

Programación Frontend y Backend

BLOQUE SPRING

Spring



01

Conceptos Básicos



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



- Spring es el framework Java utilizado por excelencia para el desarrollo de aplicaciones empresariales de manera ***simplificada***.
- Una de las mayores ventajas de Spring, es la forma ***modular*** en el que fue creado, permitiendo ***habilitar / deshabilitar*** las características a utilizar según se requiera.
- Spring es utilizado en proyectos muy diversos, como puede ser en Instituciones Bancarias, Aseguradoras, Instituciones Educativas y de Gobierno, entre muchos otros tipos de proyectos y empresas.

Debido al aumento de la complejidad que presentan la mayoría de los sistemas web, tanto en temas de seguridad, configuración, funcionalidades, la comunidad de desarrolladores se vio en la necesidad de diseñar ciertas ayudas, a fin de no tener que repetir código, reduciendo de este modo el tiempo y el espacio para el desarrollo de aplicaciones.



Spring Framework

Es el framework más popular Java para crear código de alto rendimiento, liviano y reutilizable. Ya que su finalidad es estandarizar, agilizar, manejar y resolver los problemas que puedan ir surgiendo en el proceso de la programación.



Ligero



Extendido



Mantenible



Ampliable

¿Framework?

Un framework (entorno de trabajo o marco de trabajo) es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas similares.



Spring Framework - Módulos

Spring Framework está actualmente dividido en módulos, cada uno orientado a una finalidad concreta. Cada proyecto que creemos podrá utilizar los módulos que necesiten sin la necesidad importar todos los módulos de Spring, aunque existe un core necesario para empezar a utilizarlo.

Documentación oficial:

<http://spring.io>

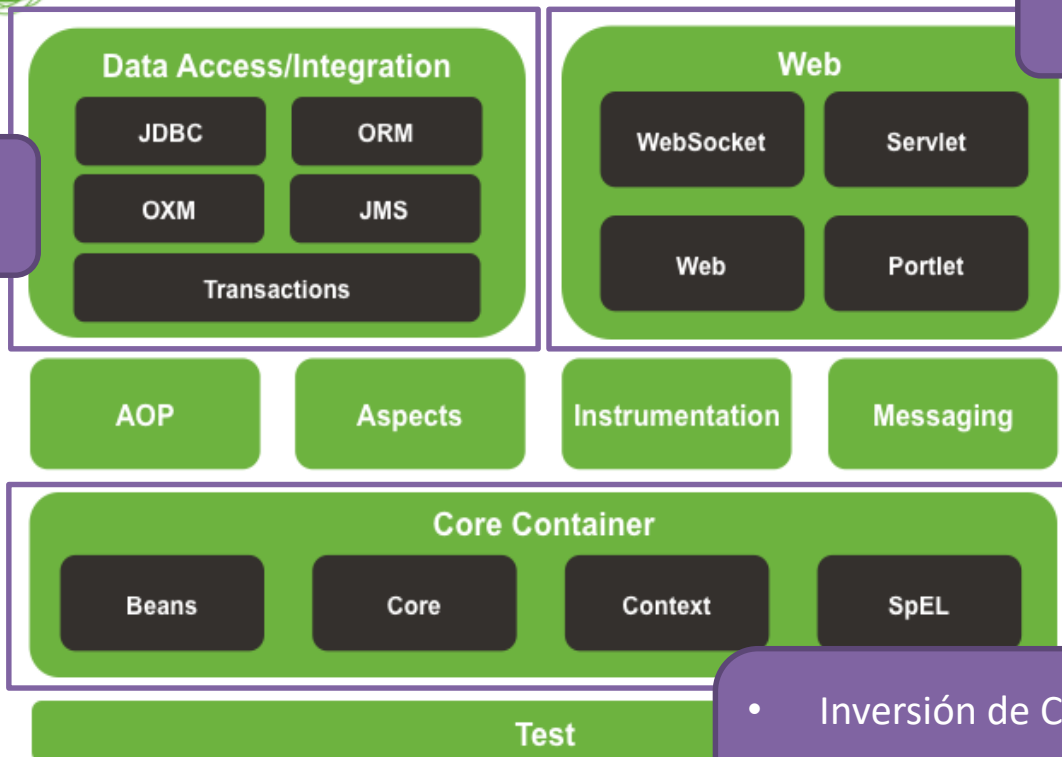
Spring



Spring Framework Runtime

Spring MVC

Spring Data



- Inversión de Control (IoC)
- Inyección de Dependencia (ID)
- Spring Container (IoC Container)



