

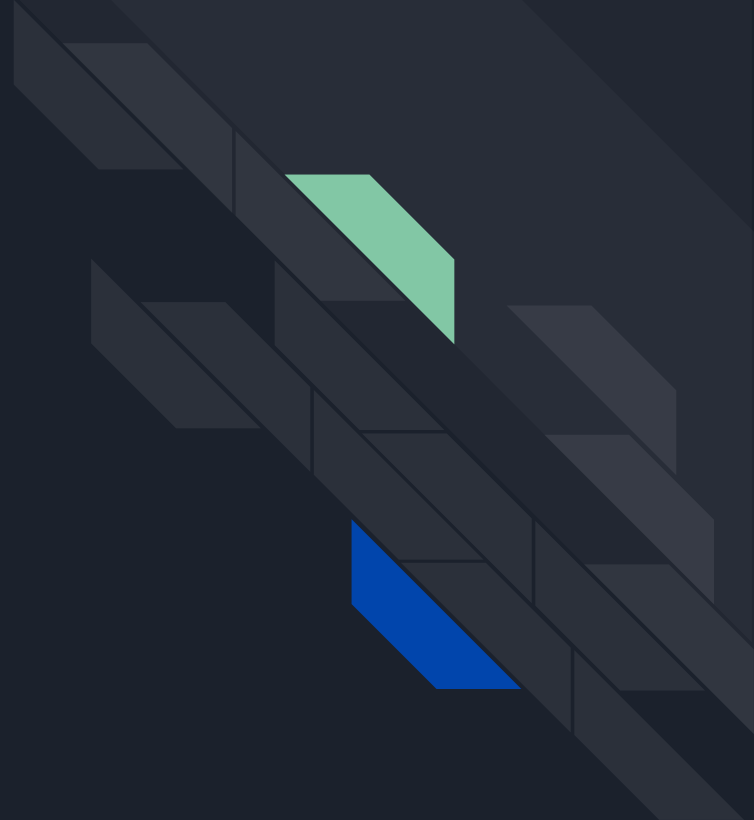


Introducción a Spring

Tema 1. Desarrollo de APIs REST
2º DAM. Curso 2023-24



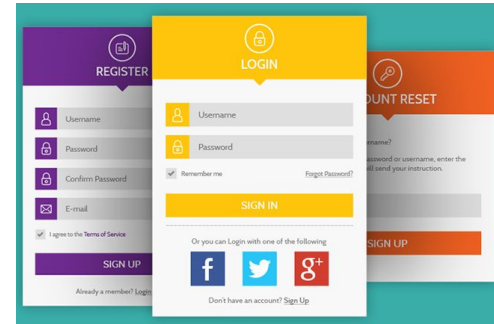
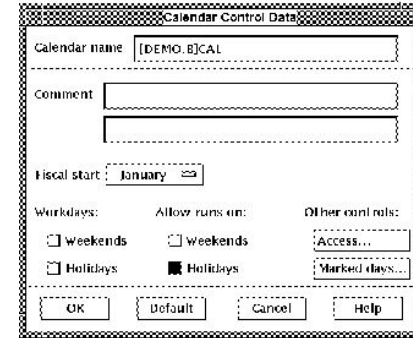
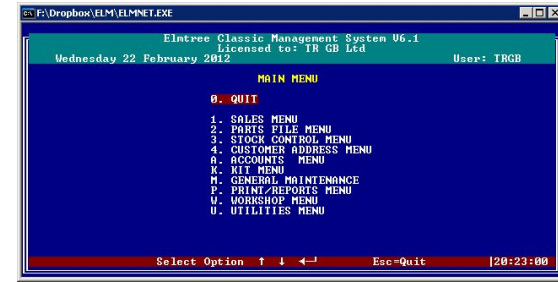
INVERSIÓN DE CONTROL E INYECCIÓN DE DEPENDENCIAS



Inversión de Control

- Principio de diseño (o patrón)
- El objetivo es conseguir desacoplar objetos.

Desacoplar: que sean poco dependientes entre sí, por si alguno de los objetos tiene que cambiar.





Principio de Hollywood

*No nos llames, nosotros
te llamaremos a ti.*





Inversión de Control

- Propuesto por [Martin Fowler](#).
- *Dejar que sea otro el que controle el flujo del programa (por ejemplo, un framework)*

```
#ruby
puts 'What is your name?'
name = gets
→ process_name(name)
puts 'What is your quest?'
quest = gets
→ process_quest(quest)
```

Ejemplos propuestos por Martin Fowler

```
require 'tk'
root = TkRoot.new()
name_label = TkLabel.new() {text "What is Your Name?"}
name_label.pack
name = TkEntry.new(root).pack
name.bind("FocusOut") {process_name(name)}
quest_label = TkLabel.new() {text "What is Your Quest?"}
quest_label.pack
quest = TkEntry.new(root).pack
quest.bind("FocusOut") {process_quest(quest)}
Tk.mainloop()
```



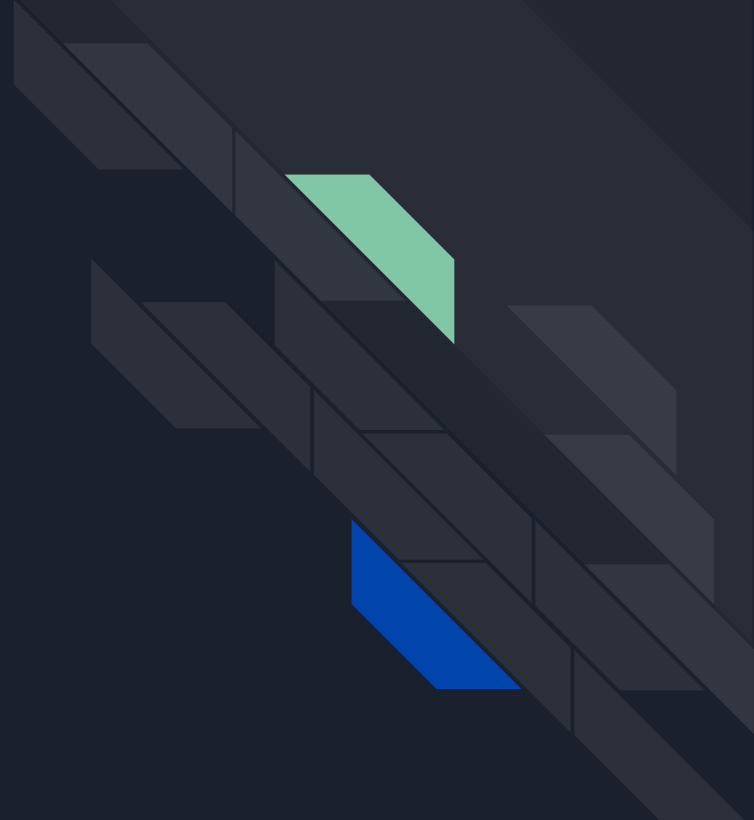
Inversión de Control

Ralph Johnson and Brian Foote
Journal of Object-Oriented Programming
Junio/Julio 1988

*Una característica importante de un **framework** es que los **métodos definidos por el usuario** para adaptar el mismo a menudo **serán llamados desde el framework**, en lugar de desde el código de aplicación del usuario. El framework a veces **desempeña el papel de programa principal** en la coordinación y secuenciación de actividad de la aplicación. Esta **inversión de control** proporciona al framework la posibilidad de servir como un **esqueleto extensible**. El usuario proporciona métodos que adaptan los algoritmos genéricos.*

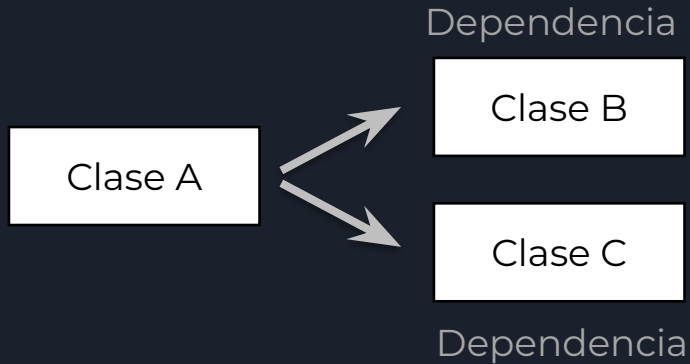


INYECCIÓN DE DEPENDENCIAS

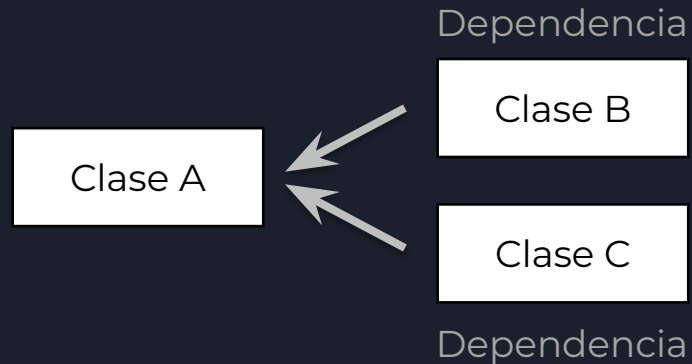


Inyección de dependencias

Es una forma de inversión de control (IoC)



Modelo
tradicional



Inversión de
Control con
inyección de
dependencias



Motivación para el uso de DI

```
public class MovieLister {  
    public List<Movie> moviesDirectedBy(String director)  
    {  
        List<Movie> allMovies = finder.findAll();  
        List<Movie> result = new ArrayList<>();  
  
        for(Movie m : allMovies) {  
            if (m.getDirector().equals(director))  
                result.add(m);  
        }  
  
        return Collections.unmodifiableList(result);  
    }  
}
```



