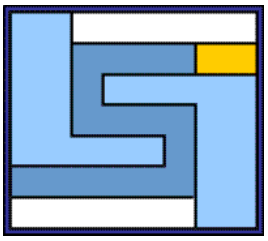



# Item 8 (A++) - Report

Diseño y Pruebas

Grado de Ingeniería del Software

Curso 3

Armando Garrido Castro
Jorge Puente Zaro
Manuel Enrique Pérez Carmona
Cesar García Pascual
Pablo Tabares García
Rafael Trujillo González

Fecha: 13 de abril de 2018

## 1. Introducción

Cada vez que avanzan los entregables la cantidad de trabajo es mayor, sobre todo cuando hay que empezar trabajos desde cero. Esto implica realizar tareas repetitivas como crear los convertidores, dominio, etc. Para ello proponemos como A++ crear una factoría que tome un archivo .json y genere todas las clases necesarias. De esta forma solo habría que implementar las reglas de negocio.

## 2. Archivo JSON

En primer lugar, necesitamos añadir al pom.xml la dependencia necesaria para poder trabajar con JSON. La dependencia es la siguiente:

```
<!-- GSON For Factory -->
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.2</version>
</dependency>
```

A continuación, mostramos el archivo JSON, adjunto en la carpeta, el cual hay que rellenar para que la factoría lo lea y genere los archivos necesarios.


*Ilustración 1: Fragmento JSON.*

Como podemos ver en la ilustración anterior mostramos un fragmento del JSON en él podemos ver los siguientes campos:

- name: Nombre de la clase.
- spanishName: Nombre en español, necesario para la internacionalización de la pagina.
- abs: Indica si es una clase abstracta.
- ext: Indica si extiende de alguna clase.
- attributes: Contiene un array en el que se le indica el nombre del atributo el tipo y las anotaciones sin el @.
- relationships: De forma parecida a los atributos, se indica el nombre de la clase con la que está asociada, el tipo, si es una colección y las anotaciones pertinentes de spring.
- queries: En este campo se indica el nombre de la query, la entidad a la que pertenece esta query, los valores de retorno de la query y por último la query que se desea ejecutar con un comentario para saber cual es la función de la query.
- serviceMethods: En este campo se indica los métodos CRUD que tendrá el servicio.
- list, display, edit: un booleano que genera dichas vistas.

- auth: recibiría el rol del actor que debe crear la entidad.

## 3. Factoría

La factoria, Group16Factory.java adjunta en la carpeta , es la clase java que se encarga de generar los archivos del proyecto a partir del JSON y los templates.
En este archivo, que se encuentra en la ruta java>utiles>Group16Factory.java, encontramos un primer método **main,** en él se toma el JSON se trata y posteriormente se llama a los métodos encargados de crear el dominio, repositorios, servicios y controladores. Acto seguido, se hace una llamada al método **createAuxiliaryStructure(entities)** la cual recibe como parámetro un lista del tipo "FactoryObject" que contiene las entidades que alberga el JSON.

