

SPRING

Michael Jaramillo Palacio

Contenido Seminario

01. Spring

02. Anotaciones

03. IoC y Dependencias

04. Interfaces

05. AOP

06. Spring Boot

01

Spring

Spring

Spring es un framework de desarrollo cuya base es el lenguaje Java, aunque también ofrece soporte para Kotlin y, de manera limitada, para Groovy.

Es uno de los frameworks más populares y ampliamente utilizados para crear aplicaciones Java modernas, gracias a su modularidad, facilidad para gestionar dependencias y amplio ecosistema de proyectos complementarios.

Modulos especializados

Spring MVC	Aplicaciones web
Spring Data	Bases de datos
Spring Security	Autenticacion y autorizacion
Spring Cloud	Es un conjunto de herramientas para aplicaciones distribuidas.
Spring Boot	Es un framework impulsor que facilita configurar y arrancar proyectos Spring.

02

Anotaciones

Anotaciones

@Autowired	@Controller
@Configuration	@GetMapping("/")
@Component	@RestController
@ComponentScan	@PostMapping
@Service	@PutMapping
@Aspect	@DeleteMapping

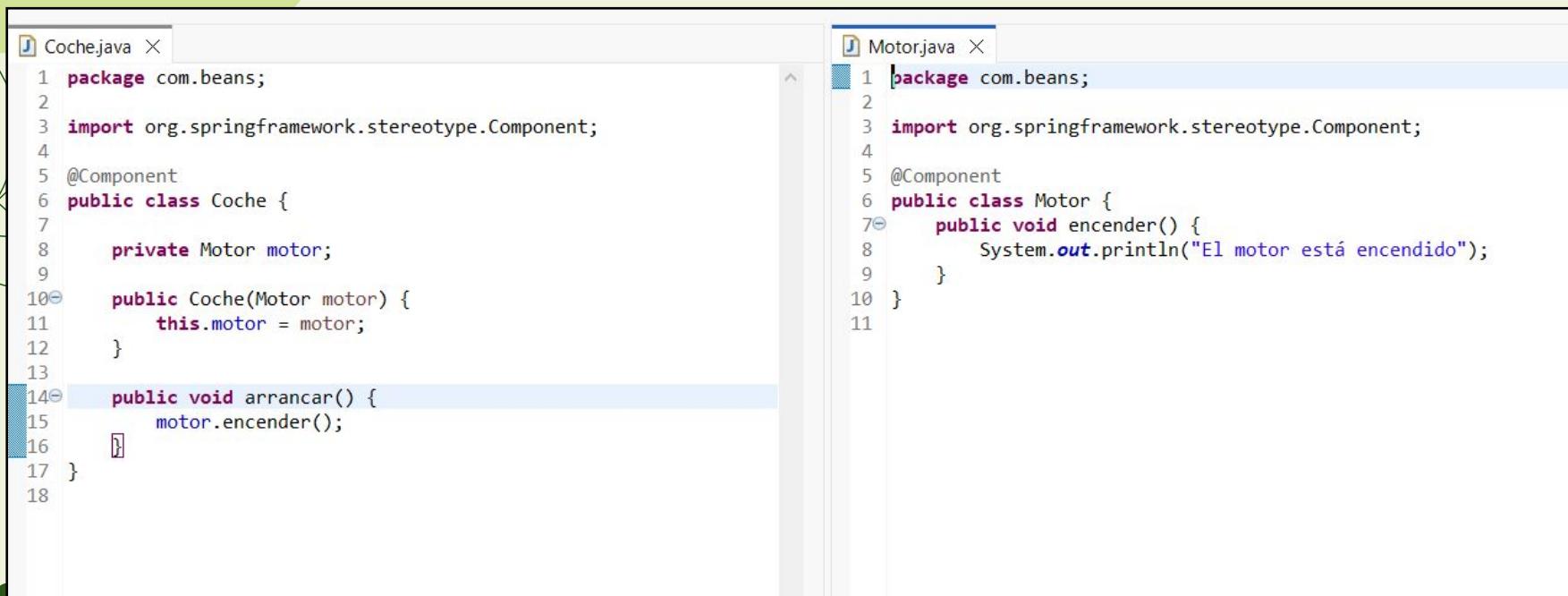
03

IoC y dependencias

IoC y dependencias

Spring se encarga de crear, configurar y conectar los objetos de la aplicación automáticamente, en lugar de instanciarse con new. Esto se hace mediante el contenedor de Spring, que gestiona los beans y resuelve sus dependencias. Permite cambiar implementaciones fácilmente usando interfaces o configuraciones externas.

