

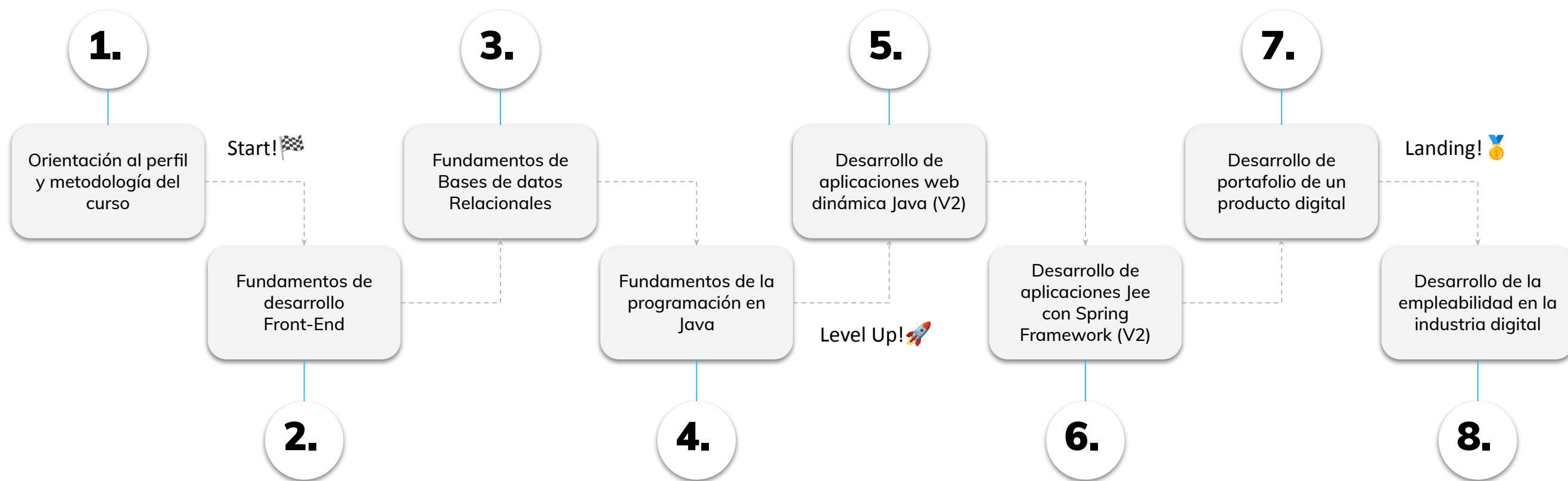
# » El Framework Spring MVC

---

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



# REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ Fase de empaquetamiento
- ✓ Instalación y ejecución

# LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.1

Start! 🚩

El Framework Spring  
MVC

SpringBoot

Agregando anotaciones

Características del framework SpringBoot  
El concepto y el uso de las anotaciones  
Qué es un Bean en Spring y cómo definirlo  
Inyección de dependencias y la anotación Autowired

# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



*Conocer las características de Spring Boot*



*Aprender conceptos de anotaciones y Bean*



*Comprender el concepto de inyección de dependencias*



# › Bean

# Bean



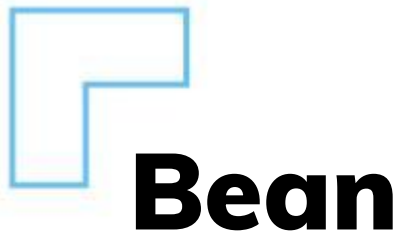
Un concepto fundamental en el framework de Spring es el "Bean".

Un Bean es un objeto gestionado por el contenedor de Spring, que **se encarga de su creación, configuración y administración durante el ciclo de vida de la aplicación.**



Los Beans son componentes clave en el desarrollo de aplicaciones basadas en Spring, y comprenden **la base de la Inyección de Dependencias** (DI) y la **Inversión de Control** (IoC) que Spring promueve.





¿Qué es un Bean en Spring?

Un Bean en Spring es un objeto que es administrado por el contenedor de Spring. Puede ser cualquier clase de Java, desde simples POJOs (Plain Old Java Objects) hasta componentes más complejos. La principal característica de un Bean es que su ciclo de vida, configuración y administración están a cargo de Spring, en lugar de ser gestionados directamente por el programador. Esto ofrece ventajas significativas en términos de modularidad, reutilización y mantenibilidad del código.





