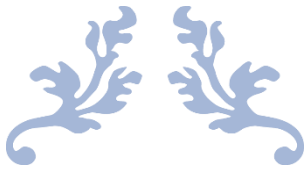


PRACTICA 2

TEMA 01 STREAMS, FICHEROS Y EXPRESIONES REGULARES

2-DAM – ACCESO A DATOS
POR: SERGIO PÉREZ Y FEDERICO TOLEDO

Diseño:

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

- Lectura y parseo de los archivos CVS a XML
- Lectura y filtatrado de los XML por JDOM
- Generación de Clases que modelan y almacenan los datos
- Generación del reporte HTML junto a sus gráficos
- Generación del XML base de datos de las mediciones
- Generación del informe final de todas las mediciones de la ciudad tanto por consola como en fichero Markdown

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 lines 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.exist 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.

CSVReader(del paquete ioutils):

Esta clase mediante JDOM atraves de SAX y DOM se encarga de mediante Hilos de leer los datos de los CSV utilizando Split para separar por el carácter “;” con API Stream.

Clases Runnable():

Que implementan XML de JDOM Lectura para generar los objetos atraves de la clase padre RunnableXMLReader

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 parsean 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

Runnables RunnableXMLReader 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 cinco hilos (uno para cada documento). 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.

JDOM(del paquete service):

Que implementa la tecnología de JDOM con SAX para la lectura y DOM para la escritura. Esta clase es consumida por otras durante la ejecución para resolver la lectura y escritura de los XML. Además de sus métodos más básicos para cargar, inicializar, formatear, escribir e imprimir por consola, posee el método getAnalyticsOfCity que obtiene mediante una expresión XPath todas las mediciones almacenadas en el fichero de base de datos final XML, para luego filtrarlas con API Stream para obtener sus medias y generar los objetos CityMeans.

JAXBController(del paquete service):

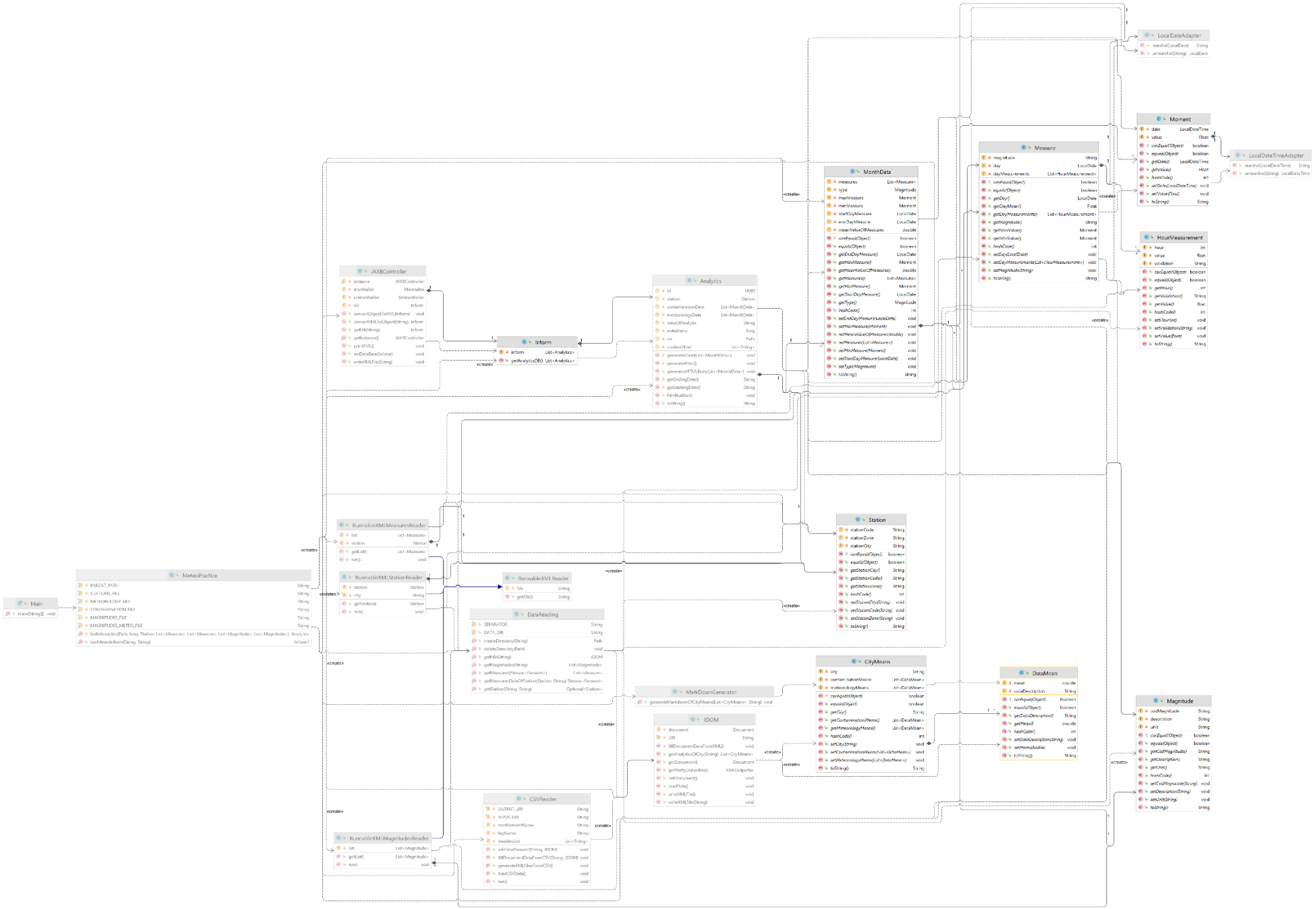
Clase que implementa la tecnología de JAXB para tanto generar el XML de informe mediante Marshaller atraves de la generación automática del esquema con las clases aprovechando las etiquetas de formateo de JAXB. También se encarga del proceso inverso con Unmarshall para del informe final XML obtener los objetos mediciones de ejecuciones anteriores.

