



POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL



04/03/2015 •

Back-end con Tecnologías de Código Abierto

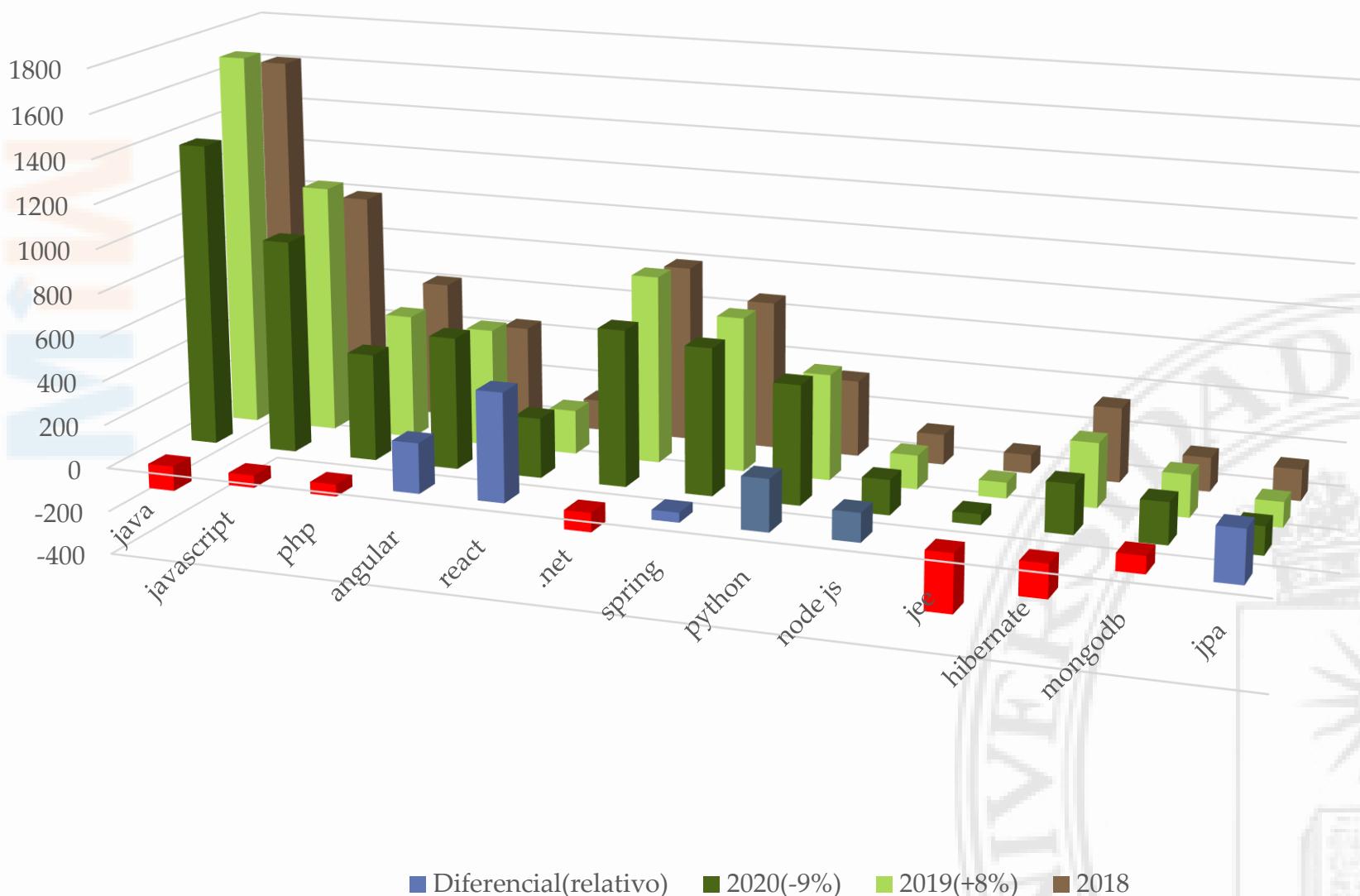
Spring

<https://github.com/miw-upm/betca-spring> (Teoría)

[https://github.com/miw-upm/betca\(tpv-angular](https://github.com/miw-upm/betca(tpv-angular)

[https://github.com/miw-upm/betca\(tpv-spring](https://github.com/miw-upm/betca(tpv-spring)

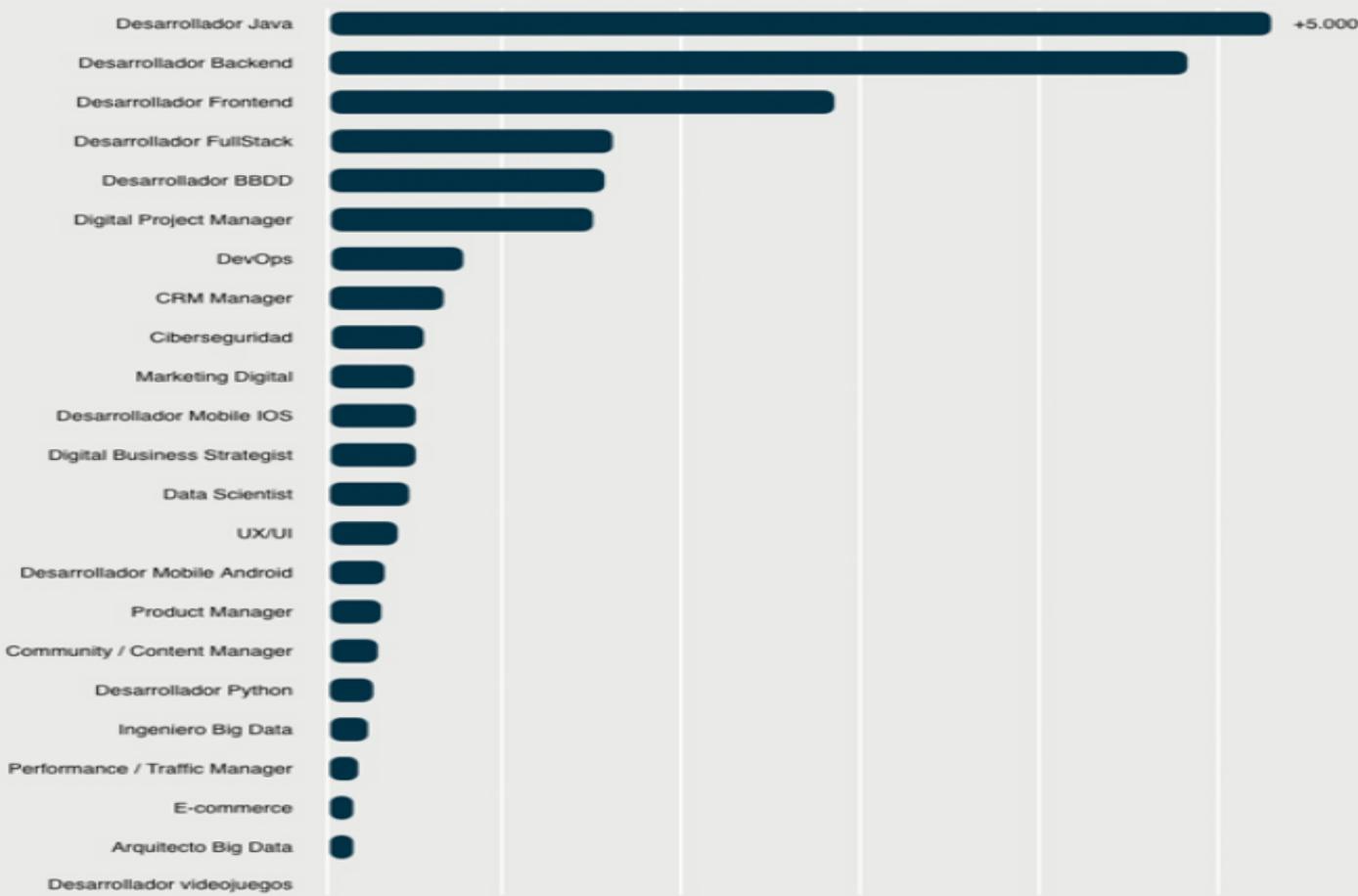
InfoJobs. Ofertas a 03/02/2020



InfoJobs. septiembre/2018

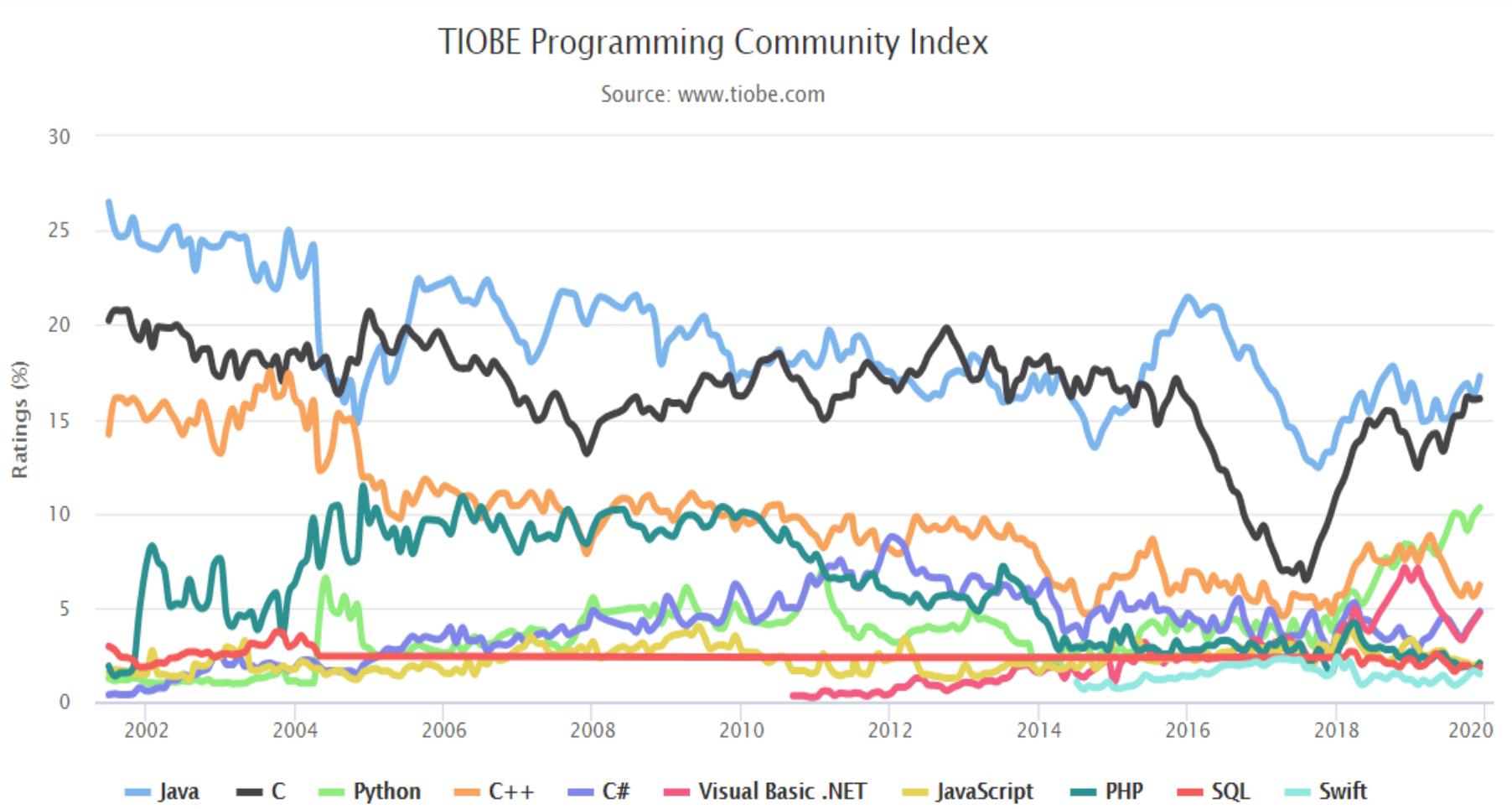
Los puestos más demandados en España

Estos son los perfiles digitales con mayor demanda de empleo en los últimos 3 meses, según datos obtenidos de Infojobs y de otras fuentes. Pulta en cada una de las profesiones para ver video.



Tecnologías de servidor

- En el servidor, se utilizan tecnologías, propietarias o abiertas, para el desarrollo de aplicaciones web, los estándares no son necesarios en el servidor



Spring

- Spring es un framework ligero de código abierto, creado por Rod Jhonson, para facilitar el desarrollo de Aplicaciones Empresariales modernas (sobre *Java* o *kotlin*):
<https://spring.io/>
- Está organizado por módulos independientes, y puede integrarse con otros frameworks.
 - **Spring Boot**
 - Construye y arranca rápidamente la aplicación con una configuración inicial mínima de Spring.
 - **Spring Framework**
 - Proporciona un modelo completo de programación y configuración para aplicaciones empresariales modernas.
- Alternativa: JEE (*Java Enterprise Edition*)
 - Es un conjunto de especificaciones desarrollado e implementado por una coalición de empresas lideradas por *Oracle*, *IBM*, *Red Hat*, etc..

Spring Framework 5

■ Spring MVC

- Web síncrona, bloqueante.
- API servlet. Por cada petición un hilo.
- Contenedor de servlets:
 - Tomcat, Jetty...
- Repositorios:
 - JDBC, JPA, NoSQL (MongoDB)

■ Spring WebFlux

- Web asíncrona, no bloqueante.
- Utiliza el potencial de los procesadores *multi-core*. Mantiene un numero masivo de conexiones simultáneas con pocos hilos.
- Basado en *Reactive Streams* (<http://www.reactive-streams.org>)
 - Publisher: *onSubscribe*, *onNext*, *onError* & *onComplete*
 - *Mono* & *Flux*
- Contenedores de servlets (3.1+):
 - Netty
- Repositorios:
 - MongoDB, Cassandra, Redis, Couchbase

Spring



SPRING BOOT



SPRING FRAMEWORK



SPRING DATA



SPRING SECURITY



SPRING CLOUD



SPRING CLOUD DATA FLOW



SPRING BATCH



SPRING REST DOCS



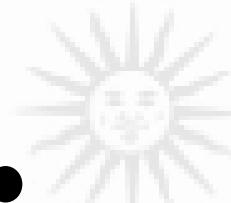
SPRING SOCIAL



SPRING LDAP



SPRING HATEOAS



Spring: start

The screenshot shows the Spring Initializr web application at start.spring.io. The interface is a form for generating a Spring Boot project. It includes sections for Project (Maven Project selected), Language (Java selected), Spring Boot version (2.2.2 selected), Project Metadata (Group: com.example, Artifact: demo), and Dependencies (Search bar: Web, Security, JPA, Actuator, Devtools...). Two dependencies are selected: Spring Reactive Web and Spring Security, each with a checked checkbox.

Spring Initializr
Bootstrap your application

Project Maven Project Gradle Project

Language Java Kotlin Groovy

Spring Boot 2.2.3 (SNAPSHOT) 2.2.2 2.1.12 (SNAPSHOT) 2.1.11

Project Metadata

Group: com.example

Artifact: demo

Options

Dependencies

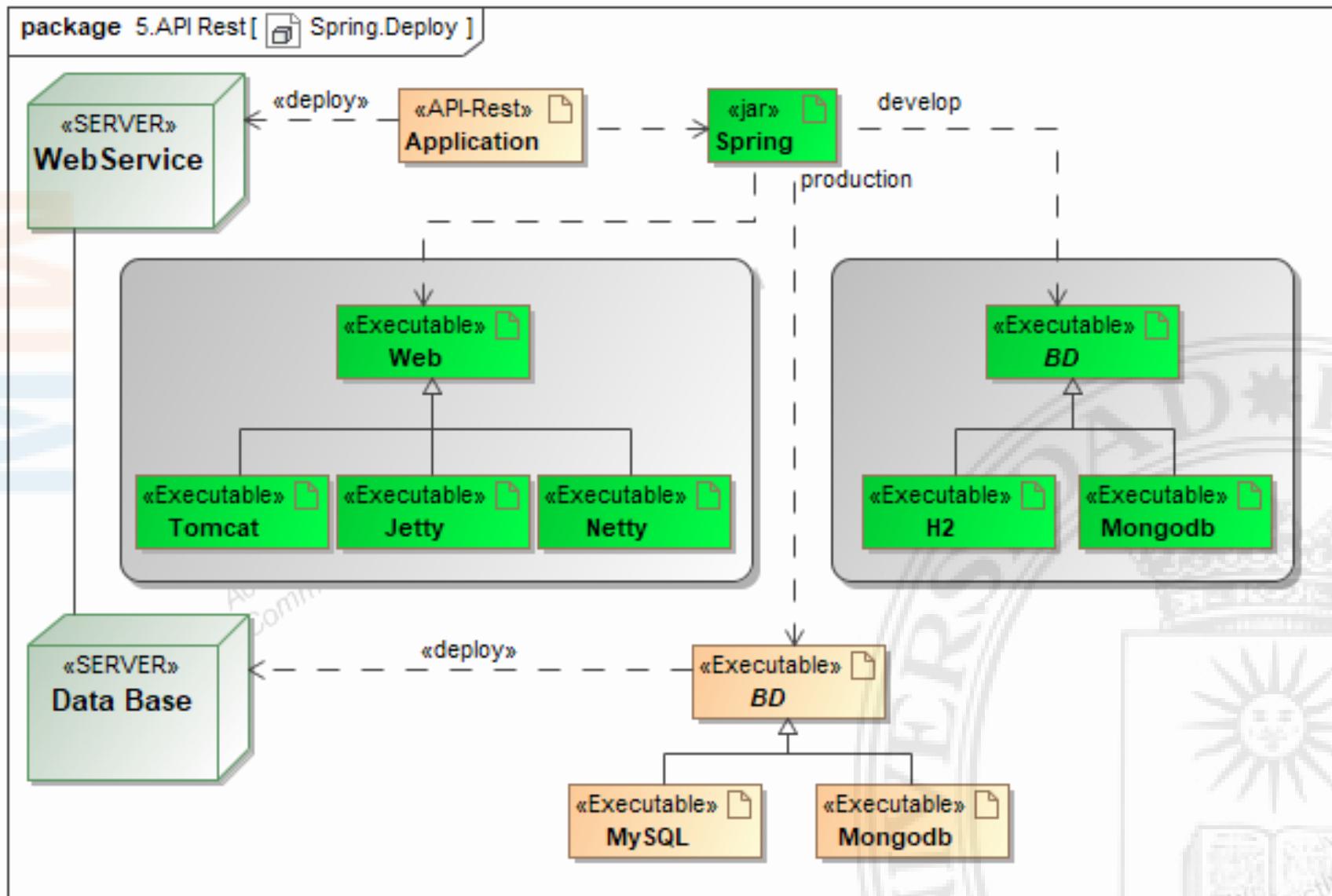
5 selected

Search dependencies to add: Web, Security, JPA, Actuator, Devtools...

Selected dependencies:

- Spring Reactive Web**
Build reactive web applications with Spring WebFlux and Netty.
- Spring Security**
Highly customizable authentication and access-control framework for Spring applications.

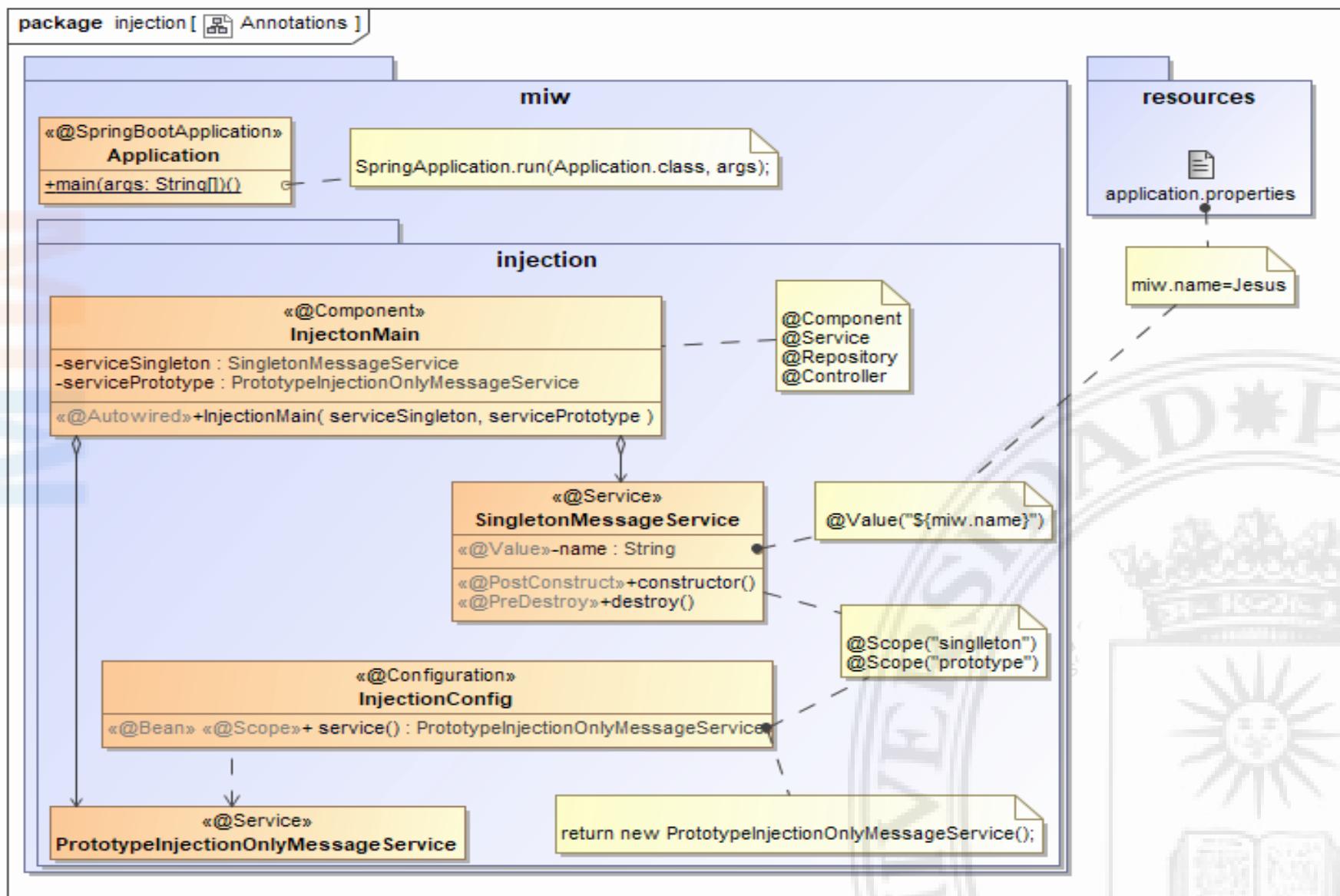
Arquitectura Web



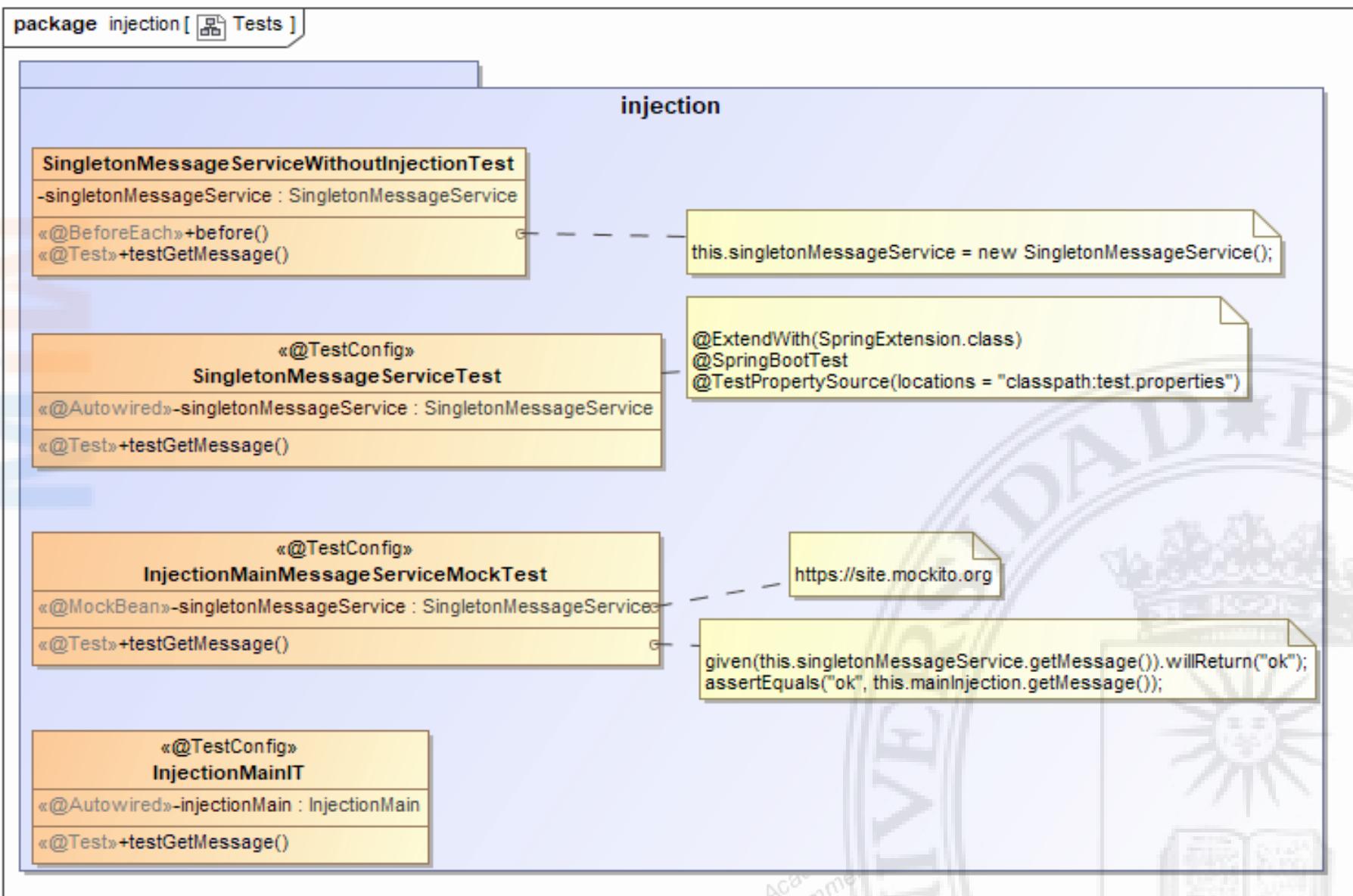
Inyección de dependencias

- Spring framework implementa la DI (*Dependency Injection*), se hace responsable de crear objetos, configurarlos y ensamblarlos. Es una gran *Factoría*
- A los objetos manejados por la Factoría se les denomina *beans*
- El interface *ApplicationContext*, es el responsable de construir la Factoría a partir de metadatos, mediante:
 - Anotaciones (tendencia actual)
 - XML(<bean id="miBean" class="package.MiBean">)
 - Código de Java
- Esto nos lleva a un cambio importante en la manera de desarrollar nuestro sistema software

