



POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MIW

04/03/2015 •

Back-end con Tecnologías de Código Abierto

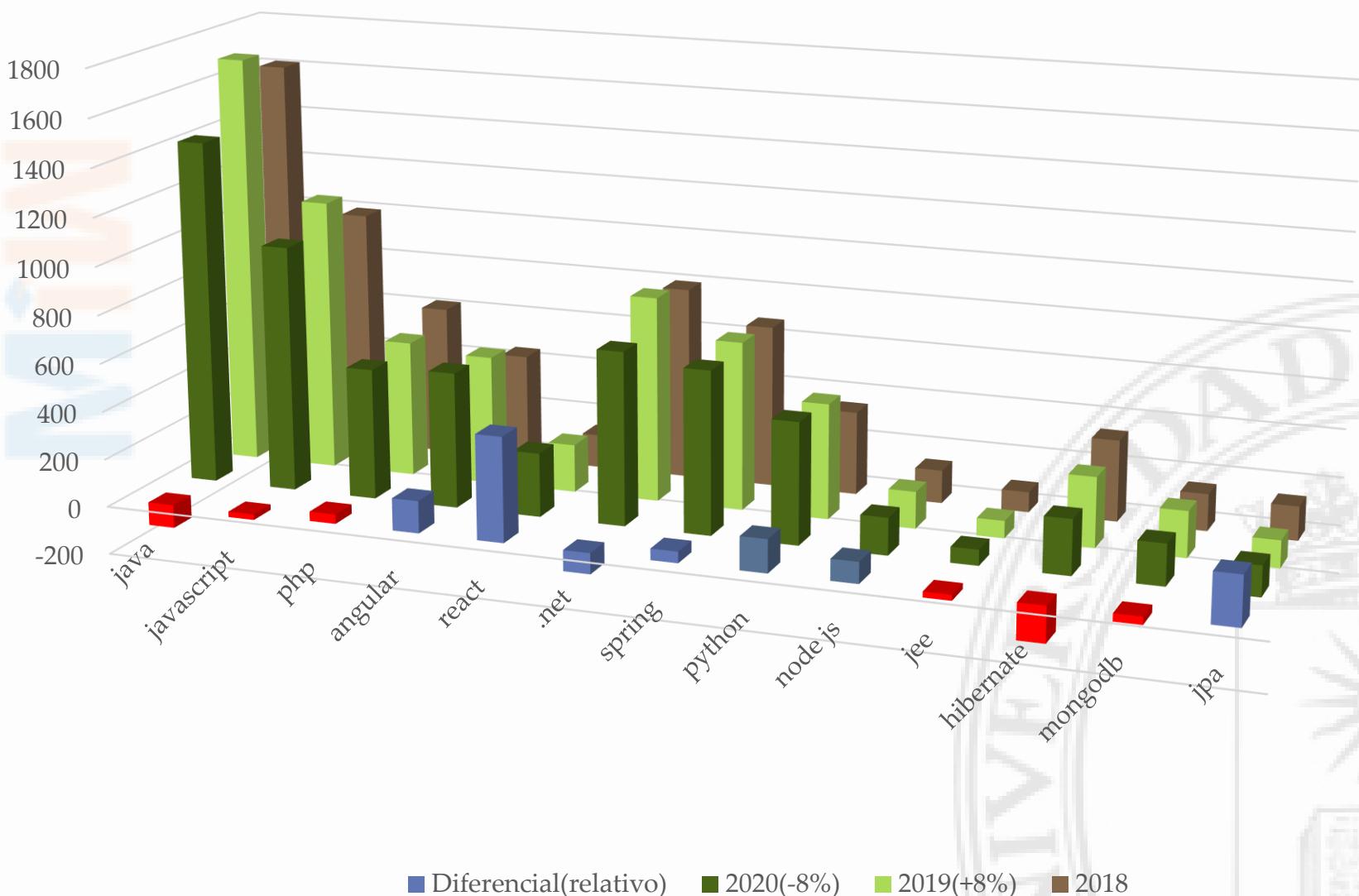
Spring

<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)

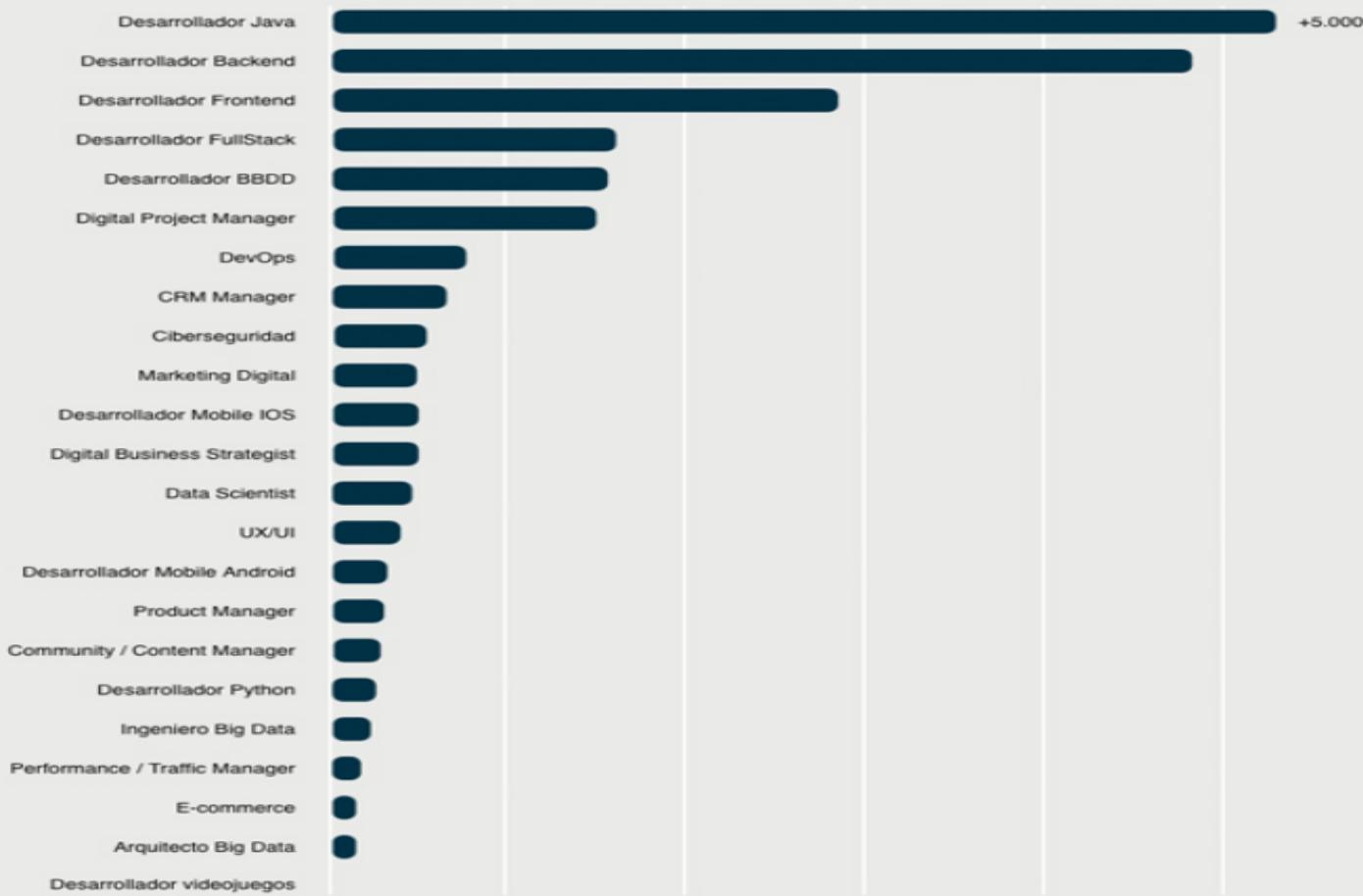
InfoJobs. Ofertas a 27/01/2020



InfoJobs. septiembre/2018

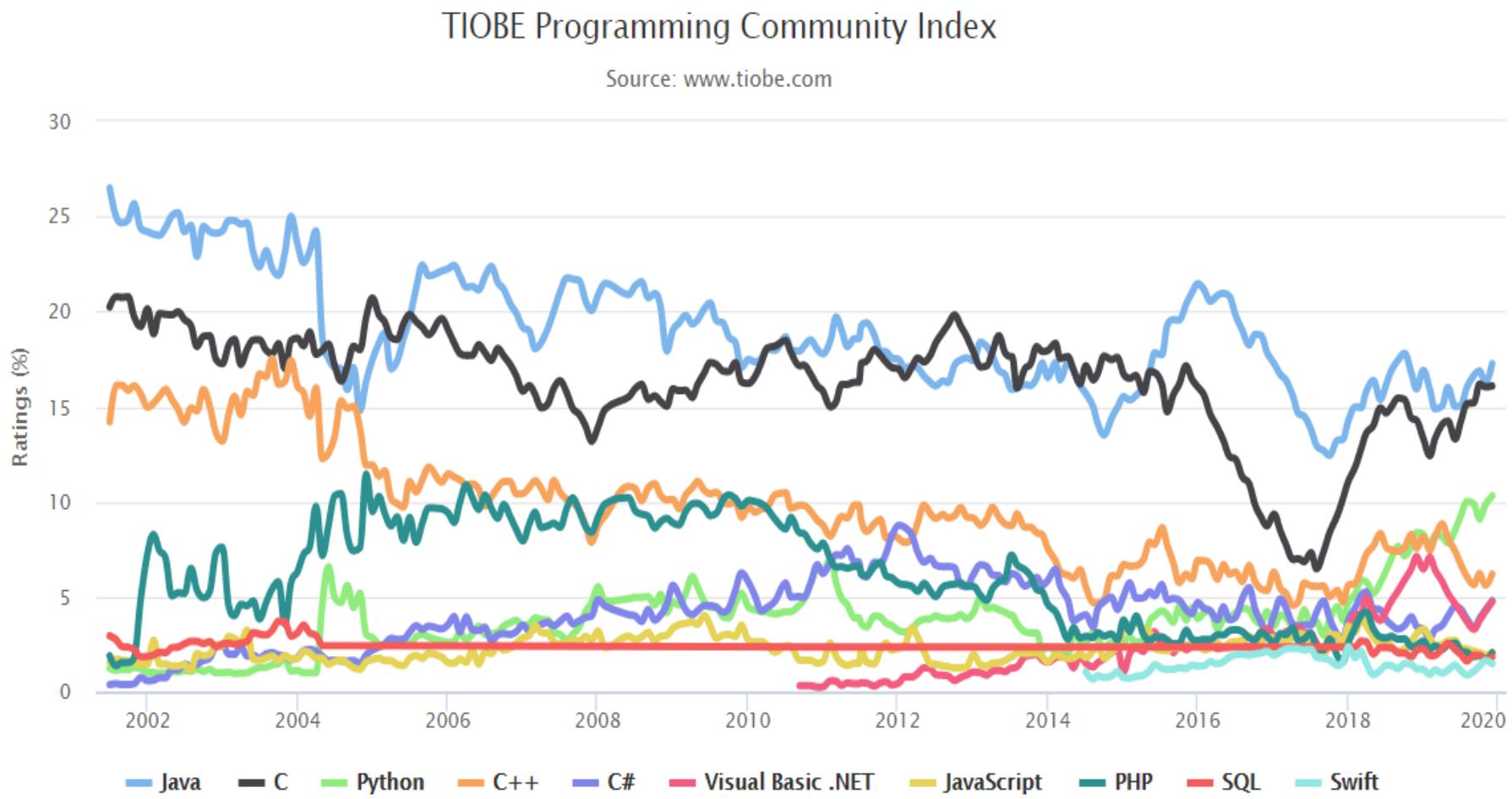
Los puestos más demandados en España

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

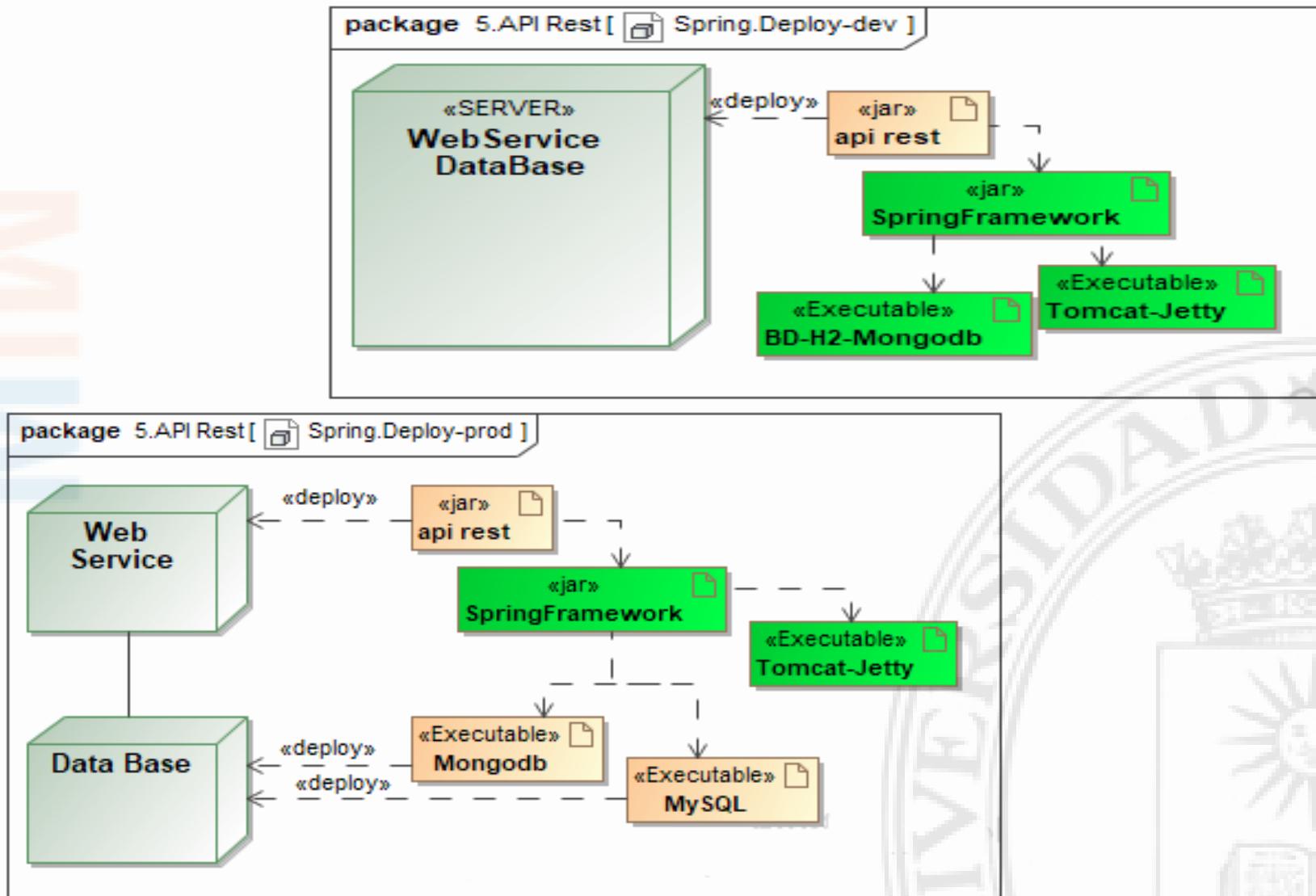


Tecnologías de servidor

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



Arquitectura Web



Java Enterprise Edition (JEE)

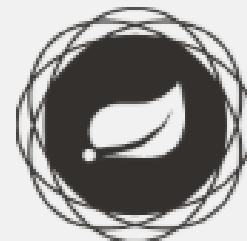
- Tecnología basada en Java
- Desarrollada por una coalición de empresas lideradas por *Oracle*, *IBM*, *Red Hat*, etc..
- <http://www.oracle.com/technetwork/java/javaee/overview/index.html>
- El objetivo de Java EE es simplificar el desarrollo y proporcionar una base común, independiente del fabricante, que permita concentrar los esfuerzos en el desarrollo del modelo de negocio
- Java EE es un conjunto de especificaciones (Java Specification Requests: JSR)
- Plataforma Java EE es un Servidor de Aplicaciones que implementa las especificaciones de JSR. Se basa en el desarrollo de Componentes Software
- Servidores de Aplicaciones Java EE
 - Grátuitos: Oracle (*GlassFish*), Apache (*TomEE*), Red Hat (*JBoss*) con asistencia profesional
 - Propietarias: IBM (*WebSphere*), Oracle (*WebLogic*)

Spring

- Spring es un framework ligero de código abierto, creado por *Rod Jhonson*, para facilitar el desarrollo de Aplicaciones Empresariales modernas mediante Java: <https://spring.io/>
- Está organizado por módulos independientes, y puede integrarse con otros frameworks.



SPRING BOOT



SPRING FRAMEWORK



SPRING DATA



SPRING SECURITY



SPRING BATCH



SPRING REST DOCS



SPRING CLOUD



SPRING SOCIAL



SPRING LDAP



SPRING HATEOAS



SPRING CLOUD DATA FLOW

Spring. Proyectos

■ Proyecto: *Spring-boot*

- *Spring Boot* facilita la creación de aplicaciones, gestionando las dependencias y embebiendo en el JAR final las aplicaciones complementarias necesarias (motor BD, ORM, servidor Web...)
- La mayoría de las aplicaciones *Spring Boot* necesitan muy poca configuración de Spring
- <https://projects.spring.io/spring-boot/>

■ Proyecto: *Spring-framework*

- *Spring Framework* proporciona un modelo completo de programación y configuración para aplicaciones empresariales modernas basadas en Java
- <http://projects.spring.io/spring-framework>

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
 - Código de Java
- Esto nos lleva a un cambio importante en la manera de desarrollar nuestro sistema software

Inyección de dependencias

- Definición de Bean:
 - Anotación: `@Component public class MiBean{...}`. Identificación del bean por defecto, nombre de la clase empezando en minúsculas. Estereotipos:
 - `@Controller`. Controlador en la capa web
 - `@Repository`. Persistencia
 - `@Service`. Servicio
 - XML: `<bean id="miBean" class="package.MiBean">`
- Alcance:
 - `@Scope("singleton")`: una sola instancia, por defecto
 - `@Scope("prototype")`: una instancia diferente para cada dependencia
- Inyección de un Bean:
 - `@Autowired`. Puede estar en constructores, en los **atributos** o en el método `set`
 - `@Qualifier("type-bean")` Califica el *bean* de un tipo
- Ciclo de vida
 - **>>>Ojo! En el constructor no se han resuelto las dependencias<<<**
 - `@PostConstruct`
 - `@PreDestroy`

Recursos

- Acceso a fichero de recursos para configuración de la aplicación
 - Tipos: *properties, yaml, json...*
 - Localización
 - `@PropertySource("classpath:application.properties")` **Valor por defecto**
 - `@PropertySource("file:///C:/Temp/application.properties")`. En la ruta indicada
 - `@PropertySource("http://server/application.properties")`. En la URL indicada
 - Acceso. Existen dos opciones:
 - `@Value("${miw.name}") private String name;`
 - Se debe pedir la inyección del bean *environment*
 - `@Autowired private Environment environment;`
 - Para consultar una propiedad: `environment.getProperty("miw.name");`

✍ Empezando en Spring Boot: configuración

The screenshot shows the Spring Initializr web interface. At the top, it says "Generate a Maven Project with Java and Spring Boot 2.0.2". In the "Project Metadata" section, "Group" is set to "es.upm.miw" and "Artifact" is set to "demo". In the "Dependencies" section, under "Selected Dependencies", the following starters are selected: Web, Reactive Web, JPA, MongoDB, Reactive MongoDB, Security, Aspects, and Mail. A large green button at the bottom left says "Generate Project alt + d".

