ACCESO A DATOS

RECICLAJE Y LIMPIEZA DE MADRID

Realizado por:

Azahara Blanco Rodríguez
Daniel Rodríguez Fernández

15/10/2022

Proyecto	Reciclaje y Limpieza de Madrid		
Entregable	Especificación de Requisitos		
Autor	Daniel Rodríguez Fernández y Azahara Blanco Rodríguez		
Versión/Edición	V1.6	**Fecha Versión**	15/10/2022
Aprobado por		**Fecha Aprobación**	
		Nº Total de Páginas	11

PROJECT MANAGER
Azahara Blanco Rodríguez Daniel Rodríguez Fernández

Tabla de contenido

1. Introducción

2. Descripción del problema propuesto

Nos ha contratado el Ayuntamiento de Madrid para que hagamos un estudio del análisis de limpieza y gestión de basuras que se hace en la ciudad.

Para ello nos han dado unos ficheros con la información deseada que debemos procesar.

Nuestro programa se debe llamar de la siguiente manera: java -jar basuras.jar opcion [parámetros_opcion]

Si la opción es parser directorio_origen directorio_destino: debe tomar los ficheros csv del directorio origen y los trasformalos en JSON y XML en el directorio destino. En dicho directorio destino deberán estar las tres versiones: CSV, JSON y XML.

Si la opción es resumen directorio_origen directorio_destino: debe tomar la información de los contenedores y de la recogida, independientemente de la extensión que tenga (si no corresponde a la extensión o al formato deberá indicar error) y deberá procesarla generando en directorio_destino un resumen.html, aplicándoles los estilos que creas oportunos, con la siguiente información:

- Titulo: Resumen de recogidas de basura y reciclaje de Madrid
- Fecha de generación: Fecha y hora en formato español.
- Autores: Nombre y apellidos de los dos autores.
- Número de contenedores de cada tipo que hay en cada distrito.
- Media de contenedores de cada tipo que hay en cada distrito.
- Gráfico con el total de contenedores por distrito.
- Media de toneladas anuales de recogidas por cada tipo de basura agrupadas por distrito.
- Gráfico de media de toneladas mensuales de recogida de basura por distrito.
- Máximo, mínimo , media y desviación de toneladas anuales de recogidas por cada tipo de basura agrupadas por distrito.
- Suma de todo lo recogido en un año por distrito.
- Por cada distrito obtener para cada tipo de residuo la cantidad recogida.
- Tiempo de generación del mismo en milisegundos.

Si la opción es resumen distrito directorio_origen directorio_destino: debe tomar la información de los contenedores y de la recogida, independientemente de la extensión que tenga (si no corresponde a la extensión o al formato deberá indicar error) y deberá procesarla generando en directorio_destino un resumen_distrito.html (solo si el distrito existe, si no deberá mostrar error), aplicándoles los estilos que creas oportunos, con la siguiente información:

- Titulo: Resumen de recogidas de basura y reciclaje de “Distrito”
- Fecha de generación: Fecha y hora en formato español.

- Autores: Nombre y apellidos de los dos autores.
- Número de contenedores de cada tipo que hay en este distrito.
- Total de toneladas recogidas en ese distrito por residuo.
- Gráfico con el total de toneladas por residuo en ese distrito.
- Máximo, mínimo , media y desviación por mes por residuo en dicho distrito.
- Gráfica del máximo, mínimo y media por meses en dicho distrito.
- Tiempo de generación del mismo en milisegundos.

Por cada ejecución debemos guardar un fichero bitacora.xml donde tengamos en este XML un listado de las ejecuciones con la siguiente información:

- ID de la ejecución en base a uuid
- Instante: Instante de la ejecución en formato ISO 8601
- Tipo de opción elegida (parser, resumen global, resumen ciudad).
- Éxito: si tuvo éxito o no su procesamiento.
- Tiempo de ejecución: tiempo de ejecución si tuvo éxito en milisegundos.

Tienes los PDFs necesarios donde te explica cómo y de qué manera la información está estructurada. Debes leerlos detalladamente para saber cómo y dónde está la información relevante y cómo está expresada. Antes de programar a lo loco, te recomiendo estudiar y analizar esta información para pensar cómo y de qué manera puedes hacer lo que se te pide de la forma más efectiva y eficiente. No hay una solución buena única, pero sí muchas malas. Tú decides.