# Bean



## Características clave de un Bean en Spring:

- **Administración centralizada:** Spring se encarga de crear, configurar y administrar los Beans, lo que significa que no tienes que preocuparte por instanciarlos manualmente o configurar sus dependencias.
- **Inversión de Control (IoC):** Los Beans siguen el principio de IoC, donde el control sobre la creación y el flujo de objetos se invierte en el contenedor de Spring. En lugar de que las clases creen y gestionen sus propias dependencias, Spring se encarga de inyectarlas.



# Bean




## Características clave de un Bean en Spring:



- **Inyección de Dependencias (DI):** Los Beans pueden depender de otros Beans, y Spring se encarga de inyectar esas dependencias. Esto promueve la modularidad y facilita la sustitución de implementaciones sin cambiar el código fuente.
- **Configuración flexible:** La configuración de un Bean se realiza a través de metadatos, generalmente en un archivo de configuración XML o mediante anotaciones en el código fuente. Esto permite una configuración flexible y la posibilidad de cambiar la configuración sin modificar el código.
- **Ciclo de vida gestionado:** Spring controla el ciclo de vida de un Bean, lo que significa que se pueden realizar tareas específicas en momentos clave, como la inicialización o la destrucción del Bean.





# Bean



## ¿Cómo se define un Bean en Spring?

Para definir un Bean en Spring, puedes seguir varios enfoques, pero dos de los más comunes son mediante configuración XML y mediante anotaciones.

### **Definición de Beans mediante configuración XML:**

- 1- Crear el archivo de configuración: Comienza creando un archivo XML de configuración de Spring. Puede llamarse, por ejemplo, "applicationContext.xml".
- 2- Registrar el Bean en el archivo de configuración: En el archivo XML, define el Bean utilizando el elemento <bean>. Debes especificar el nombre del Bean, su clase y cualquier propiedad que desees configurar. Aquí tienes un ejemplo:

# Bean

```
<bean id="miBean" class="com.ejemplo.MiClase">  
  <property name="propiedad1" value="valor1" />  
</bean>
```

En este ejemplo, se define un Bean llamado "miBean" que es una instancia de la clase "com.ejemplo.MiClase". Además, se configura la propiedad "propiedad1" con el valor "valor1".

# Bean



3- Configurar el contenedor Spring: Esto se hace generalmente mediante un archivo de configuración que define el contexto de la aplicación y registra el archivo XML de configuración. En el archivo XML de configuración principal, puedes usar el elemento `<import>` para incluir el archivo de configuración de los Beans.



```
<import resource="applicationContext.xml" />
```

Obtener el Bean del contenedor: Una vez que los Beans están definidos en el archivo de configuración, puedes obtenerlos del contenedor de Spring en tu código mediante el contexto de la aplicación. Por ejemplo:

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
```



```
MiClase miBean = (MiClase) context.getBean("miBean");
```

# Bean



## Definición de Beans mediante anotaciones:

1- **Anotar la clase:** Marca la clase que deseas que sea un Bean con la anotación `@Component` o alguna de sus variantes, como `@Service`, `@Repository`, o `@Controller`, dependiendo del tipo de Bean que sea. Por ejemplo:



```
@Component  
public class MiClase {  
    // ...  
}
```



# Bean



## Definición de Beans mediante anotaciones:

2- Habilitar la exploración de componentes: Asegúrate de habilitar la exploración de componentes en tu configuración de Spring. Esto se logra con la anotación `@ComponentScan` en una configuración de Spring:



```
@Configuration
@ComponentScan("com.ejemplo")
public class AppConfig {
    // ...
}
```



# Bean



## Definición de Beans mediante anotaciones:

3- Configurar el contenedor Spring: Debes configurar el contenedor Spring, generalmente mediante un archivo de configuración Java, como se muestra en el paso anterior.



4- Obtener el Bean del contenedor: Una vez configurado el contenedor, puedes obtener el Bean anotado en tu código de la siguiente manera:

```
ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
```

```
MiClase miBean = context.getBean(MiClase.class);
```





# LIVE CODING

Ejemplo en vivo

## Creando Beanes:

*En esta segunda parada, agregaremos algunos @Beans en el proyecto ejemplo!*

  **Tiempo: 10 minutos**



# › Inyección de Dependencias

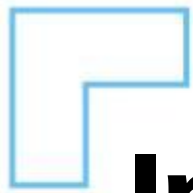
# Inyección de Dependencias

La Inyección de Dependencias (Dependency Injection, DI) es un patrón de diseño fundamental en el desarrollo de software que se utiliza ampliamente en el marco de Spring y en muchos otros marcos y lenguajes de programación. La DI **es una técnica que se enfoca en la gestión de dependencias entre componentes de una aplicación.**



**1**





# Inyección de Dependencias

Los principales beneficios de la Inyección de Dependencias son:



- **Desacoplamiento:** Hace que cada componente se concentre en su propia responsabilidad, sin necesidad de conocer los detalles de implementación de sus dependencias.
- **Reutilización de componentes:** Los mismos componentes pueden ser utilizados en contextos diferentes simplemente inyectando diferentes dependencias.
- **Facilita la prueba unitaria:** Al inyectar dependencias, es más fácil reemplazar las implementaciones reales con implementaciones de prueba o "falsas" durante las pruebas unitarias.
- **Mejora el mantenimiento:** Ya que los cambios en las implementaciones de las dependencias no afectan a los componentes que las utilizan. Esto facilita la incorporación de nuevas funcionalidades y corrección de errores.





# Inyección de Dependencias



## @Autowired

Spring facilita la implementación de la Inyección de Dependencias mediante el uso de la anotación **@Autowired**. Esta anotación se utiliza para marcar campos, constructores o métodos de configuración en una clase para indicar que Spring debe inyectar las dependencias correspondientes en esos puntos.



Existen diferentes maneras de implementar esta anotación:





# Inyección de Dependencias



**Inyección de dependencias en campos:**

```
@Component
public class MiClase {
    @Autowired
    private OtraClase otraClase;

    // ...
}
```



En este ejemplo, la anotación **@Autowired** se utiliza en un campo de la clase `MiClase`. Spring se encargará de inyectar una instancia de `OtraClase` en el campo `otraClase` cuando se cree una instancia de `MiClase`.

# Inyección de Dependencias

## Inyección de dependencias en el constructor:

```
@Component
public class MiClase {
    private OtraClase otraClase;

    @Autowired
    public MiClase(OtraClase otraClase) {
        this.otraClase = otraClase;
    }

    // ...
}
```

En este ejemplo, la anotación **@Autowired** se utiliza en el constructor de `MiClase`. Cuando Spring crea una instancia de `MiClase`, inyectará una instancia de `OtraClase` como argumento del constructor.





# Inyección de Dependencias

Inyección de dependencias en métodos de configuración:



```
@Component
public class MiClase {
    private OtraClase otraClase;

    @Autowired
    public void setOtraClase(OtraClase otraClase) {
        this.otraClase = otraClase;
    }

    // ...
}
```



En este ejemplo, la anotación **@Autowired** se utiliza en un método de configuración llamado `setOtraClase`. Spring llamará este método y pasará una instancia de `OtraClase` como argumento.



# Inyección de Dependencias



Consideraciones adicionales:

La anotación `@Autowired` en Spring es una característica poderosa y versátil que simplifica la Inyección de Dependencias. Sin embargo, es importante tener en cuenta que, en caso de múltiples implementaciones de una interfaz o clase, Spring puede tener dificultades para determinar cuál de ellas debe inyectarse. En tales casos, es posible que debas proporcionar más información a Spring para ayudar en la resolución de dependencias, como utilizando las anotaciones `@Qualifier` o `@Primary`.



# LIVE CODING

Ejemplo en vivo

## Inyección de dependencias:

*En esta tercera parada, agregaremos algunos @Autowired en el proyecto ejemplo para practicar las diferentes maneras que existen de implementar DI con la anotación @Autowired.*

 **Tiempo: 15 minutos**



○

# ¿Alguna consulta?

+



# RESUMEN

¿Qué logramos en esta clase?

- ✓ Conocer el framework Spring Boot y sus características
- ✓ Aprender sobre anotaciones e inyección de dependencias



# #WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 🙌 🙌 🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
  - a. *Lectura Modulo 6, Lección 2: páginas 1 - 6*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

# ¡Muchas Gracias!

Nos vemos en la próxima clase 🙌



Momento: ✚

# Time-out!

🕒 5 min.

