



---

# PRACTICA 1

---

## TEMA 01 STREAMS, FICHEROS Y EXPRESIONES REGULARES



## Diseño:

El diseño de la solución que hemos decidido implementar consiste en:

- Lectura y filtrado de los archivos
- Generación de Clases que modelan y almacenan los datos
- Generación del reporte HTML junto a sus gráficos

## Estructura:

### [DataReading \(del paquete ioutils\):](#)

Hemos generado esta clase para la lectura y filtrado de los archivos.

Esta clase contiene métodos estáticos que se basan en métodos de Java NIO para lectura y gestión de ficheros.

El método `getFile` implementa la clase `Files.lines` para obtener las líneas de los Path obtenidos mediante `Paths.get`.

Usamos Java IO para el método `deleteDirectory` en caso de que el directorio existiera de Path obtenido por el argumento pasado al programa.

El método `createDirectory` aprovecha el `Files.createDirectory` y el método `Files.exists` para realizar la creación del directorio donde se generará el informe.

Los métodos `getStation` y `getStationDataStream` utilizan la API Stream para obtener mediante filtrado la estación de la ciudad pasada por parámetro y sus datos respectivos.

Para realizar este filtrado hemos comparado la cadena obtenida por parámetro con los nombres de las estaciones almacenados en el documento estaciones obteniendo así su código de estación. Dicho código es posteriormente empleado para filtrar los datos de dicha estación.

Para la división de los campos de los documentos usamos `Arrays.asList` y de cada línea `Split`.

Y por último el método `getMeasures` es el encargado de parsear de líneas de datos a objetos `HourMeasurement` contenidos en objetos `Measure` junto con su fecha y el código su magnitud de medición. Para realizar el parseo separamos cada línea usando el separador del documento (";"), accediendo al valor por pares para almacenar los valores en el objeto `Measure`.

### [MeteoPractica \(del paquete service\)](#)

Esta clase es la fachada principal del proyecto.

El método `generateMeteoAnalysis` es el encargado de ejecutar cada uno de los submétodos que lean y parseen los datos que componen el objeto `Analytics` (reporte final HTML).

La parte de lectura y filtrado se ejecutan en hilos usando las clases `ThreadPoolExecutor` y los `Runnable` `RunnableStreamReader` y `RunnableCSVReader` y después utiliza todos los datos obtenidos para construir `Analytics`. También llama a la creación del directorio final del informe mediante el método de `DataReading` `createDirectory`. Hemos instanciado el

`ThreadPoolExecutor` mediante `Executors.newFixedThreadPool` al cual le hemos indicado que debe crear cuatro hilos (uno para cada documento). Luego lanzamos dos objetos

`RunnableStreamReader`, uno para cada documento de datos de medidas, y dos

`RunnableCSVReader` para los archivos de magnitudes. Estas clases simplemente ejecutan

concurrentemente los métodos de `DataReading` explicados anteriormente. El método

`buildAnalytics` es el encargado de recibir las listas y generar las dos listas finales de `MonthData`

filtradas por los objetos Magnitude, dichos datos son los que utilizamos para crear el objeto Analytics.

### Analytics(del paquete service)

Es la clase encargada de generar los gráficos y el informe HTML.

El informe se genera de dichos gráficos junto con los datos de los listados finales de mediciones. Desarrollamos dos métodos `getStartingDate` y `getEndingDate`, que devuelven mediante API Stream las fechas de primera medición de la estación pasada por argumento. El método `generateChart` utiliza las listas finales para generar objetos chart del api JFreeChart pasándole los datos obtenidos de medidas. En caso de no existir un directorio `images` hijo del directorio pasado por parámetro lo crea, para luego introducir los png de los charts creados. Luego creamos el método `htmlBuilder` que utilizando un listado de String de la clase va añadiendo línea a línea las etiquetas HTML para el informe final, para el contenido del cuerpo de dicho informe se llama al método `generatorHTMLBody` con cada lista de las mediciones finales. El método `generatorHTMLBody` simplemente recorre en bucle los listados de mediciones para añadirlos a lista de String de la clase que utiliza el método anterior en el cuerpo del informe. En caso de que los valores de mediciones se han nulos, dicha medición cambiara a solo una línea de mensaje de datos no disponible. Por último, `generateHtml` consiste en crear la ruta final del informe junto con su nombre. Comprueba si existe y lo elimina, luego crea el fichero y saca por el navegador.

### Clases modelo de datos(POJOs):

**MonthData:** Lista de Measures, objeto Magnitude, objetos de tipo Moment máximo y mínimo, objetos de tipo LocalDate para principio y final de la medición y valor de medio de los valores de ese mes.

**Moment:** objeto LocalDateTime de la medición y un valor float de dicha medición.

**Magnitude:** valor cadena de código de magnitud, valor cadena de descripción, y valor cadena de la unidad de medición.

**Measure:** valor cadena de código de magnitud, objeto LocalDate de la fecha de medida, tres métodos para obtener: máximo, mínimo y media de objetos HourMeasurement que colecciona.

**HourMeasurement:** valor numérico de hora, valor float de valor de medición y valor de carácter de validación.

**Station:** 3 valores cadena de código de estación, zona de estación y ciudad de estación.

