

# Acceso a Datos

---

UT5 – ACCESO WEB CON SPRING BOOT

INICIACIÓN A SPRING

# 1. ¿Qué es Spring?

---



- Spring es un framework que da soporte para el desarrollo de aplicaciones y páginas webs basadas en Java.
- Se trata de uno de los frameworks más populares, y ayuda a los desarrolladores back-end a crear aplicaciones con un alto rendimiento empleando objetos de java sencillos.
- Spring se puede considerar como el padre de los frameworks Java, ya que da soporte a varios frameworks como: Hibernate, Struts, Tapestry, EJB, JSF, entre otros.

## 2. Otros conceptos de Spring



---

### Otros conceptos asociados a Spring

- **Spring Web MVC** es un framework para implementar aplicaciones web bajo el patrón Modelo-Vista-Controlador. Nos permite (conjuntamente con Spring Core) realizar inversión de control.
- **Spring Data JPA** es una capa de abstracción sobre JPA (Java Persistence API). Añade a esta algunas características, como CRUDs automáticos, generación sencilla de consultas, ...

## 2. Otros conceptos de Spring

---



- **Spring Boot** nos permite configurar casi instantáneamente una aplicación Spring (por ejemplo, una aplicación web, que utilice un ORM, la base de datos, ...) aplicando el principio de *Convention over Configuration* (CoC).
- “Convención sobre Configuración, también conocido como CoC es un paradigma de programación de software que busca minimizar el número de decisiones que un desarrollador necesita hacer, ganando así en simplicidad pero no perdiendo flexibilidad por ello.”

## 2. Otros conceptos de Spring

---



Para construir una aplicación con Spring necesitamos:

- 1º) Crear un proyecto Maven/Gradle y descargar las dependencias necesarias.
- 2º) Desarrollamos la aplicación
- 3º) Desplegamos en un servidor.

Si pensamos en detalle en el tema , únicamente el paso dos es una tarea de desarrollo. Los otros pasos están más orientados a infraestructura. SpringBoot nace con la intención de simplificar los pasos 1 y 3 y que nos podamos centrar en el desarrollo de nuestra aplicación.

### 3. IDE

---



- Spring Tool Suite, mismo IDE que para UT4
- Página de descarga y referencias: <https://spring.io/tools>

