Clase File

La clase File se usa para **obtener información** sobre archivos y directorios. Además la clase File permite crear y eliminar archivos y directorios.

Un objeto de la clase Java File **representa** un archivo o directorio.

CONSTRUCTORES

La clase proporciona los siguientes constructores para crear objetos File:

```
public File(String nombreFichero|path);
public File(String path, String nombreFichero|path);
public File(File path, String nombreFichero|path);
```

La ruta o **path** puede ser absoluta o relativa.

Ejemplos utilizando el primer constructor:

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 el segundo constructor:

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.

Ejemplos utilizando el tercer constructor:

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.

MÉTODOS

Algunos métodos de la clase File son los siguientes:

MÉTODO	DESCRIPCIÓN
boolean canRead()	Devuelve true si se puede leer el fichero
boolean canWrite()	Devuelve true si se puede escribir en el fichero
boolean createNewFile()	Crea el fichero asociado al objeto File. Devuelve true si se ha podido crear. Para poder crearlo el fichero no debe existir. Lanza una excepción del tipo **IOException**.
boolean delete()	Elimina el fichero o directorio. Si es un directorio debe estar vacío. Devuelve true si se ha podido eliminar.
boolean exists()	Devuelve true si el fichero o directorio existe
String getName()	Devuelve el nombre del fichero o directorio
String getAbsolutePath()	Devuelve la ruta absoluta asociada al objeto File.
String getCanonicalPath()	Devuelve la ruta única absoluta asociada al objeto File. Puede haber varias rutas absolutas asociadas a un File pero solo una única ruta canónica. Lanza una excepción del tipo **IOException**.
String getPath()	Devuelve la ruta con la que se creó el objeto File. Puede ser relativa o no.
String getParent()	Devuelve un String conteniendo el directorio padre del File.

	Devuelve null si no tiene directorio padre.
File getParentFile()	Devuelve un objeto File conteniendo el directorio padre del File. Devuelve null si no tiene directorio padre.
boolean isAbsolute()	Devuelve true si es una ruta absoluta
boolean isDirectory()	Devuelve true si es un directorio válido
boolean isFile()	Devuelve true si es un fichero válido
long lastModified()	Devuelve un valor en milisegundos que representa la última vez que se ha modificado (medido desde las 00:00:00 GMT, del 1 de Enero de 1970). Devuelve 0 si el fichero no existe o ha ocurrido un error.
long length()	Devuelve el tamaño en bytes del fichero. Devuelve 0 si no existe. Devuelve un valor indeterminado si es un directorio.
String[] list()	Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File. Si no es un directorio devuelve null. Si el directorio está vacío devuelve un array vacío.
String[] list(FilenameFilter filtro)	Similar al anterior. Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File que cumplen con el filtro indicado.
boolean mkdir()	Crea el directorio. Devuelve true si se ha podido crear.
boolean mkdirs()	Crea el directorio incluyendo los directorios no existentes especificados en la ruta *padre* del directorio a crear. Devuelve true si se ha creado el directorio y los directorios no existentes de la ruta padre.
boolean renameTo(File dest)	Cambia el nombre del fichero por el indicado en el parámetro dest. Devuelve true si se ha realizado el cambio.

Los puedes consultar todos en la API de Java:

http://docs.oracle.com/javase/8/docs/api/java/io/File.html

EJEMPLOS DE UTILIZACIÓN DE LA CLASE JAVA FILE

Ejemplo 1: El siguiente programa muestra el uso de algunos métodos de la clase File.

Se crea un objeto File *ruta* asociado al directorio *c:/ficheros* y un objeto File *f* asociado al fichero *datos.txt* que se encuentra en ese directorio.

