Tema 2 – Spring Inyección de dependencias

Grado en Ingeniería Informática en Tecnologías de la Información

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

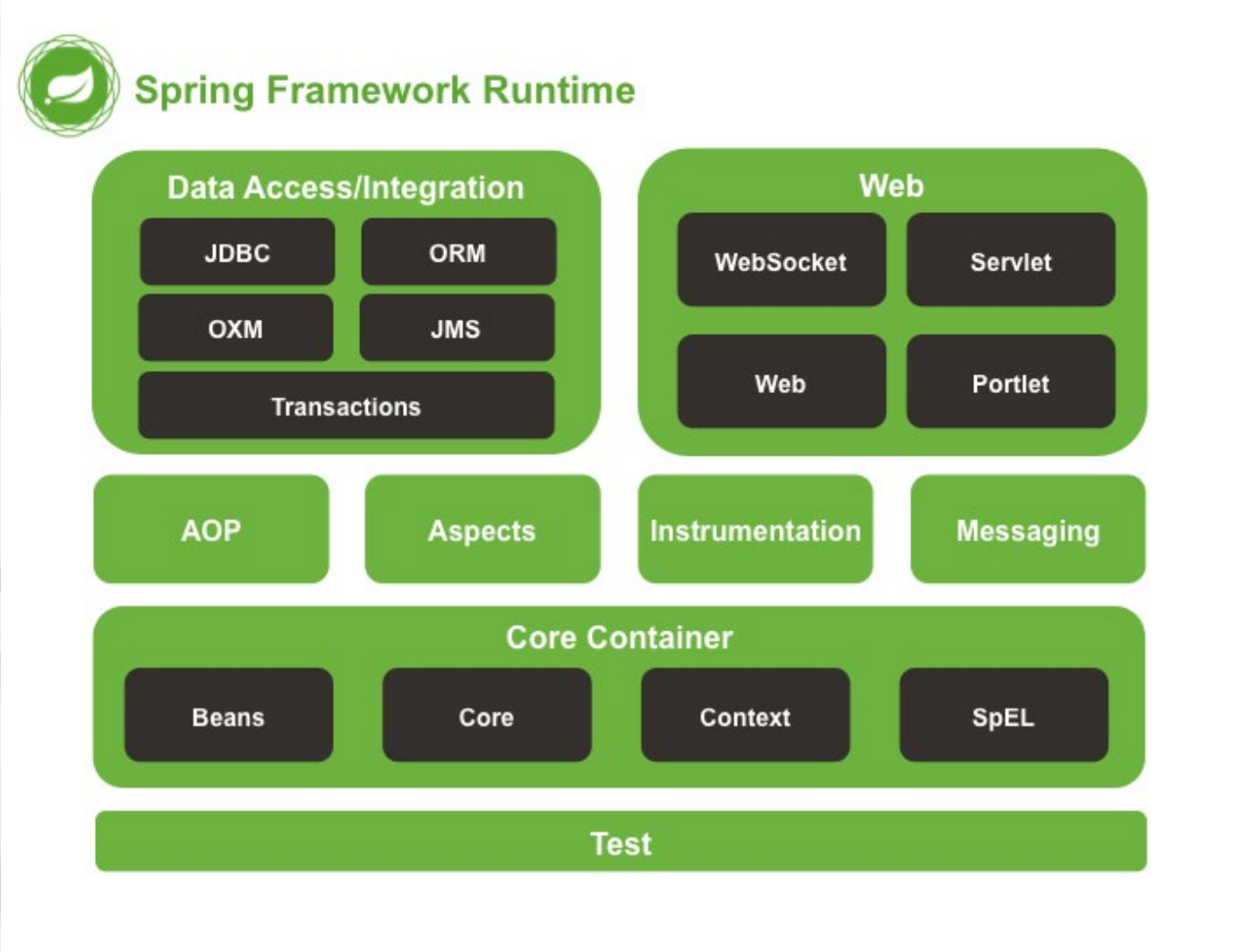
Área de Lenguajes y Sistemas Informáticos

Dr. Luis V. Calderita

Spring

- Spring tiene una estructura modular que le permite elegir sólo aquellos componentes que necesita.
- Su filosofía de diseño se basa:
  - Inversión de Control (Inversion of Control).
    - Consigue bajo acoplamiento entre los componentes
  - Inyección de dependencias (Dependencies Inyection)
  - Ambas implementadas en el módulo Spring Core

Módulos principales de Spring



Spring Core

- ¿Qué es la Inyección de Dependencias (ID)?
  - Es un patrón de diseño de software usado en la POO, que trata de solucionar las necesidades de creación de los objetos de una manera práctica, útil, escalable y con una alta versatilidad del código.
  - Permite describir cómo los objetos deberían ser creados en lugar de crearlos directamente.

