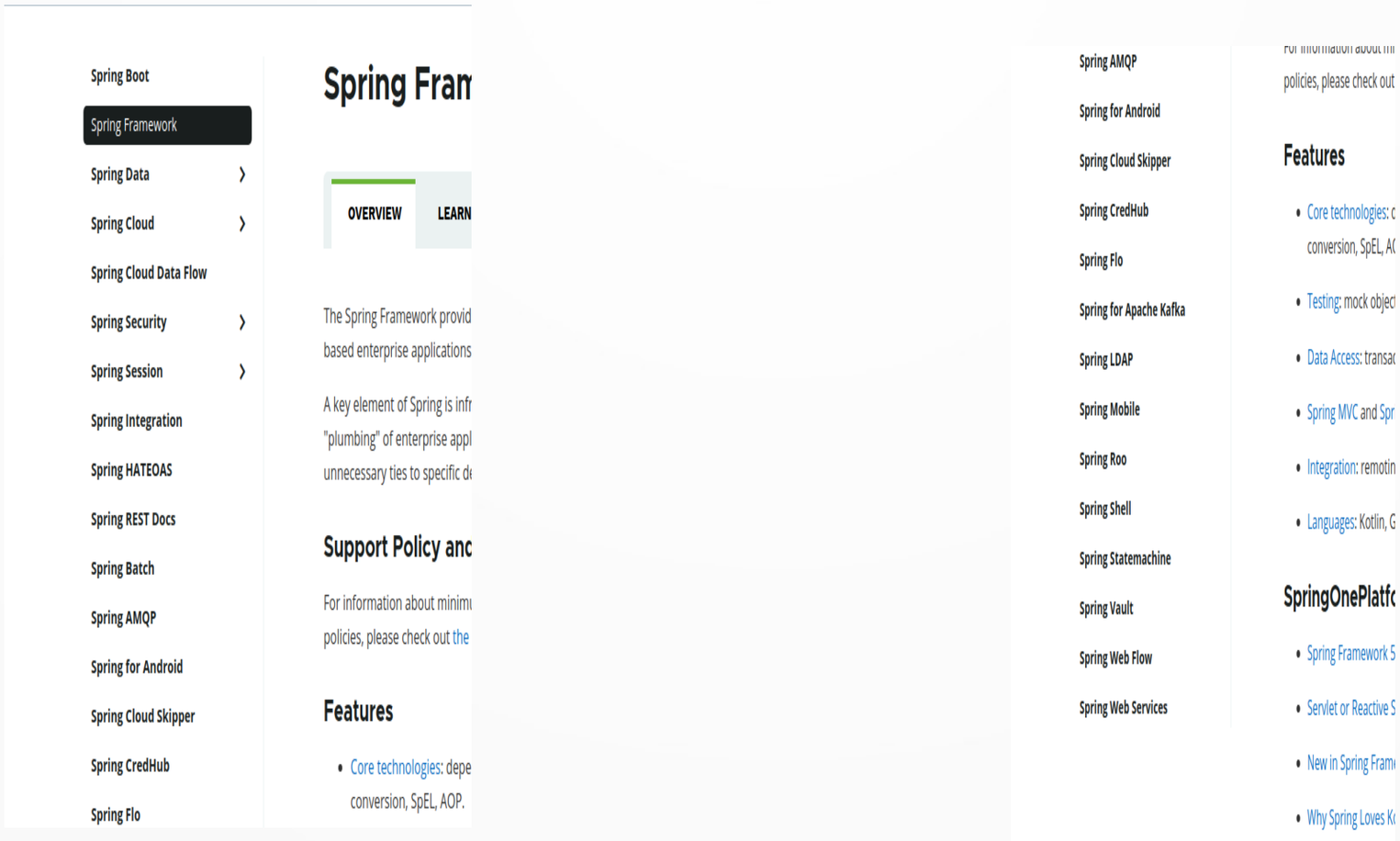
Spring Framework

Introducción a Spring

Proyecto semestral

- Desarrollar una aplicación Web personalizada y que sirva como plantilla como página oficial de los profesores de la Universidad de la Sierra Sur.