LocalDateAdapter y LocalDateTimeAdapter (del paquete service):

Creadas para parsear de String a LocalDate y LocalDateTime respectivamente para ayudar al Marshall y Unmarshall de JAXB.

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.
Inform: Lista de Analytics de las mediciones almacenadas en la base de datos en el XML.
DataMean: valor double de media y cadena de descripción de magnitud.
CityMeans: cadena de nombre de la ciudad, Lista de medias de contaminación y de metereologia.


Main
mainString[]: void
MeteoPractice
PARENT_PATH: String
STATIONS_HLE: String
METEOROLOGY_HLE: String
CONTAMINATION_FILE: String
MAGNITUDES_FILE: String
MAGNITUDES_METEO_FILE: String
buildAnalytics(Path, long, Station, List<Measures>, List<Measures>, List<Magnitudes>, List<Magnitudes>): Analytics
runMeteoInform(String, String): Inform?
JAXBController
instance: JAXBController
marshaller: Marshaller
unmarshaller: Unmarshaller
ois: Inform
convertObjectToXMLInform(): void
convertXMLToObject(String): Inform
getJAXB(String): Inform
getInstance(): JAXBController
printXML(): void
setDataBase(Inform): void
writeXMLFile(String): void
Inform
inform: List<Analytics>
getAnalytics0B(): List<Analytics>
Analytics
id: UUID
station: Station
contaminationData: List<MonthData>
meteorologyData: List<MonthData>
timeOfAnalysis: String
installtime: long
uri: Path
contentHtml: List<String>
generateChart(List<MonthData>): void
generateHtml(): void
generateHTMLBody(List<MonthData>): void
getEndingDate(): String
getStartingDate(): String
htmlBuilder(): void
toString(): String
MonthData
measures: List<Measure>
type: Magnitude
maxMeasure: Moment
minMeasure: Moment
startDayMeasure: LocalDate
endDayMeasure: LocalDate
meanValueOfMeasures: double
canEqual(Object): boolean
equals(Object): boolean
getEndDayMeasure(): LocalDate
getMaxMeasure(): Moment
getMeanValueOfMeasures(): double
getMeasures(): List<Measure>
getMinMeasure(): Moment
getStartDayMeasure(): LocalDate
getType(): Magnitude
hashCode(): int
setEndDayMeasure(LocalDate): void
setMaxMeasure(Moment): void
setMeanValueOfMeasures(double): void
setMeasures(List<Measure>): void
setMinMeasure(Moment): void
setStartDayMeasure(LocalDate): void
setType(Magnitude): void
toString(): String
Measure
magnitude: String
day: LocalDate
dayMeasurements: List<HourMeasurement>
canEqual(Object): boolean
equals(Object): boolean
getDay(): LocalDate
getDayMean(): Float
getDayMeasurements(): List<HourMeasurement>
getMagnitude(): String
getMinValue(): Moment
hashCode(): int
setDayLocalDate(): void
setDayMeasurements(List<HourMeasurement>): void
setMagnitude(String): void
toString(): String
Moment
date: LocalDateTime
value: Float
canEqual(Object): boolean
equals(Object): boolean
getDate(): LocalDateTime
getValue(): Float
hashCode(): int
setDate(LocalDateTime): void
setValue(Float): void
toString(): String
LocalDateAdapter