Inyección de dependencias



Test



✍ Empezando en Spring Boot: pom.xml

m betca-spring x

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://maven.apache.org/POM/4.0.0"
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4      <modelVersion>4.0.0</modelVersion>
5
6      <parent>
7          <groupId>org.springframework.boot</groupId>
8          <artifactId>spring-boot-starter-parent</artifactId>
9          <version>2.1.11.RELEASE</version>
10         <relativePath/> 
11     </parent>
12
13     <groupId>es.upm.miw</groupId>
14     <artifactId>betca-spring</artifactId>
15     <version>1.1.0-SNAPSHOT</version>
16     <packaging>jar</packaging>
17
18     <name>${project.groupId}.${project.artifactId}</name>
19     <description>Project for teaching support to the BETCA-Spring (MIW-UPM)</description>
20     <url>http://github.com/miw-upm/betca-spring</url>
21
22     <licenses>
23         <license>
24             <name>MIT License</name>
25             <url>http://www.opensource.org/licenses/mit-license.php</url>
26         </license>
27     </licenses>
```

Spring version

✍ Empezando en Spring Boot: pom.xml

The screenshot shows a code editor window with the title bar "m betca-spring". The editor displays a portion of a Maven configuration file (pom.xml) with syntax highlighting for XML tags and Java code. The code includes properties for encoding, Java version, and various Maven plugin versions, along with dependency sections for Spring Boot and Spring Security.

```
<properties>
    <!-- Encode -->
    <encoding>UTF-8</encoding>
    <project.build.sourceEncoding>${encoding}</project.build.sourceEncoding>
    <project.reporting.outputEncoding>${encoding}</project.reporting.outputEncoding>
    <project.resources.sourceEncoding>${encoding}</project.resources.sourceEncoding>
    <!-- Java -->
    <jdk.version>1.8</jdk.version>

    <deploy>${project.artifactId}-${project.version}</deploy>

    <!-- Maven -->
    <maven.compiler.version>3.5</maven.compiler.version>
    <!-- Maven. Test de Integración -->
    <maven-surefire-plugin.version>2.21.0</maven-surefire-plugin.version>
    <maven-failsafe-plugin.version>2.21.0</maven-failsafe-plugin.version>
    <!-- JUnit -->
    <junit.platform.version>1.2.0</junit.platform.version>
    <!-- various -->
    <iTextpdf.version>7.0.2</iTextpdf.version>
    <auth0.version>3.4.1</auth0.version>
    <swagger.version>2.9.2</swagger.version>
    <lombok.version>1.18.8</lombok.version>

</properties>

<dependencies>
    <!-- Spring Boot ===== -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
```

✍ Empezando en Spring: código fuente

1. Clonar el repositorio
 - >`git clone https://github.com/miw-upm/betca-spring`
2. Importarlo con *IntelliJ*
3. Repasar código fuente, paquetes: *miw*, *miw.injection*
4. Crear un servicio con un método que devuelve un *String* con el día de la semana, crear un test para probarlo.
 - `LocalDate.now().getDayOfWeek().toString()`
5. Crear un componente que utiliza el servicio anterior y tiene un método que indica si es fin de semana (*isWeekEnd*), crear *test unitario* y de *integración*.
6. Añadir la propiedad “*miw.author=****” con el nombre del autor en el fichero *application.properties* y añadir al servicio un segundo método que devuelve un *String* con el nombre del autor, ampliar test de prueba.

Persistencia. Introducción

- Consiste en guardar los datos de forma permanente y ordenada
- Tipos de bases de datos
 - Bases de datos relacionales (*MySQL...*)
 - Bases de datos NoSQL (*Mongodb...*)
- Proyecto: *Spring Framework*
 - Acceso a datos: JDBC, ORM, JPA (Java EE), Transacciones...
- Proyecto: *Spring Data*
 - Proporciona un modelo de ayuda y facilita el uso de tecnologías de acceso a datos, bases de datos relacionales y no relacionales y servicios basados en la nube
 - Se trata de un proyecto global que contiene muchos sub proyectos que son específicos de una base de datos concreta
 - Ofrece la capa DAO implementada
- Documentación de referencia
 - <http://projects.spring.io/spring-data>
 - <https://projects.spring.io/spring-data-jpa/>
 - <https://projects.spring.io/spring-data-mongodb/>

DAO (Data Access Object)

- Data Transfer Object (DTO), también llamado *Entity* (BD relacionales) o *Document* (NoSQL). Son los objetos utilizados para la comunicación con la capa de *persistencia*.
- DAOs (*Repository*). Son las clases que se encargan de convertir los datos almacenados de forma persistente en objetos Java o viceversa.
- Propósito
 - Abstraer y encapsular el acceso a la capa de persistencia, permitiendo desacoplar la capa de negocio con la capa de persistencia.
 - Adapta el modelo de persistencia con el modelo de objetos.
 - Permite el intercambio de implementaciones de persistencia sin afectar a la capa de negocio.

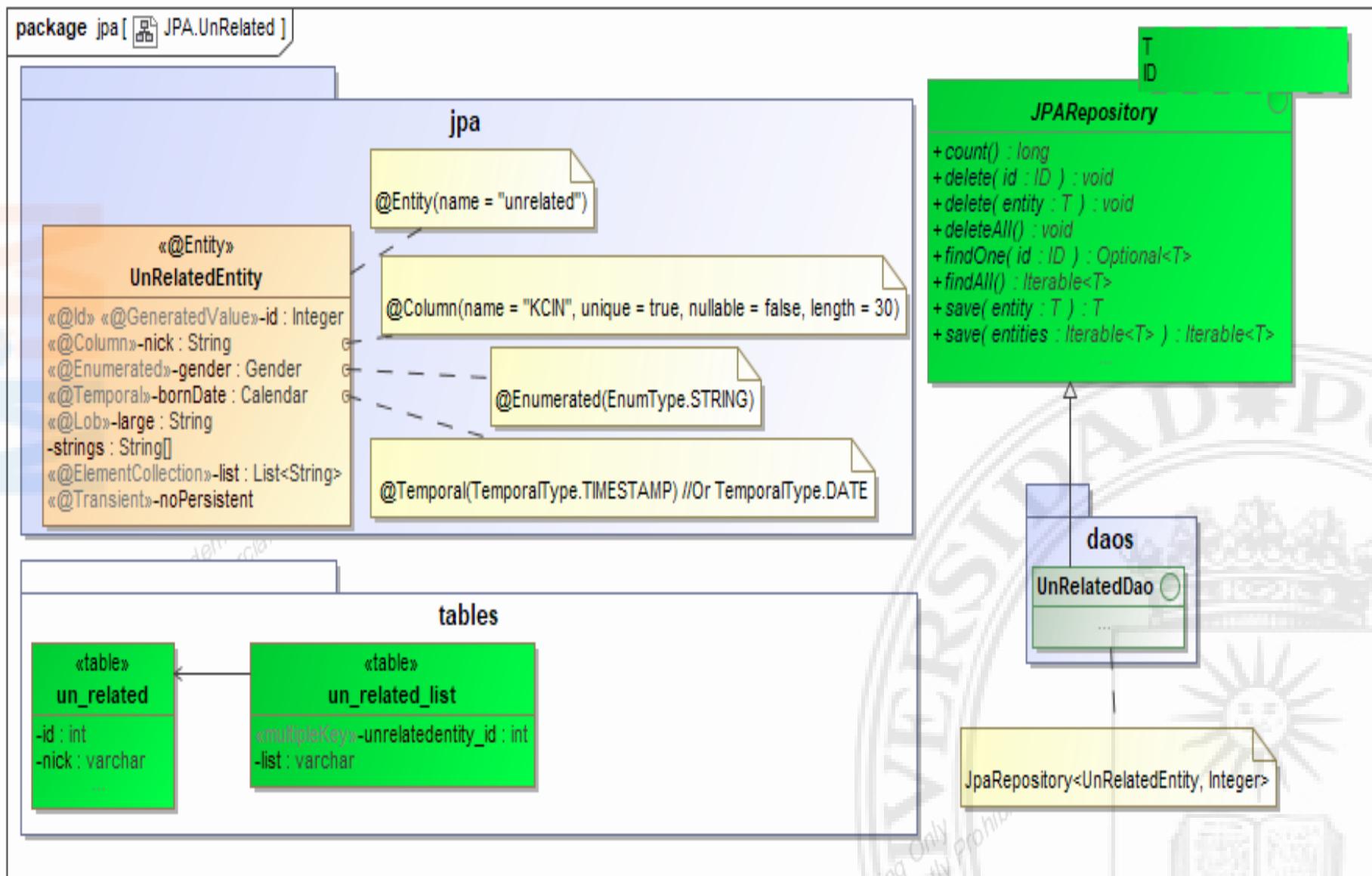
Java Persistencia API: JPA

- Spring
 - DOCS: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>
 - API: <https://docs.spring.io/spring-data/jpa/docs/current/api/>
- Modelo de persistencia para objetos java en bases de datos relacionales. JPA es un conjunto de interfaces
- Paquete javax.persistence. Todos los import:
 - **javax.persistence.***
- Existen varias implementaciones de JPA. Cada implementación amplían las prestaciones de JPA
 - Hibernate(JPA , propia Java, .NET)
 - EclipseLink (Eclipse)
 - Documentación: <http://eclipse.org/eclipselink/documentation/>
 - TopLink (Oracle)
 - OpenJPA (Apache)
 - ObjectDB
 - Documentación: <http://www.objectdb.com/java/jpa>

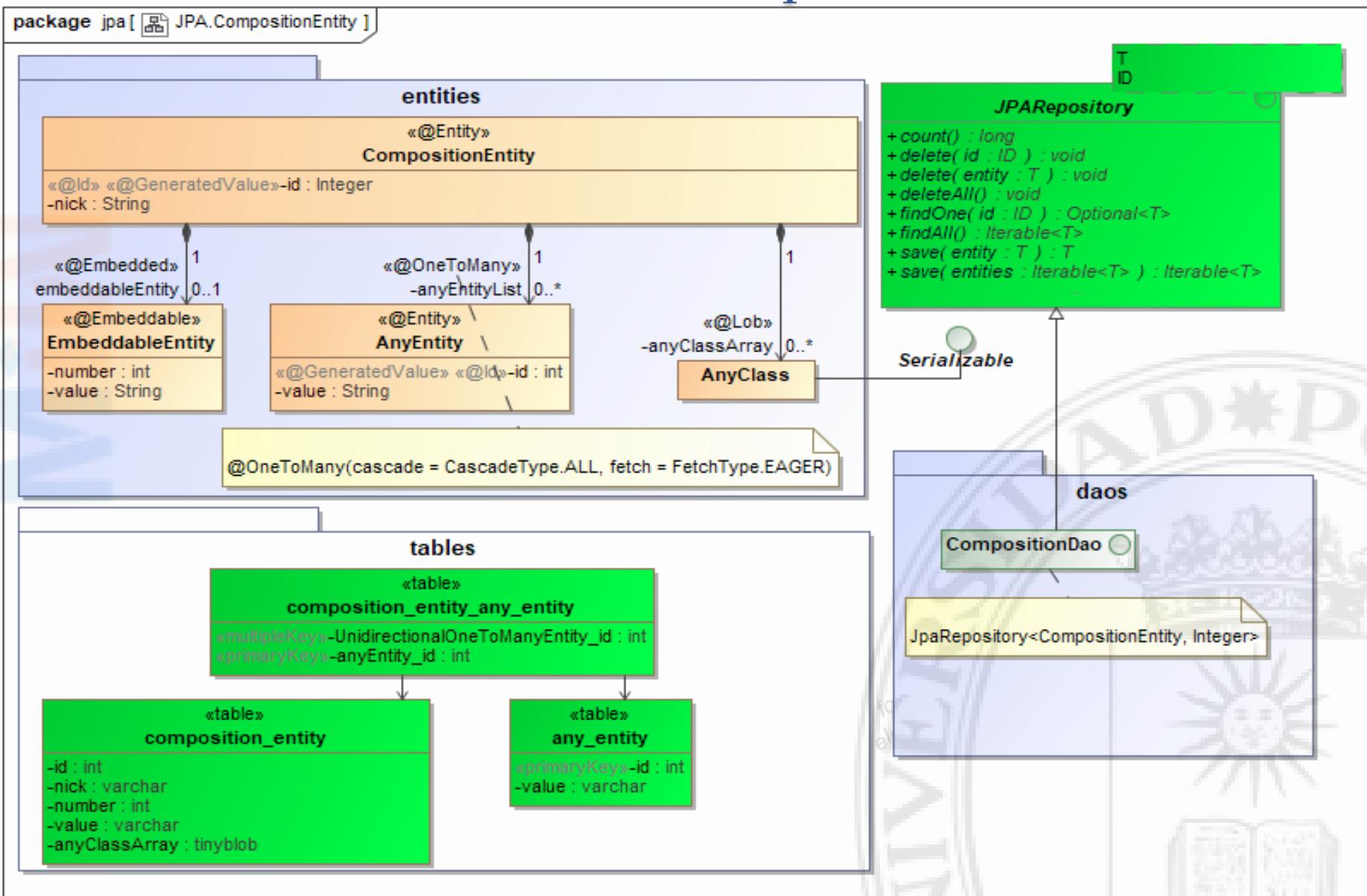
Entidades en JPA

- Una entidad es un objeto de persistencia. Normalmente, la clase entidad representa una tabla en el modelo relacional, y una instancia de la clase entidad representa una fila de la tabla
- Normalmente, los atributos de la clase entidad representan los campos de la tabla
- Requisitos de las clases entidades:
 - Debe tener la anotación: `@Entity (javax.persistence.Entity)`
 - La clase debe ser `public`
 - La clase debe tener un **constructor público sin parámetros** (puede tener más constructores)
 - Debe tener una clave primaria (`@Id`), (acceso mediante getters y setters, opcional)
 - La clase debería implementar los métodos:
 - `hashCode()`
 - `equals(Object other)`

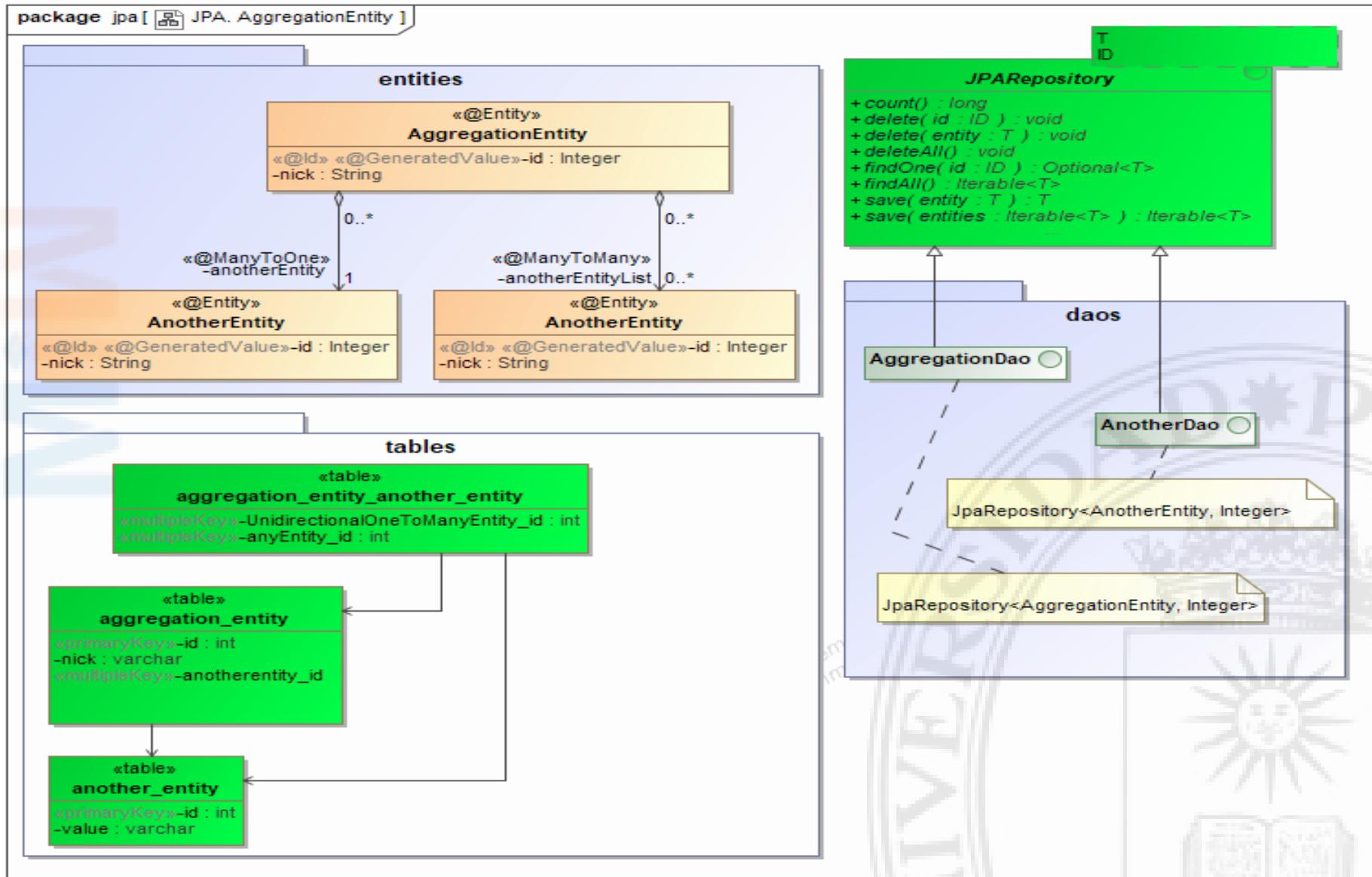
Anotaciones de Entidad



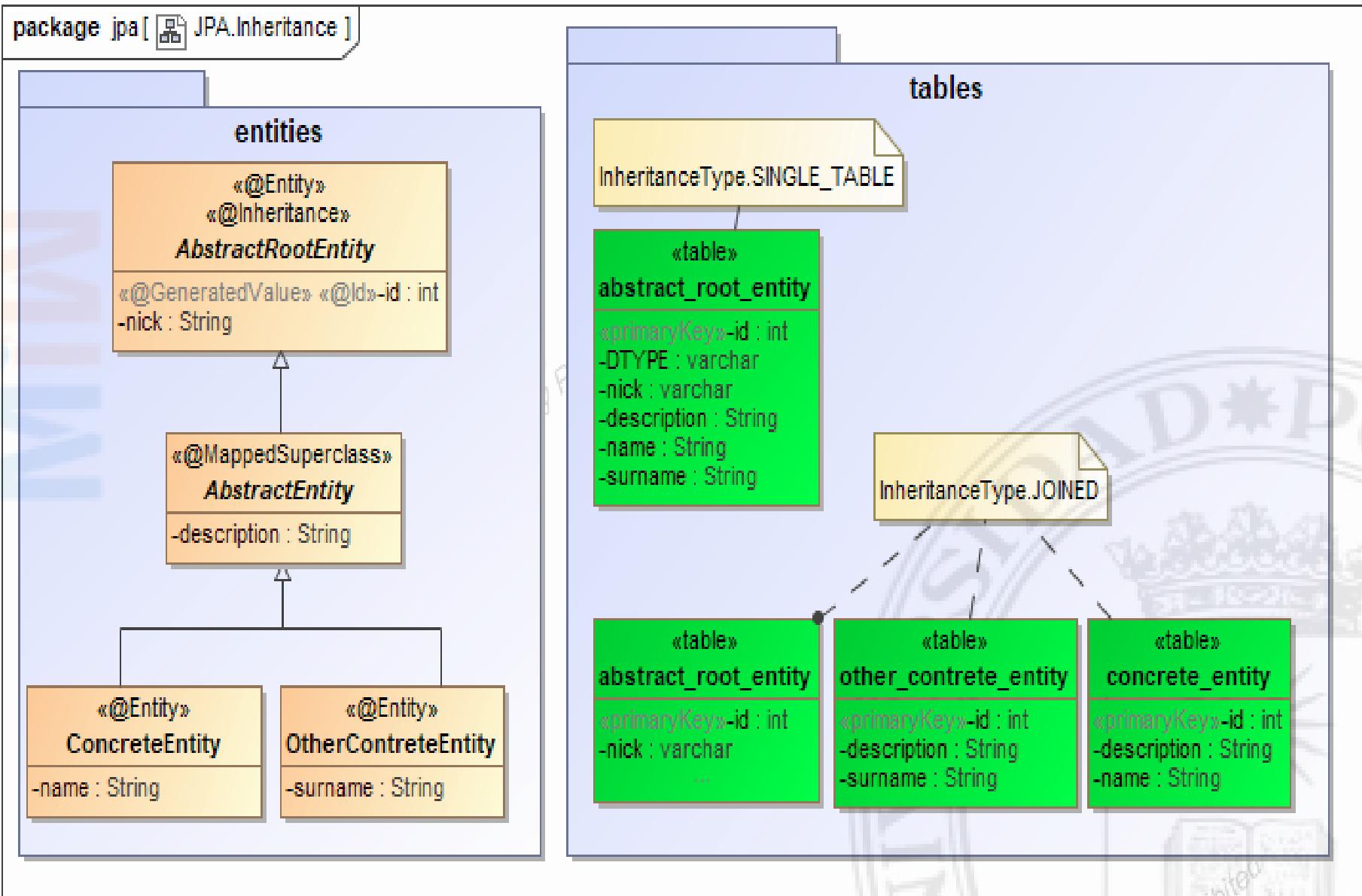
Relación de composición



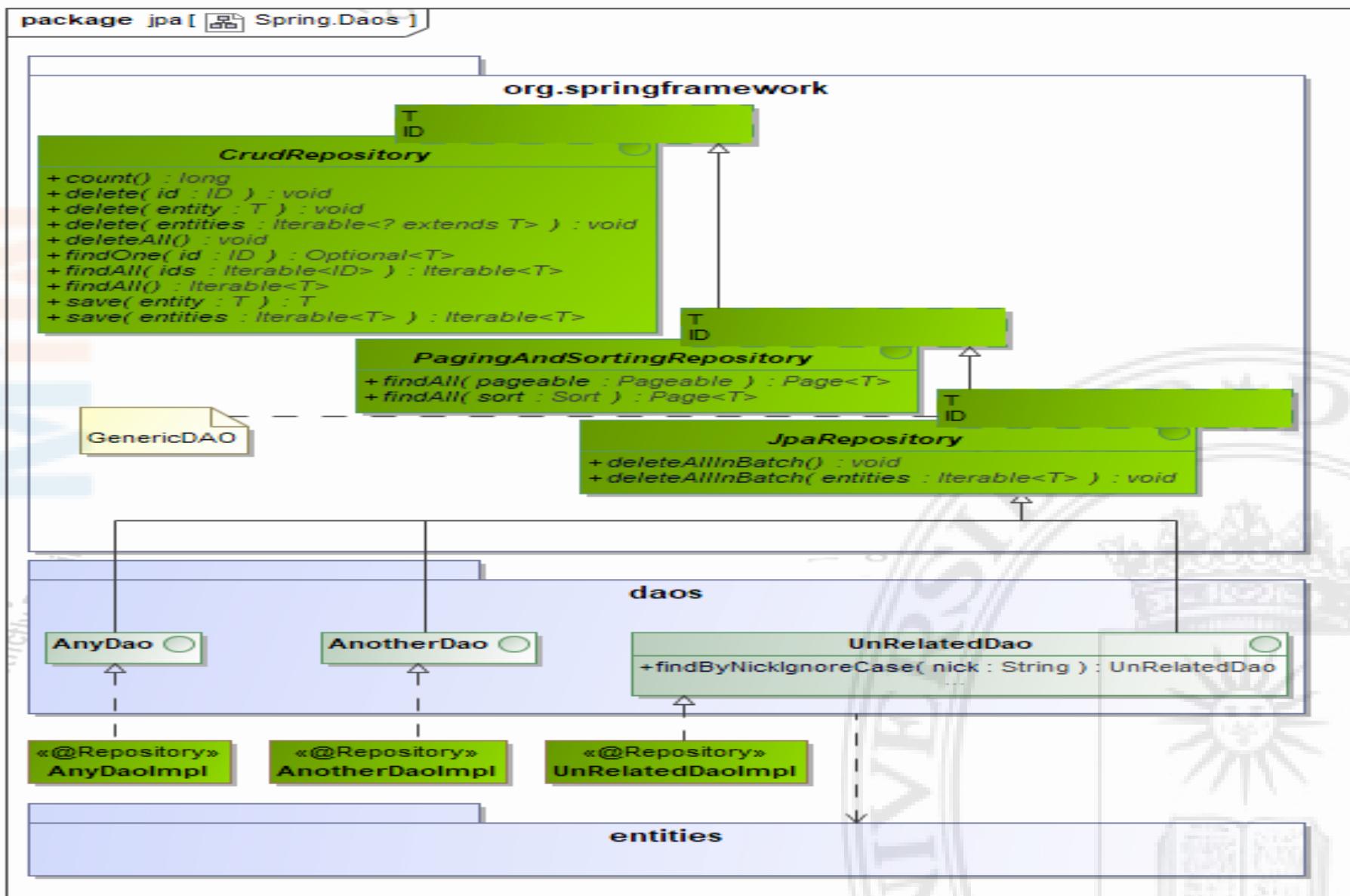
Relación de agregación & asociación



Inheritance



Spring Data. DAO



Configuración: JPA (H2 & MySQL)

