SPRING | COUPLING & IoC & SPRING CONFIGURATION

20.06.23

When we talk about coupling, we describe the degree to which classes within our system depend on each other. Our goal during the development process is to reduce coupling.

Tight Coupling

When a group of classes is highly dependent on each other, or we've classes that assume a lot of responsibilities is called tight coupling.

Loose Coupling

Loose coupling is when an object gets the object to be used from external sources (e.g. Calculator class uses IShape interface instead of Triangle or Square classes)

Dependency Injection (DI)

In software engineering, dependency injection is **a programming technique in which an object or function receives other objects or functions that it depends on**.

```
public class Main {
    public static void main(String[] args) {
        var shape = new Triangle(); // Creating object
        var calculator = new Calculator(shape); // Creating object +
Wiring of Dependencies
        // Here, shape is a dependency for calculator.
        // shape object is injected in calculator object. ==> DI:
Dependency Injection
        calculator.calculate();
    }
}
```

Dependency Injection Types

GitHub: https://github.com/HulyaMartli/SpringExercises2-Coupling-IoC

Spring'de 3 tip Dependency Injection bulunur:

1. Constructor DI
2. Setter DI
3. Field DI

Constructor DI

@Autowired anotasyonu kullanılmadan dependency injection yapılabilen tek yöntemdir. En önemlisidir. Genellikle bunu tercih edin, %98 bu yöntem kullanılır. @Autowired anotasyonu constructor üstüne eklenebilir de eklenmeyebilir de, yine de dependencyler inject edilecektir.

```
@Component
class ConstructorService{
    // Constructor Dependency Injection
    ConstructorDependency1 dependency1;
    ConstructorDependency2 dependency2;

    @Autowired
    public ConstructorService(ConstructorDependency1 dependency1,
ConstructorDependency2 dependency2) {
        this.dependency1 = dependency1;
        this.dependency2 = dependency2;
    }

    @Override
```

```
    public String toString() {
        return "Service{" +
                "dependency1=" + dependency1 +
                ", dependency2=" + dependency2 +
                '}';
    }
}

@Component
class ConstructorDependency1{

}

@Component
class ConstructorDependency2{

}

@Configuration
@ComponentScan
public class ConstructorDI {
    public static void main(String[] args) {
        var context = new
AnnotationConfigApplicationContext(ConstructorDI.class);

        // All beans controlled by Spring

Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::
println);

System.out.println("================================================
========");

System.out.println(context.getBean(ConstructorService.class));

    }
}
```

Setter DI

```
@Component
class SetterService{
    // Setter Dependency Injection
    SetterDependency1 dependency1;
    SetterDependency2 dependency2;

    @Autowired
    public void setDependency1(SetterDependency1 dependency1) {
        this.dependency1 = dependency1;
    }

    @Autowired
    public void setDependency2(SetterDependency2 dependency2) {
        this.dependency2 = dependency2;
    }

    @Override
    public String toString() {
        return "Service{" +
                "dependency1=" + dependency1 +
                ", dependency2=" + dependency2 +
                '}';
    }
}

@Component
class SetterDependency1{

}

@Component
class SetterDependency2{

}

@Configuration
@ComponentScan
public class SetterDI {
    public static void main(String[] args) {
        var context = new
AnnotationConfigApplicationContext(SetterDI.class);
```

```
        // All beans controlled by Spring

Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::
println);

System.out.println("==========================================
========");
        System.out.println(context.getBean(SetterService.class));

    }
}
```