marshal(LocalDate): String
unmarshal(String): LocalDate
LocalDateTimeAdapter
marshal(LocalDateTime): String
unmarshal(String): LocalDateTime
HourMeasurement
hour: int
value: float
validation: String
canEqual(Object): boolean
equals(Object): boolean
getHour(): int
getValidation(): String
getValue(): float
hashCode(): int
setHour(int): void
setValidation(String): void
setValue(float): void
toString(): String
RunnableXMLMeasuresReader
list: List<Measure>
station: Station
getList(): List<Measure>
run(): void
RunnableXMLStationReader
station: Station
city: String
getStation(): Station
run(): void
RunnableXMLReader
file: String
getFile(): String
Station
stationCode: String
stationZone: String
stationCity: String
canEqual(Object): boolean
equals(Object): boolean
getStationCity(): String
getStationCode(): String
getStationZone(): String
hashCode(): int
setStationCity(String): void
setStationCode(String): void
setStationZone(String): void
toString(): String
DataReading
SEPARATOR: String
DATA_DIR: String
createDirectory(String): Path
deleteDirectory(Path): void
getFile(String): JDOM
getMagnitudes(String): List<Magnitude>
getMeasures(Stream<Element>): List<Measure>
getMeasuresDataOfStation(Station, String): Stream<Element>
getStation(String, String): Optional<Station>
MarkDownGenerator
generateMarkdownOfCityMeans(List<CityMeans>, String): void
CityMeans
city: String
contaminationMeans: List<DataMean>
meteorologyMeans: List<DataMean>
canEqual(Object): boolean
equals(Object): boolean
getCity(): String
getContaminationMeans(): List<DataMean>
getMeteorologyMeans(): List<DataMean>
hashCode(): int
setCity(String): void
setContaminationMeans(List<DataMean>): void
setMeteorologyMeans(List<DataMean>): void
toString(): String
DataMean
mean: double
dataDescription: String
canEqual(Object): boolean
equals(Object): boolean
getDataDescription(): String
getMean(): double
hashCode(): int
setDataDescription(String): void
setMean(double): void
toString(): String
JDOM
document: Document
URI: String
fillDocumentDataFromXML(): void
genAnalyticsOfCity(String): List<CityMeans>
getDocument(): Document
getPrettyOutputter(): XMLOutputter
initDocument(): void
loadData(): void
printXMLFile(): void
writeXMLFile(String): void
RunnableXMLMagnitudesReader
list: List<Magnitude>
getList(): List<Magnitude>
run(): void
CSVReader
OUTPUT_URI: String
INPUT_URI: String
rootElementName: String
tagName: String
headersList: List<String>
addLineElement(String, JDOM): void
fillDocumentDataFromCSV(String, JDOM): void
generateXMLFilesFromCSV(): void
loadCSVData(): void
run(): void
Magnitude
codMagnitude: String
description: String
unit: String
canEqual(Object): boolean
equals(Object): boolean
getCodMagnitude(): String
getDescription(): String
getUnit(): String
hashCode(): int
setCodMagnitude(String): void
setDescription(String): void
setUnit(String): void
toString(): String
«create»