Practica 3

Aplicación del patrón cadena de mando a JSon

Stephan Charles Nielson

Introducción

En esta practica vamos a refactorizar las clases DatabaseJSonReader y GsonDatabaseClient para que cumplan el principio abierto/cerrado.

Para ello vamos a empezar con la clase DatabaseJSonReader, en un principio funcionaba así:

```
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();

        if (name.equals(MEDICINES_TAGNAME)) {
            readData.append(readMedicines(reader)).append("\n");
        } else if (name.equals(RESCUEMEDPRES_TAGNAME)) {
            readData.append(readRescueMedicinePresentations(reader)).append("\n");
        } else {
            reader.skipValue();
            System.err.println("Category " + name + " not processed.");
        }
    }

    reader.endObject();
    reader.close();
    usersIS.close();

    return new String(readData);
}
```

Usaba los if para identificar la entrada y luego redirigía al método adecuado, readMedicines o readRescueMedicinePresentations.

Ahora, voy a usar un método leeCategoria que en un principio formara parte de la clase abstracta CadenaDeMando, y luego se implementara en las clases hijas

```
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(name.toUpperCase()).append("\n").append(leeCategoria(reader, name)).append("\n");
    }

    reader.endObject();
    reader.close();
    usersIS.close();

    return new String(readData);
}
```

También eliminaremos todos los métodos que leían el archivo ya que esa tarea la van a llevar a cabo las clases que heredan de CadenaDeMando.

Cadena De Mando

CadenaDeMando será abstracta, y tendrá un atributo sucesor, que nos permitirá aplicar el patrón cadena de mando al tener la siguiente estructura de constructor:

```
public abstract class CadenaDeMando {
    private CadenaDeMando sucesor;

    public CadenaDeMando(CadenaDeMando siguiente) {
        this.sucesor = siguiente;
    }
```

Este método tendrá 4 métodos.
El primero sera leeCategoria, del cual parte se implementará en las clases hijas.

```
public StringBuffer leeCategoria(JsonReader reader, String nombre) throws IOException {
    return sucesor.leeCategoria(reader, nombre);
}
```

El segundo, codigoComun, tiene una función común a todas las clases hijas, empieza la lectura y crea un objeto Stringbuffer donde se introducirá el string que devolverá la siguiente clase al leer el archivo JSon

```
public StringBuffer codigoComun(JsonReader reader, String nombre) throws IOException {
    StringBuffer datos = new StringBuffer();
    reader.beginArray();
    while (reader.hasNext()) {
        reader.beginObject();
        datos.append(leeEntry(reader)).append("\n");
        reader.endObject();
    }
    datos.append("\n");
    reader.endArray();
    return datos;
}
```

El tercera método es leeEntry, leerá el archivo y devolverá los datos en un string. Sera implementada específicamente en cada clase hija.

```
public abstract String leeEntry(JsonReader reader) throws IOException;
```

Por ultimo necesitaremos un getter para acceder al atributo sucesor

```java
public CadenaDeMando getSucesor() {
    return sucesor;
}
```

Clases Hijas

Ahora voy a describir de forma general las clases hijas, no voy a describirlas todas ya que son prácticamente iguales entre si salvo alguna excepción en el método leeEntry.

Primero tendremos como atributos los string que usaremos para reconocer la entrada como la adecuada para la clase que corresponda:

```java
public class Medicines extends CadenaDeMando {
    private static final String MEDICINES_TAGNAME = "medicines";
    private static final String NAME_FIELD_TAGNAME = "name";
```

Luego, implementamos el método leeCategoria, que en caso de reconocer la sección como la correcta para la clase ejecuta el código común y si es incorrecta pasa a la siguiente clase, o en caso de ser esta la ultima, devuelve un error para indicar que no se puede procesar la categoría

```java
public StringBuffer leeCategoria(JsonReader reader, String nombre) throws IOException {
    if (nombre.equals(MEDICINES_TAGNAME)) {
        return super.codigoComun(reader, nombre);
    }

    else {
        if (getSucesor() != null) {
            return super.leeCategoria(reader, nombre);
        } else {
            reader.skipValue();
            System.err.println(nombre + " no es una categoria processable.");
            return new StringBuffer("");
        }
    }
}
```

