

YTE Stajyer Spring Eđitimi 2021

Eđitmen: Muhammed Fatih Dođmuş

Tanışma

- ▶ Muhammed Fatih Doğmuş
- ▶ ODTÜ Bilgisayar Mühendisliği 2018 mezunu
- ▶ İşe giriş eylül 2018

Ön Bilgilendirme

- Ufak bir konuşma problemim var. Eğer bir şeyi çok hızlı söylediğim için anlamadıysanız sormaktan çekinmeyin. Tekrar daha yavaş söylemeye çalışacağım.

Kurs formatı

- ▶ Soru sormaktan asla çekinmeyin! Sorum saçma olabilir, beni yargırlar diye düşünmeyin, soru sormak öğrenmenin en iyi yollarından biridir.
- ▶ Her 50 dakikada bir 10 dakika mola
- ▶ Bol bol örnek çözeceksiniz
- ▶ Yazarak sorulan soruları cevaplamayacağım. El kaldırıp söz alarak soru sormanızı rica ediyorum.

İçerik

- ▶ 4 Ana Konu
 - ▶ Spring Core
 - ▶ Spring Web
 - ▶ Spring Data
 - ▶ Spring Security

Spring'e giriş

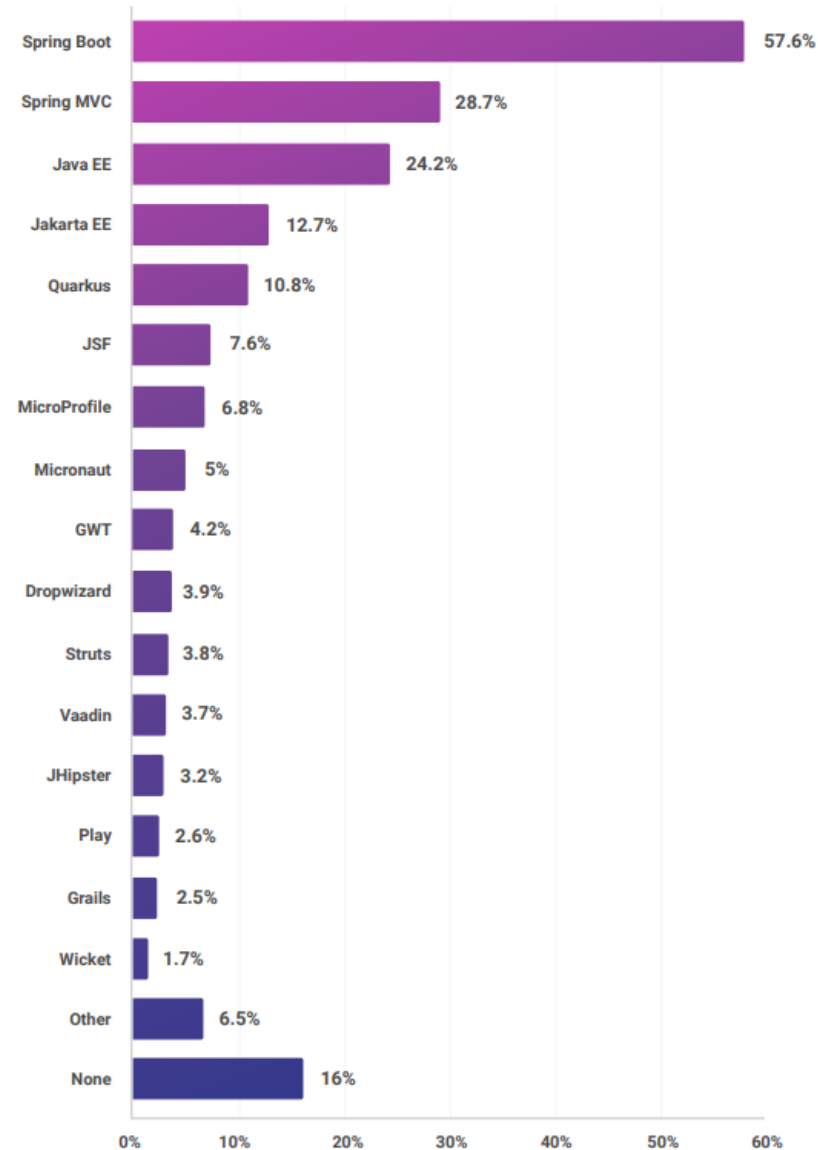
Nedir, ne yapar

- ▶ Web uygulama geliřtirmek iin tamamlayıcı bir ekosistem saėlar.

