Angular - Guía de supervivencia

Programación de Servicios y procesos - 2º DAM

Luis del Moral Martínez

versión 21.01

Contenidos del tema

1. **¿Qué es Angular?**
   - 1.1 Introducción a Angular
   - 1.2 Historia de Angular
   - 1.3 Introducción a las aplicaciones SPA
2. **Preparando nuestro entorno de desarrollo**
   - 2.1 Instalación del software necesario
   - 2.2 Comandos básicos de Angular CLI
   - 2.3 Testear la configuración
   - 2.4 Ejecutar los ejemplos
3. **Introducción a TypeScript**
   - 3.1 ¿Qué es TypeScript?
   - 3.2 Algunos conceptos básicos
4. **Patrones de diseño útiles**
   - 4.1 MVC (modelo-vista-controlador)
   - 4.2 DI (inyección de dependencias)
5. **Un poquito de estructura, por favor**
   - 5.1 Elementos de una aplicación Angular

Contenidos de la sección

1. **¿Qué es Angular?**
   - 1.1 Introducción a Angular
   - 1.2 Historia de Angular
   - 1.3 Introducción a las aplicaciones SPA

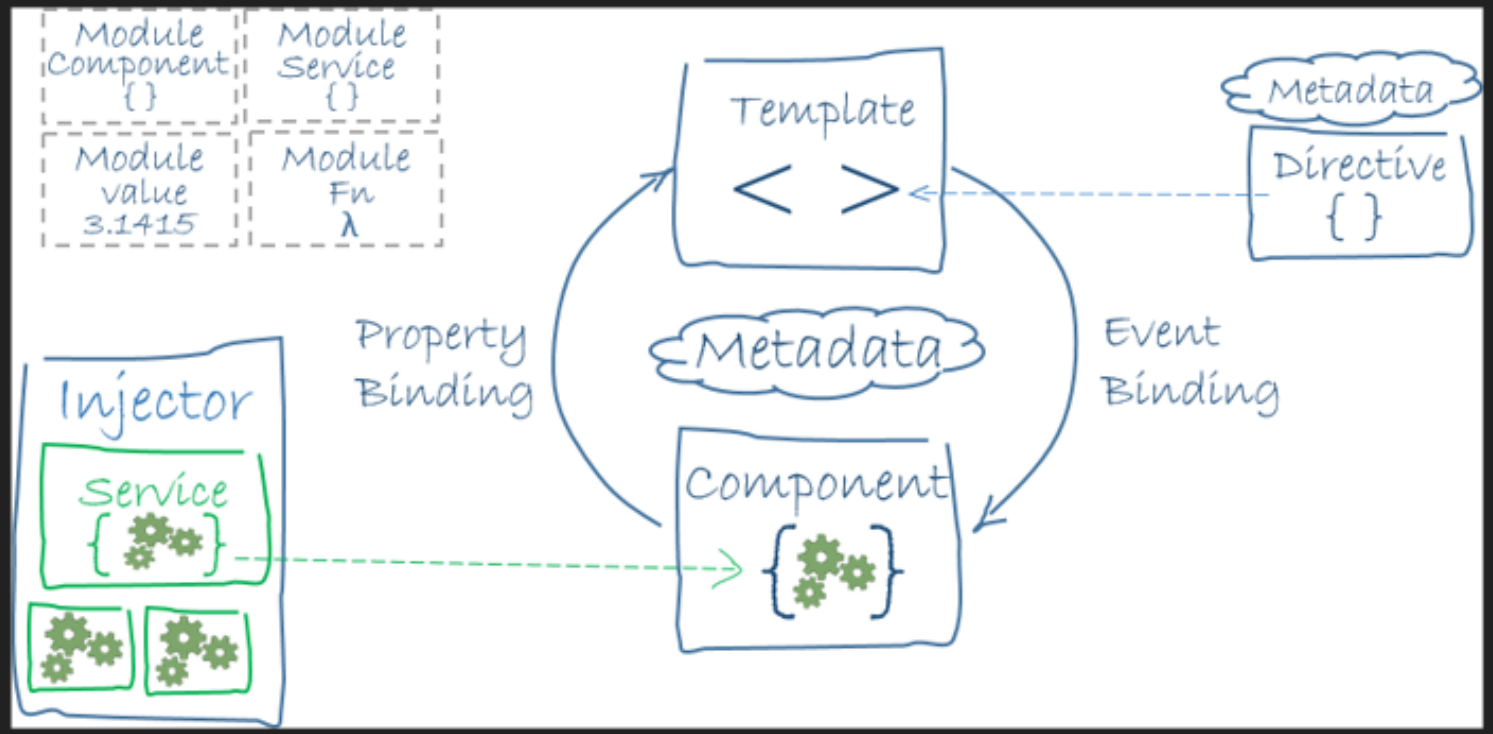
1.1 Introducción a Angular

Angular 9

- Es una **plataforma** (framework) para desarrollar aplicaciones web usando **HTML** y **JS**
- Está mantenido por Google, está **orientado a objetos** y favorece el uso del patrón **MVC**
- Utiliza **TypeScript**, desarrollado por **Microsoft** (**más información**)
- La versión más actual de **Angular** es la 9 (febrero de 2020) y utiliza **TypeScript** (versión 3.7)
- Mas información sobre el patrón **MVC** (Modelo-Vista-Controlador) en este **enlace**
- Más información sobre Angular en este **enlace**

1.1 Introducción a Angular

Diagrama de arquitectura de Angular



Fuente: [Angular.io]