```
<!-- Database JPA-->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
</dependency>
```

```
<!-- SQL embedded -->
```

```
<dependency>
```

```
    <groupId>com.h2database</groupId>
```

```
    <artifactId>h2</artifactId>
```

```
</dependency>
```

BD embebida

application.properties

```
application.properties
```

```
13 # DataSource #####
14 # H2 embedded -----
15 # spring.datasource.driver-class-name=org.h2.Driver
16 # spring.datasource.url=jdbc:h2:mem:testdb
17 # spring.datasource.username=sa
18 # spring.datasource.password=
19 spring.jpa.open-in-view=false
20 spring.jpa.properties.hibernate.hbm2ddl.auto=update
```

pom.xml

```
<!-- MySQL external -->
```

```
<dependency>
```

```
    <groupId>mysql</groupId>
```

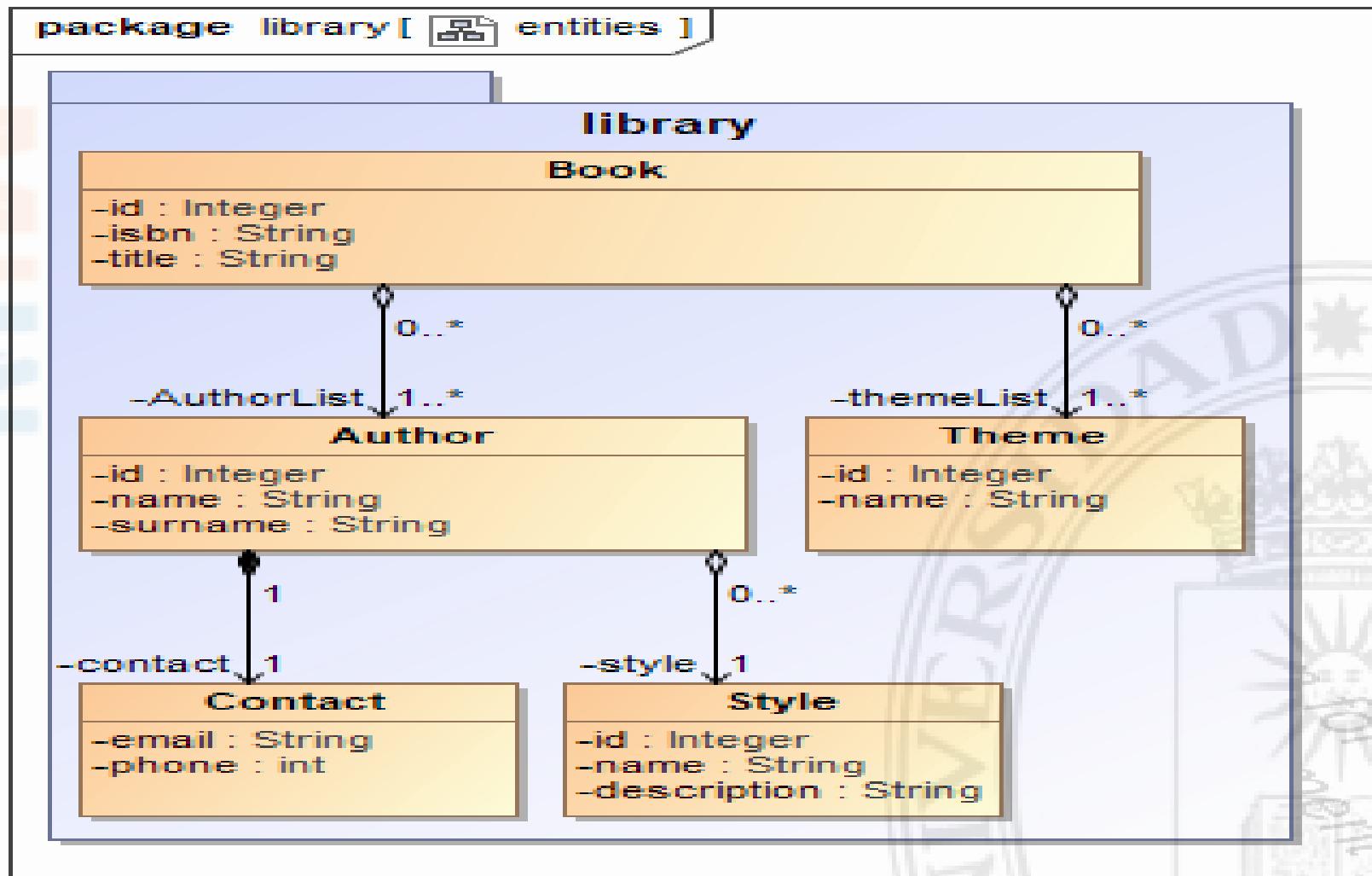
```
    <artifactId>mysql-connector-java</artifactId>
```

```
</dependency>
```

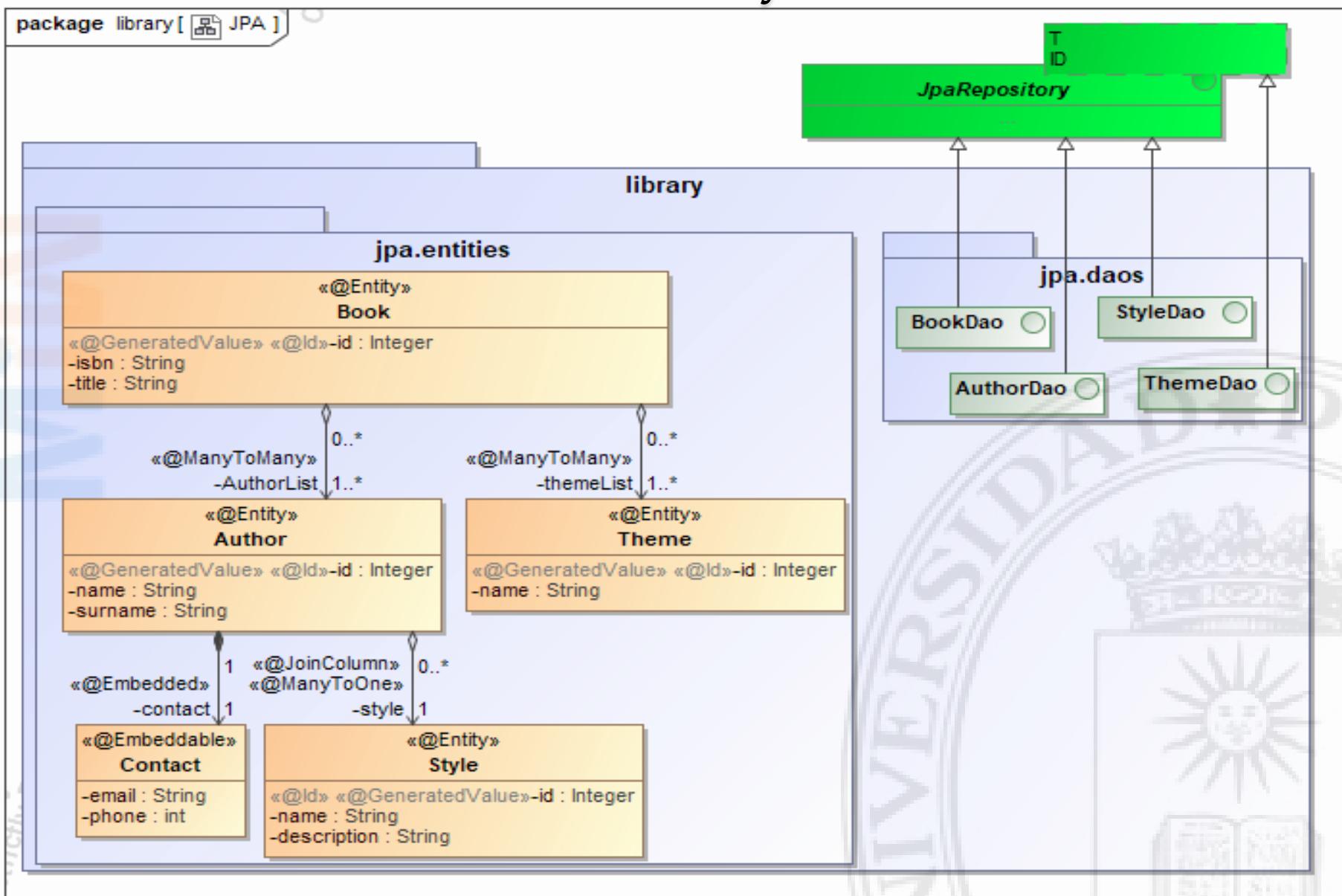
BD Externa

✍ Library

- ✍ Indicar las anotaciones (@*) necesarias en la clases y definir los DAOs
- ✍ Implementar modelos alternativos



✍ Library



Consultas: Nombre de Métodos

- **find, delete**
- **? First, First*, Distinct**
- **By**
 - Attribute 
 - Is, Equals, Between,
 - LessThan, LessThanEqual, GreaterThan, GreaterThanEqual
 - After, Before
 - IsNull, IsNotNull, NotNull
 - Like, NotLike
 - StartingWith, EndingWith, Containing
 - Not, In, NotIn, True, False
- **IgnoreCase**
- **? And, Or** 
- **? OrderBy**
 - AttributeDesc, AttributeAsc
- **Ejemplos**
 - *Entity findByAtr1(String atr1);*
 - *List<Entity> findFirst3ByAtr1StartingWith(String prefix);*
 - *List<Entity> findByAtr1OrAtr2OrderByAtr3Desc(String atr1, String atr2);*
 - *List<Entity> findByAtr1GreaterThanOrEqual(int value);*
 - *List<Entity> findByAtrIn(Collection<Integer> values);*
 - *int deleteByAtr1(String atr1);*
 - *int deleteByAtr1GreaterThanOrEqual(int value);*

Consultas: Nombre de Métodos

<i>Keyword</i>	<i>Sample</i>
And	<code>findByLastnameAndFirstname</code>
Or	<code>findByLastnameOrFirstname</code>
Is, Equals	<code>findByFirstname, findByFirstnamels, findByFirstnameEquals</code>
Between	<code>findByStartDateBetween</code>
LessThan	<code>findByAgeLessThan</code>
LessThanEqual	<code>findByAgeLessThanEqual</code>
GreaterThan	<code>findByAgeGreaterThan</code>
GreaterThanOrEqual	<code>findByAgeGreaterThanOrEqual</code>
After	<code>findByStartDateAfter</code>
Before	<code>findByStartDateBefore</code>
IsNull	<code>findByAgeIsNull</code>
IsNotNull, NotNull	<code>findByAgeNotNull, findByAgeIsNotNull</code>
Like	<code>findByFirstnameLike</code>
NotLike	<code>findByFirstnameNotLike</code>
StartingWith	<code>findByFirstnameStartingWith</code>
EndingWith	<code>findByFirstnameEndingWith</code>
Containing	<code>findByFirstnameContaining</code>
OrderBy	<code>findByAgeOrderByLastnameDesc</code>
Not	<code>findByLastnameNot</code>
In	<code>findByAgeIn(Collection<Age> ages)</code>
NotIn	<code>findByAgeNotIn(Collection<Age> ages)</code>
True	<code>findByActiveTrue()</code>
False	<code>findByActiveFalse()</code>
IgnoreCase	<code>findByFirstnameIgnoreCase</code>

Paginación

- Se pueden definir los métodos de los DAOs con un atributo del tipo interface *Pageable*, con ello se incorpora la paginación
- Con *PageRequest.of(int page, int size)*, creamos una petición
- Ejemplos
 - `List<UnRelatedEntity> findByIdGreaterThan(int id, Pageable pageable);`
 - `dao.findByIdGreaterThan(90, PageRequest.of(1, 5));`
 - `dao.count();`

JPQL (Java Persistence Query Language)

- Lenguaje similar a SQL
- JPQL. Maneja entidades (en lugar de tablas) y atributos de objetos (en lugar de campos)
 - Se obtiene una lista con todos los clientes
 - "SELECT c FROM Cliente c"
 - Se obtiene una lista con los DNI de todos los clientes
 - "SELECT c.dni FROM Cliente c"
 - Se obtiene una lista con todos los nombres diferentes entre los clientes
 - "SELECT DISTINCT c.nombre FROM Cliente c"
 - Lista con los nombres que empiecen por j
 - "SELECT c.nombre FROM Cliente c WHERE c.nombre LIKE 'j%'"
 - Lista con los nombres de clientes, cuyo id esté entre 3 y 4
 - "SELECT c.nombre FROM Cliente c WHERE c.id > 2 AND c.id < 5"
 - Lista con el nombre , cuyo DNI es...
 - "SELECT c.nombre FROM Cliente c WHERE c.dni = '42544422'"
 - Lista los nombres que no tienen pedidos
 - "SELECT c.nombre FROM Cliente c WHERE c.pedidos IS EMPTY"
- Ejemplo de consulta:
 - `@Query("select u.id from other_name_for_unrelatedentity u where u.id > ?1 and u.id < ?2")`
 - `List<Integer> findIdByIdBetween(int initial, int end);`
- Ejemplo de borrado:
 - `@Modifying @Query(value="delete from other_name_for_unrelatedentity u where u.nick = ?1")`
 - `int deleteByNickQuery(String nick);`

Nombre de Métodos

Sample	JPQL snippet
<code>findByLastnameAndFirstname</code>	... where x.lastname = ?1 and x.firstname = ?2
<code>findByLastnameOrFirstname</code>	... where x.lastname = ?1 or x.firstname = ?2
<code>findByFirstname, findByFirstnames, findByFirstnameEquals</code>	... where x.firstname = ?1
<code>findByStartDateBetween</code>	... where x.startDate between ?1 and ?2
<code>findByAgeLessThan</code>	... where x.age < ?1
<code>findByAgeLessThanEqual</code>	... where x.age <= ?1
<code>findByAgeGreaterThan</code>	... where x.age > ?1
<code>findByAgeGreaterThanOrEqual</code>	... where x.age >= ?1
<code>findByStartDateAfter</code>	... where x.startDate > ?1
<code>findByStartDateBefore</code>	... where x.startDate < ?1
<code>findByAgeIsNull</code>	... where x.age is null
<code>findByAgeNotNull, findByAgeIsNotNull</code>	... where x.age not null
<code>findByFirstnameLike</code>	... where x.firstname like ?1
<code>findByFirstnameNotLike</code>	... where x.firstname not like ?1
<code>findByFirstnameStartingWith</code>	... where x.firstname like ?1(parameter bound with appended %)
<code>findByFirstnameEndingWith</code>	... where x.firstname like ?1(parameter bound with prepended %)
<code>findByFirstnameContaining</code>	... where x.firstname like ?1(parameter bound wrapped in %)
<code>findByAgeOrderByLastnameDesc</code>	... where x.age = ?1 order by x.lastname desc
<code>findByLastnameNot</code>	... where x.lastname <> ?1
<code>findByAgeIn(Collection<Age> ages)</code>	... where x.age in ?1
<code>findByAgeNotIn(Collection<Age> ages)</code>	... where x.age not in ?1
<code>findByActiveTrue()</code>	... where x.active = true
<code>findByActiveFalse()</code>	... where x.active = false
<code>findByFirstnameIgnoreCase</code>	... where UPPER(x.firstname) = UPPER(?1)

Consultas: SQL

- Se utiliza SQL directamente sobre las tablas
- Ejemplo:
 - `@Query(value = "SQL... = ?1", nativeQuery = true)`
 - `UnRelatedEntity findByNick(String nick);`

✍ Consultas

- Definir varias consultas para la entidad: *UnRelatedEntity*
- Definir una consulta con paginación para la entidad: *UnRelatedEntity*
- Definir varias consultas mediante JPQL para la entidad *UnRelatedEntity*
- Definir una consulta en SQL para la entidad *UnRelatedEntity*
- Realizar los test correspondientes

✍ Consultas sobre Library-AuthorDao

■ Ejemplo

```
List<Author> findByStyle(Style style);  
  
@Query("select author.name from Author author where author.style.name = ?1")  
List<String> findNameByStyleName(String styleName);  
  
@Query("select distinct author.name from Book book join book.authorList author")  
List<String> findDistinctNameByAnyBook();  
  
@Query("select author.name from Book book join book.authorList author join book.themeList theme where  
theme.name = ?1")  
List<String> findNameByThemeName(String themeName);
```

■ ✍ Realizar las siguiente consultas en StyleDao

- *List<String> findDescriptionByFirstAuthorName(String name);*
- *List<String> findDescriptionByFirst3ByBook(Book book);*
- *List<Style> findByThemeName(String themeName);*

■ ✍ Realizar los test correspondientes

BD NoSQL: MongoDB

- MongoDB (*humongous: enorme*) es un sistema de base de datos NoSQL multiplataforma de licencia libre, orientado a documentos con un esquema libre
- Web: <https://www.mongodb.com/>
- Manual: <https://docs.mongodb.com/manual/>
- Tiene como modelo de almacenamiento de documento **BSON** (*Binary JavaScript Object Notation*), mas ligero y eficiente que JSON, pero el DTO es JSON
- Conceptos:
 - **Campo:** dato concreto
 - **Documento:** similar a registro de tabla, formado por un conjunto de campos
 - **Colección:** similar a tabla pero sin esquema fijo, formado por un conjunto de documentos
 - **Base de datos:** un conjunto de colecciones
- Características
 - Alta flexibilidad
 - Alta escalabilidad horizontal para trabajar con servidores distribuidos, replicados y balanceo de carga para mejorar el rendimiento

Instalación

- <https://docs.mongodb.com/getting-started/shell/installation/>
- <https://www.mongodb.com/download-center/enterprise>
- Instalación con *.zip
 - Versión *Community Server* >*All Version Binaries*
 - Descomprimir y crear las carpetas: **/data/db
 - Arrancar el servidor: **/bin>*mongod --dbpath* **/data/db
 - Arrancar la consola: **/bin>*mongo*
- Comandos:
 - <https://docs.mongodb.com/manual/reference/command/#database-commands>
 - Ayuda: >*help*
 - Versión: >*db.version()*
 - Estadísticas: >*db.stats()*
 - Muestra bases de datos: >*show dbs* >*show databases*
 - Utiliza una BD, si no existe la crea: >*use mybd*
 - Muestra las colecciones: >*show collections*
 - Borra la bd actual: >*db.dropDatabase()*
 - Borra la colección coll1: >*db.coll1.drop()*

BD NoSQL: MongoDB

■ Comandos

- Busca todo el contenido de una colección: `>db.coll1.find()`, `>db.coll1.find().pretty()`
- Muestra el número de documentos: `>db.coll1.count()`
- Inserta en una colección, la clave primaria se autogenera:
 - `>db.coll1.insert({ name:"jesus" })`
 - `→ { "_id" : ObjectId("5a86ad2e9d610a0fe33aa1ea"), "name" : "jesus" }`
- Inserta en una colección con clave primaria:
 - `>db.coll1.insert({ _id:1 , name:"jesus" })`
 - `→ { "_id" : 1, "name" : "jesus" }`
- Inserta en una colección :
 - `>db.coll1.insert({ _id: "2", name:"ny", central: { _id:1, location:"Madrid" } })`
 - `→ { "_id" : "2", "name" : "ny", "central" : { "_id" : 1, "location" : "Madrid" } }`
- Inserta en una colección :
 - `>db.coll1.insert({ _id: 3, name:"ny", central: [{ _id:2, location:"Madrid"},{ _id:3, location:"Sevilla"}] })`
 - `→ { "_id" : 3, "name" : "ny", "central" : [{ "_id" : 2, "location" : "Madrid" }, { "_id" : 3, "location" : "Sevilla" }] }`
- Inserta en una colección :
 - `>db.other.insert({ _id: 1, name:"nameInOther" })`
 - `>db.coll1.insert({ _id: 4, name:"nameInColl1", other: DBRef("other",1) })`
 - `→ { "_id" : 4, "other" : DBRef("other", 1) }`

■ Ejercicio. Instalación y manejo por consola

Configuración en Spring

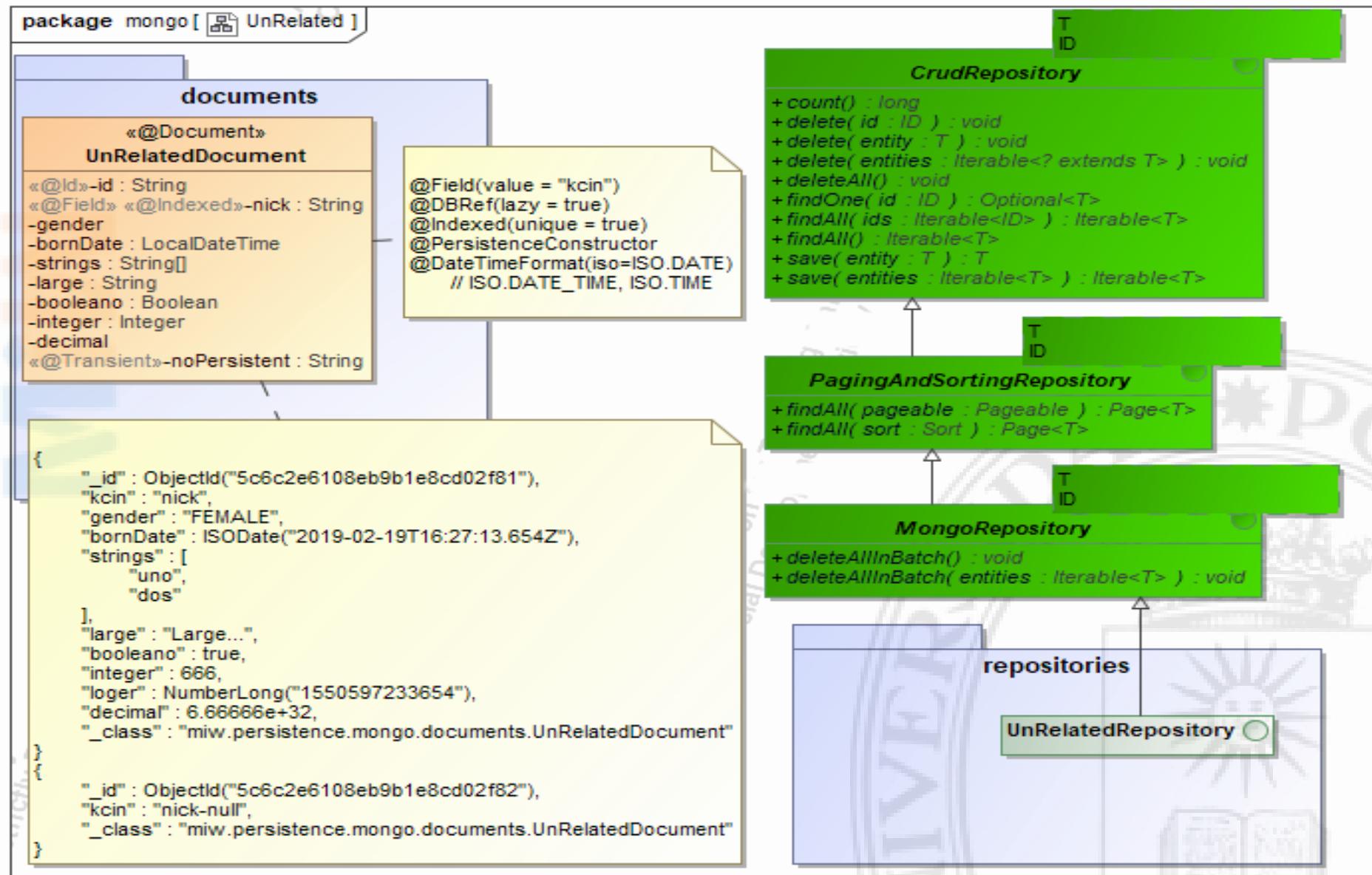
mbetca-spring

