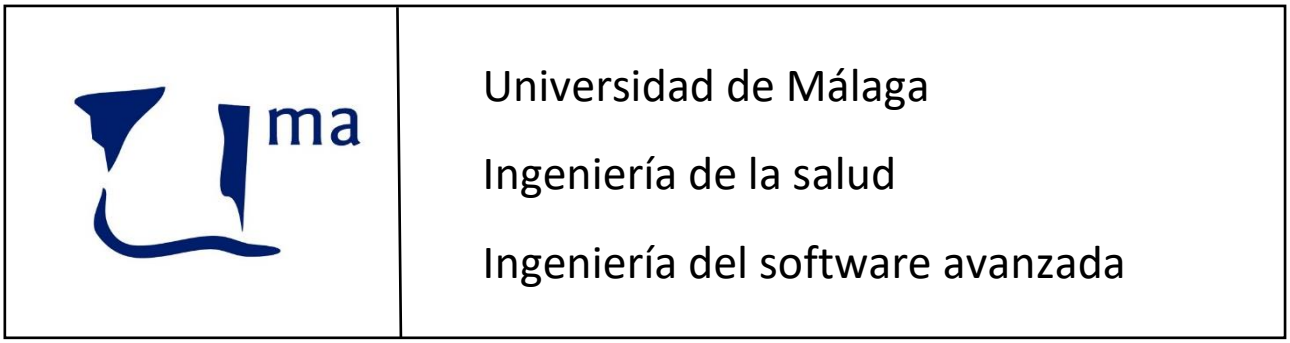


“Aplicación del patrón cadena de mando a JSon”

Profesor:	José María Álvarez Palomo
Estudiante:	Juan Sánchez Rodríguez

Tabla de contenidos

1. Importación de datos

En esta práctica vamos a aplicar la cadena de mando a las clases para la lectura de archivos JSon, lo primero que hicimos en clase fue crear el paquete *salud.isa.gsonMedDB* importar las clases *DatabaseJSonReader.java, GsonDatabaseClient.java,* los archivos de datos (datos.*json y datosshort.json*) a la carpeta raíz del proyecto y la librería *gson-2.8.6.jar* a *Classpath* para que se puedan leer los archivos de este tipo.

2. Inspeccion implementación sin cadena de mando

Una vez importados todo lo necesario comenzamos con la inspección del código para la aplicación del patrón cadena de mando. La clase *GsonDatabaseClient,* es la clase principal donde nos encontramos un único método, el *public static void main* en el que llamamos al método *public String parse(String jsonFileName)*de la clase *DatabaseJSonReader.java* que es la que empieza a recorrer el archivo JSon comparando las categorías.

En *DatabaseJSonReader.java* nos encontramos también los demás métodos para la lectura de los archivos a los que se irá accediendo, dependiendo de la categoría comparada en *public String parse(String jsonFileName).*

3. Objetivo

El objetivo es aplicar la cadena de mando de tal forma que cumpla con el principio abierto-cerrado creando clases especiales para cada una de las lecturas de cada categoría, y de esta manera poder añadir/editar las categorías sin tener que modificar el código ya implementado.

4. Aplicación cadena de mand

4.1. Clase ElementoCadenadeMando

El primer paso que he tomado ha sido crear la clase publica *ElementoCadenadeMando* con su constructor en el que declaro el ElementoCadenadeMando sig que será el que apunte al siguiente en cada iteración, como podemos comprobar en el método leerJson que a su vez llama al método leerJson de la siguiente categoría en el return *sig.leerJson(name, reader)*.

```
package salud.isa.gsonMedDB;
import java.io.IOException;
import com.google.gson.stream.JsonReader;
public class ElementodeCadenadeMando {
	protected ElementodeCadenadeMando sig;
	public ElementodeCadenadeMando(ElementodeCadenadeMando s) {
		sig=s;
	}
	public StringBuffer leerJson(String name, JsonReader reader) throws IOException {
		return sig.leerJson(name, reader);
	}
}
```

4.2. Clase DatabaseJSonReader

Una vez hecho la clase *ElementoCadenadeMando,* voy con la clase *DatabaseJSonReader* donde creo un único atributo de la clase ElementoCadenadeMando que lo inicializamos en el constructor.