Spring Core: Inyección de Dependencias

- Ahora no se crean los objetos con **new** sino que declaran qué dependencias tienen.
  - Esta dependencias se declaran en ficheros de configuración y/o **mediante anotaciones**.
  - Posteriormente un contenedor les proporciona (**inyecta**) esas dependencias
- En Spring estos *objetos* se conectan mediante el contenedor IoC (Spring Core).

Entendiendo la Inyección de Dependencias

- En la POO se distinguen dos partes:
  - una en la que creamos los objetos
  - otra en la que los usamos
- Se busca el *desacoplamiento*:
  - Se dice que un buen funcionamiento es aquel en el que los objetos nunca construyen aquellos otros objetos que necesitan para funcionar.
  - Esa parte de creación de los objetos se debe hacer en otro lugar diferente a la inicialización de un objeto.

Entendiendo la DI: Ejemplo en pseudocódigo

```
class Programador{
        ordenador
        lenguaje
        constructor(){
                this.ordenador = new Mac()
                this.lenguaje = new ObjectiveC()
        }
}
miguel = new Programador()
```

- La clase Programador (depende) o está *fuertemente acoplada* con la del tipo del ordenador (Mac) o el tipo lenguaje (ObjectiveC)

Entendiendo la Inyección de Dependencias

- Desacoplemos la dependencia con esta versión del mismo código:

```
class Programador{
        ordenador
        lenguaje
        constructor(ordenador, lenguaje){
                this.ordenador = ordenador
                this.lenguaje = lenguaje
        }
}
miguel = new Programador( new Mac(), new ObjectiveC() )
carlos = new Programador( new Windows(), new Java() )
```

Entendiendo la Inyección de Dependencias

- En la segunda versión, el *Programador* puede adaptarse a cualquier tipo de ordenador y cualquier tipo de lenguaje.
- En realidad es tan “sencillo” como apreciar que al constructor de los objetos se les están pasando (inyectando) aquellas dependencias que ellos necesitan para poder realizar sus tareas.

Inyección de Dependencias: Problema

- ¿Qué pasa si la clase Programador depende de muchos objetos ?
  - Todos deben ser pasados en el constructor, se complica el código, es poco elegante.
- La solución está en el contenedor de dependencias, también llamado inyector de dependencias, contenedor de servicios… y en invertir el control (Inversion of Control)

Inyección de Dependencias: Solución

- Contenedor de dependencias e inversión de control:
  - Básicamente es *algo* que nos permite construir los objetos con sus dependencias
  - El contenedor de dependencias tiene todos los objetos que puedas necesitar para crear cualquier objeto complejo.
  - Además, si no cuenta en ese instante con las dependencias necesarias, sabe cómo conseguirlas en el acto.

```
miguel = contenedorDependencias.crear("Programador");
```

ID: Spring IoC Container

- En Spring, los objetos que forman la columna vertebral de las aplicaciones, y que son administradas por el Spring IoC Contanier, son llamados "Beans".
- Un bean es un objeto que es instanciado, ensamblado (cuando sus dependencias son inyectadas), y en general, administrado por el contenedor de IoC.
- Un bean es uno de los muchos objetos de nuestra aplicación.
- Los Beans, y las dependencias entre ellos, se declaran en los metadatos del contenedor de IoC. Bien **mediante anotaciones**, bien en archivos de mapeo.

@Component

- La anotación **@Component** de Spring se utiliza para denotar una clase como un Componente.
- Significa que Spring detectará automáticamente estas clases para hacer la inyección de dependencia.
- Las incluirá en el Contenedor de Dependencias (IoC Container). Cuando se necesita creará el bean asociado a la clase.
- **Aviso:** Esta anotación ya no se usa debido a que se han creado tres (hijas) más específicas para marcar una clase como un tipo de Componente

Practicando: Creando un *bean*

- En la aplicación demo:
  - Crea una nueva clase Foo con un constructor vacío.
  - Pseudocódigo:

```
class Foo{

      Foo(){
            print(“\t Soy Foo”)
      }
}
```

Clase principal

- Ejecuta la aplicación. Observa la consola.
  - ¿Qué ocurre?
- **Aviso**: En este ejemplo esta clase no debe ser modificada.

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

Creando un *bean*: @component

- En la aplicación demo:
  - Anota la nueva clase Foo con @Component.
    - Fíjate en la clase principal para ver dónde crees que debe ir la anotación.
  - Ejecuta la aplicación. Observa la consola
    - ¿Qué ocurre ahora?

Recursos

- Spring IoC Container and beans
  - http://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html
- Estas diapositivas están basadas en el trabajo del profesor Dr. Luis Arévalo, LSI. CUMe-UEX.