```
93      <!-- MongoDB -->
94      <dependency>
95          <groupId>org.springframework.boot</groupId>
96          <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
97      </dependency>
98      <!-- No reactive
99      <dependency>
100         <groupId>org.springframework.boot</groupId>
101         <artifactId>spring-boot-starter-data-mongodb</artifactId>
102     </dependency> -->
103     <!-- Mongodb embedded: Test -->
104     <dependency>
105         <groupId>de.flapdoodle.embed</groupId>
106         <artifactId>de.flapdoodle.embed.mongo</artifactId>
107         <scope>test</scope>
108     </dependency>
```

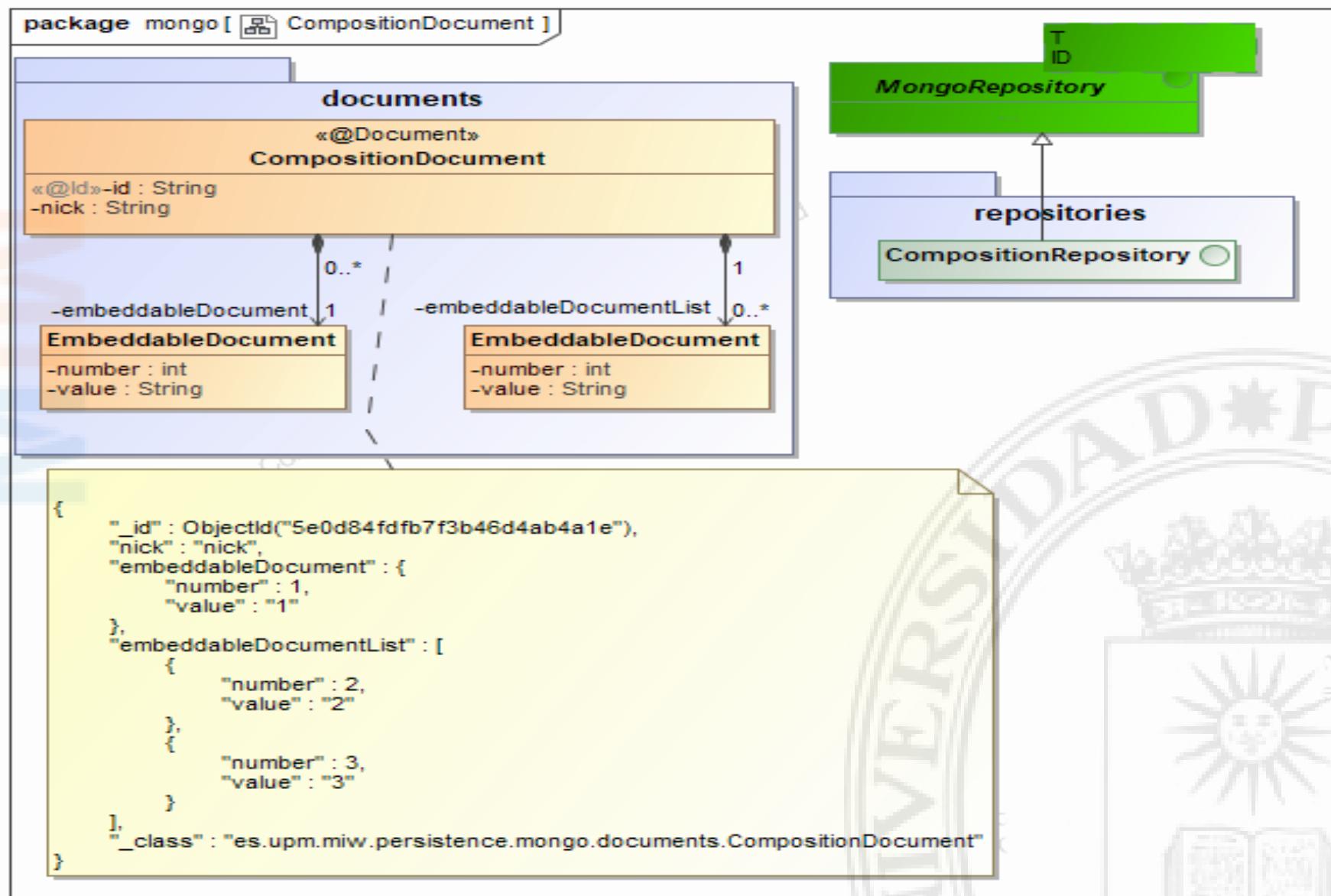
application.properties

```
21 # MONGODB -----
22 # spring.data.mongodb.database=test
23 # spring.data.mongodb.host=localhost
24 # spring.data.mongodb.port=27017
25 # spring.data.mongodb.username=
26 # spring.data.mongodb.password=
```

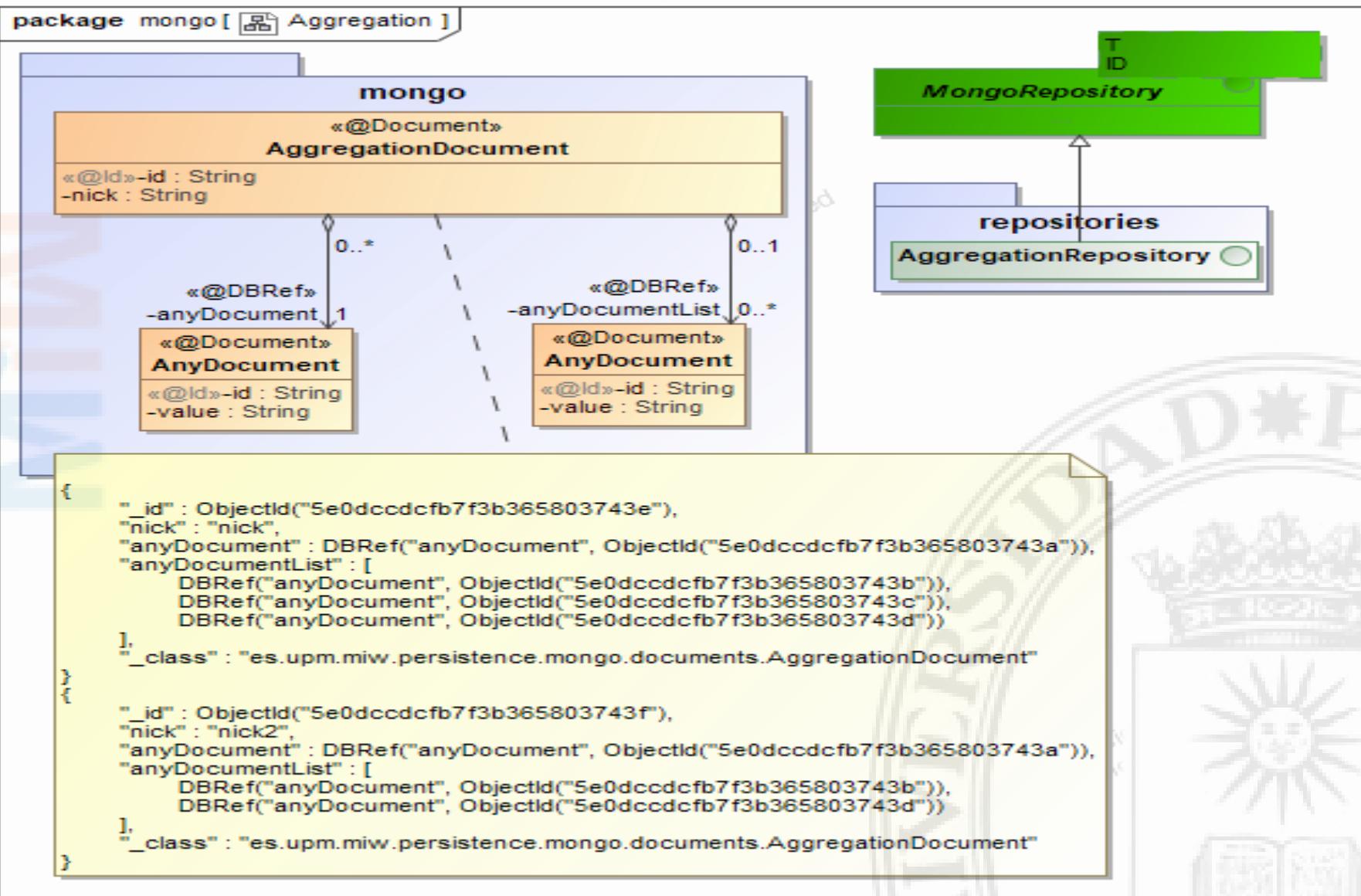
Anotaciones de documento



Relación de composición

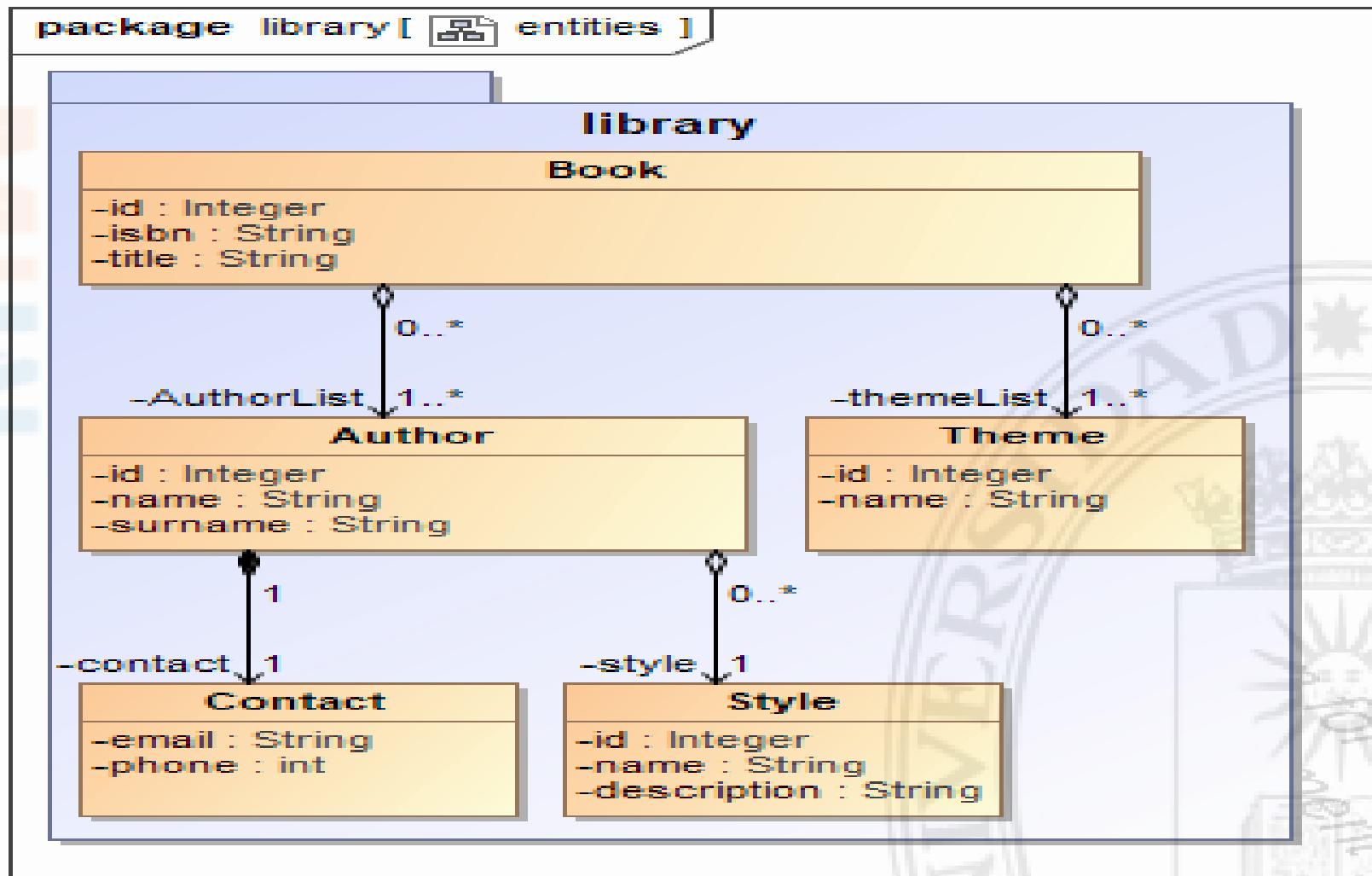


Relación de agregación & asociación

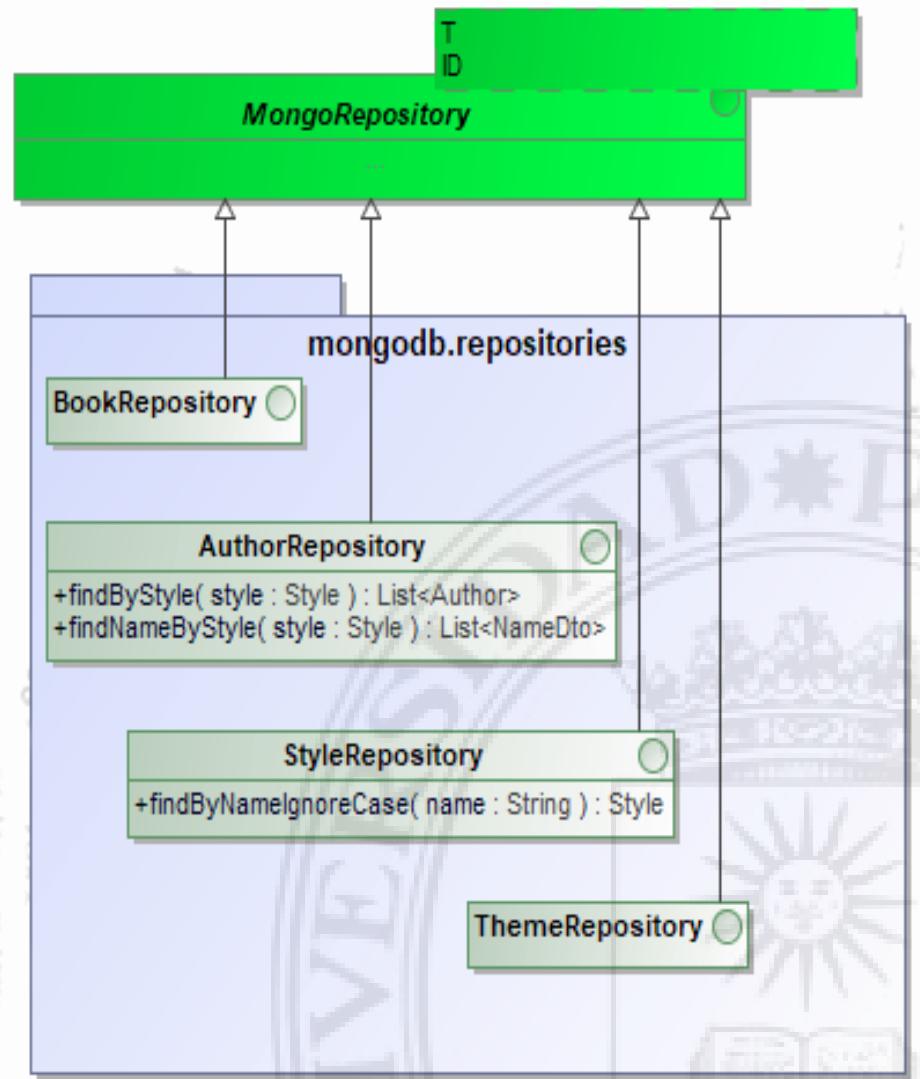
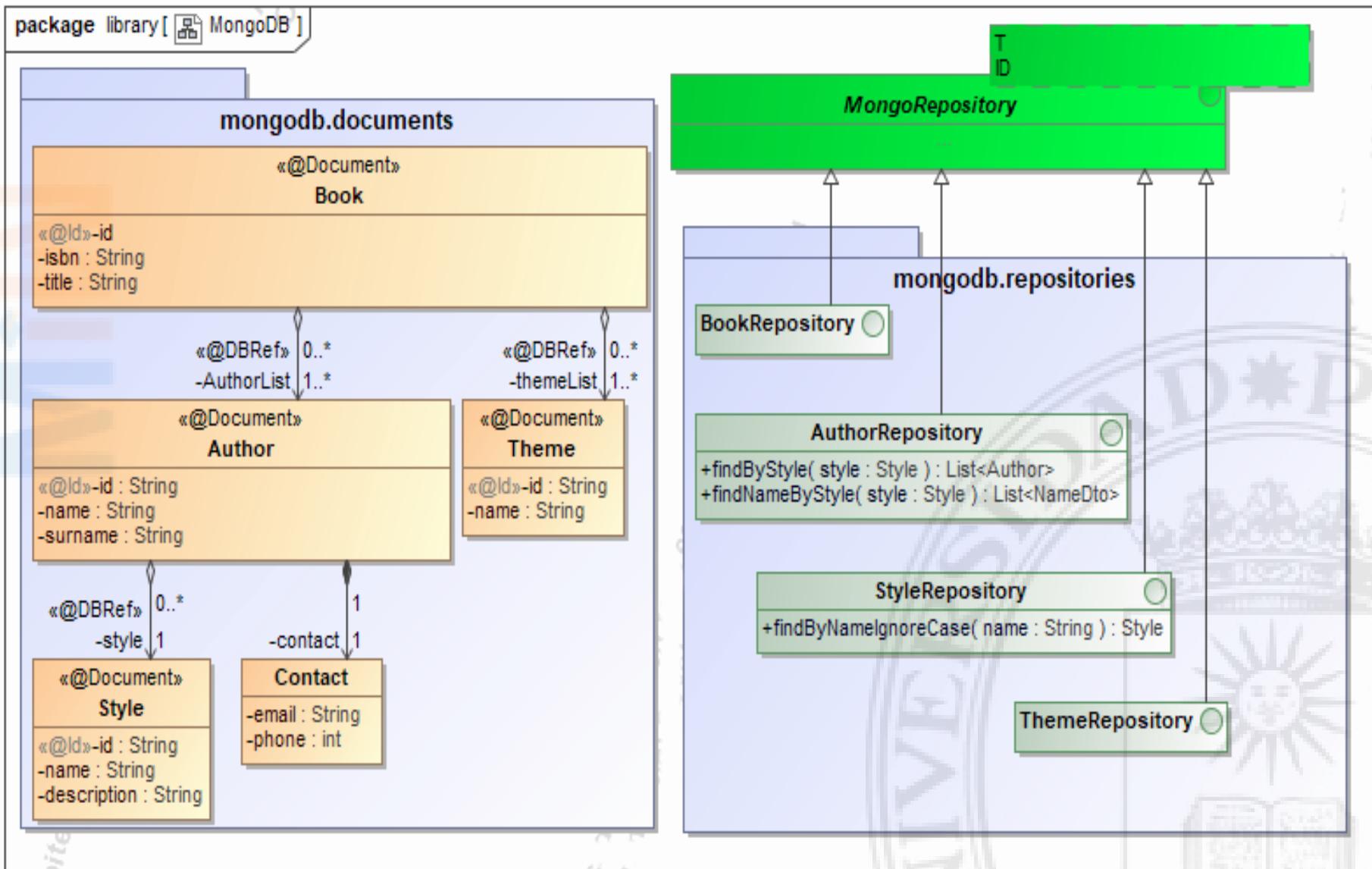


✍ Library

- ✍ Indicar las anotaciones (@*) necesarias en la clases



Library



Consultas y Borrados. Con Nombre de Métodos

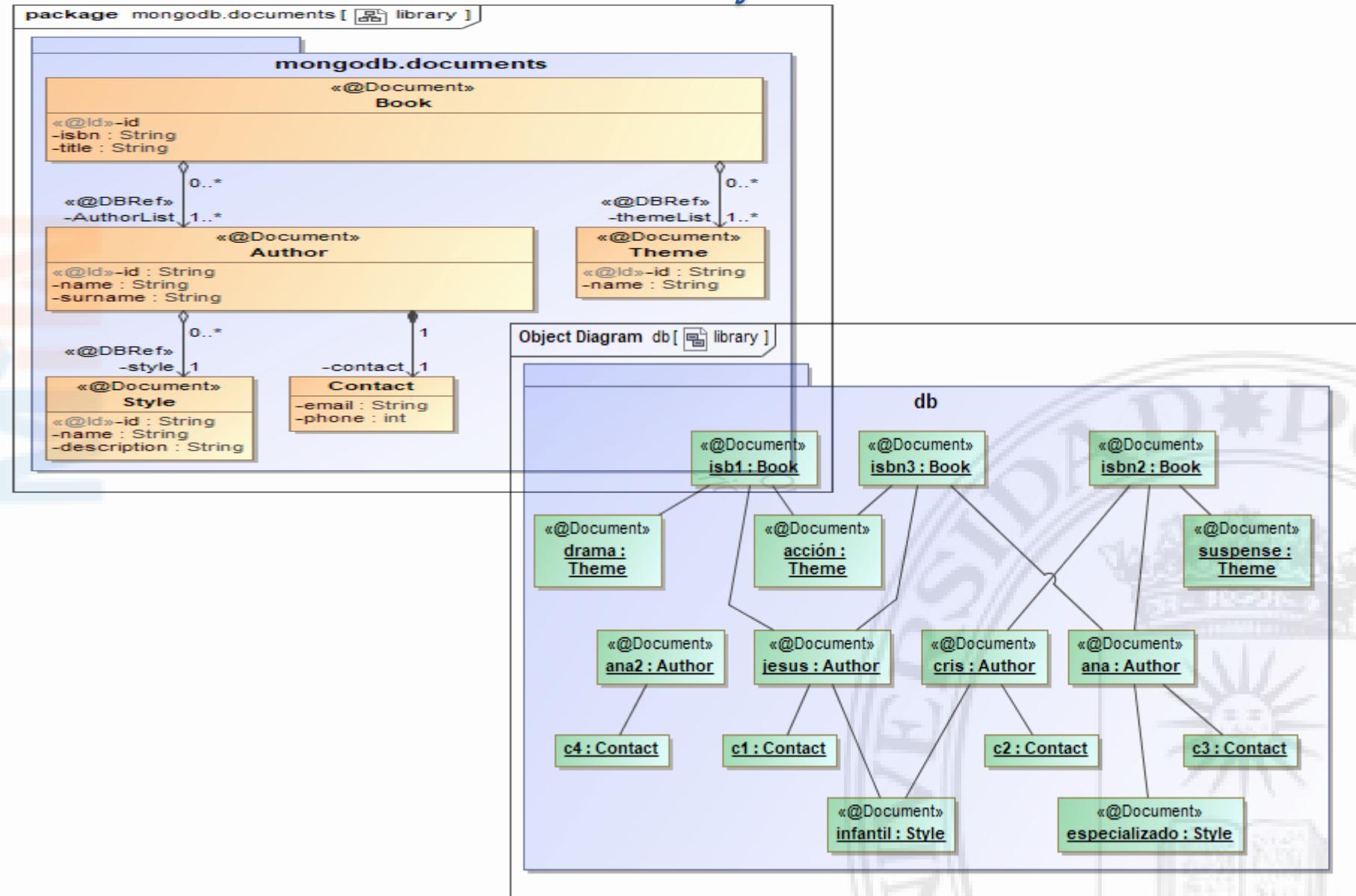
Keyword	Sample	Logical result
After	<code>findByBirthdateAfter(Date date)</code>	<code>{"birthdate" : {"\$gt" : date}}</code>
GreaterThan	<code>findByAgeGreaterThan(int age)</code>	<code>{"age" : {"\$gt" : age}}</code>
GreaterThanOrEqualTo	<code>findByAgeGreaterThanOrEqualTo(int age)</code>	<code>{"age" : {"\$gte" : age}}</code>
Before	<code>findByBirthdateBefore(Date date)</code>	<code>{"birthdate" : {"\$lt" : date}}</code>
LessThan	<code>findByAgeLessThan(int age)</code>	<code>{"age" : {"\$lt" : age}}</code>
LessThanOrEqualTo	<code>findByAgeLessThanOrEqualTo(int age)</code>	<code>{"age" : {"\$lte" : age}}</code>
Between	<code>findByAgeBetween(int from, int to)</code>	<code>{"age" : {"\$gt" : from, "\$lt" : to}}</code>
In	<code>findByAgeIn(Collection ages)</code>	<code>{"age" : {"\$in" : [ages...]}}</code>
NotIn	<code>findByAgeNotIn(Collection ages)</code>	<code>{"age" : {"\$nin" : [ages...]}}</code>
IsNotNull, NotNull	<code>findByFirstnameNotNull()</code>	<code>{"firstname" : {"\$ne" : null}}</code>
IsNull, Null	<code>findByFirstnameNull()</code>	<code>{"firstname" : null}</code>
Like, StartingWith, EndingWith	<code>findByFirstnameLike(String name)</code>	<code>{"firstname" : name} (name as regex)</code>
NotLike, IsNotLike	<code>findByFirstnameNotLike(String name)</code>	<code>{"firstname" : { "\$not" : name }} (name as regex)</code>
Containing on String	<code>findByFirstnameContaining(String name)</code>	<code>{"firstname" : name} (name as regex)</code>
NotContaining on String	<code>findByFirstnameNotContaining(String name)</code>	<code>{"firstname" : { "\$not" : name}} (name as regex)</code>
Containing on Collection	<code>findByAddressesContaining(Address address)</code>	<code>{"addresses" : {"\$in" : address}}</code>
NotContaining on Collection	<code>findByAddressesNotContaining(Address address)</code>	<code>{"addresses" : { "\$not" : { "\$in" : address}}}</code>
Regex	<code>findByFirstnameRegex(String firstname)</code>	<code>{"firstname" : {"\$regex" : firstname }}</code>
(No keyword)	<code>findByFirstname(String name)</code>	<code>{"firstname" : name}</code>
Not	<code>findByFirstnameNot(String name)</code>	<code>{"firstname" : {"\$ne" : name}}</code>
Near	<code>findByLocationNear(Point point)</code>	<code>{"location" : {"\$near" : [x,y]}}</code>
Near	<code>findByLocationNear(Point point, Distance max)</code>	<code>{"location" : {"\$near" : [x,y], "\$maxDistance" : max}}</code>
Near	<code>findByLocationNear(Point point, Distance min, Distance max)</code>	<code>{"location" : {"\$near" : [x,y], "\$minDistance" : min, "\$maxDistance" : max}}</code>
Within	<code>findByLocationWithin(Circle circle)</code>	<code>{"location" : {"\$geoWithin" : {"\$center" : [[x, y], distance]}}}</code>
Within	<code>findByLocationWithin(Box box)</code>	<code>{"location" : {"\$geoWithin" : {"\$box" : [[x1, y1], x2, y2]}}}</code>
IsTrue, True	<code>findByActiveIsTrue()</code>	<code>{"active" : true}</code>
IsFalse, False	<code>findByActiveIsFalse()</code>	<code>{"active" : false}</code>
Exists	<code>findByLocationExists(boolean exists)</code>	<code>{"location" : {"\$exists" : exists }}</code>

Json Query

