LA CLASE FILE

Un objeto File representa una ruta de fichero o directorio. No se usa para manipular los datos que contiene un Fichero.

Analizamos esta clase usando alguno de sus métodos más representativos.

Para estos ejemplos recuerda que el directorio actual:

- Si estoy en la consola se observa en el prompt o con comando cd en windows o pwd en linux
- Si estoy en IDE es la carpeta del proyecto (no src)

El método exists()

Ejemplo: Comprobar si existe un fichero en el directorio actual

```
import java.io.*;
class App{
   public static void main(String[] args) {
        File prueba = new File("prueba.txt");
        if(prueba.exists())
            System.out.println("El archivo prueba.txt existe");
        else
            System.out.println("El archivo prueba.txt NO existe");
    }
}
```

Para probar el código, crea prueba.txt por ejemplo con el bloc de notas.
Tienes que tener claro que
`File prueba = new File("prueba.txt");`
No crea un fichero prueba.txt, crea un objeto que almacena "la ruta" prueba.txt
Para crear un fichero lo normal es crear un flujo de salida apropiado a lo que necesito (FileWriter, ObjectOutputStream,etc…). Como curiosidad, también es posible usar el método createNew() de la clase File, pero sólo crea ficheros vacíos y sólo en el caso de que el fichero no exista ya. Lo importante es entender que *new File("ruta")* no crea fichero alguno

Usar rutas windows con barra \

Sólo funcionan en sistemas windows. Hay que recordar usar doble \\ barra para conseguir escape.

.

```
import java.io.*;
class App{
   public static void main(String[] args) {
        File prueba = new File("I:\\midirectorio\\prueba.txt");
        if(!prueba.exists())
            System.out.println("I:\\midirectorio\\prueba.txt NO existe");
        if(!new File("I:\\midirectorio").exists())
            System.out.println("c:\\midirectorio NO existe");
        if(!new File("m:").exists())
            System.out.println("m: NO existe");

        if(new File("I:\\midirectorio\\prueba.txt").exists())
            System.out.println("I:\\midirectorio\\prueba.txt existe");
   }

}
```

rutas con barra /.

Con la barra Linux "/", es más transportable el código. La clase File traducirá automáticamente "/" por "\" al acceder a un sistema Windows. Además utilizando la barra linux nos ahorramos en el string tener que escapar la barra \

Lógicamente las rutas absolutas windows que empiezan por letra de unidad (c:, d:, etc.) no van a funcionar en linux independientemente de la barra que usemos.

El siguiente ejemplo, escrito a propósito sin letras de unidad funciona igualmente en linux y windows

```
import java.io.*;
class App{
  public static void main(String[] args) {
      File prueba = new File("micarpeta/misubcarpeta/prueba.txt");

      if(prueba.exists())
         System.out.println("El archivo prueba.txt existe");
      else
         System.out.println("El archivo prueba.txt NO existe");
   }
}
```

Método delete()

Ejemplo: si el fichero existe lo borramos (para probar el código crea en el directorio actual micarpeta/misubcarpeta/prueba.txt

```
import java.io.*;
class App{
  public static void main(String[] args) {
      File prueba = new File("micarpeta/misubcarpeta/prueba.txt");

      if(prueba.exists()){
        System.out.println("El archivo prueba.txt existe");
        prueba.delete();
        System.out.println("prueba.txt borrado");
      }
      else{
        System.out.println("El archivo prueba.txt NO existe");
      }

   }
}
```

Crear directorios.

mkdir() -> para crear un directorio
mkdirs()->para crear una jerarquía de directorios

```
import java.io.*;
class App{
  public static void main(String[] args) {
      String directorio = "pruebacrear";
      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(directorio+"/"+varios)).mkdirs();
      if (exito)
         System.out.println("Directorios: " + varios + " creados");
  }
}
```

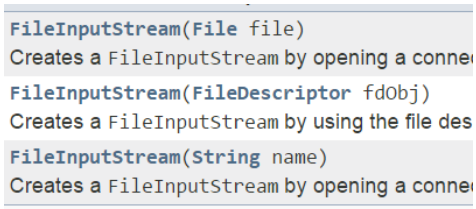
listar un directorio con list().

```
import java.io.*;
class App{
  public static void main(String[] args) {
      //String directorio = "C:/windows";
      String directorio = "/etc";
      File miruta=new File(directorio);
      String[]contenidoMiruta=miruta.list();
      for(String item:contenidoMiruta){
         System.out.println(item);
      }
  }
}
```

File vs clases de java.nio.file

El paquete import java.nio.file es una novedad del JDK7 con un montón de interfaces y clases nuevas para interactuar con el sistema de ficheros y se prefiere a la clase File para hacer las típicas operaciones de ficheros (copia, atributos, enlaces simbólicos). Se prefiere porque tiene mejorada su eficiencia y permite hacer más cosas.
Además, desde java 8 el paquete java.nio.file se fue enriqueciendo con los nuevos conceptos de programación funcional y se multiplicó su potencia.