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



SISTEMA NACIONAL DE
**GARANTÍA
JUVENIL**

Inversión de control (IoC)

```
public class Motor {
```

```
    public void acelerar() {  
    }
```

```
    public int getRevoluciones() {  
        return currentRPM;  
    }
```

```
}
```

```
public class Vehiculo {
```

```
    private Motor m;
```

```
    public Vehiculo() {  
        m = new Motor();  
    }
```

```
    public int getRevolucionesMotor() {  
        return m.getRevoluciones();  
    }
```

```
}
```

La dependencia entre las clases **Vehiculo** y **Motor** queda patente dado que una instancia de la primera alberga dentro una instancia de la segunda, el acoplamiento existente en el código es alto. El motor está fuertemente ligado al vehículo, de forma que esta relación es poco flexible.

Inversión de control (IoC)

```
public class Motor {
```

```
    public void acelerar() {  
    }
```

```
    public int getRevoluciones() {  
        return currentRPM;  
    }
```

```
}
```

```
public class Vehiculo {
```

```
    private Motor m;
```

```
    public Vehiculo() {  
        m = new Motor();
```

```
    }
```

```
    public int getRevolucionesMotor() {  
        return m.getRevoluciones();
```

```
    }
```

```
}
```

Si quisiéramos realizar cualquier tipo de modificación en la clase **Motor**, esto supondría un alto impacto en la clase **Vehiculo** (por ejemplo, si quisiéramos hacer una concreción en **MotorDiesel** o **MotorGasolina**)

Inversión de control (IoC)


```
public class Vehiculo {  
    private Motor m;  
  
    public Vehiculo(Motor motorVehiculo) {  
        m = motorVehiculo;  
    }  
  
    public int getRevolucionesMotor() {  
        return m.getRevoluciones();  
    }  
}
```



Como primer paso para desacoplar el **motor** del **vehículo**, podríamos hacer que la clase **Vehiculo** deje de encargarse de instanciar el objeto **Motor**, pasándoselo como parámetro al constructor. El constructor de vehículo se encarga de inyectar la dependencia dentro del objeto, eliminando esta responsabilidad de la propia clase. De esa forma, hemos dado un paso para desacoplar ambos objetos.

Inversión de control (IoC)

```
public interface IMotor {  
    public void acelerar();  
    public int getRevoluciones();  
}  
  
public class Vehiculo {  
    private IMotor m;  
  
    public Vehiculo(IMotor motorVehiculo) {  
        m = motorVehiculo;  
    }  
  
    public int getRevolucionesMotor() {  
        return m.getRevoluciones();  
    }  
}
```



El siguiente paso que podríamos dar con la intención de continuar con el desacoplamiento de ambos objetos es el uso de interfaces. Como podemos observar, la clase Vehiculo ya no está acoplada a la clase **Motor**, sino que bastará con un objeto que implemente la interfaz **IMotor**

Inversión de control (IoC)

```
public class MotorGasolina implements IMotor {  
    public void acelerar() {  
        realizarAdmision();  
        realizarCompresion();  
        realizarExplosion();  
        realizarEscape();  
    }  
    public int getRevoluciones() {  
        return currentRPM;  
    }  
}
```

```
public class MotorDiesel implements IMotor {  
    public void acelerar() {  
        realizarAdmision();  
        realizarCompresion();  
        realizarCombustion();  
        realizarEscape();  
    }  
    public int getRevoluciones() {  
        return currentRPM;  
    }  
}
```

Inversión de control (IoC)

```
public class Main {  
    public static void main(String[] args) {  
        Vehiculo cocheDiesel = new Vehiculo(new MotorDiesel());  
        Vehiculo cocheGasolina = new Vehiculo(new MotorGasolina());  
    }  
}
```

“Inyección de dependencias para pobres.”

Inversión de control (IoC)

Hasta ahora, hemos visto la conocida como inyección de dependencias para pobres, en la que solamente hemos utilizado elementos de Java para realizar dicha inyección, como constructores, asignación de atributos (Setter).

Antes de continuar, hagámonos la siguiente pregunta con respecto al ejemplo anterior:

¿Qué otras dependencias tiene un vehículo?