Alternatifleri nelerdir?

- ▶ Markette pek çok java web framework'ü bulunmakta
 - ▶ Quarkus
 - ▶ Micronaut
 - ▶ Jakarta EE
 - ▶ Helidon
 - ▶ Spark
 - ▶ Play
- ▶ Fakat bu framework'ler, spring kadar gelişmiş ekosisteme sahip değil

Snyk JVM Ecosystem Survey 2021

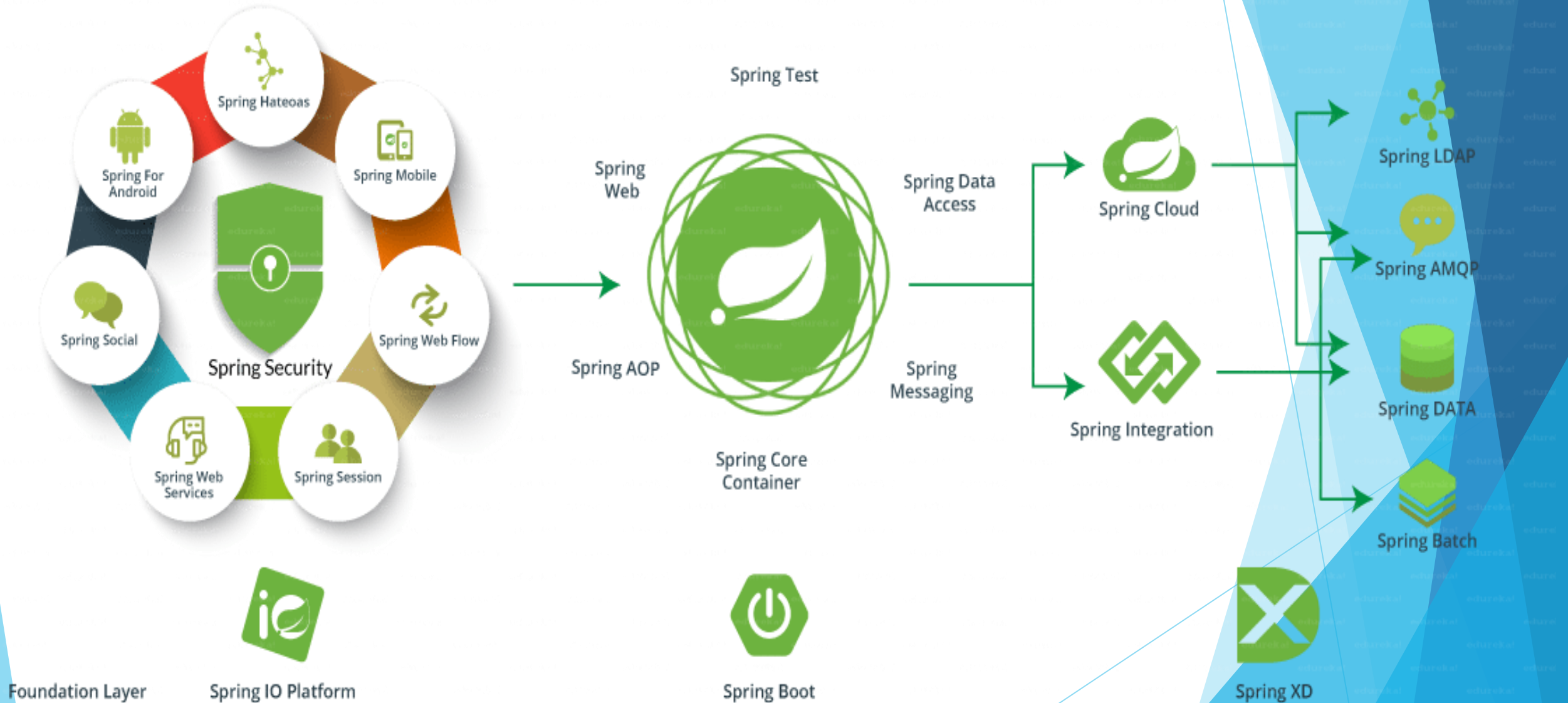


Web Layer

Common Layer

Service Layer

Data Layer



Inversion of Control

- ▶ Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. -Baeldung
- ▶ Herhangi bir programlama dilinde obje oluşturma işi elle yapılır. IoC ise bu kontrolü tersine çevirir.