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.2

Project Metadata

Artifact coordinates

Group

es.upm.miw

Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web × Reactive Web × JPA × MongoDB ×
Reactive MongoDB × Security × Aspects × Mail ×

Generate Project alt + d

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializr and Pivotal Web Services

✍ Empezando en Spring Boot: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

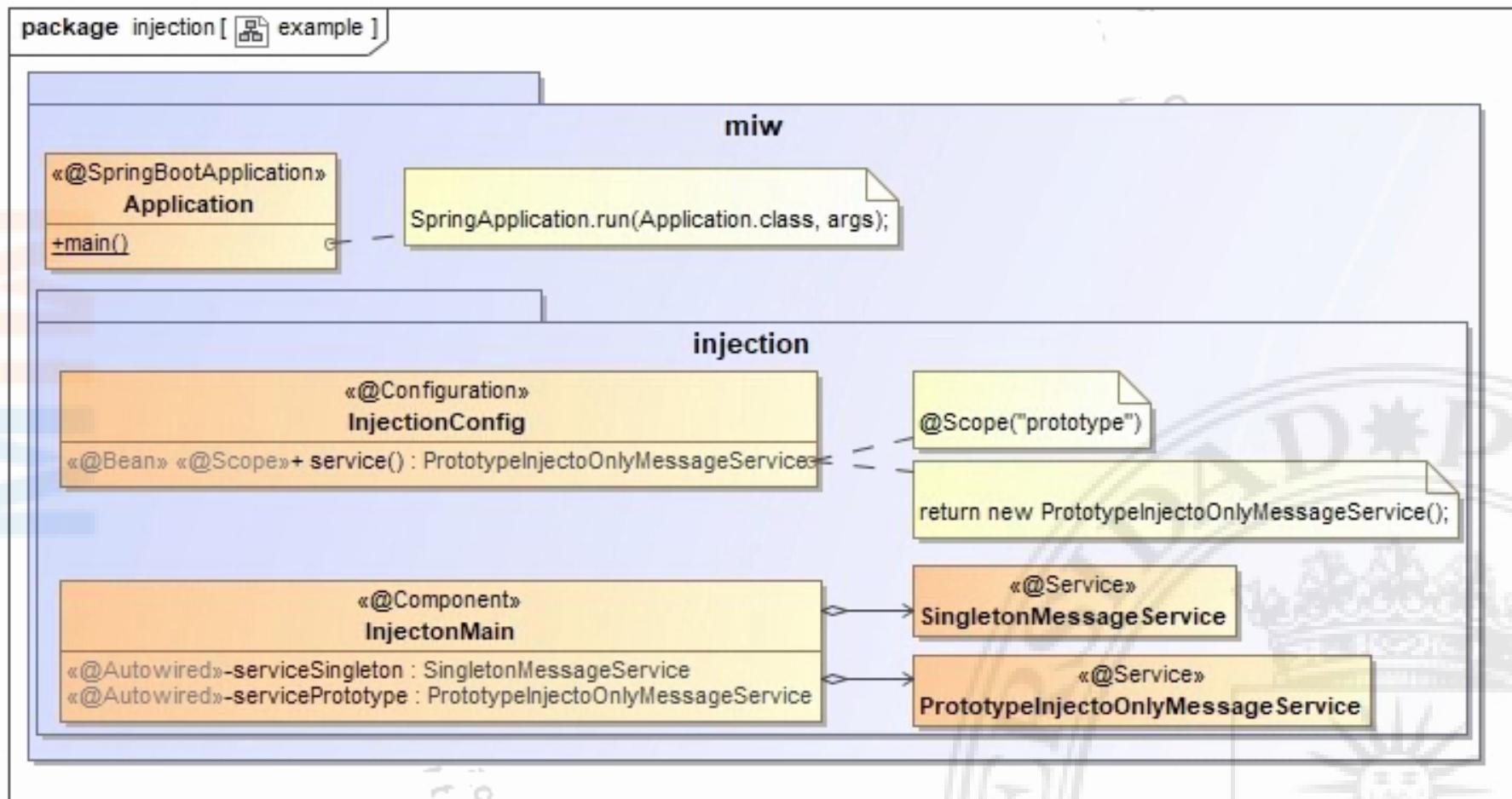
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

versión

artefacto

dependencias

✍ Empezando en Spring



✍ Empezando en Spring: código fuente

- Paquete: *miw*
 - Arranque de Spring: *Application*
 - Consola: `>mvn clean spring-boot:run`

- Paquete: *miw.injection*
 - Bean: *SingletonMessageService* (*se crea y se gestiona por Spring*)
 - Bean: *PrototypeInjectionOnlyMessageService* (*solo se gestiona por Spring*)
 - Bean: *InjectionMain*

 - Configuración: *InjectionConfig*. Se crea el bean a través de un método propio

Test

- Test Unitarios
 - Se pueden realizar mediante *JUnit*, sin inyección de Spring
 - Ejemplo: *SingletonMessageServiceWithoutInjectionTest*
- Test Unitarios con inyección de Spring
 - Se debe incluir las siguientes anotaciones en la clase de Test
 - `@ExtendWith(SpringExtension.class)`
 - `@SpringBootTest`
 - `@TestPropertySource(locations = "classpath:test.properties")`
 - Se agrupan las tres anotaciones en una propia desarrollada en el máster: `@TestConfig`
 - Ejemplo: *SingletonMessageServiceInjectionTest*
- Test Unitarios, inyección de mocks
 - Mock. Es un objeto que imita el comportamiento de otro objeto de una forma controlada y simplificada
 - Se utiliza la inyección de Spring para instanciar la clase de prueba e injectarle el mock
 - Ejemplo: *MainInjectonMessageServiceMockTest*
- Test Integración
 - Se utiliza la inyección de Spring para instanciar la clase de prueba y todos sus beans dependientes
 - Ejemplo: *MainInjectonIT*
-  Análisis de ejemplo de prueba de inyecciones

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)

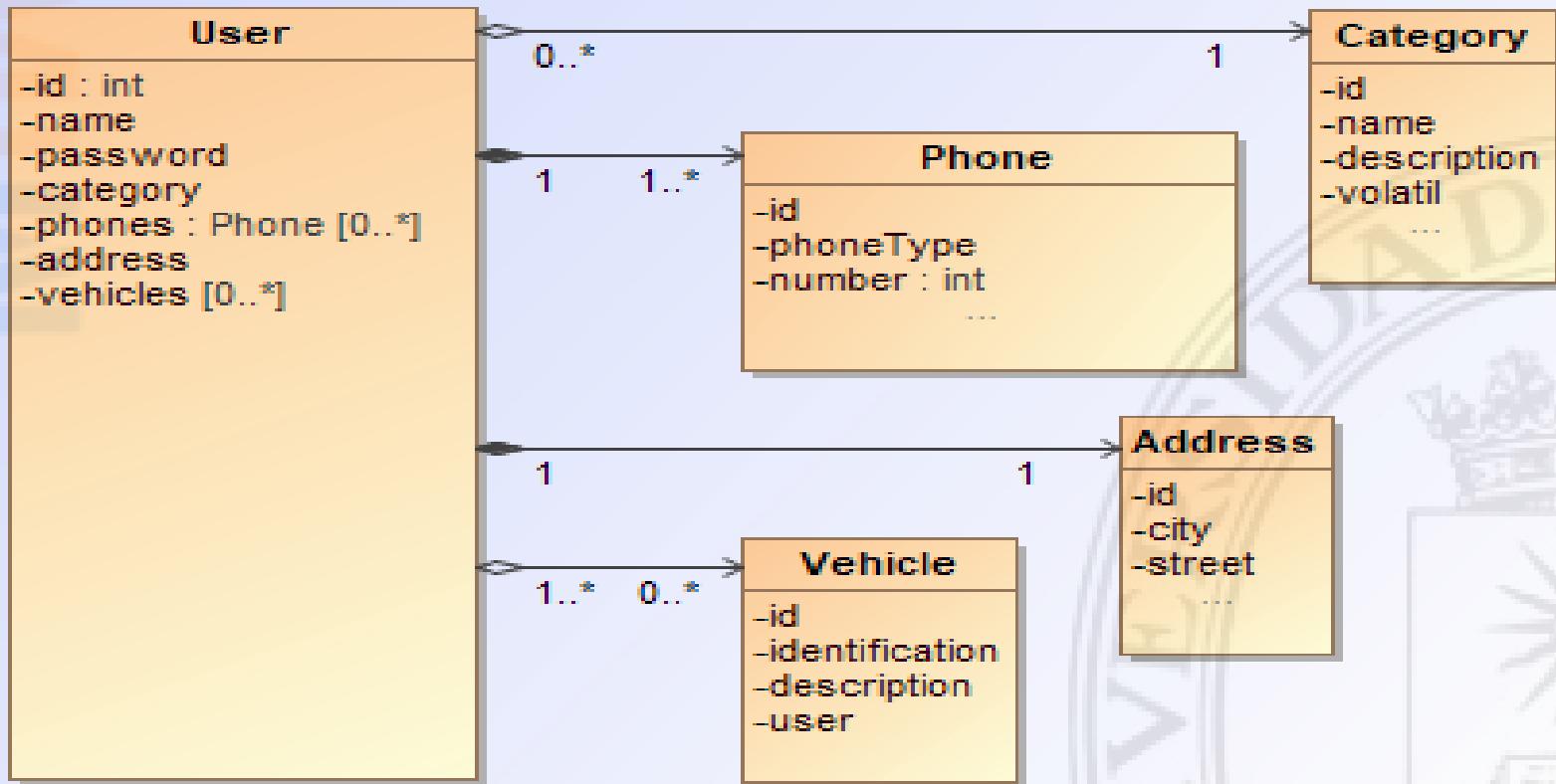
- Data Transfer Object (DTO), también llamado *Entity* (BD relacionales) o *Document* (NoSQL). Son los objetos utilizados para la comunicación con la capa de *persistencia*
- DAOs (*Repository*). Son las clases que se encargan de convertir los datos almacenados de forma persistente en objetos Java o 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

Entidad

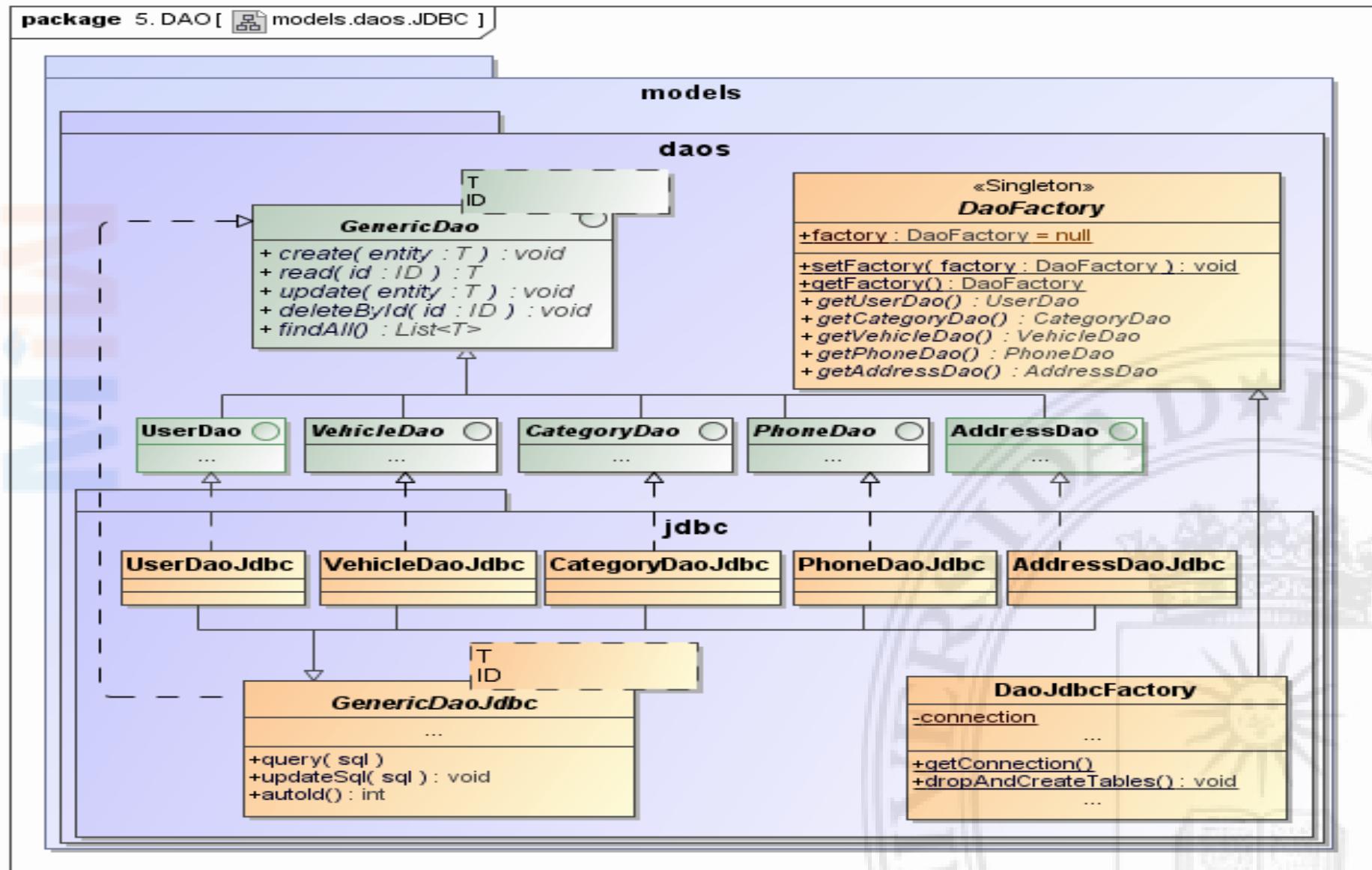
package DAO [ models.entities]

models

entities



DAO



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

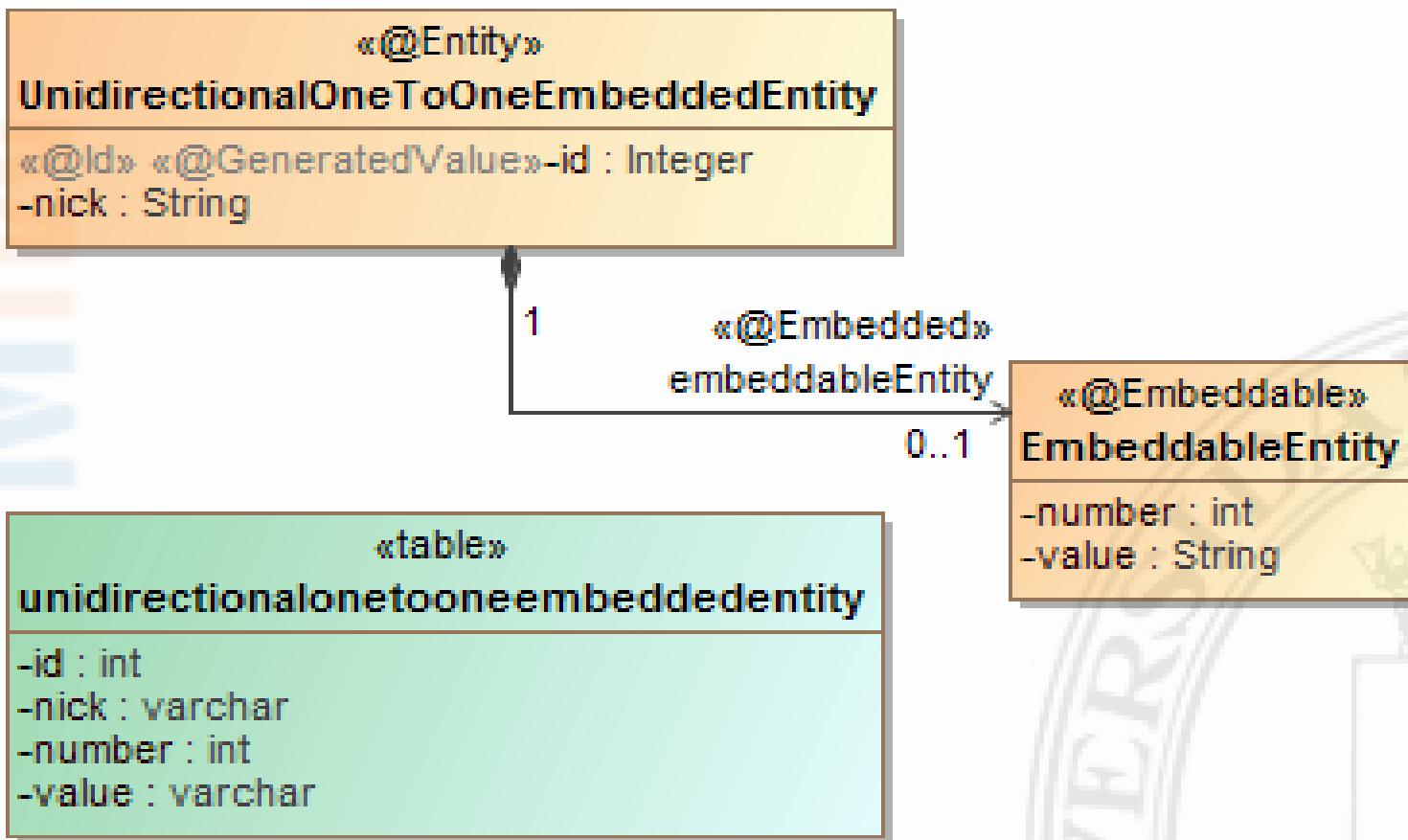
- `@Entity, @Id`
- `@IdClass(ClaveCompuestaId.class)`
- `@Transient`
- `@Entity(name=""). Nombre de la entidad/tabla`
- `@GeneratedValue(strategy=?). Generación de claves`
 - `GenerationType.AUTO` (se elige una estrategia de forma automática),
`GenerationType.IDENTITY` (se utiliza un contador autoincremental gestionado por la BD)
- `@Basic`
 - `optional` (boolean, si puede ser null)
 - `fetch` (`FetchType.LAZY`: tardía, `FetchType.EAGER`: temprana)
- `@Column. Propiedades de la anotación`
 - `name` (nombre de la columna), `unique` (valor único), `nullable` (permitir nulos), `length` (longitud para String)
- `@Enumerated(EnumType.STRING)`
- `@Temporal(TemporalType.DATE)` sólo fecha, `TemporalType.TIMESTAMP` Fecha y hora
- `@Lob.` String largo
- `@ElementCollection.` Se utiliza para `List<>`
-  **Ejemplo UnRelatedEntity**

Relaciones

- *OneToOne*. Cada entidad se relaciona con una sola instancia de otra entidad. Por ejemplo, los datos básicos de un Usuario y datos detallados del usuario
 - @OneToOne(param1="valor",param2="valor2",...)
- *OneToMany* . Una entidad, puede estar relacionada con varias instancias de otras entidades. Por ejemplo, una Cesta de la Compra tiene varios Productos comprados
- *ManyToOne* . Múltiples instancias de una entidad pueden estar relacionadas con una sola instancia de otra entidad. Varios Productos comprados están asociados con la misma Cesta
- *ManyToMany* . En este caso varias instancias de una entidad pueden relacionarse con múltiples instancias de otras entidades. Por ejemplo, cada asignatura tiene muchos estudiantes, y cada estudiante puede tener varias asignaturas
- Parámetros comunes de las relaciones
 - cascade. Tipo de operaciones en cascada a aplicar:
 - CascadeType.ALL (Todas las operaciones), CascadeType.PERSIST (Operaciones de persistencia), CascadeType.REMOVE (Operaciones de borrado)
 - optional. Si la asociación es opcional
- *@JoinColumn*. Se establece como una clave foránea en otra entidad.
 - name. Nombre de la columna

