USO DEL API STREAM CON JAVA NIO.2

NIO.2 + API STREAM

- La clase Files provee de algunos métodos *potentes* que devuelven Stream<T>.
- Podemos usarlos con lo que ya sabemos del API Stream para filtrar, mapear, reducir…

Files.list

- Devuelve todas las rutas de un directorio dado.
- De esta forma, podemos filtrar, mapear, reducir, ...

```
try (Stream<Path> stream =
Files.list(Paths.get(System.getProperty("user.home"),"ejemplo"))) {
    stream
        .map(String::valueOf)
        .filter(path -> !path.startsWith("."))
        .sorted()
        .forEach(System.out::println);
}
```

Files.find

- Devuelve todas las rutas a partir de un directorio dado que cumplen una condición. Se le puede indicar una profundidad máxima.

```
Path start = Paths.get(System.getProperty("user.home"),"ejemplo");
int maxDepth = 5;
try (Stream<Path> stream = Files.find(start, maxDepth, (path, attr) ->
                                    String.valueOf(path).endsWith(".txt"))) {
     stream
        .sorted()
        .map(String::valueOf)
        .forEach(System.out::println);
}
```

Files.walk

- Devuelve todas las rutas a partir de un directorio dado. Se le puede indicar una profundidad máxima.

```
Path start = Paths.get(System.getProperty("user.home"), "ejemplo");
int maxDepth = 5;
try (Stream<Path> stream = Files.walk(start, maxDepth)) {
   TreeMap<String, Long> groupByExtension =
      stream.filter(Files::isRegularFile)
       .sorted()
       .collect(Collectors.groupingBy(C_Walk::getExtension, TreeMap::new,
                                                    Collectors.counting()));

   groupByExtension.forEach((k, v)
           -> System.out.printf("%s -> %d ficheros%n", k, v));


}
```

Files.lines

- Devuelve las líneas de un fichero de texto en un Stream.

```
try (Stream<String> stream = Files.lines(p, Charset.forName("Cp1252"))) {
   return Optional.of(stream
        .map(s -> s.split(";"))
        .collect(Collectors.toList()));
}
```