1.2 Historia de Angular

Evolución de Angular

- En 2009 nace el proyecto **Angular JS** (Miško Hevery y Adams Abron)
- **Angular JS** era, inicialmente, un servicio online de almacenamiento de objetos **JSON**
- Google aún da soporte a Angular JS (última versión estable: 1.7.6)
- Poco después se abandonó el proyecto y se libera Angular JS como código abierto
- En la versión 2.0 se rediseña el framework y se introduce **TypeScript**, pasando a llamarse **Angular**
- Más información sobre best-practices en Angular en este **enlace**

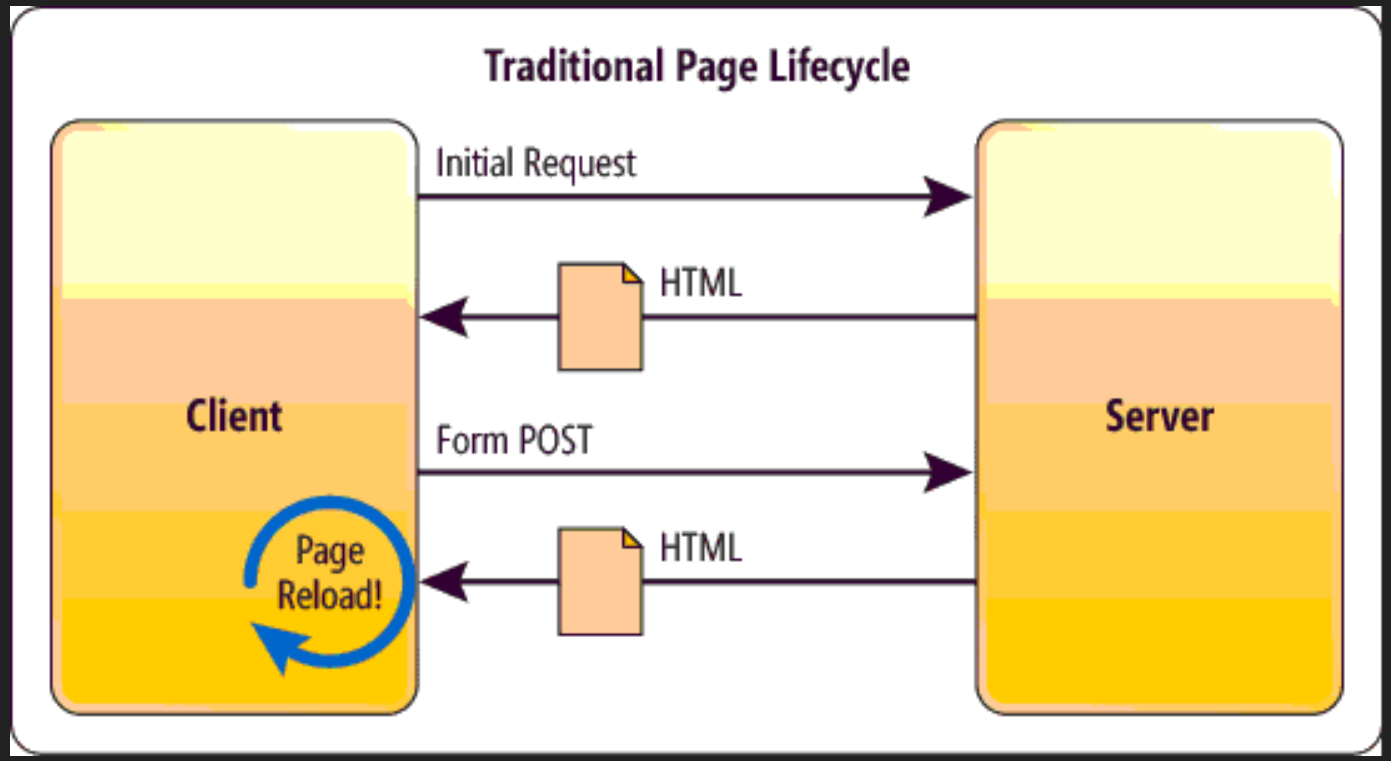
1.3 Introducción a las aplicaciones SPA

SPA (Single-Page Application)

- Toda aplicación web tiene una estructura **cliente-servidor**
- Las aplicaciones web actuales tienen una alta carga de trabajo en la parte cliente
- La parte cliente es **dinámica** y pretende disminuir las **comunicaciones** con la parte servidor
- La estructura SPA es un **patrón** enfocado a desarrollar una aplicación en **una única página**
- El patrón SPA lleva al extremo el uso de **AJAX** (Javascript asíncrono)
- Angular es un framework enfocado al desarrollo de aplicaciones SPA
- Aprende más sobre páginas SPA en este **enlace**

