

INDICE

| | |
|---|-----------|
| 1.6. CLASES PARA GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS | 1 |
| 1.6.1. FICHEROS DE TEXTO | 1 |
| 1.6.2. FICHEROS BINARIOS | 6 |
| 1.6.3. OBJETOS EN FICHEROS BINARIOS | 9 |
| 1.6.4. FICHEROS DE ACCESO ALEATORIO | 13 |
| 1.7. ACTIVIDADES | 17 |

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 del 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:

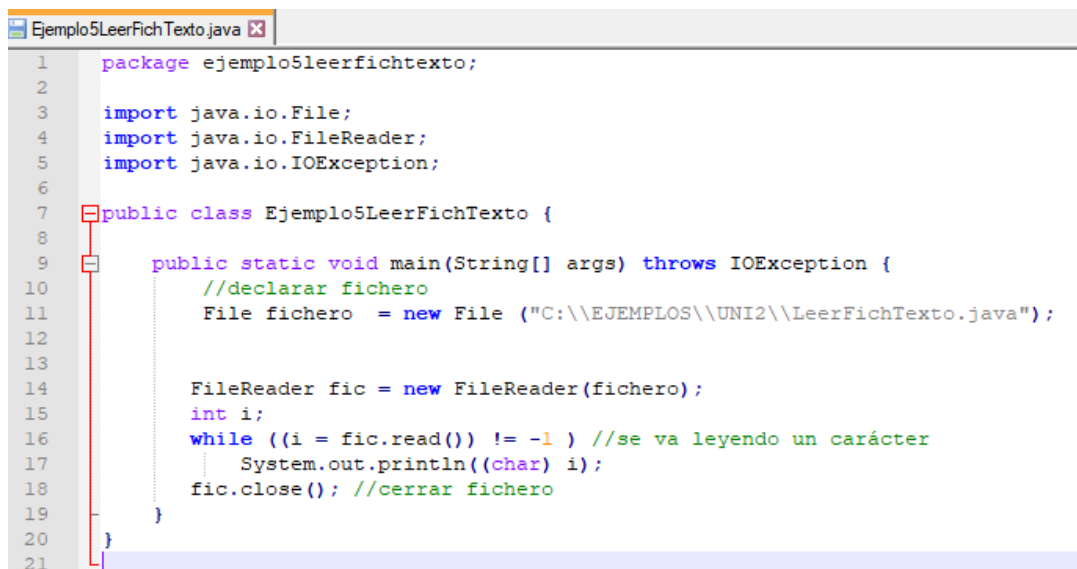
| Método | Función |
|-----------------------------------|--|
| <code>int read()</code> | Lee un carácter y lo devuelve |
| <code>int read(char[] buf)</code> | Lee hasta <i>buf.length</i> caracteres de datos de una matriz de caracteres (<i>buf</i>). Los caracteres leídos del fichero se van almacenando en <i>buf</i> . |

```
int read(char[] buf,
int desplazamiento,
int n)
```

Lee hasta n caracteres de datos de una matriz *buf* comenzando por *buf[desplazamiento]* 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**:



```
1 package ejemplo5leerfichtexto;
2
3 import java.io.File;
4 import java.io.FileReader;
5 import java.io.IOException;
6
7 public class Ejemplo5LeerFichTexto {
8
9     public static void main(String[] args) throws IOException {
10         //declarar fichero
11         File fichero = new File ("C:\\EJEMPLOS\\UNI2\\LeerFichTexto.java");
12
13
14         FileReader fic = new FileReader(fichero);
15         int i;
16         while ((i = fic.read()) != -1 ) //se va leyendo un carácter
17             System.out.println((char) i);
18         fic.close(); //cerrar fichero
19     }
20 }
21
```

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:\\EJEMPLOS\\UNI2\\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);
```

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[] cbuf)</code> | Escribe un array de caracteres. |
| <code>void write(char[] cbuf, int desplazamiento, int len)</code> | Escribe <i>n</i> 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:

```
package ejemplo5leerfichtexto;