Field DI

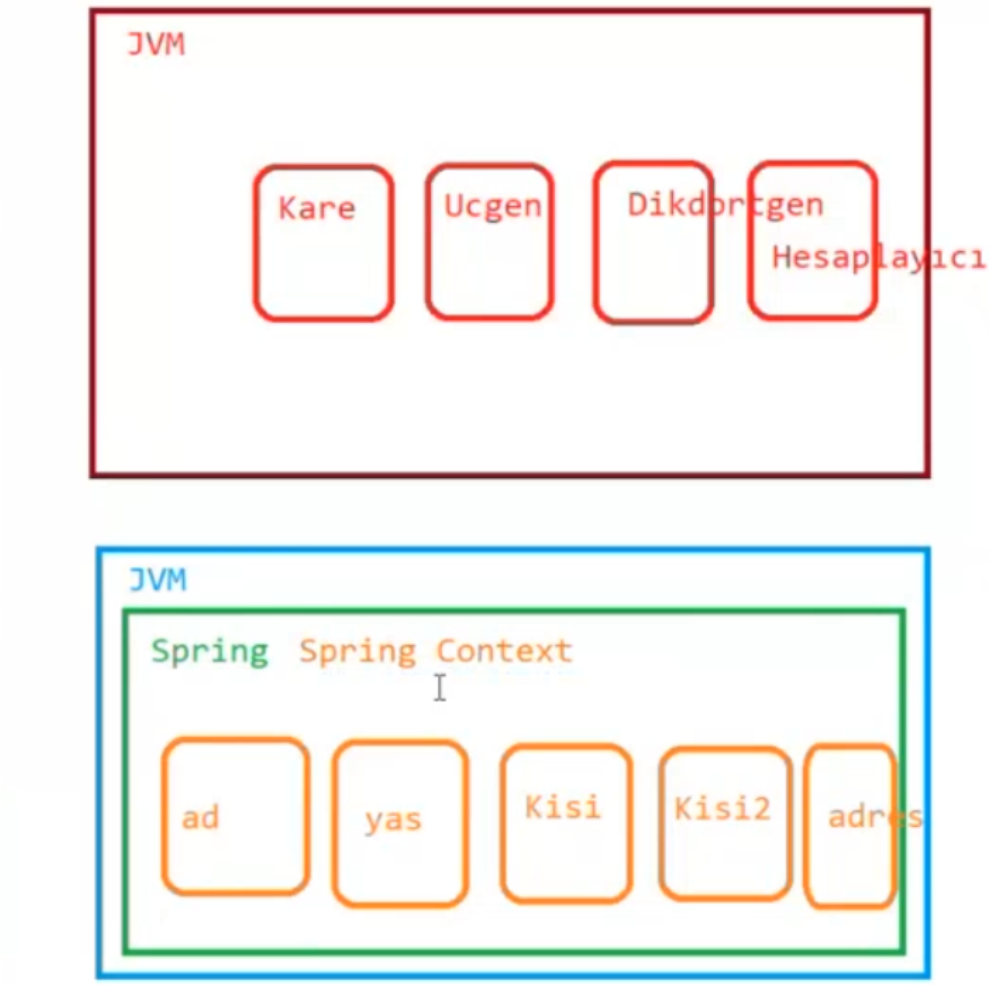
Arka planında Reflection vardır. Burada constructora veya setter metoda ihtiyaç yoktur.

```
@Component
class Service{
    // Field Dependency Injection
    @Autowired
    Dependency1 dependency1;

    @Autowired
    Dependency2 dependency2;

    @Override
    public String toString() {
        return "Service{" +
                "dependency1=" + dependency1 +
                ", dependency2=" + dependency2 +
                '}';
    }
}

@Component
class Dependency1{

}

@Component
```

```
class Dependency2{

}

@Configuration
@ComponentScan
public class Main {
    public static void main(String[] args) {
        var context = new
AnnotationConfigApplicationContext(Main.class);

        // All beans controlled by Spring

Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::
println);

System.out.println("================================================
========");
        System.out.println(context.getBean(Service.class));

    }
}
```

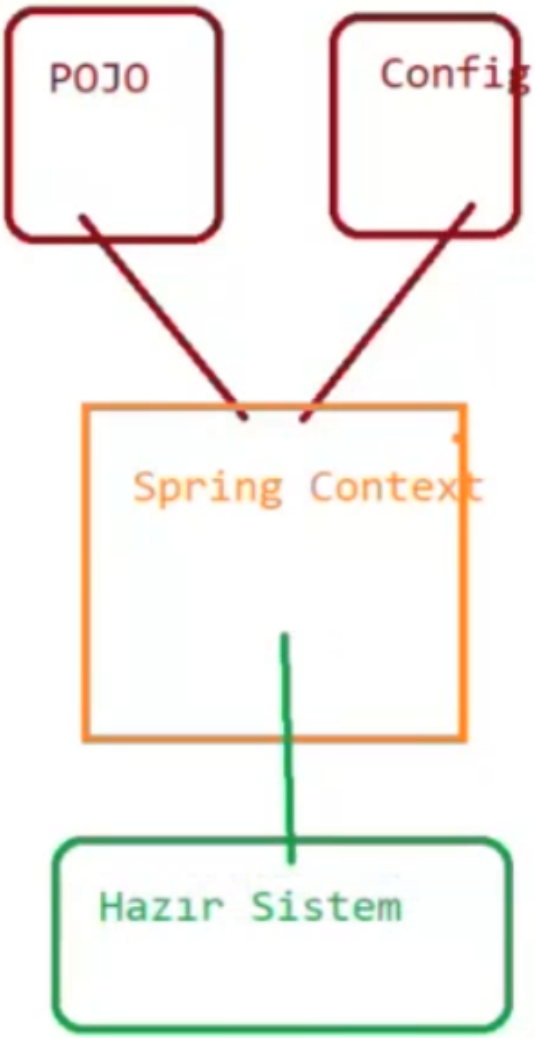
Inversion of Control (IoC)

