**TEMA 11: ENTRADA Y SALIDA EN JAVA. FICHEROS**

**INTRODUCCIÓN**

Todas las estructuras comentadas en los temas anteriores para almacenar los datos, residen en memoria. De hecho cuando el programa termina, desaparecen.

Para evitar este problema existen los archivos. Se trata de un elemento que pone a disposición el Sistema Operativo para almacenar información de todo tipo bajo un **nombre** y una **extensión**. La extensión indica el tipo de archivo que tenemos.

En definitiva es la estructura habitual en la que se almacena información y cuyo manejo parte de que sabemos sus fundamentos en base al sistema operativo que estemos manejando. Su finalidad es la de poder almacenar datos de forma permanente.

**ENTRADA/SALIDA EN JAVA**

El paquete **java.io** contiene todas las clases relacionadas con las funciones de entrada (**input**) y salida (**output**). La E/S se maneja en Java a través de Streams o flujos. Un Stream de Entrada o Lectura sirve para extraer datos almacenados en un fichero, mientras que la escritura es la operación contraria, es decir, se trata de almacenar datos en un fichero.

|  |  |  |
| --- | --- | --- |
| Escritura | http://www.saulo.net/pub/ddypart/Disco3.GIF | Lectura |

**clase File**

En el paquete **java.io** se encuentra la clase **File** pensada para poder realizar operaciones sobre archivos. No proporciona métodos de acceso a los archivos, sino operaciones a nivel de sistema de archivos (listado de archivos, crear carpetas, borrar ficheros, cambiar nombre, etc.).

Un objeto **File** representa un archivo o un directorio y sirve para obtener información (permisos, tamaño,…). También sirve para navegar por la estructura de archivos.

La clase File tiene los siguientes constructores:

File (String fichero);

File (String directorio o path);

File (String directorio, String fichero);

File (File directorio, String directorio);

La ruta o path puede ser absoluta o relativa.

**Ejemplos utilizando los constructores 1 y 2:**

1. Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio de trabajo:

File f = new File("personas.dat");

En este caso no se indica path. Se supone que el fichero se encuentra en el directorio actual de trabajo.

2. Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio ficheros dentro del directorio actual.

File f = new File("ficheros/personas.dat");

En este caso se indica la ruta relativa tomando como base el directorio actual de trabajo. Se supone que el fichero personas.dat se encuentra en el directorio ficheros. A su vez el directorio ficheros se encuentra dentro del directorio actual de trabajo.

3. Crea un Objeto File asociado al fichero personas.dat dando la ruta absoluta:

File f = new File("c:/ficheros/personas.dat");

El fichero se encuentra en el directorio ficheros. A su vez el directorio ficheros se encuentra en la raíz de la unidad C:

Si se omite la letra de la unidad, por defecto se asume la letra de la unidad en la que se encuentra el proyecto:

File f = new File("/ficheros/personas.dat");

**Ejemplos utilizando los constructores 3 y 4:**

En este caso se crea un objeto File cuya ruta (absoluta o relativa) se indica en el primer String.

1. Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio ficheros dentro del directorio actual.

File f = new File("ficheros", "personas.dat" );

En este caso se indica la ruta relativa tomando como base el directorio actual de trabajo.

2. Crea un Objeto File asociado al fichero personas.dat dando la ruta absoluta:

File f = new File("/ficheros", "personas.dat" );

En este caso se indica la ruta absoluta, indicada por la barra del principio.

Este constructor permite crear un objeto File cuya ruta se indica a través de otro objeto File.

1. Crea un Objeto File asociado al fichero personas.dat que se encuentra en el directorio ficheros dentro del directorio actual.

File ruta = new File("ficheros");

File f = new File(ruta, "personas.dat" );

2. Crea un Objeto File asociado al fichero personas.dat dando la ruta absoluta:

File ruta = new File("/ficheros");

File f = new File(ruta, "personas.dat" );

Debemos tener en cuenta que crear un objeto File no significa que deba existir el fichero o el directorio o que el path sea correcto.

