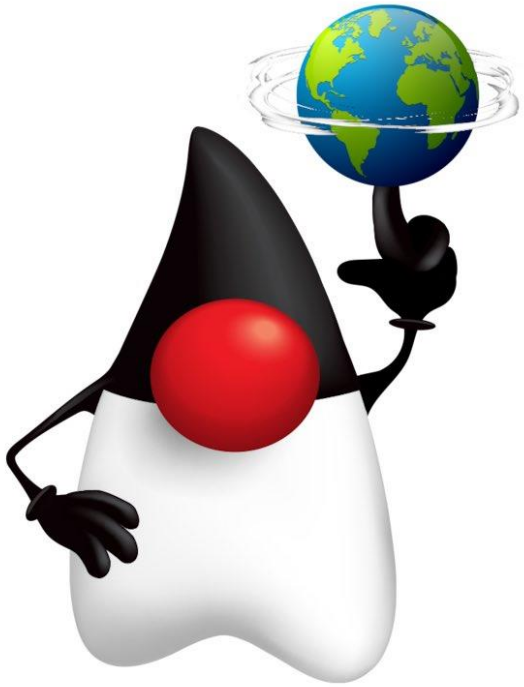


01.02. E/S desde archivos (java NIO.2)

Ficheros – Pepe Calo

1. E/S desde archivos (java NIO.2)

pepinho.com

1. E/S desde archivos. Java NIO.

- **Desde Java 7** se introdujo un mecanismo para trabajar con archivos, **Java NIO**.

- El paquete ***java.nio.file*** y su paquete relacionado, ***java.nio.file.attribute***, proporcionan soporte completo para E/S de archivos y para **acceder al sistema de archivos predeterminado.**

- La **interface *Path*** es el "**punto de entrada" principal del paquete de java NIO (New IO).**

6. E/S desde archivos: ***Files y Paths (I)***

Otras clases principales del paquete son:

a) La clase ***Paths***, **que contiene exclusivamente métodos estáticos** que devuelven un ***Path*** a partir de una **cadena o URL:**

- ***Path Paths.get(String….)***
- ***Path Paths.get(URL uri)***

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/nio/file/Paths.html

6. E/S desde archivos: ***Files y Paths (II)***

b) La **clase *Files* contiene exclusivamente métodos estáticos que se ocupan de operaciones de archivos,** directorios y otros tipos de archivo.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/nio/file/Files.html

- Por ejemplo:

  *Files.**copy**(…),Files.**move**(…), Files.**createDirectories**(…), Files.**createFile**(…), Files.**createLink**(…), Files.**createSymbolicLink**(….), Files.**delete**(Path p), Files.**deleteIfExists**(…), Files.**exists**(…), Files.**getAttribute**(…), Files.**isHidden**(Path p), Files.**find**(…), Files.**lines**(Path p), Files.**newBufferedReader**(…), Files.**readAllLines**(…), Files.**readString**(Path p),…*

Si se tiene código de E/S de archivo escrito antes de la versión de Java SE 7, que utiliza java.io.File, se puede aprovechar la funcionalidad de la interface Path utilizando el método File.toPath():

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/io/File.html#toPath()

Interface Path (java NIO.2)

pepinho.com

1. E/S desde archivos: interface ***Path***

- La interface ***Path***, introducida en la versión Java SE 7, es uno de los principales puntos de entrada del paquete ***java.nio.file***.:

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/nio/file/Path.html

- Si la aplicación utiliza E/S de archivos, **es importante conocer las características de esta interface**.

- ***Path*** es una representación mediante programación de una **ruta de acceso en el sistema de archivos**.

- Un objeto ***Path*** contiene el **nombre de archivo y la lista de directorios utilizados para construir la ruta de acceso**, y se utiliza para examinar, localizar y manipular archivos.