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

public class Ejemplo5LeerFichTexto {

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

        FileReader fic = new FileReader ("C:\\EJEMPLOS\\UNI2\\LeerFichTexto.java");

        int i;
        char b[] = new char [20];
        while ((i = fic.read(b)) != -1 ) //se va leyendo un carácter
            System.out.println(b);
        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 obtiene de un array de String, las cadenas se irán insertando en el fichero una a continuación de la otra sin saltos de línea:

```
String prov[] = {"Albacete", "Ávila", "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 de 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("LeerFichTexto.java"));
```

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**:

```
package ejemplo7leerfichtextobuf;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Ejemplo7LeerFichTextoBuf {

    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()**:

```
package ejemplo8escribirfichetextobuf;

import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;

public class Ejemplo8EscribirFichTextoBuf {

    public static void main(String[] args) {
        try{

            BufferedWriter fichero = new BufferedWriter (new FileWriter("C:\\EJEMPLOS\\UNI2\\FichTexto.txt"));

            for (int i=0; i<11; i++){
                fichero.write("Fila número: "+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("C:\\EJEMPLOS\\UNI2\\FichTexto.txt"));

for (int i=0; i<11; i++){
    fichero.println("Fila número: "+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 |
|--|---|
| <code>int read()</code> | Lee un byte y lo devuelve. |
| <code>int read(byte[] b)</code> | Lee hasta <i>b.length</i> bytes de datos de una matriz de bytes. |
| <code>int read(byte[] b, int desplazamiento, int len)</code> | Lee hasta <i>n</i> bytes de la matriz <i>b</i> comenzando por <i>b[desplazamiento]</i> y devuelve el número leído de bytes. |

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

| Método | Función |
|--|--|
| <code>void write(int b)</code> | Escribe un byte. |
| <code>void write(byte[] b)</code> | Escribe <i>b.length</i> bytes . |
| <code>void write(byte[] b, int desplazamiento, int len)</code> | Escribe <i>n</i> bytes a partir de la matriz de bytes de entrada comenzando por <i>b[desplazamiento]</i> . |

El siguiente ejemplo escribe bytes en un fichero y después los visualiza :

```

package ejemplo9escribirfichbytes;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Ejemplo9EscribirFichBytes {

    public static void main(String[] args) throws IOException {
        File fichero = new File("C:\\EJEMPLOS\\UNI2\\FichBytes.dat") ;//declara fichero

        //crea flujo de salida hacia el fichero
        FileOutputStream fileout = new FileOutputStream( fichero);

        //crea flujo de entrada
        FileInputStream filein = new FileInputStream (fichero) ;
        int i;

        for (i=1; i<100; i++)
            fileout.write( i); //escribe datos en el flujo de salida

        fileout.close(); //cerrar stream de salida

        //visualizar los datos del fichero
        while ((i = filein.read()) != -1) //lee datos del flujo de entrada
            System.out.println(i); filein.close(); //cerrar stream de entrada
        filein.close();
    }
}

```

Para añadir bytes al final del fichero usaremos **FileOutputStream** de la siguiente manera, colocando en el segundo parámetro del constructor el valor **true**:

```
FileOutputStream fileout = new FileOutputStream( fichero,true);
```

Para leer y escribir datos de tipos primitivos: *int*, *float*, *long*, etc usaremos las clases **DataInputStream** y **DataOutputStream**. Estas clases definen diversos métodos *readXXX* y *writeXXX* que son variaciones de los métodos *read()* y *write()* de la clase base para leer y escribir datos de tipo primitivo. Algunos métodos se muestran en la siguiente tabla:

| Métodos para LECTURA | Métodos para ESCRITURA |
|---------------------------------------|--|
| <code>boolean readBoolean();</code> | <code>void writeBoolean(boolean v);</code> |
| <code>byte readByte();</code> | <code>void writeBytes(int v);</code> |
| <code>int readUnsignedByte();</code> | <code>void writeBytes(String s);</code> |
| <code>int readUnsignedShort();</code> | <code>void writeShort(int v);</code> |

| | |
|-----------------------------------|--|
| <code>short readShort();</code> | <code>void writeChars(String s);</code> |
| <code>char readChar();</code> | <code>void writeChar (int v);</code> |
| <code>int readInt();</code> | <code>void writeLong(long v);</code> |
| <code>float readFloat();</code> | <code>void writeFloat(float v);</code> |
| <code>double readDouble();</code> | <code>void writeDouble(double v);</code> |
| <code>String readUTF();</code> | <code>void writeUTF(String str);</code> |