Motivación para el uso de DI

```
public interface MovieFinder
{
    List<Movie> findAll();
}

public class MovieLister {

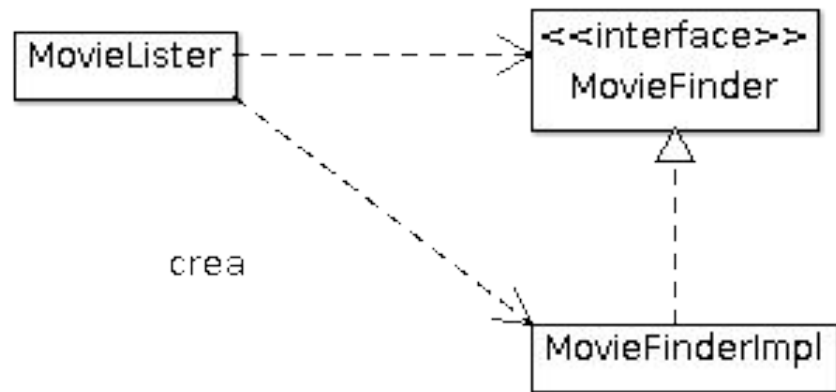
    private MovieFinder finder;

    public MovieLister() {
        finder = new MovieFinderImpl();
    }
    //...
}
```

[Ver ejemplo en el repositorio](#)

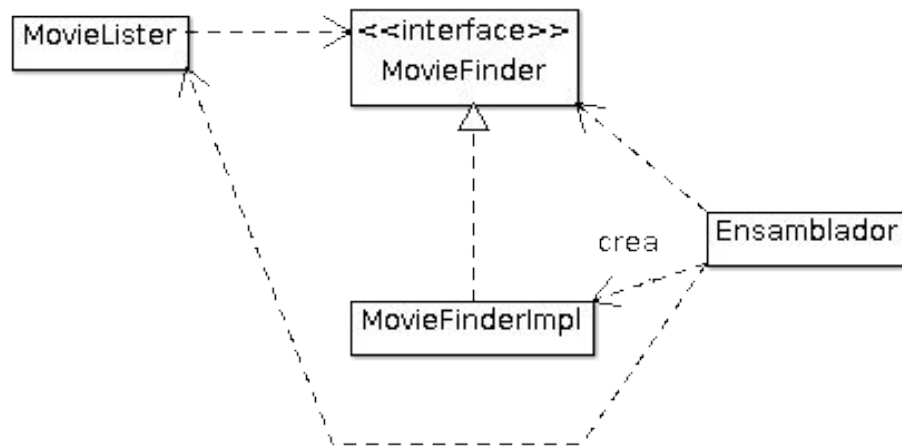
Motivación para el uso de DI

Si el objeto (*MovieLister*) que tiene la **dependencia** (de *MovieFinder*) se encarga de **crear una instancia** (de *MovieFinderImpl*) el **acoplamiento** entre estas clases es **muy alto**.



Motivación para el uso de DI

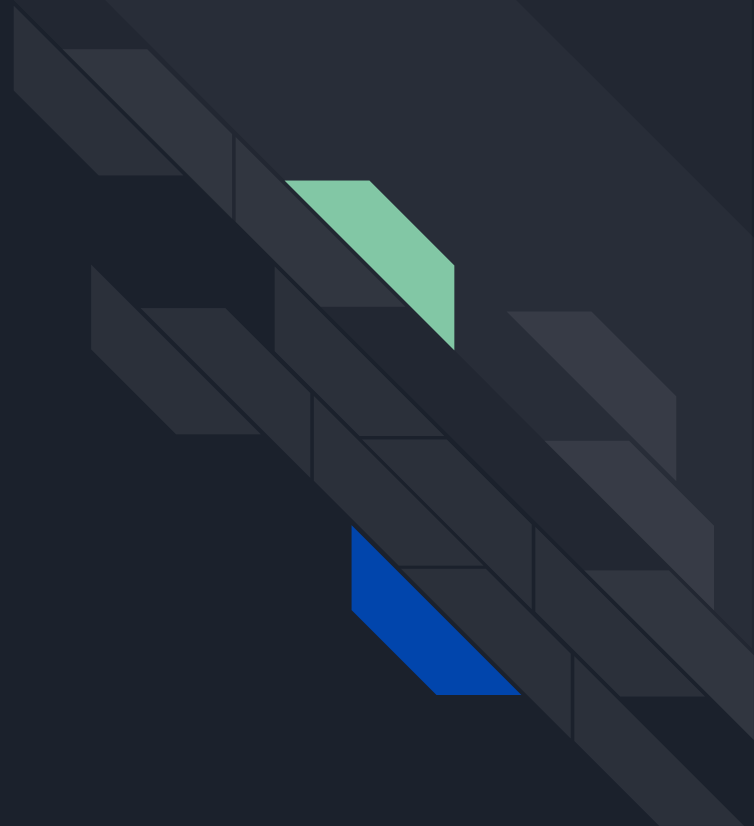
La idea básica de la Inyección de Dependencia es tener **un objeto separado, un ensamblador**, que **rellena** un campo **en** la clase *MovieLister* con una **implementación apropiada** para la interfaz *MovieFinder*, resultando un **diagrama de dependencia como el de la figura**.



Este *ensamblador* es nuestro proto-contenedor de inversión de control.



¿QUÉ ES **SPRING**?






¿Qué es Spring?

“

Spring es un **framework** de código abierto para la creación de **aplicaciones empresariales** Java, con soporte para Groovy y Kotlin; tiene estructura modular y una gran flexibilidad para implementar diferentes tipos de arquitecturas según las necesidades de la aplicación.

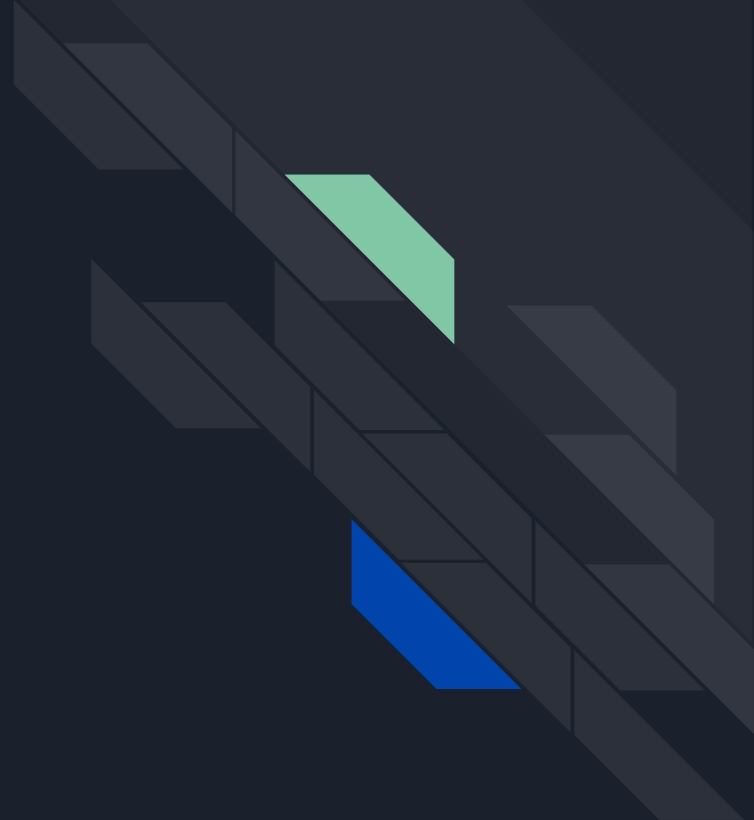


Aunque cuando digamos **Spring** también podremos referirnos a...

- Proyecto Spring Framework
 - Núcleo de toda la familia Spring
- Familia de proyectos Spring
 - Diversos módulos que abarcan múltiples necesidades
- Entorno de desarrollo Spring Tool Suite
 - Basado en Eclipse



