10.D. Aplicaciones del almacenamiento de información en ficheros.

Sitio:	VIRGEN DE LA PAZ	Imprimido por:	Cristian Esteban Gómez
Curso:	Programación	Día:	miércoles, 31 de mayo de 2023, 11:25
Libro:	10.D. Aplicaciones del almacenamiento de información en ficheros.		

Tabla de contenidos

1. Aplicaciones del almacenamiento de información en ficheros.

¿Has pensado la **diversidad de ficheros** que existe, según la información que se guarda?

Las fotos que haces con tu cámara digital, o con el móvil, se guardan en ficheros. Así, existe una gran cantidad de ficheros de imagen, según el método que usen para guardar la información. Por ejemplo, tenemos los ficheros de extensión: .jpg, .tiff, gif, .bmp, etc.

La música que oyes en tu mp3 o en el reproductor de mp3 de tu coche, está almacenada en ficheros que almacenan la información en formato mp3.

Los sistemas operativos, como Linux, Windows, etc., están constituidos por un montón de instrucciones e información que se guarda en ficheros.

El propio código fuente de los lenguajes de programación, como Java, C, etc., se guarda en ficheros de texto plano la mayoría de veces.

También se guarda en ficheros las películas en formato .avi, .mp4, etc.

Y por supuesto, se usan mucho actualmente los ficheros XML, que al fin y al cabo son ficheros de texto plano, pero que siguen una estructura determinada. XML se emplea mucho para el intercambio de información estructurada entre diferentes plataformas.

Para saber más

En el siguiente enlace a la wikipedia puedes saber más sobre el formato Mp3.

Formato mp3.

Autoevaluación

Indica si es verdadera o falsa la siguiente afirmación:

Un fichero .bmp guarda información de música codificada. ¿Verdadero o falso?

○ Verdadero.
○ Falso.

2. Utilización de los sistemas de ficheros.

Has visto en los apartados anteriores cómo operar en ficheros: abrirlos, cerrarlos, escribir en ellos, etc.

Lo que no hemos visto es lo relativo a crear y borrar directorios, poder filtrar archivos, es decir, buscar sólo aquellos que tengan determinada característica, por ejemplo, que su extensión sea: .txt.

Ahora veremos cómo hacer estas cosas, y también como borrar ficheros, y crearlos, aunque crearlos ya lo hemos visto en algunos ejemplos anteriores.

Para saber más

Accediendo a este enlace, tendrás una visión detallada sobre la organización de ficheros.

Organización de Ficheros y Métodos de Enlace

2.1. Clase File.

La clase `File` proporciona una representación abstracta de ficheros y directorios.

Esta clase, permite examinar y manipular archivos y directorios, independientemente de la plataforma en la que se esté trabajando: Linux, Windows, etc.

Las instancias de la clase `File` representan nombres de archivo, no los archivos en sí mismos.

El archivo correspondiente a un nombre dado podría ser que no existiera, por ello, habrá que controlar las posibles excepciones.

Al trabajar con File, las rutas pueden ser:

- Relativas al directorio actual.

- Absolutas si la ruta que le pasamos como parámetro empieza por
  - La barra "/" en Unix, Linux.
  - Letra de unidad (C:, D:, etc.) en Windows.
  - UNC(universal naming convention) en windows, como por ejemplo:

```
File miFile=new File("\\\\mimaquina\\download\\prueba.txt");
```

A través del objeto `File`, un programa puede examinar los atributos del archivo, cambiar su nombre, borrarlo o cambiar sus permisos. Dado un objeto `file`, podemos hacer las siguientes operaciones con él:

- **Renombrar** el archivo, con el método `renameTo`(). El objeto `File` dejará de referirse al archivo renombrado, ya que el `String` con el nombre del archivo en el objeto `File` no cambia.

- **Borrar** el archivo, con el método `delete`(). También, con `deleteOnExit()` se borra cuando finaliza la ejecución de la máquina virtual Java.

- **Crear** un nuevo fichero con un nombre único. El método estático `createTempFile()` crea un fichero temporal y devuelve un objeto `File` que apunta a él. Es útil para crear archivos temporales, que luego se borran, asegurándonos tener un nombre de archivo no repetido.

- **Establecer** la fecha y la hora de modificación del archivo con `setLastModified`(). Por ejemplo, se podría hacer: `new File("prueba.txt").setLastModified(new Date().getTime());` para establecerle la fecha actual al fichero que se le pasa como parámetro, en este caso *prueba.txt*.

- **Crear** un directorio con el método `mkdir`(). También existe `mkdirs`(), que crea los directorios superiores si no existen.

- **Listar** el contenido de un directorio. Los métodos `list()` y `listFiles()` listan el contenido de un directorio `list()` devuelve un vector de `String` con los nombres de los archivos, `listFiles()` devuelve un vector de objetos `File`.

- **Listar** los nombres de archivo de la raíz del sistema de archivos, mediante el método estático `listRoots()`.

Autoevaluación

Indica si es verdadera o falsa la siguiente afirmación:

Un objeto de la clase File representa un fichero en sí mismo. ¿Verdadero o falso?

○ Verdadero.
○ Falso.

2.2. Interface FilenameFilter.

En ocasiones nos interesa ver la lista de los archivos que encajan con un determinado criterio.

Así, nos puede interesar un filtro para ver los ficheros modificados después de una fecha, o los que tienen un tamaño mayor del que indiquemos, etc.