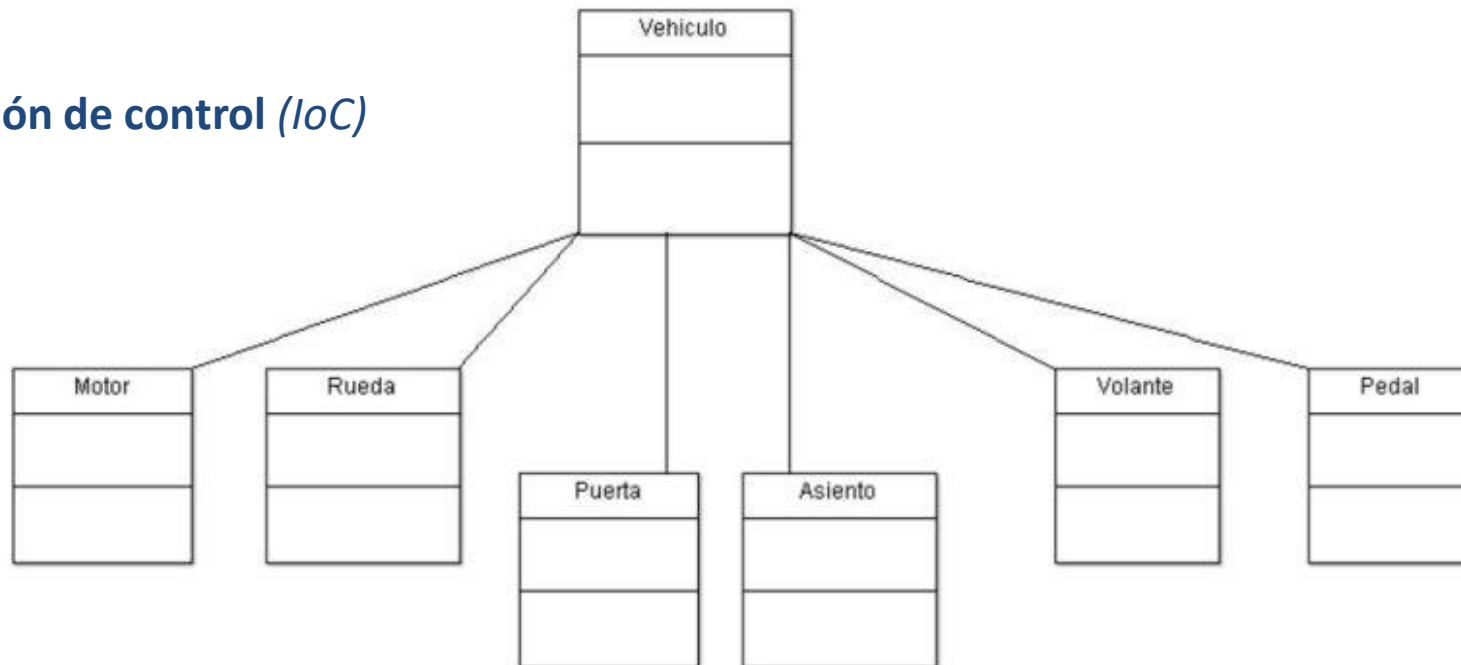
Inversión de control (IoC)

Hasta ahora, hemos visto la conocida como inyección de dependencias para pobres, en la que solamente hemos utilizado elementos de Java para realizar dicha inyección, como constructores, asignación de atributos (Setter).

Antes de continuar, hagámonos la siguiente pregunta con respecto al ejemplo anterior:

¿Qué otras dependencias tiene un vehículo?

Inversión de control (IoC)



Como podemos observar en el diagrama de clases UML, la clase **Vehiculo** tiene una gran cantidad de dependencias. ¿Quién se hará cargo de todas ellas? La respuesta es **Spring IoC Container**

Spring Container (*IoC Container*)

Forma parte del núcleo de Spring Framework el cual se encarga de crear los objetos, los inyecta, los configura y maneja el ciclo completo de vida hasta la destrucción del mismo.



02

Componentes



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Beans

Un **Bean** en Spring no es mas que un objeto configurado e instanciado en el ***contenedor de Spring*** usado entre otras cosas para la ***inyección de dependencias***.

Todos los *beans* permanecen en el contenedor durante toda la vida de la aplicación o hasta que nosotros los destruyamos.