PROYECTOS Y MÓDULOS DE SPRING



Familia de proyectos de Spring



**Spring
Framework**



Spring Boot



Spring Data



**Spring
Security**



Spring Cloud



**Spring
HATEOAS**



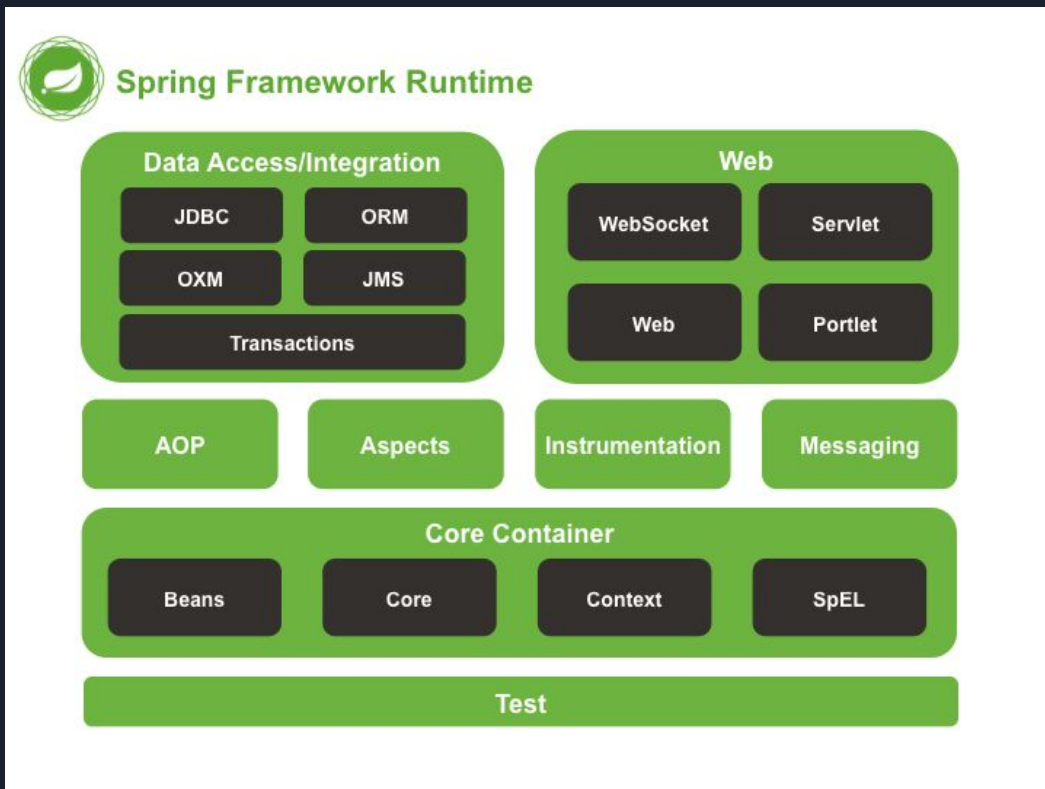
Spring Batch



**Spring for
Android**

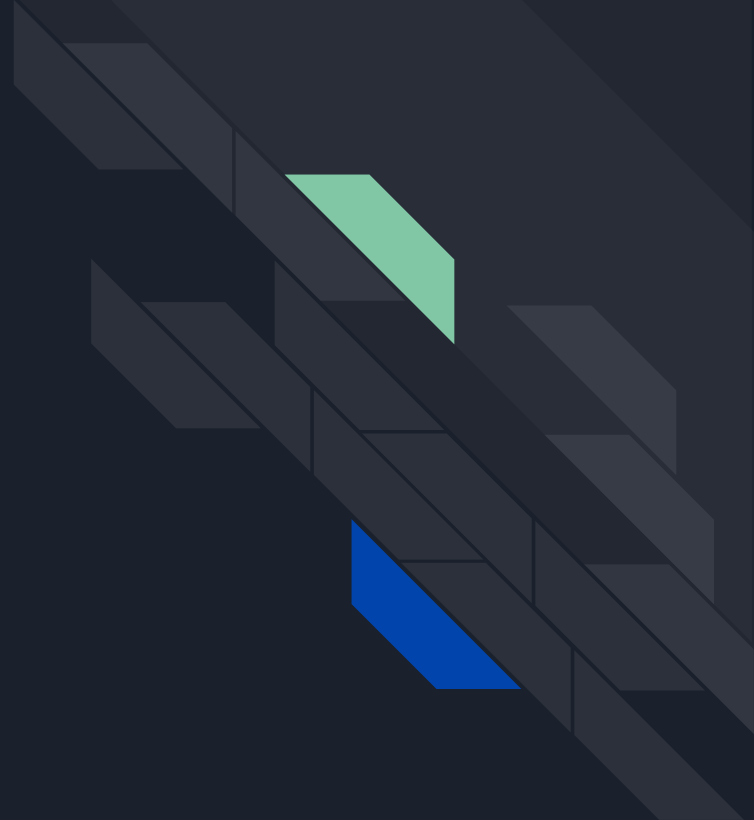
...

Módulos más usuales de **Spring Framework**





VERSIONES DE SPRING





Versión actual: 6

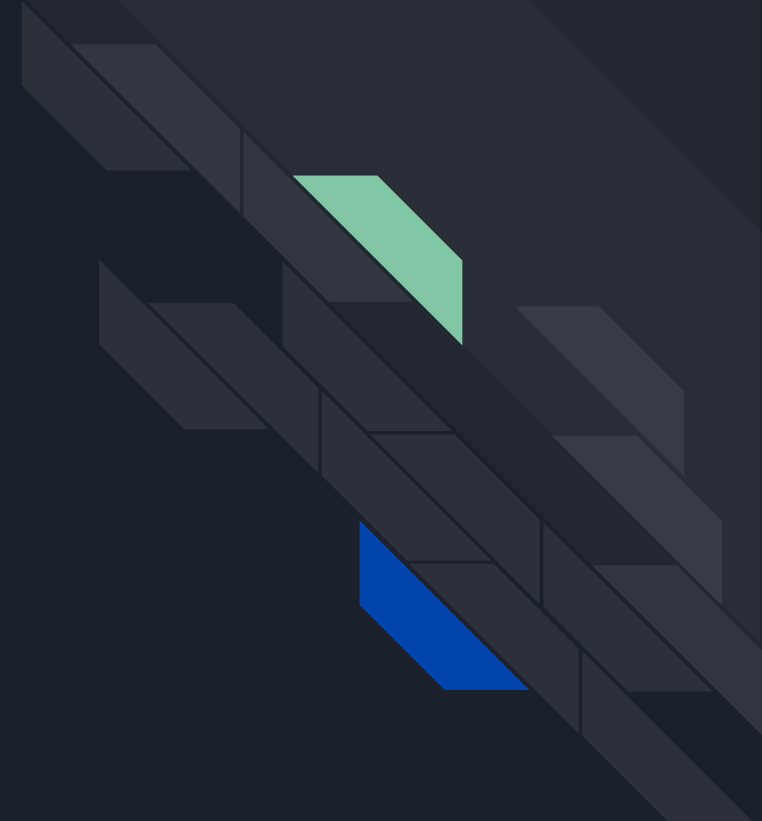
- Versión actual: 6.X

Siglas:


- **GA:** General Availability
- **CURRENT:** Versión actual
- **SNAPSHOT:** Versión aún no liberada, y susceptible de tener cambios.
- **RC:** Release Candidate, versión candidata a ser liberada.






INSTALACIÓN DEL **ENTORNO**




IntelliJ IDEA CE (Community Edition)

 **IntelliJ IDEA**

Para desarrolladores Para equipos Para aprender Soluciones Tienda   

Novidades Funcionalidades Recursos Comprar [Descargar](#)



Versión: 2021.2.1
Build: 212.5080.55
24 de agosto de 2021
[Notas de lanzamiento](#)

[Requisitos del sistema](#)
[Instrucciones de instalación](#)
[Otras versiones](#)
[Software de terceros](#)

Descargar IntelliJ IDEA

Windows macOS Linux

Ultimate

Para desarrollo web y empresarial

[Descargar](#) [.dmg \(Intel\)](#)


Prueba gratis de 30 días




Community

Para desarrollo de JVM y Android

[Descargar](#) [.dmg \(Intel\)](#)

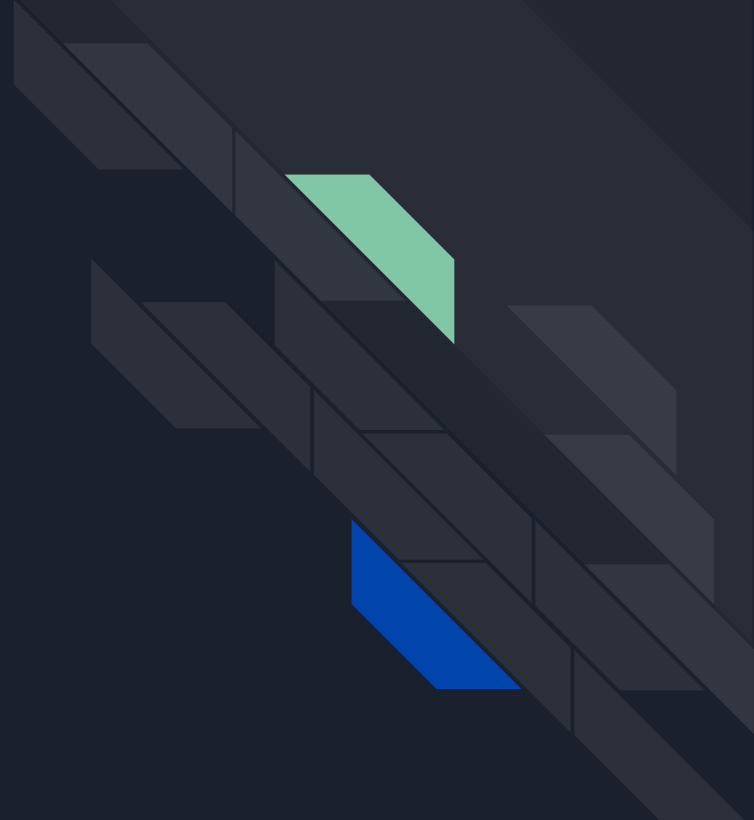
Gratis, creado en código abierto

 IntelliJ IDEA está disponible para Intel y Apple Silicon

	IntelliJ IDEA Ultimate	IntelliJ IDEA Community Edition 
Java, Kotlin, Groovy, Scala	✓	✓
Android 	✓	✓
Maven, Gradle, sbt	✓	✓
Git, GitHub, SVN, Mercurial, Perforce	✓	✓
Depurador	✓	✓
Docker	✓	✓
Herramientas de generación de perfiles 	✓	



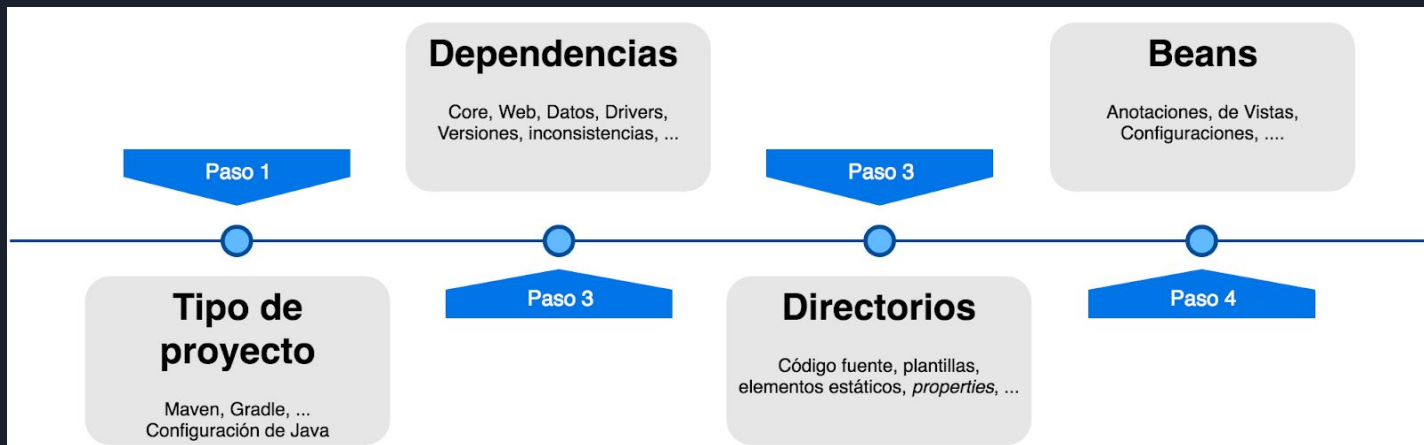
INTRODUCCIÓN A **SPRING BOOT**





Configurar un proyecto web
con Spring es ciertamente
complicado.