Esta clase tiene dos métodos, un *void setClienteCadenadeMando*, para actualizar el elemento de la cadena de mando y el método *parse* en el que se inicia la lectura del archivo JSon donde nos volvemos a encontrar el método cadenademando.leerJson(String name, JsonReader reader) que apunta al siguiente para realizar la lectura completa de los datos mientras siga teniendo líneas, este método pasará por todas las clases de las diferentes categorías, y si coinciden *name* con el nombre de las categorías dse llevará a cabo la lectura que como podemos ver nos devuelve un String(readData);

```
package salud.isa.gsonMedDB;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import com.google.gson.stream.JsonReader;
public class DatabaseJSonReader {
	ElementodeCadenadeMando cadenademando;
	public DatabaseJSonReader(ElementodeCadenadeMando cm) {
		cadenademando=cm;
	}
	public void setCadenaDeMando(ElementodeCadenadeMando ncm) {
		cadenademando=ncm;
	}
	public String parse(String jsonFileName) throws IOException {
		InputStream usersIS = new FileInputStream(new File(jsonFileName));
		JsonReader reader = new JsonReader(new InputStreamReader(usersIS, "UTF-8"));
		reader.beginObject();
		StringBuffer readData = new StringBuffer();
		while (reader.hasNext()) {
			String name = reader.nextName();
			readData.append(cadenademando.leerJson(name,reader)).append("\n");
		}
		reader.endObject();
		reader.close();
		usersIS.close();
		return new String(readData);
	}
}
```

4.3. Clase ejemplo Categorías

Una vez acabada la explicación de la clase DatabaseJSonReader, queda por explicar como se lleva a cabo la lectura de cada categoría, todas las categorías siguen la misma estructura compartiendo gran parte de implementación por lo que solo explicaré una completa que sive para generalizar en las demás.

La clase elegida será medicinePresentations, el primer paso será declarar una variable estática que se corresponderá al nombre de la categoría de la clase en este caso "*medicinePresentations*" , y después las demás variables estáticas de los nombres de los objetos de la categoría, en este caso *"medicineRef", "activeIngRef", "inhalerRef", "dose" y "posologyRef.*

Después creamos un ElementoCadenadeMando son el *super(s)* y nos encontramos con dos métodos para la lectura de los datos, el primero es el *leerJSon* del que ya habíamos hablado antes, donde comparamos el nombre con la categoría, si coinciden seguiremos con la lectura de la categoría hasta que no queden líneas con otro método *medicinePresentationEntry(reader)* que explicaré luego y si no coinciden las categorías hay dos opciones, que haya una categoría siguiente para leer, o no, que en ese caso sería la última y acaba la lectura. Este método devuelve un *StringBuffer* que puede ser la lectura de la categoría de la clase, la lectura de la categoría de la siguiente clase, o un StringBuffer en blanco.

El otro método que nos encontramos es el que lee el interior de la categoría, es decir, los diferentes objetos que hay dentro.

Para la lectura, primero creamos elementos de tipo String, uno por cada objeto que tiene la categoría, después se irá comparando el nombre del objeto, con el parámetro dado y si coinciden se devolverá el valor que corresponde a cada objeto y si no, se pasará al siguiente mientras queden líneas por comparar.

Estos objetos pueden ser recibidos como un String, o como un StringBuffer que en ese caso debemos acceder a los Strings que hay dentro para poder obtener el valor, introducirlos en los diferentes Strings declarados previamente para devolverlos en el return de este método.

```
package salud.isa.gsonMedDB;
import java.io.IOException;
import com.google.gson.stream.JsonReader;
public class medicinePresentations extends ElementodeCadenadeMando {
	private static final String MEDPRES_TAGNAME = "medicinePresentations";
	private static final String MEDREF_FIELD_TAGNAME = "medicineRef";
	private static final String ACTINGREF_FIELD_TAGNAME = "activeIngRef";
	private static final String INHREF_FIELD_TAGNAME = "inhalerRef";
	private static final String DOSE_FIELD_TAGNAME = "dose";
	private static final String POSOLREF_FIELD_TAGNAME = "posologyRef";
	private static final String FIELD_SEPARATOR = ";";
	public medicinePresentations(ElementodeCadenadeMando s) {
		super(s);
	}
	public StringBuffer leerJson(String name, JsonReader reader) throws IOException {
		if(name.equals(MEDPRES_TAGNAME)) {
			StringBuffer medicinePresentationData = new StringBuffer();
			reader.beginArray();
			while (reader.hasNext()) {
				reader.beginObject();

	medicinePresentationData.append(medicinePresentationEntry(reader)).append("\n");
				reader.endObject();
			}
			medicinePresentationData.append("\n");
			reader.endArray();
			return medicinePresentationData;
	}
```