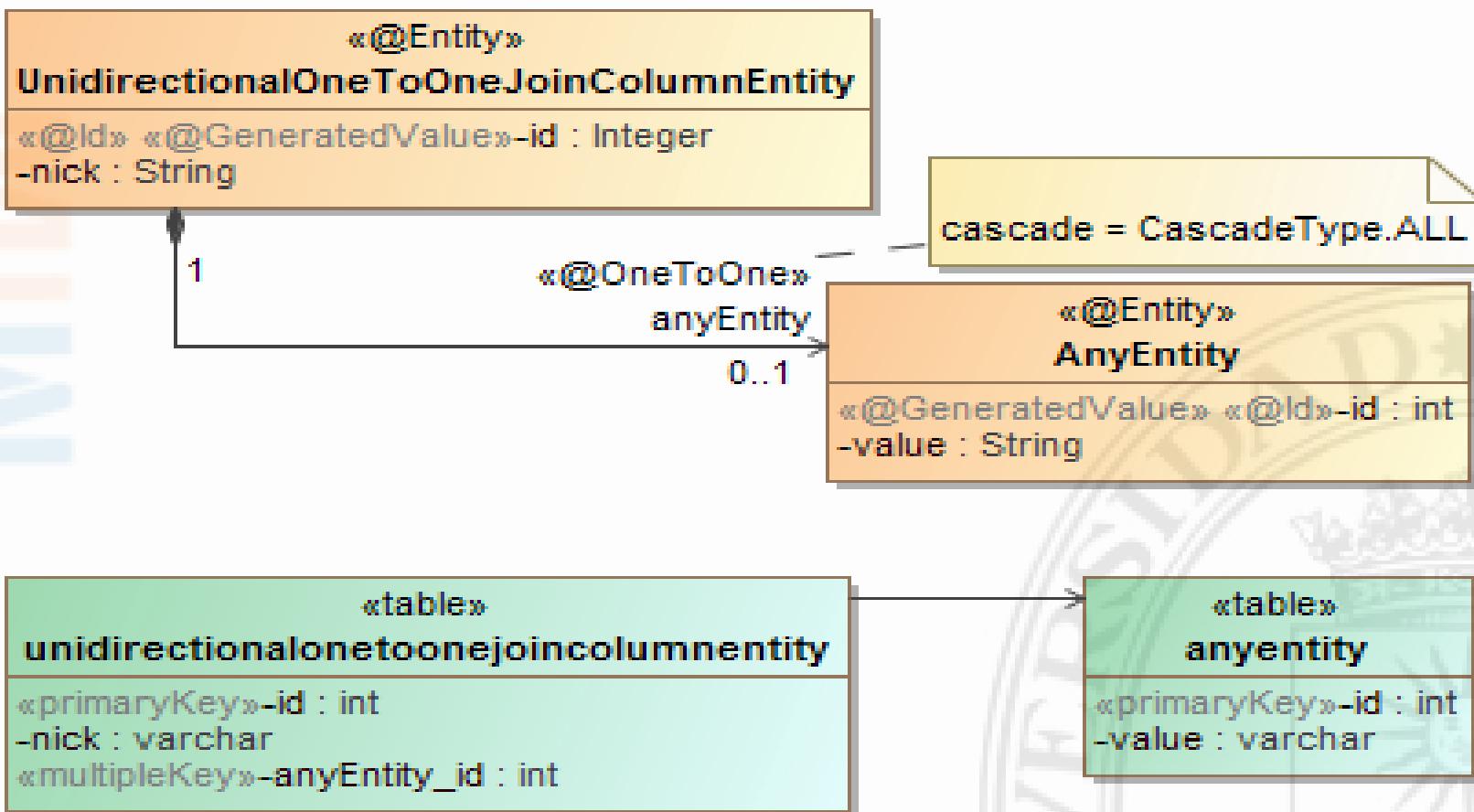
Unidirectional One To One Embedded

```
package 2.Persistence [  JPA.UnidirectionalOneToOneEmbeddedEntity ]
```



Unidirectional One To One Join Column

```
package 2.Persistence [  JPA.UnidirectionalOneToOneJoinColumnEntity ]
```



Unidirectional One To Many Embedded

```
package 2.Persistence [  JPA.UnidirectionalOneToManyEmbeddedEntity ]
```



1

anyClassArray

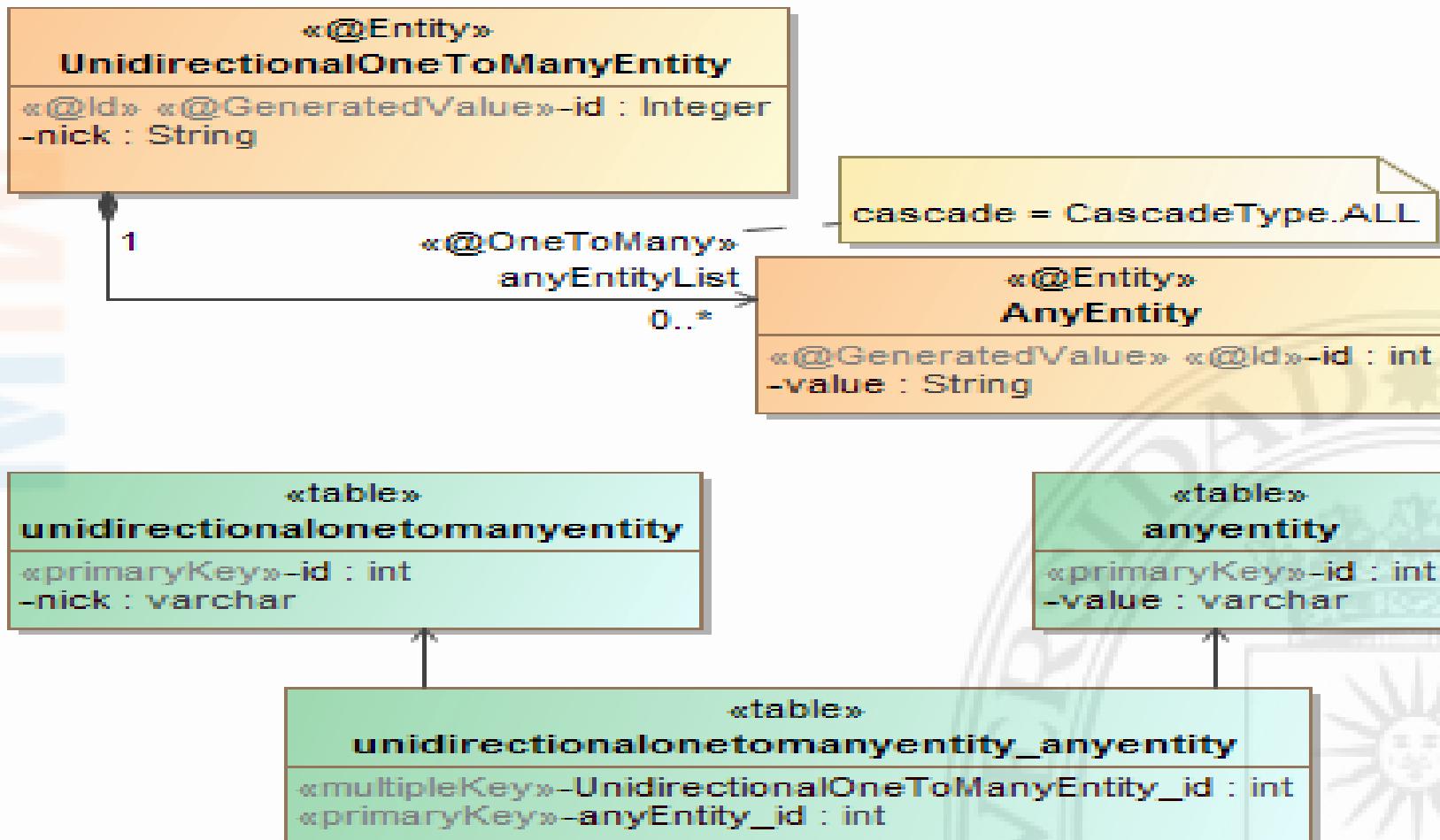
AnyClass

Serializable



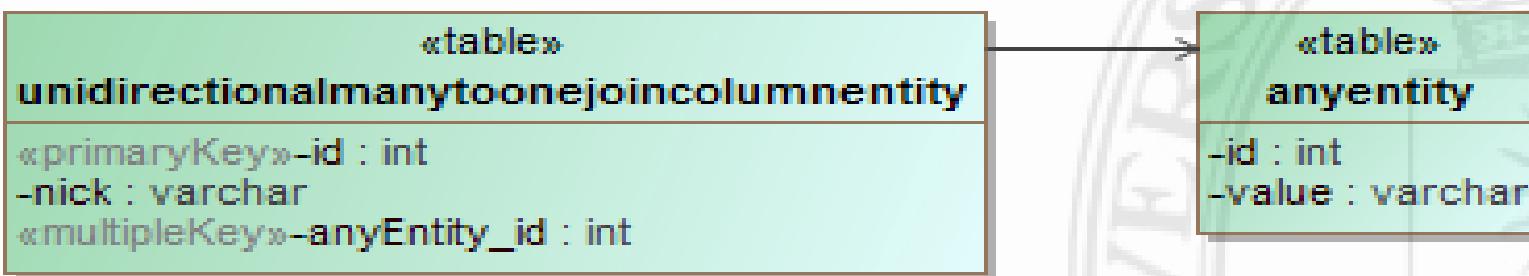
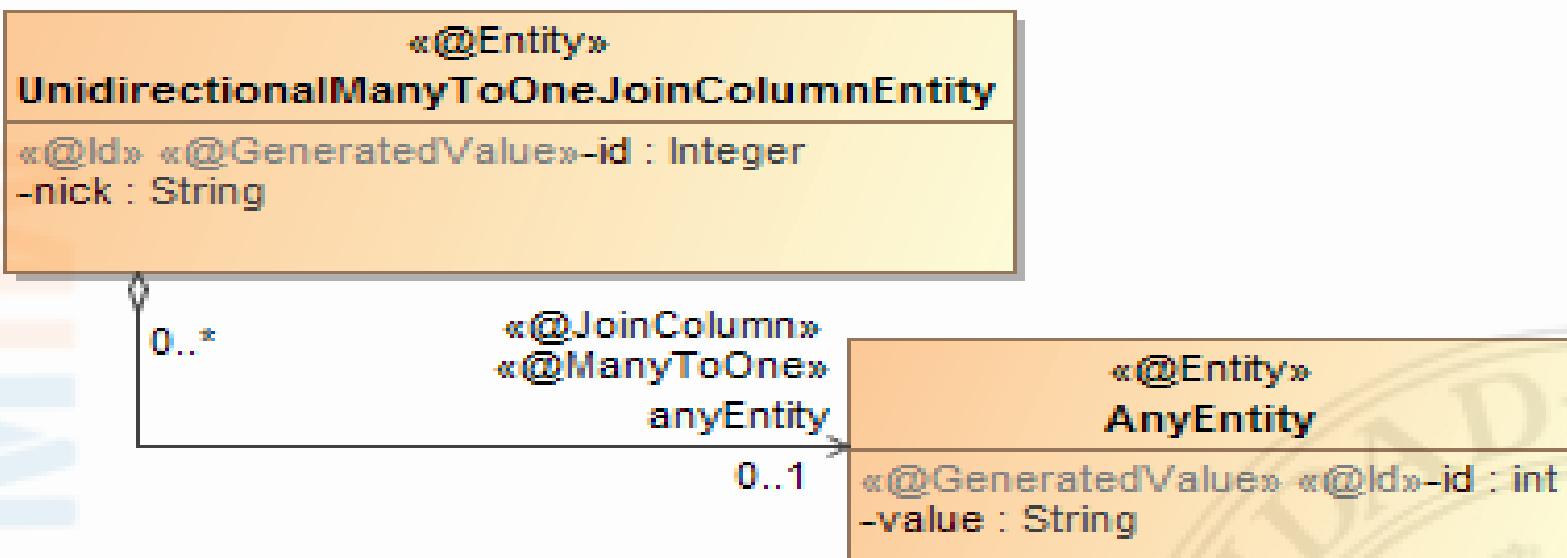
Unidirectional One To Many

```
package 2.Persistence [ JPA.UnidirectionalOneToManyEntity ]
```



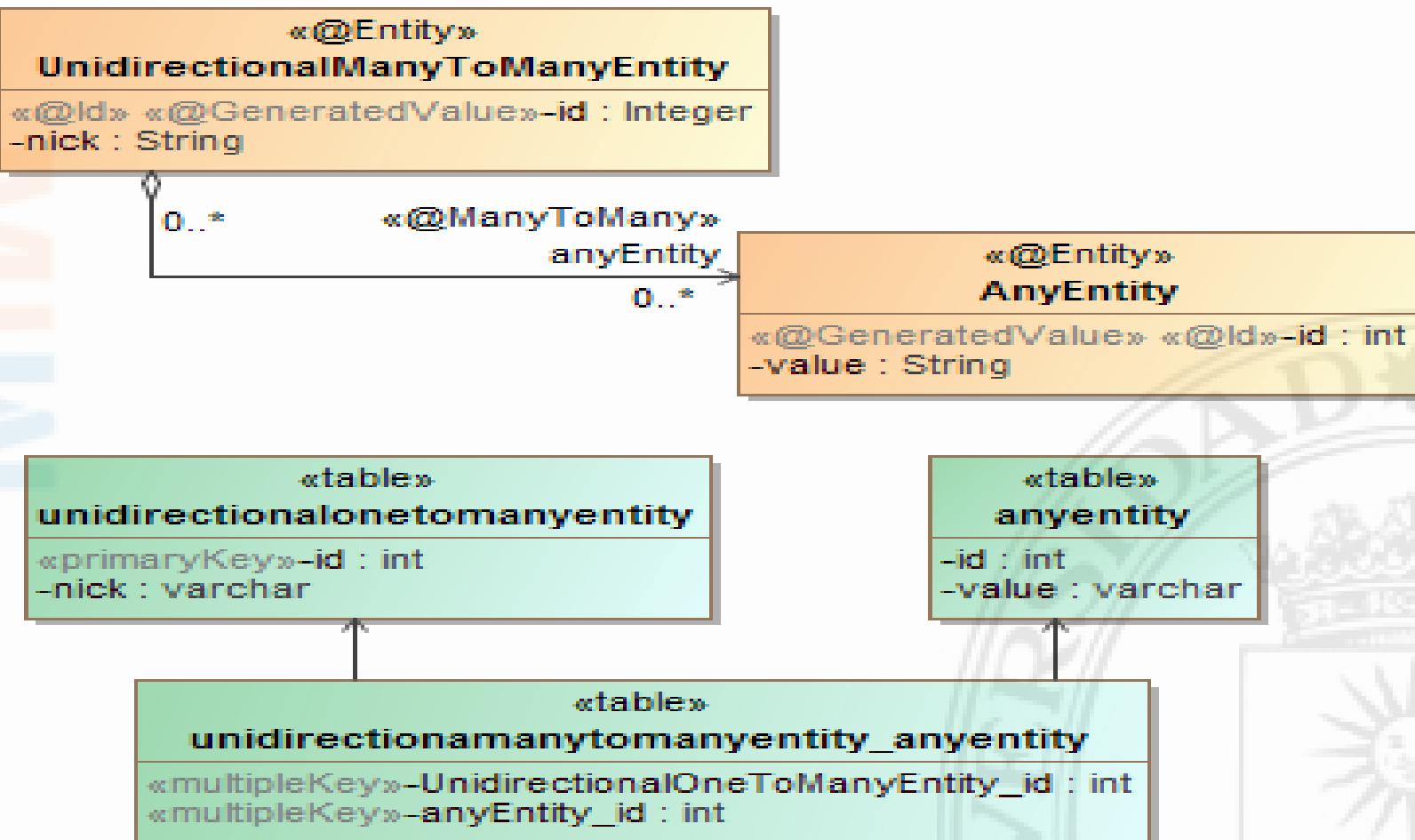
Unidirectional Many To One Join Column

```
package 2.Persistence [  UnidirectionalManyToOneJoinColumnEntity ]
```

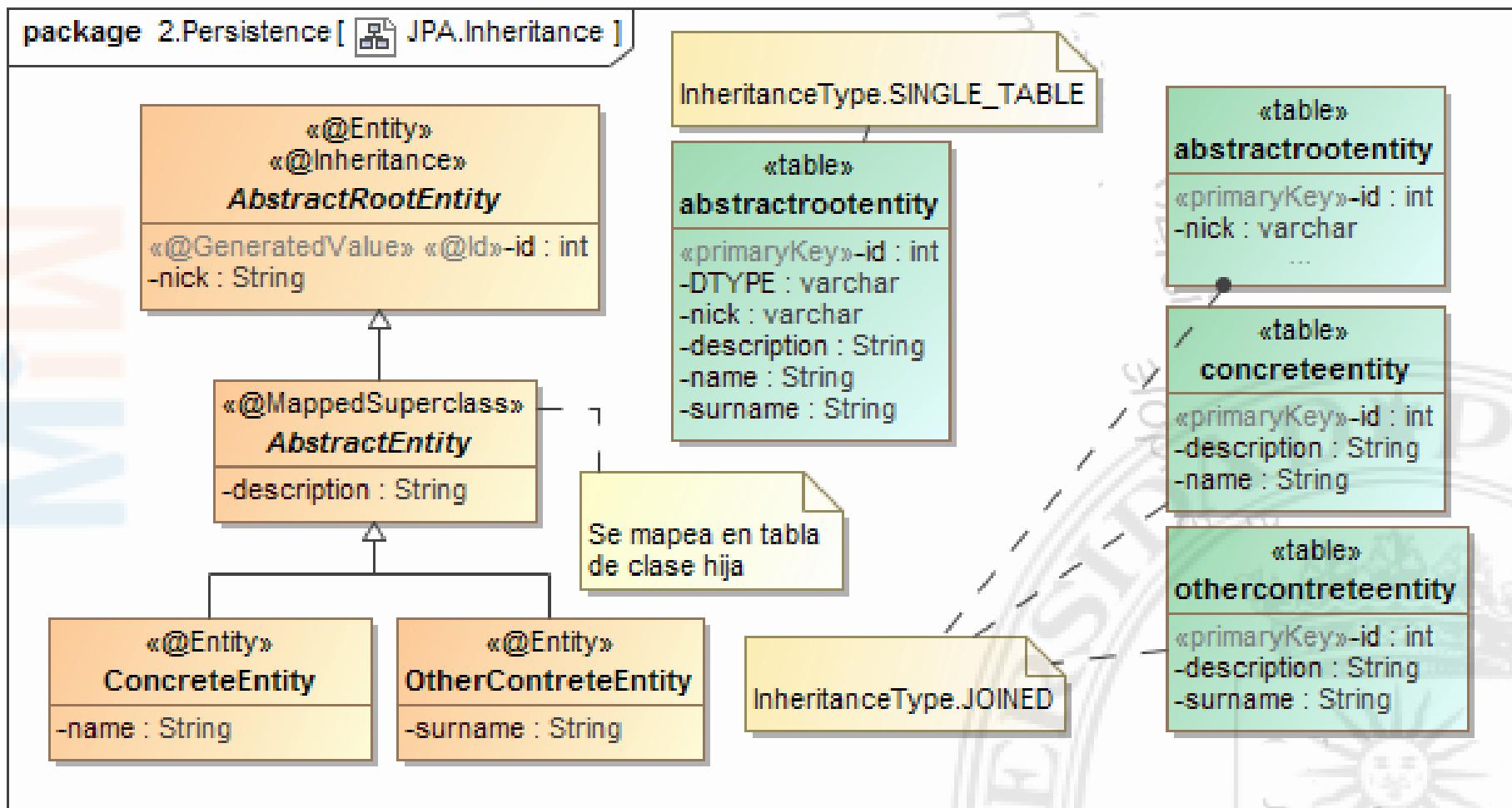


Unidirectional Many To Many

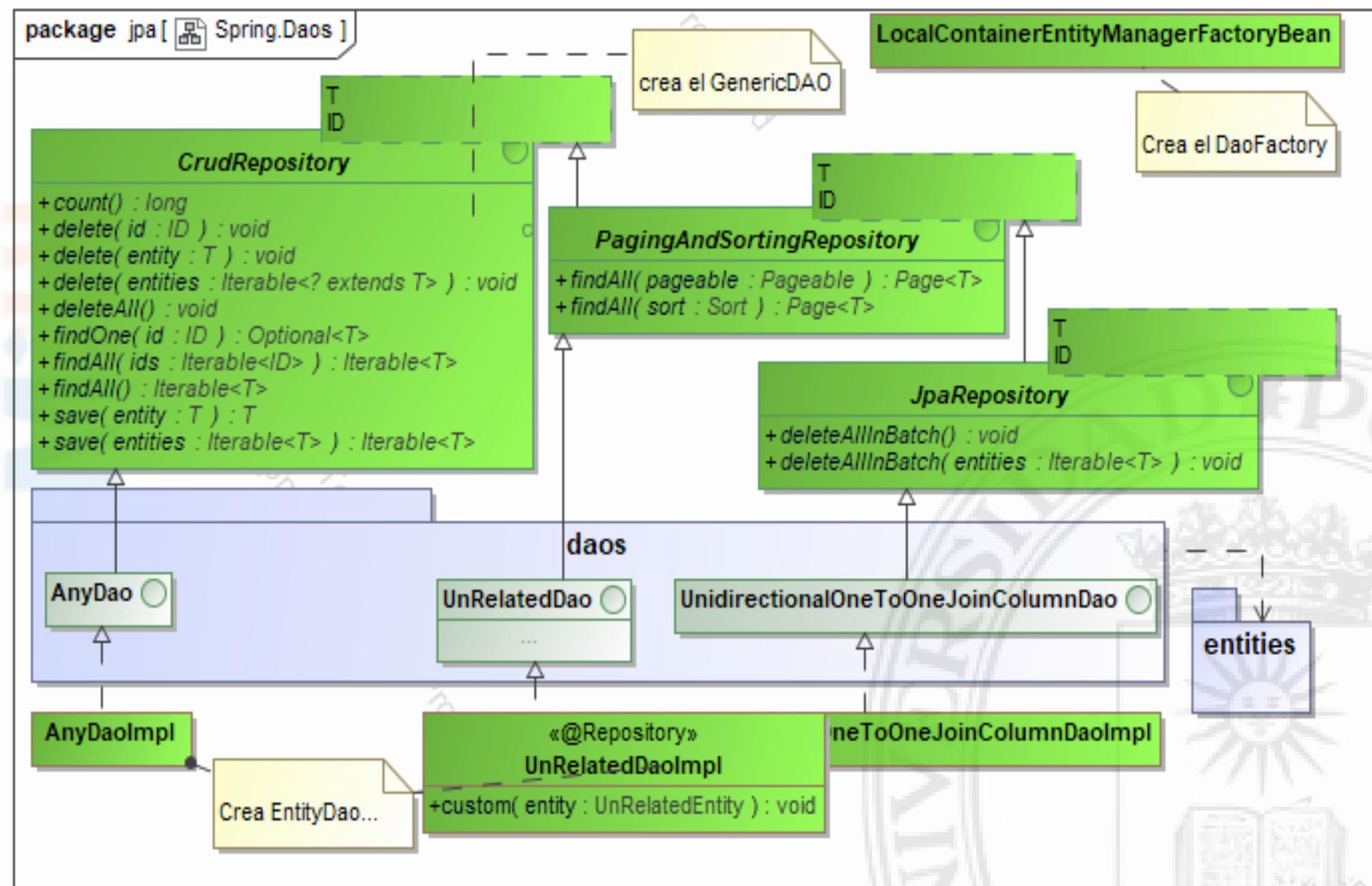
package 2.Persistence [ JPA.UnidirectionalManyToManyEntity]