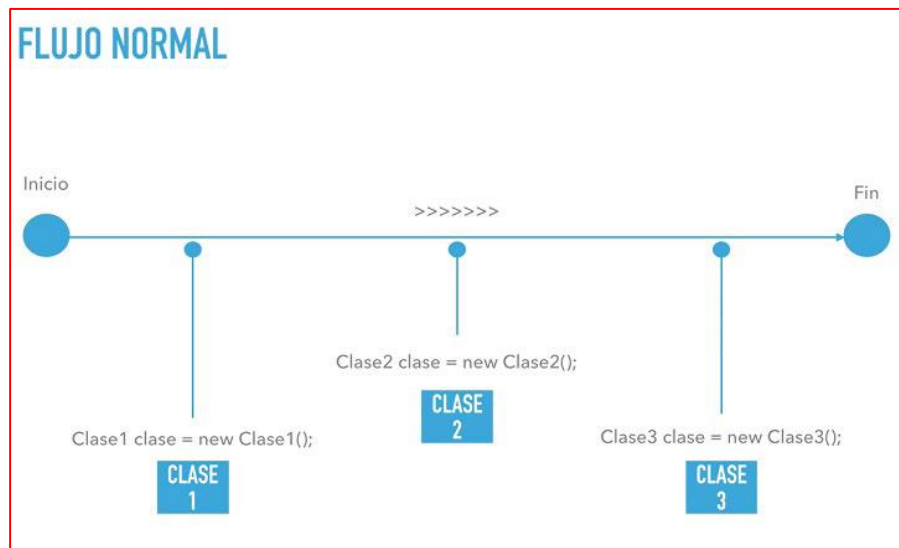


# 4. Inyección de dependencias



## Flujo normal:

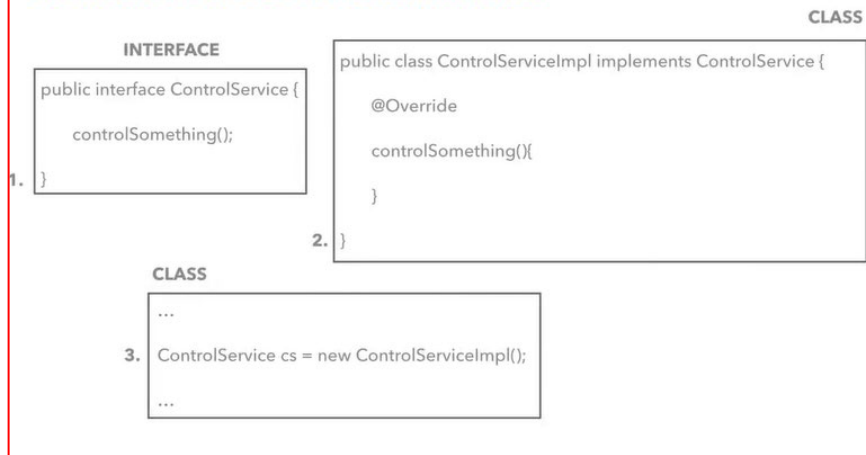
En el siguiente gráfico se muestra la ejecución de un proceso Java de principio a fin. En nuestros desarrollos de antes, a medida que lo íbamos necesitando creábamos objetos Java en tiempo de ejecución mediante el operador `new()`. El esquema muestra la creación de 3 objetos la clase 1, clase 2, y clase 3.



# 4. Inyección de dependencias



## CREACIÓN DE OBJETOS MEDIANTE 'NEW'



En una situación normal para crear este objeto en tiempo de ejecución haríamos:

```
ControlService cs= new ControlServiceImpl();
```

¿Y si un miembro de nuestro equipo crea una versión distinta de `ControlServiceImpl`? Habría que renombrar todas las partes del código donde aparezca la creación de objetos del tipo `ControlServiceImpl`. El código está muy acoplado y cualquier cambio que hagamos afectaría a toda la aplicación.

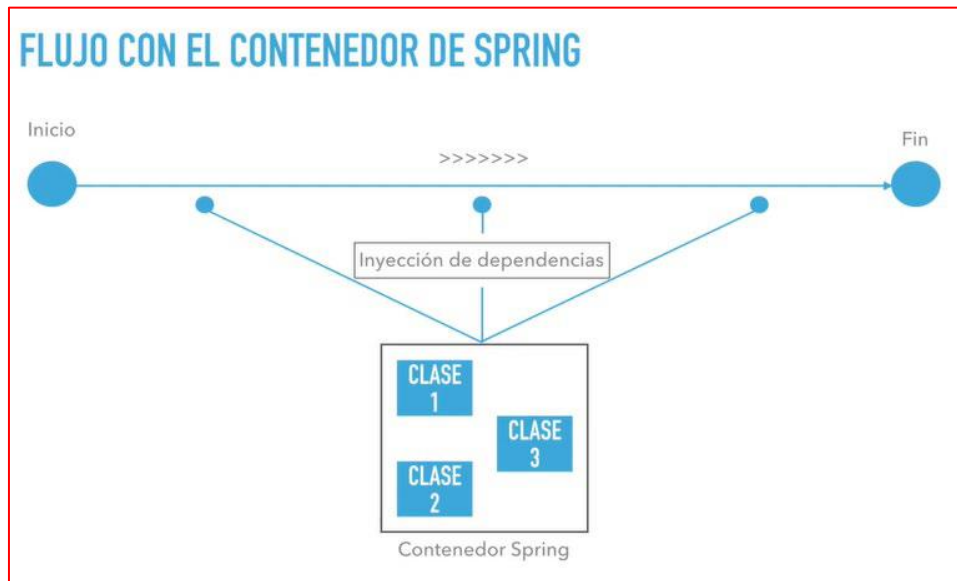


## 4. Inyección de dependencias



### Flujo con el contenedor de Spring:

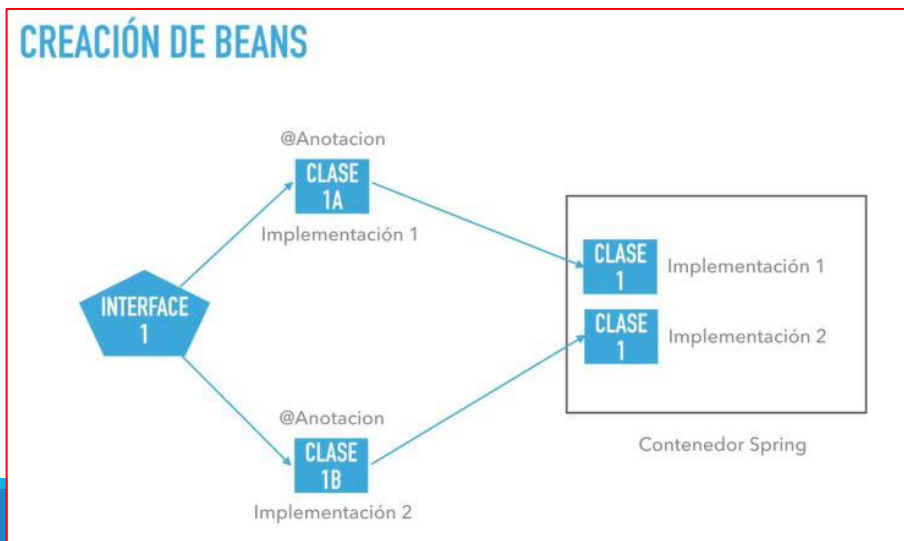
Aquí vemos el mismo flujo pero con el contenedor de Spring y la inyección de dependencias. A diferencia de crear objetos en tiempo de ejecución como hemos visto antes, Spring crea todos los objetos al arrancar la aplicación y los mete dentro de su **Container**. A la hora de hacer uso de uno de estos objetos, simplemente invocamos a la inyección de dependencias. A estos objetos los podemos llamar **beans**.