Para abrir un objeto **DataInputStream**, se utilizan los mismos métodos que para **FileInputStream**, ejemplo:

```
File fichero = new File("FichData.dat");
FileInputStream filein = new FileInputStream (fichero);
DataInputStream dataIS = new DataInputStream (filein);
```

O bien

```
File fichero = new File ("FichData.dat");
DataInputStream dataIS = new DataInputStream (new FileInputStream (fichero));
```

Para abrir un objeto **DataOutputStream**, se utilizan los mismos métodos que para **FileOutputStream**, ejemplo:

```
File fichero = new File("FichData.dat");
FileOutputStream fileout = new FileOutputStream(fichero);
DataOutputStream dataOS = new DataOutputStream(fileout);
```

O bien

```
File fichero = new File ("FichData.dat");
DataOutputStream dataIS = new DataOutputStream (new FileOutputStream (fichero));
```

El siguiente ejemplo inserta datos en el fichero *FichData.dat*, los datos los toma de dos arrays, uno contiene los nombres de una serie de personas y el otro sus edades, recorreremos los arrays y vamos escribiendo en el fichero el nombre (mediante el método **writeUTF(String)**) y la edad (mediante el método **writeInt(int)**):


```

package ejemplo10escribirfichdata;

import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class Ejemplo10EscribirFichData {

    public static void main(String[] args) throws IOException {
        File fichero = new File("C:\\EJEMPLOS\\UNI2\\FichData.dat") ;//declara fichero
        FileOutputStream fileout = new FileOutputStream( fichero);
        DataOutputStream dataOS = new DataOutputStream (fileout);

        String nombres[] = {"Ana", "Luis Miguel", "Alicia", "Pedro", "Manuel", "Andrés", "Sofía",
                            "Julio", "Antonio", "María Jesús"};

        int edades[] = {14,15,13,15,16,12,16,14,13,11};

        for (int i=0; i<edades.length;i++){
            dataOS.writeUTF(nombres[i]); //escribe nombre
            dataOS.writeInt(edades[i]); //escribe edad
        }
        dataOS.close(); //cerrar stream
    }
}

```

El siguiente ejemplo visualiza los datos grabados anteriormente en el fichero, se deben recuperar en el mismo orden en el que se escribieron, es decir, primero obtenemos el nombre y luego la edad:

Se obtiene la siguiente salida al ejecutar el programa:

```

Nombre: Ana, Edad: 14
Nombre: Luis Miguel, Edad: 15
Nombre: Alicia, Edad: 13
Nombre: Pedro, Edad: 15
Nombre: Manuel, Edad: 16
Nombre: Andrés, Edad: 12
Nombre: Sofía, Edad: 16
Nombre: Julio, Edad: 14
Nombre: Antonio, Edad: 13
Nombre: María Jesús, Edad: 11