1. E/S desde archivos: ***Path (2)***

A) Creación de un *Path*: *get(…)*

- Utilizando el método estático **get** de la clase ***Paths*** (con s):

```
Path p1 = Paths.get("/home/pepe");
Path p2 = Paths.get(args[0]);
Path p3 = Paths.get(URI.create("file:///Users/pepecalo/Arquivo.java"));
```

- Equivalente a:

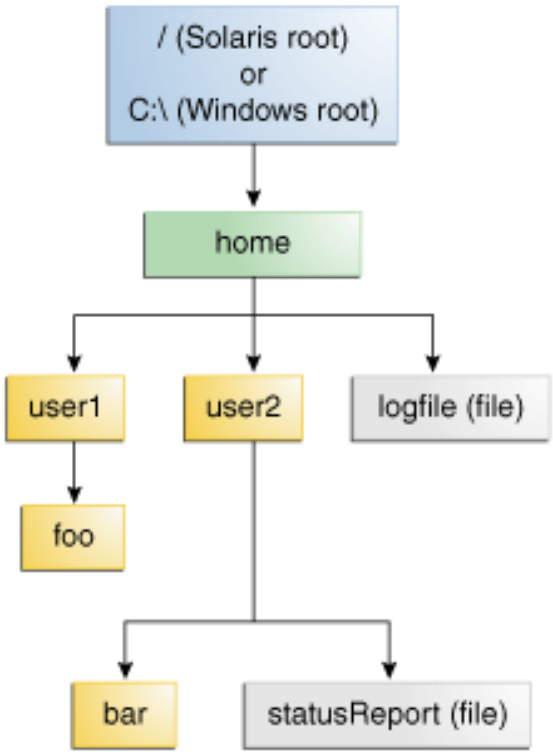
```
Path p4 = FileSystems.getDefault().getPath("/home/pepe");
```

- Por ejemplo:

```
Path p5 = Paths.get(System.getProperty("user.home"),
                    "docs", "apuntes.docx");
```

1. E/S desde archivos: ***Path (3)***

B) Recuperando información de un Path



- ***Path*** almacena una secuencia de elementos.
- El **elemento más alto** de la estructura de directorios se ubicaría en el **índice 0**. El elemento **más bajo** de la estructura de directorios se ubicaría en el **índice [n-1]** (*n es el número de elementos de nombre en la ruta)*
- Hay **métodos disponibles para recuperar elementos individuales o una subsecuencia de la ruta utilizando estos índices:**

1. E/S desde archivos: ***Path (4)***

Recuperando información de un *Path*. **Ejemplo:**

```
// Microsoft Windows
Path path = Paths.get("C:\\users\\pepe\\apuntes");
// o en Linux/Unix
Path path = Paths.get("/users/pepe/apuntes");

System.out.format("toString: %s%n", path.toString()); //
C:\users\pepe\apuntes
System.out.format("getFileName: %s%n", path.getFileName()); // apuntes
System.out.format("getName(0): %s%n", path.getName(0)); // users
System.out.format("getNameCount: %d%n", path.getNameCount()); // 3
System.out.format("subpath(0,2): %s%n", path.subpath(0,2)); // users\pepe
System.out.format("getParent: %s%n", path.getParent()); // \users\pepe
System.out.format("getRoot: %s%n", path.getRoot()); // c:\
// si la ruta es relativa getRoot devuelve null
// En Linux con ruta absoluta devuelve /, la barra del “7”
```

1. E/S desde archivos: ***Path (5)***

Ejemplo para una **ruta relativa:**

```
// Microsoft Windows
Path path = Paths.get("pepe\\apuntes");
// o en Linux/Unix
Path path = Paths.get("pepe/apuntes");

System.out.format("toString: %s%n", path.toString()); //
pepe\apuntes
System.out.format("getFileName: %s%n", path.getFileName()); //
apuntes
System.out.format("getName(0): %s%n", path.getName(0)); // pepe
System.out.format("getNameCount: %d%n", path.getNameCount()); // 2
System.out.format("subpath(0,2): %s%n", path.subpath(0,2)); // pepe
System.out.format("getParent: %s%n", path.getParent()); // pepe
System.out.format("getRoot: %s%n", path.getRoot()); // null
// si la ruta es relativa getRoot devuelve null
```