3. Requisitos

3.1. Requisitos Funcionales

Cod	Requisito Funcional	Check
RF1	Debe permitirnos convertir el archivo a tres versiones diferentes	Ok
RF2	Debe obtener la información de los contenedores y de la recogida de residuos, siendo indiferente la extensión del archivo. Debe procesarla generando en directorio destino un resumen	Ok
RF3	Debe obtener la información de los contendores y de la recogida de residuos, siendo indiferente la extensión que tenga. Debe procesarla generando un resumen del distrito	Ok
RF4	Debe realizar por cada ejecución del programa el almacenamiento de los datos en un fichero bitácora	Ok

3.2. Requisitos No Funcionales

Cod	Requisito No Funcional	Check
RNF1	El programa se debe llamar *java -jar basuras.jar opción [parámetros]*	Ok
RNF2	Poder convertir un archivo con extensión CSV a JSON o a XML.	Ok
RNF3	El formato del resumen general debe ser en formato html.	Ok
RNF4	El resumen general debe llamarse resumen_distrito y estar en formato html.	Ok
RNF5	El fichero bitácora debe estar en el formato xml.	Ok
RNF6	El resumen general debe mostrar los siguientes datos en su informe: - Titulo: Resumen de recogidas de basura y reciclaje de Madrid - Fecha de generación: Fecha y hora en formato español. - Autores: Nombre y apellidos de los dos autores. - Número de contenedores de cada tipo que hay en cada distrito. - Media de contenedores de cada tipo que hay en cada distrito. - Gráfico con el total de contenedores por distrito. - Media de toneladas anuales de recogidas por cada tipo de basura agrupadas por distrito. - Gráfico de media de toneladas mensuales de recogida de basura por distrito. - Máximo, mínimo , media y desviación de toneladas anuales de recogidas por cada tipo de basura agrupadas por distrito. - Suma de todo lo recogido en un año por distrito. - Por cada distrito obtener para cada tipo de residuo la cantidad recogida. - Tiempo de generación del mismo en milisegundos.	Ok

RNF7	El resumen por distrito debe mostrar los siguientes datos en su informe: - Titulo: Resumen de recogidas de basura y reciclaje de “Distrito” - Fecha de generación: Fecha y hora en formato español. - Autores: Nombre y apellidos de los dos autores. - Número de contenedores de cada tipo que hay en este distrito. - Total de toneladas recogidas en ese distrito por residuo. - Gráfico con el total de toneladas por residuo en ese distrito. - Máximo, mínimo , media y desviación por mes por residuo en dicho distrito. - Gráfica del máximo, mínimo y media por meses en dicho distrito. - Tiempo de generación del mismo en milisegundos.	Ok
RNF8	El fichero bitácora debe mostrar los siguientes datos en su informe: - ID de la ejecución en base a uuid - Instante: Instante de la ejecución en formato ISO 8601 - Tipo de opción elegida (parser, resumen global, resumen ciudad). - Éxito: si tuvo éxito o no su procesamiento. - Tiempo de ejecución: tiempo de ejecución si tuvo éxito en milisegundos.	Ok

3.3. Requisitos de Información

Cod	Requisito de Información	Check
RI1	Class: ContenedoresVarios - codigoInternoSituado: String - tipoContenedor: String - modelo: String - descripcionModelo: String - cantidad: String - lote: String - distrito: String - barrio: String - tipoVia: String - nombre: String - numero: String - coordenadaX: String - coordenadaY: String - TAG: String	Ok
RI2	Class: ModeloResiduo - año: int - mes: Enum - lote: int - residuo: Enum - distrito: String - nombreDistrito: String - toneladas: int	Ok

4. Evaluación y Análisis

4.1. Clase POKO

Hemos utilizado dos clases que son las que representa a cada uno de los CSV que tenemos.

Clases:

ContenedoresVarios, recoge la información necesaria sobre cada contenedor que está situados por todo el municipio de Madrid.

Contiene los siguientes atributos:

Hemos decidido que los campos sean nulls para que a la hora de leer el CSV este vaciado podamos continuar sin que nos salgan excepciones.

```
codigoInternoSituado: String?,
tipoContenedor: String?,
modelo: String?,
descripcionModelo: String?,
cantidad: String?,
lote: String?,
distrito: String?,
barrio: String?,
tipoVia: String?,
nombre: String?,
numero: String?,
coordenadaX: String?,
coordenadaY: String?,
TAG: String?
```

ModeloResiduo, es la otra clase que tiene nuestro programa, en esta clase recogemos los datos necesarios sobre los residuos recogidos, el tipo de residuo y la cantidad en todos los distritos del municipio de Madrid.

Contiene los siguientes atributos:

Hemos decidido que los campos sean nulls para que a la hora de leer el CSV este vaciado podamos continuar sin que nos salgan excepciones.

Esta clase también incluye dos enums uno de ellos hace referencia a los Meses y el otro Enum que hemos aplicado hace referencia a los diferentes tipos de residuo que tenemos.

```
class ModeloResiduo (año:Int?, mes: Meses?, lote: Int?, residuo:
TipoResiduo?, distrito : String?, nombreDistrito: String?, toneladas: Int?)
```

4.2. Enums

Utilizamos dos Enums el primer de ellos es **Meses** este enum lo utilizamos en la clase ModeloResisudo para saber el mes en el que se recoge un residuo.