1.3 Introducción a las aplicaciones SPA

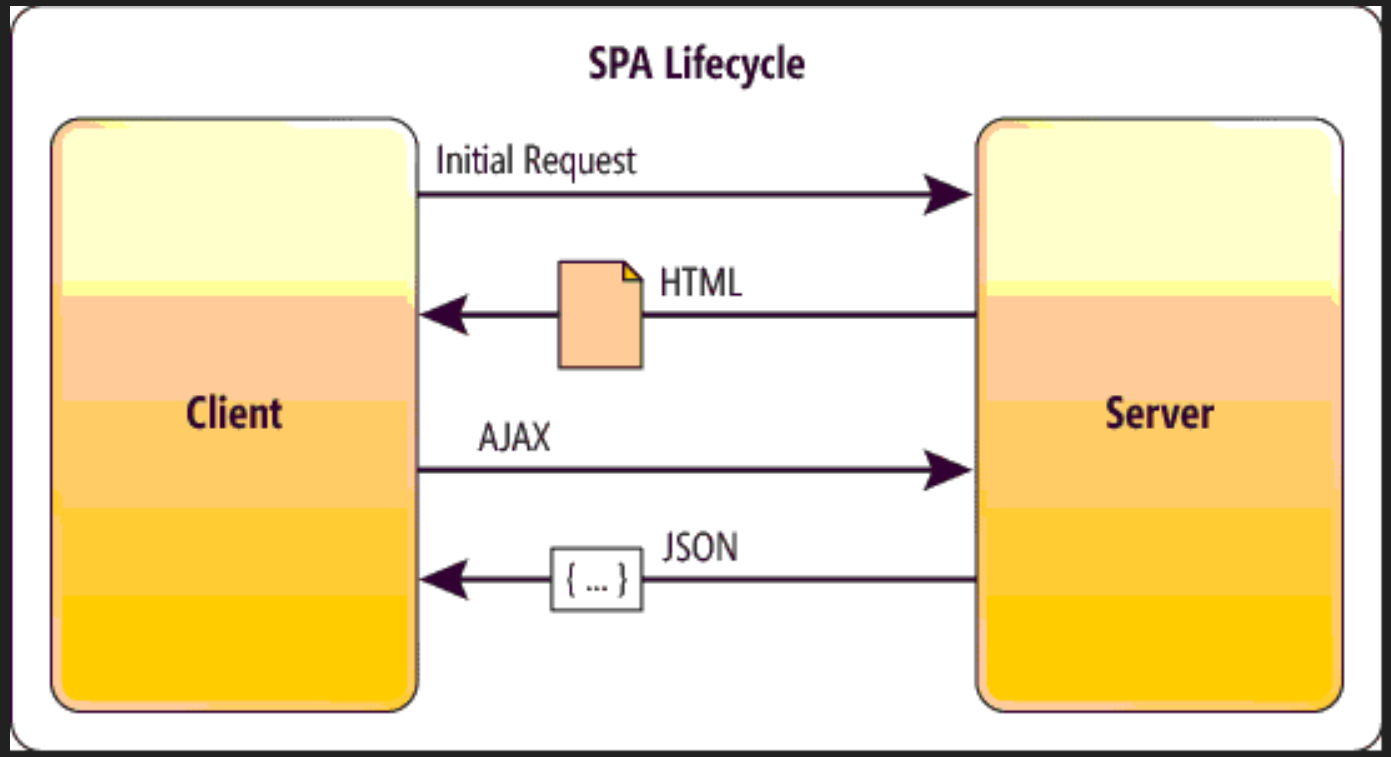
SPA (Single-Page Application)



Fuente: Microsoft

1.3 Introducción a las aplicaciones SPA

SPA (Single-Page Application)



Fuente: Microsoft

Contenidos de la sección

2. **Preparando nuestro entorno de desarrollo**
   - 2.1 Instalación del software necesario
   - 2.2 Comandos básicos de Angular CLI
   - 2.3 Testear la configuración
   - 2.4 Ejecutar los ejemplos

2.1 Instalación del software necesario

Pasos para instalar Angular en nuestro sistema

1. Descargar e instalar **Visual Studio Code**: enlace de descarga
2. Descargar e instalar **GIT**: enlace de descarga
   - Después de la descarga, podremos usar Git Bash
3. Descargar e instalar **Node LTS**: enlace de descarga
   - Node incluye **NPM** (sistema de gestión de paquetes de node.js)
4. Instalar la **CLI** (Command-Line Interface de Angular):
   - **npm install -g @angular/cli**

2.2 Comandos básicos de Angular CLI

Guía básica de comandos de Angular CLI

- La CLI permite generar nuestra aplicación, así como generar código
- Comandos básicos
  - **ng serve -o** (inicia el proyecto y lo abre en el navegador)
  - **ng g component nombre** (genera un componente denominado 'nombre')
  - **ng g interface nombre** (genera una interfaz denominada 'nombre')
  - **ng g service nombre** (genera un servicio denominado 'nombre')
- Aprende más sobre los comandos de **Angular CLI** en este **enlace**