1. E/S desde archivos: ***Path (6)***

C) Eliminado redundancias de un Path: *normalize()*

- Muchos sistemas de archivos usan **la notación "." para denotar el directorio actual y ".." para denotar el directorio padre**. Ejemplos redundancias:
  - */home/./pepe/apuntes*
  - */home/otto/.. /pepe/apuntes*

- El método ***normalize()*** **quita cualquier elemento redundante**, que incluye cualquier ocurrencia "." o "*directorio*/.." *En los dos ejemplos anteriores se normalizan a /home/pepe/apuntes*

1. E/S desde archivos: ***Path (7)***

- ***normalize()*** **no comprueba en el sistema de archivos** cuando limpia una ruta *(por ejemplo, en el segundo ejemplo, si otto fuera un enlace simbólico, eliminando otto/.. puede dar como resultado una ruta de acceso inalcanzable).*

- Para limpiar una ruta de acceso y **asegurarse de que el resultado localiza el archivo correcto se puede utilizar el método *toRealPath***.

  https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/nio/file/Path.html#toRealPath(java.nio.file.LinkOption...)

  ```
  Path toRealPath(LinkOption... options) throws IOException
  ```

  - Devuelve la **ruta *real* de un archivo existente**.
  - El parámetro ***options*** puede usarse para indicar **cómo manejan los enlaces simbólicos**: por defecto resuelve los enlaces simbólicos, pero puede usarse el valor *LinkOption.NOFOLLOW_LINKS*.

1. E/S desde archivos: ***Path (8)***

D) Conversión de *Path* en: String, URI, Path absolutos

Existen varias formas de **convertir un Path: en un String**, en una **URI** para abrirla en un navegador, etc.

- ***Path* a *URL***: ***toURI()***

```
Path p = Paths.get("/home/pepe");
// O resultado é: file:///home/pepe
System.out.format("%s%n", p.toUri());
```

- ***URL* a *Path***: existe una versión que devuelve un ***Path*** a partir de la URI:

```
static Path of(URI uri)
```

1. E/S desde archivos: ***Path (9)***

- ***Path*** a ***Path*** absoluto: ***Path*** ***toAbsolutePath()***, el archivo **no necesita existir**.

```
public class PruebaPath {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("uso: PruebaPath archivo");
            System.exit(-1);
        }
        Path ruta = Paths.get(args[0]);
 /* Convierte un Path en un una ruta absoluta, considerando el directorio actual
(si no se indica):
java PruebaArchivo nada.txt
Los métodos getRoot y getParent deverían devolver null en la instancia original de
“ruta“.  Si se invoca getRoot y getParent en “rutaCompleta“ devuelve los valores
esperados. */
        Path rutaCompleta = ruta.toAbsolutePath();
    }
}
```

1. E/S desde archivos: ***Path (10)***

- ***Path*** a ***Path*** absoluto: ***toRealPath():***

  `Path toRealPath(LinkOption... options)`

Devuelve la **ruta real** a un ***Path*** **existente**:

- Si se pasa true a este método y el sistema de archivos admite enlaces simbólicos, **resuelve cualquier enlace simbólico en la ruta**.
- Si el ***Path*** **es relativo**, devuelve una **ruta absoluta**.
- Si el ***Path*** **contiene elementos redundantes**, devuelve una ruta con esos **elementos eliminados**.
- Puede lanzar una excepción de IO.

```
try {
    Path fp = path.toRealPath();
} catch (NoSuchFileException e) {// Si el archivo no existe
    System.err.format("%s: no existe el archivo o el directorio%n", path);
} catch (IOException e) {
    System.err.format("%s%n", e); // gestión de otro tipo de error
}
```