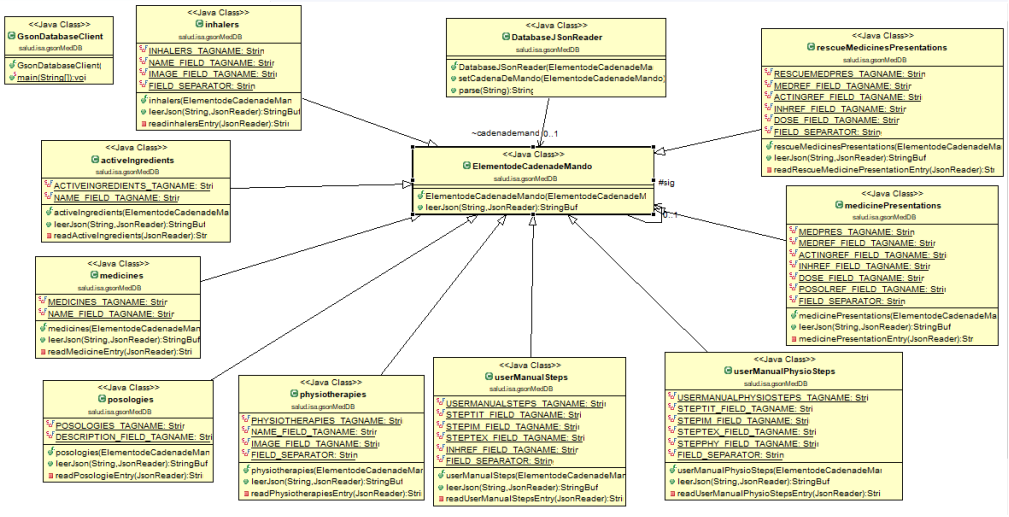
```
	else {
			if(sig!=null) {
				return super.leerJson(name, reader);
			}

	else {
				reader.skipValue();
				System.err.println("Category " + name + " not processed.");
				StringBuffer acabado = new StringBuffer();
				return acabado;
			}
		}
	}
private String medicinePresentationEntry(JsonReader reader) throws IOException {
		String medRef = null;
		String actRef = null;
		String inhRef = null;
		String dose = null;
		String posRef = null;
		while (reader.hasNext()) {
			String name = reader.nextName();
			if (name.equals(MEDREF_FIELD_TAGNAME)) {
				medRef = reader.nextString();
			} else if (name.equals(ACTINGREF_FIELD_TAGNAME)) {
				actRef = reader.nextString();
			} else if (name.equals(INHREF_FIELD_TAGNAME)) {
				StringBuffer res = new StringBuffer();
				reader.beginArray();
				while (reader.hasNext()) {
					res.append(reader.nextString()).append(",");
				}
				reader.endArray();
				res.deleteCharAt(res.length() - 1);
				inhRef = new String(res);
			} else if (name.equals(DOSE_FIELD_TAGNAME)) {
				StringBuffer res = new StringBuffer();
				reader.beginArray();
				while (reader.hasNext()) {
					res.append(reader.nextString()).append(",");
				}
				reader.endArray();
				res.deleteCharAt(res.length() - 1);
				dose = new String(res);
			} else if (name.equals(POSOLREF_FIELD_TAGNAME)) {
				StringBuffer res = new StringBuffer();
				reader.beginArray();
				while (reader.hasNext()) {
					res.append(reader.nextString()).append(",");
				}
				reader.endArray();
				res.deleteCharAt(res.length() - 1);
				posRef = new String(res);
			} else {
				reader.skipValue();
			}
		}
		return medRef + FIELD_SEPARATOR +"  "  + actRef + FIELD_SEPARATOR+"  " + inhRef +
FIELD_SEPARATOR+"  " + dose +FIELD_SEPARATOR+"  " + posRef;
	}
```

Como hemos podido comprobar, aplicando la cadena de mando obtenemos un código ordenado, fácilmente modificable/actualizable y que cumple con el principio abierto-cerrado en todas sus clases.

5. Diagrama de clases

Aquí muestro el diagrama de clases del proyecto, también estará disponible en GitHub con las demás clases donde se muestran las relaciones que hay entre ellas.



6. Conclusión

Esta práctica me ha resultado muy interesante y ha sido una buena forma de trabajar el patrón cadena de mando, he aprendido a trabajar con los datos JSon que no conocía y creo que he afianzado bien la teoría.

La realización de esta práctica me ha resultado menos dificultosa que la anterior, ya que no he tenido que desarrollar tanto código como en la anterior, aunque haya tenido que hacer tantas categorías. También me ha ayudado mucho la explicación del patrón cadena de mando en el video-tutorial subido a *microsoftstream* donde estaba todo muy claro explicado.

Para resumir, me gustaría decir que es un patrón que me ha resultado bastante interesante a tener en cuenta cuando tenga que realizar alguna implementación ya que quita muchos problemas.

7. Enlace repositorio GitHub

https://github.com/Juansanchezrodriguez1999/Proyecto-Aplicacion-Cadena-de-Mando