- **Parcial 1.** construcción de la arquitectura de desarrollo de la aplicación, así como el diseño de la base de datos, la configuración del repositorio de código fuente, la arquitectura del proyecto con Spring Boot y Angular
- **Parcial 2.** Codificación de la lógica de negocio mediante el framework Spring Boot.
- **Parcial 3.** Codificación de la vista mediante el framework Angular.
- **Ordinario.** Entrega del proyecto funcional y previamente probado.



https://spring.io/projects/spring-framework

1.1. Introducción a Spring

El framework Spring se compone de varios módulos, todos ellos giran entorno al

- Spring Core y más concretamente al Spring Container, el cual hace uso intensivo del patrón de inyección de Dependencias o de Inversión de Control

1.1. Introducción a Spring

- Spring es un marco de trabajo de código abierto creado por Rod Johnson.
- Spring se creó para hacer frente a la complejidad del desarrollo de aplicaciones empresariales.
- Cualquier aplicación de Java puede beneficiarse de Spring, gracias a su simplicidad, capacidad de prueba y acoplamiento débil.

1.1. Introducción a Spring

Para poder reducir la complejidad de Java, Spring utiliza cuatro estrategias.

- 1. Desarrollo ligero y muy poco invasivo, con objetos Java de clase simple antiguos (POJO).
- 2. Acoplamiento débil mediante la inyección de dependencias y la orientación de interfaz.
- 3. Programación declarativa mediante aspectos y convenciones comunes.
- 4. Reducción del código reutilizable mediante aspectos y plantillas.

1.1. Introducción a Spring

Qué son la cohesión y el acoplamiento

- Cohesión y acoplamiento son dos conceptos distintos de gran importancia en el desarrollo de código de calidad, y están íntimamente ligados con el diseño del software y con la programación orienta a objetos.