Örnek

```
private MongoStudentRepository mongoStudentRepository;

public StudentService() {
    mongoStudentRepository = new MongoStudentRepository();
}

public List<Student> getAllStudents() {
    return mongoStudentRepository.findAll();
}
```

```
private IStudentRepository studentRepository;

public StudentService(IStudentRepository studentRepository) {
    this.studentRepository = studentRepository;
}

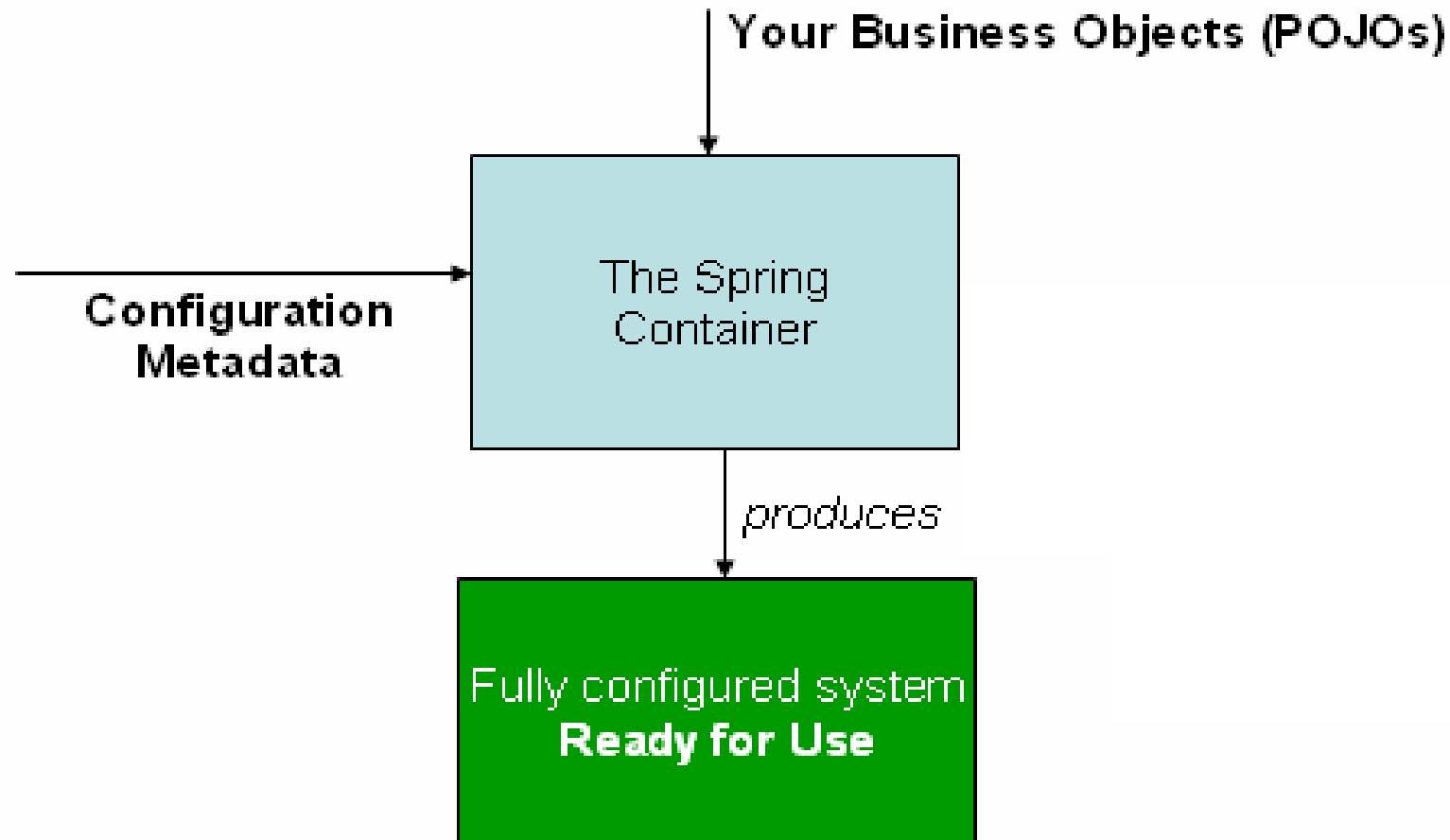
public List<Student> getAllStudents() {
    return studentRepository.findAll();
}
```