Inheritance



String Data. DAO



Configuración: JPA (H2 & MySQL)

```
<!-- Database JPA-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

pom.xml


```
<!-- SQL embedded -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```



```
<!-- MySQL external -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

application.properties


```
spring.jpa.open-in-view=false
# H2 embedded -----
# spring.datasource.driver-class-name=org.h2.Driver
# spring.datasource.url=jdbc:h2:mem:testdb
# spring.datasource.username=sa
# spring.datasource.password=
```

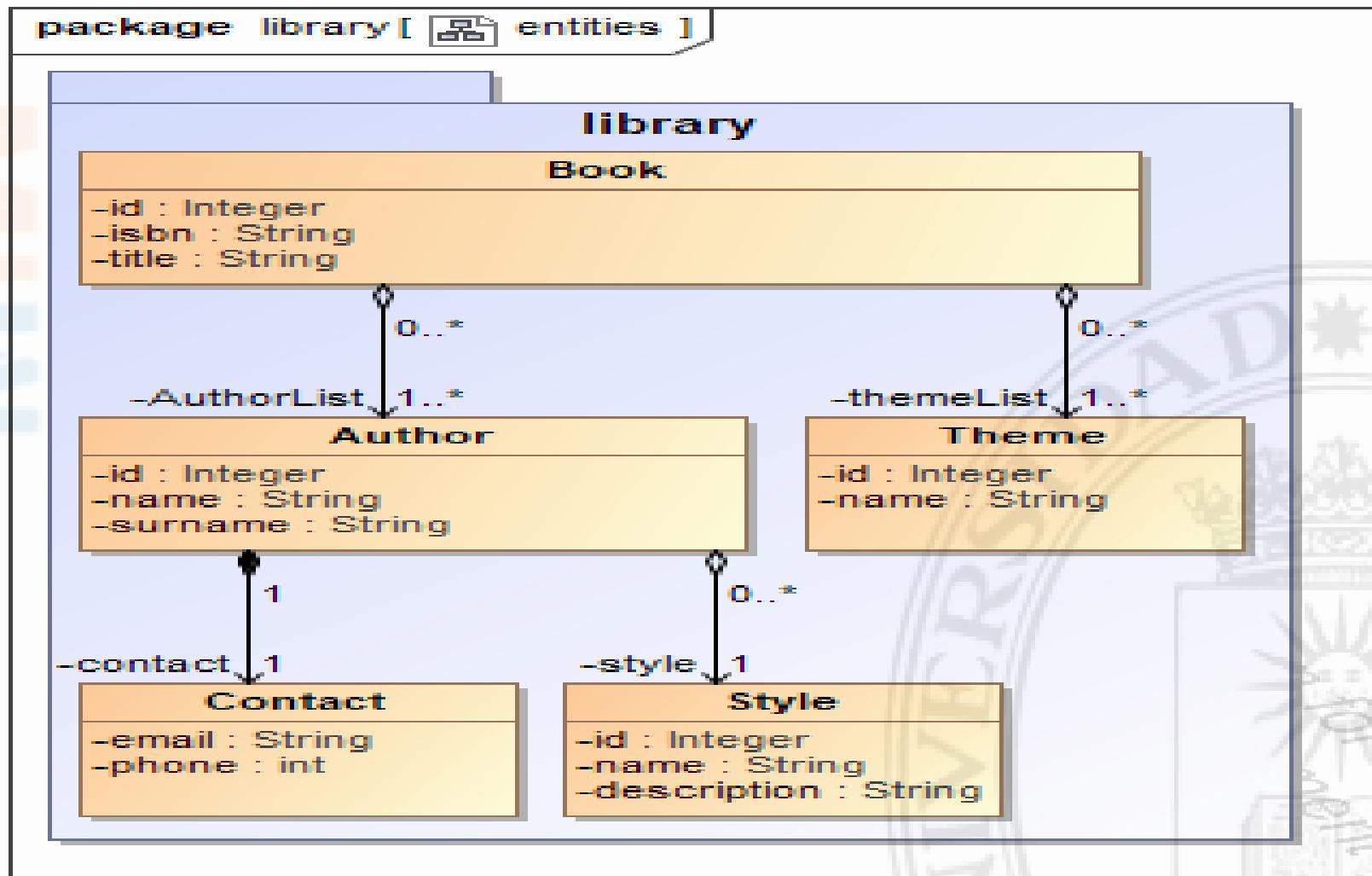
BD embebida


```
spring.jpa.open-in-view=false
# MySQL -----
spring.datasource.url = jdbc:mysql://localhost:3306/spring
# spring.datasource.driver-class-name=com.mysql.jdbc.Driver
# spring.datasource.username = root
# spring.datasource.password =
```

BD Externa

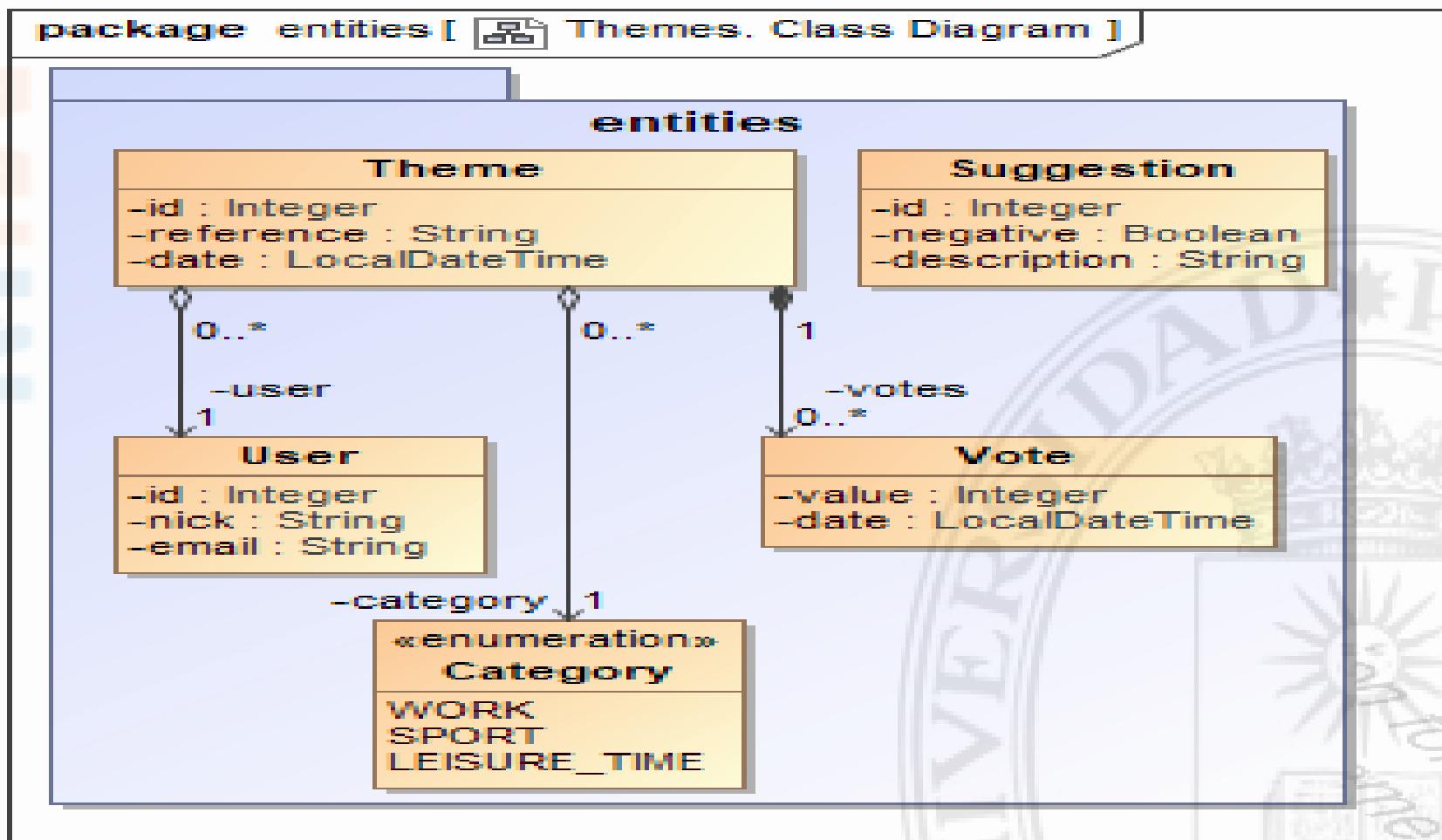
✍ Library

- ✍ Implementar el modelo de clases y DAOs
- ✍ Implementar modelos alternativos

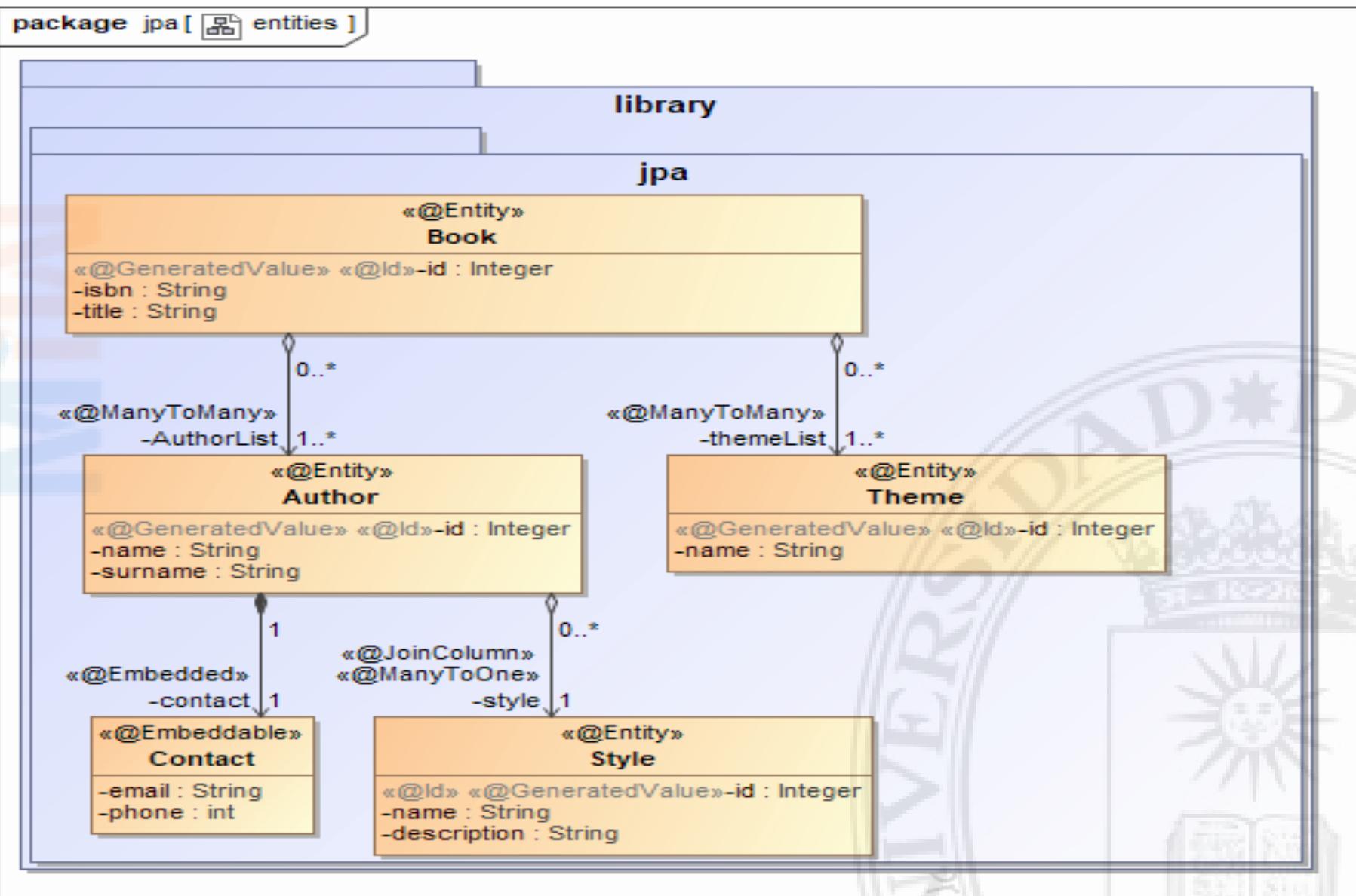


✍ Themes

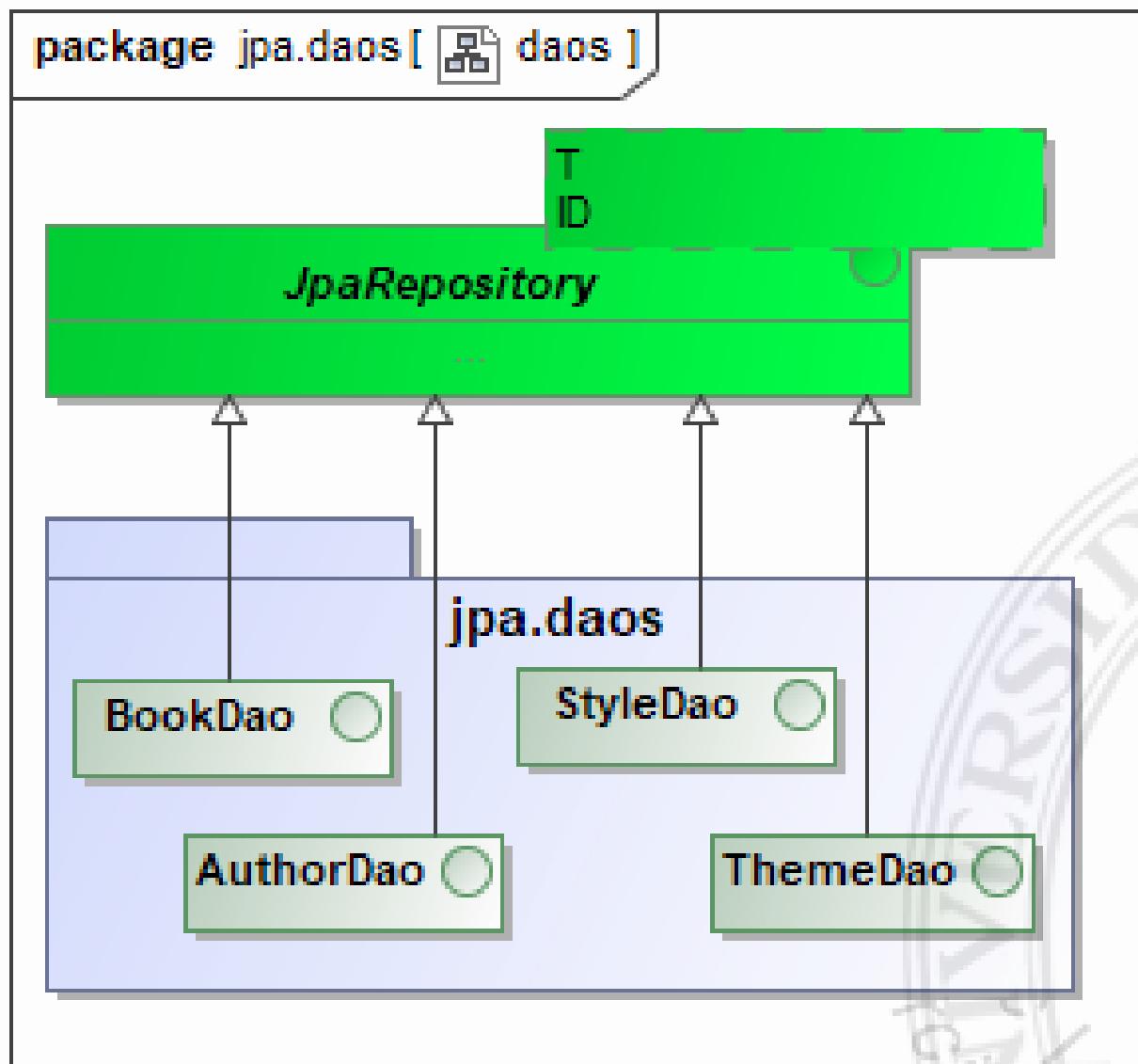
- ✍ Implementar el modelo de clases y DAOs
- ✍ Implementar modelos alternativos
- ✍ <https://github.com/miw-upm/betca-themes-jpa>



✍ Library



✍ Library



Nombre de Métodos

- Se define el método en el interface del *DAO* o *Repository*
- find / delete
 - [First], [First%n%], [Distinct]
- By
 - %Atr%:
 - 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
 - %Atr%Desc, %Atr%Asc
- 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);`

Nombre de Métodos

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname findByFirstnames findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanOrEqual	findByAgeGreaterThanOrEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1(parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1(parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1(parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnamelgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

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
- Ejemplo
 - *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);`

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 AuthorDao

- List<Author> findByStyle(Style style);
- List<String> findNameByStyleName(String styleName);
- List<String> findNameByThemeName(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/>
- 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 mibd`
 - 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

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

pom.xml
external