Tener los beans en el contenedor nos permite ***inyectarlos*** en otros beans, ***reutilizarlos***, o poder ***acceder a ellos*** desde cualquier lugar de la aplicación en el momento que queramos.

Tipos de Componentes (Beans)

@Component:

Componente genérico de Spring

@Controller:

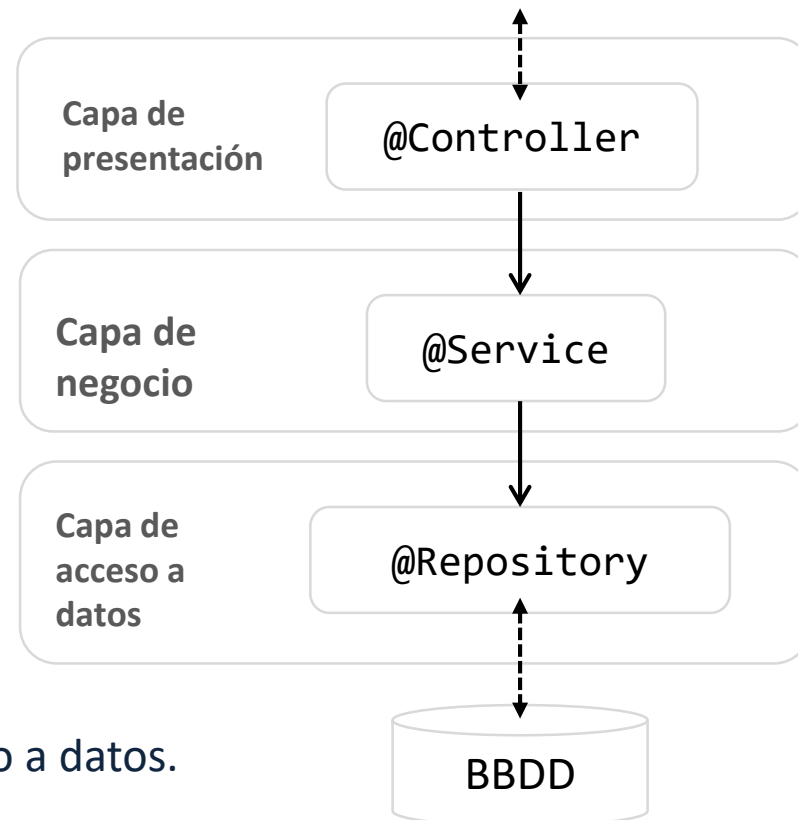
Comp. para la capa de presentación.

@Service:

Comp. para la capa de servicio.

@Repository:

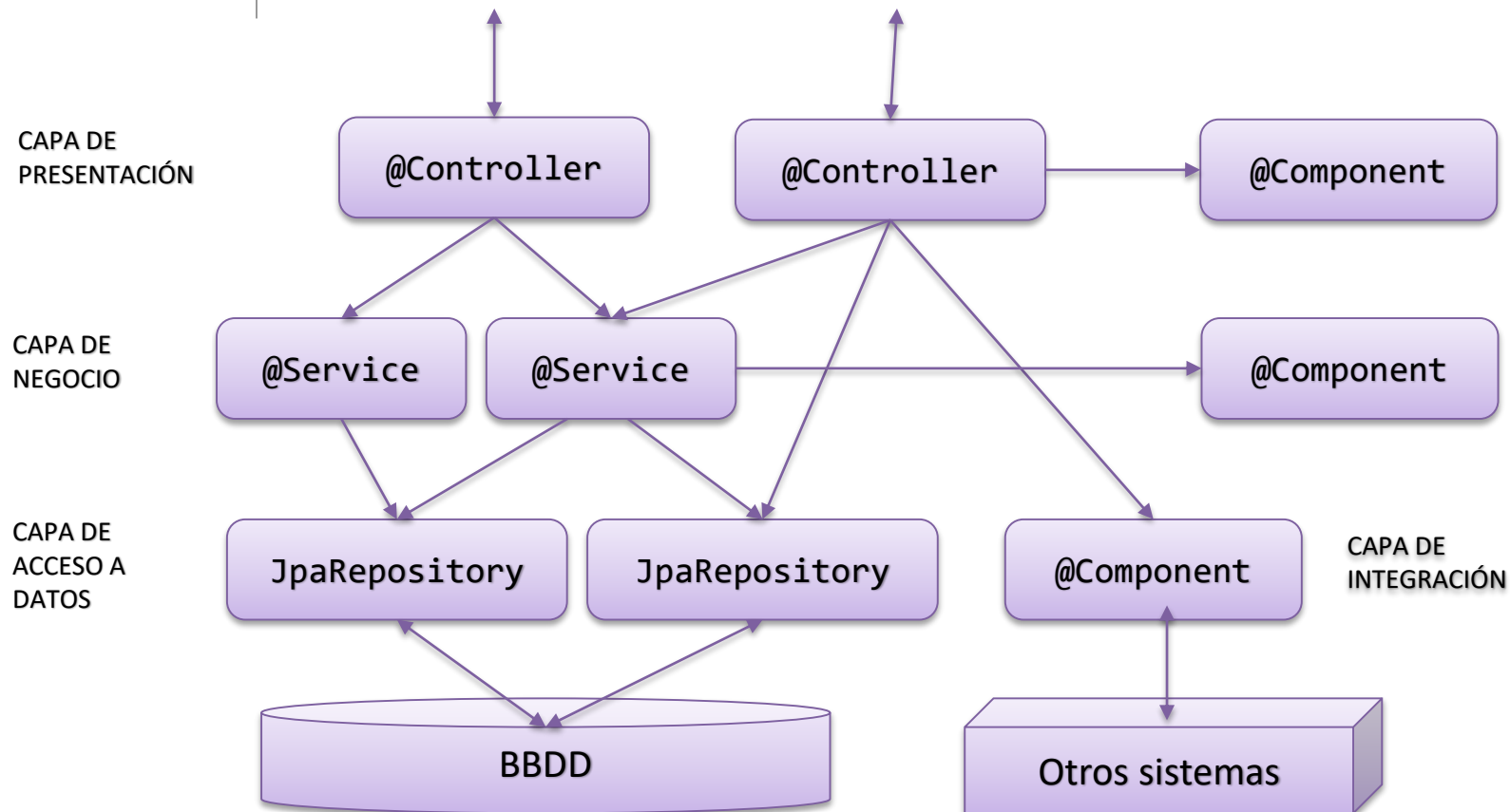
Comp. para la capa de persistencia o de acceso a datos.



Spring escanea estas anotaciones e inyecta las clases en el contexto de la aplicación.

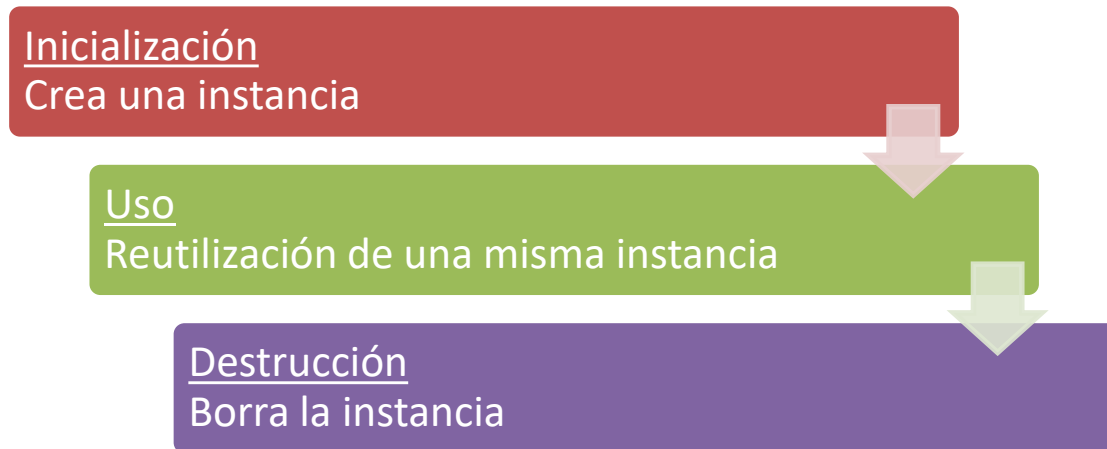
Spring

Componentes



Ciclo de Vida

Es fundamental tener en cuenta el ciclo de vida de los beans que vamos a declarar en nuestra aplicación, por una razón principal: Las dependencias de otros beans, puesto que algunos beans van a requerir que otros ya estén creados previamente.