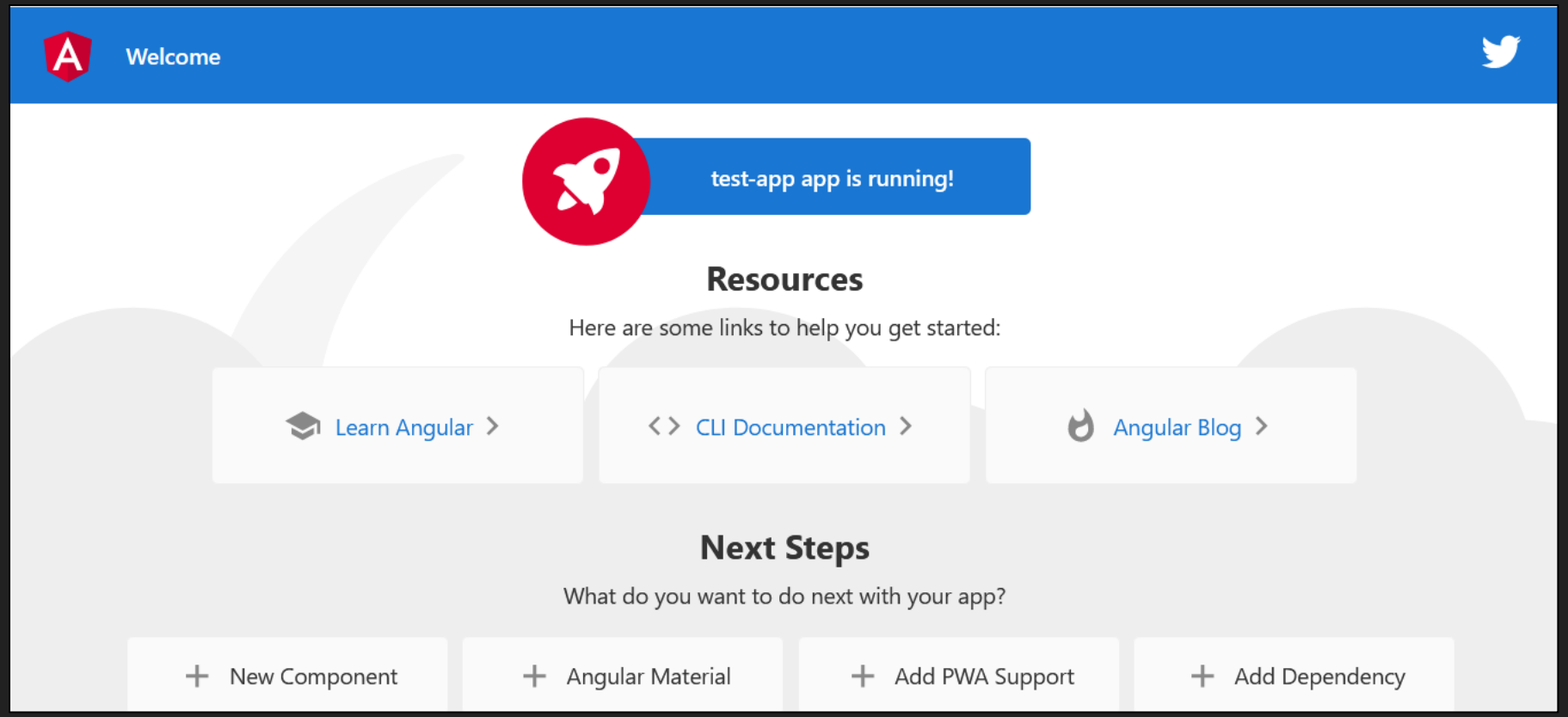
2.3 Testear la configuración

Pasos para crear un proyecto de prueba

1. Crear una carpeta para alojar nuestros proyectos (Workspace)
2. Abrir la carpeta en **VS Code** con la opción del menú contextual **Open as VS Code Project**
3. Abrir un terminal en **VS Code**
4. Ejecutar el comando **ng new nombre-proyecto**
   - No es necesario incluir **rutas** (si no vamos a definir un enrutado propio)
   - Se pueden usar las hojas de estilo **SCSS** o **CSS** (mi recomendación es empezar con CSS)
5. Se generará la **estructura** y los **ficheros básicos** del proyecto
6. Ejecutar el proyecto con el comando **ng serve -o**

2.3 Testear la configuración

Puedes acceder a la aplicación en la URL http://localhost:4200/



2.4 Ejecutar los ejemplos

¿Cómo ejecuto los ejemplos?

- Puedes descargar ejemplos de mi repositorio de **GitHub**: **enlace**
- Ubica cada ejemplo en una carpeta concreta
- Pasos para ejecutar el ejemplo:
  1. Abre la carpeta como proyecto de VS Code
  2. Instalar las dependencias necesarias con **npm init** (esto genera la carpeta **node_modules**)
  3. Ejecutar el proyecto con **ng serve -o**

Contenidos de la sección

3. **Introducción a TypeScript**
   - 3.1 ¿Qué es TypeScript?
   - 3.2 Algunos conceptos básicos

3. Introducción a TypeScript

¿Qué es TypeScript?

- Lenguaje creado por Microsoft
- Su sintaxis se basa en JavaScript (es muy similar a Java)
- Es tipado y orientado a objetos
- Se compila en JavaScript básico
- Mas información sobre TypeScript en este **enlace**
- TypeScript HandBook: **enlace**

3. Introducción a TypeScript

Algunos conceptos básicos

- Los componentes son clases importadas
- Partes de un fichero **.ts** (muy similar a la sintaxis Java)
  - **import**: para incluir componentes externos
  - **cabecera**: nombre de la clase (herencia…)
  - **variables**: se ubican al principio de la clase
  - **constructor**: primera función que se ejecuta (proporciona los servicios del componente)
  - **OnInit**: inicia el ciclo de vida de Angular (se invoca cuando se inicializa el componente en la vista)
- Analicemos ahora el código de ejemplo del proyecto de prueba que hemos creado

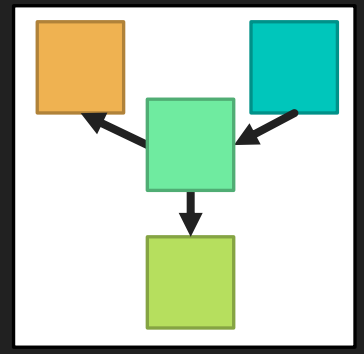
Contenidos de la sección

4. **Patrones de diseño útiles**
   - 4.1 MVC (modelo-vista-controlador)
   - 4.2 DI (inyección de dependencias)

4.1 MVC (modelo-vista-controlador)

Definición del patrón

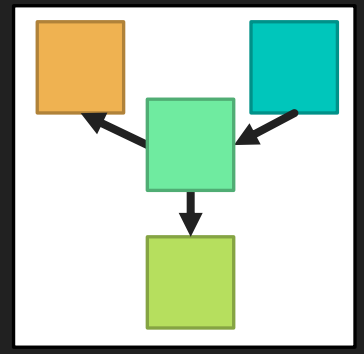
- Es un patrón de **diseño estructural**
- Permite **separar** la **lógica de negocio** del **modelo** de datos y la **vista**
- El cliente utiliza el **controlador** y éste **modifica** el **modelo**
- El controlador contiene la **lógica de negocio** y utiliza los **datos**
- La **vista** se encarga de **visualizar** la información