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework.



Spring Context - Spring Container - IoC Container

Hepsi aynı kavramı tanımlayan kavramlardır. Amaçları Spring beanleri ve onların yaşam döngülerini yönetmektir:



SPRING COUPLING, IoC CONFIG & DI EXAMPLE

GitHub: https://github.com/HulyaMartli/SpringExercises2-Coupling-IoC

Sınıfları, özellikleri, nesneleri (Spring Beans olarak geçer bunlar. Yukarıdaki resimde turuncu ile gösterilenler) yaratma ve birbirine bağlama işlerini Spring'in devralabilmesi için Spring Context (Spring Container veya IoC Container) oluşturulur.

Spring Bean: Spring tarafından yönetilen Java objeleridir. ***@Bean*** anotasyonu ile oluşturulur.

Java Bean: Bir sınıfın Java Bean olabilmesi için kesinlikle sahip olması gereken 3 özellik vardır:

1. Public boş constructor
2. Getters & Setters

3. Serializable interface'ten implement alması

> ***ejb:*** *Enterprise Java Bean*
>
> ***Neden kullanılıyordu?***

Pojo: Pojo'da ise hiçbir şey olma zorunluluğu yoktur. İçinde constructor, getter&setter vs. olabilir de olmayabilir de. Herhangi bir kısıtlaması olmayan Java sınıflarıdır.

build.gradle

Spring'in projemizde çalışabilmesi için öncelikle gerekli dependencies https://mvnrepository.com/'dan eklenmelidir.

- Spring Core (Versiyon 5'ten sonuncusunu seçtik)
- Spring Context (Yine versiyon 5'ten sonuncusu) - @Configuration anotasyonunu kullanabilmek ve SpringConfiguration classında Spring için ayarları belirtebilmek için ekliyoruz.

```
plugins {
    id 'java'
}

group = 'org.hulyam'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    //
https://mvnrepository.com/artifact/org.springframework/spring-core
    implementation 'org.springframework:spring-core:5.3.29'
    //
https://mvnrepository.com/artifact/org.springframework/spring-
context
    implementation 'org.springframework:spring-context:5.3.29'

}
```

Spring Context and SpringConfigutation

SpringConfiguration:

Spring'in ayar dosyasıdır. ***@Configuration*** anotasyonu ile belirlenir. Spring tarafından yönetilmesini istediğimiz beanleri burada belirtiriz.

```
package org.hulyam;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SpringConfiguration {

    // @Bean anotasyonu ile Spring Context tarafından yönetilecek
bir bean oluşturuluyor.
    @Bean
    public String name() {
        return "Mr. Bean";
    }

    @Bean
    public int age(){
        return 50;
    }

}
```

> ***record:*** *Java 14 ile beraber gelmiş bir özelliktir. Yeni bir class oluşturmayı sağlar.*
>
> ```
> record Person(String name, int age);
> ```
>
> *İçerisinde name ve age özelliklerini barındıran bir sınıf oluşturmuş oluyoruz.* ***Getter, setter ve constructorı kendisi oluşturur****.*

SpringConfiguration:

```
package org.hulyam;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

record Person(String name, int age){};
record Address(String city, String country){};
```

```
@Configuration
public class SpringConfiguration {

    // @Bean anotasyonu ile Spring Context tarafından yönetilecek
bir bean oluşturuluyor.
    @Bean
    public String name() {
        return "Mr. Bean";
    }

    @Bean
    public int age() {
        return 50;
    }

    @Bean
    public Person person(){
        return new Person("Hulya",32);
    }

    @Bean("addr.")
    public Address address(){
        return new Address("Ankara","Turkiye");
    }
}
```