Inyección de dependencias

Técnica que se aplica para que un objeto no tenga que obtener sus dependencias, es decir, las referencias a los objetos que colaboran con él, de forma que el contenedor las inyecta al crearlo.

Ventajas:

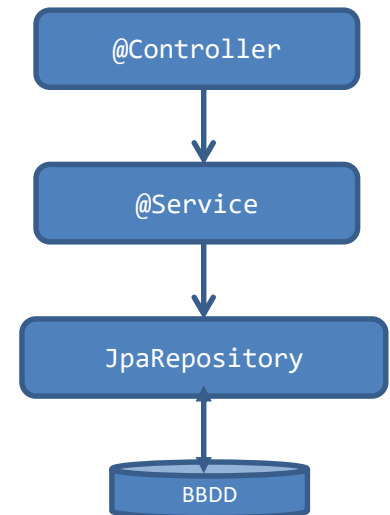
- Código más sencillo y fácil de entender.
- Facilidad para probar (mock objects y pruebas unitarias).
- Facilidad para reutilizar.
- El código no queda acoplado (dependiente) de una implementación específica.

Inyección de dependencias

@Autowired declara un constructor, un campo, un método set() o un método de configuración para que sea enlazado automáticamente por Spring con un bean del tipo correspondiente.

```
@Controller
public class UserController {
    @Autowired
    private UserService service;
}
```

@Autowired



03

Implementación



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Escuela de
organización
industrial