El otro Enum que utilizamos es **TipoResisudo** este enum lo utilizamos en la clase ModeloResiduo para saber el tipo de residuo que se ha recogido.

4.3. Interchange

En esta ruta tenemos las clases que se encargan de hacer la conversión entre los diferentes tipos de archivos que se nos solicitan poder convertir. Hemos implementado Serializable para poder realizar las conversiones entre CSV a JSON y XML y objetos. Es la zona de intercambio entre los diferentes tipos de archivos que tenemos.

4.4. DTO

Utilizamos la clase DTO para poder convertir entre los diferentes tipos de archivos que necesitamos xml, json o csv. Esta clase nos permite obtener todos los datos y convertirlos de sus tipos originales a los tipos necesarios para poder realizar la conversión.

Tenemos una clase DTO por cada Class Model que tengamos, en este caso tenemos **ContenedoresVariosDTO** y **ModeloResiduoDTO**.

Ambos implementan el método @Serializable que es propio de la librería de Kotlin y nos permite realizar la conversión a otro tipo de archivos como puede ser JSON o XML.

4.5. DataOfUse

Tenemos esta clase para crear nuestro fichero bitácora en formato xml, en esta clase tenemos los siguientes atributos id (que se genera automáticamente con un UUID), instanteFormatoISO (momento en el que se realiza la ejecución de alguna de las opciones en formato normalizado ISO), tipoOpcion (opción que se ha seleccionado), éxito(sí se ha ejecutado con éxito o no), tiempoDeEjecucion (tiempo que ha tardado en generarse el informe).

Todos estos datos los almacenaremos en nuestro archivo bitácora.xml cada vez que hagamos la ejecución de algunas de las opciones del programa.

```
class DataofUse(tipoOpcion : String, exito: Boolean, tiempoEjecucion :
Long) {
    val id = UUID.randomUUID()
    val instanteFormatoISO  =
LocalDateTime.now().format(DateTimeFormatter.ISO_DATE_TIME)
    val tipoOpcion : String =  tipoOpcion
    var exito : Boolean = exito
    val tiempoDeEjecucion : Long = tiempoEjecucion

    override fun toString(): String {
        return "DataofUse(id=$id,
instanteFormatoISO='$instanteFormatoISO', tipoOpcion='$tipoOpcion',
exito=$exito, tiempoDeEjecucion=$tiempoDeEjecucion)"
    }
}
```

4.6. HTML

Tenemos dos Clases diferentes que hacen referencia al HTML. El primero de ellos llamado **CreateFileHTML** con esta clase crearemos el fichero html y la ubicación donde se va a almacenar. La segunda clase llamada **CreateHTML** contiene el código en HTML y el diseño y estructura del fichero que se creará cada vez que saquemos el resumen.

4.7. DataframeUtils

Utilizamos Lets Plot que es una librería para diseñar gráficas a través de datos estadísticos que hemos recogido de nuestro CSV y hemos analizado con los Data Frame.

También encontramos las clases necesarias para analizar esos datos solicitados para después realizar las gráficas.

4.8. Resume

En esta clase hemos realizado todas las consultas que nos solicita el enunciado del problema para sacar las estadísticas utilizando la librería de Dataframe que es implementada en Kotlin con ella hemos conseguido sacar los datos solicitados.

4.9. Mappers

Utilizamos los mappers para cada Class Model de nuestro programa en una clase DTO y de esta forma parsear los datos, para después poder convertir nuestro CSV en un archivo JSON o XML.

4.10. CheckData

Utilizamos esta clase para comprobar que los datos que ha introducido el usuario a través de los args son correctos y comprobar que todo funciona bien y en caso de que introduzca un parámetro erróneo tenerlo controlado y que le salga un aviso.

4.11. Main

Se trata de la clase principal de nuestro programa, contine el menú principal con el cual va a interactuar el usuario, también contiene las funciones que hace que funcione cada una de las funciones principales del menú. Este menú está comprobando los datos por la clase CheckData de que todos los datos que introduzca el usuario sean correctos.

5. Enlace al vídeo y Proyecto

https://www.youtube.com/watch?v=JiJI95WEWfo

https://github.com/idanirf/AD-P1-drodriguez-ablanco

6. Referencias

https://github.com/JetBrains/lets-plot

https://kotlinlang.org/docs/home.html

https://package-search.jetbrains.com/package?id=org.jetbrains.kotlinx%3Adataframe

https://kotlin.github.io/dataframe/overview.html#syntax

https://github.com/joseluisgs/FP-NextGen-AccesoDatos

https://github.com/joseluisgs/Kotlin-LetsPlot-DataFrames