El interface `FilenameFilter` se puede usar para crear filtros que establezcan criterios de filtrado relativos al nombre de los ficheros. Una clase que lo implemente debe definir e implementar el método:

```
boolean accept(File dir, String nombre)
```

Este método devolverá verdadero (`true`), en el caso de que el fichero cuyo nombre se indica en el parámetro `nombre` aparezca en la lista de los ficheros del directorio indicado por el parámetro `dir`.

En el siguiente ejemplo vemos cómo se listan los ficheros de la carpeta c:\datos que tengan la extensión .odt. Usamos `try` y `catch` para capturar las posibles excepciones, como que no exista dicha carpeta.

```
public class Filtro implements FilenameFilter {
    String extension;
    Filtro(String extension){
        this.extension=extension;
    }
    public boolean accept(File dir, String name){
        return name.endsWith(extension);
    }

    public static void main(String[] args) {
        try {
            File fichero=new File("c:\\datos\\.");
            String[] listadeArchivos = fichero.list();
            listadeArchivos = fichero.list(new Filtrar(".odt"));
            int numarchivos = listadeArchivos.length ;

            if (numarchivos < 1)
                System.out.println("No hay archivos que listar");
            else
            {
                for(int conta = 0; conta < listadeArchivos.length; conta++)
                    System.out.println(listadeArchivos[conta]);
            }
        }
        catch (Exception ex) {
            System.out.println("Error al buscar en la ruta indicada");
        }
    }
}
```

Mismo código copiable:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package filtrarficheros;
import java.io.File;
import java.io.FilenameFilter;

public class Filtro implements FilenameFilter {
    String extension;
    Filtro(String extension){
```

```
        this.extension=extension;
    }
    public boolean accept(File dir, String name){
        return name.endsWith(extension);
    }

    public static void main(String[] args) {
        try {
            File fichero=new File("c:\\datos\\.");
            String[] listadeArchivos = fichero.list();
            listadeArchivos = fichero.list(new Filtrar(".odt"));
            int numarchivos = listadeArchivos.length ;

            if (numarchivos < 1)
                System.out.println("No hay archivos que listar");
            else
            {
                for(int conta = 0; conta < listadeArchivos.length; conta++)
                    System.out.println(listadeArchivos[conta]);
            }
        }
        catch (Exception ex) {
            System.out.println("Error al buscar en la ruta indicada");
        }
    }
}
```

Autoevaluación

Indica si la siguiente afirmación es verdadera o falsa:

Una clase que implemente FileNameFilter puede o no implementar el método accept. ¿Verdadero o Falso?

○ Verdadero.
○ Falso.

2.3. Creación y eliminación de ficheros y directorios.

Podemos **crear un fichero** del siguiente modo:

- Creamos el objeto que encapsula el fichero, por ejemplo, suponiendo que vamos a crear un fichero llamado miFichero.txt, en la carpeta C:\\prueba, haríamos:

```
File fichero = new File("c:\\prueba\\miFichero.txt");
```

- A partir del objeto `File` creamos el fichero físicamente, con la siguiente instrucción, que devuelve un `boolean` con valor `true` si se creó correctamente, o `false` si no se pudo crear:

```
fichero.createNewFile()
```

Para **borrar un fichero**, podemos usar la clase `File`, comprobando previamente si existe, del siguiente modo:

- Fijamos el nombre de la carpeta y del fichero con:

```
File fichero = new File("C:\\prueba", "agenda.txt");
```

- Comprobamos si existe el fichero con `exists()` y si es así lo borramos con:

```
fichero.delete();
```

Para **crear directorios**, podríamos hacer:

```
package crearcarpetas;

import java.io.File;
/**
 *
 * @author JJBH
 */
public class CrearDirec {

    public static void main(String[] args) {
      try {
          // Declaración de variables
          String directorio = "C:\\micarpeta";
          String varios = "carpeta1/carpeta2/carpeta3";

          // Crear un directorio
          boolean exito = (new File(directorio)).mkdir();
          if (exito)
              System.out.println("Directorio: " + directorio + " creado");
          // Crear varios directorios
          exito = (new File(varios)).mkdirs();
          if (exito)
              System.out.println("Directorios: " + varios + " creados");
        }catch (Exception e){
                System.err.println("Error: " + e.getMessage());
        }
    }

}
```

Mismo código copiable:

package crearcarpetas;

import java.io.File;

/**

```
 *
 * @author JJBH
 */
public class CrearDirec {


   public static void main(String[] args) {
    try {
       // Declaración de variables
       String directorio = "C:\\micarpeta";
       String varios = "carpeta1/carpeta2/carpeta3";


       // Crear un directorio
       boolean exito = (new File(directorio)).mkdir();
       if (exito)
          System.out.println("Directorio: " + directorio + " creado");
       // Crear varios directorios
       exito = (new File(varios)).mkdirs();
       if (exito)
          System.out.println("Directorios: " + varios + " creados");
        }catch (Exception e){
                System.err.println("Error: " + e.getMessage());
        }
   }
}
```

Para borrar un directorio con Java tenemos que borrar cada uno de los ficheros y directorios que éste contenga. Al poder almacenar otros directorios, se podría recorrer recursivamente el directorio para ir borrando todos los ficheros.

Se puede listar el contenido del directorio con:

```
File[] ficheros = directorio.listFiles();
```

y entonces poder ir borrando. Si el elemento es un directorio, lo sabemos mediante el método `isDirectory`,