```
I UnRelatedRepository.java ×

26
27     // @Query("{ 'nick':?0 })" // Query NOT necessary
28     UnRelatedDocument findByNick(String nick);
29
30     // FIELDS: _id: incluido por defecto, 1: solo los especificados, 0: todos excepto el especificado
31     @Query(value = "{ 'nick':?0 }", fields = "{ 'bornDate':1,'large':1 }")
32     Query1Dto findIdAndBornDateAndLargeByNick(String nick);
33
34     @Query(value = "{ 'nick':?0 }", fields = "{ '_id':0,'bornDate':1 }")
35     Query2Dto findBornDateByNick(String nick);
36
37     // @Query("{ $and:[{ 'nick':?0 },{'large':?1}]}") // Query NOT necessary
38     UnRelatedDocument findByNickAndLarge(String nick, String large);
39
40     // @Query("{nick:{ $in:?0 } }") // Query NOT necessary
41     List<UnRelatedDocument> findByNickIn(Collection nicks);
42
43     // $options: 'i': Case insensitivity // allow NULL: all elements
44     @Query(" ?#{{ [0] == null ? { $where : 'true' } : { nick : { $regex:[0], $options: 'i' } } } }")
45     List<UnRelatedDocument> findByNickLikeIgnoreCaseNullSafe(String nick);
46
47     @Query("{ $and:[ " // allow NULL: all elements
48             + " ?#{{ [0] == null ? { $where : 'true' } : { nick : { $regex:[0], $options: 'i' } } } },"
49             + " ?#{{ [1] == null ? { $where : 'true' } : { large : { $regex:[1], $options: 'i' } } } }"
50             + " ] }")
51     List<UnRelatedDocument> findByNickLikeAndLargeLikeNullSafe(String nick, String large);
52 }
```

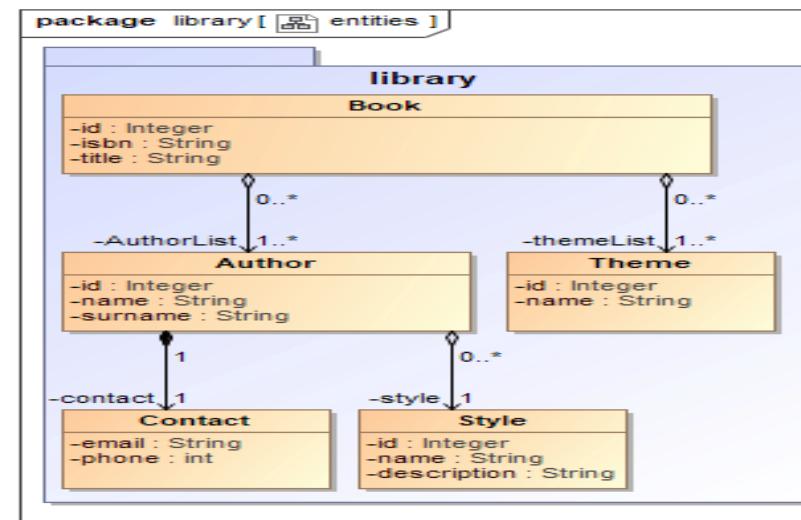
Library



Library

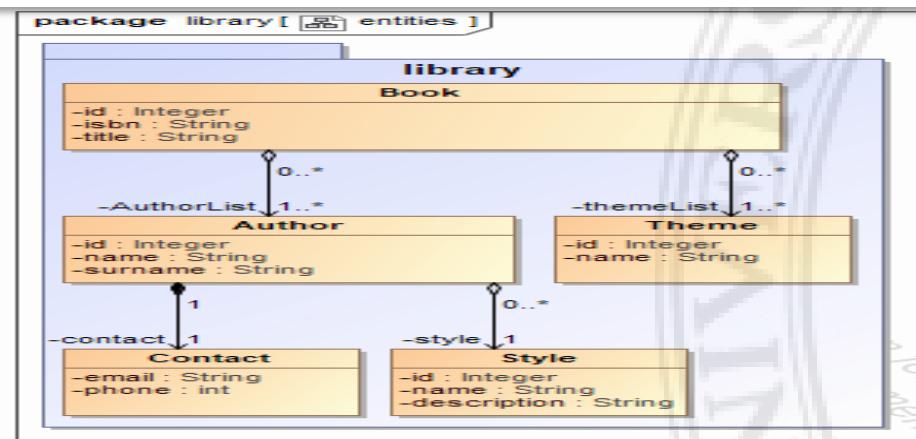
I AuthorRepository.java X

```
12 public interface AuthorRepository extends MongoRepository<Author, String> {  
13  
14     List<Author> findByStyle(Style style);  
15  
16     List<NameDto> findNameByStyle(Style style);  
17  
18     List<Author> findByContactEmail(String email);  
19  
20     @Query(value = "{ 'name':?0}", fields = "{ '_id':0,'contact':1}")  
21     List<ContactDto> findContactByName(String name);  
22  
23     @Query(value = "{ 'name':?0}", fields = "{ '_id':0,'contact.email':1}")  
24     List<ContactDto> findContactEmailByName(String name); // [{"contact": {"email": "a@gmail.com"} }]  
25 }
```



Library

```
c LibraryService.java x
36
37     public List<EmailDto> findContactEmailByAuthorName(String name) {
38         return this.authorRepository.findContactEmailByName(name) .
39             stream().map(contact -> new EmailDto(contact.getContact().getEmail()))
40             .collect(Collectors.toList());
41     }
42
43     public List<Author> findAuthorByBookByThemeName(String name) {
44         List<Theme> themes = this.themeRepository.findByName(name);
45         return this.bookRepository.findByThemeListIn(themes)
46             .map(Book::getAuthorList).flatMap(Collection::stream).collect(Collectors.toList());
47     }
48
49     public List<String> findAuthorNameByExampleAuthor(Author author) {
50         return this.authorRepository.findAll(Example.of(author))
51             .stream().map(Author::getName).collect(Collectors.toList());
52     }
```



Library

```
c RepositoriesLibraryService.java x
24
25     public void seedDb() {
26         Theme[] themes = {
27             new Theme( name: "Acción"),
28             new Theme( name: "Suspense"),
29             new Theme( name: "Drama") };
30         themeRepository.saveAll(Arrays.asList(themes));
31
32         Style[] styles = {
33             new Style( name: "Infantil", description: "Lectura sencilla"),
34             new Style( name: "Especializado", description: "Expertos de la temática") };
35         styleRepository.saveAll(Arrays.asList(styles));
36
37         Author[] authors = {
38             new Author( name: "Jesús", surname: "Ber", new Contact( email: "j@gmail.com", phone: 666666661), styles[0]),
39             new Author( name: "Cris", surname: "Ber", new Contact( email: "c@gmail.com", phone: 666666662), styles[0]),
40             new Author( name: "Ana", surname: "Ber", new Contact( email: "a@gmail.com", phone: 666666663), styles[1]),
41             new Author( name: "Ana", surname: "Reb", new Contact( email: "a2@gmail.com", phone: 666666663), styles[1])};
42         authorRepository.saveAll(Arrays.asList(authors));
43
44         Book[] books = {
45             new Book( isbn: "isbn1", title: "La naranja", Arrays.asList(themes[0], themes[2]), Arrays.asList(authors[0])),
46             new Book( isbn: "isbn2", title: "El limón", Arrays.asList(themes[1]), Arrays.asList(authors[1], authors[2])),
47             new Book( isbn: "isbn3", title: "El pepino", Arrays.asList(themes[0]), Arrays.asList(authors[2]))};
48         bookRepository.saveAll(Arrays.asList(books));
49     }
50
51     public void deleteDb() {
52         bookRepository.deleteAll();
53         themeRepository.deleteAll();
54         authorRepository.deleteAll();
55         styleRepository.deleteAll();
56     }
57 }
```

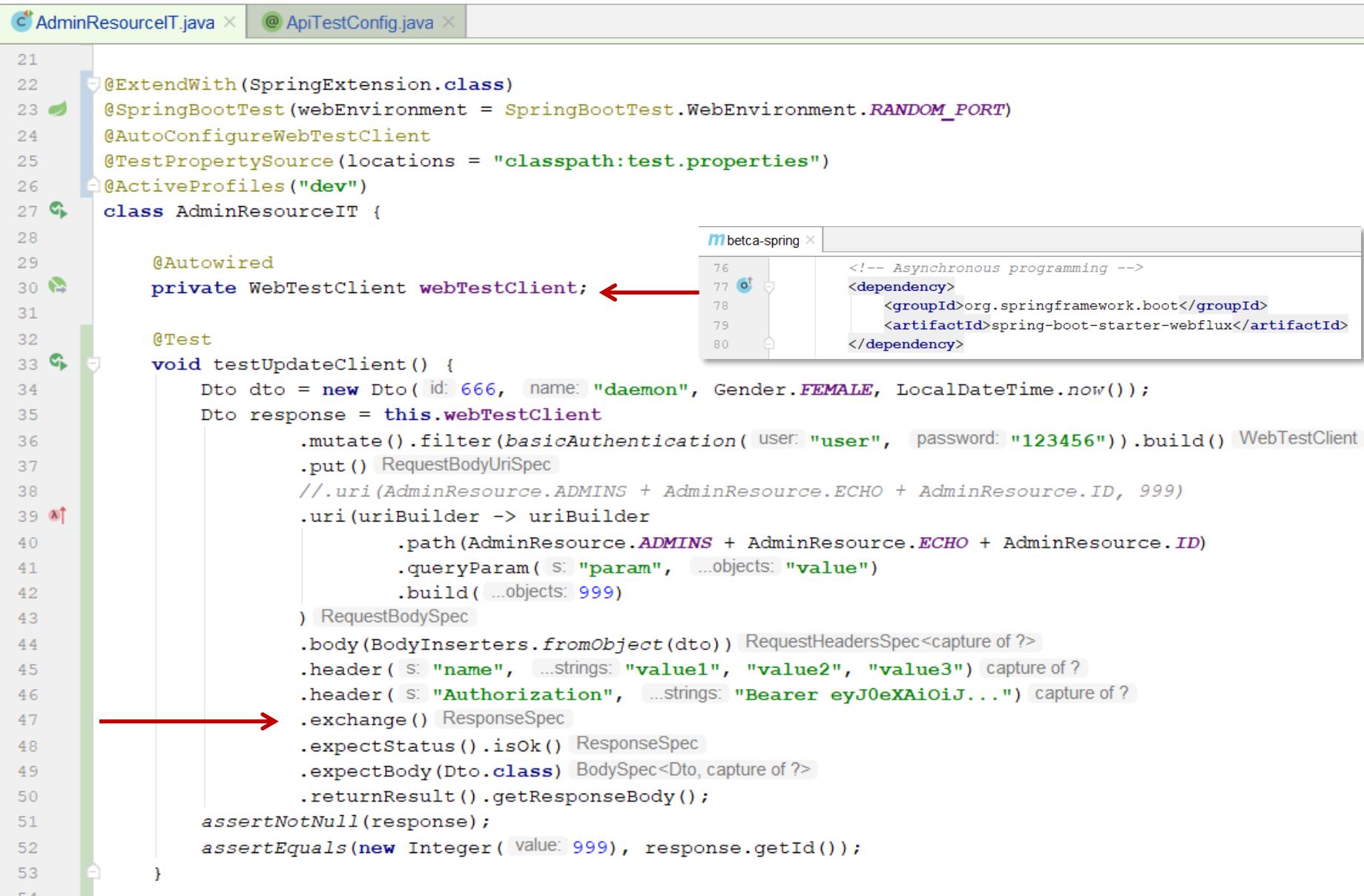
REST

- *REST (Representational State Transfer)* es un estilo arquitectónico para el desarrollo de aplicaciones sobre la Web
- Servidor sin estado. El servidor **no gestiona** los recursos de los clientes a través de sesiones
- El servicio se organiza como un conjunto de recursos con cuatro operaciones (**CRUD**), pero ofrece un servicio por encima de la capa DAO :
 - *Create* (post)
 - *Read* (get)
 - *Update* (put/patch)
 - *Delete* (delete)

Operaciones: CRUD

- Métodos de HTTP:
 - **Create: post.** Para crear un nuevo recurso
 - Operación no segura y no idempotente
 - Respuesta: 201 (OK) o el tipo de error
 - Recomendable devolver una referencia del nuevo recurso
 - Recomendable devolver el recurso
 - **Read: get.** Para obtener un recurso o un conjunto
 - Operación segura e idempotente
 - Respuesta: 200 (OK) o el tipo de error
 - Devolver el recurso o un conjunto
 - **Update: put/patch.** Para actualizar un recurso
 - Operación no segura e idempotente
 - Respuesta 200 (OK) o el tipo de error
 - Recomendable devolver el recurso
 - **Delete: delete.** Para eliminar un recurso
 - Operación no segura e idempotente
 - Respuesta 204 (OK) sin contenido o el tipo de error

Cliente Rest



The screenshot shows a Java code editor with two tabs: `AdminResourceIT.java` and `@ApiTestConfig.java`. The `AdminResourceIT.java` file contains a test class `AdminResourceIT` that uses `WebTestClient` to interact with a REST endpoint. A red arrow points from the line `private WebTestClient webTestClient;` to a dependency configuration in the build path. The build path window shows a dependency on `spring-boot-starter-webflux`.

```
21
22     @ExtendWith(SpringExtension.class)
23     @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
24     @AutoConfigureWebTestClient
25     @TestPropertySource(locations = "classpath:test.properties")
26     @ActiveProfiles("dev")
27     class AdminResourceIT {
28
29         @Autowired
30         private WebTestClient webTestClient; ←
31
32         @Test
33         void testUpdateClient() {
34             Dto dto = new Dto(id: 666, name: "daemon", Gender.FEMALE, LocalDateTime.now());
35             Dto response = this.webTestClient
36                 .mutate().filter(basicAuthentication(user: "user", password: "123456")).build() WebTestClient
37                 .put() RequestBodyUriSpec
38                 // .uri(AdminResource.ADMIN + AdminResource.ECHO + AdminResource.ID, 999)
39                 .uri(uriBuilder -> uriBuilder
40                     .path(AdminResource.ADMIN + AdminResource.ECHO + AdminResource.ID)
41                     .queryParam(s: "param", ...objects: "value")
42                     .build(...objects: 999)
43                 ) RequestBodySpec
44                 .body(BodyInserters.fromObject(dto)) RequestHeadersSpec<capture of ?>
45                 .header(s: "name", ...strings: "value1", "value2", "value3") capture of ?
46                 .header(s: "Authorization", ...strings: "Bearer eyJ0eXAiOiJ...") capture of ?
47                 .exchange() ResponseSpec
48                 .expectStatus().isOk() ResponseSpec
49                 .expectBody(Dto.class) BodySpec<Dto, capture of ?>
50                 .returnResult().getResponseBody();
51                 assertNotNull(response);
52                 assertEquals(new Integer(value: 999), response.getId());
53         }
54     }
```

metac-spring x

76
77
78
79
80

<!-- Asynchronous programming -->
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>

Cliente Rest



A screenshot of a Java code editor showing a file named `ApiTestConfig.java`. The code defines a test configuration interface with annotations for Spring Boot testing.