Si no existen no se lanzará ningún tipo de excepción ni tampoco serán creados.

**Construcción de objetos de archivo**

Utiliza como único argumento una cadena que representa una ruta en el sistema de archivo. También puede recibir, opcionalmente, un segundo parámetro con una ruta segunda que se define a partir de la posición de la primera.

**File** archivo1=**new File**(“/datos/bd.txt”);

**File** carpeta=**new File**(“datos”);

El primer formato utiliza una ruta absoluta y el segundo una ruta relativa. En Java el separador de archivos tanto para Windows como para Linux es el símbolo **/**.

Otra posibilidad de construcción es utilizar como primer parámetro un objeto **File** ya hecho. A esto se añade un segundo parámetro que es una ruta que cuenta desde la posición actual.

**File** carpeta1=**new File**(“c:/datos”);***//ó c\\datos***

**File** archivo1=**new File**(carpeta1,”bd.txt”);

Si el archivo o carpeta que se intenta examinar no existe, la clase **File** no devuelve una excepción. Habrá que utilizar el método **exists**. Este método devuelve **true** si la carpeta o archivo es válido.

*Cuando creamos una instancia de File, lo que estamos haciendo es establecer un enlace con un archivo o directorio físico al que luego le podremos consultar todas sus propiedades*

**Métodos generales de la clase File:**

mét

**String toString()** Para obtener la cadena descriptiva del objeto.

**boolean exists()** Devuelve **true** si existe la carpeta o archivo.

**boolean canRead()** Devuelve **true** si el archivo se puede leer

**boolean canWrite()** Devuelve **true** si el archivo se puede escribir

**boolean isHidden()** Devuelve **true** si el objeto File es oculto

**boolean isAbsolute()** Devuelve **true** si la ruta indicada en el objeto File es absoluta

**boolean equals(File** f2**)** Compara ***f2*** con el objeto **File** y devuelve verdadero si son iguales.

**int compareTo(File** f2**)** Compara basado en el orden alfabético del texto (sólo funciona bien si ambos archivos son de texto) ***f2*** con el objeto **File** y devuelve cero si son iguales, un entero negativo si el orden de ***f2*** es mayor y positivo si es menor.

**String getAbsolutePath()** Devuelve una cadena con la ruta absoluta al objeto File.

**File getAbsoluteFile()** Como la anterior pero el resultado es un objeto File.

**String getName()** Devuelve el nombre del objeto File.

**String getParent()** Devuelve el nombre de su carpeta superior si la hay y si no **null.**

**File getParentFile()** Como la anterior pero la respuesta se obtiene en forma de objeto File.

**boolean setReadOnly()** Activa el atributo de sólo lectura en la carpeta o archivo.

Ejemplo

package clasefile;

import java.io.File;

import java.io.IOException;

public class File1 {

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

File ruta = new File("src\\clasefile");

File f = new File(ruta, "datos.txt");

System.out.println(f.getAbsolutePath());

System.out.println(f.getParent());

System.out.println(ruta.getAbsolutePath());

System.out.println(ruta.getParent());

if (!f.exists()) { //se comprueba si el fichero existe o no

System.out.println("Fichero " + f.getName() + " no existe");

if (!ruta.exists()) { //se comprueba si la ruta existe o no

System.out.println("El directorio " + ruta.getName() + " no existe");

if (ruta.mkdir()) { //se crea la ruta. Si se ha creado correctamente

System.out.println("Directorio creado");

if (f.createNewFile()) { //se crea el fichero. Si se ha creado correctamente

System.out.println("Fichero " + f.getName() + " creado");

} else {

System.out.println("No se ha podido crear " + f.getName());

}

} else {

System.out.println("No se ha podido crear " + ruta.getName());

}

} else { //si la ruta existe creamos el fichero

if (f.createNewFile()) {

System.out.println("Fichero " + f.getName() + " creado");