Faydaları

- ▶ Farklı implementation'lar arasında geçiş yapmayı kolaylaştırır
- ▶ Daha rahat test yazılabilmesini sağlar

Spring core

Spring Container ve Application Context



Aplication Context oluşturma

- ▶ XML tabanlı
- ▶ Annotation tabanlı
- ▶ Java tabanlı

Bean Tanımı

```
@Component  
public class ComponentTest {  
  
    public void print() {  
        System.out.println("2019 YTE Yaz Stajyer Programı");  
    }  
}
```



IoC ile ilgili Örnek

- ▶ start.spring.io
- ▶ Generate Project
- ▶ Unzip
- ▶ IntelliJ ile import
- ▶ Pom.xml dosyasını açın
- ▶ Finish

Dependency Injection

- ▶ Dependency injection is a pattern through which to implement IoC, where the control being inverted is the setting of object's dependencies.

```
private IRepository studentRepository;  
  
public StudentService(IRepository studentRepository) {  
    this.studentRepository = studentRepository;  
}  
  
public List<Student> getAllStudents() {  
    return studentRepository.findAll();  
}
```

Dependency Injection yöntemleri ve @Autowired

- ▶ Wiring Spring'in yönettiği dependency'lerin birbirine bağlaması.
- ▶ Autowiring bunun otomatik olarak yapar.
- ▶ Eğer spring @Autowired annotation'ı ile işaretlenmiş bir dependency bulursa ve o dependency'e ait bir bean varsa, bunu otomatik olarak inject eder.

Field Injection

```
@Component
public class AutowireTest {

    @Autowired
    private ComponentTest componentTest;

    public void test() {
        componentTest.autowireTest();
    }
}
```

```
@Component
public class ComponentTest {

    public void print() {
        System.out.println("2019 YTE Yaz Stajyer Programı");
    }

    public void autowireTest() {
        System.out.println("Bu fonksiyon autowire testi için kullanılmaktadır.");
    }
}
```

Setter Injection

```
@Component
public class AutowireTest {

    private ComponentTest componentTest;


    public void test() {
        componentTest.autowireTest();
    }

    @Autowired
    public void setComponentTest(ComponentTest componentTest) {
        this.componentTest = componentTest;
    }
}
```


Constructor Injection

```
@Component
public class AutowireTest {

    private ComponentTest componentTest;

    
    @Autowired
    public AutowireTest(ComponentTest componentTest) {
        this.componentTest = componentTest;
    }

    public void test() {
        componentTest.autowireTest();
    }
}
```



PRACTiCE

Uygulama

- ▶ 3 class:
 - ▶ FieldInjectionBean
 - ▶ SetterInjectionBean
 - ▶ ConstructorInjectionBean
- ▶ Her class ne tip injection için kullanılacağına dair bir string print eden print fonksiyonu olacak(Ör: 'Ben bir field injection bean'iyim')
- ▶ 1 test class'ı: DependencyInjectionTest. Yukarıdaki 3 class bu class'ta üç farklı şekilde inject edilecek. Bu class'ın da bir print fonksiyonu olacak ve diğer 3 class'ın print fonksiyonlarını çağırarak.
- ▶ DependencyInjectionTest class'ını main fonksiyonundaki application context'in getBean fonksiyonu ile alın ve print fonksiyonunu çağırın
- ▶ Dikkat: Tüm class'ların başına @Component koymayı unutmayın

Hangi injections yöntemini kullanmalı?

- ▶ %99 durumlarda constructor injection
- ▶ Kalan nadir %1 için setter injection
- ▶ Field injection kullanmayın

Aynı tanımlı birden fazla
bean

@Qualifier ve @Primary

- Eğer bir dependency için birden fazla class'ınız varsa ne olur



Örnek

```
@Component
public class CassandraRepository implements IRepository {
}
```

```
@Component
public class MongoRepository implements IRepository {
}
```

```
@Component
public class AutowireTest {

    @Autowired
    private IRepository repository;
}
```

Description:

Field repository in internship.teaching.springcore.test.AutowiredTest required a single bean, but 2 were found:

- cassandraRepository: defined in file [C:\Users\Fatih\Desktop\springcore\target\classes\internship\teaching\springcore\test\CassandraRepository.class]
- mongoRepository: defined in file [C:\Users\Fatih\Desktop\springcore\target\classes\internship\teaching\springcore\test\MongoRepository.class]

Action:

Consider marking one of the beans as @Primary, updating the consumer to accept multiple beans, or using @Qualifier to identify the bean that should be consumed