```

1.6.3.Objetos en ficheros binarios

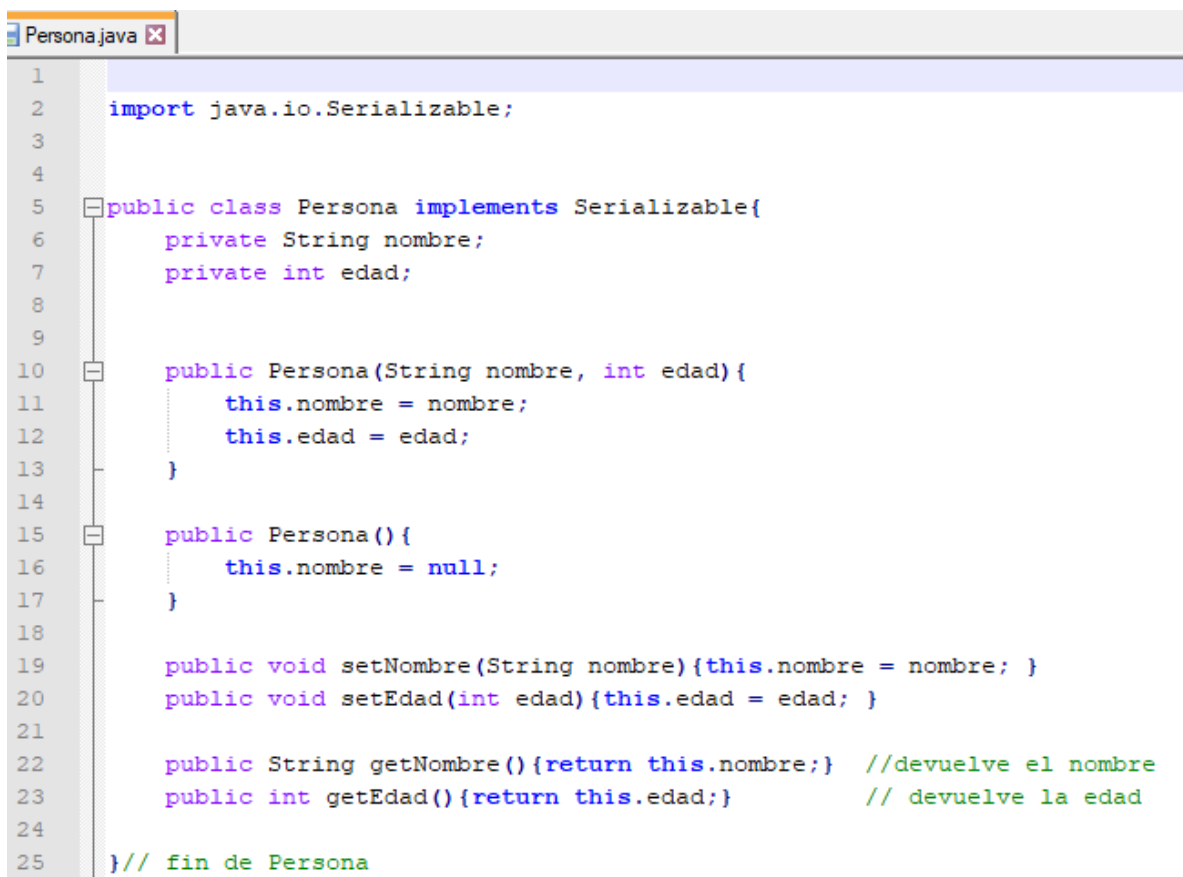
Hemos visto cómo se guardan los tipos de datos primitivos en un fichero, pero, por ejemplo, si tenemos un objeto de tipo empleado con varios atributos (el nombre, la dirección, el salario, departamento, el oficio, etc.) y queremos guardarlo en un fichero, tendríamos que guardar cada atributo que forma parte del objeto por separado, esto se vuelve engorroso si tenemos gran cantidad de objetos. Por ello Java nos permite guardar objetos en ficheros binarios, para poder hacerlo, el obje-

to tiene que implementar la interfaz **Serializable** que dispone de una serie de métodos con los que podremos guardar y leer objetos en ficheros binarios. Los más importantes a utilizar son:

- **Object readObject():** se utiliza para leer un objeto del **ObjectInputStream**. Puede lanzar las excepciones **IOException** y **ClassNotFoundException**.
- **void writeObject(Object obj):** se utiliza para escribir el objeto especificado en el **ObjectOutputStream**. Puede lanzar la excepción **IOException**.

La serialización de objetos de Java permite tomar cualquier objeto que implemente la interfaz **Serializable** y convertirlo en una secuencia de bits que puede ser posteriormente restaurada para regenerar el objeto original.