} else {

System.out.println("No se ha podido crear " + f.getName());

}

}

} else { //el fichero existe. Mostramos el tamaño

System.out.println("Fichero " + f.getName() + " existe");

System.out.println("Tamaño " + f.length() + " bytes");

}

}

}

* Salida la primera vez que se ejecuta:

RUTA ABSOLUTA DEL OBJETO f:

C:\Users\Carmen\Documents\NetBeans14-15\Ficheros\src\clasefile\datos.txt

CARPETA EN LA QUE SE ENCUENTRA f:

src\clasefile

RUTA ABSOLUTA:

C:\Users\Carmen\Documents\NetBeans14-15\Ficheros\src\clasefile

CARPETAPADRE:

src

Fichero datos.txt no existe

Fichero datos.txt creado

La segunda vez ya ha creado el fichero con **f.createNewFile()** y la salida es

................................................................................................................................

.................................................

Fichero datos.txt existe

Tamaño 0 bytes

Igualmente podríamos poner una ruta que no existiese y nos la crearía. PROBARLO directamente en NetBeans

Ejemplo de programa que visualiza los atributos de un fichero:

package clasefile;

import java.io.\*;

import java.util.\*;

public class **Atributos**

{

public static void main (String [] args)

{

File f = null;

String nombfich;

Scanner tec = new Scanner(System.in);

System.out.println("Dame fichero");

nombfich = tec.nextLine();

f = new File(nombfich);

if(f.exists())

{

System.out.println("El fichero existe");

System.out.println("Tamano:" + f.length());

if(f.canRead() && f.canWrite())

{

System.out.println("Tiene permisos de lectura y escritura");

if(f.isDirectory())

{

String cadenas[] = f.list();

System.out.println("El contenido del directorio "+f.getName()+ " es");

for (String elem : cadenas)

System.out.println(elem);

System.out.println("Otra forma =>El contenido del directorio "+f.getPath()+ " es");

for(int i = 0; i < cadenas.length; i++)

System.out.println(cadenas[i]);

}

else

{

System.out.println("no es un directorio");

System.out.println("Tamano:" + f.length());

}

}

else

System.out.println("No tiene permisos de lectura y escritura");

}

else

System.out.println("Fichero inexistente");

}

}

**Métodos de carpetas**

**boolean isDirectory()** Devuelve **true** si el objeto File es una carpeta y **false** si es un archivo o si no existe.

**boolean mkdir()** Intenta crear una carpeta y devuelve **true** si fue posible hacerlo

**boolean mkdirs()** Usa el objeto para crear una carpeta con la ruta creada para el objeto y si hace falta crea toda la estructura de carpetas necesaria para crearla.

**boolean delete()** Borra la carpeta y devuelve **true** si puede hacerlo.

**String[] list()** Devuelve la lista de archivos de la carpeta representada en el objeto File.

**static File[] listRoots()** Devuelve un array de objetos File, donde cada objeto del array representa la carpeta raíz de una unidad de disco.

**File[] listfiles()** Igual que la anterior, pero el resultado es un array de objetos File.

/\*

\*Utilizar los métodos de la clase File para indicar las propiedades del archivo

\* autoexec.bat en un sistema Windows y para listar el contenido del directorio

\* actual.

\*/

package clasefile;

import java.io.\*;

public class **CFile** {

public static void main(String[] args) {

File f1, f2, f3;

String[] listaArchivos;

int i;

f1 = new File("c:\\autoexec.bat");

f2 = new File("c:\\carmen\\carmen.txt"); //No crea físicamente el archivo.

f3 = new File(".");

System.out.println("Nombre(f1): " + f1.getName());

System.out.println("Path(f1): " + f1.getPath());

System.out.println("Absolute Path(f1): " + f1.getAbsolutePath());

System.out.println("Parent(f1): " + f1.getParent());

System.out.println("Existe(f2): " + f2.exists());

System.out.println("Permiso de escritura(f1): " + f1.canWrite());