```
<dependency>
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo</artifactId>
    <scope>test</scope>
</dependency>
```

pom.xml
embedded

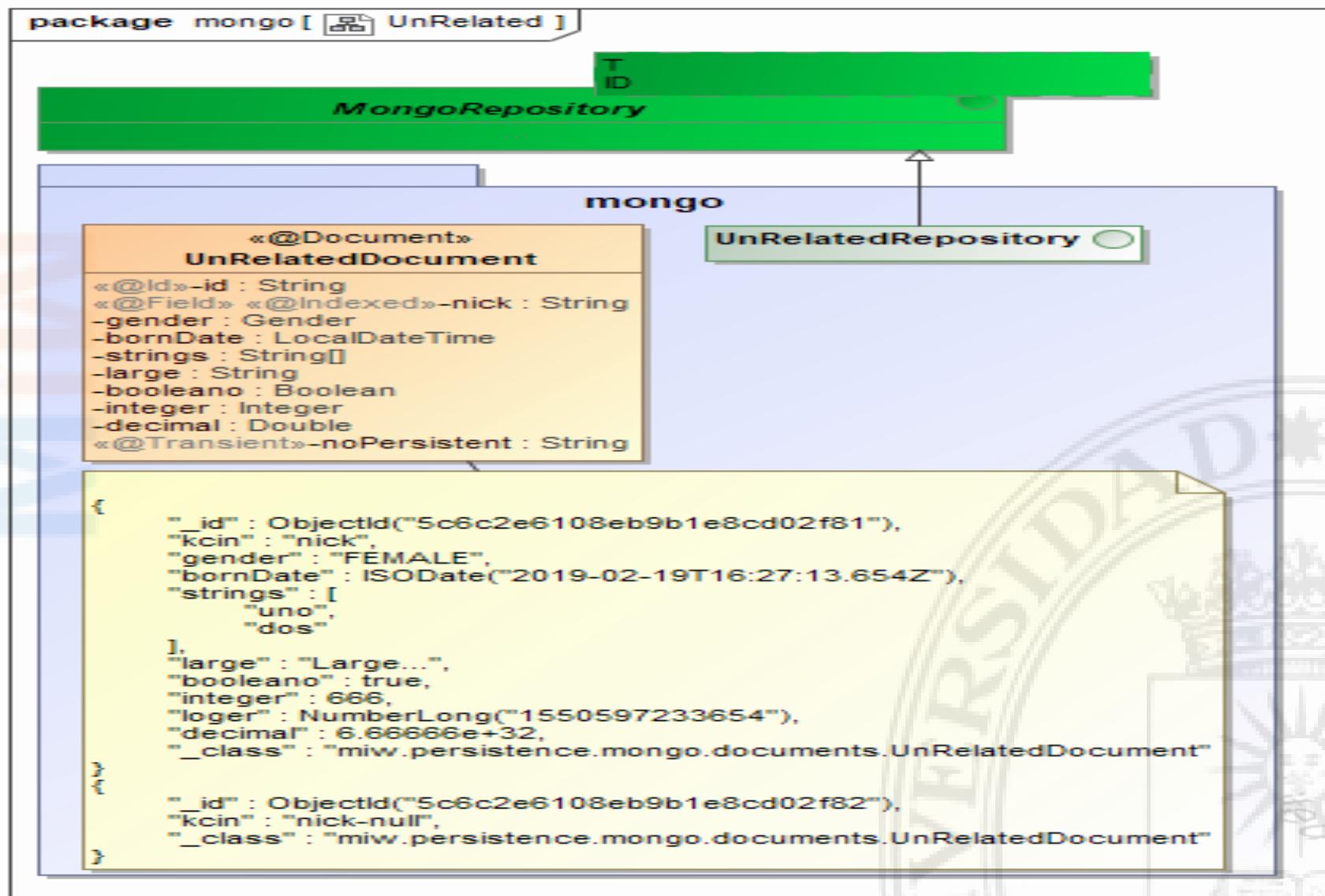
```
#MONGODB ---
# spring.data.mongodb.database=test
# spring.data.mongodb.host=localhost
# spring.data.mongodb.port=27017
# spring.data.mongodb.username=
# spring.data.mongodb.password=
```

application.properties

@Anotaciones

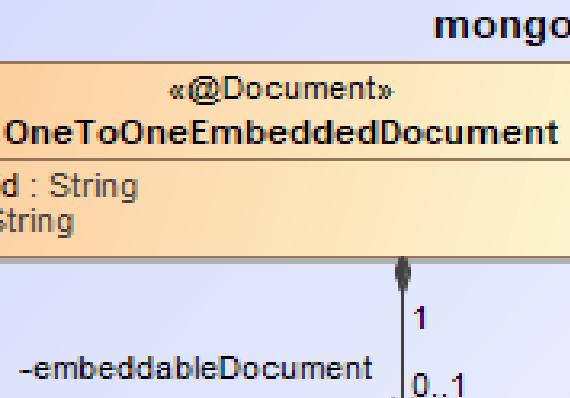
- **@Document**
 - Nombre de la colección, por defecto el nombre de la clase
 - `@Document(collection = "un_related")`
- **@Id**
 - Clave primaria. Si no se define, se autogenera
- **@Field(value = "kcin")**
 - **Opcional**, nombre del campo, por defecto el nombre del atributo
- **@Transient**
 - **Opcional**, no es persistente
- **@DBRef, @DBRef(lazy = true)**
 - **Opcional**, referencia otro Document o List<Document>
- **@Indexed, @Indexed(unique = true)**
 - **Opcional**, marca un campo como índice
- **@PersistenceConstructor**
 - **Opcional**, constructor utilizado para su persistencia
- **@DateTimeFormat(iso=ISO.DATE) // ISO.DATE_TIME, ISO.TIME**
 - **Opcional**, Formato de fecha, por defecto, ISO.DATE

Un Related



Unidirectional One To One Embedded

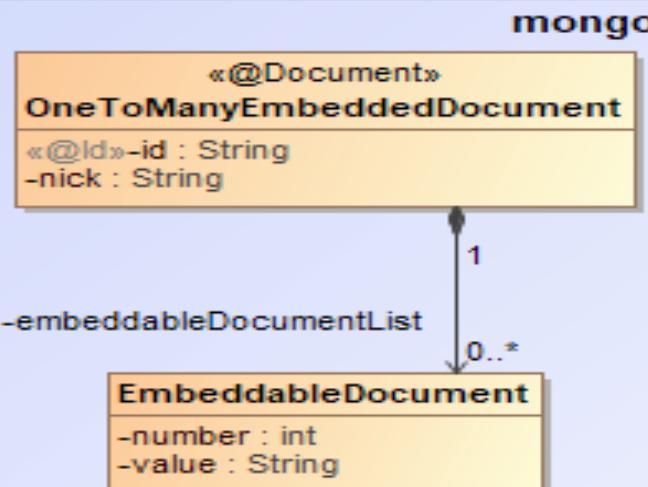
```
package mongo [  OneToOneEmbedded ]
```



```
{
    "_id" : ObjectId("5c6c323008eb9b2b488a2baa"),
    "nick" : "nick",
    "embeddableDocument" : {
        "number" : 1,
        "value" : "1"
    },
    "_class" : "miw.persistence.mongo.documents.OneToOneEmbeddedDocument"
}
```

Unidirectional One To Many Embedded

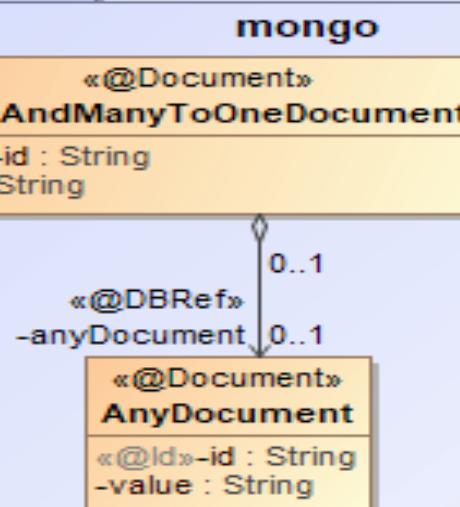
```
package mongo [  OneToManyEmbedded ]
```



```
{
    "_id" : ObjectId("5c6c2b6f08eb9b4bd45dc0e4"),
    "nick" : "nick",
    "embeddableDocumentList" : [
        {
            "number" : 1,
            "value" : "1"
        },
        {
            "number" : 2,
            "value" : "2"
        }
    ],
    "_class" : "miw.persistence.mongo.documents.OneToManyEmbeddedDocument"
}
```

Unidirectional One And Many To One

```
package mongo [  OneAndManyToOne ]
```

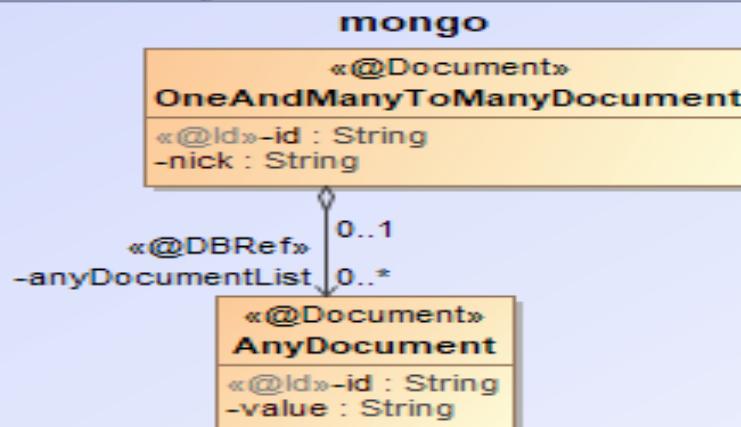


```
{
    "_id" : ObjectId("5c6c2b6f08eb9b4bd45dc0e1"),
    "nick" : "nick1",
    "anyDocument" : DBRef("anyDocument", ObjectId("5c6c2b6f08eb9b4bd45dc0e0")),
    "_class" : "miw.persistence.mongo.documents.OneAndManyToOne"
}
```

```
{
    "_id" : ObjectId("5c6c2b6f08eb9b4bd45dc0e0"),
    "value" : "any",
    "_class" : "miw.persistence.mongo.documents.AnyDocument"
}
```

Unidirectional One And Many To Many

```
package mongo [  OneAndManyToMany ]
```

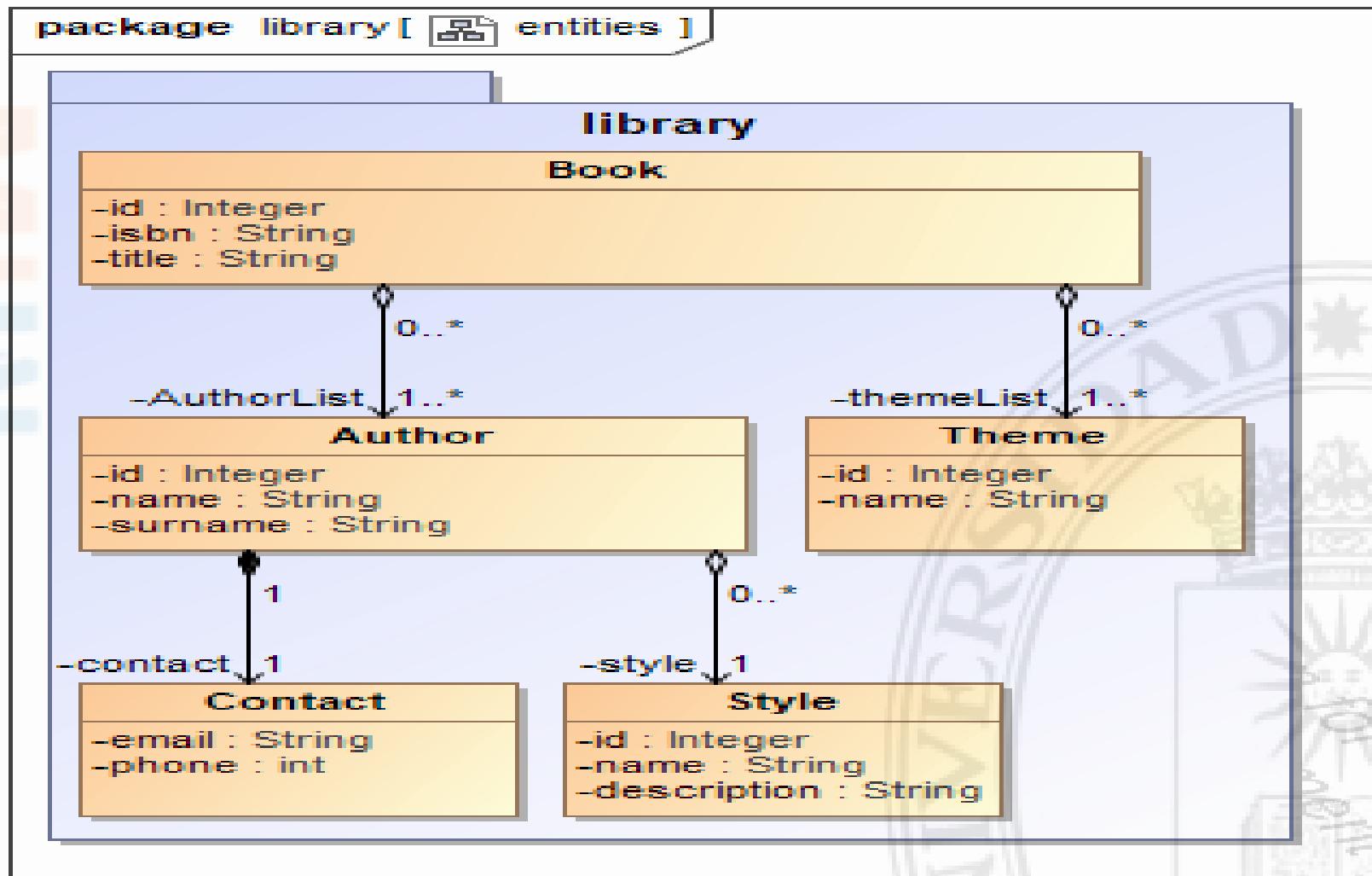


```
{
    "_id" : ObjectId("5c6c5cb608eb9b41a0f7a494"),
    "nick" : "nick",
    "anyDocumentList" : [
        DBRef("anyDocument", ObjectId("5c6c5cb608eb9b41a0f7a492")),
        DBRef("anyDocument", ObjectId("5c6c5cb608eb9b41a0f7a493"))
    ],
    "_class" : "miw.persistence.mongo.documents.OneAndManyToManyDocument"
}
```

