EFA
MORATALAZ

2º CFGS Desarrollo de Aplicaciones Web

DESARROLLO WEB EN ENTORNO SERVIDOR

JESÚS SANTIAGO RICO

UT3 – INYECCIÓN DE DEPENDENCIAS

EFA
MORATALAZ

2º CFGS Desarrollo de Aplicaciones Web

DESARROLLO WEB EN ENTORNO SERVIDOR

UT3 – INYECCIÓN DE DEPENDENCIAS

1. ¿QUÉ ES LA INVERSIÓN DE CONTROL?
2. ¿QUÉ ES LA INYECCIÓN DE DEPENDENCIAS?
3. ¿QUÉ ES UN SERVICIO? CREACIÓN DE SERVICIOS EN SPRING
4. ANOTACIÓN @PRIMARY
5. ANOTACIÓN @QUALIFIER
6. REGISTRO DE COMPONENTES CON ANOTACIÓN @BEAN

¿QUÉ ES LA INVERSIÓN DE CONTROL?

La **inversión de control** es un principio de software en el que el control de nuestros objetos es transferido a un contenedor o framework.

Delegamos, por lo general, en el framework para que tome el flujo del programa. Este sistema tiene una serie de ventajas tales como:

- ✓ Desacopla el código.
- ✓ Mayor facilidad a la hora de testear debido a que se pueden probar partes aisladas.
- ✓ Tiene una gran modularidad.

¿QUÉ ES LA INYECCIÓN DE DEPENDENCIAS?

La inyección de dependencias es un patrón de Software que implementa la inversión de control para resolver dependencias. Los objetos definen sus dependencias con otros objetos.

Todas estas dependencias, se encuentran en un contenedor, que será el responsable de inyectarlas y crear los bean (objetos) necesarios, por un proceso que se llama Inversión de Control (IoC).

Podríamos definir y resumir este concepto como que la Inyección de Dependencias es el proceso de suministrar una dependencia externa a un componente de software.

¿QUÉ ES UN SERVICIO? CREACIÓN DE SERVICIOS EN SPRING

Un **Servicio** es una clase que contiene la lógica de negocio de la aplicación: procesa datos, realiza cálculos, gestiona reglas de negocio y coordina entre diferentes componentes, pero no se encarga directamente de mostrar la interfaz

En Spring, tenemos diversas formas de indicar que una determinada clase se comporta como un servicio, siendo estas las siguientes:

@Component
@Service

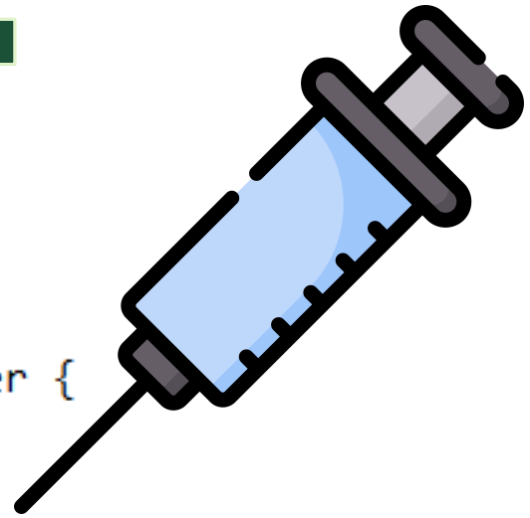
Las dos anotaciones anteriores tienen la misma funcionalidad, salvo que el **service** tiene un valor semántico.

```java
@Service
public class LibroServiceImplement implements LibroService {

    private List<LibroDTO> libros = new ArrayList<>();
    private List<LibroConCantidad> librosConCantidad = new ArrayList<>();
    long idCounter = 1L;

    @Override
    public List<LibroDTO> obtenerTodosLosLibros() {
        return libros;
    }
```

Una vez que hemos creado la clase que será anotada como servicio, estamos en disposición de realizar la inyección de dependencias en donde nos haga falta.