4.1 MVC (modelo-vista-controlador)

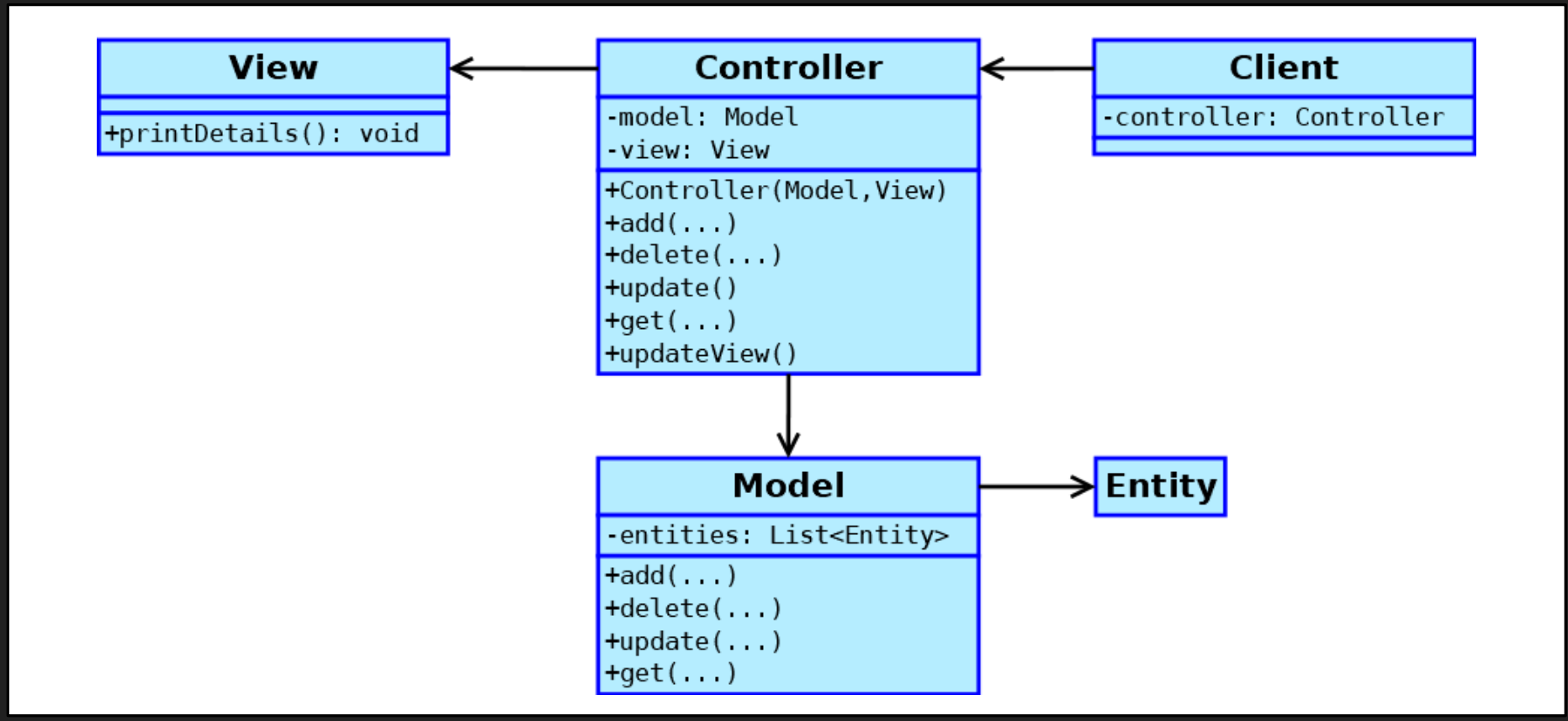
Problemática

- Tenemos clases **fuertemente acopladas** y queremos separarlas
- Se quiere **separar** la **lógica de negocio** de los **datos** y la **vista**



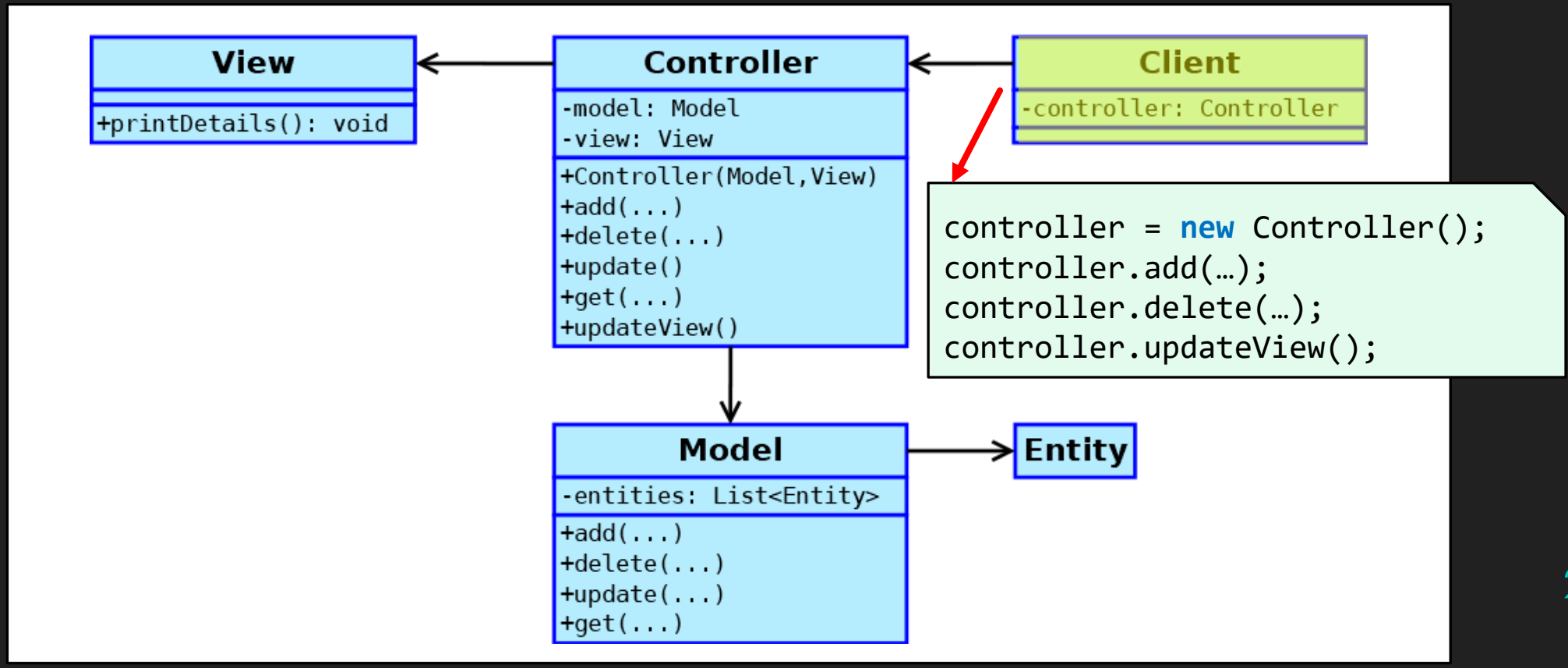
4.1 MVC (modelo-vista-controlador)