Si el fichero no existe se crea y si el directorio no existe se crea y a continuación se crea el fichero. Si el fichero existe se muestra el tamaño del mismo.

```
import java.io.File;
import java.io.IOException;

public class File2 {

   public static void main(String[] args) throws IOException {
      File ruta = new File("c:/ficheros");
```

```
        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");
        }
    }
}
```

Si suponemos que no existe el fichero ni el directorio la ejecución del programa produce la siguiente salida:

```
c:\ficheros\datos.txt
c:\ficheros
```

```
c:\ficheros
c:\
Fichero datos.txt no existe
El directorio ficheros no existe
Directorio creado
Fichero datos.txt creado
```

Si volvemos a ejecutar el programa después de crear la ruta y el fichero, se muestra:

```
c:\ficheros\datos.txt
c:\ficheros
c:\ficheros
c:\
Fichero datos.txt existe
Tamaño 0 bytes
```

Ejemplo 2: El siguiente programa muestra como eliminar un fichero y como cambiar el nombre de un fichero usando la clase File.

```
import java.io.File;
import java.util.Scanner;
public class File3 {


   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String nombre;

      //Eliminar un fichero
      System.out.println("Introduce el nombre del fichero a eliminar: ");
      nombre = sc.nextLine();
      File f = new File(nombre);
      if(f.exists()){
         System.out.println(f.getAbsolutePath());
         if(f.delete()){
            System.out.println("Fichero eliminado");
         }  else{
            System.out.println("No se ha podido eliminar");
         }
      }  else{
         System.out.println("El fichero " + f.getAbsolutePath() + " no existe");
      }
```

```
    //Cambiar el nombre de un fichero
    String nuevoNombre;
    System.out.println("Introduce el nombre del fichero a renombrar: ");
    nombre = sc.nextLine();
    File f1 = new File(nombre);
    if(f1.exists()){
       System.out.println(f1.getAbsolutePath());
       System.out.println("Introduce nuevo nombre: ");
       nuevoNombre = sc.nextLine();
       File f2 = new File(nuevoNombre);
       if(f1.renameTo(f2)){
          System.out.println("Se ha cambiado el nombre");
       }  else{
          System.out.println("No se ha podido cambiar el nombre");
       }
    }  else{
       System.out.println("El fichero " + f1.getAbsolutePath() + " no existe");
    }
  }
}
```

Ejemplo 3: Programa que muestra el contenido de un directorio. En el ejemplo se muestra el contenido del directorio actual.

```
public static void main(String[] args) {
    File directorio = new File(".");  //directorio actual
    String[] lista = directorio.list();
    for (int i = 0; i < lista.length; i++) {
       System.out.println(lista[i]);
    }
}
```

Ejemplo 4: El siguiente programa muestra la **diferencia entre getPath(), getAbsolutePath() y getCanonicalPath()**. Además usa el método getProperty() de la clase System para obtener el directorio de trabajo actual.

```
import java.io.File;
import java.io.IOException;
public class File8 {
```

```
  public static void main(String[] args) throws IOException{
     File f = new File("../../datos.dat");
     String dirActual = System.getProperty("user.dir");
     System.out.println("Directorio actual   " + dirActual);
     System.out.println("getPath()          " + f.getPath());
     System.out.println("getAbsolutePath()   " + f.getAbsolutePath());
     System.out.println("getCanonicalPath()  " + f.getCanonicalPath());
  }
}
```

La salida de este programa es (según mi carpeta de trabajo):

```
Directorio actual F:\curso\programación java\Ficheros\ejemplos\File8
getPath()          ..\..\datos.dat
getAbsolutePath() F:\curso\programación java\Ficheros\ejemplos\File8\..\..\datos.dat
getCanonicalPath()F:\curso\programación java\Ficheros\datos.dat
```