Pasos a seguir





UFFFFFFFFF!!!!!!

SPRING BOOT al rescate





¿Qué hace **Spring Boot** ?

“

Facilita la creación de aplicaciones basadas en Spring **independientes** y **listas para usar**, con un **mínimo esfuerzo**.



Características

- Creación de aplicaciones Spring independientes
- Con servidor embebido (Tomcat, Jetty, ...)
- Dependencias iniciales que facilitan la configuración de componentes.
- Configuración automática de librerías de 3ºs allá donde sea posible.
- Sin generar código o configuración XML



Convención sobre configuración

“

La **convención sobre configuración** es un patrón de diseño de software usado por muchos frameworks que trata de **minimizar el número de decisiones** que un programador tiene que tomar al usar dicho framework, pero **sin perder la flexibilidad**.



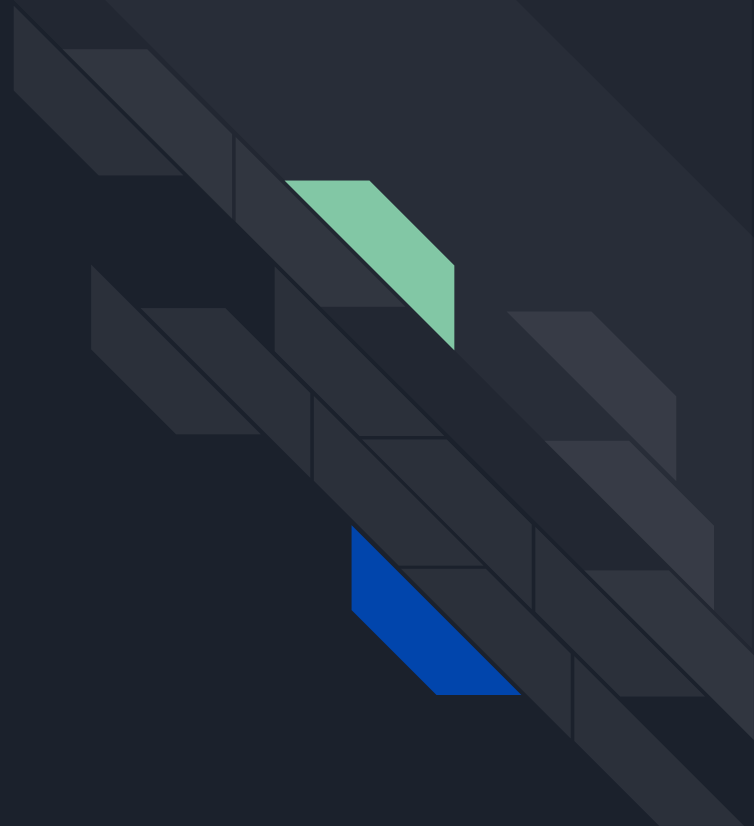
Convención sobre configuración

- Ejemplo
 - Una clase llamada *Ventas* se almacenará en una tabla en base de datos llamada *Ventas*.
 - Si quiero que la tabla se llame *VentaProductos*, entonces lo puedo configurar explícitamente.




PRIMER PROYECTO CON **SPRING BOOT**

Añadiendo elementos a la coctelera...



1) Spring Starter (<https://start.spring.io/>)



spring initializr

Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M2) ☐ 2.5.5 (SNAPSHOT) ☒ 2.5.4

☐ 2.4.11 (SNAPSHOT) ☐ 2.4.10

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 16 ☒ 11 ☐ 8

Dependencies

[ADD DEPENDENCIES...](#) ⌘ + B

No dependency selected

GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

SHARE...

2. Asistente

- Maven
- Java 17
- Jar
- Cambiar *GroupId* y *ArtifactId*
- Ajustar *Description* y *Package*



Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M2) ☐ 2.5.5 (SNAPSHOT) ☒ 2.5.4
☐ 2.4.11 (SNAPSHOT) ☐ 2.4.10

Project Metadata

Group

Artifact

Name

Description

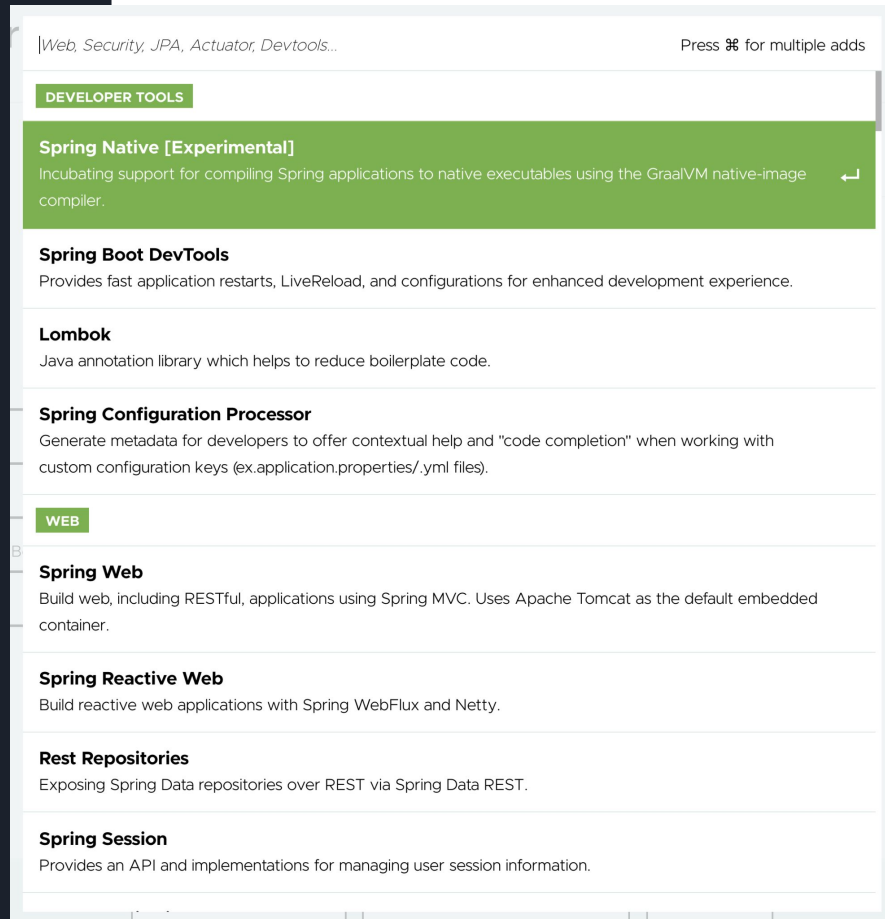
Package name

Packaging ☒ Jar ☐ War

Java ☐ 16 ☒ 11 ☐ 8

3. Dependencias

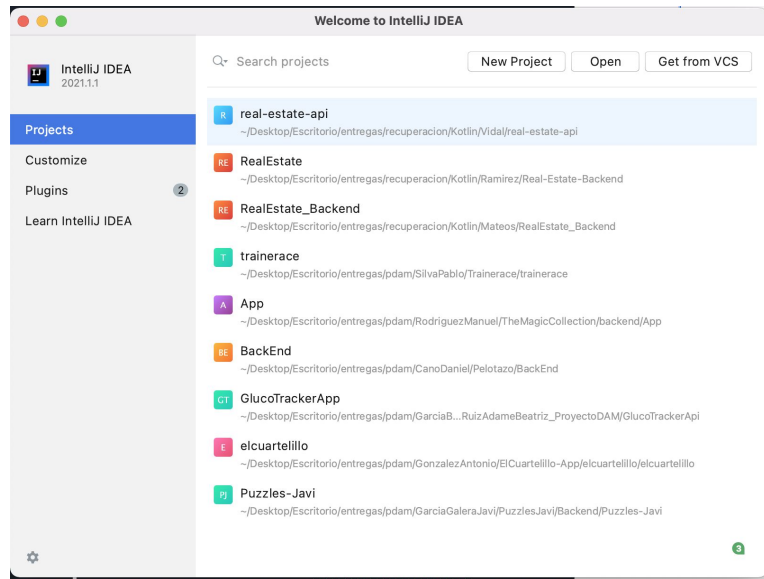
- Dependencias Maven
- Las de Spring son especiales (*starters*)
- En este primer ejemplo solamente añadimos Lombok.
- Spring Core va de serie.



4. Finalizar

- *Generate* y Se descarga .zip
- Hay que descomprimir
- Ubicar la carpeta en lugar adecuado ¿repositorio?
- Importar desde IDEA CE

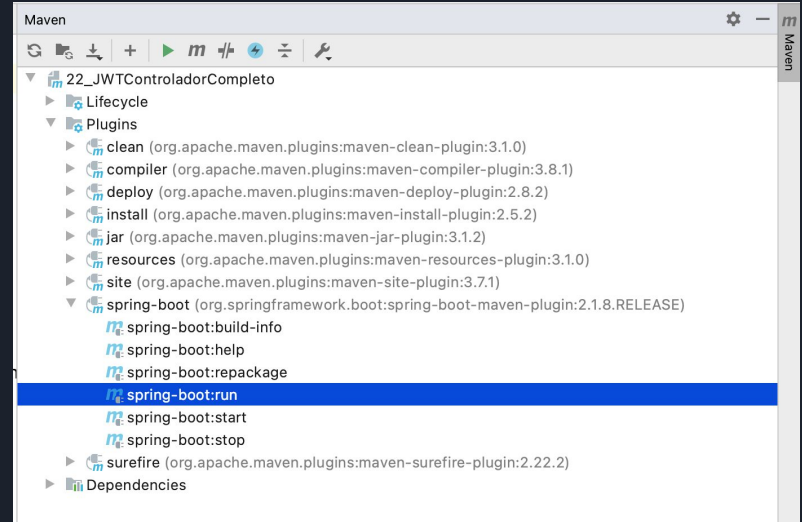
GENERATE ⌘ + ↵



¿Cómo se ejecuta este proyecto?