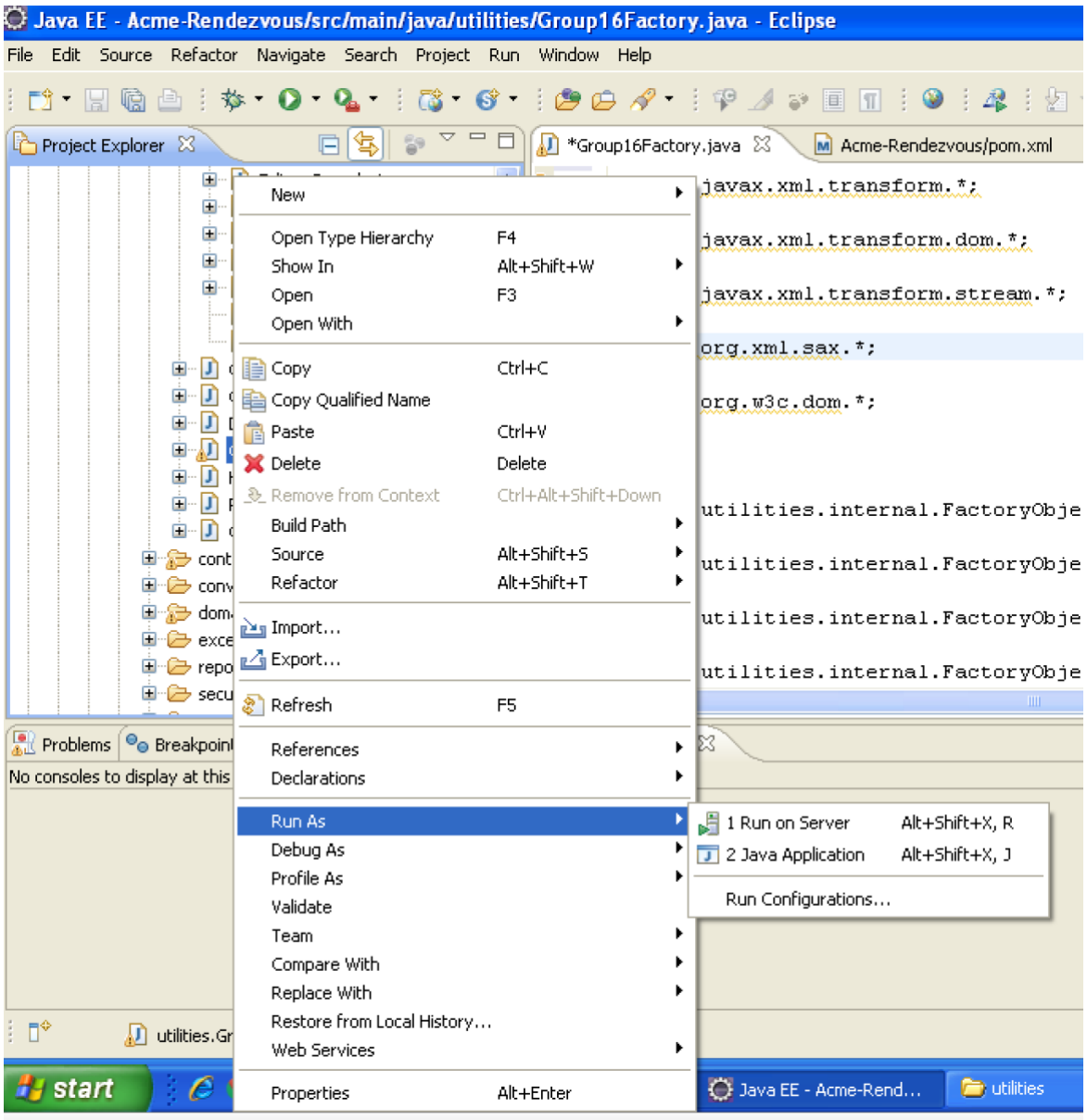
La función **createAuxiliaryStructure(entities)** realiza un bucle en la que comienza renombrando los convertidores para cada entidad. Tras esto, Pasa a generar los archivos de la carpeta Webapp, con los métodos **createDisplayView, createEditView, createListView, createMessages** se crean las vistas y los mensajes de las vistas en sus

correspondientes carpetas.

Por otro lado, en la carpeta java>utiles>internal encontramos el archivo FactoryObject.Java. En él se crean los atributos que necesita la factoria para ejecutarse.

Al finalizar el script encontramos la definición de todos los métodos necesarios para crear las clases y la definición de constantes con el fin de que el código quede escrito de la forma mas simple posible.

Para ejecutar la factoría, seguimos la metodología utilizada en la ejecución del PopulateDatabase.java o el QueryDatabase.java



En la imagen anterior observamos la forma de ejecutar la factoría. Una vez ejecutada es necesario refrescar el proyecto para que aparezcan los archivos generados. Es posible que en ellos aparezcan fallos ya que es necesario realizar los imports necesarios.

## 4. Templates

En la carpeta Template, se encuentra en src>main>resources, en ella se albergan las plantillas necesarias para crear las clases.

```
package converters;

import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import domain.%ent%;

@Component
@Transactional
public class %ent%ToStringConverter implements Converter<%ent%, String> {

	@Override
	public String convert(final %ent% %ea%) {
		String result;

		if (%ea% == null) {
			result = null;
		} else {
			result = String.valueOf(%ea%.getId());
		}

		return result;
	}

}
```

*Ilustración 2: Ejemplo template.*

En la imagen anterior mostramos la plantilla de uno de los convertidores. En ella podemos observar que aparece la estructura básica, tan solo hay que añadir los nombres de las clases. Que se pasan como parámetro en la factoría y estos parámetros se añaden en las cadenas de texto %ent% y %ea% respectivamente.

## 5. Conclusiones

Entre las desventajas de utilizar la factoría se encuentra el tiempo que se tarda en implementarla, sin embargo, en caso de tener que realizar varios proyectos, como es la asignatura Diseño y Pruebas, a la larga este esfuerzo se ve recompensado. También, Entre los problemas encontrados es que para que la factoría funcione de manera

correcta hay que aplicar una programación muy minuciosa ya que hay que indicar el nombre que recibirán las clases, atributos que con un fallo a la hora e indicar sus nombres puede hacer que el proyecto no funcione de manera correcta. Por otro lado, La factoría contiene algunos bugs ya que por falta de tiempo no han sido posible solucionar. En siguientes entregables se añadirá una versión mejorada de la misma

Las ventajas de tener una factoría es sobre todo el ahorro de tiempo en tareas repetitivas como por ejemplo la creación de convertidores, creación de las clases (que la mayoría de las veces todas siguen una misma estructura), o añadir rutas a los ficheros de configuración. De esta forma conseguimos un ahorro del tiempo que puede ser aplicado en testear la aplicación mas en profundidad lo hace que los fallos se reduzcan. También la facilidad de utilizarlo ya que solo es necesario ejecutar el archivo Group16Factory.java