```
{
    "_id" : ObjectId("5c6c5cb608eb9b41a0f7a492"),
    "value" : "any",
    "_class" : "miw.persistence.mongo.documents.AnyDocument"
}
{
    "_id" : ObjectId("5c6c5cb608eb9b41a0f7a493"),
    "value" : "any2",
    "_class" : "miw.persistence.mongo.documents.AnyDocument"
}
```

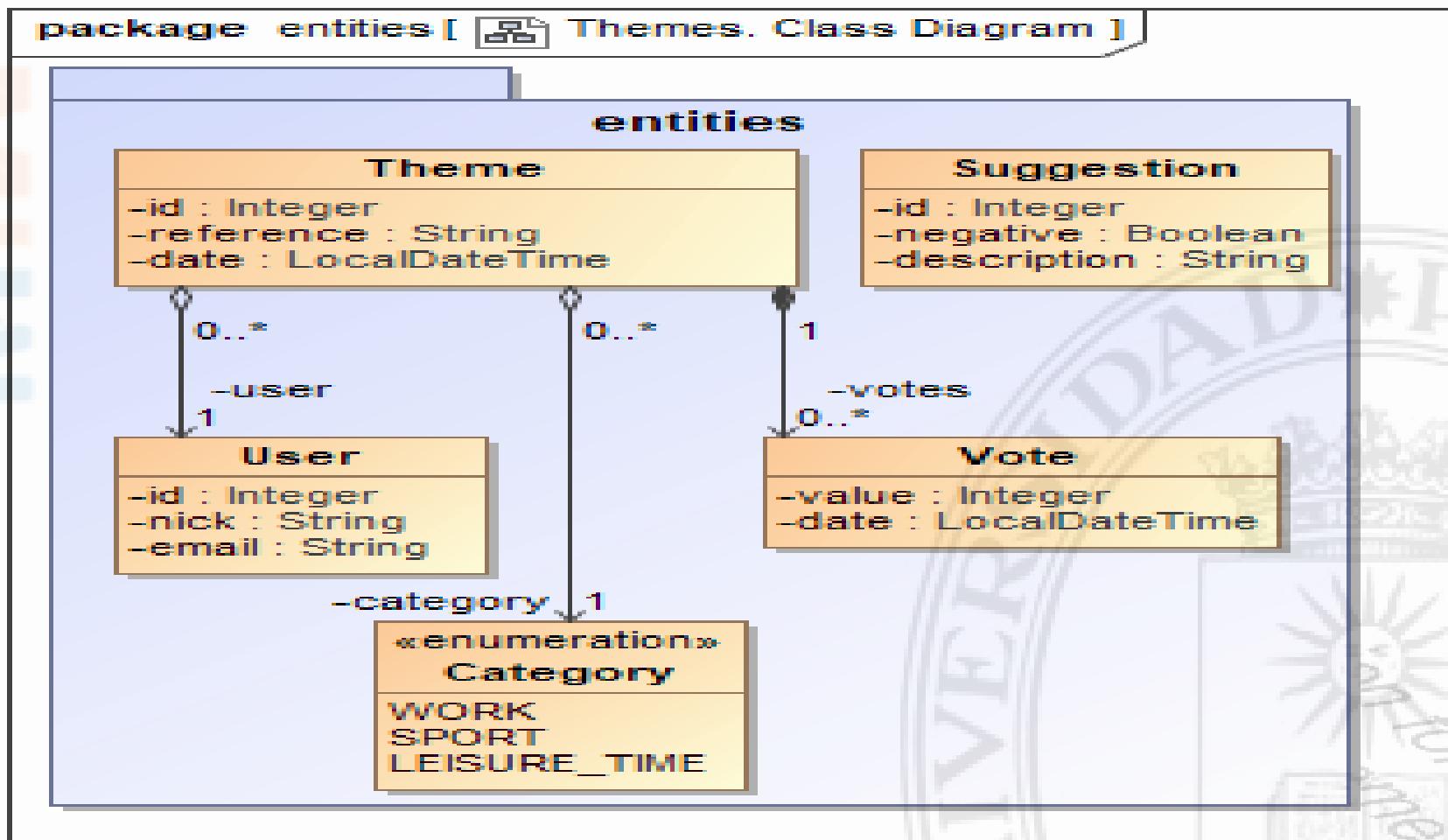
✍ Library

- ✍ Implementar el modelo de clases y repositorios

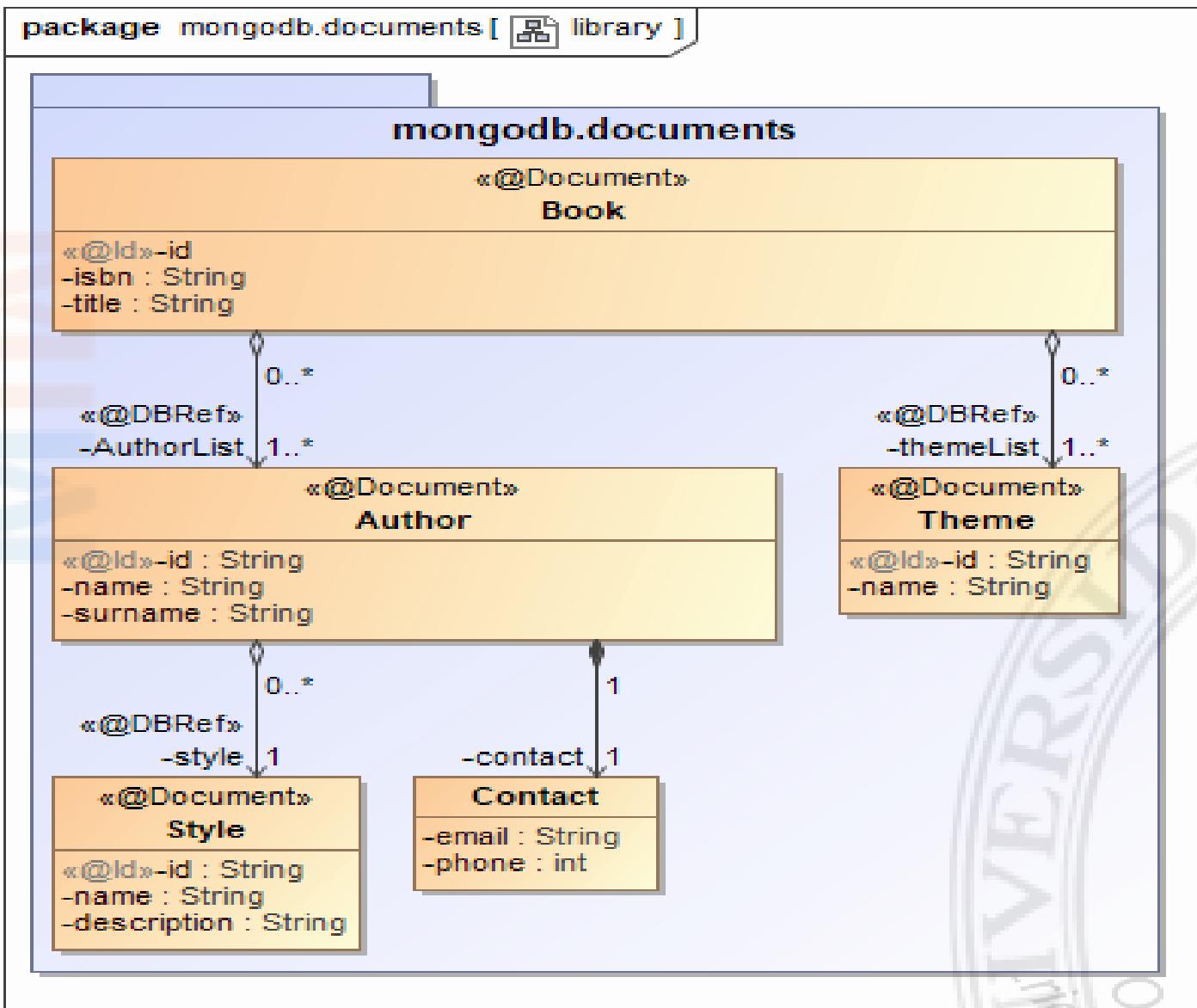


✍ Themes

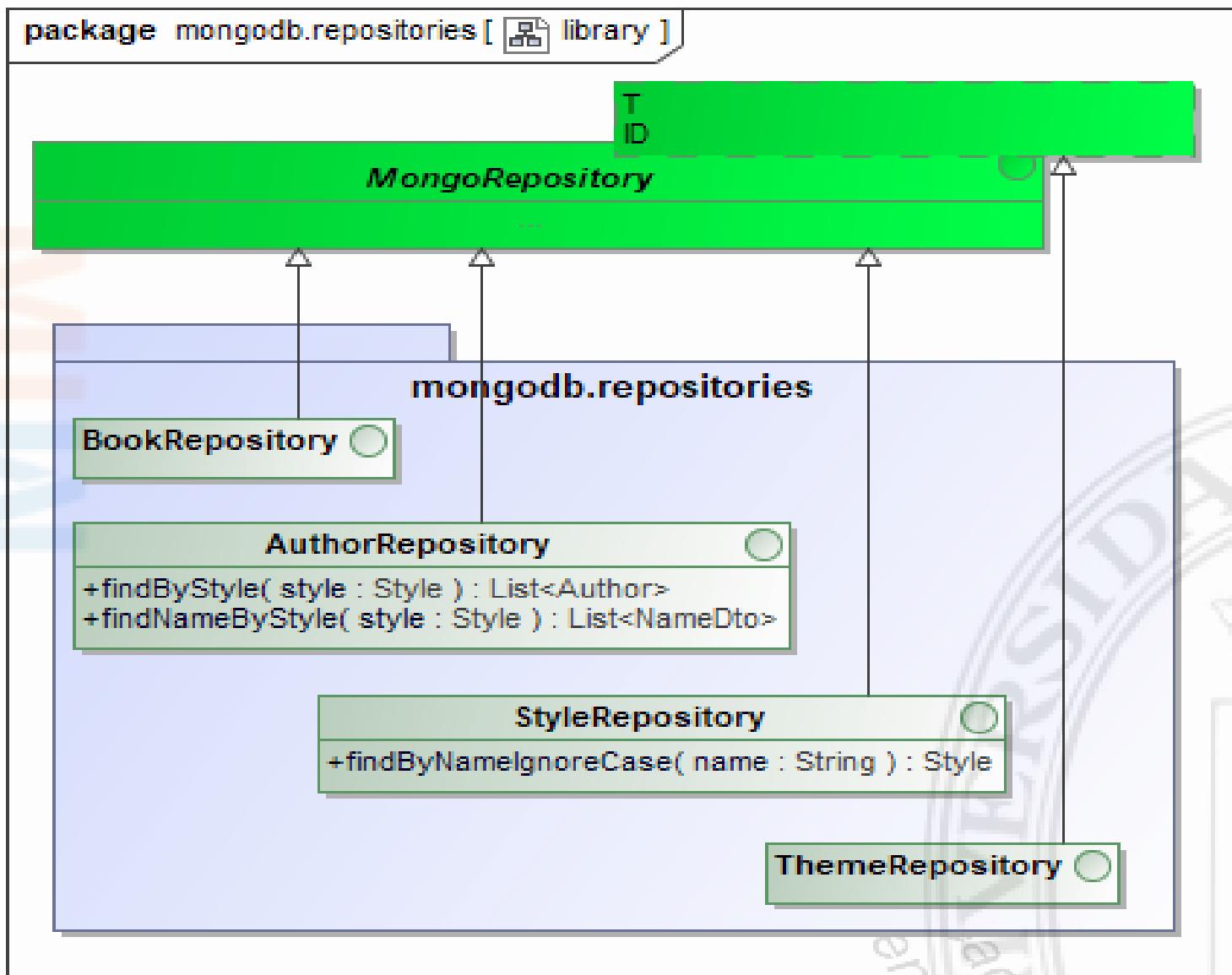
- ✍ Implementar el modelo de clases y repositorios
- ✍ <https://github.com/miw-upm/betca-themes-mongodb>



✍ Library



✍ Library



Consultas y Borrados. Con Nombre de Métodos

- Se define el método en el interface del *Repository*
- find / delete
 - [First], [First%n%], [Distinct]
- By
 - %Atr%:
 - 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
 - %Atr%Desc, %Atr%Asc
- Ejemplos
 - *Entity findByAtr1(String atr1);*
 - *List<Entity> findFirst3ByAtr1StartingWith(String prefix);*
 - *List<Entity> findByAtr1OrAtr2OrderByAtr3Desc(String atr1, String atr2);*
 - *List<Entity> findByAtr1GreaterThan(int value);*
 - *List<Entity> findByAtrIn(Collection<Integer> values);*
 - *int deleteByAtr1(String atr1);*
 - *int deleteByAtr1GreaterThan(int value);*
 - *OneToOneEmbeddedDocument findByEmbeddableDocumentValue (String embeddableDocumentValue);*
 - *List<OneToOneEmbeddedDocument> findFirst10ByEmbeddableDocumentValue (String embeddableDocumentValue);*

Consultas y Borrados. Con Nombre de Métodos

Table 7. Supported keywords for query methods

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

Consultas y Borrados. Con Nombre de Métodos

Table 12. Query keywords

Logical keyword	Keyword expressions
AND	And
OR	Or
AFTER	After, IsAfter
BEFORE	Before, IsBefore
CONTAINING	Containing, IsContaining, Contains
BETWEEN	Between, IsBetween
ENDING_WITH	EndingWith, IsEndingWith, EndsWith
EXISTS	Exists
FALSE	False, IsFalse
GREATER_THAN	GreaterThan, IsGreaterThan
GREATER_THAN_EQUALS	GreaterThanOrEqualTo, IsGreaterThanOrEqualTo
IN	In, IsIn
IS	Is, Equals, (or no keyword)
IS_EMPTY	IsEmpty, Empty
IS_NOT_EMPTY	IsNotEmpty, NotEmpty
IS_NOT_NULL	NotNull, IsNotNull
IS_NULL	Null,IsNull
LESS_THAN	LessThan, IsLessThan
LESS_THAN_EQUAL	LessThanOrEqualTo, IsLessThanOrEqualTo
LIKE	Like, IsLike
NEAR	Near, IsNear
NOT	Not, IsNot
NOT_IN	NotIn, IsNotIn
NOT_LIKE	NotLike, IsNotLike
REGEX	Regex, MatchesRegex, Matches
STARTING_WITH	StartingWith, IsStartingWith, StartsWith
TRUE	True, IsTrue
WITHIN	Within, IsWithin

Paginación

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

Json Query

```
@Query("{ 'nick':?0}")
UnRelatedDocument findByNick(String nick); // Query NOT necessary

// FIELDS: _id: incluido por defecto, 1: solo los especificados, 0: todos excepto el especificado
@Query(value = "{ 'nick':?0}", fields = "{ 'bornDate':1,'large':1}")
Query1Dto findIdAndBornDateAndLargeByNick(String nick);

@Query(value = "{ 'nick':?0}", fields = "{ '_id':0,'bornDate':1}")
Query2Dto findBornDateByNick(String nick);

@Query("{ $and:[{ 'nick':?0},{ 'large':?1}]}")
UnRelatedDocument findByNickAndLarge(String nick, String large); // Query NOT necessary

@Query("{nick:$in:?0} ")
List<UnRelatedDocument> findByNickIn(Collection nicks); // Query NOT necessary

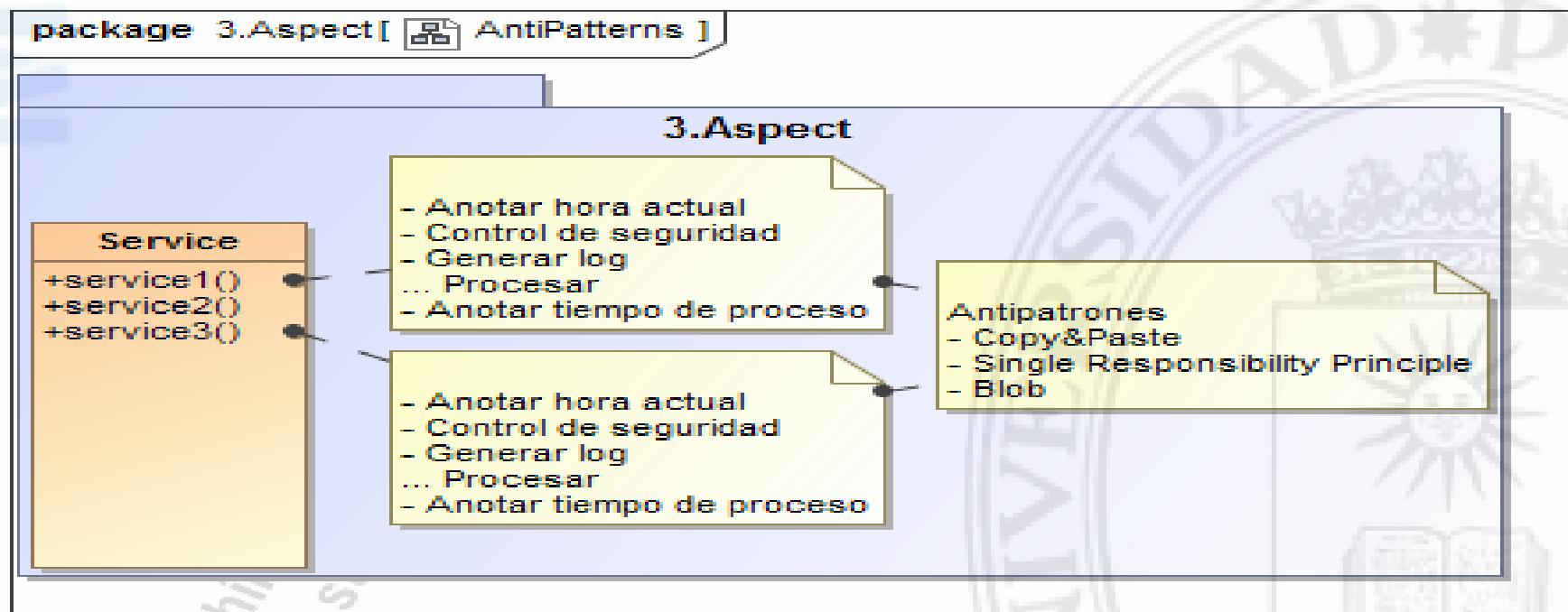
//options: 'i': Case insensitivity
@Query("?" + "[0] == null ? { $where : 'true' } : { nick : {$regex:[0], $options: 'i'} } ")
List<UnRelatedDocument> findByNickLikeIgnoreCaseNullSafe(String nick); // allow NULL: all elements

@Query("{ $and:["
    + "?" + "[0] == null ? { $where : 'true' } : { nick : {$regex:[0], $options: 'i'} } ,"
    + "?" + "[1] == null ? { $where : 'true' } : { large : {$regex:[1], $options: 'i'} } }"
    + "] }")
List<UnRelatedDocument> findByNickLikeAndLargeLikeNullSafe(String nick, String large); // allow NULL: all elements

List<OneAndManyToManyDocument> findByAnyDocumentListContaining(AnyDocument anyDocument);
```

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

- 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() {}`

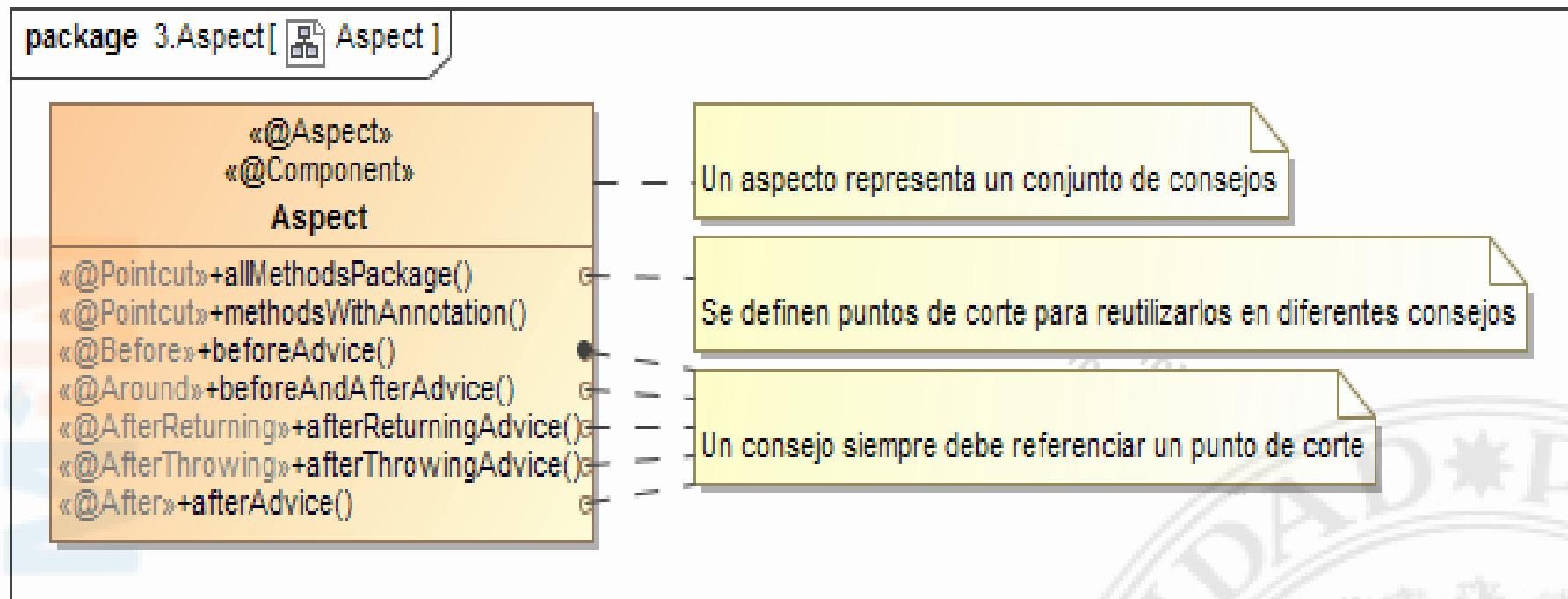
Ejemplos

- "*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, sino 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



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

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

Spring. Introducción

- Proyecto Spring Framework

```
<!-- Fix RestBuilder to support PATCH -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
</dependency>

# Server web #####
# server.contextPath=/api/v0
```

pom.xml

application.properties

Cliente Rest

- Cliente Rest: Clase de Spring: *RestTemplate*
 - *new RestTemplate().exchange(requestEntity, responseType)*
 - **requestEntity**
 - Uri:
 - *URI uri = UriComponentsBuilder.fromHttpUrl(url).build().encode().toUri();*
 - Método:
 - *HttpMethod.GET, HttpMethod.POST...*
 - Cuerpo (opcional)
 - *Object (String, String[], Dto, Dto[], ...)*
 - Cabecera (opcional)
 - *HttpHeaders headers = new HttpHeaders(); headers.set("token", "tooken");*
 - **responseType: Class**
 - *Object.class, Dto.class, String[].class, Dto[].class...*
- Se procesa la respuesta:
 - *restTemplate.getBody();*
 - *restTemplate.getHeaders();*
 - *restTemplate.getStatusCode()*

Cliente Rest

```
@Test
void testState() {
    URI uri = UriComponentsBuilder
        .fromHttpUrl("http://localhost:" + port + AdminResource.ADMININS + AdminResource.STATE)
        .build().encode().toUri();
    RequestEntity<Object> requestEntity = new RequestEntity<>(HttpMethod.GET, uri);
    ResponseEntity<String> responseEntity = new RestTemplate().exchange(requestEntity, String.class);
    String response = responseEntity.getBody();

    assertEquals("{\"state\":\"ok\"}", response);
}

@Test
void testStateRestBuilder() {
    String json = new RestBuilder<String>(port).clazz(String.class)
        .path(AdminResource.ADMININS).path(AdminResource.STATE).get().log().build();
    assertEquals("{\"state\":\"ok\"}", json);
}
```

Cliente Rest

```
package rest-client[  RestBuilder ]
```

rest-client

```
T
RestBuilder
- SERVER_URI_DEFAULT : String = http://localhost
- PORT_DEFAULT : int = 8080

+RestBuilder( port : int )
+RestBuilder( serverUri : String, port : int )
+path( path : String ) : RestBuilder<T>
+expand( values : Object... ) : RestBuilder<T>
+clazz( clazz : Class<T> ) : RestBuilder<T>
+accept( mediaType : MediaType )
+basicAuth( nick : String, pass : String ) : RestBuilder<T>
+bearerAuth( token : String ) : RestBuilder<T>
+header( key : String, value : String ) : RestBuilder<T>
+param( key : String, value : String ) : RestBuilder<T>
+loadFile( fileName : String ) : RestBuilder<T>
+body( body : Object ) : RestBuilder<T>
+notError() : RestBuilder<T>
+log() : RestBuilder<T>
+post() : RestBuilder<T>
+get() : RestBuilder<T>
+put() : RestBuilder<T>
+patch() : RestBuilder<T>
+delete() : RestBuilder<T>
+build() : T
```

```
Dto response = new RestBuilder<Dto>(port).clazz(Dto.class).path(AdminResourceADMINS).path(AdminResourceBODY)
    .body(dto).post().build();

Dto response = new RestBuilder<Dto>(port).clazz(Dto.class).path(AdminResourceADMINS).path(AdminResourceECHO)
    .path(AdminResourceID).expand(999).body(dto).put().build();

List<Dto> response = Arrays.asList(new RestBuilder<Dto[]>(port).clazz(Dto[].class)
    .path(AdminResourceADMINS).path(AdminResourceBODY) .path(AdminResourceDTO_LIST).get().build());

new RestBuilder<>(port).basicAuth("user", "123456").path(SecurityResourceSECURITY).path(SecurityResourceALL).get().build();