```
1 package es.upm.miw.restControllers;
2 import ...
3
4 @Target(ElementType.TYPE)
5 @Retention(RetentionPolicy.RUNTIME)
6
7 @ExtendWith(SpringExtension.class)
8 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
9 @AutoConfigureWebTestClient
10 @TestPropertySource(locations = "classpath:test.properties")
11 @ActiveProfiles("dev")
12 public @interface ApiTestConfig {
13 }
```

Recursos

- Son clases Java, con la anotación `@RestController ("path")` para representar un recurso API Rest. Son las clases que responden finalmente a las peticiones
- Cada método de la clase, para que responda a una petición, debe llevar la anotación:
 - `@RequestMapping(value = "path", method = RequestMethod.POST)`
 - value: es la ruta parcial, si es una variable, se establece entre llaves, por ejemplo, `"/{id}"`
 - method: método de la petición (POST, GET, PUT/PATH, DELETE)
 - `@GetMapping` → `@RequestMapping(method = RequestMethod.GET)`
 - `@PostMapping` → `@RequestMapping(method = RequestMethod.POST)`
 - `@PutMapping`
 - `@PatchMapping`
 - `@DeleteMapping`

Recursos

■ Inyección de parámetros

- `@RequestHeader(...), @PathVariable, @RequestParam`
 - *value*: nombre del parámetro, si no se establece, es el nombre de la variable
 - *defaultValue*: valor por defecto, si se establece el parámetro es opcional
 - *required*: se indica si es requerido (*true o false*)

■ Datos del cuerpo

- Por defecto, se transportan en JSON
 - `@JsonIgnore` para aquellos campos que no queramos que se conviertan a JSON
 - `@JsonInclude (Include.NON_NULL)` para que no se añada a la respuesta si es null
- Listas: `List<String>, List<Integer>, List<Dto> ...`

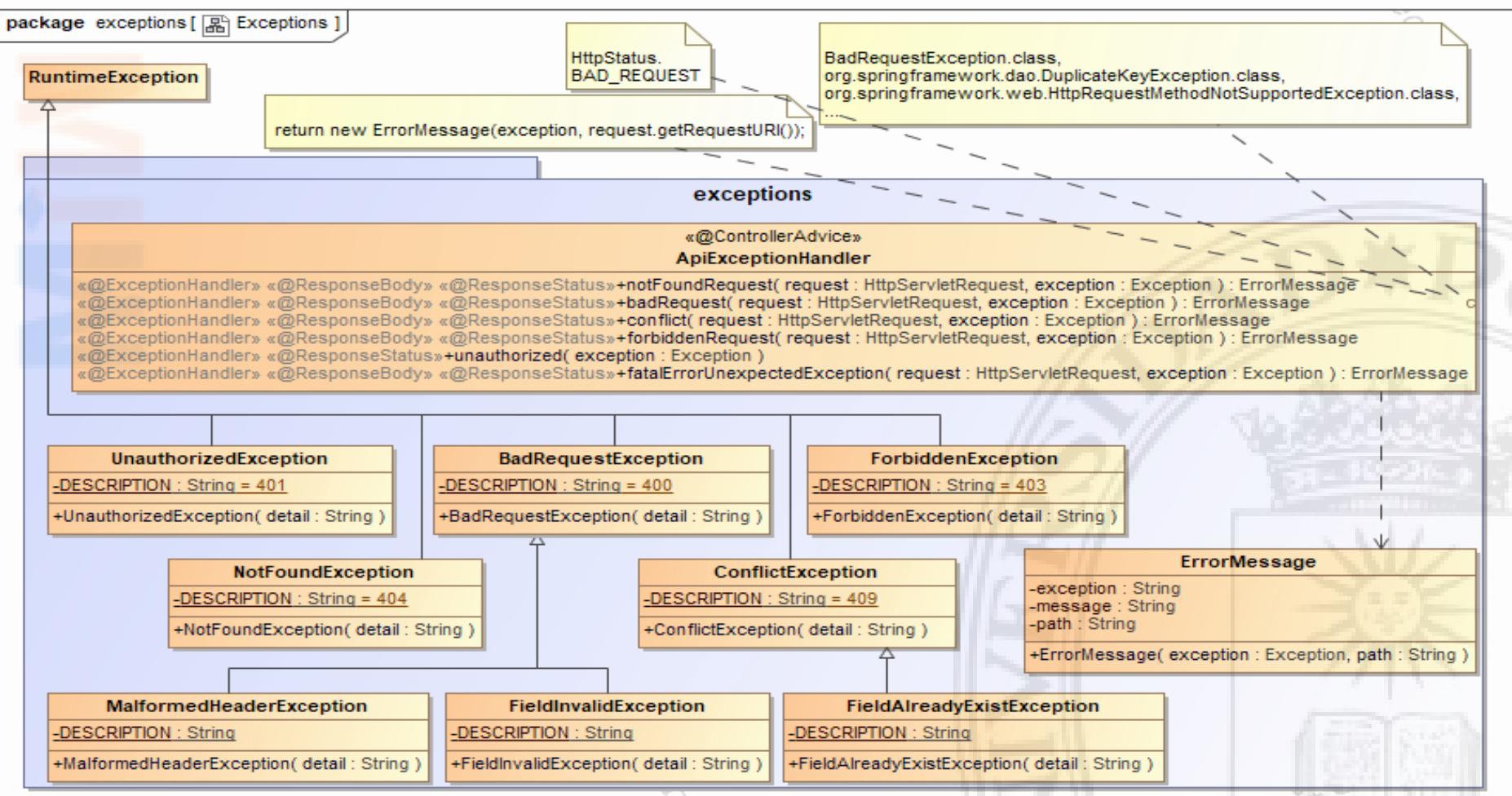
■ Ejemplo: *AdminResource*

■ Ejercicio

1. Añadir un recurso (`admins/calculator`), reciben dos parámetros(`dividendo`, `divisor`), devuelve un String: {"division":xxx}: GET
2. Añadir un recurso (`admins/calculator2`), recibe un *DTO* que representa la división (atributos `dividendo`, `divisor`) y devuelve un *Double* en el cuerpo: POST
3. Añadir un recurso (`admins/calculator3`), recibe una lista de *DTO's* (`dividendo`, `divisor`) y devuelve una `List<Double>`: POST

Manejo de errores HTTP . Excepciones

- Si la petición no es satisfactoria, se provoca una excepción. El manejador de excepciones captura la excepción y se le envía al cliente el mensaje de error en el cuerpo y con la respuesta HTTP correspondiente

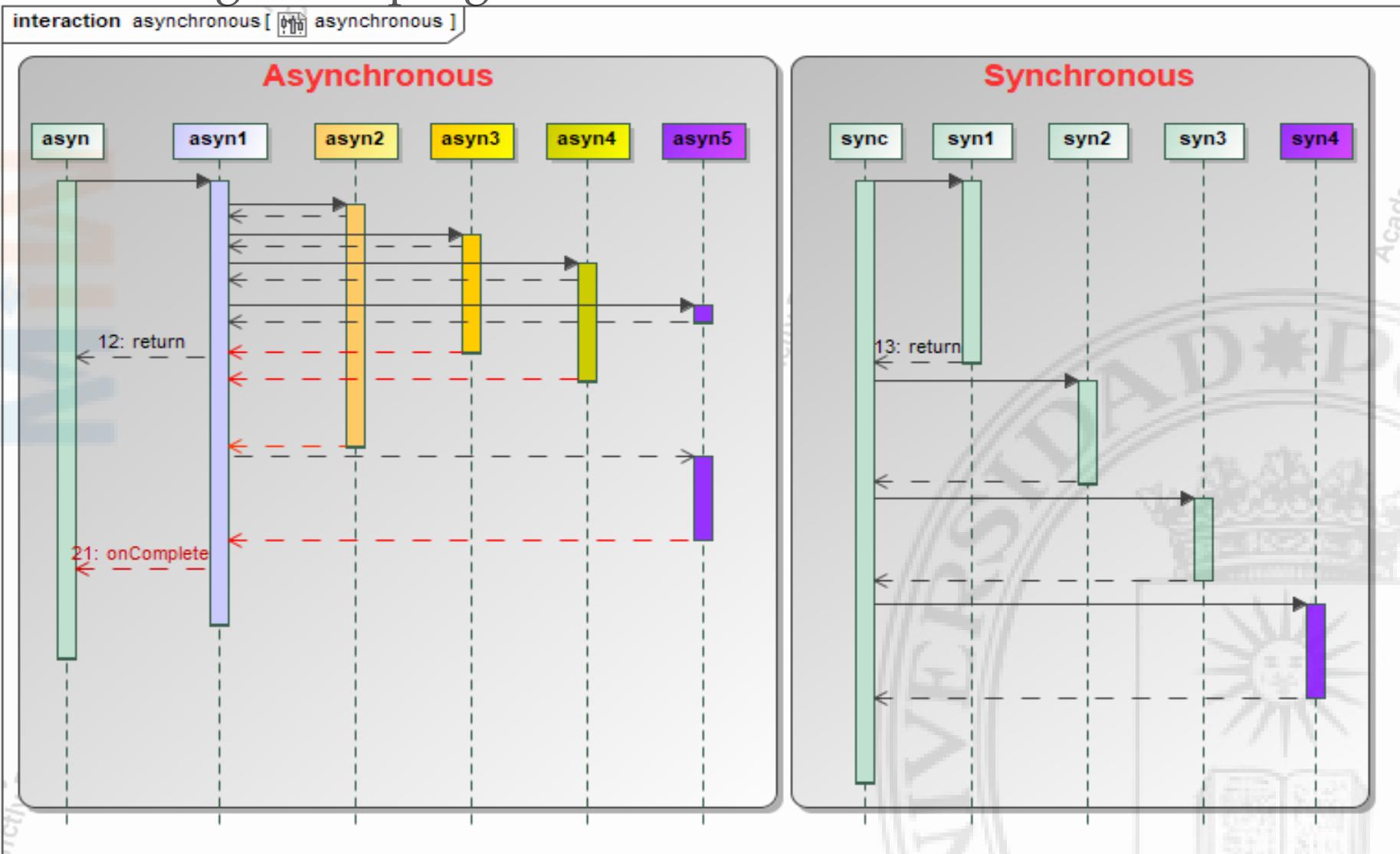


Respuestas mediante excepciones

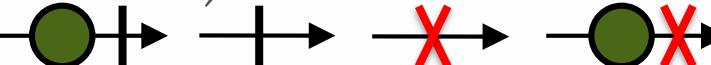
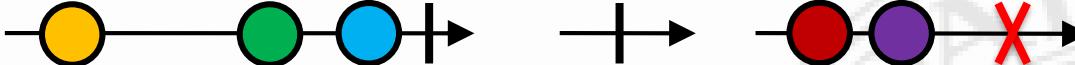
- Enumerado: *org.springframework.http.HttpStatus*
 - SUCCESSFUL:
 - OK: 200, CREATED: 201, NO_CONTENT: 204
 - REDIRECTION:
 - MOVED_PERMANENTLY: 301
 - CLIENT_ERROR:
 - BAD_REQUEST: 400, UNAUTHORIZED: 401, FORBIDDEN: 403, NOT_FOUND: 404, METHOD_NOT_ALLOWED: 405, CONFLICT: 409
 - SERVER_ERROR:
 - INTERNAL_SERVER_ERROR: 500, NOT_IMPLEMENTED: 501
-  *Ejemplo: Exception Resource*
-  *Ejercicio*
 -  Añadir al recurso: si la “id” es 999, provocar la excepción de prohibido (*ForbiddenException*) +++ TEST
 -  Añadir al recurso: si la “id” es 900, provocar una excepción propia, que hereda de “*ForbiddenException*” +++ TEST

Programación asíncrona: proyecto Reactor

- Paradigma de programación diferente



Programación asíncrona: proyecto Reactor

- Proyecto: *Reactor*
 - <https://projectreactor.io/docs/core/release/reference>
 - <https://projectreactor.io/docs/core/release/api/>
- Publisher (Mono - Flux)
 - Un Mono <T> 
 - Emite de forma asíncrona 0 () o 1 elemento (*onNext*) y termina con una señal (*onComplete*).
 - Termina con una señal (*onError*).
 - Un Flux<T> 
 - Emite una secuencia asíncrona de 0 a N elementos (*onNext*) y termina con una señal (*onComplete*).
 - Termina con una señal (*onError*).
- Métodos que devuelven un tipo Publisher (Mono, Flux) no deberían suscribirse (*subscribe* o *block*) porque ello podría romper la cadena del publicador
-  *ReactiveProgramming & ReactiveProgrammingTest*

Programación asíncrona: proyecto *Reactor*

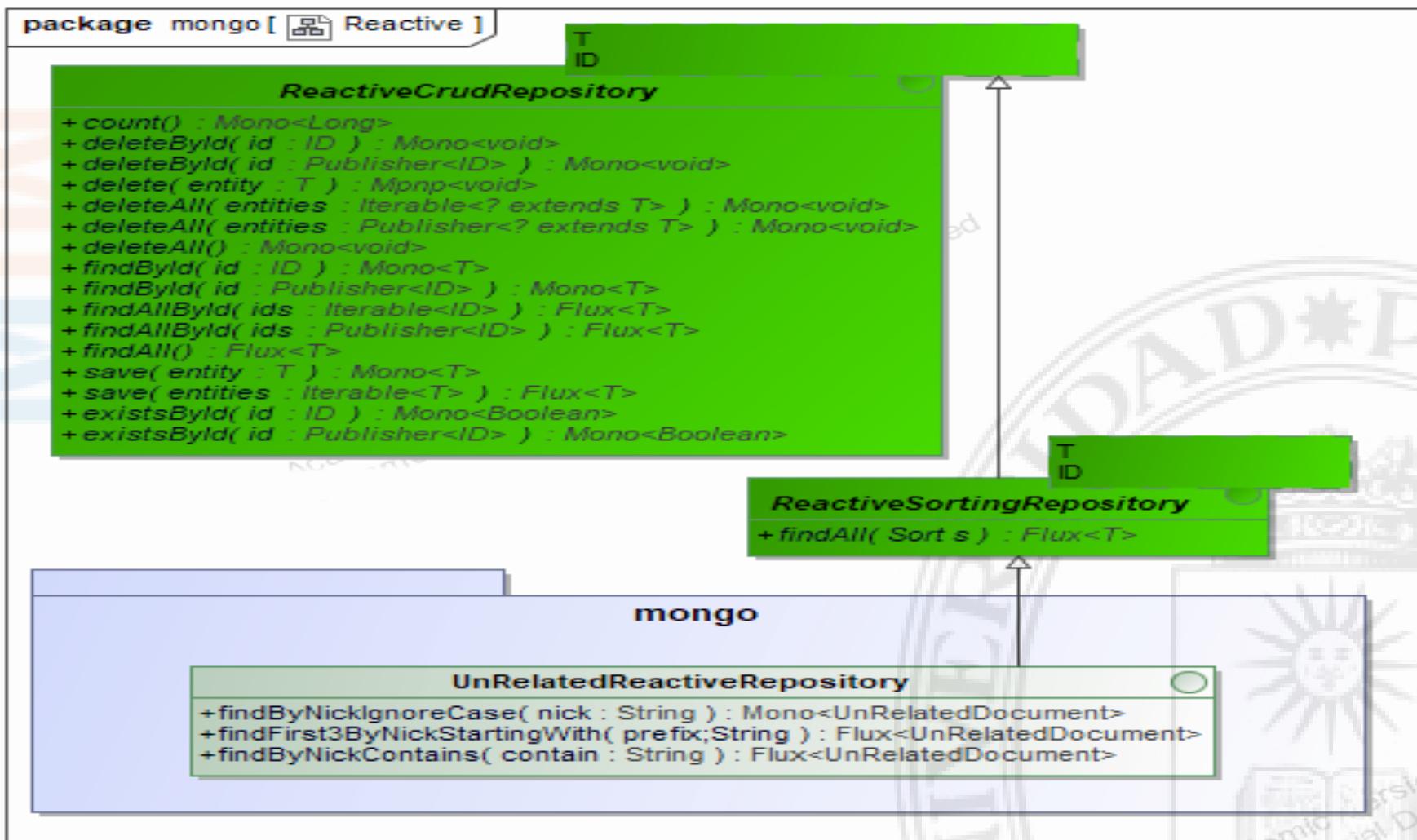
- Métodos para pre-procesos de datos
 - **map(Function<T,R>)**. Transforma el ítem<T> en el tipo <R>
 - `Mono.just("10").map(BigDecimal::new);`
 - `Mono.just("10").map(s->new BigDecimal(s+",99");`
 - `reactRepository.findAll().map(...);`
 - **filter(Predicate<T>)**. Se descartan los ítems que no cumplen
 - `Flux.just(1,4,7,8,13).filter(n->n%2==0);`
 - `reactRepository.findByName("Jesus").filter(...)`
 - **doOnNext(Consumer<T>)**. Añade comportamiento sobre terceros
 - `doOnSuccess, doOnError, doOnSubscribe...`
 - `Flux.just(1,4,7,8,13).doOnNext(System.out::println);`
 - **handle(Biconsumer<T,SynchronousSink>)**. Se puede cambiar el sincronismo
 - `Flux.just(1,4,-7,8,13).handle((n,sink)-> {`
 - `if(n<0) sink.error(new RuntimeException("..."));`
 - `else sink.next(n);`
 - `});`
 - **switchIfEmpty(Publisher<T>)**
 - `Mono.empty().switchIfEmpty(Mono.just("one"))`

Programación asíncrona: proyecto *Reactor*

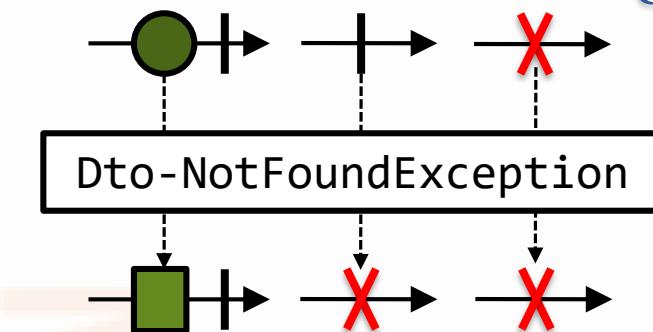
- Métodos para procesos o sincronizaciones de flujos
 - *concat, merge, zip*
 - *and, or, when, then...*
- *Repositorios*
 - Repositorio síncrono (*documentRepository*)
 - *Optional<T> findById(ID id)*
 - *Iterable<T> findAll()*
 - Repositorio asíncrono (*documentReactRepository*)
 - *Mono<T> findById(ID id)*
 - *Flux<T> findAll()*

Reactive

- ✍ *UnRelatedController & UnRelatedControllerIT*



Programación asíncrona

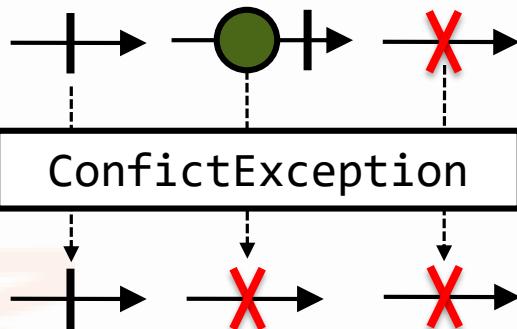


c UnRelatedController.java x c UnRelatedControllerIT.java x

```
29     public LocalDateTime findBornDateByNickAssured(String nick) {  
30         return this.unRelatedRepository.findByNick(nick)  
31             .orElseThrow(() -> new NotFoundException("..."))  
32             .getBornDate();  
33     }  
34  
35     public Mono<LocalDateTime> findBornDateByNickAssuredReact(String nick) {  
36         return this.unRelatedReactRepository.findByNick(nick)  
37             .switchIfEmpty(Mono.error(new NotFoundException("...")))  
38             .map(UnRelatedDocument::getBornDate);  
39     }  
40  
41     public LocalDateTime findBornDateByNickAssuredReactERRORChainBreak(String nick) {  
42         return this.unRelatedReactRepository.findByNick(nick)  
43             .switchIfEmpty(Mono.error(new NotFoundException("...")))  
44             .map(UnRelatedDocument::getBornDate).block();  
45     }
```

The code snippet shows three methods in the `UnRelatedController` class. The first method, `findBornDateByNickAssured`, is annotated with a yellow box and labeled "Synchronous". The second method, `findBornDateByNickAssuredReact`, is annotated with a yellow box and labeled "Asynchronous". The third method, `findBornDateByNickAssuredReactERRORChainBreak`, is crossed out with a large red line and has its annotations removed.

Programación asíncrona

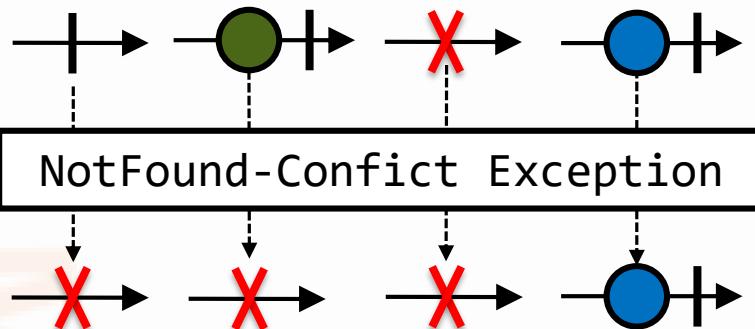


c UnRelatedController.java x

c UnRelatedControllerT.java x

```
46  
47     public void notExistsByNickAssured(String nick) { → Synchronous  
48         if (this.unRelatedRepository.findByNick(nick).isPresent()) {  
49             throw new ConflictException("...");  
50         }  
51     }  
52  
53     public Mono<Void> notExistsByNickAssuredReact(String nick) { → Asynchronous  
54         return this.unRelatedReactRepository.findByNick(nick)  
55             .handle((document, sink) -> sink.error(new ConflictException("...")));  
56     }
```

Programación asíncrona



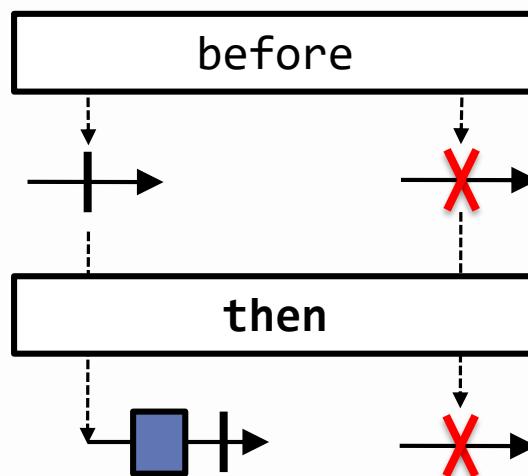
c UnRelatedController.java x c UnRelatedControllerIT.java x

```
57
58     public UnRelatedDocument findByNickIsFemaleAssuredSynchronous(String nick) {
59         UnRelatedDocument document = this.unRelatedRepository.findByNick(nick)
60             .orElseThrow(() -> new NotFoundException("..."));
61         if (document.getGender().equals(Gender.MALE)) {
62             throw new ConflictException("...");
63         }
64         return document;
65     }
66
67     public Mono<UnRelatedDocument> findByNickIsFemaleAssuredAsynchronous(String nick) {
68         Mono<UnRelatedDocument> unRelated = this.unRelatedReactRepository.findByNick(nick)
69             .switchIfEmpty(Mono.error(new NotFoundException("...")));
70         return unRelated.handle((document, sink) -> {
71             if (document.getGender().equals(Gender.FEMALE)) {
72                 sink.next(document);
73             } else {
74                 sink.error(new ConflictException("..."));
75             }
76         });
77     }
```

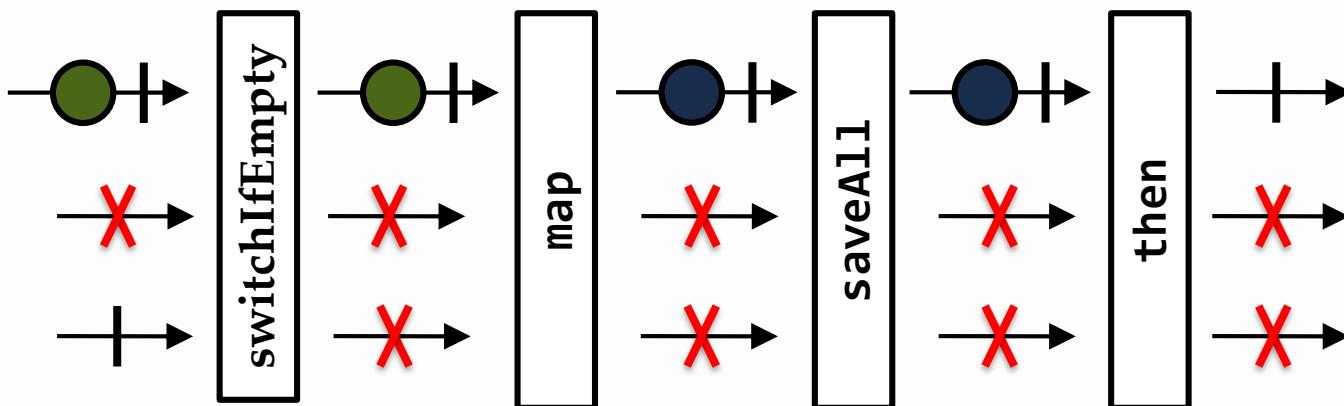
The code block shows two methods: `findByNickIsFemaleAssuredSynchronous` and `findByNickIsFemaleAssuredAsynchronous`. The synchronous method uses `orElseThrow` to handle a `NotFoundException`, while the asynchronous method uses `switchIfEmpty` and a `Mono.error` block to handle the exception. Callouts point to these sections with the labels "Synchronous" and "Asynchronous".

Programación asíncrona

- Sincronizar el inicio de una petición a la finalización correcta de otra petición
 - Síncrono
 - ... *before*
 - *document = new ...;*
 - *repository.save(document);*
 - Asíncrono
 - *Mono<Void> before = ...*
 - *document = new ...;*
 - *Mono<> t = before.then(...)*



Programación asíncrona



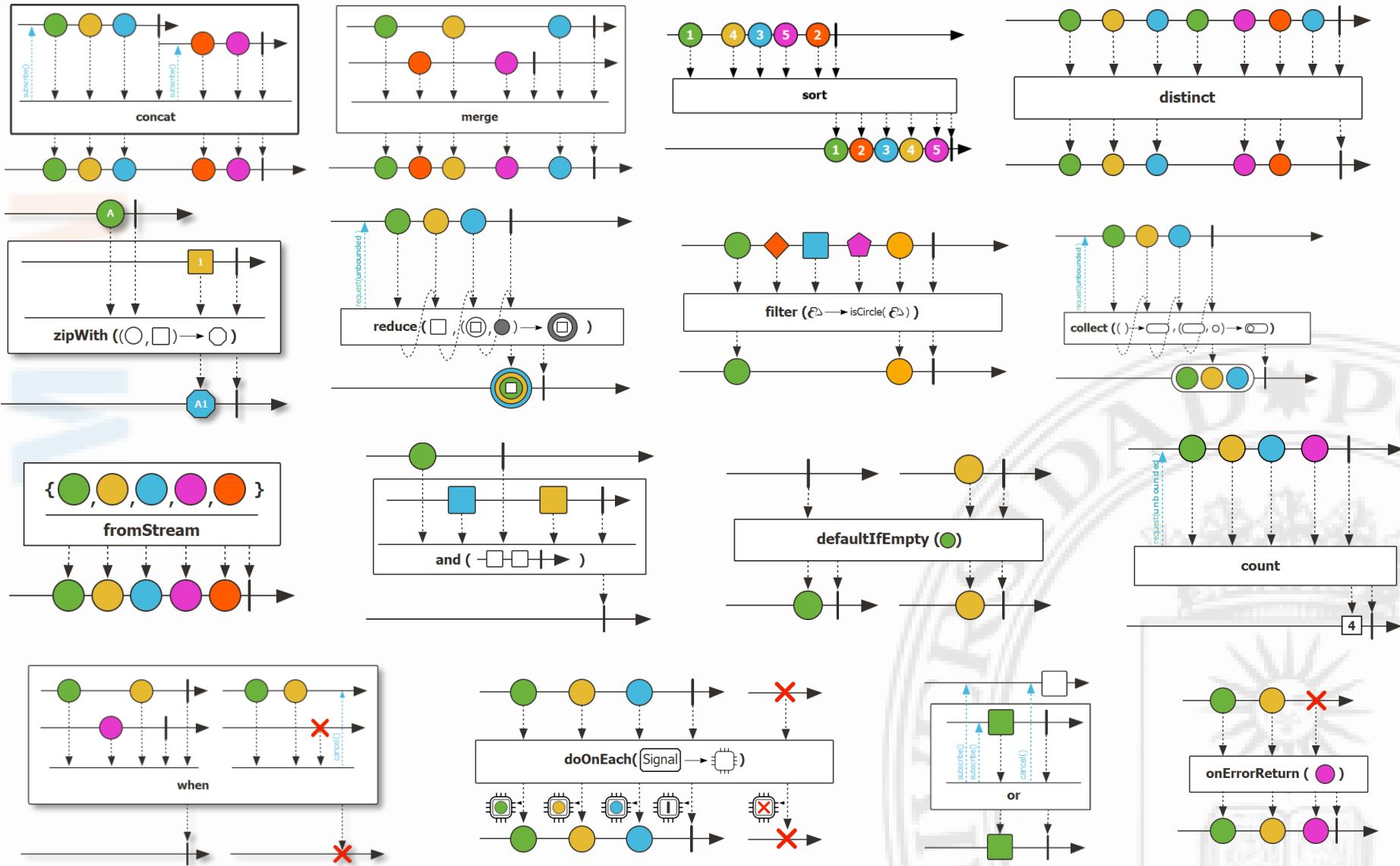
UnRelatedController.java

UnRelatedControllerIT.java