Estructura del patrón (1)



4.1 MVC (modelo-vista-controlador)

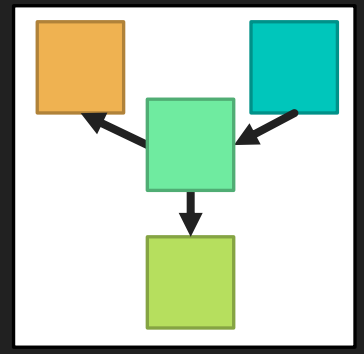
Estructura del patrón (2)



4.1 MVC (modelo-vista-controlador)

Estructura del patrón (3)

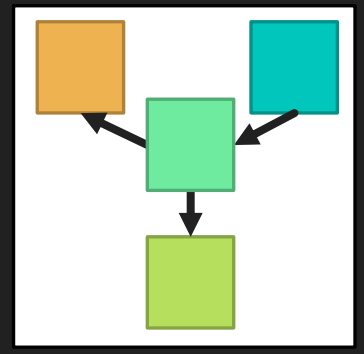
- El cliente utiliza el controlador
- El controlador **define** y **administra** el **modelo** y la **vista**
- El controlador contiene la **lógica de negocio**
- El controlador sabe cómo **gestionar** los datos
- La vista sólo sirve para **representar** la información
- El modelo de datos utiliza o referencia a entidades
- El modelo de datos puede obtener los datos de una BBDD



4.1 MVC (modelo-vista-controlador)

Aplicabilidad

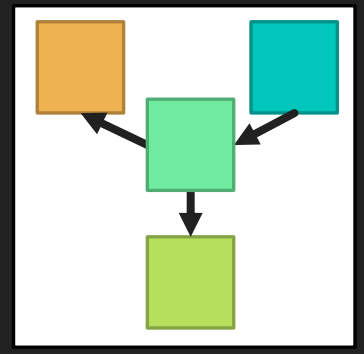
- Para **organizar** mejor el código
- Para separar la **lógica de negocio** del resto de clases
- La estructura es más **estable** y **fácil** de administrar



4.1 MVC (modelo-vista-controlador)

Ventajas

- El proceso de desarrollo es más **rápido** y se puede hacer en **paralelo**
- Un cambio en el modelo no afecta a la **arquitectura** (y viceversa)



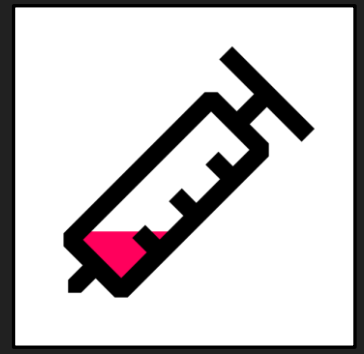
Inconvenientes

- Puede complicar un poco la estructura de la aplicación

4.2 DI (inyección de dependencias)

Definición del patrón

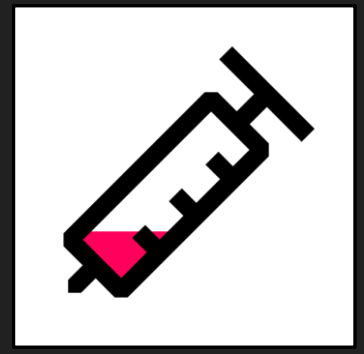
- Es un patrón de **diseño estructural**
- Permite **invertir** el **sentido de las dependencias** en nuestro modelo
- Posibilita la **extensibilidad** y **elimina el fuerte acoplamiento**
- El cliente puede **inyectar** dependencias cuando éstas son necesarias
- Al **inyectar dependencias**, se **modifica** el comportamiento



