CAPÍTULO 1

MANEJO DE FICHEROS

OBJETIVOS

Utilizar clases para la gestión de ficheros y directorios.

Valorar las ventajas y los inconvenientes de las distintas formas de acceso.

Utilizar las operaciones básicas para acceder a ficheros de acceso secuencial y aleatorio.

Utilizar clases para almacenar y recuperar información almacenada en un fichero XML.

Utilizar clases para convertir a otro formato información contenida en un fichero XML.

Gestionar excepciones.

Serializar objetos Java a representaciones XML.

CONTENIDOS

Clases asociadas a las operaciones de gestión de ficheros. Flujos o stream. Tipos.

Formas de acceso a un fichero.

Clases para gestión de flujos de datos desde/hacia ficheros.

Operaciones básicas sobre ficheros de acceso secuencial.

Operaciones básicas sobre ficheros de acceso aleatorio.

Ficheros XML. Librerías para conversión de documentos XML a otros formatos.

Excepciones: detección y tratamiento.

Introducción a JAXB.

RESUMEN

En este capítulo aprenderemos a a leer y escribir datos en ficheros secuenciales y directos en Java. Utilizaremos diferentes clases Java para el acceso a ficheros. Utilizaremos distintas librerías para conversión de ficheros XML a otros formatos. Aprenderemos a utilizar y gestionar excepciones. Utilizaremos la tecnología JAXB para serializar objetos Java a representaciones XML.

1.1. INTRODUCCIÓN

Un **fichero** o **archivo** es un conjunto de bits almacenados en un dispositivo, como por ejemplo, un disco duro. La ventaja de utilizar ficheros es que los datos que guardamos permanecen en el dispositivo aun cuando apaguemos el ordenador, es decir, no son volátiles. Los ficheros tienen un nombre y se ubican en directorios o carpetas, el nombre debe ser único en ese directorio; es decir, no puede haber dos ficheros con el mismo nombre en el mismo directorio. Por convención cuentan con diferentes extensiones que por lo general suelen ser de 3 letras (PDF, DOC, GIF, ...) y nos permiten saber el tipo de fichero.

Un fichero está formado por un conjunto de registros o líneas y cada registro por un conjunto de campos relacionados, por ejemplo, un fichero de empleados puede contener datos de los empleados de una empresa, un fichero de texto puede contener líneas de texto correspondientes a líneas impresas en una hoja de papel. La manera en que se agrupan los datos en el fichero depende completamente de la persona que lo diseñe.

En este tema aprenderemos a utilizar los ficheros con el lenguaje Java.

1.2. CLASES ASOCIADAS A LAS OPERACIONES DE GESTIÓN DE FICHEROS

El paquete **java.io** contiene las clases para manejar la entrada/salida en Java, por tanto, necesitaremos importar dicho paquete cuando trabajemos con ficheros. Antes de ver las clases que leen y escriben datos en ficheros vamos a manejar la clase **File**. Esta clase proporciona un conjunto de utilidades relacionadas con los ficheros que nos van a proporcionar información acerca de los mismos, su nombre, sus atributos, los directorios, etc. Puede representar el nombre de un fichero particular o los nombres de un conjunto de ficheros de un directorio, también se puede usar para crear un nuevo directorio o una trayectoria de directorios completa si esta no existe. Para crear un objeto **File**, se puede utilizar cualquiera de los tres constructores siguientes:

- **File(String directorioyfichero)**: en Linux: *new File("/directorio/fichero.txt");* en plataformas Microsoft Windows: *new File("C:\\directorio\\fichero.txt").*
- **File(String directorio, String nombrefichero)**: *new File("directorio", "fichero.txt").*
- **File(File directorio, String fichero)**: *new File(new File("directorio"), "fichero.txt").*

En Linux se utiliza como prefijo de una ruta absoluta "/". En Microsoft Windows, el prefijo de un nombre de ruta consiste en la letra de la unidad seguida de ":" y, posiblemente, seguida por "\\" si la ruta es absoluta.

Ejemplos de uso de la clase **File** donde se muestran diversas formas para declarar un fichero:

```
//Windows
File fichero1 = new File( "C:\\EJERCICIOS\\UNI1\\ejemplo1.txt");
//Linux
File fichero1 = new File( "/home/ejercicios/uni1/ejemplo1.txt");

String directorio= "C:/EJERCICIOS/UNI1";
File fichero2 = new File(directorio, "ejemplo2.txt");

File direc = new File(directorio);
```

```
File fichero3 = new File(direc, "ejemplo3.txt");
```

Algunos de los métodos más importantes de la clase **File** son los siguientes:

Método	Función
`String[] list()`	Devuelve un array de String con los nombres de ficheros y directorios asociados al objeto **File**
`File[] listFiles()`	Devuelve un array de objetos **File** conteniendo los ficheros que estén dentro del directorio representado por el objeto **File**
`String getName()`	Devuelve el nombre del fichero o directorio
`String getPath()`	Devuelve el camino relativo
`String getAbsolutePath()`	Devuelve el camino absoluto del fichero/directorio
`boolean exists()`	Devuelve *true* si el fichero/directorio existe
`boolean canWrite()`	Devuelve *true* si el fichero se puede escribir
`boolean canRead()`	Devuelve *true* si el fichero se puede leer
`boolean isFile()`	Devuelve *true* si el objeto **File** corresponde a un fichero normal
`boolean isDirectory()`	Devuelve *true* si el objeto **File** corresponde a un directorio
`long length()`	Devuelve el tamaño del fichero en bytes
`boolean mkdir()`	Crea un directorio con el nombre indicado en la creación del objeto **File**. Solo se creará si no existe
`boolean renameTo(File nuevonombre);`	Renombra el fichero representado por el objeto **File** asignándole *nuevonombre*
`boolean delete()`	Borra el fichero o directorio asociado al objeto **File**
`boolean createNewFile()`	Crea un nuevo fichero, vacío, asociado a **File** si y solo si no existe un fichero con dicho nombre
`String getParent()`	Devuelve el nombre del directorio padre, o *null* si no existe