Para leer y escribir objetos serializables a un stream se utilizan las clases Java **ObjectInputStream** y **ObjectOutputStream** respectivamente. A continuación se muestra la clase *Persona* que implementa la interfaz **Serializable** y que utilizaremos para escribir y leer objetos en un fichero binario. La clase tiene dos atributos: el nombre y la edad y los métodos *get* para obtener el valor del atributo y *set* para darle valor:



```
1
2 import java.io.Serializable;
3
4
5 public class Persona implements Serializable{
6     private String nombre;
7     private int edad;
8
9
10    public Persona(String nombre, int edad){
11        this.nombre = nombre;
12        this.edad = edad;
13    }
14
15    public Persona(){
16        this.nombre = null;
17    }
18
19    public void setNombre(String nombre){this.nombre = nombre; }
20    public void setEdad(int edad){this.edad = edad; }
21
22    public String getNombre(){return this.nombre;} //devuelve el nombre
23    public int getEdad(){return this.edad;} // devuelve la edad
24
25 }// fin de Persona
```

El siguiente ejemplo escribe objetos *Persona* en un fichero. Necesitamos crear un flujo de salida a disco con **FileOutputStream** y a continuación se crea el flujo de salida **ObjectOutputStream** que es el que procesa los datos y se ha de vincular al fichero de **FileOutputStream**:

```
FileOutputStream ficheroSalida = new FileOutputStream("FichPersona.dat");
ObjectOutputStream dataOS = new ObjectOutputStream(ficheroSalida);
```

El método **writeObject()** escribe los objetos al flujo de salida y los guarda en un fichero en disco: **dataOS.writeObject(persona)**. El código es el siguiente:

```
public class EscribirLeerFichObject {

    public static void main(String[] args){
        escribirFichObject();
        leerFichObject();
    } //main

    private static void escribirFichObject(){
        try{
            //Preparamos los flujos de salida para escribir
            FileOutputStream ficheroSalida = new FileOutputStream("FichPersona.dat");
            ObjectOutputStream dataOS = new ObjectOutputStream(ficheroSalida);

            String nombres[] = {"Ana", "Luis Miguel", "Alicia", "Pedro", "Manuel", "Andrés", "Julio", "Antonio", "María Jesús"};
            int edades[] = {14, 15, 13, 15, 16, 12, 16, 14, 13};

            Persona persona;

            for (int i = 0; i < edades.length; i++){ //recorro los arrays
                persona = new Persona(nombres[i], edades[i]);
                dataOS.writeObject(persona); //escribo la persona en el fichero
            } //fin for

            //Cerrar fichero
            dataOS.close();
            ficheroSalida.close();
        }
        catch(IOException erro){
            System.out.println("Error E/S "+erro.getMessage());
        }
        catch(Exception erro){
            System.out.println("Otro error "+erro.getMessage());
        }
    } //escribirFichObject
}
```

Para leer objetos *Persona* del fichero necesitamos el flujo de entrada a disco **FileInputStream** y a continuación crear el flujo de entrada **ObjectInputStream** que es el que procesa los datos y se ha de vincular al fichero de **FileInputStream**:

```
FileInputStream ficheroEntrada = new FileInputStream("FichPersona.dat");
ObjectInputStream entrada = new ObjectInputStream(ficheroEntrada);
```

El método **readObject** lee los objetos del flujo de entrada, puede lanzar la excepción **ClassNotFoundException** e **IOException**, por lo que será necesario controlarlas. El proceso de lectura se hace en un bucle *while (ficheroEntrada.available() > 0)*, *avai-*

`lable` es un método de `InputStream` que devuelve el número de bytes de entrada, de nuevo todo el código se encierra en un bloque **try-catch** para controlar los posibles errores que se puedan producir.

El código es el siguiente:

```
private static void leerFichObject(){
    try{
        //Preparamos los flujos de entrada para leer
        FileInputStream ficheroEntrada = new FileInputStream("FichPersona.dat");
        ObjectInputStream entrada = new ObjectInputStream(ficheroEntrada);

        //Leer los objetos del fichero
        Persona persona = null;
        while (ficheroEntrada.available() > 0){
            persona = (Persona) entrada.readObject();
            System.out.printf("Nombre: %s, edad: %d %n", persona.getNombre(), persona.getEdad());
        }
        //Cerrar fichero
        entrada.close();
        ficheroEntrada.close();
    }
    catch(IOException error){
        System.out.println("Error E/S " + error.getMessage());
    }
    catch(Exception error){
        System.out.println("Otro error " + error.getMessage());
    }
} //leerFichObject
```