No obstante *la vieja File* sigue siendo importante en algunos contextos, por ejemplo fíjate en los constructores de FileInputStream, sigue habiendo una versión con File



La clase Files

Ejemplo de uso de Files: copiar un fichero.
Observa que usa un Interface nuevo *Path* que recuerda precisamente a la clase vieja File.
Para este ejemplo debes de crear *midirectorio/listadecompra.txt* y *tudirectorio*

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class App {
   public static void main(String[] args) throws IOException {
      Path origen = Paths.get("midirectorio/listadecompra.txt");
      Path destino = Paths.get("tudirectorio/miscompras.txt");

      if(Files.exists(origen)){
         Files.copy(origen, destino);
      }else{
          System.out.println("La ruta origen es inaccesible");
      }

   }
}
```

Ejercicio U8_B4_E1:
En args[0] indicas un directorio y el programa imprime su contenido (no recursivo). Ejemplo listando el contenido de c:\ de un equipo windows 🪟:

```
C:\Users\Pilt\Documents\VSC\java\MiApp>cd src

C:\Users\Pilt\Documents\VSC\java\MiApp\src>javac App.java

C:\Users\Pilt\Documents\VSC\java\MiApp\src>java App   c:\
$Recycle.Bin
$WinREAgent
Archivos de programa
Documents and Settings
DumpStack.log.tmp
```

Hazlo de dos formas:

1. Con la vieja File y su método list()
2. Con Files.walk()

Ejercicio U8_B4_E2: Repetimos el ejercicio anterior pero ahora el programa imprime también el contenido de los subdirectorios. Hazlo de nuevo con las dos técnicas indicadas anteriormente. Con File hay que escribir un método recursivo. Con walk no hay que hacer nada

DESDE JAVA 8: FILES +STREAMS
Vimos que el api Streams se aplica para procesar datos de un origen como por ejemplo los datos de una List. Las clases del api java.nio.file devuelven mucha información acerca del sistema de ficheros y esta información puede ser un interesante origen de información para procesar en un Stream razón por la que nos encontramos en el api java.nio.file muchos métodos que devuelve Streams.

Listar ficheros con Files.list
List devuelve un Stream de objetos Path. Parecido a walk() pero sin posibilidades recursivas
Ejemplo: obtenemos todas las rutas del directorio actual, las ordenamos y hacemos una lista en un String separándolas por ;

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class App {

    public static void main(String[] args) throws IOException {
        Stream<Path> stream = Files.list(Paths.get(""));
        String listaRutas = stream
                .map(String::valueOf)
                .sorted()
                .collect(Collectors.joining("; "));
        System.out.println("Lista de rutas: " + listaRutas);
    }
}
```

Para repasar la sintaxis de referencia a métodos, recuerda que con
.map(String::valueOf)

Estamos invocando al método valueOf de la clase String que le pasamos de parámetro por defecto un path del stream que a su vez lo que va a hacer es invocar al toString() de la clase Path
También lo pudimos haber solucionado directamente , invocando al toString() de Path

```
.map(Path::toString)
```

Y Si lo haces con lambda puede estar más claro

```
.map(p->String.valueOf(p))
```

Ejercicio U8_B4_E3: El ejemplo anterior ahora sólo lista ficheros, es decir, no lista directorios. Para ello filtra con Files.isDirectory(path)

Encontrar ficheros con Files.find
Ejemplo, todos los ficheros de texto del directorio actual(sin mirar subdirectorios)

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class App {
    public static void main(String[] args) throws IOException {
        Path comienzo = Paths.get("");//ruta de comienzo busqueda
        Stream<Path> stream = Files.find(comienzo, 1,(path,attr)-> path.toString().endsWith("txt"));
        String listaRutas = stream
                .map(String::valueOf)
                .sorted()
                .collect(Collectors.joining("; "));
        System.out.println("Lista de rutas: " + listaRutas);
    }
}
```

La lambda de find tiene dos parámetros, el segundo es para referirse a los atributos de una ruta y simplemente no lo usamos en el cuerpo

Ejercicio U8_B4_E4: El trabajo de find() también recuerda un poco al de walk. Si te fijas walk() también devuelve un Stream pero sin filtrar. De hecho si al stream que devuelve walk le aplicamos un filtro, obtenemos un efecto similar al find. Escribe el ejemplo anterior con walk

Ejercicio U8_B4_E5: Observa el siguiente ejemplo
Podemos trabajar con el contenido de un fichero zip con ZipFile. Con programación funcional es muy fácil manipularlo.

```
import java.io.IOException;
import java.util.zip.ZipFile;

public class App {
    public static void main(String[] args) throws IOException {
        try (ZipFile zipFile = new ZipFile("mizip.zip")) {
            zipFile.stream()
                    .forEach(zpe -> System.out.println(zpe.getName() + "  tamaño: " + zpe.getSize()));
        }
     }
}
```

Cada elemento del Stream es un ZipEntry (consulta API). Intenta ordenar el listado de forma que sea por tamaño de mayor a menor aplicando el método sorted() al stream