El siguiente ejemplo muestra la lista de ficheros en el directorio actual. Se utiliza el método ***list()*** que devuelve un array de String con los nombres de los ficheros y directorios contenidos en el directorio asociado al objeto **File**. Para indicar que estamos en el directorio actual creamos un objeto **File** y le pasamos la variable *dir* con el valor ".". Se define un segundo objeto **File** utilizando el tercer constructor para saber si el fichero obtenido es un fichero o un directorio:

```
import java.io.*;
public class VerDir {
  public static void main(String[] args) {
    String dir = "."; //directorio actual
    File f = new File(dir);
    String[] archivos = f.list();
    System.out.printf("Ficheros en el directorio actual: %d %n",
                      archivos.length);
    for (int i = 0; i < archivos.length; i++) {
      File f2 = new File(f, archivos[i]);
      System.out.printf("Nombre: %s, es fichero?: %b, es directorio?:
                %b %n", archivos[i], f2.isFile(),f2.isDirectory());
    }
  }
}
```

Un ejemplo de ejecución de este programa mostraría la siguiente salida:

```
Ficheros en el directorio actual: 3
Nombre: VerDir.class, es fichero?: true, es directorio?: false
Nombre: VerDir.java, es fichero?: true, es directorio?: false
Nombre: VerInf.java, es fichero?: true, es directorio?: false
```

La siguiente declaración aplicada al ejemplo anterior mostraría la lista de ficheros del directorio *d:\db*:

```
File f = new File("d:\\db");
```

Con la siguiente declaración se mostraría la lista de ficheros del directorio introducido desde la línea de comandos al ejecutar el programa:

```
String dir=args[0];
System.out.println("Archivos en el directorio " +dir);
File f = new File(dir);
```

ACTIVIDAD 1.1

Realiza un programa Java que utilice el método **listFiles()** para mostrar la lista de ficheros en un directorio cualquiera, o en el directorio actual.

Realiza un programa Java que muestre los ficheros de un directorio. El nombre del directorio se pasará al programa desde los argumentos de *main()*. Si el directorio no existe se debe mostrar un mensaje indicándolo.

El siguiente ejemplo muestra información del fichero *VerInf.java*:

```
import java.io.*;
public class VerInf {
public static void main(String[] args) {
  System.out.println("INFORMACIÓN SOBRE EL FICHERO:");
  File f = new File("D:\\ADAT\\UNI1\\VerInf.java");
  if(f.exists()){
    System.out.println("Nombre del fichero  : "+f.getName());
    System.out.println("Ruta               : "+f.getPath());
    System.out.println("Ruta absoluta      : "+f.getAbsolutePath());
    System.out.println("Se puede leer      : "+f.canRead());
    System.out.println("Se puede escribir  : "+f.canWrite());
    System.out.println("Tamaño             : "+f.length());
    System.out.println("Es un directorio   : "+f.isDirectory());
    System.out.println("Es un fichero      : "+f.isFile());
    System.out.println("Nombre del directorio padre: "+f.getParent());
  }
 }
}
```

Visualiza la siguiente información del fichero:

```
INFORMACIÓN SOBRE EL FICHERO:
Nombre del fichero  : VerInf.java
Ruta                : D:\ADAT\UNI1\VerInf.java
Ruta absoluta       : D:\ADAT\UNI1\VerInf.java
Se puede leer       : true
Se puede escribir   : true
```

```
Tamaño                : 824
Es un directorio      : false
Es un fichero         : true
Nombre del directorio padre: D:\ADAT\UNI1
```

El siguiente ejemplo crea un directorio (de nombre *NUEVODIR*) en el directorio actual, a continuación crea dos ficheros vacíos en dicho directorio y uno de ellos lo renombra. En este caso para crear los ficheros se definen 2 parámetros en el objeto **File**: *File(File directorio, String nombrefich)*, en el primero indicamos el directorio donde se creará el fichero y en el segundo indicamos el nombre del fichero:

```
import java.io.*;
public class CrearDir {
 public static void main(String[] args) {
   File d = new File("NUEVODIR"); //directorio que creo
   File f1 = new File(d,"FICHERO1.TXT");
   File f2 = new File(d,"FICHERO2.TXT");

   d.mkdir();//CREAR DIRECTORIO

   try {
    if (f1.createNewFile())
       System.out.println("FICHERO1 creado correctamente...");
    else
       System.out.println("No se ha podido crear FICHERO1...");

    if (f2.createNewFile())
       System.out.println("FICHERO2 creado correctamente...");
    else
       System.out.println("No se ha podido crear FICHERO2...");
   } catch (IOException ioe) {ioe.printStackTrace();}

   f1.renameTo(new File(d,"FICHERO1NUEVO"));//renombro FICHERO1

   try {
        File f3 = new File("NUEVODIR/FICHERO3.TXT");
        f3.createNewFile();//crea FICHERO3 en NUEVODIR
   } catch (IOException ioe) {ioe.printStackTrace();}
 }
}
```

Para borrar un fichero o un directorio usamos el método ***delete()***, en el ejemplo anterior no podemos borrar el directorio creado porque contiene ficheros, antes habría que eliminar estos ficheros. Para borrar el objeto *f2* escribimos:

```
if(f2.delete())
  System.out.println("Fichero borrado...");
else
  System.out.println("No se ha podido borrar el fichero...");
```

El método **createNewFile()** puede lanzar la excepción ***IOException***, por ello se utiliza el bloque **try-catch**.