y cargar el documento con el método **parse()**

```
File file = new File("concesionario.xml");
Document doc=null;
try {
    DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
    doc = dBuilder.parse(file);
} catch(Exception e) {
    e.printStackTrace();
}
```

9. ACCESO FICHEROS XML CON DOM

Lectura de archivos - Ejemplo 1

- A continuación usaremos getDocumentElement() para acceder al nodo raíz del documento y normalize() para elimina nodos vacíos si los hubiera
- Para obtener la lista de nodos (NodeList) de todo el documento, utilizaremos el método Document.getElementsByTagName ("coche").
- Por último, se realiza un bucle para recorrer esa lista de nodos. Por cada nodo se obtienen sus etiquetas y sus valores.

```
doc.getDocumentElement().normalize();
NodeList nList = doc.getElementsByTagName("coche");
System.out.println("Número de coches: " + nList.getLength());

for(int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    if(nNode.getNodeType() == Node.ELEMENT_NODE) {
        Element eElement = (Element) nNode;
        System.out.println("\nCoche id: " + eElement.getAttribute("id"));
        System.out.println("Marca: " + eElement.getElementsByTagName("marca").item(0).getTextContent());
        System.out.println("Modelo: " + eElement.getElementsByTagName("modelo").item(0).getTextContent());
        System.out.println("Cilindrada: " + eElement.getElementsByTagName("cilindrada").item(0).getTextContent());
    }
}
```

10. ACCESO FICHEROS XML CON SAX

- SAX (API Simple para XML) es un conjunto de clases e interfaces que ofrecen una herramienta muy útil para el procesamiento de documentos XML. Permite analizar documentos de forma secuencial (es decir, no carga todo el fichero en memoria como hace DOM), esto implica:
  - Poco consumo de memoria aunque los documentos sean de gran tamaño
  - Impide tener una visión global del documento
  - El acceso es secuencial NO aleatorio
  - SAX NO permite generar archivos XML
- Por tanto, SAX es útil cuando queremos buscar información en ficheros grandes XML o cuando no tenemos acceso completo al documento (acceso mediante *Stream*)
- SAX es una API totalmente escrita en Java e incluida dentro del JRE que nos permite crear nuestro propio parser de XML

10. ACCESO FICHEROS XML CON SAX

- La lectura de un documento XML produce eventos que ocasiona la llamada a métodos:

Documento XML (alumnos.xml)	**Métodos asociados a eventos del documento**
<?xml version=”1.0”?>	startDocument()
<listadealumnos>	startElement()
<alumno>	startElement()
<nombre>	startElement()
Juan	characters()
</nombre>	endElement()
<edad>	startElement()
19	characters()

10. ACCESO FICHEROS XML CON SAX

</edad>	endElement()
</alumno>	endElement()
<alumno>	startElement()
<nombre>	startElement()
María	characters()
</nombre>	endElement()
<edad>	startElement()
20	characters()
</edad>	endElement()
</alumno>	endElement()
</listadealumnos>	endElement()
	endDocument()

10. ACCESO FICHEROS XML CON SAX

- Como se observa en la tabla:
  - La etiqueta de inicio (<?xml versión = “1.0”?>) → provoca el evento **startDocument()**
  - El final de document → provoca el evento **endDocument()**
  - La etiqueta de inicio de un elemento → provoca el evento **startElement()**
  - La etiqueta de final de un elemento → provoca el evento **endElement()**
  - Los caracteres entre etiquetas → provocan el evento **characters()**
- Los objetos que poseen los métodos que tratarán los eventos son:
  - **ContentHandler** → recibe las notificaciones de los eventos que ocurren en el documento
  - **DTDHandler** → recoge eventos relacionados con la DTD (Definición de Tipo de Documento)
  - **ErrorHandler** → define métodos de tratamientos de errores
  - **EntityResolver** → sus métodos se llaman cada vez que hay una referencia a una entidad
  - **DefaultHandler** → clase que provee una implementación por defecto para todos sus métodos, el programador definirá los métodos que serán utilizados por el programa. Esta clase es la que extenderemos para poder crear nuestro parser de XML. En el ejemplo que veremos a continuación, la clase se llama *GestionContenido* y se tratan solo los eventos básicos: inicio y fin de documento, inicio y fin de etiqueta encontrada, encuentra datos carácter.

10. ACCESO FICHEROS XML CON SAX

Ejemplo lectura SAX

```
public static void main(String[] args) throws Exception {
    //A continuación se crea objeto procesador XML - XMLReader
    // Durante la creación de este objeto se puede producir una excepción SAXException.
    SAXParserFactory parserFactory = SAXParserFactory.newInstance();
    SAXParser parser = parserFactory.newSAXParser();
    XMLReader procesadorXML = parser.getXMLReader();

    // A continuación, mediante setContentHandler establecemos que la clase que gestiona los
    //eventos provocados por la  lectura del XML será GestionContenido
    SaxReader gestor = new SaxReader();
    procesadorXML.setContentHandler(gestor);
    // Por último, se define el fichero que se va leer mediante InputSource y se procesa el
    // documento XML mediante el método parse() de XMLReader */
    InputSource fileXML = new InputSource ("concesionario.xml");
    procesadorXML.parse(fileXML);
}
```

10. ACCESO FICHEROS XML CON SAX

Ejemplo lectura SAX

La clase SaxReader implementa los métodos necesarios para crear nuestro parser de XML.

Es decir, definimos los métodos que serán llamados al provocarse los eventos comentados anteriormente: startDocument, startElement, characters, etc.

Si quisieramos tratar más eventos definiríamos el método asociado en esta clase.

SaxReader.java

```
2
3  import org.xml.sax.Attributes;
6
7  public class SaxReader extends DefaultHandler {
8      public SaxReader(){
9          super();
10     }
11     public void startDocument() throws SAXException {
12         System.out.println("**********************************************************************" );
13         System.out.println("Comienzo del documento XML");
14     }
15
16     public void endDocument() throws SAXException {
17         System.out.println("Final del documento XML");
18         System.out.println("**********************************************************************" );
19     }
20
21     public void endElement(String uri, String localName, String qName) throws SAXException {
22         System.out.println("---END ELEMENT (<"+qName+">)---------------------------------" );
23     }
24
25     public void characters(char[] ch, int start, int length) throws SAXException {
26         String contenido = new String(ch, start, length).trim();
27         if(contenido.length() > 0){
28             System.out.println("-----El valor del campo es: " + new String(ch, start, length));
29         }
30     }
31
32     public void startElement(String uri, String localName, String qName, Attributes attributes)  throws SAXException {
33         System.out.println("---START ELEMENT (<"+qName+">)---------------------------------" );
34         if(attributes.getLength()>0){
35             System.out.print("----Atributos de la etiqueta :" );
36             for (int i=0;i<attributes.getLength();i++){
37                 System.out.println("----"+attributes.getQName(i)+":"+attributes.getValue(i)+" ");
38             }
39         }
40     }
41 }
42
```