System.out.println("Permiso de lectura(f1): " + f1.canRead());

System.out.println("Permiso de ejecución(f1): " + f1.canExecute());

System.out.println("Es directorio(f1): " + f1.isDirectory());

System.out.println("Es archivo(f1): " + f1.isFile());

System.out.println("Es absoluto(f1): " + f1.isAbsolute());

System.out.println("Tamaño(f1): " + f1.length());

listaArchivos = f3.list();

//Si f3 no es un directorio da error

System.out.println("Contenido del directorio actual");

for(i=0; i<listaArchivos.length; i++)

{

System.out.println("listaArchivos["+i+"] = " + listaArchivos[i]);

}

}

}

**SALIDA**

run:

Nombre(f1): autoexec.bat

Path(f1): c:\autoexec.bat

Absolute Path(f1): c:\autoexec.bat

Parent(f1): c:\

Existe(f2): false

Permiso de escritura(f1): true

Permiso de lectura(f1): true

Permiso de ejecución(f1): true

Es directorio(f1): false

Es archivo(f1): true

Es absoluto(f1): true

Tamaño(f1): 24

Contenido del directorio actual, es el proyecto donde se han creado las carpetas

listaArchivos[0] = build

listaArchivos[1] = build.xml

listaArchivos[2] = datos

listaArchivos[3] = datosf2

listaArchivos[4] = fich.txt

listaArchivos[5] = fich1.txt

listaArchivos[6] = manifest.mf

listaArchivos[7] = nbproject

listaArchivos[8] = prueba.txt

listaArchivos[9] = prueba2

listaArchivos[10] = src

listaArchivos[11] = test

Ejemplo de visualización del contenido de un directorio:

package clasefile;

import java.util.\*;

import java.io.\*;

public class **VerDirectorio2** {

public static void main(String args[]){

String nombreF;

InputStreamReader f=new InputStreamReader(System.in);

Scanner teclado = new Scanner(System.in);

System.out.println( "Nombre del directorio");

nombreF = teclado.nextLine();

File fc = new File( nombreF );

if (fc.exists()) //busca en el directorio actual

{

if (!fc.isDirectory())

{

System.out.println("No es un directorio");

}

else

{

System.out.println("Es un directorio");

File [] lista = fc.listFiles();

for(int i=0; i<lista.length;i++)

System.out.println(lista[i]);

}

}

else

System.out.println("Fichero no existente");

}

}

**Métodos de archivos**

**boolean isFile()** Devuelve **true** si el objeto File es un archivo y **false** si es carpeta o si no existe.

**boolean renameTo(File** f2**)** Cambia el nombre del archivo por el que posee el archivo pasado como argumento. Devuelve **true** si se pudo completar la operación.

**boolean delete()** Borra el archivo y devuelve **true** si puede hacerlo.

**long length()** Devuelve el tamaño del archivo en bytes (en el caso del texto devuelve los caracteres del archivo).

**boolean createNewFile() throws IOException** Crea un nuevo archivo basado en la ruta dada al objeto File. Hay que capturar la excepción *IOException* que ocurriría si hubo error crítico al crear el archivo. Devuelve **true** si se hizo la creación del archivo

vacío y **false** si ya había otro archivo con ese nombre.

**static File createTempFile(String** prefijo**, String** sufijo**) throws IOException** Crea un objeto File de tipo archivo temporal con el prefijo y sufijo indicados. Se creará en la carpeta de archivos temporales por defecto del sistema. El ***prefijo*** y el ***sufijo*** deben de tener al menos tres caracteres (el sufijo suele ser la extensión), de otro modo se produce una excepción del tipo **IllegalArgumentsException** Requiere capturar la excepción **IOException** que se produce ante cualquier fallo en la creación del archive.

**static File createTempFile( String** prefijo**, String** sufijo**, File** directorio**)** Igual que el anterior, pero utiliza el directorio indicado.

**void deleteOnExit()** Borra el archivo cuando finaliza la ejecución del programa