

---

# Reciclaje y Limpieza de Madrid

## Práctica Acceso a Datos

Mohamed Asidah & Rocío Palao



---

<b>DESCRIPCIÓN</b>	<b>3</b>
<b>PROPUESTA DE SOLUCIÓN</b>	<b>3</b>
<b>Diseño</b>	<b>3</b>
<b>Clases</b>	<b>4</b>
Modelos	4
Residuos	4
Contenedores	5
DTO	5
ResiduosDTO:	5
ContenedoresDTO:	6
Mapeadores:	7
Mapeador de residuos:	7
Mapeador de contenedores:	8
Mapeador de bitácora:	9
Controladores:	10
BasuraController:	10
DataFrameController:	11
<b>Transformación de formatos</b>	<b>12</b>
Lectura y escritura de CSV	12
Lectura y escritura de Json	12
Lectura y escritura de XML	12
<b>Realización de consultas</b>	<b>13</b>
<b>Gráficas</b>	<b>13</b>

---

## **DESCRIPCIÓN**

En este proyecto vamos a realizar el tratamiento de los datos sobre el tratamiento de residuos en la comunidad de Madrid. Contamos con datos sobre los contenedores de residuos y los distintos tipos de residuos que se tiran mensualmente en la comunidad. Los datos que trataremos son del año 2022.

El programa tiene que ser capaz de realizar tres operaciones:

- Parsear: dado un directorio de origen, tiene que poder leer los archivos csv que hay ahí y convertirlos a Json, XML y CSV en un archivo destino dado.
- Resumen: recibiendo como argumento de entrada un directorio origen y uno destino, se deben conseguir los datos de ambos residuos y contenedores, realizar un conjunto de consultas y gráficas y luego generar en el destino un HTML con los resultados.
- Resumen distrito: Realizará lo mismo que resumen, pero acotandolo a un solo distrito y realizando otro conjunto distinto de consultas.

**Importante: El directorio destino a la hora de crear un resumen ya sea específico a un distrito o no debe o ser un directorio vacío, o un directorio inexistente que se creará durante la ejecución, o un directorio donde no se ha realizado antes un resumen**

## **PROPUESTA DE SOLUCIÓN**

Para la solución atacaremos el problema en cuatro partes:

- Lectura/escritura de archivos CSV.
- Lectura/escritura de archivos Json/XML.
- Ejecución de las consultas.
- Generación de HTML con las consultas.

---

# **Diseño**

El diseño que vamos a seguir es un diseño Modelo-Mapeador-Controlador.

- Modelo: serán las clases encargadas de almacenar los datos sobre los contenedores y los residuos.
- Mapeadores: estas clases serán usadas para leer y escribir los distintos archivos de intercambio que vamos a utilizar y para convertir desde los modelos a sus versiones DTO (data transfer object).
- Controlador: tendremos dos y serán los encargados de casar los elementos de los modelos con los mapeadores y de contener parte de la lógica del programa.

---

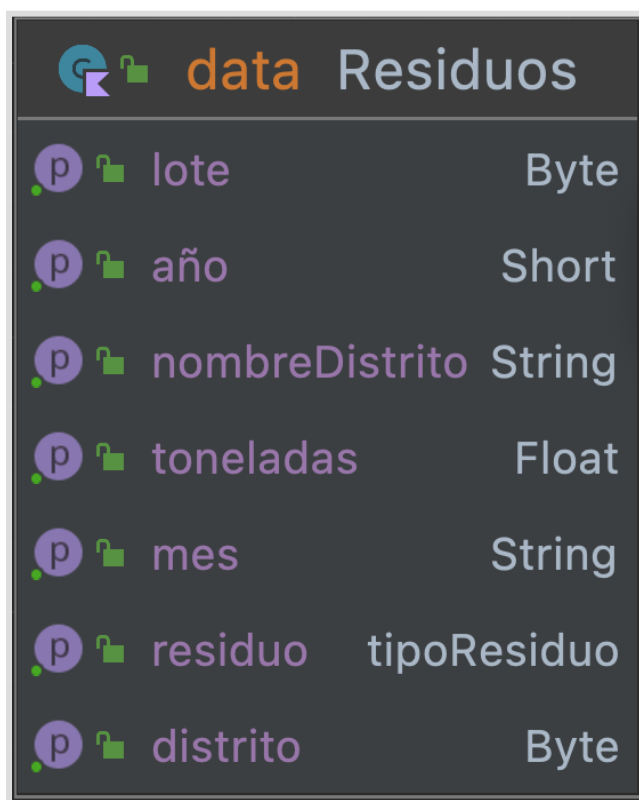
## Clases

Las clases utilizadas en el proyecto están divididas en:

### Modelos

Los encargados de almacenar los datos sobre residuos, contenedores y la bitácora de ejecución:

#### *Residuos*



	data	Residuos
• p	lote	Byte
• p	año	Short
• p	nombreDistrito	String
• p	toneladas	Float
• p	mes	String
• p	residuo	tipoResiduo
• p	distrito	Byte

Almacenará los datos sobre los residuos. Estos datos están especificados en el pdf sobre los datos que la Comunidad de Madrid pone a disposición de la gente.

## Contenedores

data Contenedor		
tipoVia		String
cantidad		Int
codigoSituacion		String
numeroVia		Int
descripcionModelo		String
barrio		String
lote		Int
modeloContenedor		String
nombreVia		String
coordenadasGeo	Pair<Double , Double >	
distrito		String
coordenadas	Pair<Double , Double >	
tipoContenedor	TipoContenedor	

Almacena los datos sobre los contenedores que hay repartidos por toda la comunidad.

## DTO

### ResiduosDTO:


data ResiduosDTO		
toLine ()		String
lote		Byte
año		Short

---

Este tipo lo usamos para traspasar entre el ordenador y los archivos de intercambio.

### *ContenedoresDTO:*

Utilizado para traspasar la información entre el ordenador y los archivos.



data	ContenedorDTO
toLine ()	String
tipoVia	String
tipoContenedor	String
longitud	Double
cantidad	Int
direccion	String
latitud	Double
codigoSituacion	String
numeroVia	Int
descripcionModelo	String
barrio	String
lote	Int
modeloContenedor	String
nombreVia	String
coordX	Double
coodY	Double
distrito	String

## Mapeadores:

*Mapeador de residuos:*

ResiduosMapper		
m	toXml(String, ListaResiduosDTO)	Unit
m	mapToResiduo(String)	ResiduosDTO
m	fromJson(String)	ListaResiduosDTO
m	fromDto(ResiduosDTO)	Residuos
m	readCsvResiduo(String)	List<ResiduosDTO>
m	checkCSV(File)	Boolean
m	fromXml(String)	List<ResiduosDTO>
m	toJson(String, ListaResiduosDTO)	Unit
m	toTipoResiduo(String)	tipoResiduo
m	mapListToDTO(List<Residuos>)	List<ResiduosDTO>
m	toDto(Residuos)	ResiduosDTO
m	writeCsvResiduo(ListaResiduosDTO, String)	Unit
m	mapListFromDTO(List<ResiduosDTO>)	List<Residuos>
P	CABECERA	String

El mapeador de residuos se encargará de convertir de residuos a DTO y viceversa, y, de convertir entre los distintos tipos archivos de intercambio que vamos a utilizar.



### *Mapeador de contenedores:*

ContenedorMapper		
fromDto(ContenedorDTO)		Contenedor
toXML(String, ListaContenedorDTO)		Unit
checkRutaCSV(String)		Boolean
toJson(String, ListaContenedorDTO)		Unit
mapListFromDTO(List<ContenedorDTO>)		List<Contenedor>
toDTO(Contenedor)		ContenedorDTO
fromJson(String)		ListaContenedorDTO
mapContenedorDTO(List<String>)		ContenedorDTO
writeCsv(List<ContenedorDTO>, String)		Unit
mapListToDTO(List<Contenedor>)		List<ContenedorDTO>
checkCSV(File)		Boolean
readCSV(String)		List<ContenedorDTO>
fromXML(String)		ListaContenedorDTO

El mapeador de residuos se encargará de convertir de residuos a DTO y viceversa, y, de convertir entre los distintos tipos archivos de intercambio que vamos a utilizar.