Ejemplo



```
Coche.java
1 package com.beans;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Coche {
7
8     private Motor motor;
9
10    public Coche(Motor motor) {
11        this.motor = motor;
12    }
13
14    public void arrancar() {
15        motor.encender();
16    }
17}
18

Motor.java
1 package com.beans;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Motor {
7    public void encender() {
8        System.out.println("El motor está encendido");
9    }
10}
11
```

Ejemplo



```
AppConfig.java X
1 package com.example.demo;
2
3+ import org.springframework.context.annotation.ComponentScan;[]
4
5
6 @Configuration
7 @ComponentScan(basePackages = "com.beans,com.interfaces")
8 public class AppConfig { }
9
```

Ejemplo



```
 AppConfig.java *Main.java X
 1 package com.example.demo;
 2
 3+ import org.springframework.context.ApplicationContext;...
 4
 5
 6 public class Main {
 7     public static void main(String[] args) {
 8         ApplicationContext context =
 9             new AnnotationConfigApplicationContext(AppConfig.class);
10
11
12
13         Coche coche = context.getBean(Coche.class);
14         coche.arrancar();
15         //Controlador controlador = context.getBean(Controlador.class);
16         //controlador.ejecutar();
17     }
18 }
19
20
```

04

Interfaz

Interfaz

Interfaz en Spring: es un contrato que define qué métodos debe tener una clase sin decir cómo implementarlos. Se usa para desacoplar código, facilitar la inyección de dependencias

Ejemplo con interfaces

```
1 package com.interfaces;  
2  
3 public interface ServicioSaludo {  
4     void saludar();  
5 }  
6
```

Ejemplo con interfaces

```
1 package com.interfaces;
2
3+ import org.springframework.beans.factory.annotation.Autowired;[]
4
5
6
7 @Component
8 public class Controlador {
9
10     public ServicioSaludo servicioSaludo;
11
12     @Autowired
13     public Controlador(@Qualifier("saludoInformal") ServicioSaludo servicioSaludo) {
14         this.servicioSaludo = servicioSaludo;
15     }
16
17     public void ejecutar() {
18         servicioSaludo.saludar();
19     }
20 }
21
```

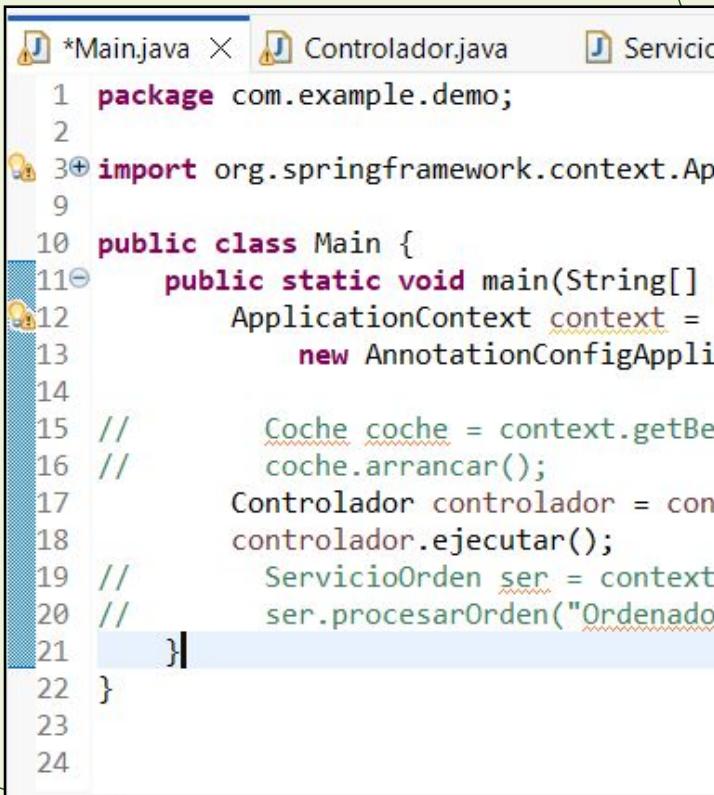
Ejemplo con interfaces

```
1 package com.interfaces;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("saludoFormal")
6 public class ServicioSaludoFormal implements ServicioSaludo {
7     @Override
8     public void saludar() {
9         System.out.println("Buenos días, señor/señora.");
10    }
11 }
12
```

Ejemplo con interfaces

```
1 package com.interfaces;
2
3 import org.springframework.stereotype.Component;
4
5 @Component("saludoInformal")
6 public class ServicioSaludoInformal implements ServicioSaludo {
7     @Override
8     public void saludar() {
9         System.out.println("¡Hola, qué tal!");
10    }
11 }
12
```

Ejemplo con interfaces



The screenshot shows a Java code editor with the following code:

```
1 package com.example.demo;
2
3 import org.springframework.context.ApplicationContext;
4
5 public class Main {
6     public static void main(String[] args) {
7         ApplicationContext context = new AnnotationConfigApplicationContext();
8
9         // Get the car bean
10        Coche coche = context.getBean("coche");
11        coche.arrancar();
12
13        Controlador controlador = context.getBean("controlador");
14        controlador.ejecutar();
15
16        ServicioOrden ser = context.getBean("orden");
17        ser.procesarOrden("Ordenado");
18    }
19}
```

The code demonstrates the use of Spring's `ApplicationContext` to manage beans like `Coche`, `Controlador`, and `ServicioOrden`.