Main:

```
package org.hulyam;

import
org.springframework.context.annotation.AnnotationConfigApplicationCo
ntext;

public class Main {
    public static void main(String[] args) {
        // SpringConfiguration dosyasında oluşturduğumuz beane
ulaşmaya çalışıyoruz.
        // Spring Context'i oluşturmak için kullanılır, içine class
olarak ayar dosyası verilir:
        var context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);
```

```
        // BEAN CALLS
        // With Method Names
        System.out.println(context.getBean("name")); //same as the
method name in SpringConfiguration
        System.out.println(context.getBean("age"));
        System.out.println(context.getBean("person"));
        System.out.println(context.getBean("addr."));
    }
}
```

> ***Application Context:*** *Spring Context/Spring Container/IoC Container'a ulaşmanın bir yoludur. Başka yollar da vardır ama en çok bu kullanılır.*
>
> *NEDENİ:*
>
> - *Kurumsal yapılara daha uygun*
> - *Web uygulamaları için de uygun*
> - *Aspect Oriented Programming (AOP) ile de uyumludur.*

Bean içinde bean çağırma

SpringConfiguration:

```
    @Bean
    public String name() {
        return "Mr. Bean";
    }

    @Bean
    public int age() {
        return 50;
    }

    @Bean
    public Person person2(){
        return new Person(name(),age());
    }
```

```
record Person2(String name, int yas, Address address){};

@Configuration
public class SpringConfiguration {

    @Bean
```

```
    public String name() {
        return "Mr. Bean";
    }

    @Bean
    public int age() {
        return 50;
    }

    @Bean("addr.")
    public Address address(){
        return new Address("Los Angeles","USA");
    }

    @Bean
    public Person2 person2(){
        return new Person2("Nihal",40,new
Address("Istanbul","Turkiye"));
    }

    @Bean
    public Person2 person2MethodCalling(){
        return new Person2(name(),age(),address());
    }
}
```

Beanlerin Method Calling İçin Önceliklendirilmesi

@Primary

Method Calling yaparken Main'de "contect.getBean(Address)" kod parçacığı ile class name kullanılarak çağrı yapılırsa, iki veya daha fazla aynı sınıftan dönüş yapan bean varsa, bu "expected single matching bean but found 2" hatasına neden olacaktır. Bunu gidermek ve hangisinin öncelikli olarak çağırılacağını belirtmek için @Primary anotasyonu kullanılır.

SpringConfiguration:

```
    @Bean("addr.")
    public Address address(){
        return new Address("Los Angeles","USA");
    }

    @Bean
    @Primary
    public Address address2(){
        return new Address("Ankara","Turkiye");
    }
}
```

Main:

```
package org.hulyam;

import
org.springframework.context.annotation.AnnotationConfigApplicationCo
ntext;

public class Main {
    public static void main(String[] args) {
        var context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);
        // BEAN CALLS
        // With Class Names
        System.out.println(context.getBean(Address.class));
    }
}
```

@Qualifier

SpringConfiguration:

```
package org.hulyam;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;

record Address(String city, String country){};

record Person2(String name, int yas, Address address){};
@Configuration
```

```
public class SpringConfiguration {

    @Bean
    public String name() {
        return "Mr. Bean";
    }

    @Bean
    public int age() {
        return 50;
    }

    @Bean("addr.")
    public Address address(){
        return new Address("Los Angeles","USA");
    }

    @Bean
    public Person2 person2(){
        return new Person2("Nihal",40,new
Address("Istanbul","Turkiye"));
    }

    @Bean
    @Primary
    public Address address2(){
        return new Address("Ankara","Turkiye");
    }

    @Bean
    public Person2 person2Qualifier(String name, int age, Address
address){
        return new Person2(name,age,address);
    }
}
```

Burada yeni Person2 oluştururken parametre olarak metod isimlerini girdiğimiz için ve address2 metodunda @Primary anotasyonu olduğu için Main sınıfında aşağıdaki bean call yapıldığında:

Main:

```
package org.hulyam;

import
org.springframework.context.annotation.AnnotationConfigApplicationCo
ntext;

public class Main {
    public static void main(String[] args) {
        var context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);
        // BEAN CALLS
        // With Method Names
        System.out.println(context.getBean("person2Qualifier"));
```

sonuç aşağıdaki şekilde olur ve name, age metodlarından gelen değerler ile @Primary anotasyonunun kullanıldığı adres ile Person2 oluşturulur:

ÇIKTI:

```
Person2[name=Mr. Bean, yas=50, address=Address[city=Ankara,
country=Turkiye]]
```

Ama biz primary adresi değil de başka bir adres alarak bir Person2 yaratmak istiyorsak burada @Qualifier anotasyonunu kullanırız:

SpringConfiguration:

```
@Bean
    @Primary
    public Address address2(){
        return new Address("Ankara","Turkiye");
    }

    @Bean
    @Qualifier("romeQualifier")
    public Address address3(){
        return new Address("Rome","Italy");
    }

    @Bean
    public Person2 person2Qualifier(String name, int age,
@Qualifier("romeQualifier") Address address){
        return new Person2(name,age,address);
    }
```

Spring Framework Tarafından Yönetilen Tüm Beanler

```
public class Main {
    public static void main(String[] args) {
        var context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);

        // All Beans Controlled by Spring Framework
Arrays.stream(context.getBeanDefinitionNames()).forEach(System.out::
println);

    }
}
```

> *Spring'in çalışma prensibi **Eager** ve **Singleton**dır. Program çalışmaya başlar başlamaz @Bean ile belirttiğimiz tüm nesneleri oluşturur ve her biri için tek bir nesne oluşturur.*

@Configuration & @ComponentScan

@Configuration Spring beanlerinin belirtildiği sınıfı işaret eder. @ComponentScan componentlerin nerede aranması gerektiğini belirtir. Eğer bulması gereken component anotasyonu koyduğumuz sınıf ile aynı package içindeyse yol belirtilmesine gerek kalmaz.

Main:

```
package org.hulyam;

import
org.springframework.context.annotation.AnnotationConfigApplicationCo
ntext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class Main {

    @Bean
    public Calculator calculator(IShape shape) {
        return new Calculator(shape);
    }
```

```
    public static void main(String[] args) {



        var context = new
AnnotationConfigApplicationContext(Main.class);
        context.getBean(Calculator.class).calculate();

    }
}
```

IShape

```
package org.hulyam;

public interface IShape {
    void calculateArea();

    void calculatePerimeter();

    void calculateVolume();
}
```

Triangle

```
package org.hulyam;

import org.springframework.stereotype.Component;

@Component
public class Triangle implements IShape {
    public void calculateArea() {
        System.out.println("The area of the Triangle is
calculated...");
    }

    public void calculatePerimeter() {
        System.out.println("The perimeter of the Triangle is
calculated...");
    }

    public void calculateVolume() {
        System.out.println("The volume of the Triangle is
calculated...");
    }
```

```
}
```

Calculator

```
package org.hulyam;

public class Calculator {
    // Square shape;
    IShape shape;

    public Calculator(IShape shape) {
        this.shape = shape;
    }

    public void calculate() {
        System.out.println("Calculate method is running for the
shape... " + shape);
        shape.calculateArea();
        shape.calculatePerimeter();
        shape.calculateVolume();
    }
}
```

@Bean ve @Component Arasındaki Farklar

- @Component bütün Java classları üzerine yazılabilir. @Bean ise configuration dosyası üzerindeki metotların üzerine yazılır.
- @Component bütün işi kendi başına yapar ancak @Bean ile metot içinde yeni nesne yaratılması için kodlama yapılması gerekir.
- @Component ile 3 farklı yöntemle autowiring/dependency injection yapılabilir. @Bean ile 2 farklı yöntemle: 1. metot çağırımı ile, 2. parametre çağırımı ile
- @Component ile kodun hiçbir yerinde "new" kullanarak nesne oluşturmayız, hepsini Spring kendisi yapar ama @Beande metotlar içinde newleme işlemi yaparız.

> *Genellikle @Component kullanılır ancak, nesne yaratılmadan önce kontroller yapmamız gerekirse @Bean kullanımı tercih edilebilir. @Component'te biz sürece dahil olmadan kendi kendine üretir. Nesne üretilmeden önce işlemler yapmak istiyorsak @Bean ile metot içerisinde bunları yapabiliriz. Bu nedenle tercihe göre @Bean kullanımı gerekebilir.*