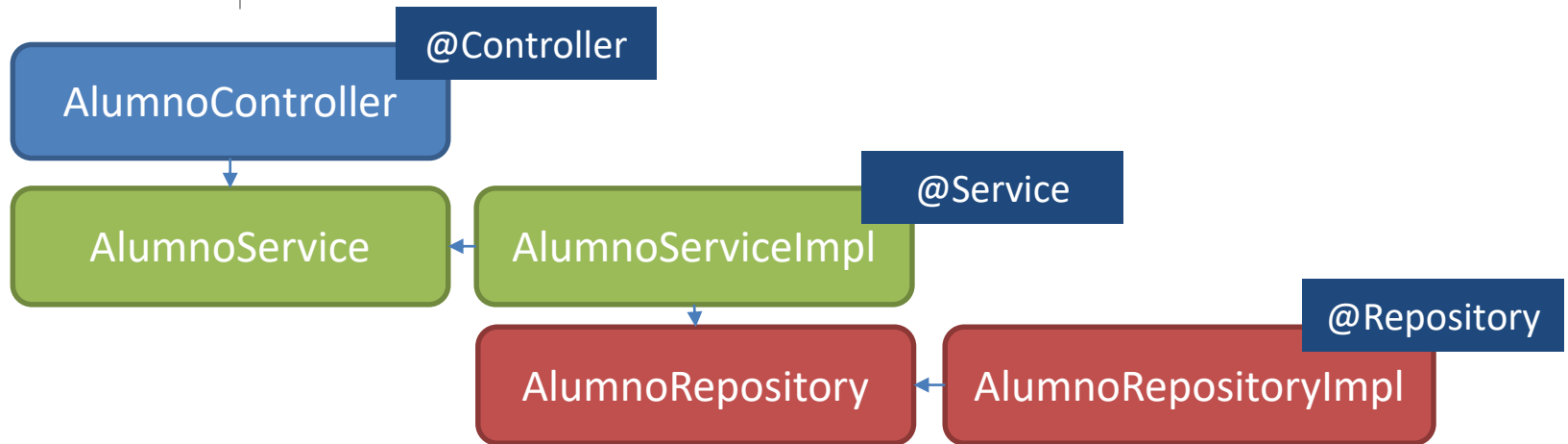


Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Spring

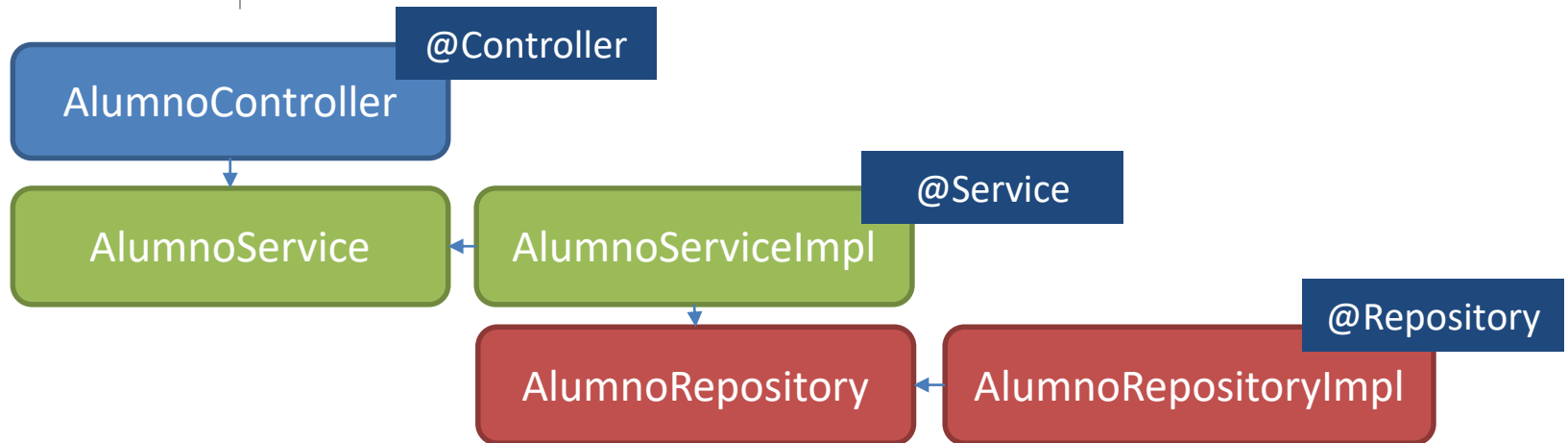
Implementación



```
@Controller
public class AlumnoController {
    @Autowired
    private AlumnoService service;
}
```

Spring

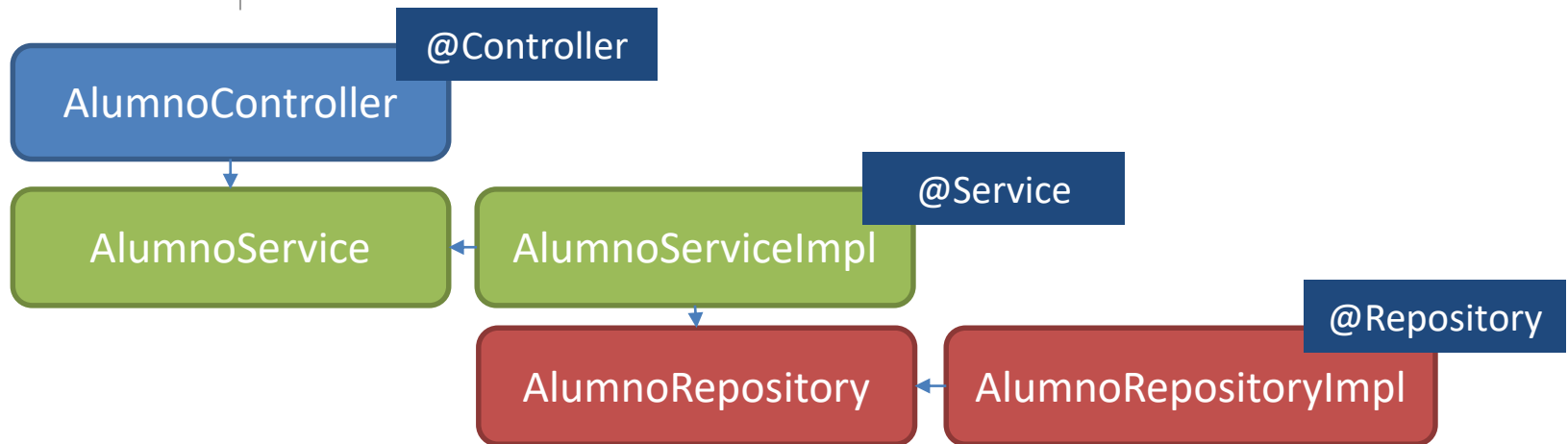
Implementación



```
public interface AlumnoService {  
  
}  
  
@Service  
public class AlumnoServiceImpl implements AlumnoService {  
    @Autowired  
    AlumnoRepository repository;  
}
```

Spring

Implementación



```
public interface AlumnoRepository {
```

```
}
```

```
@Repository
```

```
public class AlumnoRepositoryImpl implements AlumnoRepository {
```

```
}
```