## 4. Inyección de dependencias



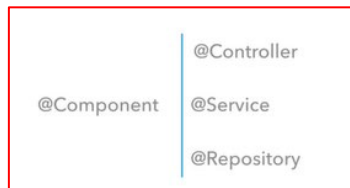
El siguiente gráfico representa una interfaz cualquiera y dos clases que la implementan y la clase 1A y la clase 1B. Al arrancar el servidor una de las tareas que realiza es escanear nuestro proyecto y todas las clases Java que tengan X anotación (@Anotacion) a nivel de clase las meterá en su contenedor y pasarán a ser llamadas beans de la aplicación.



## 4. Inyección de dependencias



Las anotaciones principales encargadas de esta función son las que se muestran a continuación:



@Component es el padre de todas y sus hijos @Controller, @Service y @Repository según nos encontremos desarrollando en una capa u otra de la aplicación Spring.

- @Controller es un componente con funcionalidades para la capa de presentación.
- @Service es un componente con funcionalidades para la capa de servicios.
- @Repository es un componente con funcionalidades para la capa de acceso a datos.

## 4. Inyección de dependencias

---



A continuación, en el cuadro 1 y en el 2 tenemos la misma interfaz y la misma clase que en el ejemplo utilizado para explicar la creación de objetos en tiempo de ejecución, la diferencia que vemos es que en el cuadro 2 hemos marcado a la clase `ControlServiceImpl` con la notación `@Service`. Esto le indicará a Spring que lo que debe de hacer es crear un objeto al arrancar el servidor y meterlo en su contenedor le podemos indicar como vemos en la imagen el nombre, en este caso `miServicio`.

## 4. Inyección de dependencias

---

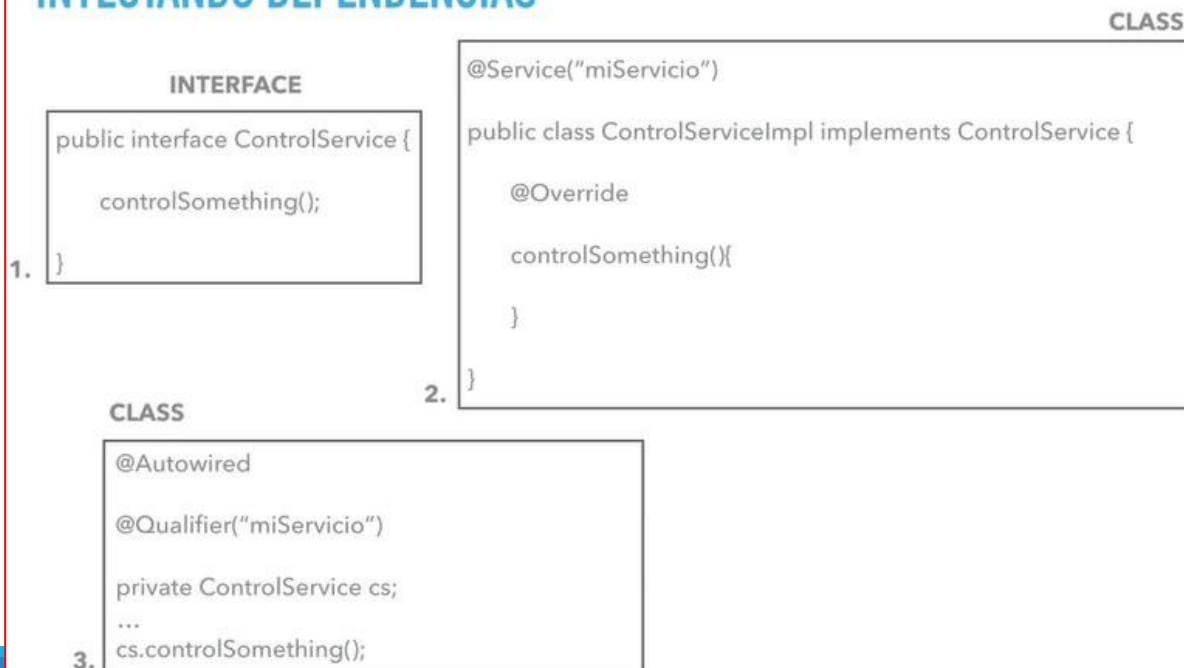


Ahora en el cuadro 3 aplicamos el concepto de inyección de dependencias. Desde una clase cualquiera declaramos la interfaz `private ControlService cs` y arriba ponemos `@Autowired`. Esto le indica a Spring que lo que acabamos de declarar se encuentra en su container y luego `@Qualifier` donde le decimos el nombre del **bean** que queremos obtener, Spring se encargará de inyectarlo y que lo podamos utilizar sin necesidad de usar el operador `new` es una de las grandes ventajas de la inyección de dependencias.

## 4. Inyección de dependencias



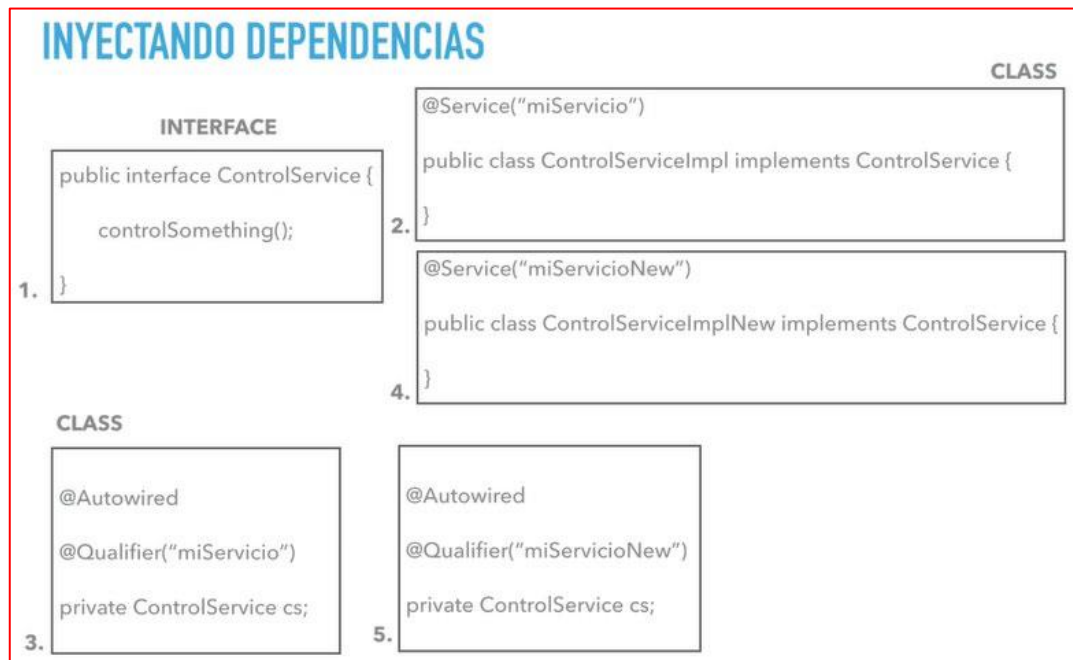
### INYECTANDO DEPENDENCIAS



## 4. Inyección de dependencias



En el siguiente ejemplo, en el cuadro 4 nos hemos declarado otro servicio y lo anotamos como `@Service("miServicioNew")`. Las clases del cuadro 3 y 5 son iguales, a diferencia del `@Qualifier` donde le decimos que coja beans distintos:



## 4. Inyección de dependencias

---



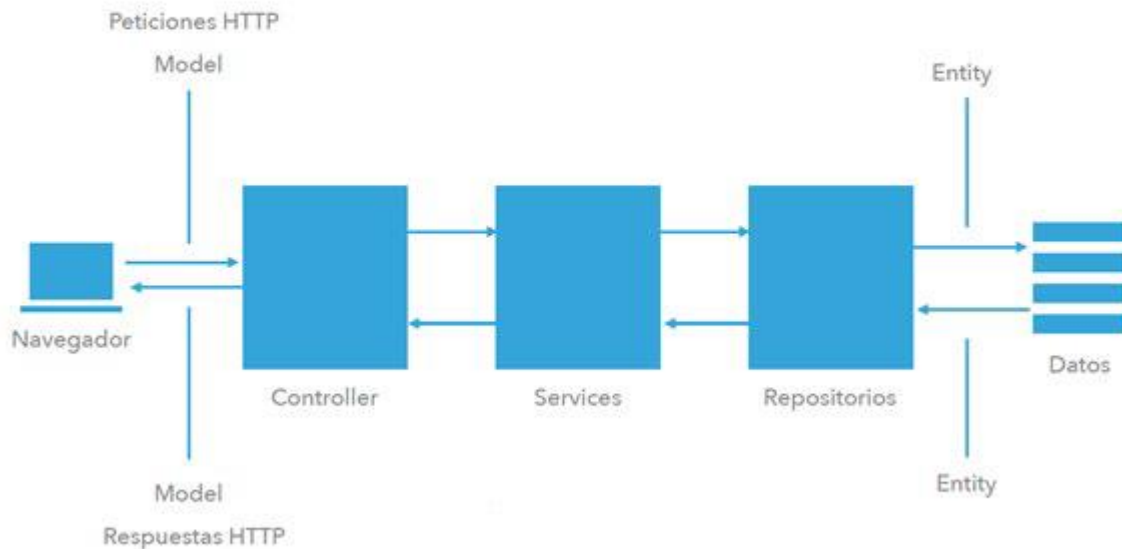
Lo que nos ofrece la inyección de dependencias es desacoplamiento, y también que los objetos son instanciados en el Contenedor DI y se inyectan donde sea necesario de forma que pueden ser reutilizados.



## 5. Arquitectura de una app



### ARQUITECTURA DE UNA APLICACIÓN WEB SPRING



## 5. Arquitectura de una app

---



- Si nos fijamos en la arquitectura de una aplicación web en Spring a la izquierda está el navegador (Mozilla Firefox, Google Chrome,..) este realizará peticiones a un Servidor Web que a su vez llamará a una base de datos. Los **Controllers**, los **Servicios** y los **Repositorios** pertenecen al servidor web y será nuestra aplicación de Spring.

## 5. Arquitectura de una app

---



- Las peticiones HTTP llevan consigo un **Model**, que no son más que datos que nos envían del front-end, estas peticiones se encargan de despacharlas los **Controllers** que van a transformar los datos a una clase Java a la que llamaremos Model.
- Los controladores llamarán a uno o varios servicios, estos tienen toda la lógica de la aplicación y es donde haremos las consultas a base datos correspondientes mediante los **Repositorios**, y aquí se inicia el flujo inverso: los Repositorios retornarán a los Servicios una Entity, los Servicios aplicarán más lógica de negocio y por último retornan el Model al controlador y este a su vez dará una respuesta HTTP al navegador.

## 5. Arquitectura de una app

---



- En definitiva, los Controllers, los Modelos, Services, las Entities, y los Repositorios son clases Java que se van llamando unas a otras hasta completar este flujo y por último aunque no sepas que es un controller un service on repository es importante que sepas el flujo que sigue cada petición web desde que sale del navegador hasta que regresa.

## 5.1. Ejemplo proyecto MVC




Vamos a crearnos un proyecto de Spring pero utilizando el generador de proyectos que se nos proporciona en <https://start.spring.io> :

Seleccionamos la opción de un proyecto Maven, lenguaje Java versión 17, la versión estable de Spring 3.2.0, añadimos los metadatos al proyecto y añadimos las siguientes dependencias (si se nos olvida alguna la podemos añadir después en el pom.xml):

- **Web:** Spring Web: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- **Template Engines:** Thymeleaf: Thymeleaf templating engine (<https://es.wikipedia.org/wiki/Thymeleaf> )
- **SQL:** Spring Data JPA y MySQL Driver

# 5.1. Ejemplo proyecto MVC



 **spring** initializr

**Project**

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy

☒ Maven

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 3.2.1 (SNAPSHOT) ☒ 3.2.0 ☐ 3.1.7 (SNAPSHOT) ☐ 3.1.6

**Project Metadata**

Group

com.iesvjp

Artifact

SpringContactMVC

Name

SpringContactMVC

Description

Demo project for Spring Boot

Package name

com.iesvjp

Packaging

☒ Jar ☐ War

Java

☐ 21 ☒ 17

**Dependencies**

**Spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Thymeleaf** TEMPLATE ENGINES

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

**Spring Data JPA** SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**MySQL Driver** SQL

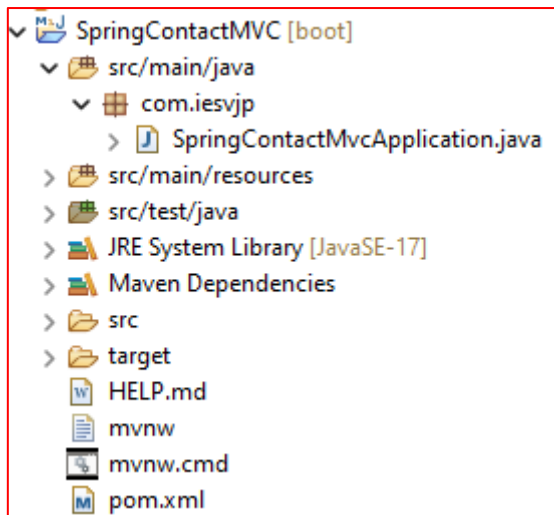
MySQL JDBC driver.

ADD DEPENDENCIES... CTRL + B

## 5.1. Ejemplo proyecto MVC



Pulsamos el botón **Generate** y se nos descarga un zip con la estructura del proyecto, lo podemos utilizar en cualquier entorno de desarrollo. Lo descomprimos, y una vez guardado abrimos Eclipse/STS y lo importamos (Import Maven Project).

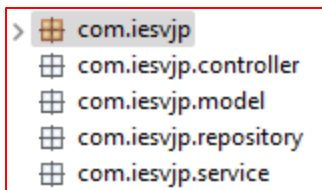


## 5.1. Ejemplo proyecto MVC



### Estructura de una aplicación Spring Boot:

Carpeta src/main/java: aquí estarán todas las clases que programemos, nos crearemos paquetes para ello: *com.iesvjp.controller*, *com.iesvjp.service*, *com.iesvjp.repository* y *com.iesvjp.model*

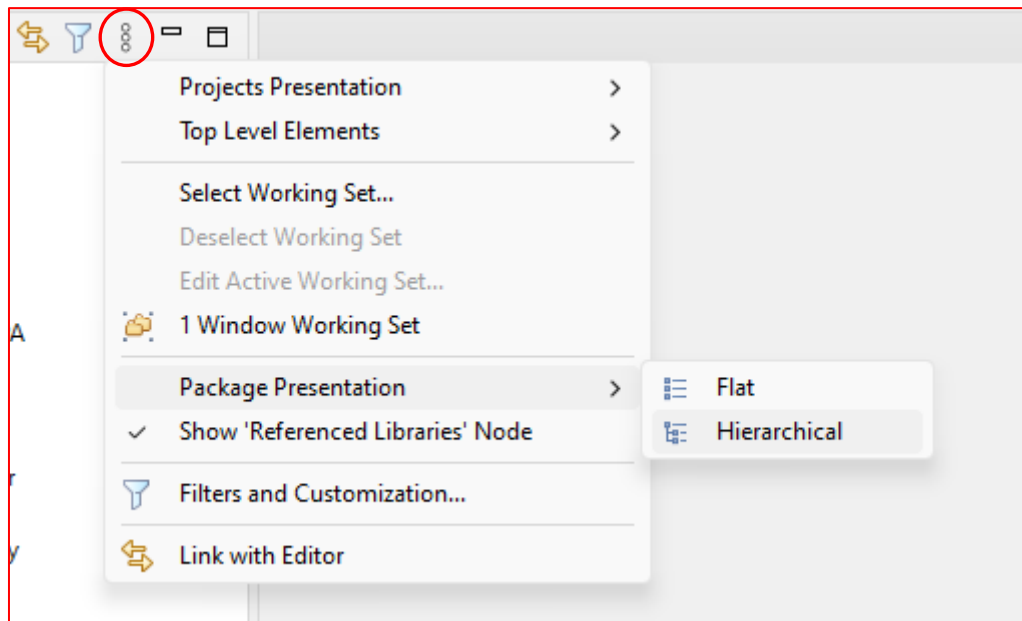




## 5.1. Ejemplo proyecto MVC



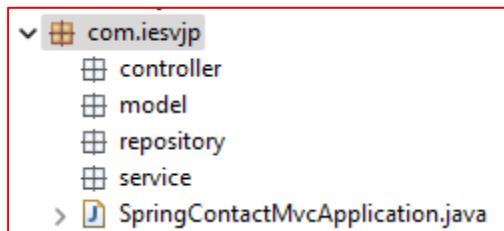
Y ordenamos los paquetes en forma jerárquica:



## 5.1. Ejemplo proyecto MVC



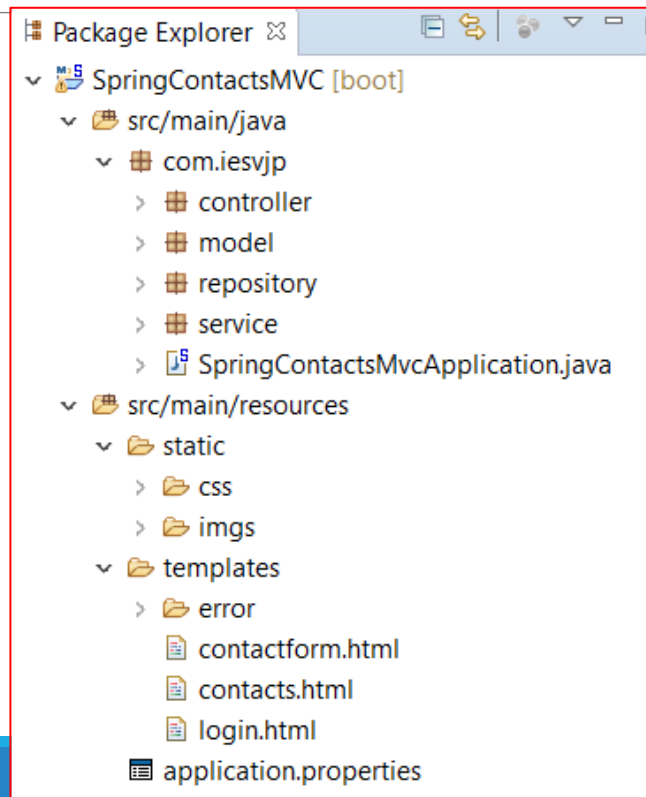
Este sería el resultado de ordenarlo jerárquicamente:



## 5.1. Ejemplo proyecto MVC



En la carpeta **src/main/resources**: guardaremos todos los recursos estáticos, en **static** guardaremos los archivos .css, .js, y en la subcarpeta **templates** guardaremos las vistas, y en el archivo **properties** guardaremos la configuración de nuestra aplicación.



## 5.1. Ejemplo proyecto MVC



Las propiedades de configuración más comunes de Spring podemos consultarlas en la siguiente página:

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

Para la configuración de las propiedades podemos utilizar el formato de propiedad-valor del tipo de archivo **application.properties**:

application.properties application.yml

```
1 spring.datasource.url= jdbc:mysql://localhost/springboot
2 spring.datasource.username= root
3 spring.datasource.password=
4 spring.jpa.show-sql= true
5 spring.jpa.hibernate.ddl-auto= update
6 |
```

¡OJO! Si indicamos otra BDD, necesitamos una nueva conexión

## 5.1. Ejemplo proyecto MVC

---



- En la carpeta **src/test/java**: tendremos todos los test de la aplicación y debería tener la misma estructura de clases y carpetas que en `src/main/java`
- **src**: nuestra aplicación en local
- **target**: recursos que genera Maven en su ciclo de vida
- **pom.xml**: donde tenemos las características del proyecto y sus dependencias

## 5.1. Ejemplo proyecto MVC

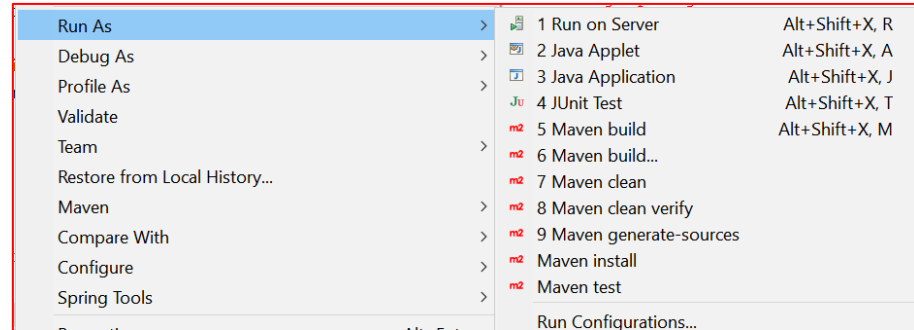


Una vez importado es necesario bajarnos todas las dependencias de Maven, seguiremos los siguientes pasos, nos pondremos encima del proyecto, botón derecho:

1º) Run AS → Maven clean

2º) Run AS → Maven install

3º) Maven → Update Project



## 5.1. Ejemplo proyecto MVC



---

### Posibles errores:

Si nos da una Excepción de **Maven MojoExecutionException**, serán necesario antes de instalar ejecutar la opción de **Maven generate-sources**

Para más información sobre MAVEN:

- <http://jarroba.com/maven/>
- <http://jarroba.com/maven-en-eclipse/>
- <https://www.adictosaltrabajo.com/tutoriales/maven/>

## 5.1. Ejemplo proyecto MVC



### **Posibles errores:**

Si cuando ejecutamos Maven install nos muestra el siguiente error, se debe a un bug de las últimas versiones de Spring:

**[ERROR] Failed to execute goal org.apache.maven.plugins:maven-resources-plugin:3.2.0:resources (default-resources) on project SpringContactsMVC: Input length = 1 -> [Help 1]**

Lo solucionaremos añadiendo al pom.xml el siguiente plug-in:

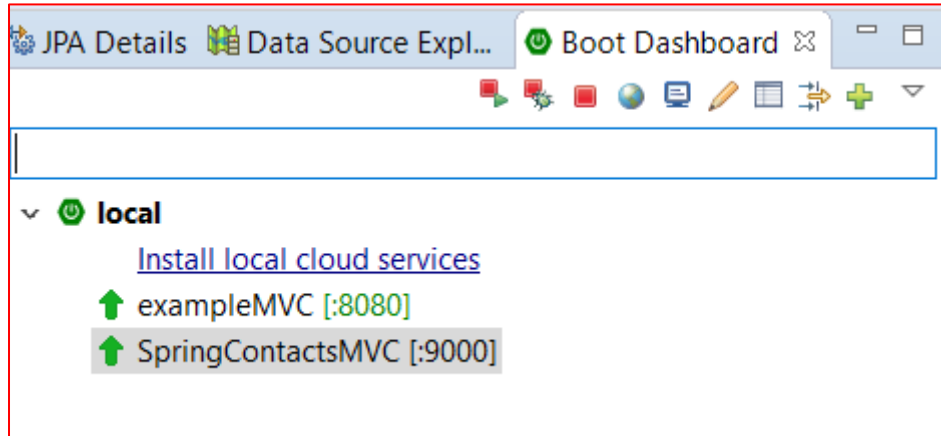
```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-resources-plugin</artifactId>
  <version>3.1.0</version>
</plugin>
```



## 5.1. Ejemplo proyecto MVC



Para arrancar la aplicación y comprobar que no haya ningún error, abrimos el **Boot Dashboard** (Window→Show View), nos situamos encima del nombre de la aplicación y pulsamos sobre el icono de play



## 5.1. Ejemplo proyecto MVC



Si nos diese un error porque el puerto 8080 está ocupado, podemos cambiar el puerto por defecto 8080 de la aplicación de Spring Boot al puerto 9000, añadiendo la siguiente línea al archivo de `application.properties`:

```
application.properties ✕  
1 server.port=9000  
2 |
```

# 5.1. Ejemplo proyecto MVC



Si todo ha ido correcto se mostrará esta pantalla de Spring Boot:

```
SpringContactsMVC [boot]
├── Spring Elements
├── src/main/java
│   └── com.iesvp
│       └── App
│           ├── SpringContactsMvcApplication.java
│           ├── controller
│           ├── model
│           ├── repository
│           └── service
└── src/main/resources

JPA Details Data Source Expl... Boot Dashboard
local
  Install local cloud services
  ↑ exampleMVC [8080]
  ↑ SpringContactsMVC [9000]

Spring Boot (v2.1.8.RELEASE)

2019-09-08 13:56:08.238 INFO 15236 --- [main] c.i.App.SpringContactsMvcApplication : Starting SpringContactsMvcApplication on AnArribas with
2019-09-08 13:56:08.242 INFO 15236 --- [main] c.i.App.SpringContactsMvcApplication : No active profile set, falling back to default profiles
2019-09-08 13:56:09.133 INFO 15236 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9000 (http)
2019-09-08 13:56:09.137 INFO 15236 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2019-09-08 13:56:09.157 INFO 15236 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.24]
2019-09-08 13:56:09.252 INFO 15236 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2019-09-08 13:56:09.252 INFO 15236 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in
2019-09-08 13:56:09.440 INFO 15236 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2019-09-08 13:56:09.518 WARN 15236 --- [main] ion$DefaultTemplateResolverConfiguration : Cannot find template location: classpath:/templates/ (p
2019-09-08 13:56:09.630 INFO 15236 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9000 (http) with context patl
2019-09-08 13:56:09.633 INFO 15236 --- [main] c.i.App.SpringContactsMvcApplication : Started SpringContactsMvcApplication in 1.725 seconds ('
```

# Dudas y preguntas

---