4.2 DI (inyección de dependencias)

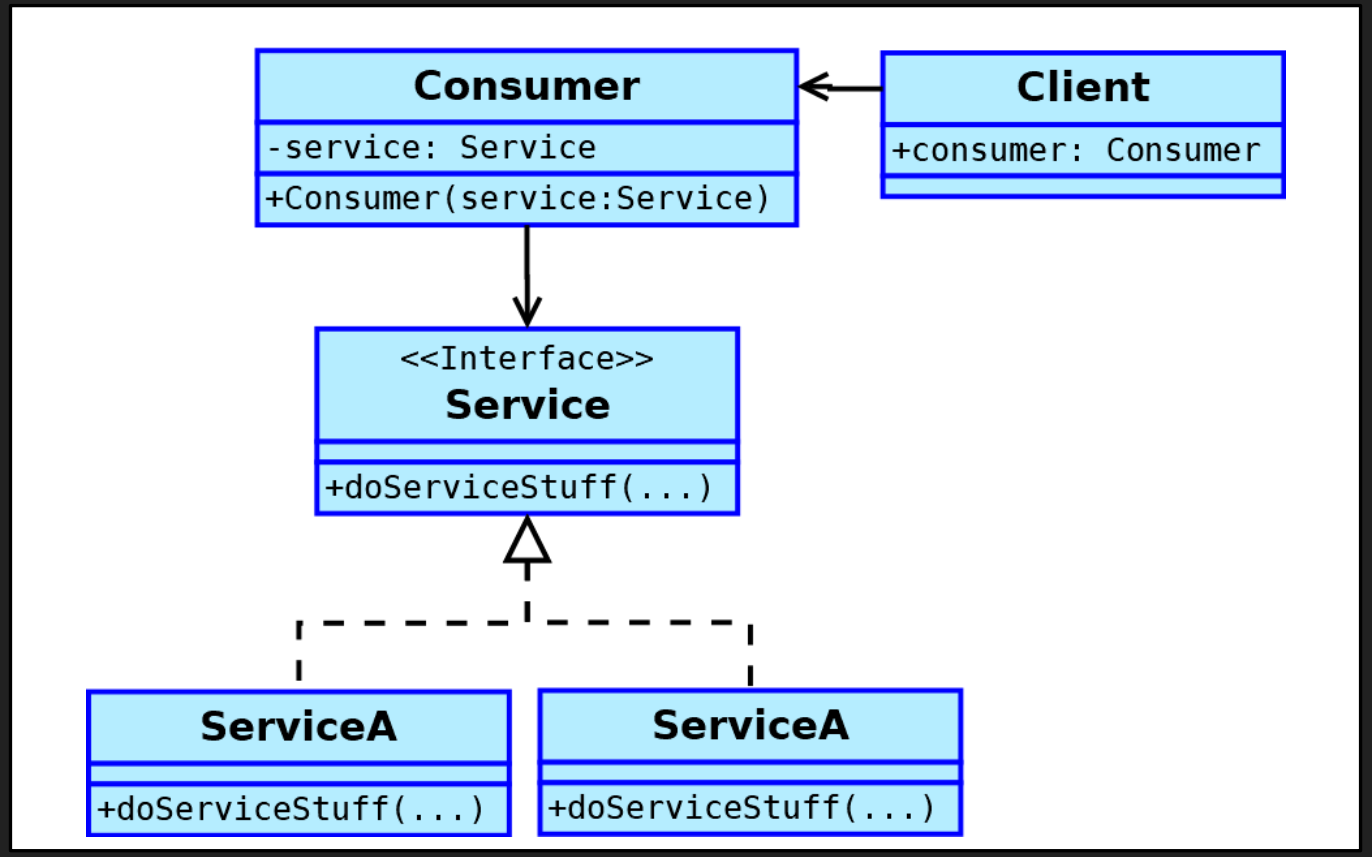
Problemática

- Los objetos se **relacionan** mediante **dependencias**
- El objeto **dependiente** se **acopla fuertemente** al otro objeto
- El código es **poco extensible**
- Cualquier modificación implica **descomponer** el código



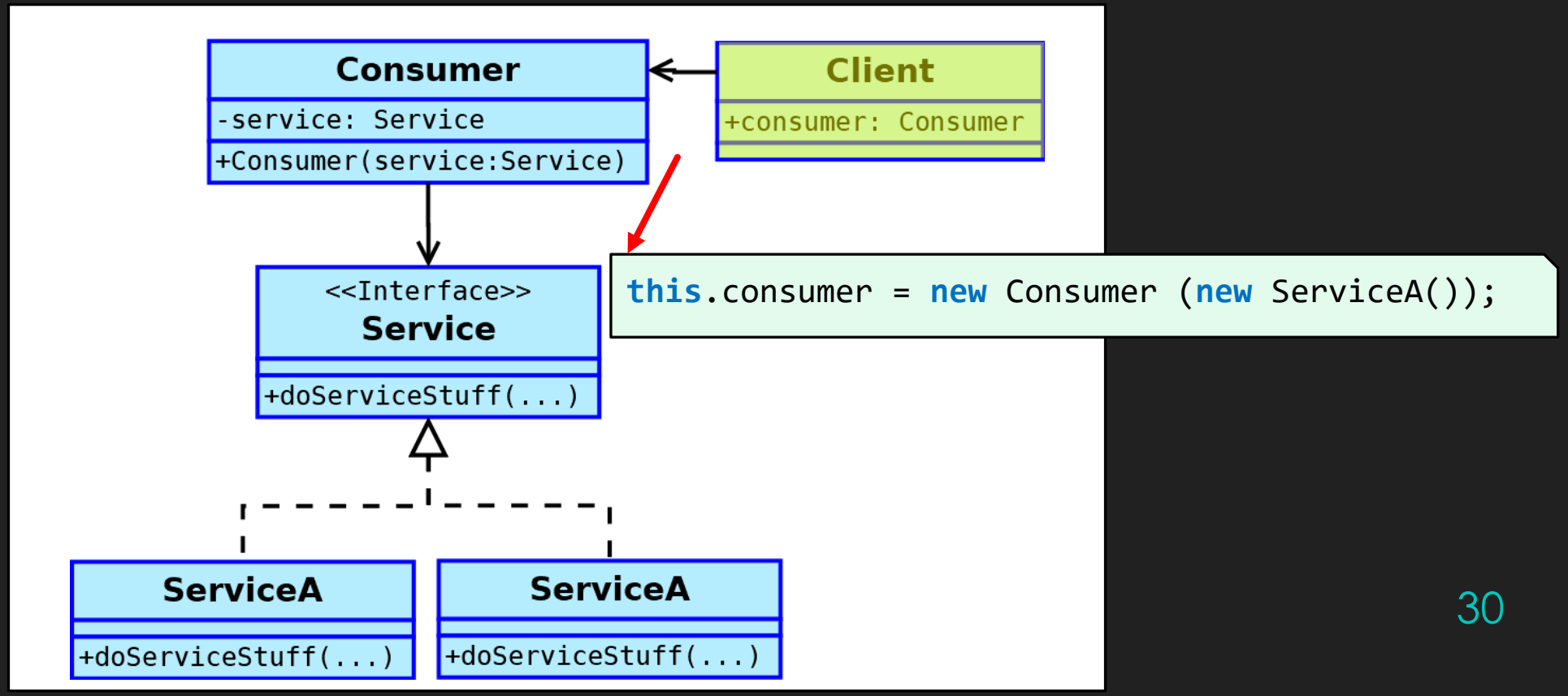
4.2 DI (inyección de dependencias)

