Iván Azagra y Daniel Rodríguez

DOCUMENTACIÓN DEL TRABAJO DE ACCESO A DATOS REALIZADO POR IVÁN AZAGRA Y DANIEL GARCÍA

Documentación ad-practica-p01-procesamientodatos-2022-2023:

Trabajo acceso a datos

Contenido

[DataProcessor: 2](#_Toc117112447)

[TemplateGenerator: 2](#_Toc117112448)

[CSVReader: 2](#_Toc117112449)

[BitacoraCreator: 2](#_Toc117112450)

[Contenedor: 3](#_Toc117112451)

[Ejecucion: 3](#_Toc117112452)

[Ejecuciones: 3](#_Toc117112453)

[Residuos: 3](#_Toc117112454)

[Tipos: 3](#_Toc117112455)

[CSVParser: 4](#_Toc117112456)

[Util: 4](#_Toc117112457)

[Main: 4](#_Toc117112458)

DataProcessor:

Clase que se dedica a procesar los datos para sacar las consultas y gráficos, cuenta de tres métodos.

* El primer método es dataToDataFrame() en el que se hacen la mayoría de las consultas requeridas
* El segundo método es graphics() donde se generan todos los gráficos requeridos por resumen
* El tercer método es graphicsDistrito() donde se generan los gráficos requeridos para el resumen distrito

Hemos optado por la utilización de DataFrames y letsplot para la realización de esta clase, ya que era la forma más rápida tanto a la hora de programarlo como en la ejecución, ya que usando api stream teníamos una clase larguísima e innecesariamente complicada para sacar los resultados, los cuales eran bastante dudables por mi poca soltura con dicha api stream.

TemplateGenerator:

Clase que se dedica a generar un template html en el que se depositan todos los datos de las consultas requeridos para la generación del documento que se mostrará en el navegador, consta de dos métodos.

* El primer método es generateHTML() en el que se genera el archivo html en el que después se escribirá en el siguiente método
* El segundo método es writeHTML() en el que escribe el contenido

CSVReader:

Clase que se encarga de leer los csv y chequear si es posible leerlos, esta clase introduce los datos de los csv leídos a una lista que después usamos para el procesado de datos en DataProcessor. La clase cuenta también con los parseadores de tipos de residuo y contenedor para evitar los conflictos que dan los enumeradores al no ser Strings comparables con los valores del csv, también por esto mismo usamos los enums con un String acoplado a ellos para poder hacer esa comparación entre valor y tipo y poder hacer la distinción posterior para pasarlo a la lista.

También cuenta con una función que evita el doble delimitador para ahorrarnos los valores nulos.

BitacoraCreator:

En esta clase nos encargamos de crear el archivo xml donde se guardan los datos de ejecución, consta de 3 funciones.

* saveIntoBitacora(): Coge los datos del POKO de ejecución y los introduce en el archivo, comprueba si existe el directorio y si no es así lo crea, también si no existe el archivo llama al método createCosas() para que se encargue de introducir los datos usando una lista de ejecuciones. Si existe el archivo pasa los datos con el método resultsToXML()
* resultsToXML(): se encarga de pasar los datos usando las funciones de la librería JAXB
* createCosas(): Coge las ejecuciones de una lista y los introduce en el XML

Contenedor:

Clase POKO de contenedor, implementa las anotaciones para poder funcionar con JAXB. Esta clase guarda los datos de los contenedores que se encuentran en el csv y que nos eran necesarios, por lo que hemos prescindido de los valores no necesarios como las coordenadas y la dirección, ya que en caso de necesitar esta última se puede sacar simplemente concatenando los strings que componen este mismo campo. En la clase igualamos los campos con un constructor()

Ejecucion:

Clase POKO de ejecución con las anotaciones de JAXB, contiene el constructor donde se le pasan los parámetros de la clase y las anotaciones en los atributos necesarios para pasárselo al XML.

Ejecuciones:

Clase POKO que consta únicamente de una lista de objetos Ejecucion. Hace la función de elemento “root” del XML que será generado por JAXB.

Residuos:

Clase POKO con los datos de los residuos del csv necesarios para las consultas, contiene las anotaciones DataSchema para usar DataFrames, y las anotaciones para usarlo con JAXB y pasar esos atributos al XML que generamos en ejecución.

Tipos:

Los pongo juntos porque su contenido es muy pequeño, básicamente se limitan a ser un enumerador para comparar los tipos de contenedor y residuos que parseamos en el lector de csv.

CSVParser:

Clase encargada de parsear los csv (compatibles con el modelo de residuos o contenedores) de la carpeta origen, pasándose a csv sin valores nulos ni dobles delimitadores, a xml y a json, en la carpeta de destino que se le haya asignado por constructor.

Antes de empezar a leer, se asegura de que el directorio de origen sea un directorio, exista y no esté vacío y que contenga ambos csv. De lo contrario sale del programa.

Para el directorio destino, si existe, se asegura de que sea un directorio, y si lo es, pide por consola confirmación para borrarlo, ya que si existe previamente, lo borrará para dar paso a la nueva ejecución. Si el usuario se lo deniega, sale del programa, y si al intentar borrarlo no pudiera, sale del programa con el código 1708. Tras un borrado exitoso, este método se llama a sí mismo recursivamente. Finalmente, cuando el directorio no exista, crea dicho directorio.

Una vez está todo esto listo, procede a hacer su función principal, parsear:

Esta clase utiliza corrutinas debido a que parsear a los tres tipos de archivo es una tarea que puede ser realizada concurrentemente. De esta manera, se usa un runBlocking{ } para bloquear la ejecución de las corrutinas y que dejen de existir después de que termine dicho runBlocking. Acto seguido, se lanzan tres corrutinas mediante launch { } en el Dispatcher de Input/Output (porque como van a hacer operaciones de lectura/escritura, es el que más sentido tiene usar), y cada corrutina parsea a un tipo de archivo distinto: la que parsea a xml usa JAXB, la que parsea a json usa Gson y la que parsea a csv lee el csv pasándolo a lista de objetos y esa misma lista la vuelca en el csv destino. Una vez acaban las tres, se unen mediante join y terminan. Si todo salio bien, el metodo parse() devolverá un 0; y por cada cosa que haya salido mal, devolverá un número mayor.

Util:

Clase Object que se usa para obtener los tiempos de ejecución y la fecha del momento dado en formato español

Main:

La clase main se encarga de la ejecución de todas las clases y sus funciones, se crea un atributo de clase Ejecucion en la que se registra la ejecución actual, con su tiempo y opción elegida. Compara el número de argumentos y el texto introducido mediante la consola de comandos para realizar la acción deseada por el usuario, haciendo la aplicación una client line interface la cual genera a través de sus funcionalidades una web con los datos para poder visualizarlos de forma sencilla, son imágenes de las gráficas y datos ordenados.

Así el main se encarga de hacer la comprobación de la existencia de los directorios y archivos necesarios, como la comprobación de si se trata de un directorio o no para evitar que la aplicación reviente, la aplicación consta de 3 opciones comprobadas a través de esta clase, que son la opción de parsear y generar distintos documentos a partir de los csv dados, puede generar un csv parseado sin espacios, un json y el xml.