Por ultimo tendremos el método leeEntry, que creara un string a partir de lo que lea en la categoría seleccionada.

```java
public String leeEntry(JsonReader reader) throws IOException {
    String medicina = null;
    while (reader.hasNext()) {
        String nombre = reader.nextName();
        if (nombre.equals(NAME_FIELD_TAGNAME)) {
            medicina = reader.nextString();
        } else {
            reader.skipValue();
        }
    }
    return medicina;
}
```

En caso de que tenga subcategorías, primero la identifica y luego realiza la lectura correcta para formar el string adecuado.

```java
public String leeEntry(JsonReader reader) throws IOException {
    String nombrePhy = null;
    String imagen = null;
    while (reader.hasNext()) {
        String nombre = reader.nextName();
        if (nombre.equals(NAME_FIELD_TAGNAME)) {
            nombrePhy = reader.nextString();
        } else if (nombre.equals(IMAGE_FIELD_TAGNAME)) {
            imagen = reader.nextString();
        } else {
            reader.skipValue();
        }
    }
    return nombrePhy + FIELD_SEPARATOR + " " + imagen;
}
```

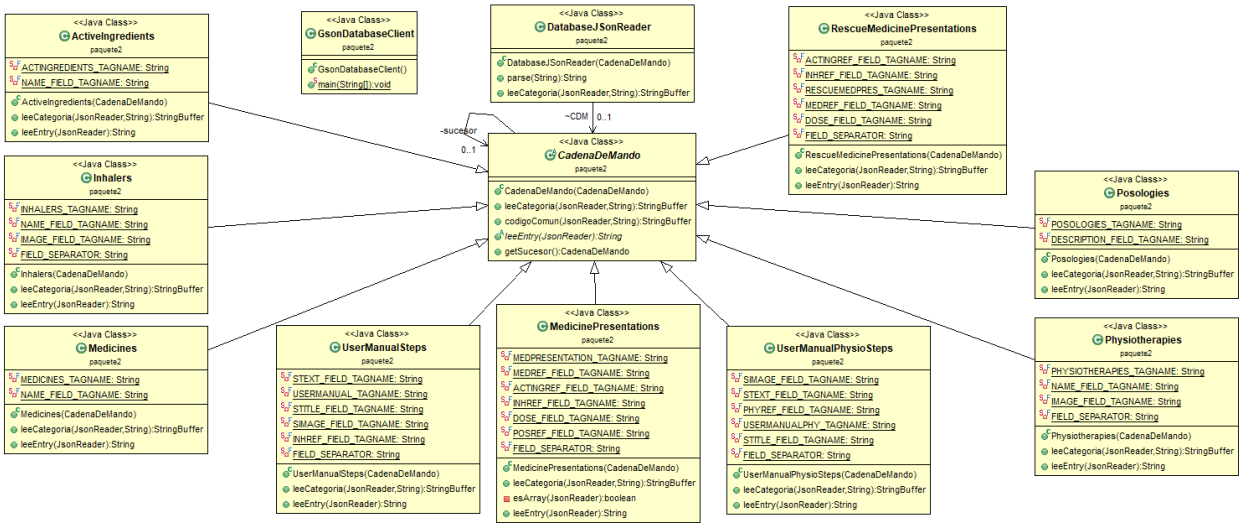
GsonDatabaseClient

Por ultimo, creamos todas las clases en GsonDatabaseClient con la estructura adecuada para que funcione el patrón cadena de mando.

```java
public class GsonDatabaseClient {

    public static void main(String[] args) {
        try {
            Medicines m = new Medicines(null);
            ActiveIngredients ai = new ActiveIngredients(m);
            Physiotherapies pt = new Physiotherapies(ai);
            Inhalers i = new Inhalers(pt);
            Posologies p = new Posologies(i);
            MedicinePresentations mp = new MedicinePresentations(p);
            RescueMedicinePresentations rmp = new RescueMedicinePresentations(mp);
            UserManualPhysioSteps umps = new UserManualPhysioSteps(rmp);
            UserManualSteps ums = new UserManualSteps(umps);
            DatabaseJSonReader dbjr = new DatabaseJSonReader(ums);
            try {
                System.out.println(dbjr.parse("./datos.json"));
            } finally {
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Diagrama de Clases



Use un diagrama temporal en sucio mientra hacia el trabajo, pero este el resultado final, hecho con ObjectAid UML Diagram a partir del proyecto en java. Pondré una foto mas grande y el archivo en formato ucls en el repositorio de github