Çözüm? @Qualifier

```
@Component
public class AutowireTest {

    @Autowired
    @Qualifier("mongoRepository")
    private IRepository repository;
}
```

```
@Component
public class AutowireTest {

    private IRepository repository;

    @Autowired
    public AutowireTest(@Qualifier("mongoRepository") IRepository repository) {
        this.repository = repository;
    }
}
```

```
@Component
public class AutowireTest {

    private IRepository repository;

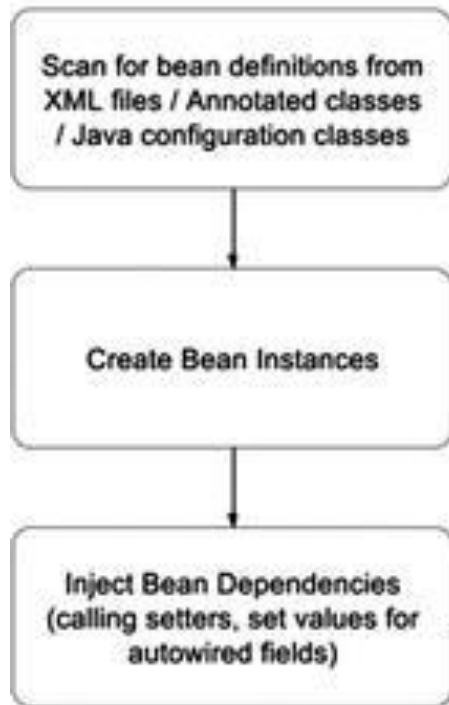
    @Autowired
    public void setRepository(@Qualifier("mongoRepository") IRepository repository) {
        this.repository = repository;
    }
}
```

Veya @Primary

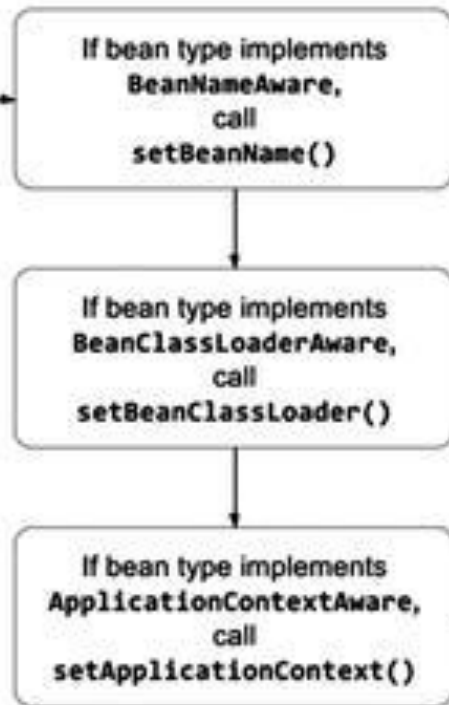
```
@Component
@Primary
public class MongoRepository implements IRepository {
}
```

