BOLETÍN 01.02: JAVA NIO

Ejercicio 1. Path y Paths. Files.

1

Realiza los siguientes programas Java **empleando las clases *Path*, *Paths* y *Files* de Java NIO**:

a) Escribe un programa Java que compruebe si una ruta de archivo **es absoluta o relativa y si existe**.

b) Escribe un programa Java que **copie un archivo en otro**, sustituyéndolo si existe, y lo **mueva un archivo de una ubicación en otra**, empleando Files.

c) Crea un programa Java que recoja una ruta de archivo como entrada del usuario (con JFileChooser) y muestre el nombre del archivo y su extensión en una ventana emergente (JOptionPane). **Crea un Path y recupera la posición a partir del nombre del archivo** (emplea el método *lastIndexOf*).

Ejercicio 2. Creación de directorios.

Escribir un programa en Java que, empleando las clases de Java NIO 2 Path y File, cree un directorio (con toda la tura) y un archivo vacío dentro de ese directorio.

Ejercicio 3. Contenido de un directorio con list.

a) Escribid un programa en Java que, empleando las clases de Java NIO 2, liste los archivos de un directorio por medio del método list(). Debe mostrar sólo los archivos fuente Java, no los directorios que contiene. Recuerda el uso de filtros en Stream y de forEach.

b) Haz el mismo ejercicio con el método list() de File y compara el tiempo de ejecución en cada caso.

Ejercicio 4. Listar y copiar archivos fuente java de un directorio a otro.

Escribid un programa en Java que, empleando las clases de Java NIO 2, liste los contenidos copiando cada archivo a una ruta destino. Pidiendo la extensión del archivo a copiar. Resuelve previamente el fichero origen.

Ejercicio 5. Descarga Internet con Java NIO.2.

Haz un programa que descargue un archivo de Internet y lo copie en un directorio concreto. Emplea ventanas de tipo JFileChooser para indicar la ruta en la que guardarlo y una caja de texto para escribir la URL.

2

Este código debe operar del siguiente modo:

- Encapsular en flujo al archivo de internet, *new URL(url).openStream()*, dentro de un Buffer.
- Guardarlo en un directorio destino empleando *Path*.
- Emplead el método estático copy de Files para descargar el archivo, la versión que recoge un InputStream como origen y lo guarda en un Path, sustituyendo el destino si existe.
- Cópialos con la opción REPLACE_EXISTING, que indica que si el archivo de destino ya existe, se debe reemplazar.

Abre el archivo descargado con un BufferedInputStream, pero creando el InputStream con el método estático de Files.

Ejercicio 6. Listar los archivos de un directorio.

Emplea un ***JFileChooser*** para seleccionar la carpeta a explorar. Además, debe realizar la **comprobación de la existencia y de que es un directorio**.

Realiza un programa que muestre el listado del contenido de un directorio de los siguientes modos:

a) Repite el ejercicio anteriormente realizado que muestre el contenido de todos los archivos y directorios de una carpeta dada con el método ***listFiles()*** de ***File***, pero a partir de un Path.

El método listFiles() devuelve una array de objetos File que son el contenido del directorio:

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

b) De manera recursiva muestre todo el contenido, incluidos subdirectorios:

- Crea un ***ArrayList*** para guardar las rutas.
- Crea un método ***recorrer***, que, de manera recursiva recoja el archivo o directorio:

```
recorrer(ruta){
  obtener elementos
  para cada elemento de
    si es un directorio
        recorrer(elemento)
    sino si es un archivo
        añadir a la lista de rutas
    fin si
  fin para
}
```


c) A modo de **repaso de Stream**, hazlo aplicando el mismo método, pero creando un Stream con el array de archivos.

   i. Stream<File> flujo = Stream.of(**new File**(dir).listFiles()); // Crea un Stream
   ii. Aplica un filtro para ver si es archivo o directorio.
   iii. Aplica un mapa obtener el nombre de los archivos.
   iv. Muestra los archivos recorriéndolos.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/stream/Stream.html

Un Stream es una **secuencia de elementos que admiten operaciones secuenciales o agregaciones** (filtrar –**filter**, contar –**count**, devolver un flujo como resultado de una opración –**map**, ordenar –**sorted**(), convertirlo en una lista –**toList**(), etc.)

Puedes ver ejemplos y ayuda en:
https://www.geeksforgeeks.org/java-8-stream-tutorial/
https://docs.oracle.com/javase/tutorial/collections/streams/index.html
https://stackify.com/streams-guide-java-8/

Operaciones **intermedias** son (pueden anidarse):

- *filter()*
- *map()*
- *sorted()*

Operaciones terminales son:

- *forEach()*
- *collect()*
- *match()*
- *count()*
- *reduce()*

Operaciones en cortocircuito son:

- *anyMatch()*
- *findFirst()*

Ejemplo:

4

```
List<String> = miStreamDeString.map(nome -> nome.toUpperCase())
.collect(Collectors.toList());List<String> = miStreamDeString.map(nome ->
nome.toUpperCase())
.collect(Collectors.toList());
```

d) Con ***Files.newDirectoryStream*** (obsoleto)

Haz uso de streams. Emplea la clase ***Files*** *y el método* ***newDirectoryStream*** *que devuelve un stream al directorio:*

```
public static DirectoryStream<Path> newDirectoryStream(Path dir)
                                    throws IOException
```

Que **itera con un for sobre todas las entradas del directorio**. Los elementos devueltos por el ***iterador*** del DirectoryStream **son de tipo Path** y cada uno representa una entrada en el directorio.

Cuando no se utiliza la construcción **try-with-resources**, se debe invocar el método ***close()*** de flujo de directorio después de completar la iteración para liberar cualquier recurso retenido para el directorio abierto.

e) Con ***Files.walk***:

Haz uso del método:

public static ***Stream***<*Path*> ***walk***(*Path start, FileVisitOption... options)*

Que devuelve un Stream a recorrer el árbol desde un directorio dado. El recorrido se hace en profundidad.

Ejercicio 3. Eliminación de directorio

Haz un programa que, si existe elimine un directorio y todo su contenido de manera recursiva. Ten en cuenta, que el método *Files.delete* lanza una excepción si el directorio no está vacío.

a) De manera recursiva, recorriendo los directorios, con el método listFiles de File.
b) Por medio del *Files.walkFileTree* y *FileVisitor* de Java 7+
c) Por medio de Files.walk de Java 8+