En este punto vamos a ver las diferentes formas de usar nuestros servicios

- ✓ Instanciando el objeto.
- ✓ Usando la clase.
- ✓ Usando una interfaz.



```java
@Controller
public class LibroController {

    @Autowired
    private LibroService libroService;

    // Listar libros
    @GetMapping(value = "/libros")
    public String listarLibros(Model model) {
        model.addAttribute("libros", libroService.obtenerTodosLosLibros());
        return "libros/mostrarLibros";
    }
```

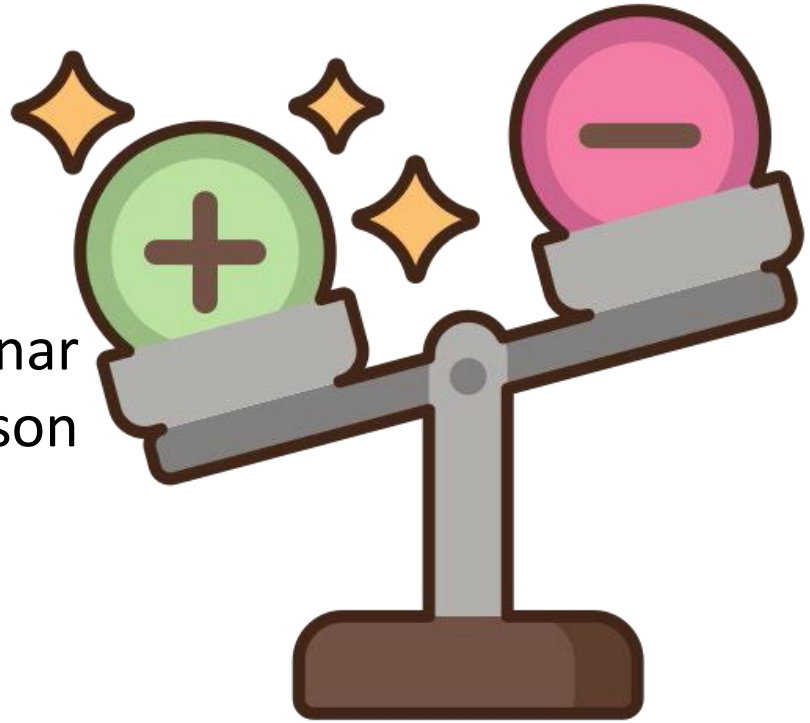
Instanciando el objeto

```
@Controller
public class IndexController {

    private MiServicio servicio = new MiServicio();

    @GetMapping({"/index", "", "/"})
    public String index(Model model) {
        model.addAttribute("objeto", servicio.operacion());
        return "index";
    }

}
```

Instanciando el objeto

Ventajas

- ✓ Podemos elegir que implementación usar.
- ✓ No tenemos que gestionar que implementaciones son prioritarias.



Desventajas

- ✓ Acoplamos nuestro código.
- ✓ Dificil de mantener.
- ✓ Se deben editar clases de diferentes niveles.

Usando la clase

```
@Controller
public class IndexController {

    @Autowired
    private MiServicio servicio;

    @GetMapping({"/index", "", "/"})
    public String index(Model model) {
        model.addAttribute("objeto", servicio.operacion());
        return "index";
    }

}
```

Usando la clase

Ventajas

- ✓ Podemos elegir que implementación usar.
- ✓ Se pueden usar anotaciones y delegar a Spring a que instancie el servicio correspondiente.

Desventajas

- ✓ Acoplamos nuestro código.
- ✓ Más costoso de mantener.
- ✓ En caso de tener varias implementaciones debemos especificar exactamente cuál es prioritaria.

Usando una interfaz

```java
public interface LibroService {
    List<LibroDTO> obtenerTodosLosLibros();
    void crearLibro(LibroDTO libro);
    void eliminarLibroPorISBN(String isbn);
}
```

```java
@Service
public class LibroServiceImplement implements LibroService {

    private List<LibroDTO> libros = new ArrayList<>();
    private List<LibroConCantidad> librosConCantidad = new ArrayList<>();
    long idCounter = 1L;

    @Override
    public List<LibroDTO> obtenerTodosLosLibros() {
        return libros;
    }
```

```java
@Controller
public class LibroController {

    @Autowired
    private LibroService libroService;

    // Listar libros
    @GetMapping(value = "/libros")
    public String listarLibros(Model model) {
        model.addAttribute("libros", libroService.obtenerTodosLosLibros());
        return "libros/mostrarLibros";
    }
```

Usando una interfaz

Ventajas

- ✓ Código desacoplado
- ✓ Fácil de mantener.
- ✓ Se pueden crear nuevas implementaciones fácilmente.

Desventajas

- ✓ ……

A los componentes/servicios se les puede asignar un nombre tal y como se muestra a continuación:

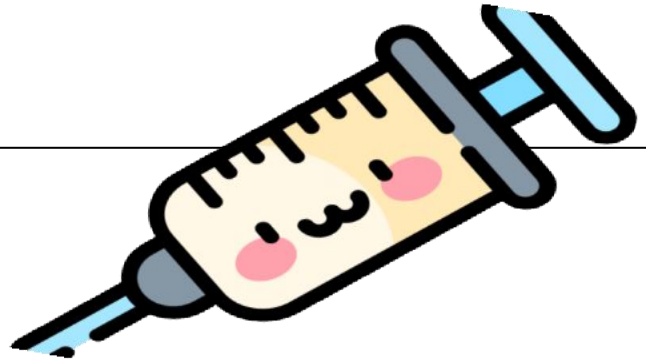
@Component("nombreComponente")
@Service("nombreComponente")

Adicionalmente, la inyección de dependencias se pude realizar de formas distintas:

- ✓ Por atributo
- ✓ Por constructor
- ✓ Por método set

Por atributo

Autowired, nos ayuda a seleccionar el candidato que queremos inyectar

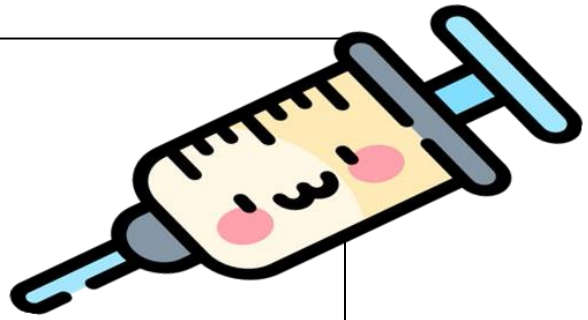


```java
@Controller
public class LibroController {

    @Autowired
    private LibroService libroService;

    // Listar libros
    @GetMapping(value = "/libros")
    public String listarLibros(Model model) {
        model.addAttribute("libros", libroService.obtenerTodosLosLibros());
        return "libros/mostrarLibros";
    }
```

Por constructor

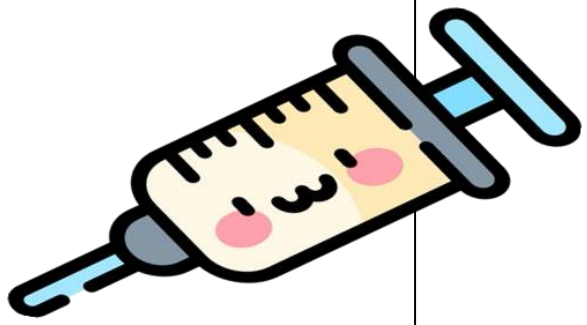
En este tipo se puede omitir el **Autowired**, aunque es preferible indicarlo.

```java
@Controller
public class LibroController {

    @Autowired
    private LibroService libroService;

    public LibroController(LibroService libroService) {
        this.libroService = libroService;
    }

    // Listar libros
    @GetMapping(value = "/libros")
    public String listarLibros(Model model) {
        model.addAttribute("libros", libroService.obtenerTodosLosLibros());
        return "libros/mostrarLibros";
    }
```



Por método set

```java
@Controller
public class LibroController {

    @Autowired
    private LibroService libroService;

    public void setServicio(LibroService libroService) {
        this.libroService = libroService;
    }

    // Listar libros
    @GetMapping(value = "/libros")
    public String listarLibros(Model model) {
        model.addAttribute("libros", libroService.obtenerTodosLosLibros());
        return "libros/mostrarLibros";
    }
```



ANOTACIÓN @PRIMARY

ANOTACIÓN @QUALIFIER

REGISTRO DE COMPONENTES CON LA ANOTACIÓN @BEAN