También desde la consola con
Maven

mvn spring-boot:run



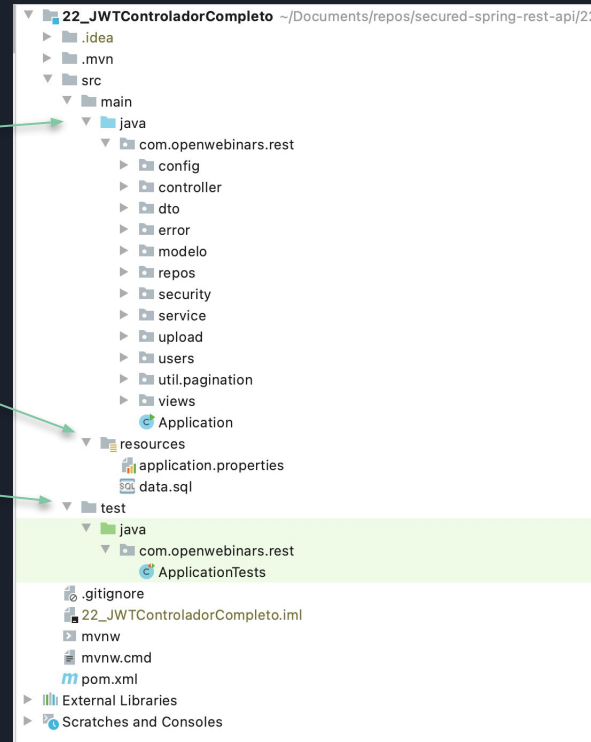
Estructura de un proyecto **Spring Boot**

Código fuente

Nuestro código de aplicación

Recursos: imágenes, plantillas,
ficheros de propiedades, sql, ...

Tests



Estructura de un proyecto **Spring Boot**

Librerías



```
22_JWTControladorCompleto ~/Documents/repos/secured-spring-rest-api/22_
External Libraries
< 14 > /Library/Java/JavaVirtualMachines/jdk-14.jdk/Contents/Home
Maven: antlr:antlr:2.7.7
Maven: ch.qos.logback:logback-classic:1.2.3
Maven: ch.qos.logback:logback-core:1.2.3
Maven: com.fasterxml.jackson.core:jackson-annotations:2.9.0
Maven: com.fasterxml.jackson.core:jackson-core:2.9.9
Maven: com.fasterxml.jackson.core:jackson-databind:2.9.9.3
Maven: com.fasterxml.jackson.datatype:jackson-datatype-jdk8:2.9.9
Maven: com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.9.9
Maven: com.fasterxml.jackson.module:jackson-module-parameter-names:2.9.9
Maven: com.fasterxml.classmate:1.4.0
Maven: com.google.guava:guava:20.0
Maven: com.h2database:h2:1.4.199
Maven: com.jayway.jsonpath:json-path:2.4.0
Maven: com.vaadin.external.google:android-json:0.0.20131108.vaadin1
Maven: com.zaxxer:HikariCP:3.2.0
Maven: io.jsonwebtoken:jjwt-api:0.10.7
Maven: io.jsonwebtoken:jjwt-impl:0.10.7
Maven: io.jsonwebtoken:jjwt-jackson:0.10.7
Maven: io.springfox:springfox-core:2.9.2
Maven: io.springfox:springfox-schema:2.9.2
Maven: io.springfox:springfox-spi:2.9.2
Maven: io.springfox:springfox-spring-web:2.9.2
Maven: io.springfox:springfox-swagger2:2.9.2
Maven: io.springfox:springfox-swagger-common:2.9.2
Maven: io.springfox:springfox-swagger-ui:2.9.2
Maven: io.swagger:swagger-annotations:1.5.21
Maven: io.swagger:swagger-models:1.5.21
Maven: javax.activation:javax.activation-api:1.2.0
Maven: javax.annotation:javax.annotation-api:1.3.2
Maven: javax.persistence:javax.persistence-api:2.2
Maven: javax.transaction:javax.transaction-api:1.3
Maven: javax.validation:validation-api:2.0.1.Final
Maven: javax.xml.bind:jaxb-api:2.3.1
Maven: junit:junit:4.12
Maven: net.bytebuddy:byte-buddy:1.9.16
```