Ejemplo 5: El siguiente programa muestra el contenido del directorio raíz de la unidad actual de trabajo y de todos sus subdirectorios de forma recursiva. Para cada directorio se muestran primero los archivos y a continuación las carpetas que contienen de forma recursiva.

```
import java.io.File;
public class File5 {

  public static void main(String[] args) {
     recorrerDirectorios("/");
  }

  public static void recorrerDirectorios(String ruta) {
     //Se crea un objeto file con la ruta del directorio
     File directorio = new File(ruta);
     //Se comprueba si la ruta existe
     if (!directorio.exists()) {
        System.out.println("La ruta " + directorio.getAbsolutePath() + " no existe.");
        return;
     }
     //Se comprueba si es un directorio
     if (!directorio.isDirectory()) {
        System.out.println("La ruta " + directorio.getAbsolutePath() + " no es un directorio");
        return;
     }
```

```
        System.out.println(directorio.getAbsolutePath());
        //obtener todo el contenido del directorio
        File[] lista = directorio.listFiles();
        //se recorre el directorio y se muestran primero los archivos
        for (File s : lista) {
            if(s.isFile())
                System.out.println("Archivo ->  " + s.getName());
        }
        //se recorre de nuevo el directorio y se obtienen los subdirectorios
        for (File s : lista) {
            //Si es un directorio se vuelve a llamar al método
            if (s.isDirectory()) {
                recorrerDirectorios(s.getAbsolutePath());
            }
        }
    }
```

CREACIÓN DE FILTROS

Un filtro sirve para que el método list devuelva solo aquellos archivos o carpetas que cumplan una determinada condición. (que tengan una extensión determinada, contengan en su nombre una cadena determinada, empiecen por..., etc)

Un filtro es un objeto de una clase que implementa el interface **FilenameFilter**.

FilenameFilter tiene un solo método llamado *accept* que devuelve un valor de tipo boolean:

```
public interface FilenNameFilter{
        boolean accept (File ruta, String nombre);
}
```

El método recibe el directorio donde se encuentra el archivo (objeto File) y el nombre del archivo (String).

Este método lo utiliza el método list de File para decidir si un archivo o directorio determinado se incluye o no en el array que devuelve. Si accept devuelve true se incluye y si devuelve false no se incluye.

El método list llama de form automática al método accept para cada uno de los archivos o directorios.

Ejemplo de creación y uso de un filtro:

Vamos a crear un filtro para obtener todos los archivos que tiene una extensión determinada.

Como dijimos antes, un filtro es un objeto de una clase que implementa el interface FileNameFilter, por lo tanto tenemos que crear esta clase.

La clase se llamará *Filtro* y debe implementar el método accept de FilenameFilter.
En este caso como queremos saber si un archivo tiene una determinada extensión el método accept lo podemos escribir utilizamos el método endsWith de String.

```
public boolean accept(File ruta, String nombre) {
            return nombre.endsWith(extension);
}
```

Para entender mejor el método accept tenemos que ver la clase Filtro completa:

```
import java.io.FilenameFilter;
//Clase Filtro implementa el interface FilenameFilter
public class Filtro implements FilenameFilter {
   String extension;
   Filtro(String extension) {
      this.extension = extension;
   }

   //implementación del método accept del interface
   @Override
   public boolean accept(File ruta, String nombre) {
      return nombre.endsWith(extension);
   }
}
```

El programa que utiliza el filtro para mostrar archivos que tienen una extensión determinada, en este caso los que tienen la extensión .pdf puede ser este:

```
import java.io.File;
//Clase Principal
public class File6 {
   public static void main(String[] args) {
      File ruta = new File("/temas/teoria");
      System.out.println("Archivos .pdf en el directorio " + ruta.getAbsolutePath());
      String[] lista = ruta.list(new Filtro(".pdf")); //se crea el filtro y se le pasa a list
      if (lista == null) {
         System.out.println("Total: 0 archivos");
      } else {
         for (int i = 0; i < lista.length; i++) {
```

```
            System.out.println(lista[i]);
        }
        System.out.println("Total: " + lista.length);
    }
  }
}
```

Más información sobre FilenameFilter en la API de Java:

http://docs.oracle.com/javase/8/docs/api/java/io/FilenameFilter.html