Código método MAIN:

```
package escribirleerfichobject;
import java.io.*;

public class EscribirLeerFichObject {

    public static void main(String[] args){
        escribirFichObject();
        leerFichObject();
    } //main
```

Se obtiene la siguiente salida al ejecutar el programa:

```
run:
Nombre: Ana, edad: 14
Nombre: Luis Miguel, edad: 15
Nombre: Alicia, edad: 13
Nombre: Pedro, edad: 15
Nombre: Manuel, edad: 16
Nombre: Andrés, edad: 12
Nombre: Julio, edad: 16
Nombre: Antonio, edad: 14
Nombre: María Jesús, edad: 13
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.6.4. Ficheros de acceso aleatorio

Hasta ahora todas las operaciones que hemos realizado sobre los ficheros se realizaban de forma secuencial. Se empezaba la lectura en el primer byte o el primer carácter o el primer objeto, y seguidamente se leían los siguientes uno a continuación de otro hasta llegar al fin del fichero. Igualmente cuando escribíamos los datos en el fichero se iban escribiendo a continuación de la última información escrita. Java dispone de la clase **RandomAccessFile** que dispone de métodos para acceder al contenido de un fichero binario de forma aleatorio (no secuencial) y para posicionarnos en una posición concreta del mismo. Esta clase no es parte de la jerarquía **InputStream/OutputStream**, ya que su comportamiento es totalmente distinto puesto que se puede avanzar y retroceder dentro de un fichero.

Disponemos de dos constructores para crear el fichero de acceso aleatorio, estos pueden lanzar la excepción **FileNotFoundException**:

- **RandomAccessFile(String nombrefichero, String modoAcceso)**: escribiendo el nombre del fichero incluido el path.
- **RandomAccessFile(File objetoFile, String modoAcceso)**: con un objeto **File** asociado a un fichero.

El argumento *modoAcceso* puede tener dos valores:

| Modo de ACCESO | Significado |
|----------------|--|
| r | Abre el fichero en modo solo lectura. Si fichero debe existir. Una operación de escritura en este fichero lanzará la excepción IOException |
| rw | Abre el fichero en modo solo lectura y escritura. Si fichero no existe se crea. |

Una vez abierto el fichero pueden usarse los métodos *readXXX* y *writeXXX* de las clases **DataInputStream** y **DataOutputStream**. La clase **RandomAccessFile** maneja un puntero que indica la posición actual en el fichero. Cuando el fichero se crea el puntero al fichero se coloca en 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:

| Modo de ACCESO | Significado |
|---------------------------------|--|
| long getFilePointer() | Devuelve la posición actual del puntero del fichero. |
| void seek(long posición) | Coloca el puntero del fichero en una posición determinada desde el comienzo del mismo. |
| long length | Devuelve el tamaño del fichero en bytes. La posición |

| | |
|---|---|
| | length() marca el final del fichero |
| <code>int skipBytes (int desplazamiento)</code> | Desplaza el puntero desde la posición actual el numero de bytes indicados en desplazamiento |

El ejemplo que se muestra a continuación inserta datos de empleados en un fichero aleatorio.

Los datos a insertar: apellido, departamento y salario, se obtienen de varios arrays que se llenan en el programa, los datos se van introduciendo de forma secuencial por lo que no va a ser necesario usar el método `seek()`. Por cada empleado también se insertará un identificador (mayor que 0) que coincidirá con el índice +1 con el que se recorren los arrays. La longitud del registro de cada empleado es la misma (36 bytes) y los tipos que se insertan y su tamaño en bytes es el siguiente:

- Se inserta en primer lugar un entero, que es el identificador, ocupa 4 bytes.
- A continuación una cadena de 10 caracteres, es el apellido. Como Java utiliza caracteres UNICODE, cada carácter de una cadena de caracteres ocupa 16 bits (2 bytes), por tanto, el apellido ocupa 20 bytes.
- Un tipo entero que es el departamento, ocupa 4 bytes.
- Un tipo Double que es el salario, ocupa 8 bytes.

Tamaño de otros tipos: short (2 bytes), byte (1 byte), long (8 bytes), boolean (bit), float (4 bytes), etc.

El fichero se abre en modo "**rw**" para lectura y escritura. El código es el siguiente:

```
package ejemplol2escribirfichaleatorio;

import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;

public class Ejemplol2EscribirFichAleatorio {

    public static void main(String[] args) throws IOException {
        File fichero = new File("C:\\EJEMPLOS\\UNI2\\AleatorioEmple.dat");

        //declara el fichero de acceso aleatorio
        RandomAccessFile file = new RandomAccessFile( fichero, "rw");

        //arrays con los datos
        String apellido[] = {"FERNANDEZ", "GIL", "LOPEZ", "RAMOS", "SEVILLA", "CASILLA", "REY"}; //apellidos
        int dep[] = {10, 20, 10, 10, 30, 30, 20}; //departamentos
        Double salario[] = {1000.45, 2400.60, 3000.0, 1500.56, 2200.0, 1435.87, 2000.0}; //salarios

        StringBuffer buffer = null; //buffer para almacenar el apellido
        int n = apellido.length; //número de elementos del array
        for (int i = 0; i < n; i++) { //recorro los arrays
            file.writeInt(i+1); //uso i+1 para identificar empleado
            buffer = new StringBuffer(apellido[i]);
            buffer.setLength(10); //10 caracteres para el apellido
            file.writeChars(buffer.toString()); //insertar apellido

            file.writeInt(dep[i]); //insertar departamento
            file.writeDouble(salario[i]); //insertar salario
        }
        file.close();
    }
}
```

El siguiente ejemplo toma el fichero anterior y visualiza todos los registros. El posicionamiento para empezar a recorrer empieza en 0, para recuperar los siguientes registros hay que sumar 36 (tamaño del registro) a la variable utilizada para el posicionamiento:

```
package ejemplo13leerfichaleatorio;

import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;

public class Ejemplo13LeerFichAleatorio {

    public static void main(String[] args) throws IOException {
        File fichero = new File("C:\\EJEMPLOS\\UNI2\\AleatorioEmple.dat");

        //declara el fichero de acceso aleatorio
        RandomAccessFile file = new RandomAccessFile( fichero, "r");
        //
        int id, dep, posicion;
        Double salario;
        char apellido[] = new char[10], aux;

        posicion = 0; //para situarnos al principio

        for (;;) { //recorro el fichero
            file.seek(posicion); //nos posicionamos en posicion
            id = file.readInt(); //obtengo id de empleado

            //recorro uno a uno los caracteres del apellido
            for (int i = 0; i < apellido.length; i++){
                aux = file.readChar();
                apellido[i] = aux; //los voy guardando en el array
            }

            //convierto a String el array
            String apellidos = new String(apellido);
            dep = file.readInt(); //obtengo dep
            salario = file.readDouble(); //obtengo salario

            if (id > 0)
                System.out.printf("ID: %s, Apellido: %s, Departamento: %d,"
                                   + " Salario: %.2f %n", id, apellidos.trim(), dep, salario);
            //me posiciono para el siguiente empleado, cada empleado ocupa 36 bytes

            posicion = posicion + 36;

            //Si he recorrido todos los bytes salgo del for
            if (file.getFilePointer() == file.length()) break;
        } //fin bucle for
        file.close();
    }
}
```

La ejecución muestra la siguiente salida:

```
ID: 1, Apellido: FERNANDEZ, Departamento: 10, Salario: 1000,45
ID: 2, Apellido: GIL, Departamento: 20, Salario: 2400,60
ID: 3, Apellido: LOPEZ, Departamento: 10, Salario: 3000,00
ID: 4, Apellido: RAMOS, Departamento: 10, Salario: 1500,56
ID: 5, Apellido: SEVILLA, Departamento: 30, Salario: 2200,00
ID: 6, Apellido: CASILLA, Departamento: 30, Salario: 1435,87
ID: 7, Apellido: REY, Departamento: 20, Salario: 2000,00
```

Para consultar un empleado determinado no es necesario recorrer todos los registros del fichero, conociendo su identificador podemos acceder a la posición que ocupa dentro del mismo y obtener sus datos. Por ejemplo, supongamos que se desean obtener los datos del empleado con identificador 5, para calcular la posición hemos de tener en cuenta los bytes que ocupa cada registro (en este ejemplo son 36 bytes):