05

AOP

AOP/Programación Orientada a Aspectos

La AOP es un paradigma de programación que permite separar las funcionalidades transversales de la lógica principal de tu aplicación.

Son tareas que afectan a varias partes del código, pero no forman parte de la lógica principal.

Separa esas funcionalidades en aspectos, y el framework las “inyecta” donde se necesiten, sin tocar la lógica principal.

Ejemplo AOP

```
<!-- Dependencias de AOP -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

Ejemplo AOP

```
1 package com.example.demo;
2
3+import org.springframework.context.annotation.ComponentScan;□
4
5
6
7 @Configuration
8 @EnableAspectJAutoProxy |
9 @ComponentScan(basePackages = "com.beans,com.interfaces,com.AOP")
10 public class AppConfig { }
11
```

Ejemplo AOP



The screenshot shows a Java code editor with several tabs at the top: AppConfig.java, Main.java, ServicioOrden.java (selected), LoggingAspect.java, and runjava. The ServicioOrden.java tab is active, displaying the following code:

```
1 package com.AOP;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class ServicioOrden {
7
8     public void procesarOrden(String producto) {
9         System.out.println("Procesando orden: " + producto);
10    }
11 }
```

Ejemplo AOP

```
package com.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {

    @Before("execution(* com.AOP.ServicioOrden.procesarOrden(..))")
    public void logAntesDeProcesar() {
        System.out.println("LOG: Antes de procesar la orden");
    }
}
```

Ejemplo AOP

```
1 package com.example.demo;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
5
6 import com.AOP.*;
7 import com.beans.*;
8 import com.interfaces.Controlador;
9
10 public class Main {
11     public static void main(String[] args) {
12         ApplicationContext context =
13             new AnnotationConfigApplicationContext(AppConfig.class);
14
15 //         Coche coche = context.getBean(Coche.class);
16 //         coche.arrancar();
17 //         Controlador controlador = context.getBean(Controlador.class);
18 //         controlador.ejecutar();
19         ServicioOrden ser = context.getBean(ServicioOrden.class);
20         ser.procesarOrden("Ordenador");
21     }
22 }
23
24
```

Ejemplo AOP sin AppConfig

```
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.ApplicationContext;
6 import com.AOP.ServicioOrden;
7
8 @SpringBootApplication(scanBasePackages = {"com.AOP", "com.beans", "com.interfaces"})
9 public class run {
10     public static void main(String[] args) {
11         ApplicationContext context = SpringApplication.run(run.class, args);
12
13         ServicioOrden ser = context.getBean(ServicioOrden.class);
14         ser.procesarOrden("Ordenador");
15     }
16 }
17 }
```

06

Spring Boot

Spring Boot

Spring Boot nace como una evolución de Spring que busca hacerlo rápido, simple y automático. Tiene como objetivo la creación de listas para poder ejecutar con la mejor configuración posible.

Dispone de su propio servidor embebido que es Apache Tomcat por defecto.

Spring Boot Dinamico

Para poder hacer que una página web sea dinámica deberemos realizar los mismos pasos de creación de un proyecto que el anterior con la diferencia de que deberemos añadir una dependencia llamada Thymeleaf.

Spring Boot MVC

Vamos a crear un nuevo proyecto donde podremos ver la diferencia con el anterior y deberemos crear las siguientes carpetas que deben ir ubicadas dentro de la carpeta donde se ejecutará el main

Spring Boot MVC Service

Vamos a crear un nuevo proyecto donde el método que te da el chiste aleatoriamente se gestione por el servicio.

Spring Boot API

Deberemos crear un nuevo proyecto sin la necesidad del Thymeleaf ya que solo vamos a comprobar que los datos se pasan correctamente.

Ejercicio Propuesto

The screenshot shows a web application titled "Chistes" running on a local server at `localhost:8080`. The application displays a table of jokes with columns for ID, Texto, Autor, and a "Borrar" (Delete) button. Below the table is a form for adding a new joke, featuring fields for ID, Texto, Autor, and a green "Agregar Chiste" (Add Joke) button.

ID	Texto	Autor	Borrar
1	¿Qué le dice un 0 a un 8? Bonito cinturón.	Juan	<button>Eliminar</button>
2	¿Qué hace una abeja en el gimnasio? ¡Zum-ba!	Marcos	<button>Eliminar</button>
3	¿Cuál es el colmo de un electricista? Que su hijo sea un apagado.	Luis	<button>Eliminar</button>
4	¿Por qué estás hablando con la pared? ¡Porque la mesa no me responde!	Anonimo	<button>Eliminar</button>

Formulario para Agregar Chiste:

ID:

Texto:

Autor:

Agregar Chiste