```
57
58     public void findByNickAssuredAndUpdateLarge(String nick, String large) {
59         UnRelatedDocument entity = this.unRelatedRepository.findByNick(nick)
60             .orElseThrow(() -> new NotFoundException("..."));
61         entity.setLarge(large);
62         this.unRelatedRepository.save(entity);
63     }
64
65     public Mono<Void> findByNickAssuredAndUpdateLargeReact(String nick, String large) {
66         Mono<UnRelatedDocument> unRelated = this.unRelatedReactRepository.findByNick(nick)
67             .switchIfEmpty(Mono.error(new NotFoundException("...")))
68             .map(document -> {
69                 document.setLarge(large);
70                 return document;
71             });
72         return this.unRelatedReactRepository.saveAll(unRelated).then();
73     }
```

A diagram on the right side of the code editor highlights the difference between synchronous and asynchronous code. A callout box labeled "Synchronous" points to the first code block (lines 57-64). Another callout box labeled "Asynchronous" points to the second code block (lines 65-73).

Programación asíncrona



Reactive API-Rest

- *✍ WebFluxResource & WebFluxResourceIT*

```
c WebFluxResource.java x
26     @GetMapping(value = MONO)
27     public Mono<Dto> readMono() {
28         return Mono.just(new Dto( id: 666));
29     }
30
31     @GetMapping(value = MONO_EMPTY)
32     public Mono<Dto> readMonoEmpty() {
33         return Mono.empty();
34     }
35
36     @GetMapping(value = MONO_ERROR)
37     public Mono<Dto> readMonoError() {
38         return Mono.error(new BadRequestException("Error details"));
39     }
40
41     @GetMapping(value = FLUX)
42     public Flux<Dto> readFlux() {
43         return Flux.interval(Duration.ofMillis(100)).map(value -> new Dto(value.intValue())).take(5);
44     }
45
46     @GetMapping(value = FLUX_EMPTY)
47     public Flux<Dto> readFluxEmpty() {
48         return Flux.empty();
49     }
50
51     @GetMapping(value = FLUX_ERROR)
52     public Flux<Dto> readFluxError() {
53         return Flux.interval(Duration.ofMillis(100)).map(value -> new Dto(value.intValue())).take(2)
54             .concatWith(Mono.error(new ConflictException("flux error")));
55     }
56
57 }
```

Reactive API-Rest

```
WebFluxResource.java
48
49
50     @Test
51     void testReadFlux() {
52         this.webTestClient
53             .get().uri( $: WEB_FLUX + WebFluxResource.FLUX) capture of ?
54             .exchange() ResponseSpec
55             .expectStatus().isOk() ResponseSpec
56             .expectBodyList(Dto.class) ListBodySpec<Dto>
57             .value(response -> {
58                 assertEquals(expected: 5, response.size());
59                 assertEquals(new Integer(value: 0), response.get(0).getId());
60             })
61     }
62
63     @Test
64     void testReadFluxEmpty() {
65         this.webTestClient
66             .get().uri( $: WEB_FLUX + WebFluxResource.FLUX_EMPTY) capture of ?
67             .exchange() ResponseSpec
68             .expectStatus().isOk() ResponseSpec
69             .expectBodyList(Dto.class) ListBodySpec<Dto>
70             .value(response -> assertTrue(response.isEmpty()));
71     }
72
73     @Test
74     void testReadFluxError() {
75         this.webTestClient
76             .get().uri( $: WEB_FLUX + WebFluxResource.FLUX_ERROR)
77             .exchange()
78             .expectStatus().isEqualTo(HttpStatus.CONFLICT);
79     }
80 }
```

Spring Security

- Proyecto Spring Security
 - Spring Security se centra en proporcionar la autenticación y autorización para aplicaciones Java
- Dependencias

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

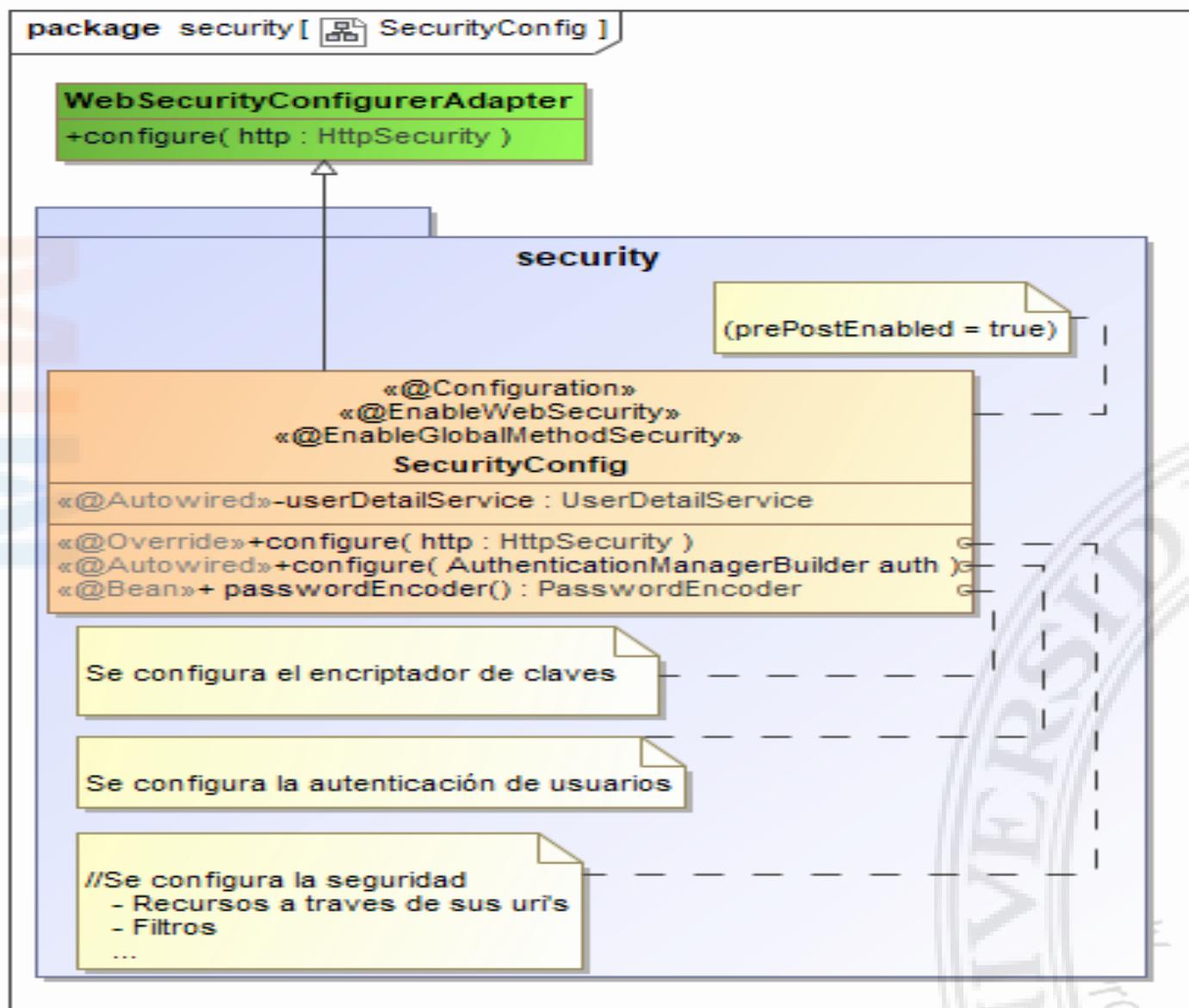
pom.xml

- Configuración
 - *SecurityConfig*: Configurar la seguridad de los recursos
 - *UserDetailsService*: autentica a los usuarios

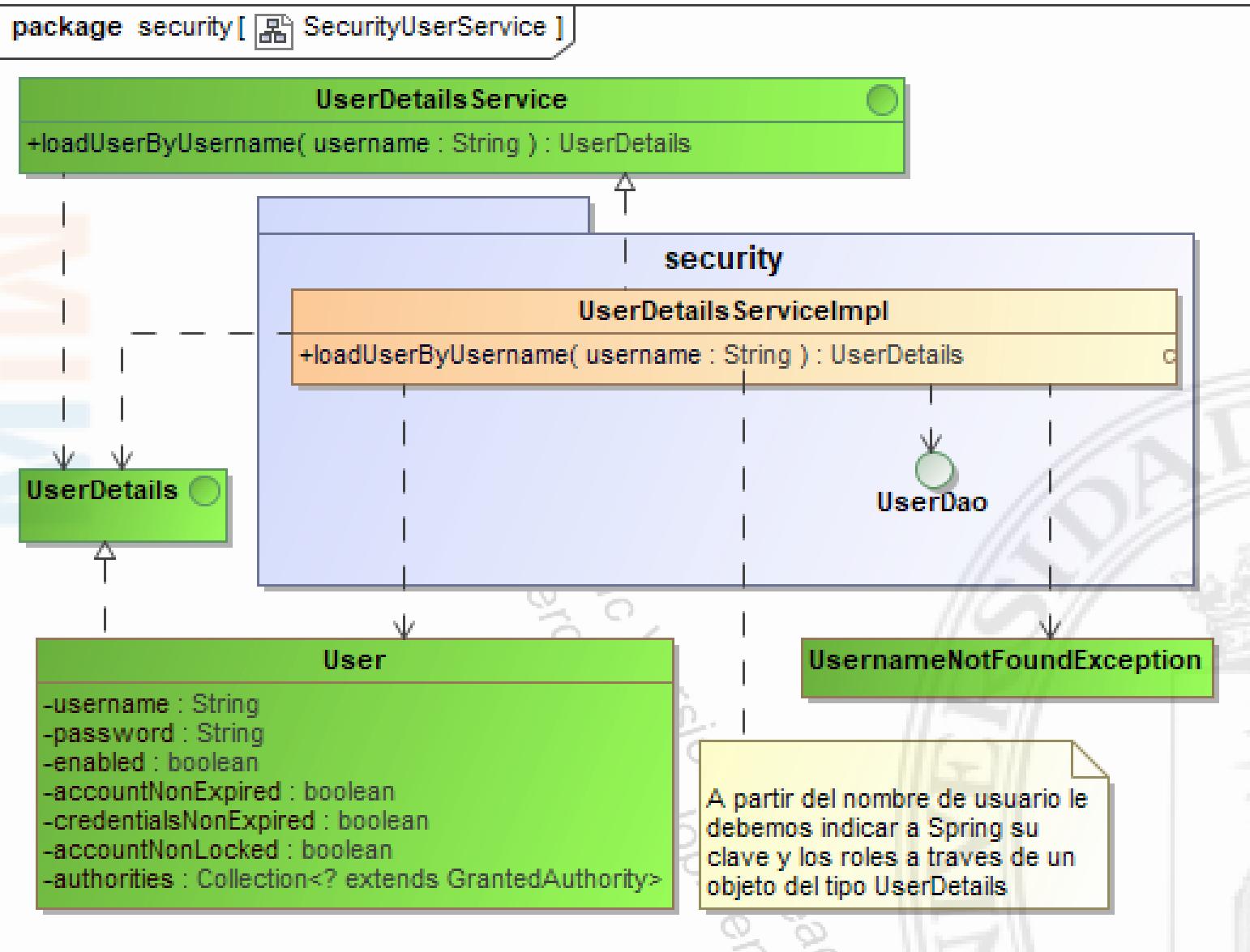
Spring Security

- **Rol.** Es un nombre abstracto para configurar el permiso de acceso de un conjunto de recursos de una aplicación
- **Usuario.** Es una identidad única reconocida por el servidor. Un usuario puede tener varios roles
- A los recursos se le debe configurar las diferentes operaciones asociado a los roles. Se puede definir en:
 - En el propio recurso, mediante anotaciones
 - En la configuración de seguridad, mediante las diferentes uri's
- Existen varias maneras de autenticar a un usuario
 - A través de memoria, lista definida estáticamente mediante Java
 - A través de servicio de autenticación personalizado
 - A través de JDBC
 - A través de LDAP
 - ...

Configuración



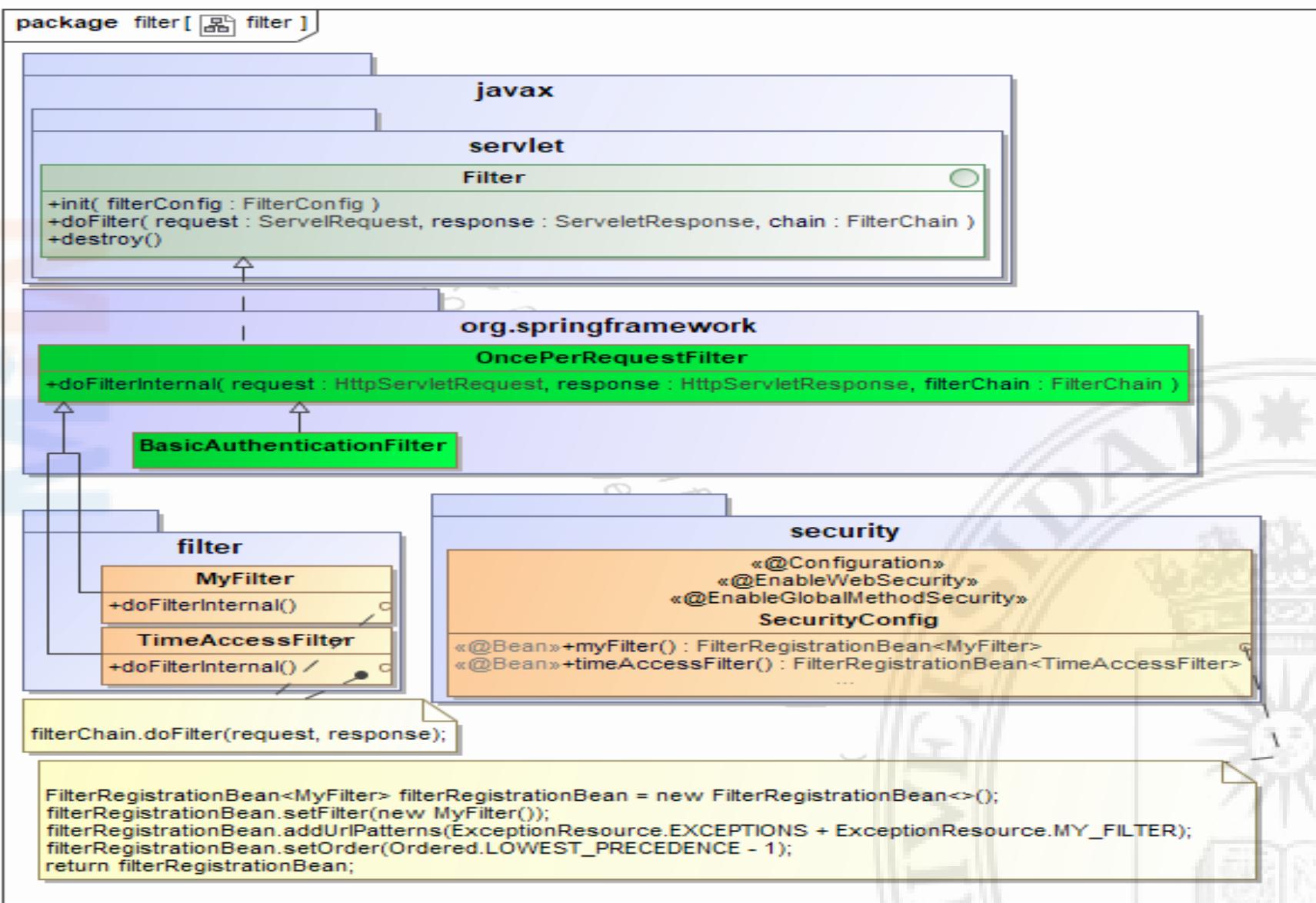
Servicio de Usuarios



✍ Ejercicios

- Analizar el ejemplo de seguridad del proyecto ejemplo de la asignatura
- Crear el recurso */security/securityAnnotation*, sólo podrá acceder un usuario “*u1*” con el rol de “*PLAYER*”, se configura con anotaciones
- Crear una entidad *User(id, name, role)* y el *UserDao*. Probar con el usuario “*u2*” que se encuentra en BD para realizar la autenticación

Filter



Filter

MyFilter.java

```
12 public class MyFilter extends OncePerRequestFilter {  
13     @Override  
14     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,  
15                                     FilterChain filterChain) throws ServletException, IOException {  
16         // pre process  
17         LogManager.getLogger(this.getClass().getName()).debug(">>> FILTER MyFilter...");  
18         // passing chain  
19         filterChain.doFilter(request, response);  
20         // post process  
21     }  
22 }
```

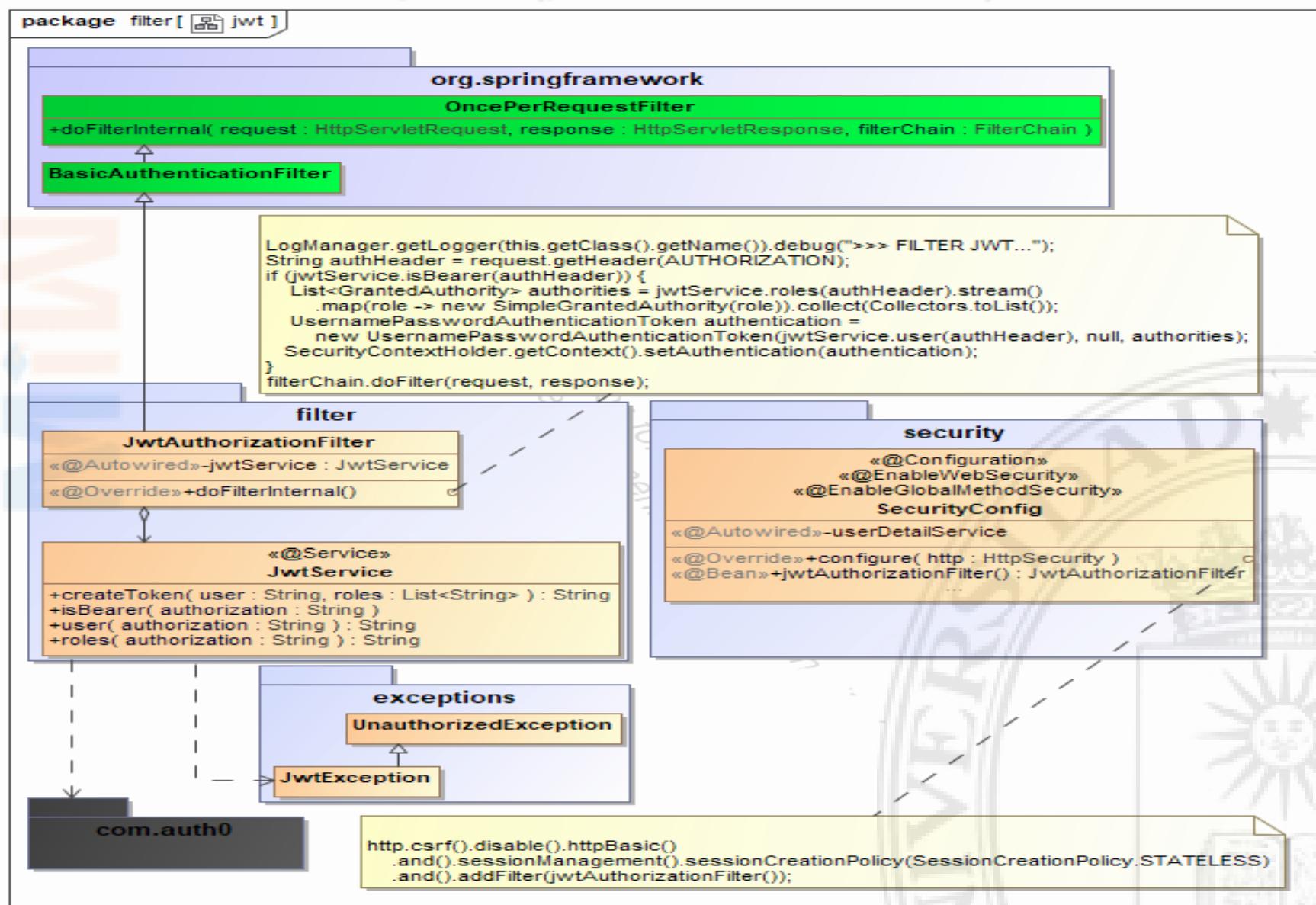
SecurityConfig.java

```
58 @Bean  
59     public FilterRegistrationBean<MyFilter> myFilter() {  
60         FilterRegistrationBean<MyFilter> filterRegistrationBean = new FilterRegistrationBean<>();  
61         filterRegistrationBean.setFilter(new MyFilter());  
62         filterRegistrationBean.addUrlPatterns(ExceptionResource.EXCEPTIONS + ExceptionResource.MY_FILTER);  
63         return filterRegistrationBean;  
64     }
```

JWT (*JSON Web Token*)

- **Servidor sin estado**
 - Es un estándar abierto basado en JSON propuesto por IETF (*RFC 7519*) para la creación de tokens de acceso
 - Documentación y servicios: <https://jwt.io>
 - Estructura
 - **Header:** '{"alg":"*algoritmo*","typ":"JWT"}'
 - **Payload:** privilegios del token
 - *Issuer:* proveedor
 - *Subject:* usuario
 - *Audience:* receptores del token
 - *Expiration time:* fecha de expiración
 - *Not before:* fecha de validación
 - *Issued at:* fecha de expedición
 - *JWT ID:* identificador
 - *Claims:* privilegios
 - **Signature:** firma
 - Token: *header*_{encodeBase64Url}'.'*payload*_{encodeBase64Url}'.'*signature*_{encodeBase64Url}
 - `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyYmYiOjE1NTA3NDMwNzAsInJvbGVzIjpbIlJPTEVfVVNFUiJdLCJpc3MiOiJtaXctc3ByaW5nNSIiImV4cCI6MTU1MDc0NjY3MCwiaWF0IjoxNTUwNzQzMDCwLCJ1c2VyIjoidXNlciJ9.I0AoZk52f1TziZ0AqbQWDnnUqR651tQJLdmpA_eJWXY`
 - Implementaciones: *auth0* (<https://auth0.com>)
 - Instalación en Java:
 - <dependency>
 - <groupId>com.auth0</groupId><artifactId>java-jwt</artifactId><version>3.4.1</version>
 - </dependency>
 - Instalación con npm:
 - `npm install @auth0/angular-jwt`

JWT (JSON Web Token)



JWT (JSON Web Token)

The screenshot shows a Java code editor with several tabs at the top: JwtResource.java (selected), JwtService.java, JwtAuthorizationFilter.java, and SecurityConfig.java. TheJwtResource.java tab contains the following code:

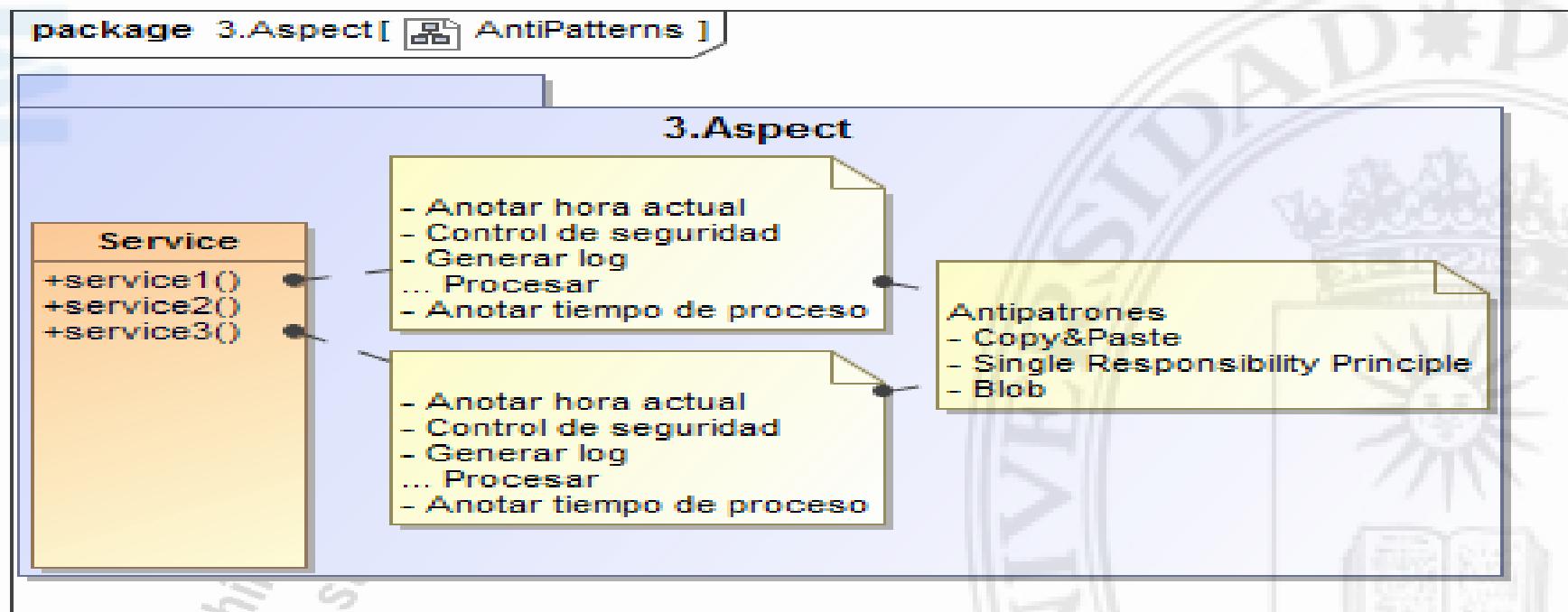
```
1 package es.upm.miw.restControllers.jwt;
2
3 import ...
4
5
6 @RestController
7 @RequestMapping(JwtResource.JWTS)
8 public class JwtResource {
9
10     public static final String JWTS = "/jwts";
11
12     public static final String TOKEN = "/token";
13
14     private JwtService jwtService;
15
16     @Autowired
17     public JwtResource(JwtService jwtService) {
18         this.jwtService = jwtService;
19     }
20
21     @PreAuthorize("authenticated")
22     @PostMapping(value = TOKEN)
23     public String login(@AuthenticationPrincipal User activeUser) {
24         List<String> roleList = activeUser.getAuthorities().stream().map(
25             GrantedAuthority::getAuthority).collect(Collectors.toList());
26         return jwtService.createToken(activeUser.getUsername(), roleList);
27     }
28
29     @PreAuthorize("hasRole('USER')")
30     @GetMapping
31     public String verify() {
32         return "OK. permitido JWT";
33     }
34 }
```

Perfiles Spring

- Configuración
 - @Profile("dev") @Service
 - @Profile("prod") @Config
 - El servicio se configura en el perfil indicado
- Propiedades
 - application.properties
 - **spring.profiles.active=dev**
 - application-dev.properties
 - application-prod.properties
 - **spring.data.mongodb.uri=\${MONGODB_URI}**
 - test.properties
- Cuando se crea la rama *release*, cambiar el *pom* quitando "SNAPSHOT" de la versión, cambiar a **spring.profiles.active=prod** en el fichero de propiedades
- Para desplegar en local:
 - **>mvn clean -Dspring.profiles.active=prod spring-boot:run**
 - **>java -jar -Dspring.profiles.active=prod *.jar**

Programación Orientada a Aspectos (AOP)

- La programación orientada a aspectos (AOP - *Aspect Oriented Programming*) es un paradigma de programación que intenta formalizar los elementos que son transversales (*cross-cutting*) a todo el sistema
- Como ejemplo, la AOP da solución a muchas funcionalidades: creación de registros (logs), control de seguridad, control de tiempos, trazabilidad de software...



Programación Orientada a Aspectos (AOP)

- Configuración (es parte del proyecto Spring Framework)

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
</dependency>
```