```
int identificador = 5;
//Calculo donde empieza el registro
posicion = (identificador - 1) * 36;

if(posicion >= file.length())
    System.out.printf("ID: %d, NO EXISTE EMPLEADO...", identificador);
else{
    file.seek(posicion); //nos posicionamos en posicion
    id = file.readInt(); //obtengo id de empleado

    //Obtener el resto de los datos como en el ejemplo anterior
```

Para añadir registros a partir del último insertado hemos de posicionar el puntero del fichero al final del mismo:

```
long posicion= file.length();
file.seek(posicion);
```

Para insertar un nuevo registro aplicamos la función al identificador para calcular la posición.

El siguiente ejemplo inserta un empleado con identificador 20, se ha de calcular la posición donde irá el registro dentro del fichero $(identificador - 1) * 36$ bytes:


```

StringBuffer buffer = null;           //buffer para almacenar el apellido
String apellido = "MIGUELEZ";        // apellido a insertar
Double salario = 1230.87;             //salario
int id = 30;                          //id del empleado
int dep = 10;                         //dep del empleado

long posicion = (id -1 ) * 36;        //calculamos la posición

file.seek(posicion);                 //nos posicionamos
file.writeInt(id);                   //se escribe id
buffer = new StringBuffer(apellido);
buffer.setLength(10);                //10 caracteres para el apellido
file.writeChars(buffer.toString());  //insertar apellido
file.writeInt(dep);                  //insertar departamento
file.writeDouble(salario);           //insertar salario

...

file.close();                        //cerrar fichero

```

1.7. ACTIVIDADES

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.

ACTIVIDAD 1.3

Consulta. Crea un programa Java que consulte los datos de un empleado del fichero aleatorio. El programa se ejecutará desde la línea de comandos y debe recibir un identificador de empleado. Si el empleado existe se visualizarán sus datos, si no existe se visualizará un mensaje indicándolo.

Insertión. Crea un programa Java que inserte datos en el fichero aleatorio. El programa se ejecutará desde la línea de comandos y debe recibir 4 parámetros: identificador de empleado, apellido, departamento y salario. Antes de insertar se comprobará si el identificador existe, en ese caso se debe visualizar un mensaje indicándolo; si no existe se deberá insertar.

Para modificar un registro determinado, accedemos a su posición y efectuamos las modificaciones. El fichero debe abrirse en modo "rw". Por ejemplo, para cambiar el departamento y salario del empleado con identificador 4 escribo lo siguiente:

```

int registro = 4;                      //id a modificar
long posicion = (registro -1 ) * 36;   //calculo la posición
posicion = posicion + 4 + 20;         //sumo el tamaño de ID + apellido
file.seek(posicion);                  //nos posicionamos
file.writeInt(40);                    //modifico departamento
file.writeDouble(4000.87);            //modifico salario

```

ACTIVIDAD 1.4

Modificación. Crea un programa Java que reciba desde la línea de comandos un identificador de empleado y un importe. Se debe realizar la modificación del salario. La modificación consistirá en sumar al salario del empleado el importe introducido. El programa debe visualizar el apellido, el salario antiguo y el nuevo. Si el identificador no existe se visualizará mensaje indicándolo.

Borrado. Crea un programa Java que al ejecutarlo desde la línea de comandos reciba un identificador de empleado y lo borre. Se hará un borrado lógico marcando el registro con la siguiente información: el identificador será igual a -1, el apellido será igual al identificador que se borra, y el departamento y salario serán 0.

A continuación haz otro programa Java (o crea un método dentro del anterior programa) que muestre los identificadores de los empleados borrados.