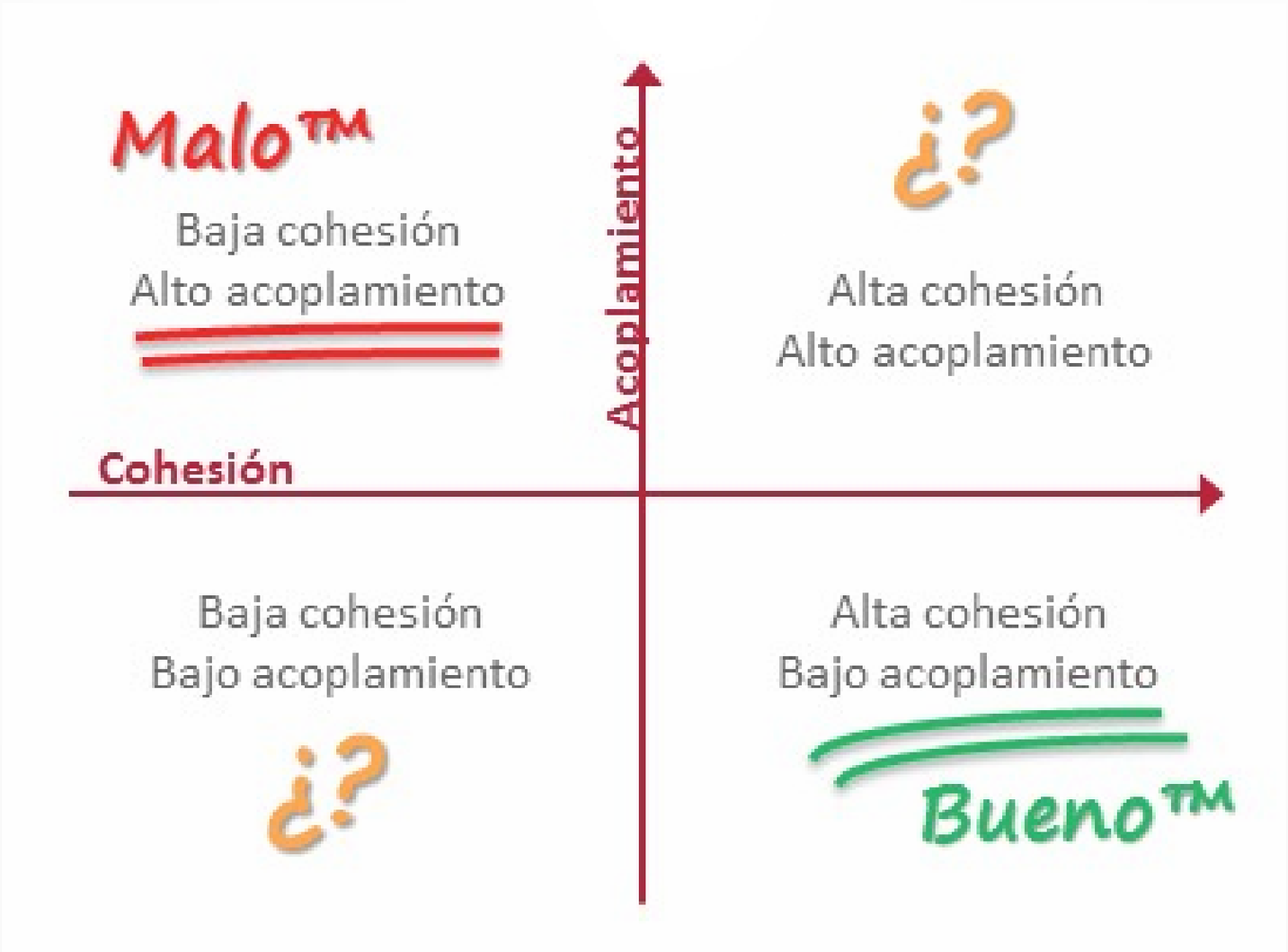
Qué es la cohesión

- La cohesión nos indica el grado de relación existente entre los distintos elementos de una clase. Una clase tendrá una cohesión alta cuando todos sus métodos estén relacionados entre si, bien mediante llamadas entre ellos o bien mediante el uso de variables cuyo ámbito se encuentre a nivel de clase. Y esto suele ocurrir cuando una clase realiza una única tarea.
- Cuanto más alta sea la cohesión mejor.

Qué es el acoplamiento

- Y si la cohesión trata sobre la relación existente entre los elementos de la propia clase, el acoplamiento lo hace sobre el nivel de dependencia de la clase con respecto a otros elementos externos. Por lo tanto, diremos que una clase tiene un nivel de acoplamiento alto cuan hace uso en gran medida de otros componentes.
- Nuestro objetivo será siempre construir clases con un nivel de acoplamiento bajo.

1.1. Introducción a Spring



1.1. Introducción a Spring

Beneficios de una cohesión alta

- Mejora de la mantenibilidad del código. Al realizar una única tarea, los cambios realizados sobre la propia clase afectarán en menor medida al código que hace uso de ella.
- Mejora de la lectura y comprensión. Es mucho más sencillo leer y entender una clase con una única responsabilidad, pues será más pequeña y simple.

Beneficios de un bajo acoplamiento

- Mejora de la facilidad de reutilización. Al no tener dependencia de elementos externos será muchos más sencillo portar nuestra clase a otros componentes.
- Mejora de la mantenibilidad. En este caso por partida doble: en primer lugar los cambios realizados en la clase no estarán condicionados por requisitos de estructuras de software externas, y en segundo lugar porque al no tener dependencias externas no se verá afectado por modificaciones en éstas.
- Mayor facilidad de comprensión. Pues no tendremos que buscar ni comprender otros módulos para aprender cómo funciona el que nos interesa.