Estructura del patrón (1)



4.2 DI (inyección de dependencias)

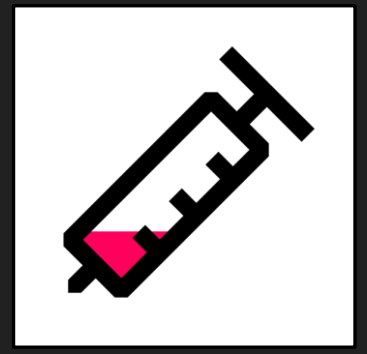
Estructura del patrón (2)



4.2 DI (inyección de dependencias)

Estructura del patrón (3)

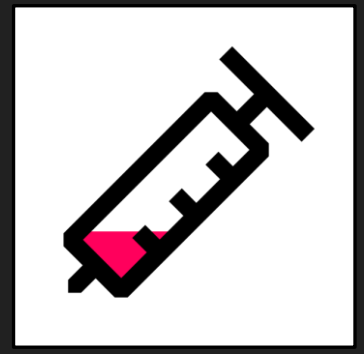
- El cliente utiliza el **Consumer** (componente) e inyecta el **servicio**
- El cliente puede **inyectar** cualquier servicio que implemente Service
- Esta estructura **invierte las dependencias** y elimina el **acoplamiento**
- El modelo evita la excepción **NullPointerException**
- Se puede cambiar el **comportamiento** de un objeto dinámicamente



4.2 DI (inyección de dependencias)

Aplicabilidad

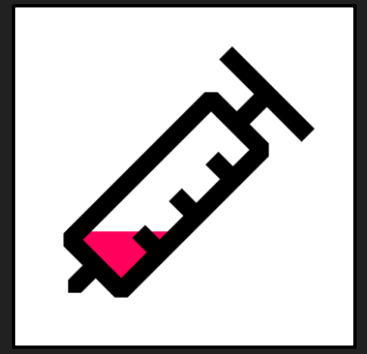
- Este patrón se utiliza en los modelos **IoC** (Inversion of Control)
- Permite eliminar el **fuerte acoplamiento** entre las clases
- Permite **cambiar el comportamiento** de un objeto dinámicamente
- Los frameworks **Spring** y **Angular** utilizan el patrón de DI



4.2 DI (inyección de dependencias)

Ventajas

- Se elimina la **inicialización** de las clases desde el cliente
- Se pueden **inyectar subclases** (o clases que implementen **Service**)
- El compilador encontrará las **dependencias** necesarias al ejecutar



Inconvenientes

- El concepto puede ser un poco complejo de entender al principio

Contenidos de la sección

5. **Un poquito de estructura, por favor**
    - 5.1 Elementos de una aplicación Angular

5.1 Elementos de una aplicación Angular

¿Cuáles son los elementos básicos de una aplicación Angular?

- **Módulo**: agrupa varios componentes (funcionalidad similar al del paquete en Java)
- **Componente**: es una parte de la aplicación (plantilla + metadatos + clase)
- **Plantilla**: define la vista de un componente (plantilla de HTML)
- **Data Binding**: intercambia datos entre la plantilla y la clase del componente (vista-controlador)
- **Directiva**: añade comportamiento dinámico a HTML (***ngFor**, ***ngIf**)
- **Servicio**: clases que ponen funcionalidad a disposición de otros componentes (ej: BBDD…)
- **Inyección de dependencia**: proporciona funcionalidad sin que un componente la cree

¿Y ahora qué?

Próximos pasos

- Estudiaremos los elementos básicos más relevantes de Angular (libro y ejemplos)
- Ejecutaremos y analizaremos en profundidad cada uno de los ejemplos
- Seguiremos los tutoriales básicos de Angular.io (enlace)
- Objetivo: modificar el ejemplo de API-REST que os propongo (GitHub)

Créditos de las imágenes y figuras

Cliparts e iconos

- **Obtenidos mediante la herramienta web IconFinder** (según sus disposiciones):
  - Diapositivas 27-28, 31-33
  - Según la plataforma IconFinder, dicho material puede usarse libremente (free comercial use)
  - A fecha de edición de este material, todos los cliparts son free for comercial use (sin restricciones)

Diagramas, gráficas e imágenes

- Se han desarrollado en PowerPoint y se han incrustado en esta presentación
- Todos estos materiales se han desarrollado por el autor
- Para el resto de recursos se han especificado sus fabricantes, propietarios o enlaces
- Si no se especifica copyright con la imagen, entonces es de desarrollo propio o CC0
- El logo de Angular es propiedad de *Google, Inc*.