Estructura de un proyecto **Spring Boot**

Clase Main

```
@SpringBootApplication ← Anotación mágica que autoconfigura nuestra aplicación
public class DemoApplication {

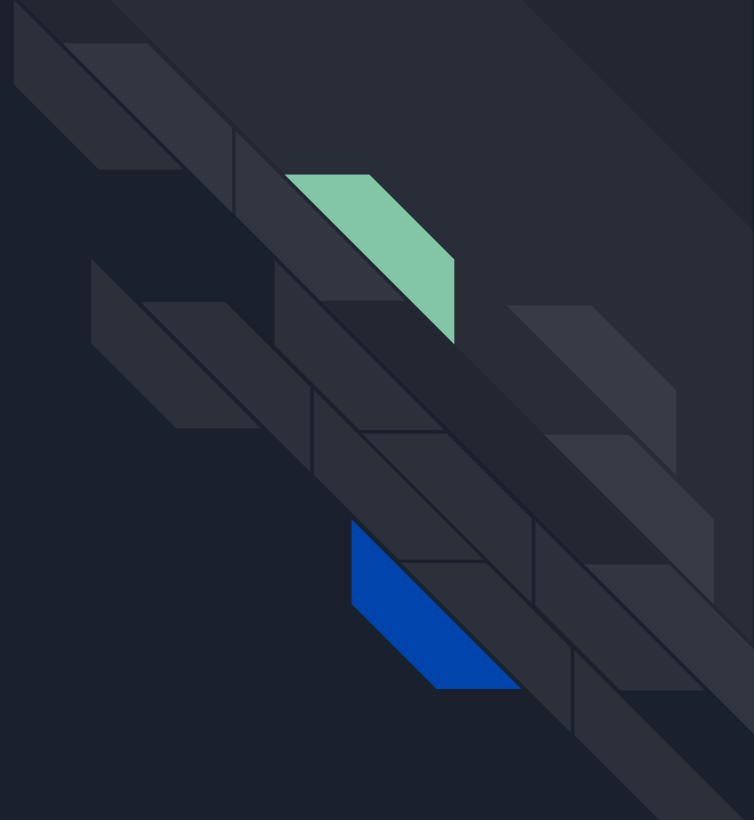
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```



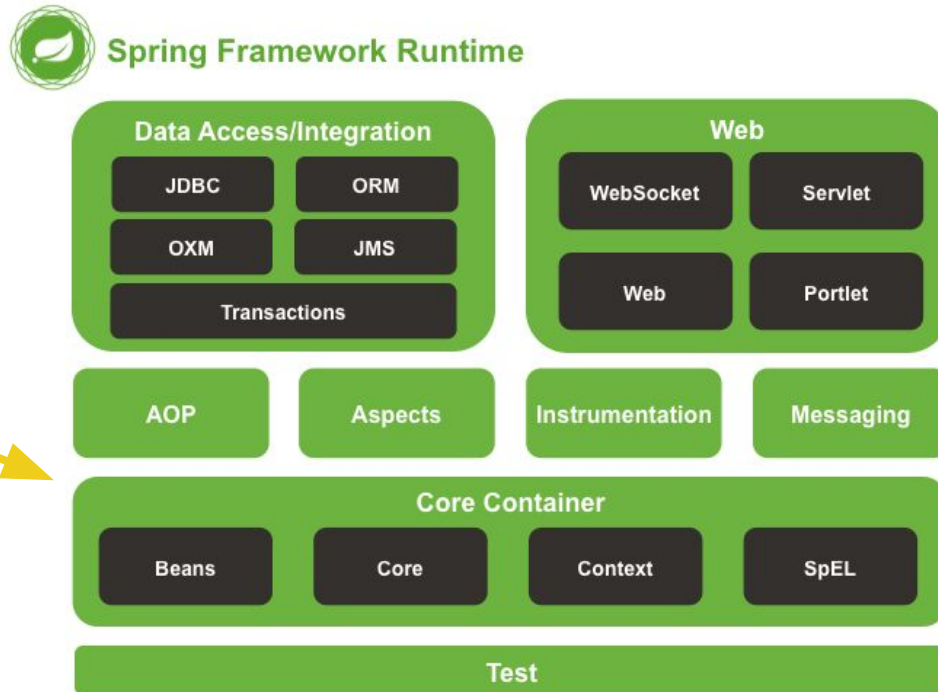
Comienza a ejecutar Spring, el servidor (si es web) y tras eso nuestra aplicación.



INVERSIÓN DE CONTROL CON SPRING



Nos situamos...

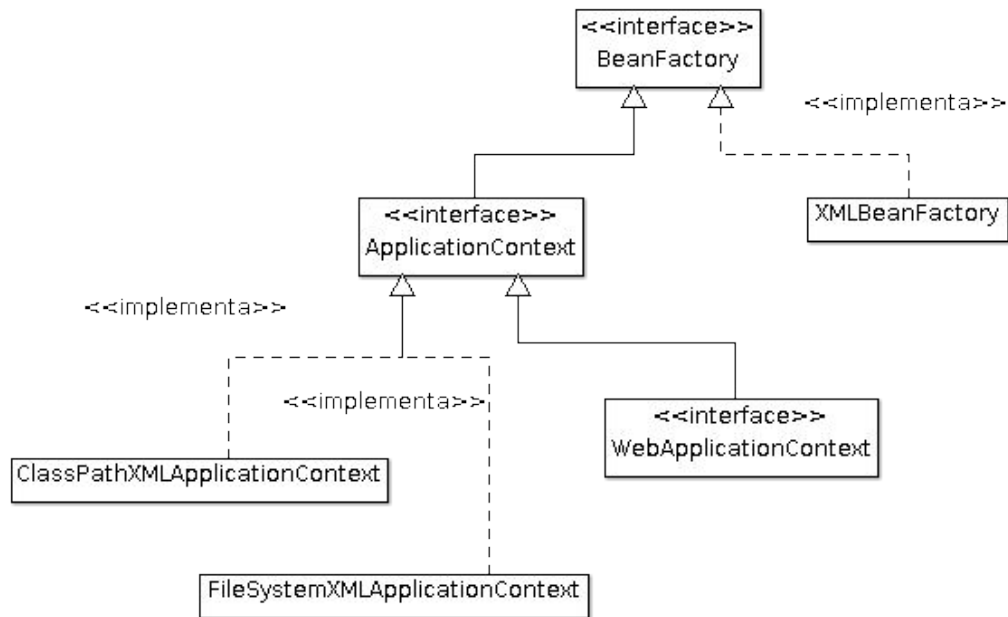




La base del *IoC container*

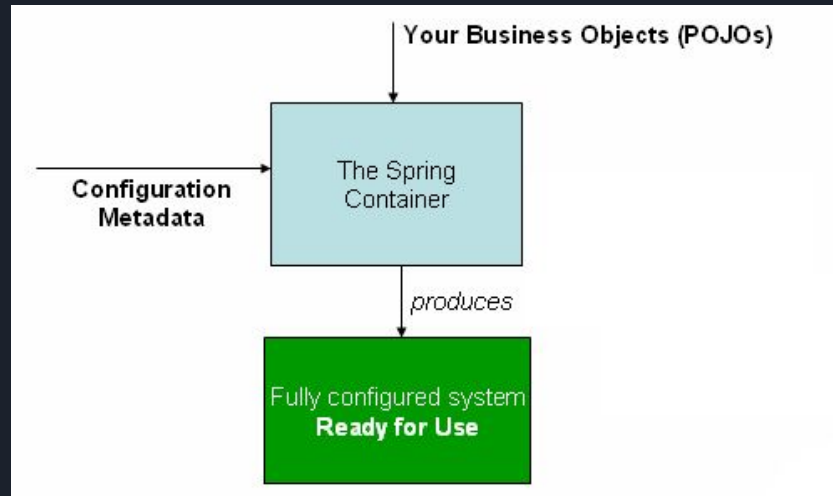
- Paquetes
 - *org.springframework.beans*
 - *org.springframework.context*
- Los elementos más básicos
 - *BeanFactory*: lo elemental para poder manejar cualquier ~~bean~~ objeto.
 - *ApplicationContext*: superset del anterior. Añade AOP, manejo de recursos, internacionalización, contextos específicos, ...

La base del *IoC container*



BEANS

Se trata de un objeto (cualquiera) gestionado por nuestro contenedor de inversión de control.



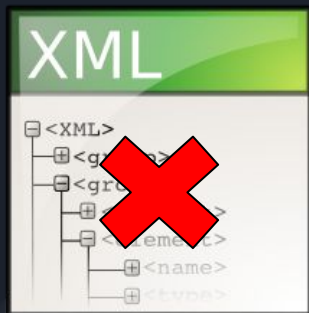
Podríamos decir que es como un objeto *empoderado*.



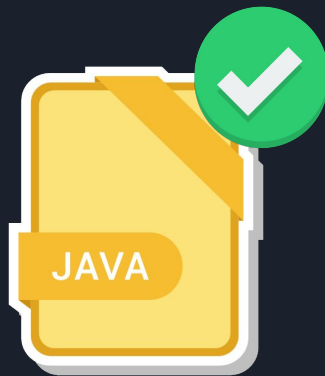
¿**CÓMO** SE DEFINEN
ESOS **BEANS**?



¿CÓMO SE **DEFINEN** ESOS BEANS?



XML
(*legacy*)



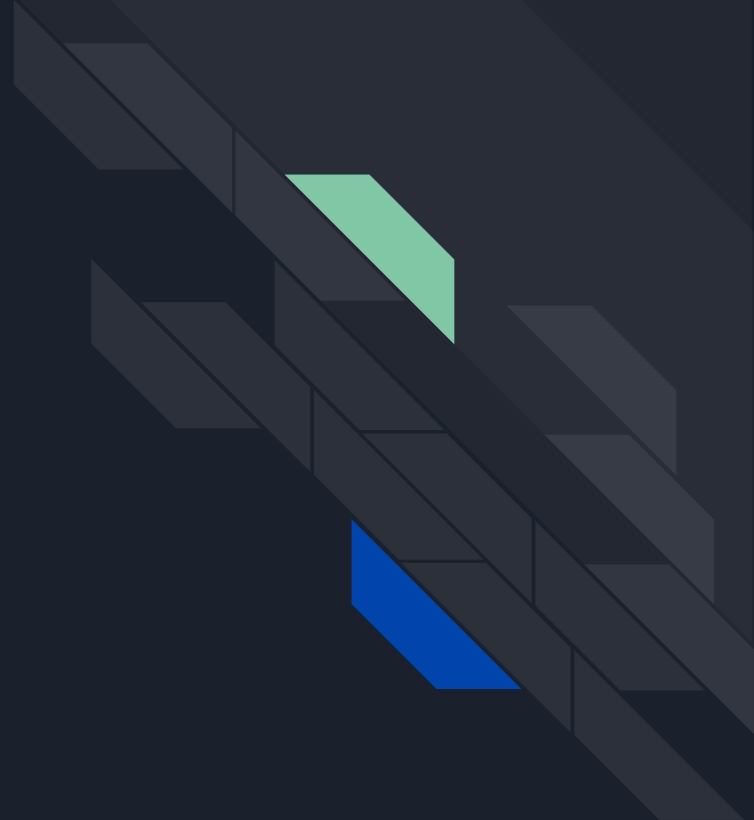
JavaConfig



Anotaciones



DEFINICIÓN DE BEANS CON **ANOTACIONES**



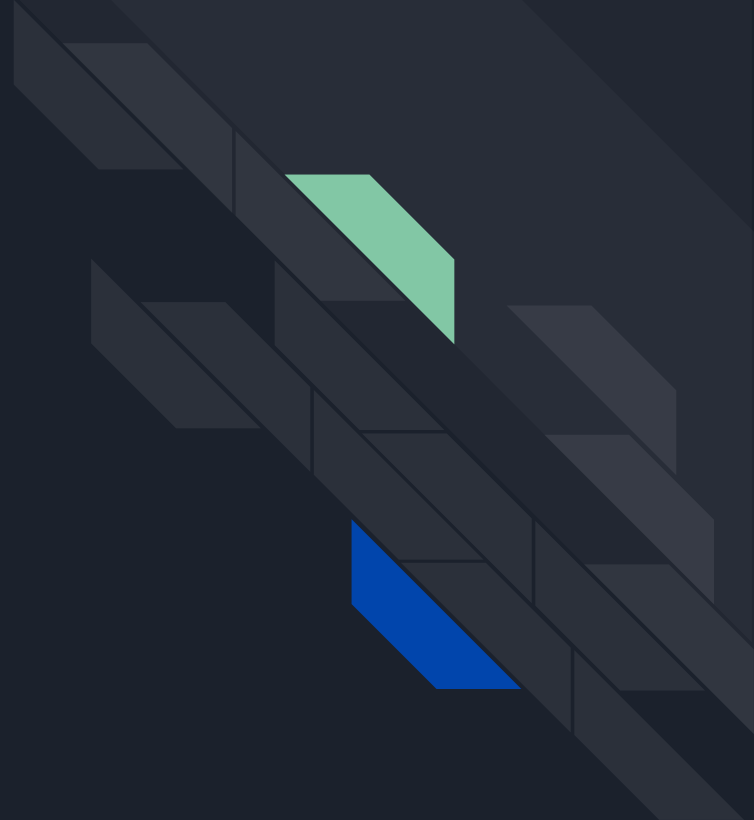


ESTEREOTIPOS

- **@Component:** Estereotipo básico; los demás son derivados de él.
- **@Service:** orientado a las clases servicio, lógica de negocio, ...
- **@Repository:** clases de acceso a datos (DAO, bases de datos, ...)
- **@Controller:** clases que sirven para gestionar las peticiones recibidas y producir respuestas.



DEFINICIÓN DE BEANS CON **JAVACONFIG**





DEFINICIÓN DE BEANS CON **JAVA**CONFIG

- Útil cuando no tenemos acceso al código fuente de las clases.
- **@Configuration** y **@Bean**

```
@Configuration  
public class Configuracion {
```

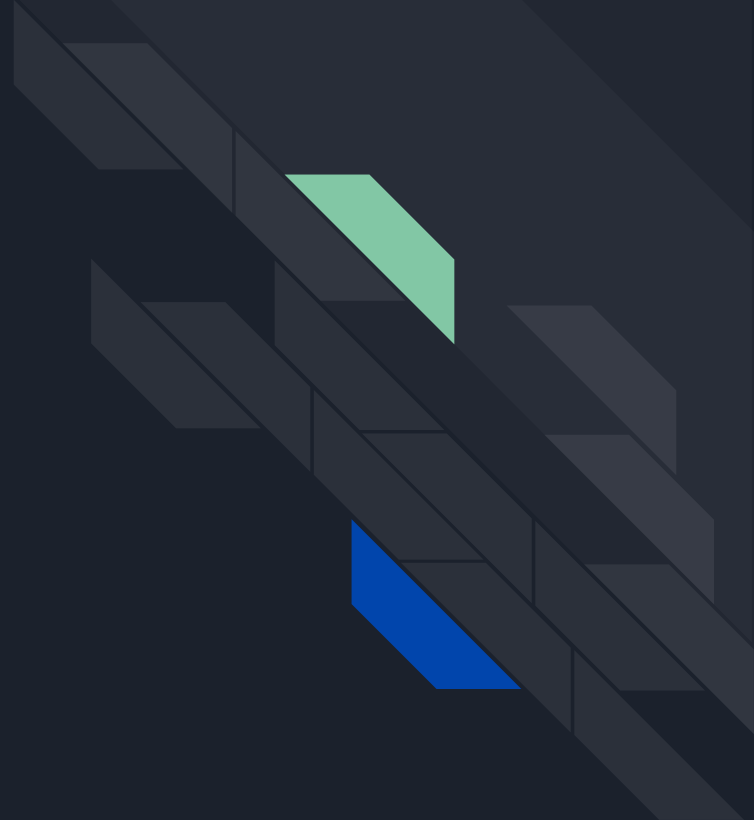
```
    @Bean  
    @Primary  
    public Saludador spanishSaludador() {  
        return new SpanishSaludador();  
    }
```