1. E/S desde archivos: ***Path (11)***

E) Uniendo *Path*

Se pueden unir varios Path usando el método ***resolve***. Se pasa una ruta parcial (sin elemento raíz) que se añade a la ruta original:

```
Path p = Paths.get("C:\\users\\pepe\\docs");
// El resultado es C:\users\pepe\docs\apuntes:
System.out.format("%s%n", p.resolve("apuntes"));
```

Si se le pasa la ruta absoluta a una relativa revuelve la ruta absoluta:

```
// El resultado es C:\users\pepe\docs\
Paths.get("apuntes").resolve("C:\\users\\pepe\\docs");
```

1. E/S desde archivos: ***Path (12)***

F) Creación de *Path* relativas entre dos *Path*:

Construir una ruta (Path) de una localización en otra:

Path ***relativize****(Path ruta)*, que **construye una ruta relativa desde una localización en otra con respecto a la original**.

```
Path p1 = Paths.get(“pepe");
Path p2 = Paths.get(“otto");

// El resultado es ../otto si están en /home
System.out.format("%s%n", p1.relativize(p2));
// El resultado es ../pepe
System.out.format("%s%n", p2.relativize(p1));
```

1. E/S desde archivos: ***Path (13)***

Otro ejemplo de rutas relativas entre Path:

```
Path p1 = Paths.get(“home");
Path p2 = Paths.get(“home/otto/apuntes");

// El resultado es otto/apuntes si están en /home
System.out.format("%s%n", p1.relativize(p2));
// El resultado es ../..
System.out.format("%s%n", p2.relativize(p1));
```

Un *Path* **no puede ser construido si sólo uno de ellos tiene un elemento raíz**. Si ambos tiene elemento raíz, depende del sistema.

1. E/S desde archivos: ***Path (14)***

G) Comparación de dos *Path*:

- Un *Path* admite ***equals*** para comprobar **si dos Path son iguales**:

```
boolean equals(Object outro)
```

- Los métodos ***startsWith*** y ***endsWith***: permiten comprobar si una ruta empieza o termina en determinado String:

```
boolean startsWith(Path other)
boolean startsWith(String other)
boolean endsWith(Path other)
boolean endsWith(String other)
```

1. E/S desde archivos: ***Path (15)***

- Ejemplo de comparación de *Path*:

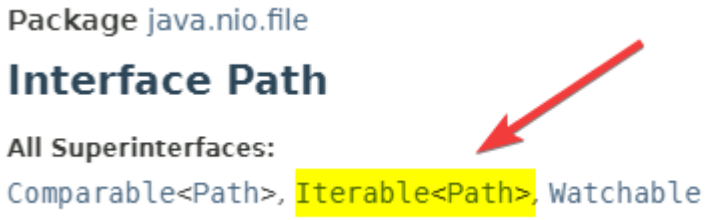
```
Path p1 = ...;
Path p2 = ...;
Path inicio = Paths.get("/home");
Path fin = Paths.get("apuntes");

if (p1.equals(p2)) {
    // son iguales
} else if (p1.startsWith(inicio)) {
    // está en el directorio "/home"
} else if (p1.endsWith(fin)) {
    // dentro de la carpeta de "apuntes"
}
```

1. E/S desde archivos: ***Path (16)***

G) Iteración de *Path*:

- La interface Path hereda de la interface ***Iterable<Path>*** por lo que tiene un método

```
Iterator<Path> iterator()
```



Permite **iterar sobre los nombres de los elementos del Path**.

El **primer elemento** devuelto es el **más cercano al raíz** del árbol de directorios.

Además, es ***Comparable<Path>***, por lo que puede compararse con el método ***compareTo*** o ordenarse en una colección, por ejemplo.

```
Path path = ...;
for (Path nombre: path) {
    System.out.println(nombre);
}
```