1.1. Introducción a Spring

Patrón de inyección de dependencias (DI)

- DI es un patrón de diseño que sirve para “inyectar” componentes a las clases que tenemos implementadas. Esos componentes son contratos que necesitan nuestras clases para poder funcionar, de ahí el concepto de “dependencia”. La diferencia sustancial en este patrón de diseño es que nuestras clases no crearán esos objetos que necesitan, sino que se les suministrará otra clase “contenedora” perteneciente al Framework DI que estemos utilizando y que inyectarán la implementación deseada a nuestro contrato, y todo ello sin tener que hacer un solo “new”.

1.1. Introducción a Spring

Patrón de inyección de dependencias

- La inyección de dependencias es quizás la característica más destacable del core de Spring Framework.
  - Consiste que en lugar de que cada clase tenga que instanciar los objetos que necesite, sea Spring el que inyecte esos objetos
    - lo que quiere decir que es Spring el que creara los objetos y cuando una clase necesite usarlos se le pasarán (como cuando le pasas un parámetro a un método).

1.1. Introducción a Spring

Spring Container

- El contenedor de Spring es uno de los puntos centrales de Spring, se encarga de crear los objetos, conectarlos entre si, configurarlos y además controla los ciclos de vida de cada objeto mediante el patrón de Inyección de Dependencias (Dependency Injection ó DI).

1.1. Introducción a Spring

Spring Container

- El contenedor de Spring es uno de los puntos centrales de Spring, se encarga de crear los objetos, conectarlos entre si, configurarlos y además controla los ciclos de vida de cada objeto mediante el patrón de Inyección de Dependencias (Dependency Injection ó DI).

1.1. Introducción a Spring

Spring Container

- El contenedor de Spring es uno de los puntos centrales de Spring, se encarga de crear los objetos, conectarlos entre si, configurarlos y además controla los ciclos de vida de cada objeto mediante el patrón de Inyección de Dependencias (Dependency Injection ó DI).

1.1. Introducción a Spring

- Sin el patrón DI

```
1  public class GeneradorPlaylist {
2
3      private BuscadorCanciones buscadorCanciones;
4
5      public GeneradorPlaylist(){
6          this.buscadorCanciones = new BuscadorCanciones();
7      }
8
9      //Resto de métodos de la clase
10
11 }
```

1.1. Introducción a Spring

- Patrón DI mediante el constructor

```
1  public class GeneradorPlaylist {
2
3      private BuscadorCanciones buscadorCanciones;
4
5      public GeneradorPlaylist(BuscadorCanciones buscadorCanciones){
6          this.buscadorCanciones = buscadorCanciones;
7      }
8
9      //Resto de métodos de la clase
10
11 }
```

1.1. Introducción a Spring

- Patrón DI mediante el constructor

```
1  public class GeneradorPlaylist {
2
3      private BuscadorCanciones buscadorCanciones;
4
5      public GeneradorPlaylist(BuscadorCanciones buscadorCanciones){
6          this.buscadorCanciones = buscadorCanciones;
7      }
8
9      //Resto de métodos de la clase
10
11 }
```

1.1. Introducción a Spring

Patrón DI mediante el configuración del archivo xml

```
<!-- Le decimos a Spring que cree el Bean que luego inyectaremos  -->
    <bean id="buscadorCanciones" class="com.autentia.BuscadorCanciones">

    <!-- Creamos el Bean y le inyectamos el buscadorCanciones que hemos creado arriba -->
    <bean id="generadorPlaylist" class="com.autentia.GeneradorPlaylist">
        <constructor-arg type="com.autentia.BuscadorCanciones" ref="buscadorCanciones">
    </constructor-arg></bean>

</bean>
```

1.1. Introducción a Spring

- Patrón DI mediante Setter

```java
public class GeneradorPlaylist {

    @autowired
    private BuscadorCanciones buscadorCanciones;

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

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

@autowired le indicamos a Spring que se tiene que encargar de buscar un Bean que cumpla los requisitos para ser inyectado, en este caso el único requisito es que sea del tipo BuscadorCanciones, en caso de que hubiese mas de un Bean que cumpliese esos requisitos tendríamos que decirle a Spring cuál es el Bean correcto.