```
    @Bean  
    public Saludador englishSaludador() {  
        return new EnglishSaludador();  
    }
```

```
}
```

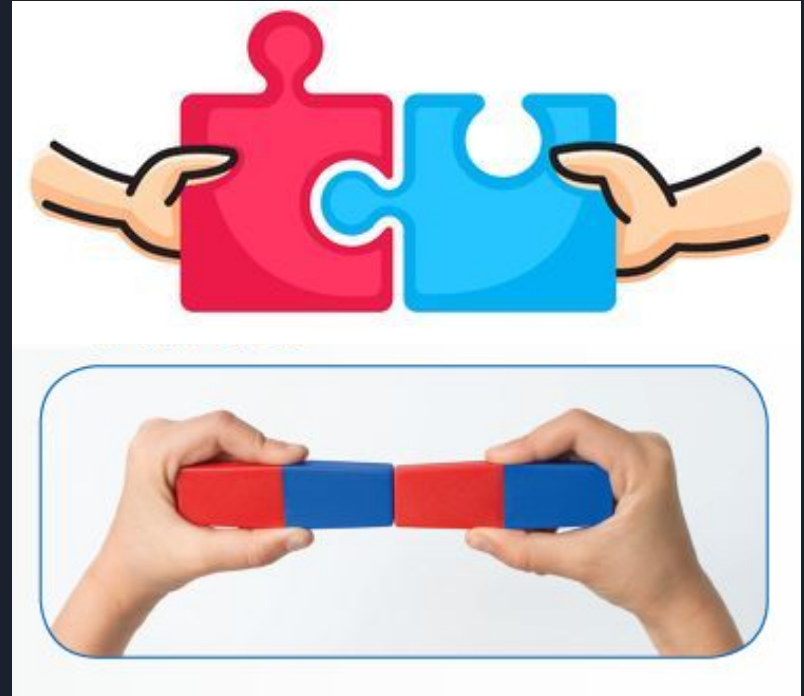


INYECCIÓN DE DEPENDENCIAS **AUTOMÁTICA**



INYECCIÓN DE DEPENDENCIAS AUTOMÁTICA

- Las dependencias de los beans se deben satisfacer.
 - Manualmente
 - **Automáticamente**





INYECCIÓN DE DEPENDENCIAS AUTOMÁTICA

- Spring permite la inyección *automática* entre beans que se *necesitan*.
- Busca candidatos dentro del *IoC container*.
- Ventajas
 - Reduce la configuración necesaria
 - Útil durante el desarrollo. Permite requerir objetos sin configurarlo explícitamente.



INYECCIÓN DE DEPENDENCIAS AUTOMÁTICA

- Inconvenientes
 - Es útil si se usa siempre en un proyecto.
 - En otro caso, puede ser confuso.
 - No se pueden *autoinyectar* tipos primitivos o String.
 - Menos exacto que la inyección explícita
 - Posible ambigüedad en inyección *byType*.



Uso de **@Autowired**

- Busca un bean adecuado y lo inyecta en la dependencia.
- Se realiza un autocableado por tipo

```
@Controller  
public class PseudoMain {  
  
    @Autowired  
    private Saludator saludador;  
  
}
```




¿Dónde puedo usar **@Autowired**?

- ▶ Método setter

```
@Autowired  
public void setPelículaDao(PelículaDao películaDao) {...}
```

- ▶ Definición de la propiedad

```
@Autowired  
private PelículaDao películaDao;
```

- ▶ Constructor

```
@Autowired  
public PelículaService(PelículaDao películaDao) {...}
```



¿Dónde puedo usar **@Autowired**?

- Se pueden mezclar los 3 tipos de uso de autowired.
 - En la propiedad es muy cómodo.
 - Si el método *setter* tiene alguna “lógica especial”, sería adecuado.
 - Para atributos *final*, usamos el constructor.

¿Autoinyección **sin** **@Autowired**?



- Spring 5 lo permite.
- Intenta realizar la inyección automática de todo lo que reciba en el constructor.
- Muy potente si se utiliza junto a Lombok con *@RequiredArgsConstructor*

```
@Controller
@RequiredArgsConstructor
public class PseudoMain {
    private final Saludator saludador;
    //Resto del código
}
```

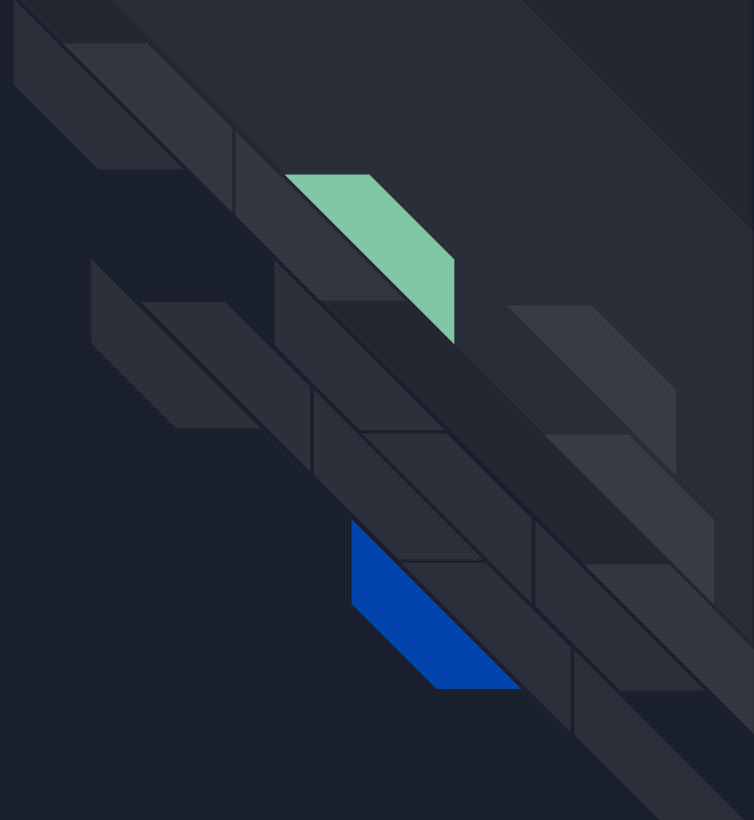


@Autowired **no satisfecho**

- ¿Y si no hay un bean del tipo que necesito?
 - Se produce una excepción
- Podemos modificar este comportamiento para que deje la dependencia sin satisfacer, pero sin excepción:
 - `@Autowired(required=false)`
 - `@Nullable` (Spring 5)
 - `Optional<?>` (Java 8)



@Primary y @Qualifier





¿Y si hay más de un bean de un tipo?

- Al intentar inyectar, se produce también una excepción.
- Tenemos dos opciones
 - **@Primary**: *primum inter pares*
 - **@Qualifier**: escoger uno específicamente



@Primary

- Si se define el bean con **anotaciones**
 - Anotación sobre el estereotipo

```
@Service @Primary public class Servicio { ... }
```

- Si se define el bean con **JavaConfig**
 - Anotación en el método anotado con *@Bean*