String json = new RestBuilder<String>(port).clazz(String.class) .header("token", "tooken")
    .path(AdminResourceADMINS).path(AdminResourceECHO).path(AdminResourceID).expand(666)
    .param("param", "paaaaram").get().log().build();

new RestBuilder<>(port).loadFile("test.yml").path(AdminResourceADMINS).path(AdminResourceDB).post().log().build();

HttpClientErrorException exception = assertThrows(HttpClientErrorException.class, () ->
    new RestBuilder<Dto>(port).header("token", "kk").path(ExceptionResourceEXCEPTIONS).path(ExceptionResourceERROR)
        .path(AdminResourceID).expand(66).param("param", "good").get().build()
);
assertEquals(HttpStatus.BAD_REQUEST, exception.getStatusCode());
```

Cliente Rest

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)

@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestPropertySource(locations = "classpath:test.properties")
@ActiveProfiles("dev")
@Tag("api")
public @interface ApiTestConfig {
}
```

```
@ApiTestConfig
class AdminResourceIT {

    @Value("${local.server.port}")
    private int port = 8080;

    @Test
    void testStateRestBuilder() {
        String json = new RestBuilder<String>(port).clazz(String.class)
            .path(AdminResource.ADMIN).path(AdminResource.STATE).get().log().build();
        assertEquals("{\"state\":\"ok\"}", json);
    }
}
```

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`
-  *Ejemplo: AdminResource*

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

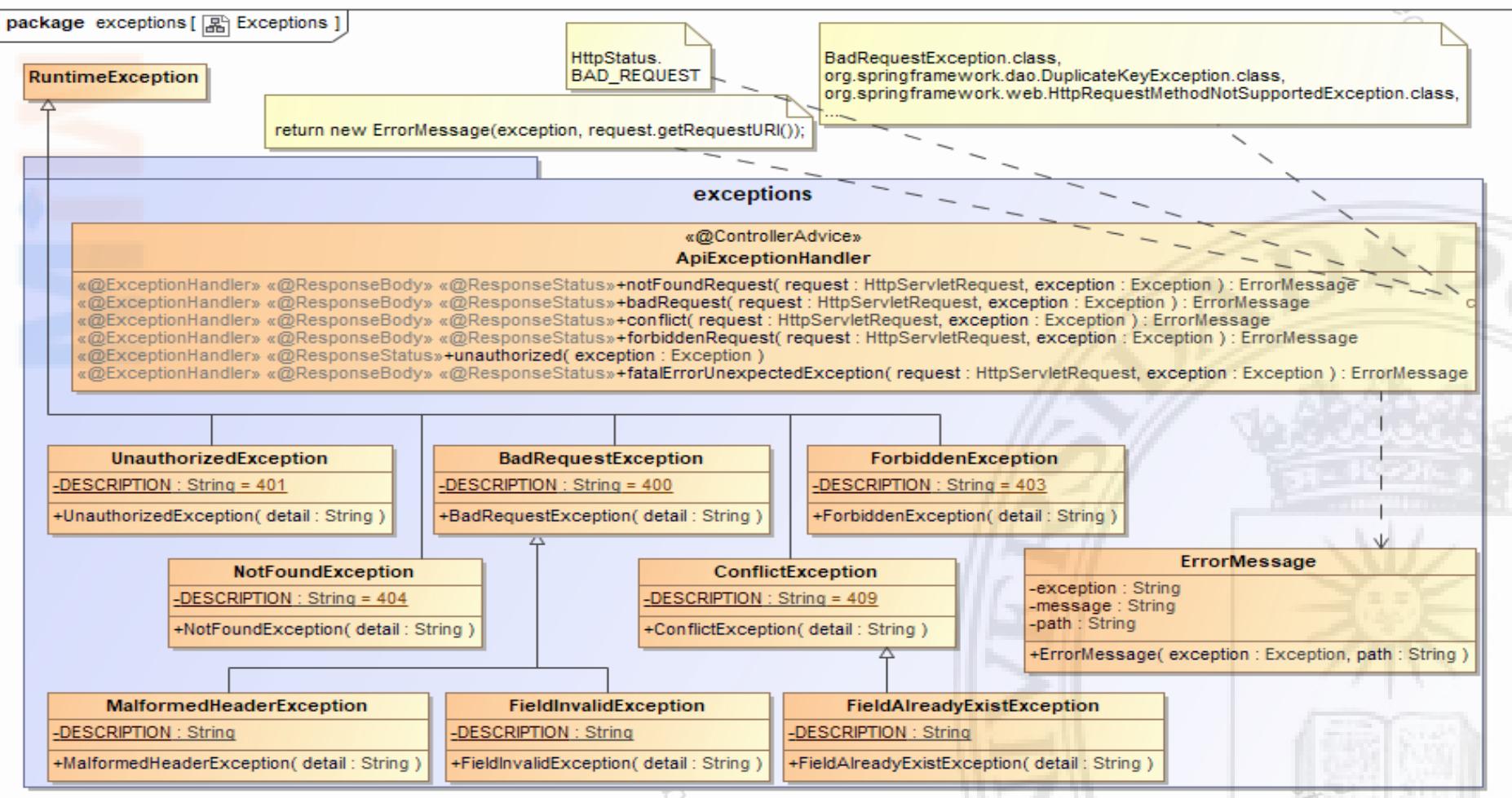
- Por defecto, se transportan en JSON
 - `@JsonIgnore` para aquellos campos que no queramos que se conviertan a JSON
 - `@JsonInclude (Include.NON_NULL)` para que no se añada a la respuesta si es null
- Listas: `List<String>, List<Integer>, List<Dto> ...`

■ 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*
 -  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

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>
```

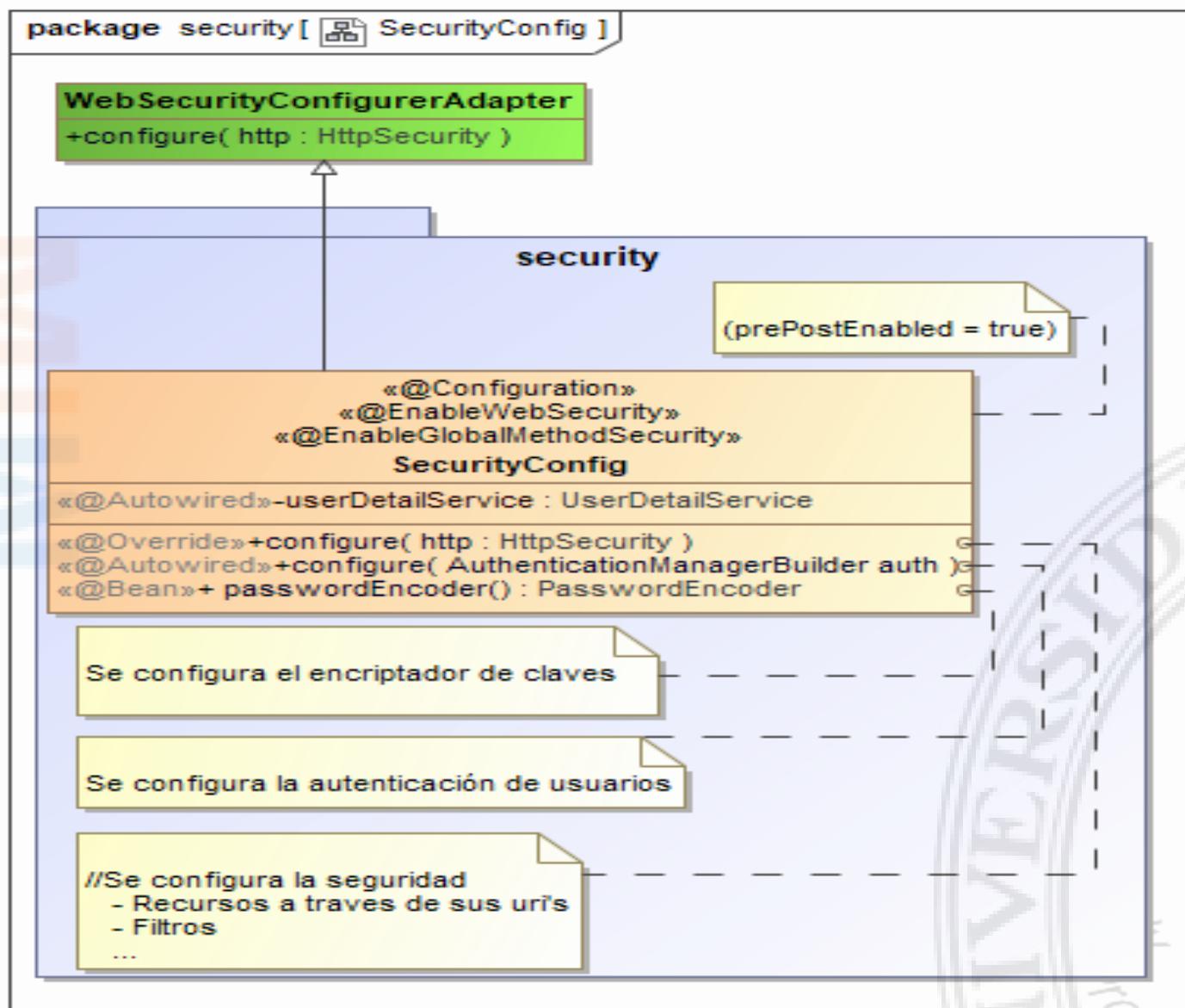
pom.xml

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

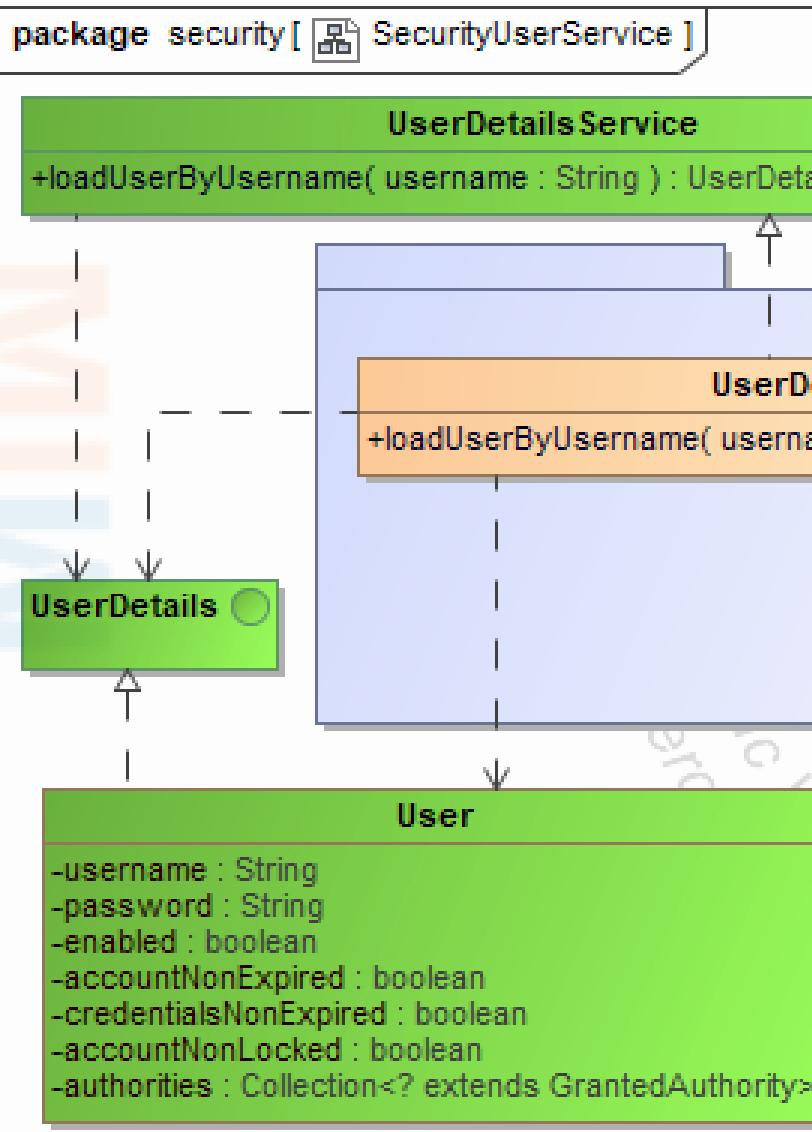
Spring Security

- **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



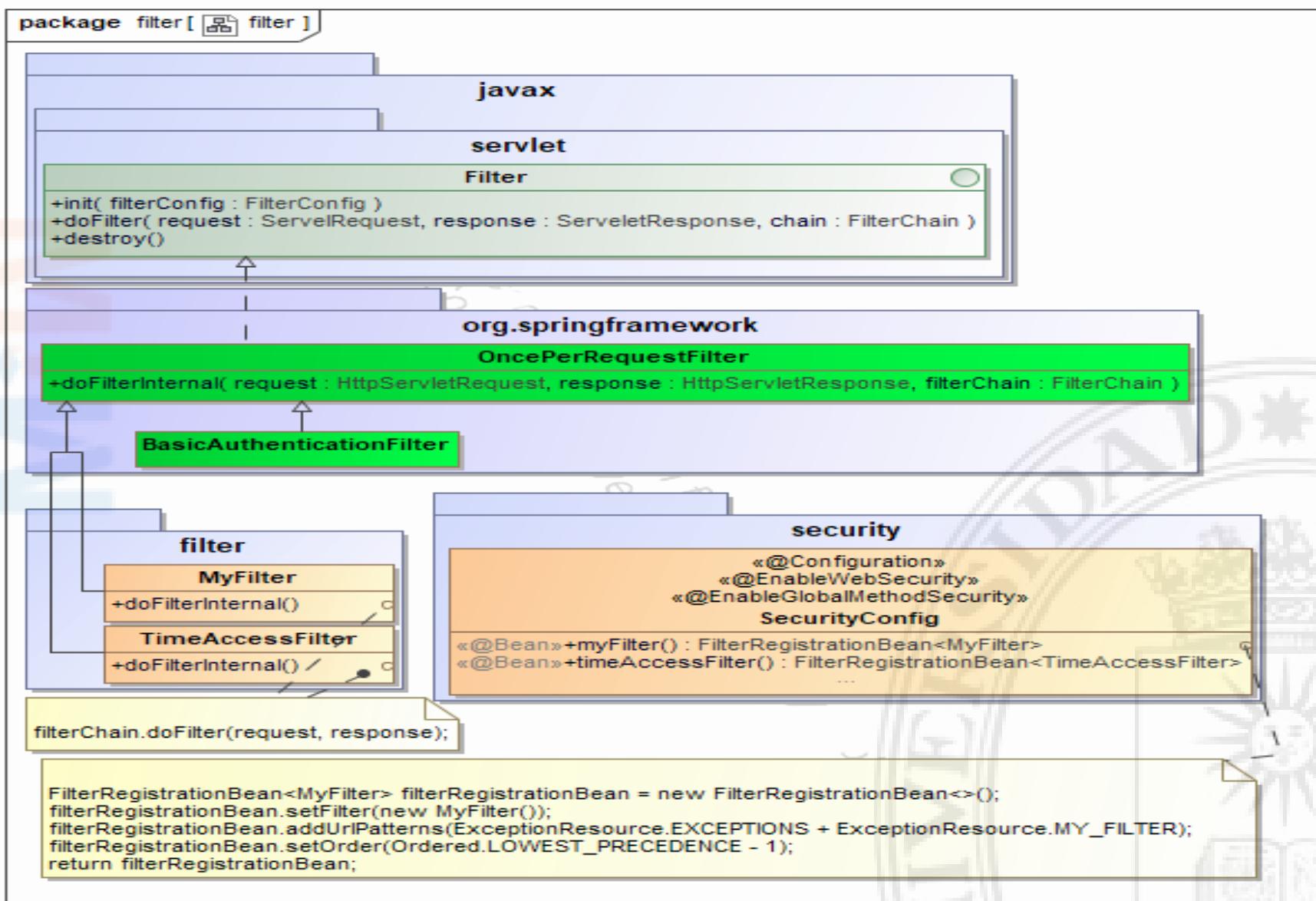
Servicio de Usuarios



✍ 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

```
public class MyFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        LogManager.getLogger(this.getClass().getName()).debug(">>> FILTER MyFilter...");  

        filterChain.doFilter(request, response);
    }
}
```

```
public class TimeAccessFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                FilterChain filterChain) throws IOException, ServletException {

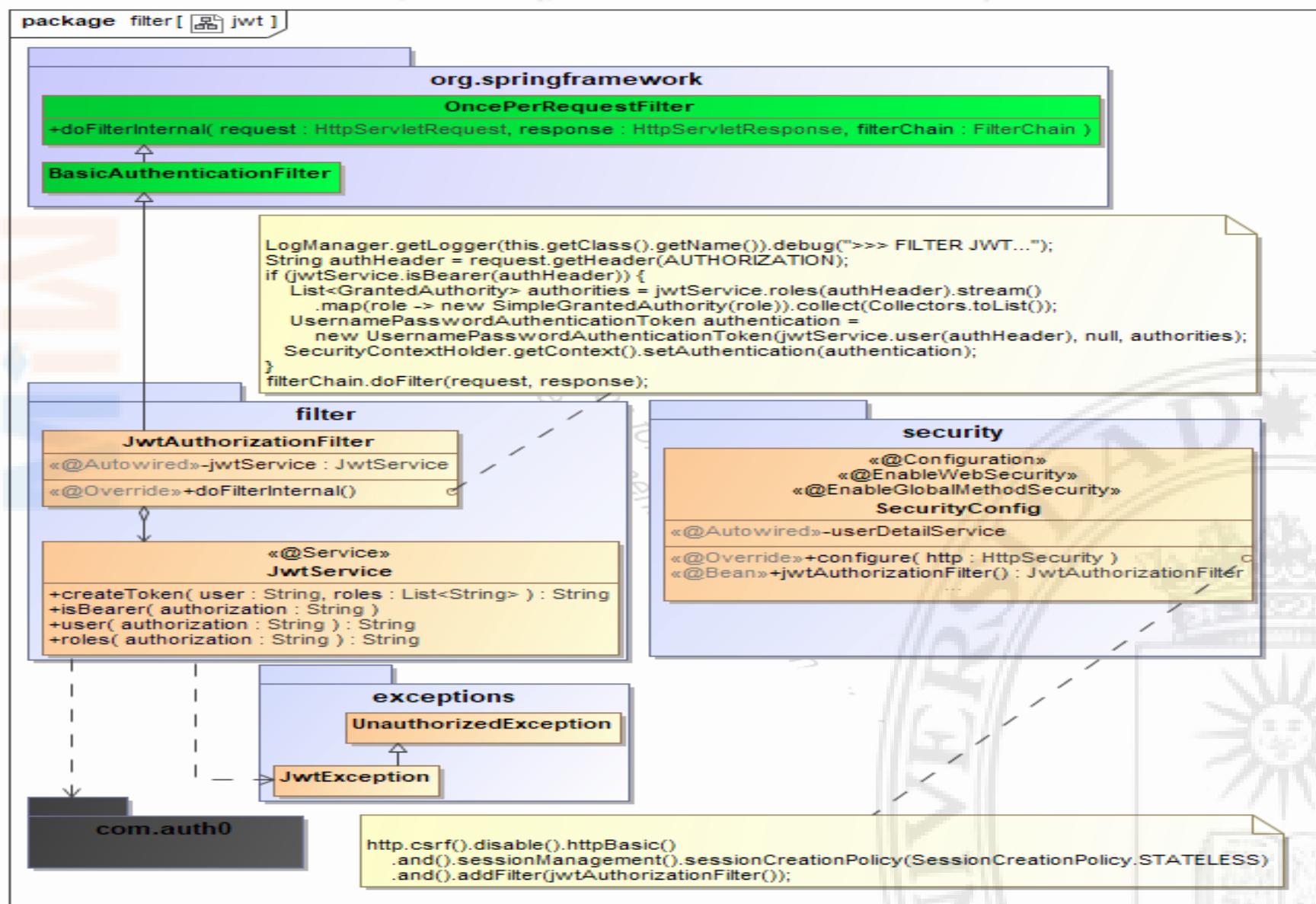
        LogManager.getLogger(this.getClass().getName()).debug(">>>>> TimeAccessFilter... Open: 4:00 to 4:59");
        int hour = LocalDateTime.now().getHour();
        if (4 <= hour && hour < 4) {
            filterChain.doFilter(request, response);
        } else {
            response.sendError(HttpStatus.SC_FORBIDDEN);
        }
    }
}
```

```
@Bean
public FilterRegistrationBean<TimeAccessFilter> timeAccessFilter() {
    FilterRegistrationBean<TimeAccessFilter> filterRegistrationBean = new FilterRegistrationBean<>();
    filterRegistrationBean.setFilter(new TimeAccessFilter());
    filterRegistrationBean.addUrlPatterns(ExceptionResource.EXCEPTIONS + ExceptionResource.OUT_OF_TIME);
    filterRegistrationBean.setOrder(Ordered.LOWEST_PRECEDENCE - 1);
    return filterRegistrationBean;
}
```

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.eyJJuYmYiOjE1NTA3NDMwNzAsInJvbGVzIjpbIlJPTEVfVVNFUiJdLCJpc3MiOiJtaXctc3ByaW5nNSIsImV4cCI6MTU1MDc0NjY3MCwiaWF0IjoxNTUwNzQzMDCwLCJ1c2VyIjoidXNlciJ9.eyJ0AOzk52f1TziZ0AqbQWDnnUqR651tQJLdmpA_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)

```
@RestController
@RequestMapping(JwtResource.JWT)
public class JwtResource {

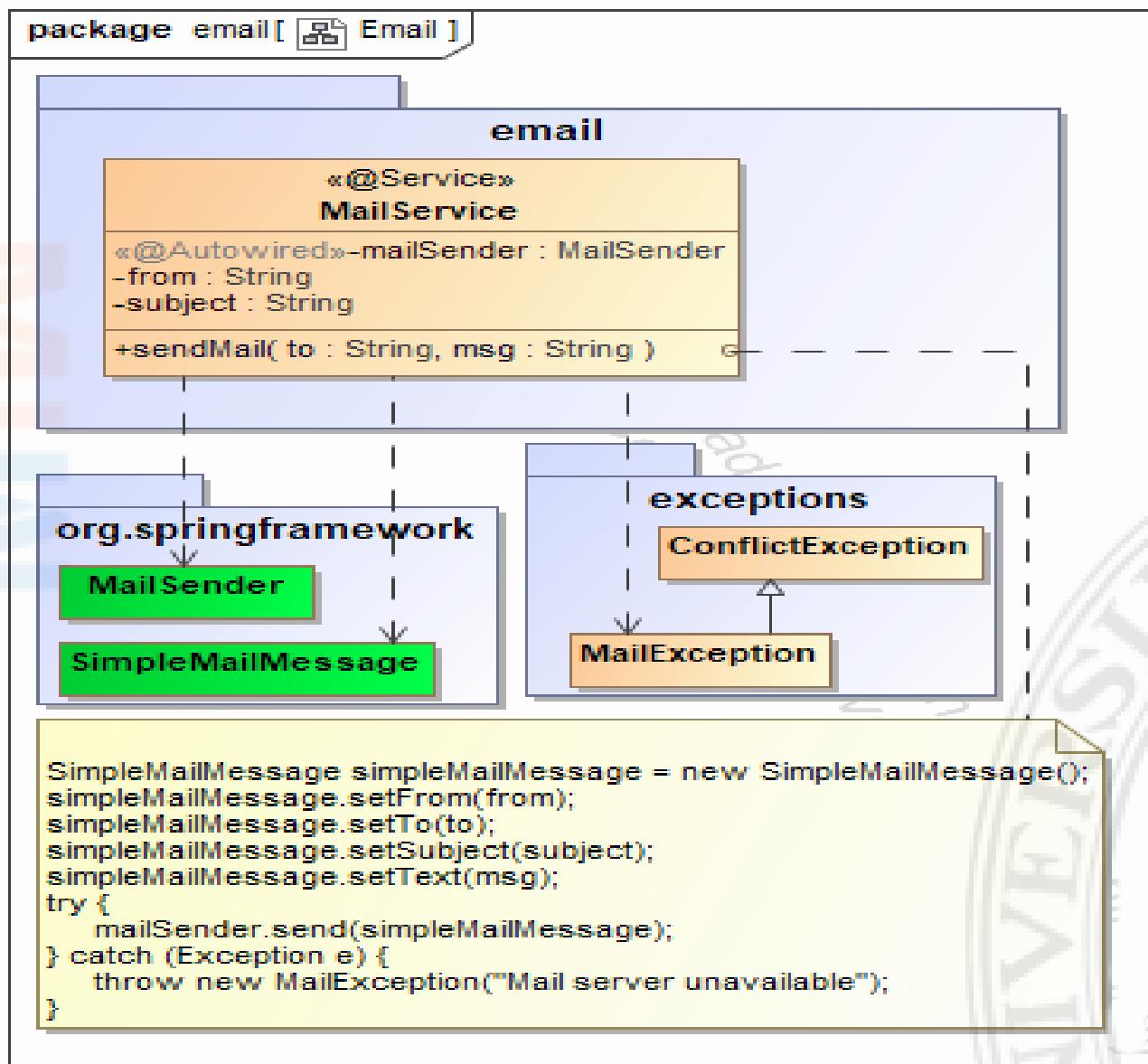
    public static final String JWT = "/jwt";
    public static final String TOKEN = "/token";

    @Autowired
    private JwtService jwtService;

    @PreAuthorize("authenticated")
    @PostMapping(value = TOKEN)
    public String login(@AuthenticationPrincipal User activeUser) {
        List<String> roleList = activeUser.getAuthorities().stream().map
            (authority -> authority.getAuthority()).collect(Collectors.toList());
        return jwtService.createToken(activeUser.getUsername(), roleList);
    }

    @PreAuthorize("hasRole('USER')")
    @GetMapping
    public String verify() {
        return "OK. permitido JWT";
    }
}
```

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

```
@TestConfig
public class MailIT {

    @MockBean
    private MailSender mailSender;

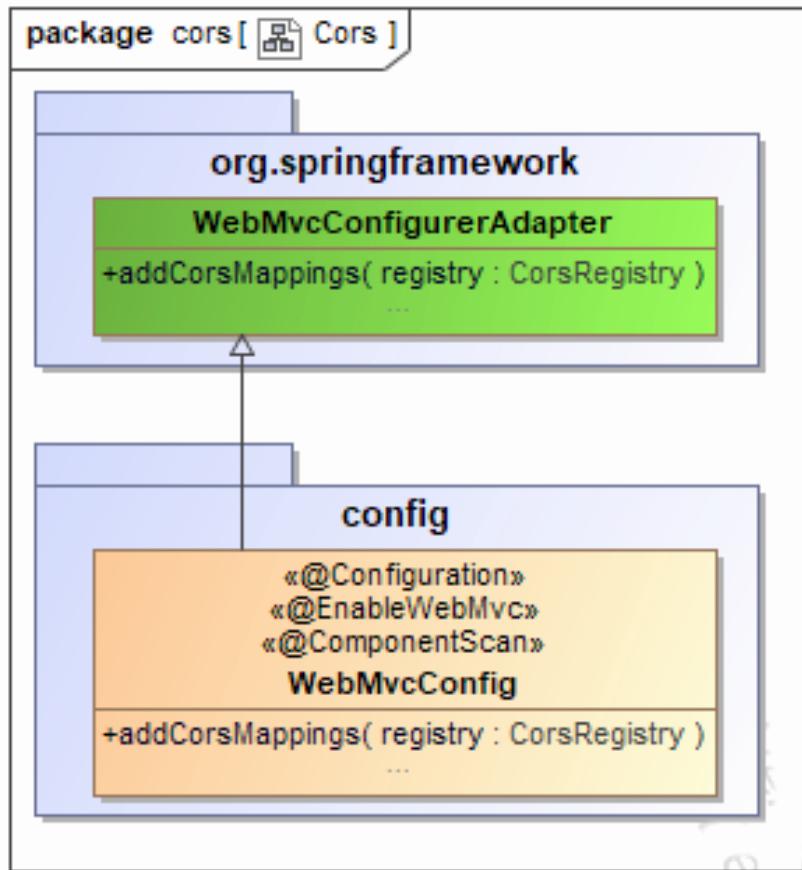
    @Autowired
    private MailService mailService;

    @Test
    public void testSendMessage() {
        final ArgumentCaptor<SimpleMailMessage> captor =
            ArgumentCaptor.forClass(SimpleMailMessage.class);
        this.mailService.send("test@gmai.com", "test message");
        verify(mailSender).send(captor.capture());
        assertEquals(mailService.getFrom(), captor.getValue().getFrom());
        assertEquals("test@gmai.com", captor.getValue().getTo()[0]);
        assertEquals(mailService.getSubject(), captor.getValue().getSubject());
        assertEquals("test message", captor.getValue().getText());
    }
}
```

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

The diagram illustrates the class structure for generating PDF documents. It features two main classes: **PdfBuilder** and **PdfTableBuilder**. The **PdfBuilder** class contains methods for adding text, code, images, and tables. The **PdfTableBuilder** class extends **PdfBuilder** and adds methods for building tables. Both classes depend on the **com.itextpdf** library.

```
new PdfBuilder()
    .paragraphEmphasized(title)
    .line()
    .paragraph(paragraph)
    .paragraph("Lorem ipsum dolor sit amet, sea ea dico suas iracundia, iediocritatem."
        + "Altera qualisque eum at, eam id animal appareat, veri prompta duo ne. Choro civibus,"
        + " ei nam brute graecis, quo ea inimicus interpretaris.")
    .line()
    .barCode("8015187008499").line()
    .qrCode("BETCA: Spring").line()
    .image("logoMiw.png").line()
    .table(15, 90, 15, 25, 35, 15)
    .tableCell("1", "2", "3", "4", "5", "6")
    .tableCell("1", "2", "3", "4", "5", "6")
    .tableCell("1", "2", "3", "4", "5", "6")
    .tableColumnsHeader("", "Desc.", "Ud.", "Dto.%", "€", "E.")
    .tableColspanRight("TOTAL")
    .closeTable().line()
    .build();
```

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



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

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`**

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.

Swagger

- Proyecto: <http://springfox.github.io/springfox>
- Documentación del API automátizado:
 - CLI > mvn spring-boot:run
 - http://localhost:8080/swagger-ui.html

```
<!-- Swagger ===== -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

The screenshot shows a code editor with Java code. The code defines a class named **SwaggerConfig** which implements the **Docket** interface. The class uses annotations from the **Swagger** library to enable Swagger 2 support.

```
package config [  swagger ]
```

```
config
```

```
  «@Configuration»  
  «@EnableSwagger2»  
  SwaggerConfig  
  «@Bean» +api() : Docket
```

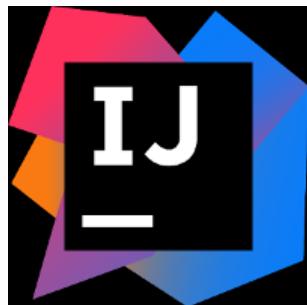


Swagger

The screenshot shows a web browser window displaying the Swagger UI at localhost:8080/swagger-ui.html. The title bar says "Swagger UI". The main content area features a green header bar with the "swagger" logo and a dropdown menu set to "default". Below this, the title "Máster en Ingeniería Web. Universidad Politécnica de Madrid" is displayed in large, bold, dark blue font. A subtitle "[Base URL: localhost:8080/]" and a link "<http://localhost:8080/v2/api-docs>" are shown. A note "BETCA. Back-end con Tecnologías de Código Abierto (SPRING).<https://github.com/miw-upm/betca-tpv-spring>" is also present. On the right side of the page, there is a "Authorize" button with a lock icon. The main content area lists several API resources:

- admin-resource** Admin Resource >
- basic-error-controller** Basic Error Controller >
- exception-resource** Exception Resource >
- jwt-resource** Jwt Resource >
- pdf-resource** Pdf Resource >
- security-resource** Security Resource >
- Models** >

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



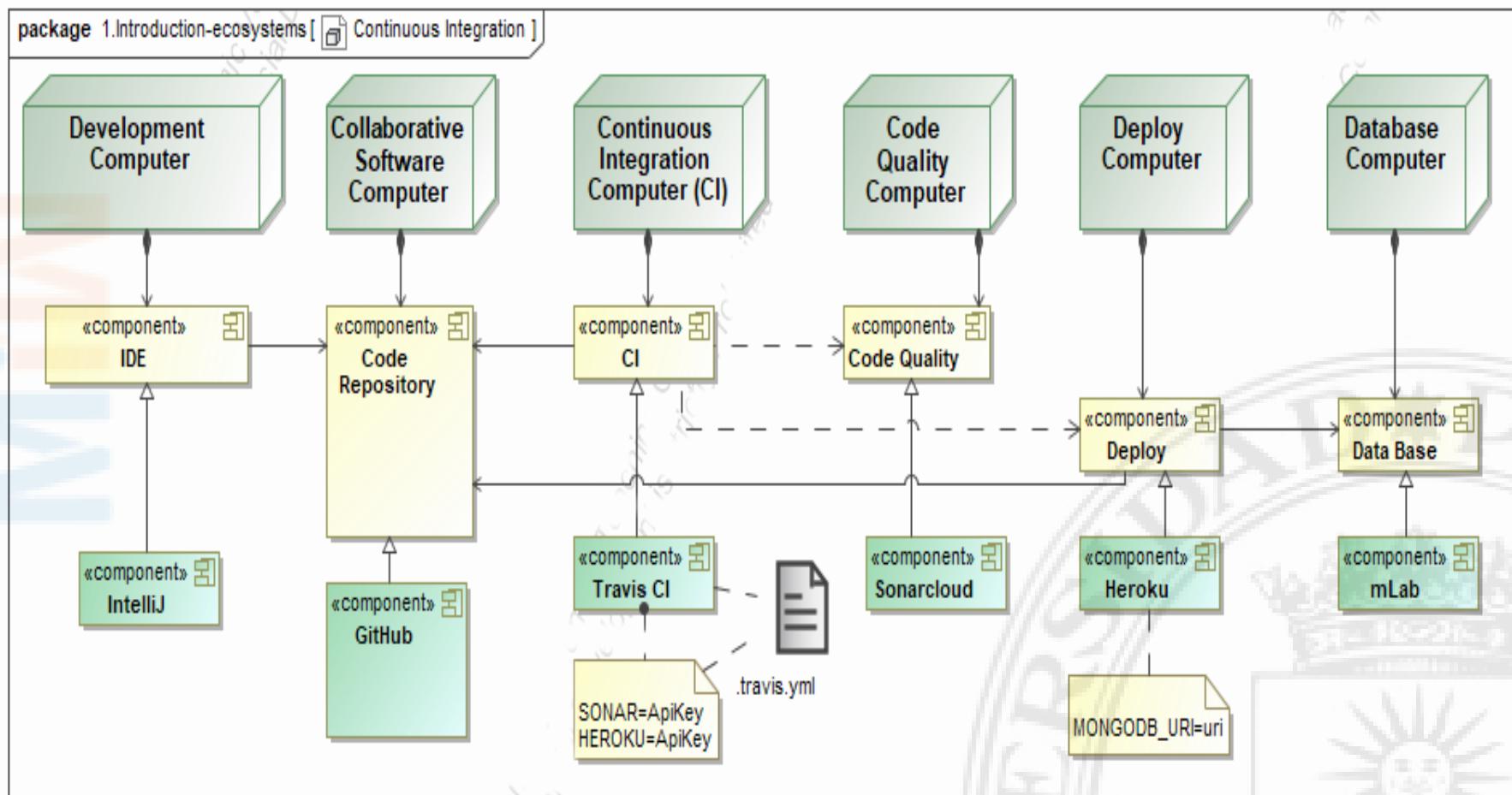
Travis CI sonarcloud



HEROKU



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



HEROKU

- Heroku (www.heroku.com) es una plataforma *PaaS* (*Plataforma como Servicio*) comercializada por *Salesforce*
- Es posible desarrollar prácticamente con cualquier lenguaje de programación: *Java, Ruby, PHP, NodeJS, etc.*
- Las aplicaciones Heroku
 - Están preparadas para poder tener una muy alta escalabilidad. Esto se consigue utilizando los procesos *dyno*, que son instancias independientes que *Heroku* puede levantar y bajar en cualquier momento (en función de la carga de trabajo)
 - Cada instancia contiene una copia del código fuente de la aplicación
 - Es la plataforma Heroku la que se ocupa de colocar un servidor web delante de los *dyno*, de hacer las comunicaciones SSL, etc.
 - Los procesos *dyno* pueden tener una vida muy efímera (por ejemplo, para atender tan solo una petición) con lo cual no es recomendable almacenar información en memoria, ni tampoco escribir ningún fichero en disco
- Cliente *Heroku CLI* (<https://devcenter.heroku.com/articles/heroku-cli>)
 - **>heroku version**
 - **>heroku login**
 - **>heroku logs --app=<proyecto> -n 100** //logs pasados, los 100 últimos
 - **>heroku logs --tail --app=<proyecto>** //logs en tiempo real

HEROKU (despliegue)

- En la Web de Heroku: <https://dashboard.heroku.com>
 1. Crear una aplicación en Heroku (Botón >>**New**)
 - *name: betca-spring*
 - *zone: Europe*
 2. Configurar Heroku
 - >>**Personal>\${proyecto}>Settings**
 - Es automático. Se puede elegir dependencias (>>**Add buildpack**): **heroku/java**
 3. Obtener el API-key: >>**Personal>Account settings**
- En el proyecto, añadir el comando *deploy* en el fichero *.travis.yml*
- En la web de TRAVIS-CI, crear una variable de entorno (*HEROKU*) con la *ApiKey*
- Subir a **Git-hub** la rama *master*, ello dispara la integración continua con *Travis-CI*, y si no existen errores, se despliega automáticamente en *Heroku*, la ruta del despliegue será:
 - <https://<proyecto>.herokuapp.com/>
 - Swagger: <https://<proyecto>.herokuapp.com/swagger-ui.html>

```
deploy:  
  provider: heroku  
  api_key:  
    secure: $HEROKU  
  # app: template-spring5  
  # on:  
  #   repo: miw-upm/template-spring5  
  #   branch: master
```

HEROKU (despliegue) con BD

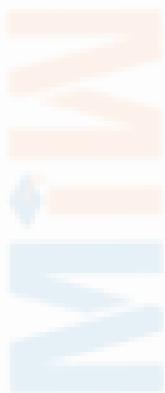
- La conexión entre *Heroku* y *mLab*, es automática al añadir en *Heroku* el *add-ons* de *mLab*
- En el proyecto:
 1. En el fichero de propiedades de producción, definir la propiedad:
 - `spring.data.mongodb.uri=${MONGODB_URI}`
- Se puede realizar manualmente
 - En la Web de **mLab**, crear una database, y en detalles, añadir un usuario con contraseña. Copiar la uri, similar a:
 - `mongodb://<user>:<pass>@ds12345.mlab.com:13615/database`
 - En la Web de **Heroku**, añadir BD:
 - `>>Personal>${proyecto}>Resources, >>add-ons: mLab`
 - `>>Personal>${proyecto}>Settings`, definir una variable de entorno (`MONGODB_URI`) con la uri de la bases de datos

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

✍ CRUD

- Realizar un CRUD completo del ejemplo asignado en la práctica de APAW
- Respetar la estructura de carpetas



Aplicación Web: TPV

A screenshot of a web browser window titled "BetcaTpvAngular". The address bar shows the URL <https://betca-tpv-angular.herokuapp.com/welcome>. The page has a blue header with the "MiW" logo on the left, the word "TPV" in the center, and a "Login" button on the right. Below the header, the text "Welcome to POS" is displayed.

Welcome to POS

Universidad Politécnica de Madrid. Máster en Ingeniería Web API: <https://betca-tpv-spring.herokuapp.com/api/v0> Version: 1.3.0 (In Production)

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

1 #	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 magnifying glass icon and a value "7". At the bottom, there are two buttons: "Checkout €" and "Budget".

Advance search

Articles family view

Universidad Politécnica de Madrid. Máster en Ingeniería Web

API: <https://betca-tpv-spring.herokuapp.com/api/v0>

Version: 1.3.0 (In Production)