- Conceptos básicos

- Aspecto (*Aspect*). Funcionalidad modular (clase)
- Consejo (*Advice*). Es una acción que se debe realizar (método). Puede ser antes, después o alrededor de la ejecución de los métodos aconsejados
- Destinatario (*Target*) : clase aconsejada
- Puntos de corte (*Pointcut*): método aconsejado. Se indica donde de nuestro sistema se aplican los consejos. En Spring deben ser métodos de clase.

Puntos de Corte (*Pointcut*)

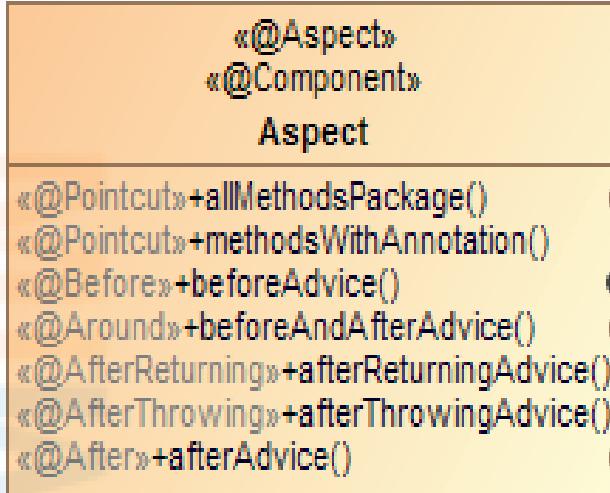
- Se indica donde de nuestro sistema se aplican los consejos. Para ello, se utilizan los siguientes elementos:
 - Nombres de paquetes, clases y métodos
 - Tipo de acceso (*public, protected...*)
 - Tipos de argumento. Se pone entre paréntesis
 - Comodines: "*" (cualquiera) y ".." (varios)
 - Se pueden combinar con operadores lógicos **&&**, **||** y **!**
 - Se puede definir un punto de corte para reutilizarlo
 - `@Pointcut("...") public void puntoDeCorte() {}`
- Ejemplos
 - "*execution(* *(..))*": todos los métodos
 - "*execution(public * *(..))*": todos los métodos públicos
 - "*execution(* package.*.*(..))*": todos los métodos de las clases de un paquete
 - "*within(package.*)(..)*": todos los métodos de las clases contenidas en un paquete
 - "*within(package..*)*": todos los métodos de las clases contenidas en un paquete y subpaquetes
 - "*execution(* suf*(..))*": todos los métodos que empiezan por "suf"
 - "*args(arg)*": todos los métodos que tienen un argumento del tipo indicado
 - "*@target(package.GenericAnnotation)*": todos los métodos cuya clase tiene una anotación
 - "*@annotation(package.MethodAnnotation)*": todos los métodos que tienen una anotación
 - "*execution(* package.method())*": un método concreto

Consejos

- Es una acción que se debe realizar
 - *@Before*. Se lanza antes de la ejecución del método
 - *@AfterReturning*. Se lanza después de la ejecución del método, si no provoca excepciones
 - *@AfterThrowing*. Se lanza después de la ejecución del método, si se genera una excepción
 - *@After*. Se lanza después de la ejecución del método, genere o no una excepción, es decir, al estilo del *finally*
 - *@Around*. Esta anotación ejecuta parte antes y parte después de la ejecución del método
- Se puede acceder a la información del punto de corte mediante el parámetro: *JoinPoint jp*

Ejemplos

package 3.aspect [ Aspect]



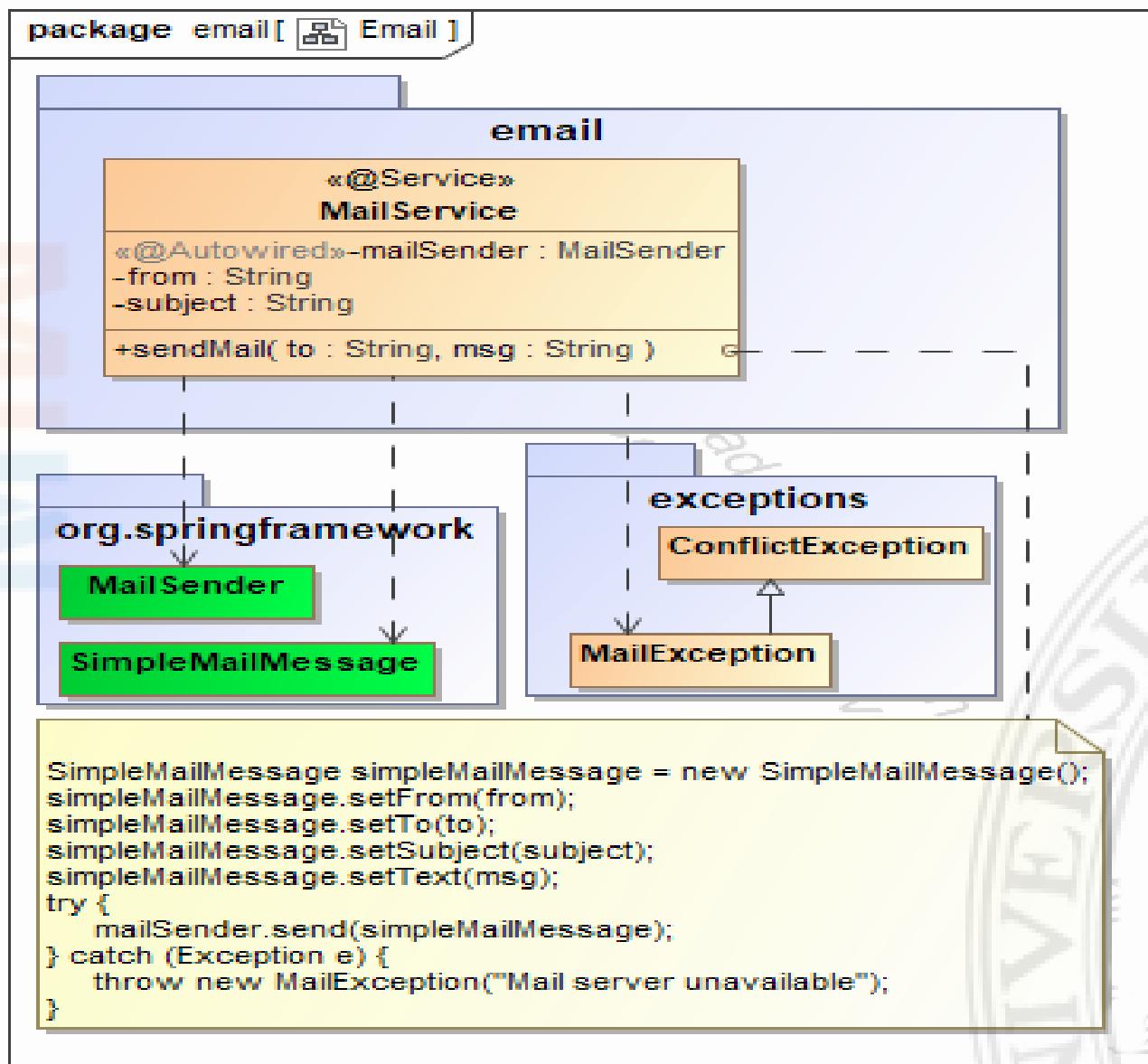
Un aspecto representa un conjunto de consejos

Se definen puntos de corte para reutilizarlos en diferentes consejos

Un consejo siempre debe referenciar un punto de corte

-  Ejemplo ApiLogs
-  Ejercicio Time
 - Generar un registro con el tiempo de ejecución de aquellos métodos que lleven la anotación personalizada @Time

Servicio email



Servicio email

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

pom.xml

```
# Email #####
# Se debe configurar la cuenta de gmail para soportar la conexion de la aplicacion
# Login to Gmail
# Access the URL as https://www.google.com/settings/security/lesssecureapps
# Select "Turn on"
spring.mail.defaultEncoding=UTF-8
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=${MAIL_USER}
spring.mail.password=${MAIL_PASS}
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.test-connection=false
```

application.properties

```
# Mail
spring.mail.username=notUser
spring.mail.password=notPass
```

test.properties

Servicio email

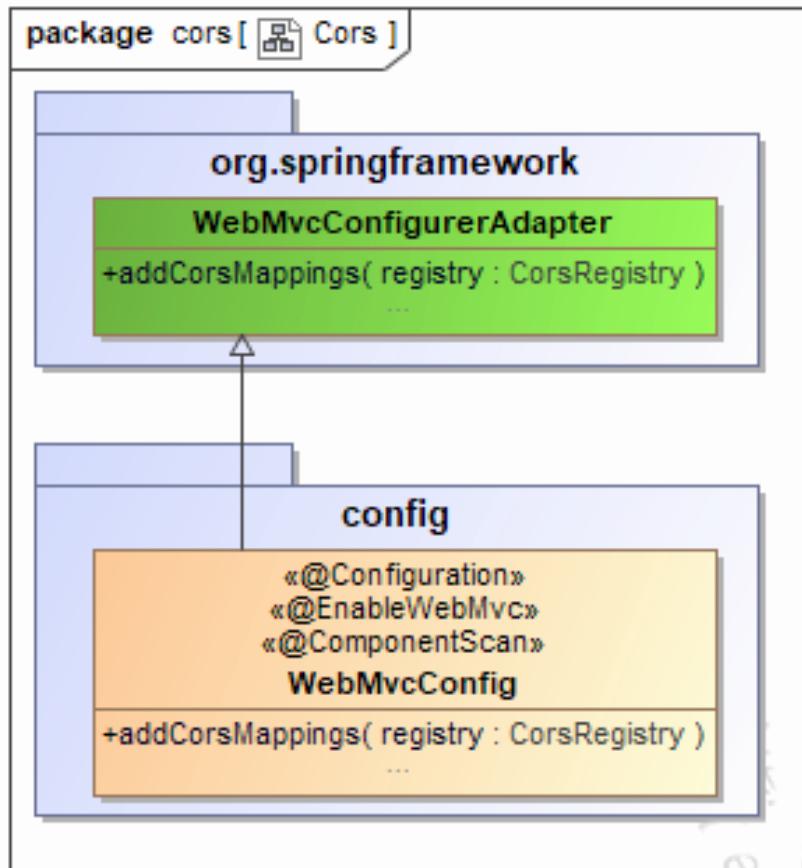
The screenshot shows a Java code editor with a file named `MailIT.java`. The code is a unit test for a mail service. It imports `... , MailSender, MailService` from `es.upm.miw.restControllers.mail`. The class `MailIT` is annotated with `@TestConfig` and contains a constructor with `@MockBean` annotations for `MailSender` and `MailService`. The `@Test` method `testSendMessage()` sends a test message to "test@gmai.com" and verifies the message content and recipient.

```
1 package es.upm.miw.restControllers.mail;
2
3 import ...
4
5 @TestConfig
6 public class MailIT {
7
8     @MockBean
9     private MailSender mailSender;
10
11     @Autowired
12     private MailService mailService;
13
14     @Test
15     public void testSendMessage() {
16         final ArgumentCaptor<SimpleMailMessage> captor = ArgumentCaptor.forClass(SimpleMailMessage.class);
17         this.mailService.send( to: "test@gmai.com", msg: "test message");
18         verify(mailSender).send(captor.capture());
19         assertEquals(mailService.getFrom(), captor.getValue().getFrom());
20         assertEquals( expected: "test@gmai.com", captor.getValue().getTo()[0]);
21         assertEquals(mailService.getSubject(), captor.getValue().getSubject());
22         assertEquals( expected: "test message", captor.getValue().getText());
23     }
24 }
```

CORS (Cross Origin Resource Sharing)

- Permite la compartición de recursos entre dominios.
- Referencia: <https://www.w3.org/TR/cors/>
- CSRF (Cross-site request forgery), es un ataque malicioso entre dominios cruzados. Consiste en enviar un script que intenta acceder a los dominios del resto de pestañas del navegador para obtener cookies y credenciales
- Por defecto, no se autorizan los accesos desde otros dominios
- La autorización de acceso al servidor desde otro dominio, se realiza a través del método: *Options*

CORS en Spring



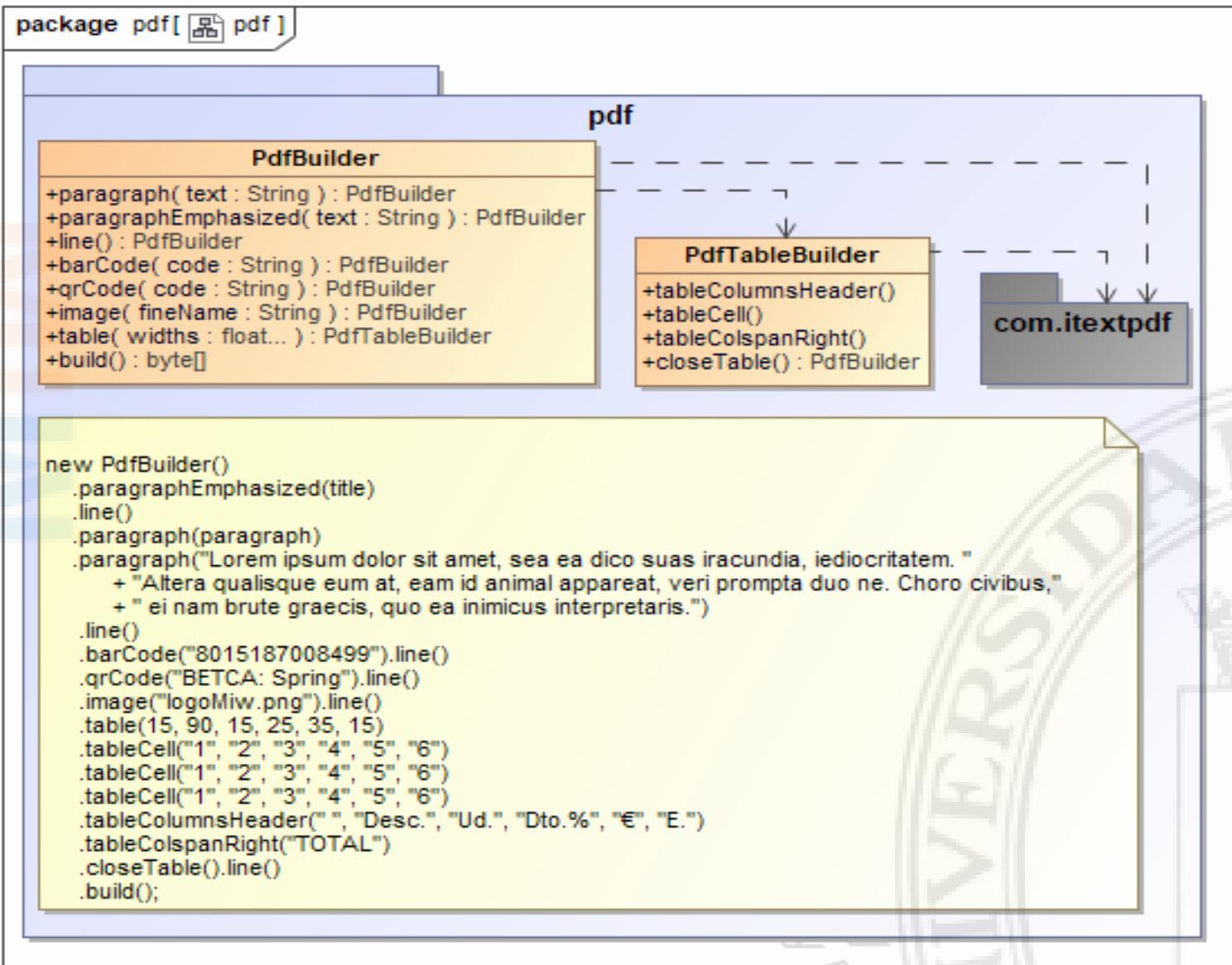
```
public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**").allowedMethods("*").allowedOrigins("*").maxAge(3600);
}
```

PDF's

- Se utiliza ITEXT: <https://itextpdf.com>

```
<!-- IText PDF ===== -->
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>kernel</artifactId>
    <version>7.0.2</version>
</dependency>
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>layout</artifactId>
    <version>7.0.2</version>
</dependency>
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>barcodes</artifactId>
    <version>7.0.2</version>
</dependency>
```

PDF's



PDF's

```
public static final String USER_HOME = "user.home";
public static final String ROOT_PDFS = "/spring/pdfs/file";
public static final String PDF_FILE_EXT = ".pdf";
```

BETCA: Spring 5. PDF

paragraph

Lorem ipsum dolor sit amet, sea ea dico suas iracundia, in has deserunt mediocritatem. Altera qualisque eum at, eam id animal appareat, veri prompta duo ne. Choro civibus ex vim, ei nam brute graecis, quo ea inimicus interpretaris.



Ref. BETCA: Spring

MIIW

	Desc.	Ud.	Dto. %	€	E.
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
TOTAL					

GraphQL

- GraphQL is a query language for APIs created by Facebook in 2012, open-sourced in 2015
- *graphql-java* is a Java library that implements the GraphQL specification. <http://graphql-java.com/>
- Integrating all these projects requires a lot of boilerplate code. Luckily, *graphql-java* is supported by Spring Boot with the use of the *graphql-spring-boot-starter* project.

Integración Continua con Github, Travis, Sonarcloud, Heroku y mLab

- Crear el proyecto
 - Bajarse la plantilla (con todas las dependencias)
 - Descomprimir en una carpeta llamada como el proyecto, y editar el *pom.xml* para cambiar el nombre del *artefacto*
 - Importar la carpeta desde *IntelliJ*
- Crear repositorio
 - Instalar *Git CLI*
 - Desde consola, crear el repositorio local, con las ramas *master* y *develop*
 - Crear el repositorio remoto en *Github* y conectarlo con repositorio local
- Conectar con *Travis-CI*
 - Configurar el fichero *.travis.yml*. Realizar *push* de la rama *develop* y comprobar que se ejecuta *Travis-CI*
- Conectar con *Sonarcloud*
 - Crear una cuenta en *Sonarcloud*
 - Obtener el *ApiKey* de *Sonarcloud*
 - Definir una variable de entorno (*SONAR-ApiKey*) asociada al proyecto en *Travis-CI*
 - Configurar *.travis.yml* con la conexión de *Sonar*. Realizar *push* en *develop* y comprobar que se dispara Sonar adecuadamente
- Conectar con *Heroku*
 - Definir en el proyecto un API con los correspondientes Tests
 - Configurar para *Swagger (Springfox)*
 - Realizar *push* en la rama *develop* y comprobar que todo es correcto
 - Instalar *Heroku CLI*, para poder ver logs
 - Crear cuenta en *Heroku*
 - Obtener el *ApiKey* de *Heroku*
 - Crear una variable de entorno (*HEROKU-ApiKey*) asociada al proyecto en *Travis-CI*
 - Configurar *.travis.yml* con el despliegue en *Heroku*. Realizar *push* en la rama master y comprobar que se despliega en *Heroku* adecuadamente
- Conectar con *MongoDB*
 - Añadir en *Heroku* un *add-ons* de *mLab*. Con ello se crea automáticamente una cuenta en *mLab* y se define una variable de entorno llamada *MONGODB_URI* con acceso a las BD
 - Configurar el fichero de propiedades con la *uri de BD*. Realizar *push* en la rama master y comprobar que se despliega en *Heroku* adecuadamente con acceso a BD

Aplicación Web: TPV

The screenshot shows a web browser window titled "BetcaTpvAngular" with the URL <https://betca-tpv-angular.herokuapp.com/welcome>. The page has a blue header with the "MiW" logo, the word "TPV" in the center, and a "Login" button on the right. Below the header, the text "Welcome to POS" is displayed. At the bottom of the page, there is a footer bar with the text "Universidad Politécnica de Madrid. Máster en Ingeniería Web API: <https://betca-tpv-spring.herokuapp.com/api/v0> Version: 1.3.0 (In Production)".

The screenshot shows a web browser window titled "BetcaTpvAngular" with the URL <https://betca-tpv-angular.herokuapp.com/home/cashier-opened>. The page has a blue header with the "MiW" logo, the word "TPV" in the center, and a timestamp "Feb 21, 2019, 9:08:27 PM" along with a "Logout" and "Profile" link. Below the header, the text "Shopping Cart €22.00" is displayed. A table lists items in the cart:

#	Description	Price	Nº	%	Total	Actions
1	Varios	12	<input type="button" value="-"/> 1 <input type="button" value="+"/>		12	<input checked="" type="checkbox"/> <input type="button" value="X"/>
2	84000000001	10	<input type="button" value="-"/> 1 <input type="button" value="+"/>	0	10	<input checked="" type="checkbox"/> <input type="button" value="X"/>

Below the table, there is a search bar labeled "Product Code" with a placeholder "Stock of 84000000001" and a search icon. At the bottom of the page, there are two buttons: "Checkout €" and "Budget".

At the very bottom of the page, there is a footer bar with the text "Universidad Politécnica de Madrid. Máster en Ingeniería Web API: <https://betca-tpv-spring.herokuapp.com/api/v0> Version: 1.3.0 (In Production)".