```
@Bean @Primary public TipoBean tipoBean() { ... }
```

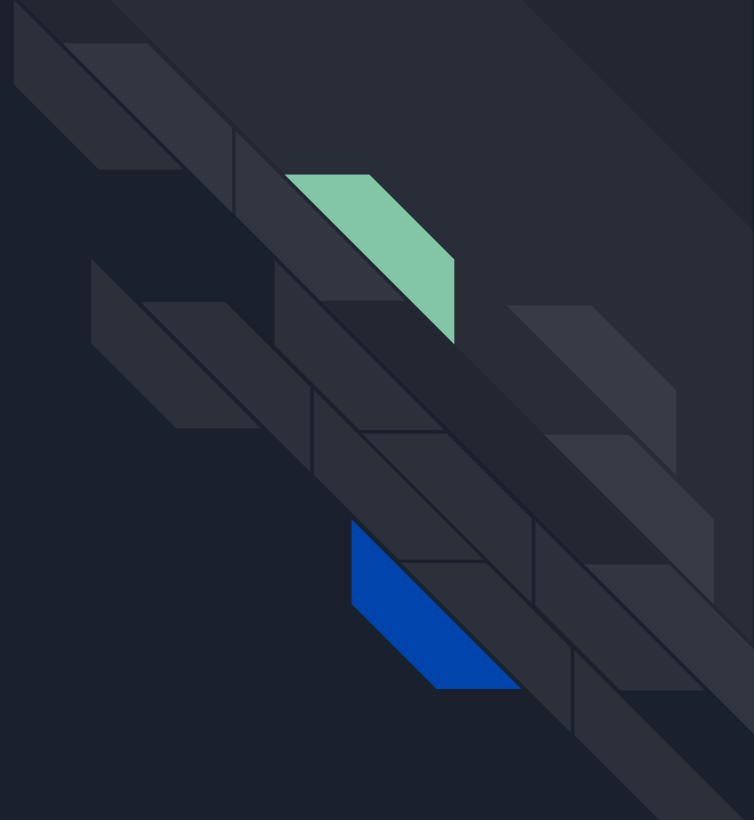


@Qualifier

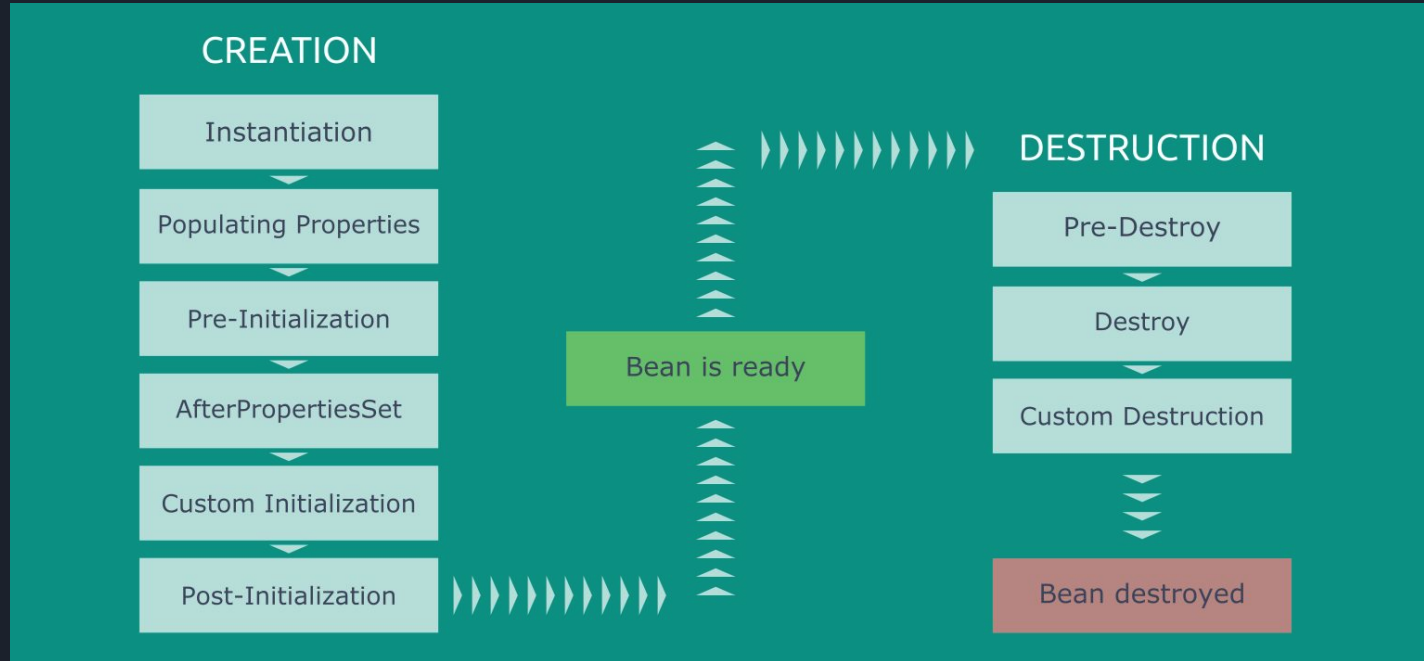
- Nos permite ajustar qué bean, de entre los de un tipo, vamos a inyectar.
- El nombre de un bean suele ser el de su clase, empezando por minúscula (*camel case*)
 - *EnglishSaludator* → *englishSaludator*
- `@Autowired + @Qualifier("nombre")`



CICLO DE VIDA DE UN BEAN



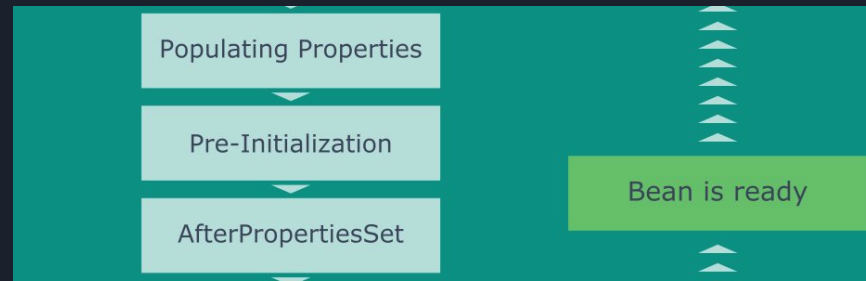
CICLO DE VIDA DE UN BEAN



Podemos participar durante varios momentos del ciclo de vida.

@PostConstruct y @PreDestroy

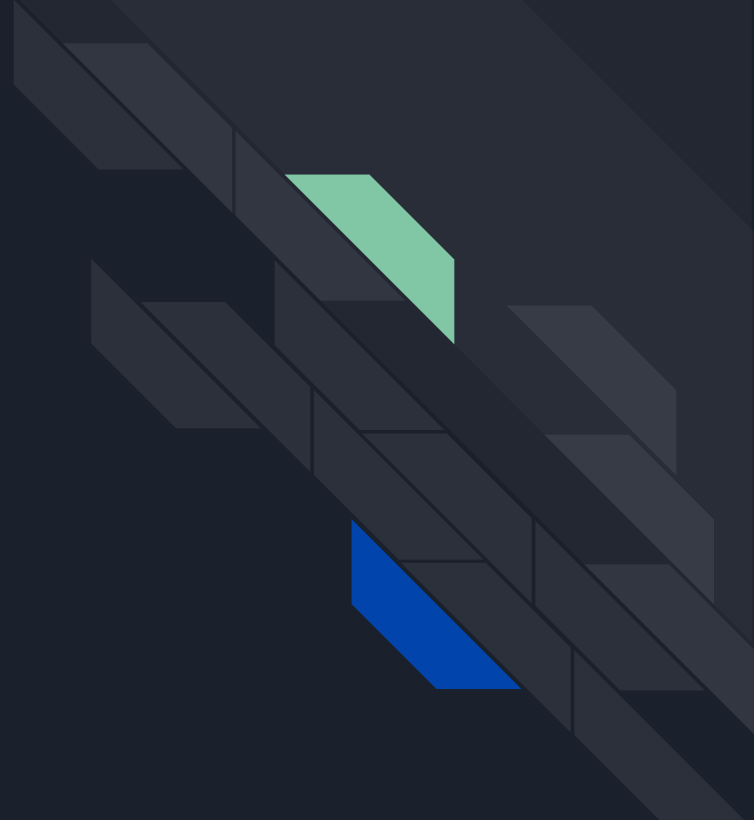
- @PostConstruct
 - Acciones que se realizan después de la inicialización del bean.
 - *Pre-Initialization*
- @PreDestroy
 - Acciones que se realizan antes de la destrucción del bean.
 - *Pre-Destroy*





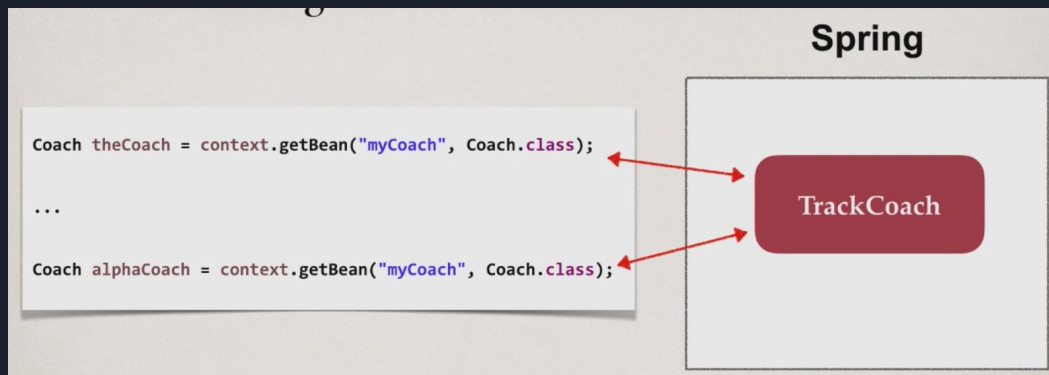
ÁMBITO DE UN BEAN

¿CUÁNTO TIEMPO VIVE UN BEAN?



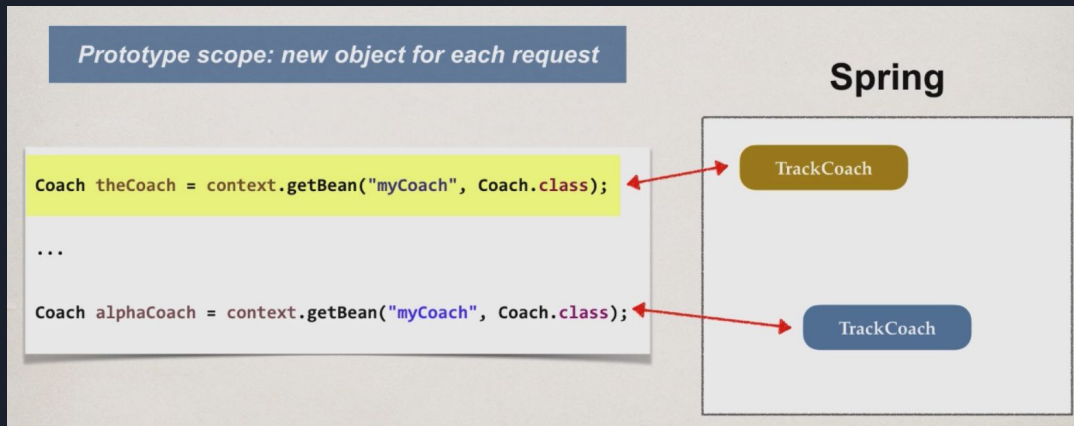
¿CUÁNTO TIEMPO VIVE UN BEAN?

- Por defecto, los beans son *singleton*
 - Se crea una instancia al levantar el *IoC container*
 - Cada vez que se requiere, inyecta, ... se utiliza la misma instancia.



¿CUÁNTO TIEMPO VIVE UN BEAN?

- Podemos cambiar el ámbito a *prototype*
 - Cada vez que se requiere, inyecta, ... se crea una nueva instancia.
 - Menos habitual de usar.
- `@Scope("prototype")`





¿CUÁNTO TIEMPO VIVE UN BEAN?

- Otros ámbitos (usados en aplicaciones web)
 - Petición (request)
 - Sesión (session)
 - Aplicación (application)
 -