### *Mapeador de bitácora:*

BitacoraMapper		
makeBitacora(Bitacora)		Unit
toXml(String, ListaBitacora)		Unit

---

El apearador de bitácora se encargará de traspasar las bitácoras creadas a un archivo XML y de leer un archivo XML con las bitacoras

---

## Controladores:

































*BasuraController:*

BasuraController		
m	executeCommand (String [])	Boolean
m	cabeceraResiduos (String )	Boolean
m	retrieveXml (String )	List<File>
m	cabeceraContenedores (String )	Boolean
m	retrieveJson (String )	List<File>
m	readContenedoresCsv (File )	List<ContenedorDTO >
m	readResiduosCsv (File )	List<ResiduosDTO >
m	checkPath (String )	Boolean
m	getOption (String [])	Int
m	resumen (String , String , String )	Boolean
m	parser (String , String )	Boolean
m	retrieveCsv (String )	List<File>
p	contenedorMapper	ContenedorMapper
p	residuosMapper	ResiduosMapper

Encargado de combinar todas las funciones de los mapeadores y los modelos para realizar la lógica del programa.

---

## *DataFrameController:*

DataframeController		
 	consultaEstadisticasDistrito (String )	String
 	consultaMediaToneladasAnuales ()	String
 	resumenDistrito (String )	String
 	resumen ()	String
 	graficoContenedoresDistrito ()	Unit
 	graficoToneladasResiduoDistrito (String )	Unit
 	consultaMaxMinMedDesvToneladasAnuales ()	String
 	consultaMediaContenedoresTipoDistrito ()	String
 	graficoMaxMinMediaMesDistrito (String )	Unit
 	consultaNumeroContenedoresTipoDistrito (String )	String
 	consultaSumaAñoDistrito ()	String
 	consultaCantidadResiduoDistrito ()	String
 	consultaNumContenedoresTipoDistrito ()	String
 	consultaToneladasDistrito (String )	String
 	compararDistrito (String , String )	Boolean
 	graficoMediaToneladasMensuales ()	Unit

Encargado de realizar todas las consultas sobre los datos y de generar las gráficas necesarias.

---

# **Transformación de formatos**

Hemos utilizado dos herramientas para las distintas transformaciones entre formatos y en ningún caso se hace una transformación directa. Las transformaciones siempre son de archivo a datos o de datos a archivo. Las transformaciones realizadas han sido:

## **Lectura y escritura de CSV**

Para leer los archivos CSV hemos utilizado operaciones sobre colecciones de Kotlin porque es la manera más sencilla que conocemos. Una vez leídos los datos desde el csv, los almacenamos en un DTO que puede luego ser convertido a su otro modelo para realizar las operaciones y para escribirlos de nuevo a CSV.

## **Lectura y escritura de Json**

Para leer los archivos de tipo Json usamos Kotlin Serialization por su sencillez y eficacia. Al igual que los csv, se lee desde el archivo y se almacena en un DTO que luego puede ser usado o convertido.

## **Lectura y escritura de XML**

Para leer los archivos de tipo xml usamos Kotlin Serialization por su sencillez y eficacia. Al igual que los csv, se lee desde el archivo y se almacena en un DTO que luego puede ser usado o convertido.

---

## Realización de consultas

Las consultas las realizaremos convirtiendo una lista de ModelosDTO a un DataFrame de Modelos. Hemos utilizado los DataFrame porque nos permiten hacer las consultas muy fácilmente y además nos otorgan herramientas que las listas por ejemplo no nos dan como la generación automática de un HTML con la tabla de las consultas y la posibilidad de generar gráficas fácilmente.

Para manejar los DataFrame usamos la librería proporcionada directamente por JettBrains.

## Gráficas

Para las gráficas usaremos la librería de lets Plot. Esta librería además de aportar un gran número de gráficas que podemos realizar, hace que sea más fácil y es un estándar en la industria.

Aquí un ejemplo de una gráfica:

