# Java

# Enterprise Edition

Platzi

# Java SE

(Java Standard Edition)

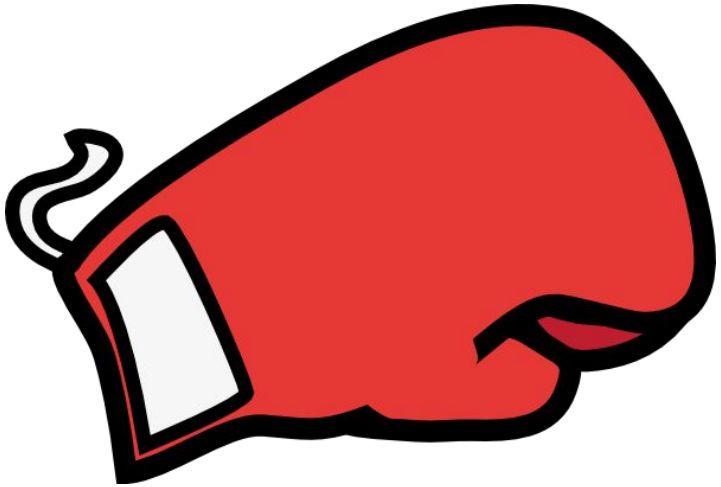
Platzi

# Java EE

(Java Enterprise Edition)

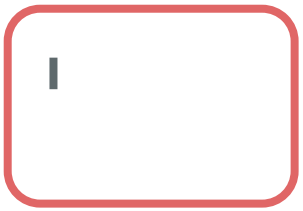
Platzi

# Java EE vs. Java SE

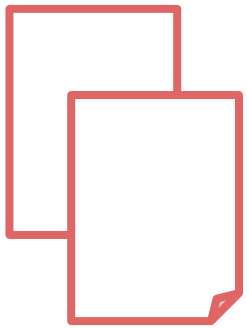


Platzi

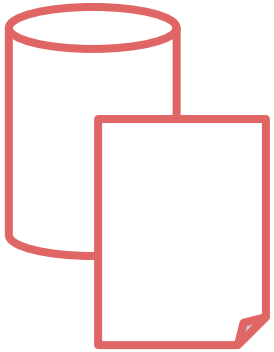
# Java SE



Vista?



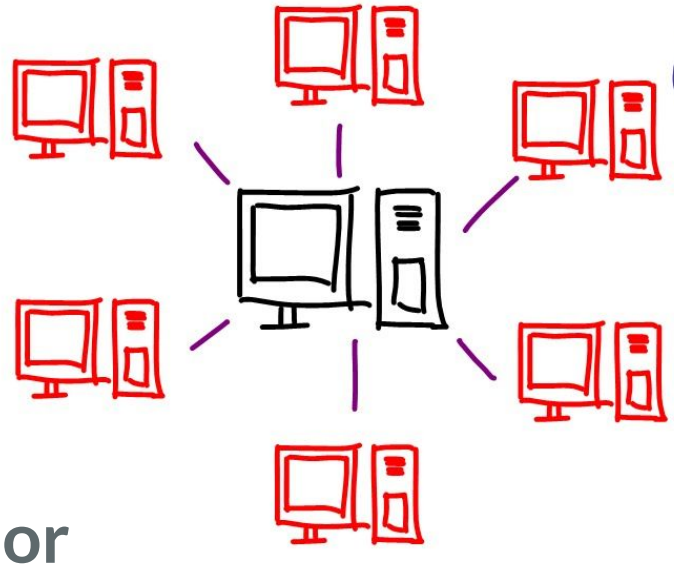
Controlador



Modelo

# JEE (Java Enterprise Edition)

- Nace porque se necesitan **aplicaciones distribuidas**, transaccionales y portables que usan

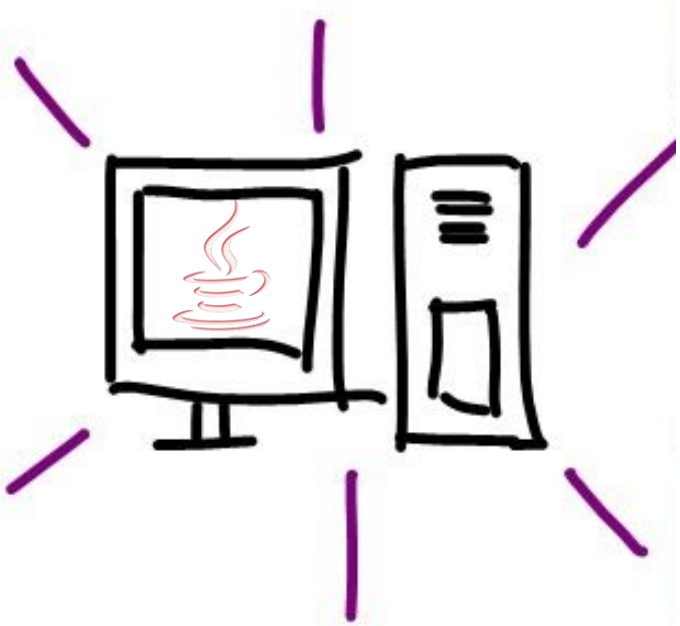


**todas las capacidades de un servidor**

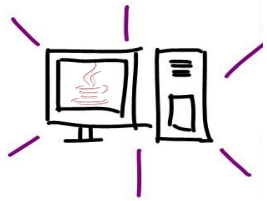
# JEE (Java Enterprise Edition)

- Servidor

  velocidad, seguridad, confiabilidad



# Servidores de Aplicaciones



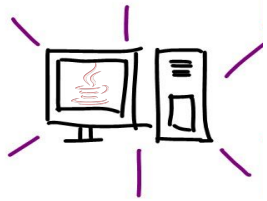
- WebLogic

  **Oralce**
- JBoss Enterprise Application Platform

  **Red Hat**
- WebSphere

  **IBM**

# Servidores de Aplicaciones



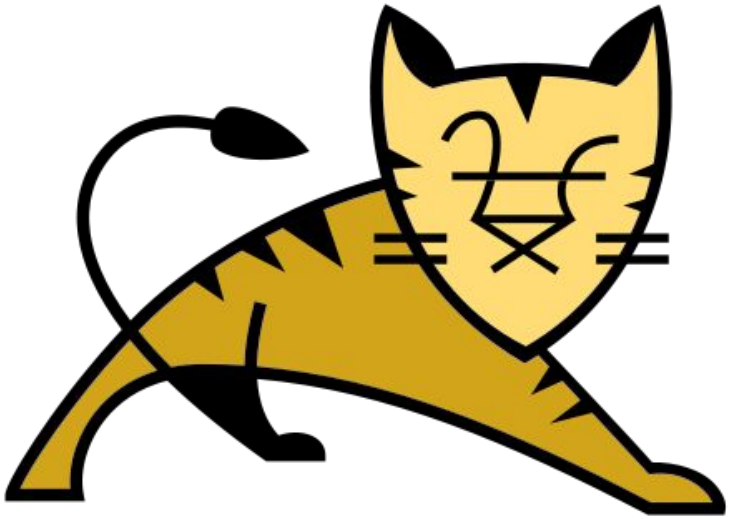
- JOnAS

  **ObjectWeb**
- Wildfly

  **Versión de JBoss por la comunidad**
- GlassFish

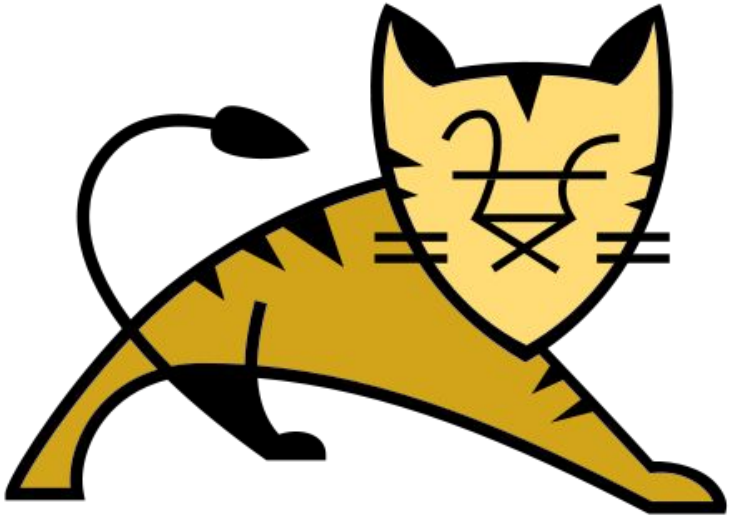
  **Oracle**
- Gernónimo y TomEE

  **Apache**



# Apache Tomcat

Platzi



# Apache Tomcat

Software que se implementa en un servidor web exclusivo para trabajar con Java Servlets

Platzi

# Contenedor Web vs. Servidor Web

## Contenedor Web

Ejecución de Servlets ()

## Servidor Web

Ejecuta muchos tipos de aplicaciones web, recursos, gestiona peticiones, y además también puede tener un Contenedor Web

# IDE para Java EE

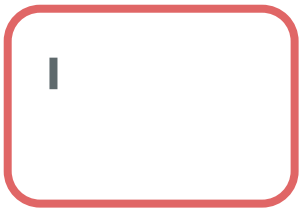

# Hola mundo!

Platzi

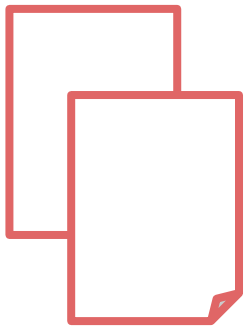
# MVC Java EE

Platzi

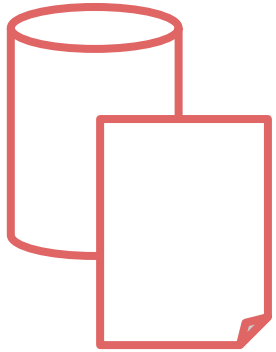
# Java SE



Vista?
Terminal



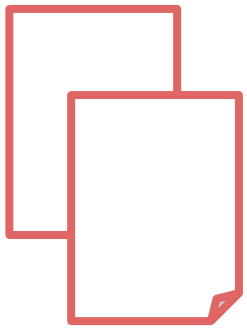
Controlador
Clases java



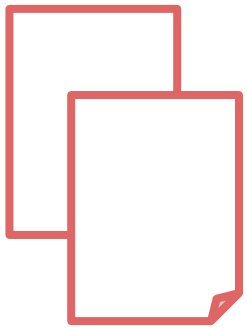
Modelo
POJO

Platzi

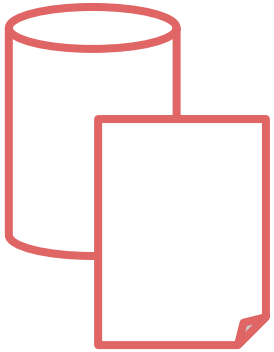
# Java EE



Vista
html / jsp



Controlador
Servlets



Modelo
POJO / Bean

# JSP

Platzi

# JSP

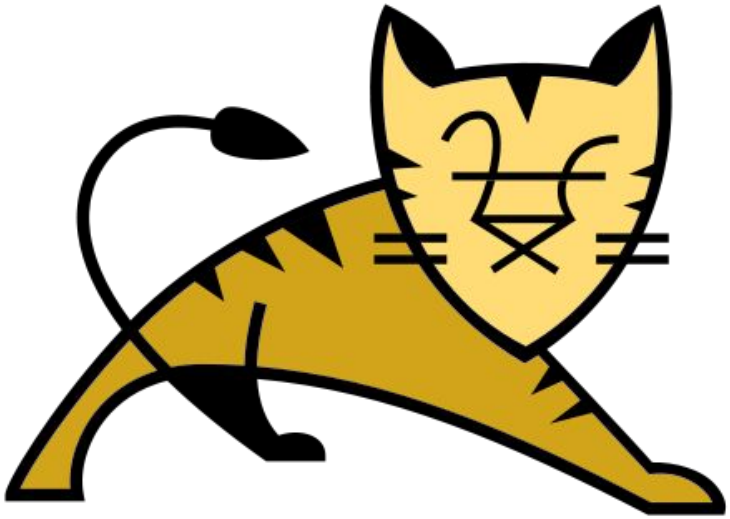
- Java Server Pages
- Diseñado para crear páginas web dinámicas
- HTML + JavaScript + CSS +**Java**
- **`<% Date d = new Date() %>`**

```
<body>
    <h1>TU NOMBRE ES: 100 VECES</h1><P>
        <%
        for(int i=0;i<=100;i++)
            {
            out.write("HOLA "+request.getParameter("nom")+i+"<p>");
            }

        %>
</body>
```

Platzi

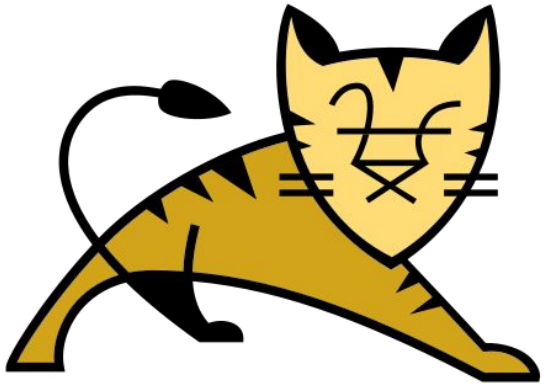
# Corriendo la Aplicación

Platzi



# Apache Tomcat

Platzi



# Apache Tomcat

https://tomcat.apache.org/download-80.cgi#8.0.42

# Servlet

Platzi

# Servlet

- Clase con super poderes
- Hereda de HttpServlet
- Tiene los métodos
  - doGet
  - doPost

```java
/**
 * Servlet implementation class Hello
 */
@WebServlet("/hello")
public class Hello extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

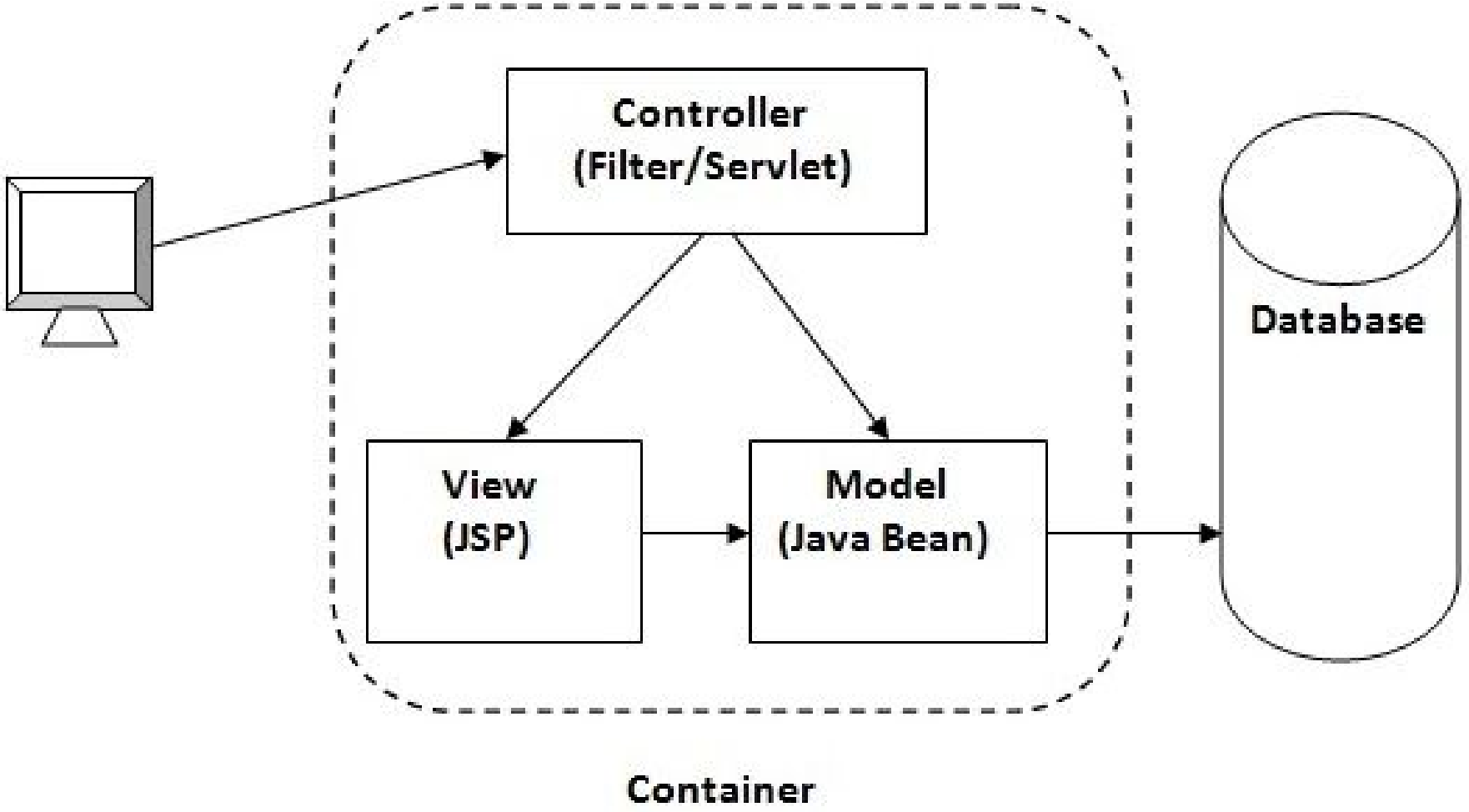
# Bean

Platzi

# Bean

- Al menos un Constructor sin argumentos
- Atributos de clase deben ser privados
- Sus propiedades deben ser accesibles mediante métodos get y set
- Debe ser serializable

```
public class Alumno implements Serializable {

    private String nombre;
    private String matricula;

    public Alumno() {

    public String getNombre() {

    public void setNombre(String nombre) {

    public String getMatricula() {

    public void setMatricula(String matricula) {

}
```


Controller
(Filter/Servlet)
Database
View
(JSP)
Model
(Java Bean)
Container

# Hola Login!

Platzi

# Empaquetados

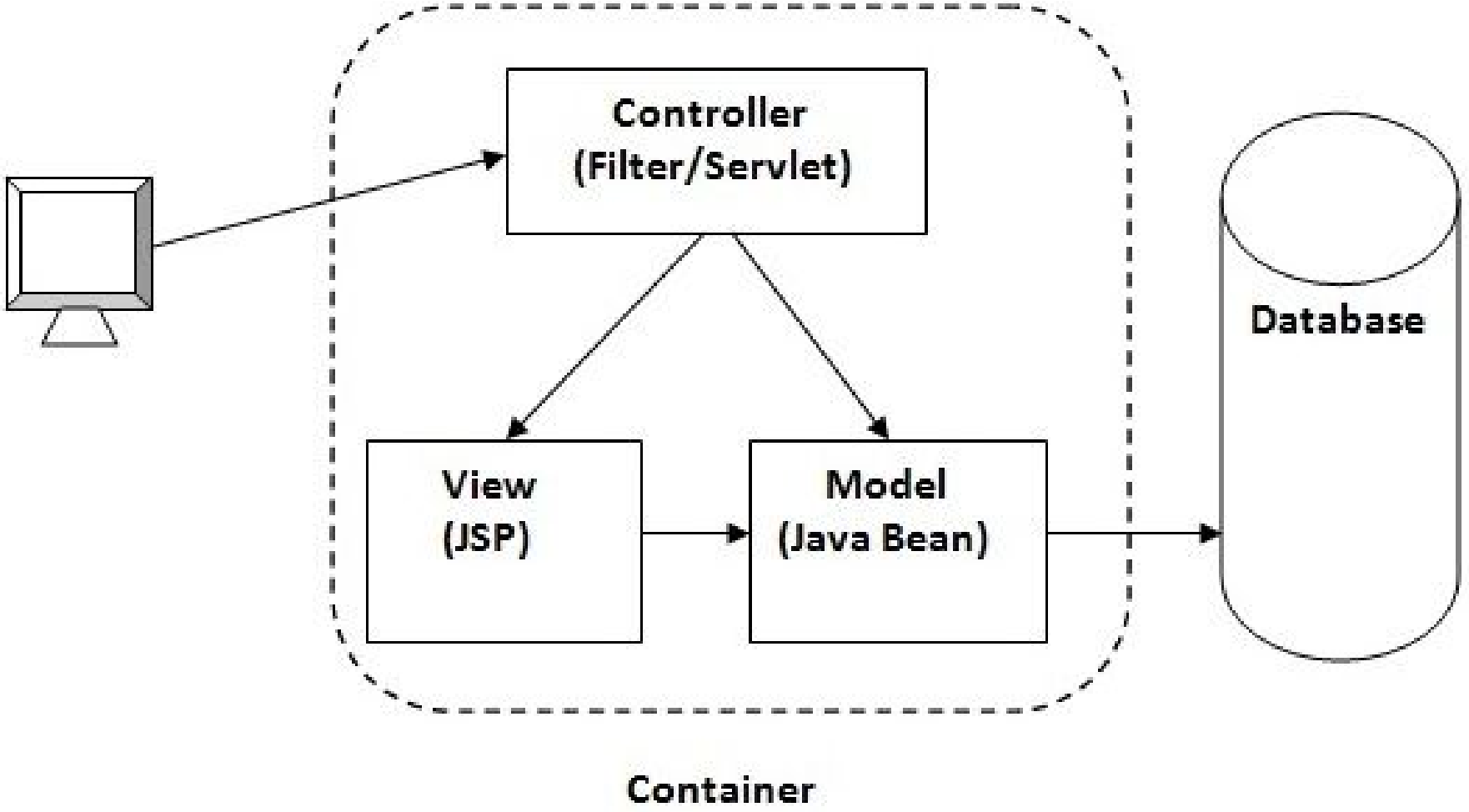
Platzi

# War

# WAR

## Web application ARchive

Platzi


Controller
(Filter/Servlet)
Database
View
(JSP)
Model
(Java Bean)
Container

# Asistentes inteligentes

Platzi

# Asistentes Inteligentes

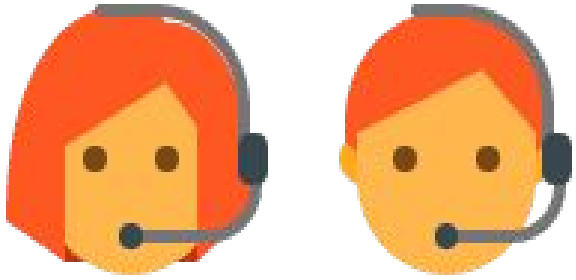
- Nos ayudan a crear proyectos basados en plantillas
- Descargan librerías de terceros (JAR)
- Crean todos los componentes y los archivos ejecutables, JAR, WAR, etc.

# Maven

Platzi

# Maven

- Es un proyecto de Apache
- Configuración y Construcción basado en XML
- Project Object Model pom.xml
  - **Describe el proyecto a construir**
  - **Dependencias**
  - **Compilación del código**
  - **Empaquetado**



# Maven - Estructura de archivos

# Maven - Project Object Model pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apa
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.platzi</groupId>
  <artifactId>hola</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>hola</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

Platzi

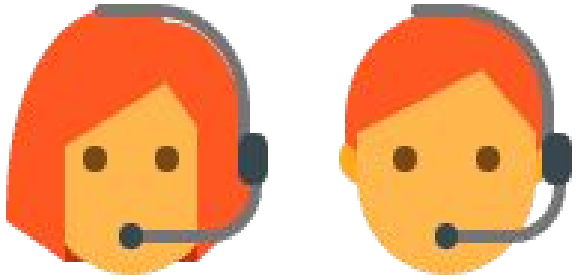
# Gradle

Platzi

# Gradle

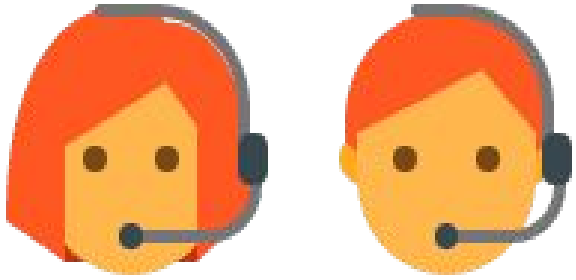
- Está basado en Groovy
- Usa Domain Specified Language - Json

```
static mapping = {
  table 'person'
  columns {
    name column:'name'
  }
}
```



# Gradle

- build.gradle
  - **Dependencias**
  - **Compilación del código**
  - **Empaquetado**



Platzi

# Gradle - Estructura de archivos

# Gradle - build.gradle

```
apply plugin: 'java'
apply plugin: 'eclipse'

sourceCompatibility = 1.5
version = '1.0'
jar {
    manifest {
        attributes 'Implementation-Title': 'Gradle Quickstart',
                   'Implementation-Version': version
    }
}

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'commons-collections', name: 'commons-collections', version: '3.2'
    testCompile group: 'junit', name: 'junit', version: '4.+'
}

test {
    systemProperties 'property': 'value'
}

uploadArchives {
    repositories {
       flatDir {
           dirs 'repos'
       }
    }
}
```

# Aplicaciones Orientadas a Servicios

Platzi

# Orientadas a presentación

## Genera sitios web dinámicos

Platzi

# Orientadas a servicios

## Se enfocan solo en el backend y usan el formato JSON como vista

# Rest API

Platzi

# Nuestro IDE

Platzi

# Spring Tool Suite

- Basado en Eclipse
- Creado por la comunidad SpringSource
- Plantillas de proyectos basados en Spring
- Integración con Maven


# https://spring.io/tools

Platzi

# La base del proyecto

## Base de Datos

Platzi

# Diagramas

# Hibernate

Platzi

# ORM

## Object Relational Mapping

Platzi

# ORM - Object Relational Mapping

- Es una herramienta de Mapeo Objeto Relacional

- Facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos

HIBERNATE

# Hibernate


| Camion | |
|---|---|
| id | int |
| matricula | String |
| potencia | double |
| tipo | String |

# ORM Hibernate

- Manipular los datos en la base de datos, operando sobre **objetos**.

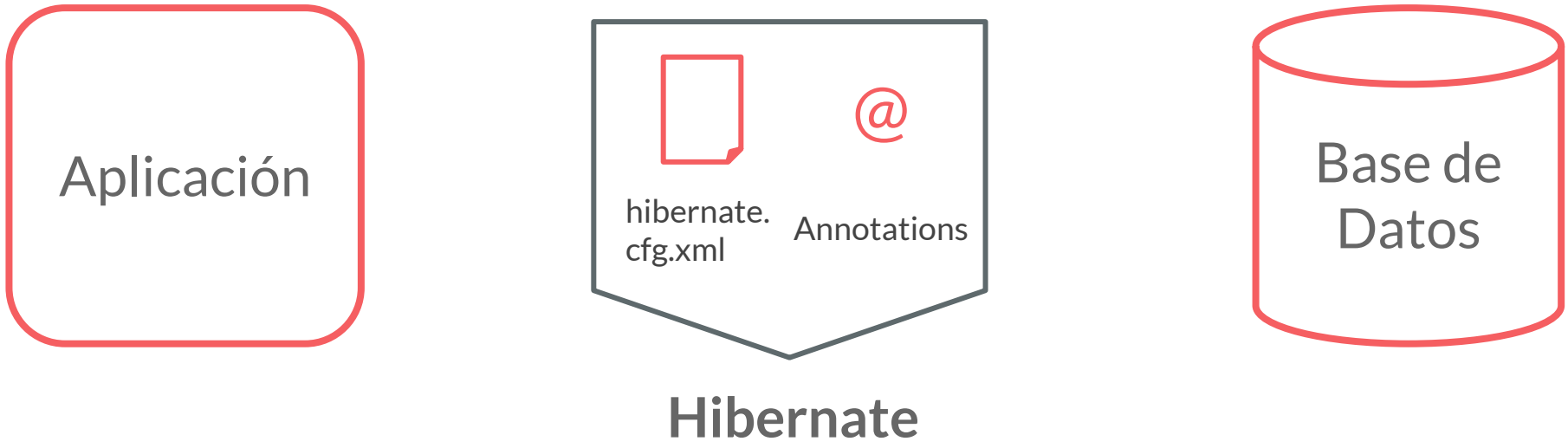
# ORM Hibernate

- Busca solucionar el problema de la diferencia entre **los dos modelos de datos coexistentes en una aplicación**

# ORM Hibernate

- Definir tipos de datos
- Genera las sentencias SQL
- Ayuda al manejo de los resultados de las sentencias SQL

# Hibernate


Aplicación
hibernate.
cfg.xml
@
Annotations
Base de
Datos
Hibernate

# Configuración Hibernate

Platzi

# Hibernate integración al IDE



- Help -> Install New Software
- Click en Add.
- Ir al sitio: **http://download.jboss.org/jbosstools/updates/stable/**

Platzi

# Hibernate integración al IDE



- Escribir:

https://download.jboss.org/jbosstools/neon/stable/updates/

Platzi

# Hibernate integración al IDE



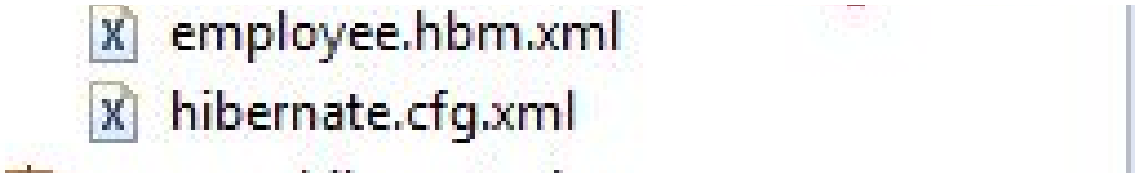
- Dentro de **JBoss Web and Java EE Development**
- Selecciona **Hibernate Tools**
- Click en Siguiente

# Mapeo en Hibernate

Platzi

# Mapeo Hibernate

- Para cada clase que queramos persistir debemos crear un archivo XML
- hbm.xml



# Infierno XML

Platzi

# Annotations @

# @Entity

Se aplica a la clase e indica que esta clase Java es una entidad a persistir

# @Table

Se aplica a la clase e indica el nombre de la tabla de la base de datos donde se persistirá la clase

# @Column

Se aplica a una propiedad Java e indica el nombre de la columna de la base de datos en la que se persistirá la propiedad

# @Id

Se aplica a una propiedad Java e indica que este atributo es la clave primaria

# @GeneratedValue

Esta anotación indica que Hibernate deberá generar el valor de la clave primaria

# Asociaciones Hibernate

Platzi

# Uno a Uno
# One to One

# Unidireccional

# Relaciones Hibernate - One to One

- Un objeto tenga una referencia a otro objeto de forma que al persistirse el primer objeto también se persista el segundo.

```
@OneToOne(cascade=CascadeType.ALL)
```

# Bidireccional

Platzi

# Relaciones Hibernate - One to One

- Los dos objetos se persisten mutuamente, ambos tienen referencias uno de otro

```
@OneToOne(cascade=CascadeType.ALL)
```

# Uno a Muchos
# One to Many

Platzi

# Relaciones Hibernate - One to Many

- Un objeto tenga una lista de otros objetos de forma que al persistirse el objeto principal también se persista la lista de objetos

# @JoinColumn

## El nombre de la columna que une las tablas

Platzi

# Relaciones Hibernate - One to Many

## Entity Owner

```
@OneToMany(mappedBy="entity", cascade = CascadeType.ALL)
```

## Entity No Owner

```
@ManyToOne(optional = true, fetch = FetchType.EAGER)
@JoinColumn(name="primary_key")
```

# Muchos a Muchos
# Many to Many

Platzi

# Relaciones Hibernate - Many to Many

- Un objeto A tenga una lista de otros objetos B y también que el objeto B a su vez tenga la lista de objetos A

# Relaciones Hibernate - Many to Many

## Entity Owner

```
@ManyToMany(cascade = {CascadeType.ALL})

@JoinTable(name="entityMapped",
joinColumns={@JoinColumn(name="campoTablaPadre")},
inverseJoinColumns={@JoinColumn(name="campoTablaHijo")})
```

## Entity No Owner

```
@ManyToMany(cascade = {CascadeType.ALL},mappedBy="entityName")
```

# Acción en Hibernate

Platzi

# Sesiones en Hibernate

- SessionFactory sessionFactory;
- Configuration configuration;
- Session session;

Platzi

# Transacciones en Hibernate

```
session.beginTransaction();

session.save(media);

session.getTransaction().commit();
```

# Transacciones en Hibernate

```
Camion camion = new Camion("ABC123", 2.0, "el tipo", 4.5);
Session session=sessionFactory.openSession();
session.beginTransaction();
session.save(camion);
session.getTransaction().commit();
```

# DAO's en Hibernate

Platzi

# DAO

## Data Access Object

Platzi

# DAO - Data Access Object

- Patrón de diseño
- verlo en forma de API
- API consiste en métodos CRUD (Create, Read, Update y Delete).

```
protected void create() {
    // code to save a book
}

protected void read() {
    // code to get a book
}

protected void update() {
    // code to modify a book
}

protected void delete() {
    // code to remove a book
}
```

# hql
# Hibernate

Platzi

# hql

## Hibernate Query Language

Platzi

# Hibernate - HQL

- Se parece a SQL. Sin embargo, comparado con SQL,

- Es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación.

# ¿Qué es un Framework?

Platzi

# Inversión de Control

Platzi

# Inversión de Control
# Principio de Hollywood?

- *No nos llames, nosotros te llamaremos*

Platzi

# Inversión de control

- **Librería vs. Framework**
- **Librería:** conjunto de clases, métodos etc. que son invocadas por el flujo del programa y que posteriormente devuelven el control a este.

# Inversión de control - Librería

- 1. Nuestro  código invoca la librería.
- 2. Se ejecuta el contenido de la librería.
- 3. Regresa al flujo de nuestro código.

# Inversión de control - Librería

- 1. Nuestro  código invoca la librería.
- 2. Se ejecuta el contenido de la librería.
- 3. Regresa al flujo de nuestro código.

## Programación Secuencial

# Inversión de control

- **Librería vs. Framework**
- **Framework:** Un framework controla el flujo del código, él decide cuándo llamar nuestro código

# Inversión de control - Framework

- 1. Se invoca la librería y el código se queda escuchando
- 2. Al darse un evento específico se ejecuta nuestro código
- 3. Regresa al flujo de la librería para continuar escuchando

# Inversión de control - Framework

- 1. Se invoca la librería y el código se queda escuchando
- 2. Al darse un evento específico se ejecuta nuestro código
- 3. Regresa al flujo de la librería para continuar escuchando

## Programación por Eventos

# Librería - Programación Secuencial

> Escribe tu nombre:

> Anahí Salgado

> Escribe tu email:

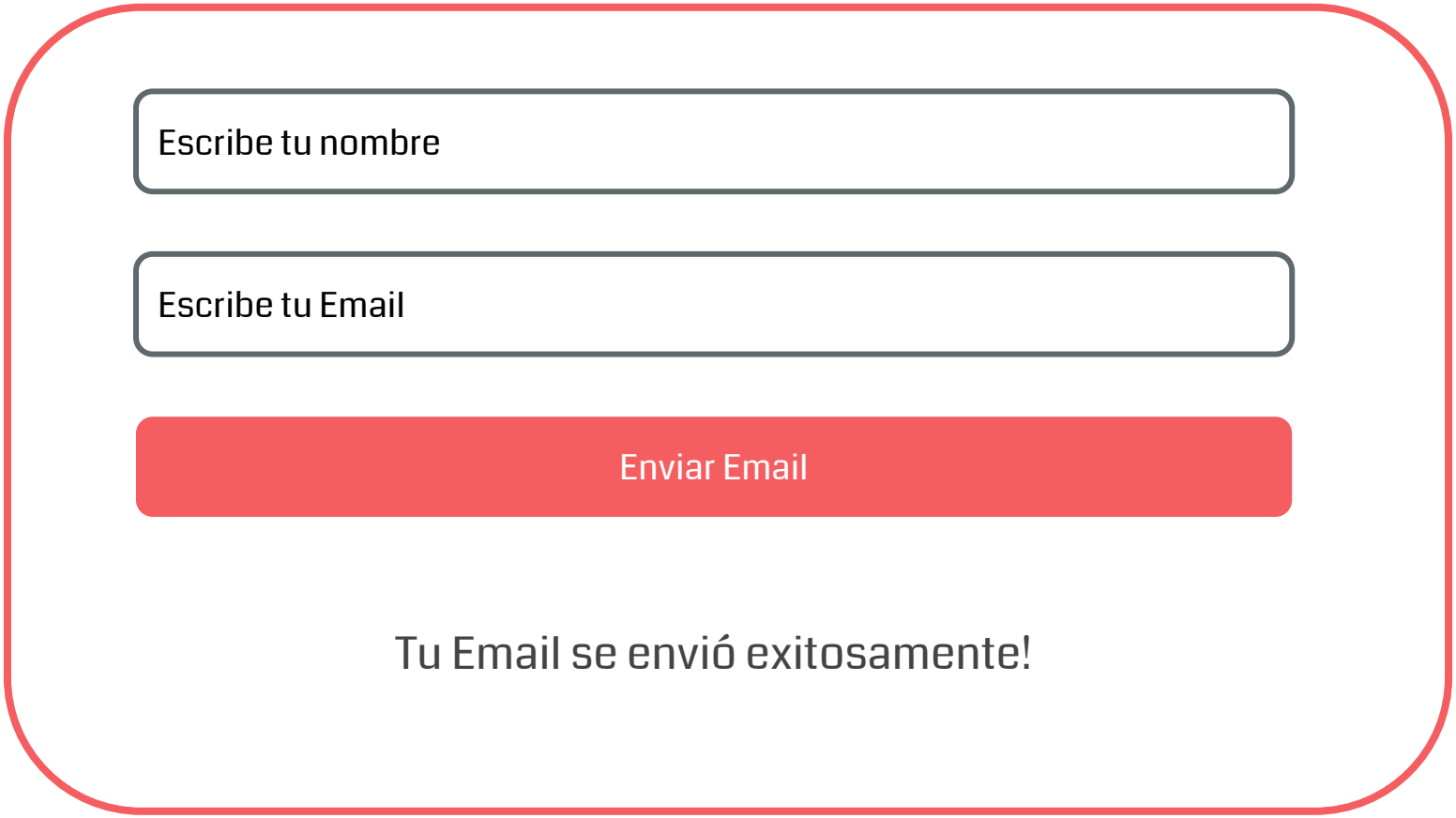
> anahi@platzi.com

> Enviando Email...

> Tu Email se envió exitosamente!

Platzi

# Framework - Programación por Eventos




Platzi

# Inversión del Flujo

# El Framework tom el control

Platzi

# Un Framework usa Inversión de Control

Platzi

# Spring

Platzi

# Spring Framework

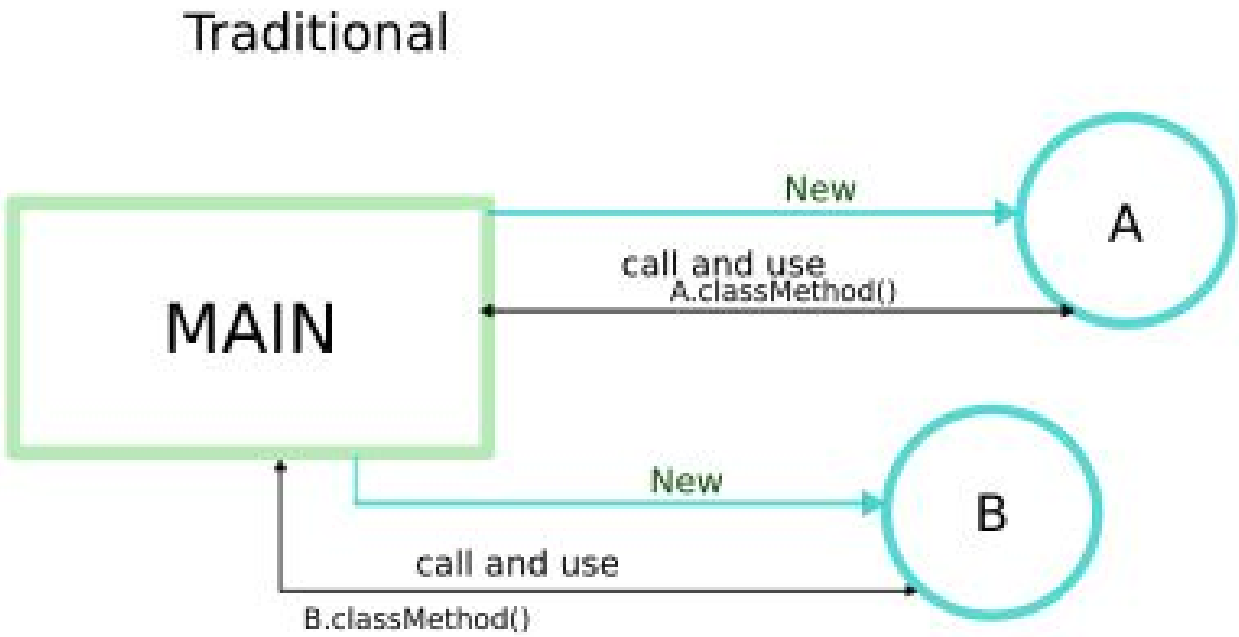
Platzi

# Spring

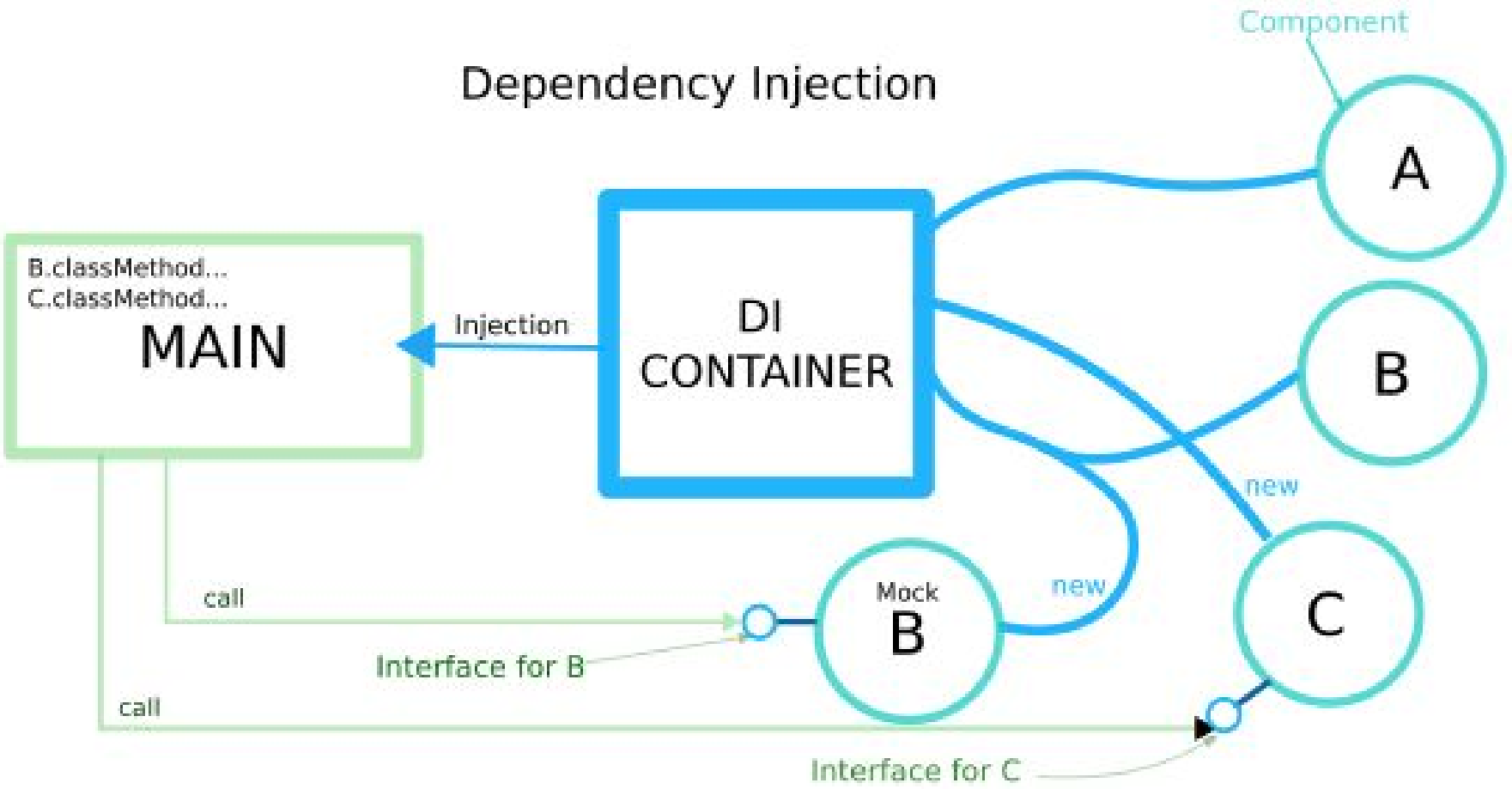
## Implementa un Contenedor que se encarga de gestionar la creación y destrucción de los objetos (instancias)

Platzi

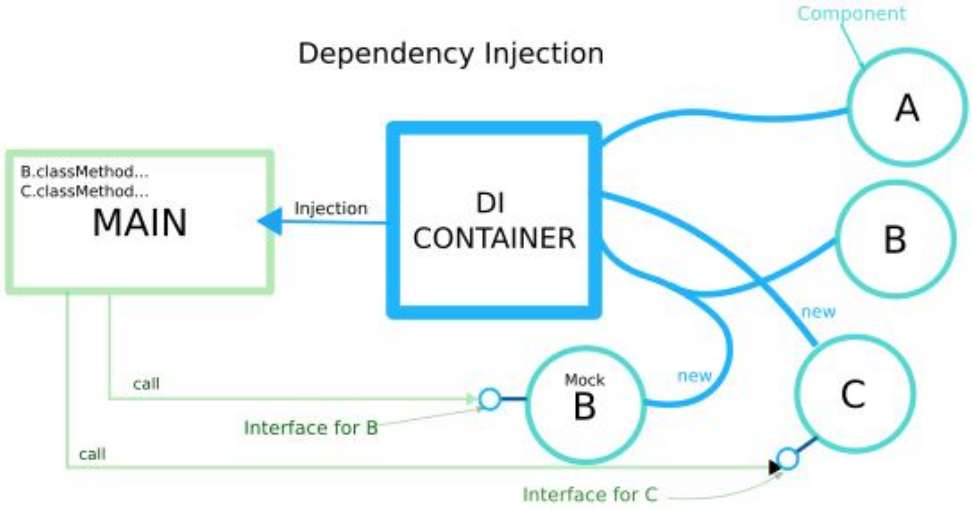
# Spring Framework

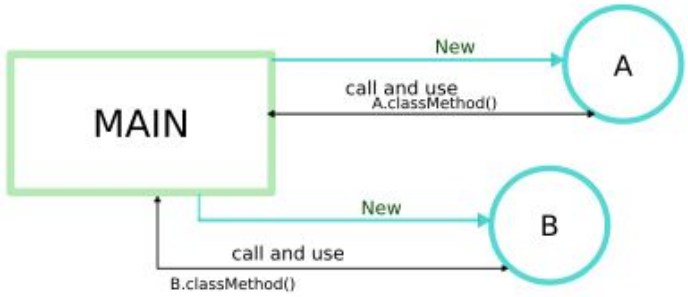


# Spring Framework



# Spring Framework

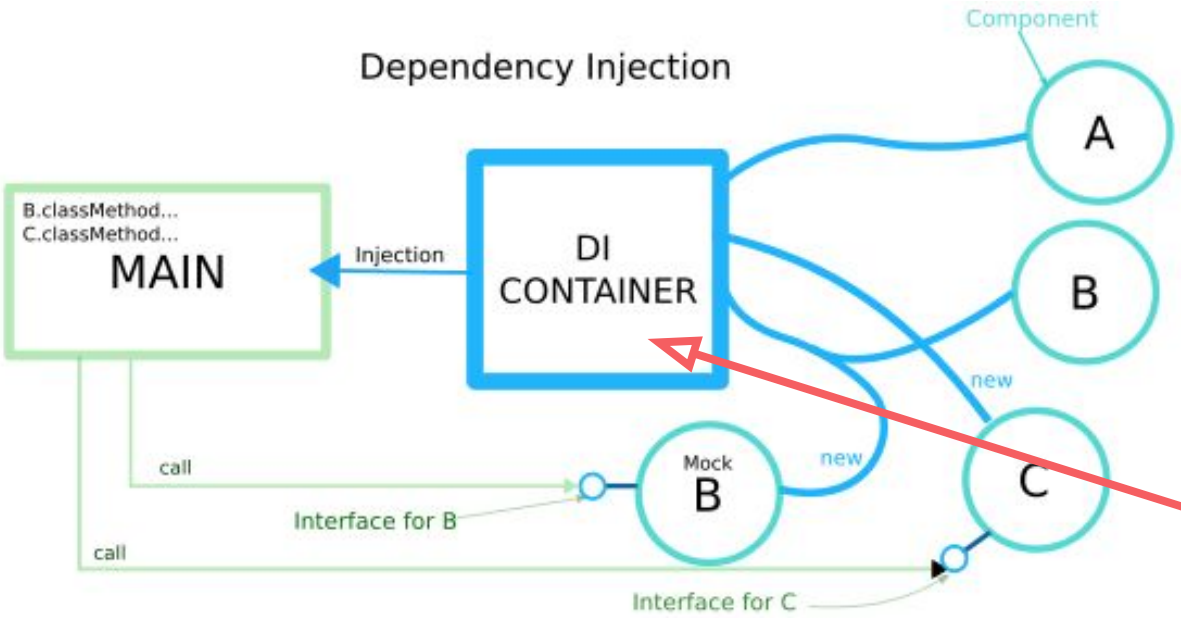



https://itblogsogeti.com/2015/10/29/inyeccion-de-dependencias-vs-inversion-de-control-eduard-moret-sogeti/

Platzi

# Spring Framework




Inversión de Control

Platzi

# Inyección de Dependencias

Platzi

# Inyección de Dependencias

## Hacer que una clase A inyecte objetos en una clase B

Platzi

# Spring - Inyección de Dependencias

La clase A le diga qué objetos crear a la clase B

Evitar que la clase B decida qué objetos crear

# Spring - Inyección de Dependencias

Los módulos de alto nivel del software no deben depender de los módulos de bajo nivel

# Spring - Inyección de Dependencias

```
public class Vehiculo
{
    private Motor m;

    public Vehiculo(Motor motorVehiculo)
    {
        // El módulo superior ya no instancia directamente el objeto Motc
        // sino que éste es pasado como parámetro en el constructor
        m = motorVehiculo;
    }

    public int GetRevolucionesMotor()
    {
        return m.GetRevoluciones();
    }
}
```

La clase “Vehiculo” quiere obtener las revoluciones del motor

Platzi

# Spring - Inyección de Dependencias

```
public class Vehiculo
{
    private Motor m;

    public Vehiculo(Motor motorVehiculo)
    {
        // El módulo superior ya no instancia directamente el objeto Motc
        // sino que éste es pasado como parámetro en el constructor
        m = motorVehiculo;
    }

    public int GetRevolucionesMotor()
    {
        return m.GetRevoluciones();
    }
}
```

El módulo superior -vehículo- depende del módulo inferior -motor-

# Spring - Inyección de Dependencias

Desacoplaremos los objetos hallando la forma más genérica de hacerlo

**Polimorfismo**

# Spring - Inyección de Dependencias

La forma ideal será usar **interfaces** pues conseguimos abstraer la relación de que una clase A depende de una clase B sin importar la implementación

# Spring - Inyección de Dependencias

```
public interface IMotor
{
    // Métodos comunes a todos los motores
    void Acelerar();
    int GetRevoluciones();
}
```

# Spring - Inyección de Dependencias

```
public class MotorGasolina : IMotor
{
    public void Acelerar()
    {
        RealizarAdmision();
        RealizarCompresion();
        RealizarExplosion();     // Propi
        RealizarEscape();
    }

    public int GetRevoluciones()
    {
        int currentRPM = 0;

        // ...

        return currentRPM;
    }
```

```
public class MotorDiesel : IMotor
{
    public void Acelerar()
    {
        RealizarAdmision();
        RealizarCompresion();
        RealizarCombustion();    // Propia
        RealizarEscape();
    }

    public int GetRevoluciones()
    {
        int currentRPM = 0;

        // ...

        return currentRPM;
    }
```

Platzi

# Spring - Inyección de Dependencias

```
public class Vehiculo
{
    private IMotor m;

    public Vehiculo(IMotor motorVehiculo)
    {
        // El módulo superior ya no instancia directamente el objeto Motc
        // sino que éste es pasado como parámetro en el constructor
        m = motorVehiculo;
    }

    public int GetRevolucionesMotor()
    {
        return m.GetRevoluciones();
    }
}
```

El módulo superior -vehículo- ya no depende del módulo inferior -motor-

# Spring - Inyección de Dependencias

```
public class Vehiculo
{
    private IMotor m;

    public Vehiculo(IMotor motorVehiculo)
    {
        // El módulo superior ya no instancia directamente el objeto Motc
        // sino que éste es pasado como parámetro en el constructor
        m = motorVehiculo;
    }

    public int GetRevolucionesMotor()
    {
        return m.GetRevoluciones();
    }
}
```

https://danielggarcia.wordpress.com/2014/01/15/inversion-de-control-e-inyeccion-de-dependencias/

Definimos una instancia para acceder a los métodos

Platzi

# Spring - Inyección de Dependencias

El contenedor de Inyección de Dependencias se encarga de  hacer la instanciación de los objetos

# Spring - Inyección de Dependencias

En este contenedor se suelen crear y almacenar **objetos de servicio, DAO's**, y **objetos** que nos **permitan conectarnos** con otras partes del sistema como **Bases de Datos**, Sistemas

# Spring
# Bean Factory

Platzi

# Spring - Bean Factory

**Los objetos que son la columna vertebral de tu aplicación** y que son administrados por el contenedor Spring se denominan **beans**.

# Spring - Bean Factory

Objetos de servicio y DAO’s y Objetos que nos permitan conectarnos con otras partes del sistema como Bases de Datos

# Spring - Bean Factory

- **Implementamos la interfaz en un solo tipo de Objeto**

- **DAO**

Los indicamos con la anotación **`@Autowired`**

# Spring - Bean Factory

```
public class GeneradorPlaylist {

    private BuscadorCanciones buscadorCanciones;

    public GeneradorPlaylist(BuscadorCanciones buscadorCanciones){
        this.buscadorCanciones = buscadorCanciones;
    }

    //Resto de métodos de la clase

}
```

https://www.adictosaltrabajo.com/tutoriales/spring-container-inyeccion-dependencias/

# Spring - Bean Factory

```
public class GeneradorPlaylist {

    @autowired
    private BuscadorCanciones buscadorCanciones;

    public setBuscadorCanciones(BuscadorCanciones buscadorCanciones){
        this.buscadorCanciones = buscadorCanciones;
    }

    //Resto de métodos de la clase
}
```

https://www.adictosaltrabajo.com/tutoriales/spring-container-inyeccion-dependencias/

Platzi

# Spring - Bean Factory

- Implementamos la interfaz en varios tipos de Objetos
- Tendríamos que decirle a Spring cuál es el Bean correcto, a partir de un archivo de configuración de Beans

Los indicamos con la anotación

`@Configuration` y `@Bean`

# Spring - Bean Factory

```
@Configuration
public class AppConfig {
    @Bean
    public TransferService transferService() {
        return new TransferServiceImpl();
    }
}
```

http://docs.spring.io/spring-javaconfig/docs/1.0.0.M4/reference/html/ch02s02.html

Platzi

# Spring - Bean Factory

Objetos de Sesión que trabajen en modo
**Singleton**

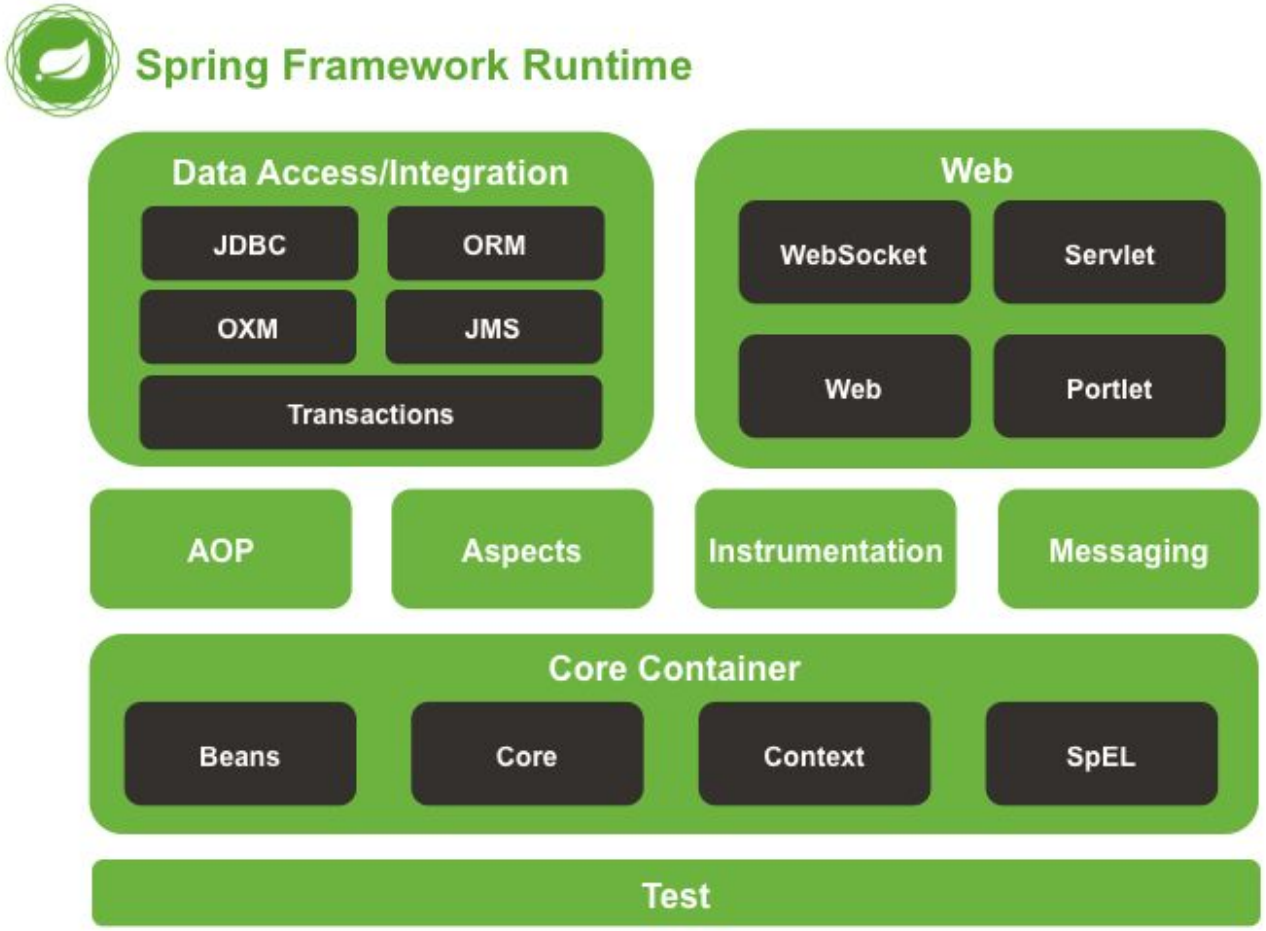
# Composición de Spring

Platzi

# Spring

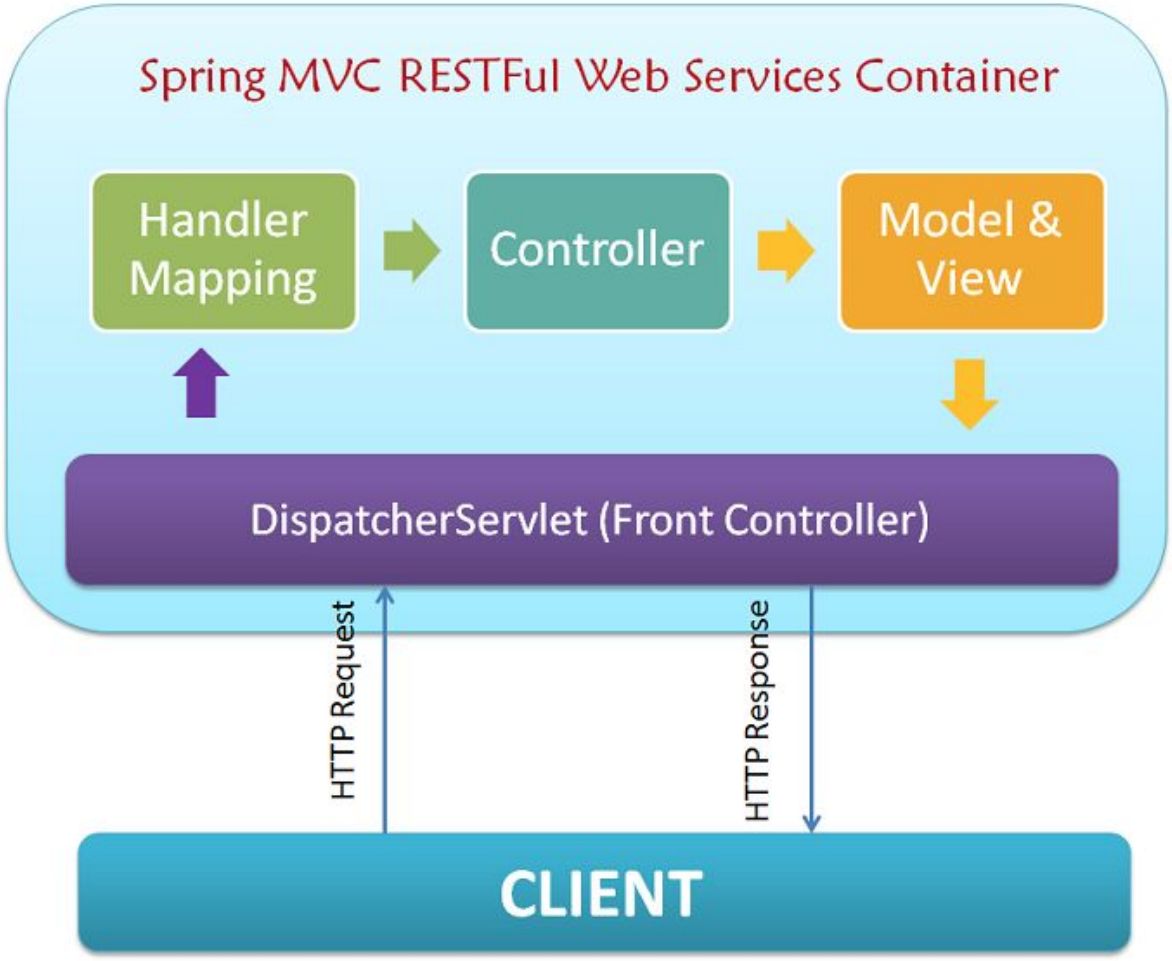

Es framework muy grande que contiene muchos componentes

# Spring Framework - Composición




Platzi

# Spring MVC - Composición




Platzi

# Spring MVC - Composición

```
<html>
<body>
    <h1>Spring 3.2.3 MVC web service</h1>

    <h3>Your message is : ${msg}</h3>
</body>
</html>
```


Platzi

# Spring MVC - Composición


```
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>

   <bean
    class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

   <bean name="/helloWorld.htm"
        class="com.javacodegeeks.snippets.enterprise.HelloWorldController" />

   <bean name="/hello*.htm"
        class="com.javacodegeeks.snippets.enterprise.HelloWorldController" />

</beans>
```

Platzi

# Spring MVC - Composición


```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
<display-name>Archetype Created Web Application</display-name>

    <servlet>
        <servlet-name>mvc-dispatcher</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>mvc-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Platzi

# Spring MVC - Composición


```java
public class HelloWorldController extends AbstractController{

    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest request,
            HttpServletResponse response) throws Exception {

        ModelAndView model = new ModelAndView("helloWorld");
        model.addObject("msg", "hello world!");

        return model;
    }

    }
```

Platzi

# Spring MVC - Composición

```
<html>
<body>
    <h1>Spring 3.2.3 MVC web service</h1>

    <h3>Your message is : ${msg}</h3>
</body>
</html>
```


Platzi

# Spring Boot

## Producto de Spring

Platzi

# Spring Boot

- Simplifica la creación de aplicaciones de aplicaciones y servicios Spring

# Spring Boot

- Proveer una forma muy sencilla de arrancar desarrollos Spring

- Proporcionar una serie de características no funcionales comunes a los proyectos (por ejemplo, servidores embebidos, seguridad, indicadores, configuración externalizada)

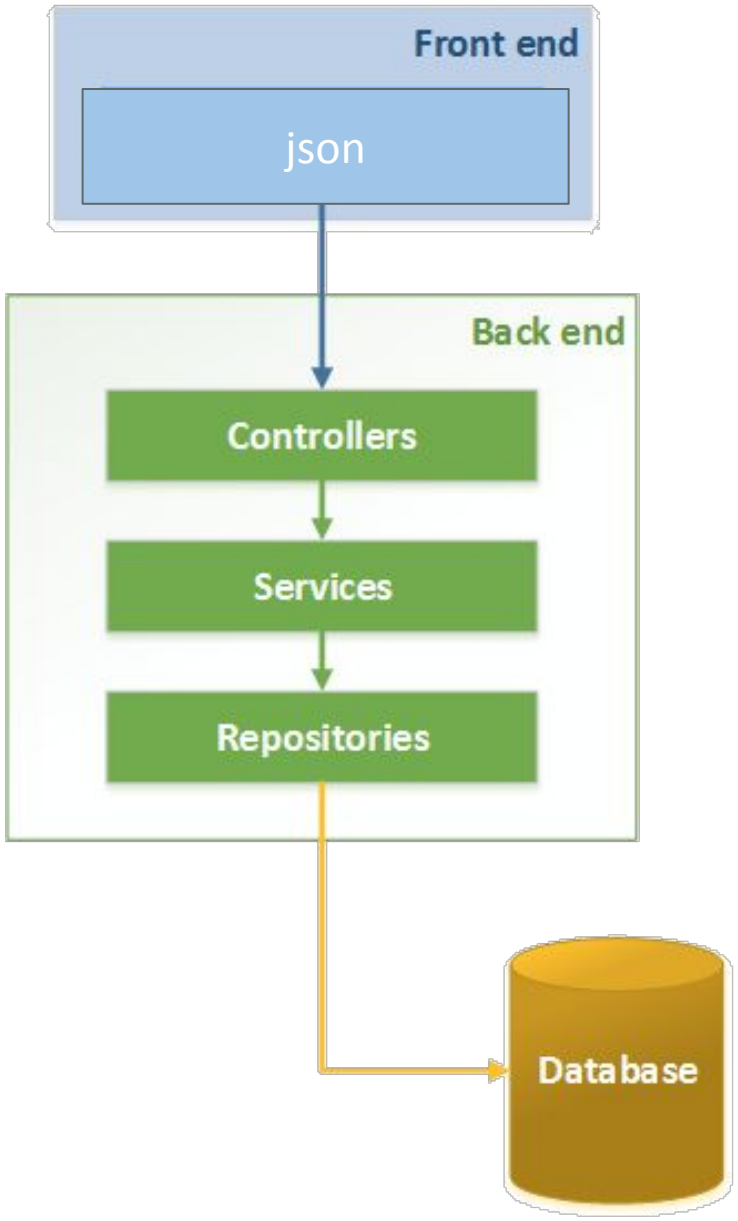
# Spring Boot

- No necesitar generación código ni configuración XML

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

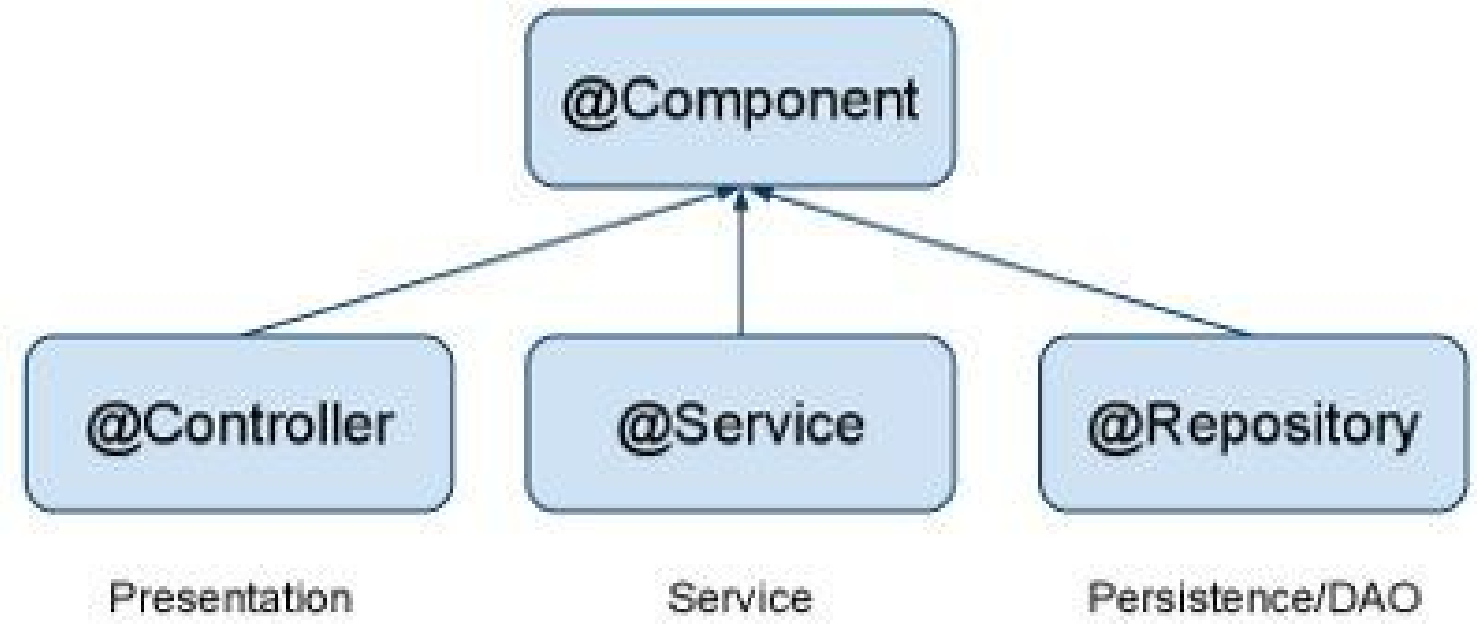
Platzi

# Spring Boot




Platzi

# Spring Boot


@Component
@Controller
@Service
@Repository
Presentation
Service
Persistence/DAO

# Spring Boot - @Component

Componente genérico

# Spring Boot - @Controller

La capa de presentación, aquí se llega cuando solicitas una ruta, también utiliza @RequestMapping

# Spring Boot - @Service

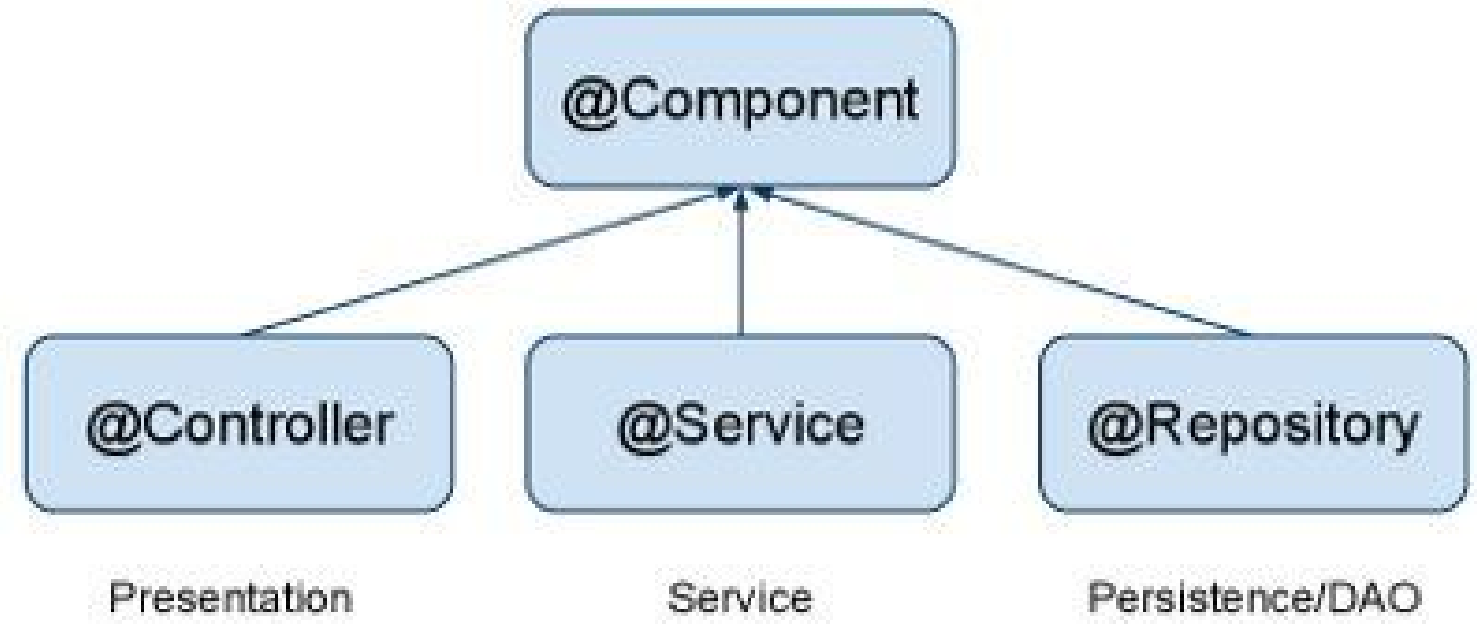
Maneja la lógica del negocio, llama a los DAO's y ejecuta cálculos relacionados
Generalmente se conecta con
@Repository

# Spring Boot - @Repository

Es la capa de persistencia de la aplicación que se utiliza para obtener datos de la base de datos.

Es decir, todas las operaciones relacionadas con la base de datos son realizadas por este repositorio.

# Spring Boot


@Component
@Controller
@Service
@Repository
Presentation
Service
Persistence/DAO

# API Rest

Platzi

# REpresentational State Transfer

# Como crear apis

https://platzi.com/blog/como-crear-apis/

# Deploy del Proyecto

Platzi


# Heroku

# Heroku


https://devcenter.heroku.com/

# Heroku


https://devcenter.heroku.com/articles/getting-started-with-java#introduction