Bean lifecycles

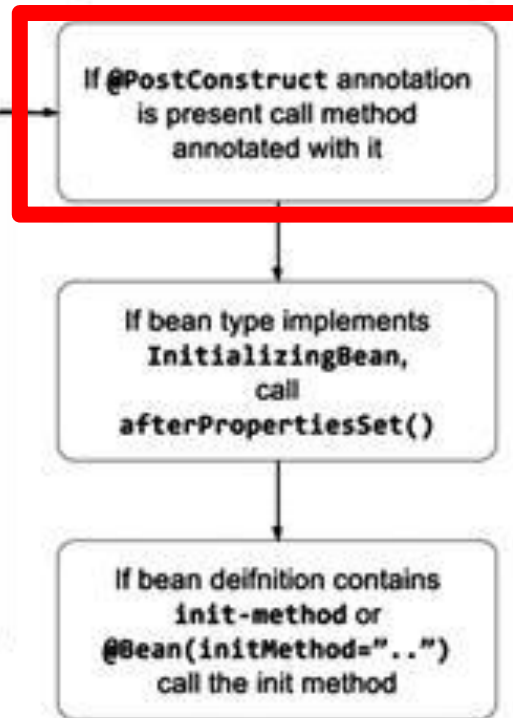
Bean Instantiation and DI



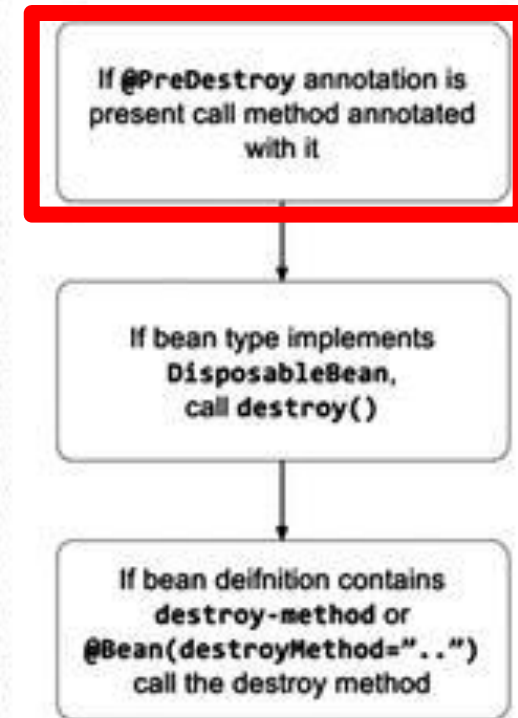
Check for Spring Awareness



Bean Creation Life-Cycle Callback



Bean Destruction Life-Cycle Callback



Callback fonksiyonları

- ▶ @PostConstruct:
 - ▶ Bu annotation ile işaretlenmiş fonksiyonlar constructor çağırıldıktan ve tüm dependency'ler inject edildikten sonra çağrılır. Çok kullanışlı bir annotation.
- ▶ @PreDestroy:
 - ▶ Bean yok edilmeden hemen önce çağrılır. Ver tabanı connection'larını kapatmap için, aldığınız kaynakları(memory, dosya vs) serbest bırakmak için, veri tabanını temizlemek gibi görevler için kullanılabilir.
- ▶ İki annotation da bean'in içerisinde methodların tepesine konur

```
@Component
public class AutowireTest {

    @PostConstruct
    public void init() {

    }

    @PreDestroy
    public void destroy() {

    }
}
```



PRACTiCE

Uygulama

- ▶ Bir component oluşturun. İçinde 4 tane print statement'ı olacak ve nerede olduklarını yazacaklar:
 - ▶ Bir print methodunun içerisinde
 - ▶ @PostConstruct'ın içinde
 - ▶ Constructor içinde
 - ▶ @PreDestroy içinde
- ▶ Main fonksiyonunda bu bean'i getBean ile alın ve print fonksiyonunu çağırın. Hangi sırada çağırıldıklarını gözlemleyin.

Bean tanımlamanın yöntemleri

- ▶ Annotation based
- ▶ Java based(@Bean)
- ▶ XML based

Bean tanımlamanın diğer yöntemleri

```
@Configuration
public class BeanConfiguration {

    @Bean
    public JavaBasedBean javaBasedBean() {
        return new JavaBasedBean();
    }
}
```

@Bean ile Dependency Injection

```
public class JavaBasedBean {  
    private MyFirstBean myFirstBean;  
  
    public JavaBasedBean(MyFirstBean myFirstBean) {  
        this.myFirstBean = myFirstBean;  
    }  
}  
  
@Component  
public class MyFirstBean {  
  
    public void print() {  
        System.out.println("My first bean!");  
    }  
}
```

```
@Configuration  
public class BeanConfiguration {  
  
    @Bean  
    public JavaBasedBean javaBasedBean(final MyFirstBean myFirstBean) {  
        return new JavaBasedBean(myFirstBean);  
    }  
}
```

@Bean ile diğer bean özellikleri

- ▶ Callback functions
- ▶ @Qualifier ve @Primary annotation'ları

XML ile bean tanımlama

```
<beans>
  <bean id="thingOne" class="x.y.ThingOne">
    <constructor-arg ref="thingTwo"/>
    <constructor-arg ref="thingThree"/>
  </bean>

  <bean id="thingTwo" class="x.y.ThingTwo"/>

  <bean id="thingThree" class="x.y.ThingThree"/>
</beans>
```



XML
based
configuration



Java or
Annotation
based
configuration

@Bean vs @Component

- ▶ Kendi class'larımız için neredeyse her zaman @Component veya diğer steryotype annotation'ları kullanmak gerek
- ▶ @Bean'in kullanım alanları:
 - ▶ Spring context'ine almamız gereken bizim olmayan class'lar(kütüphanelerden gelenler)
 - ▶ Configuration yapmak için kullanılır



PRACTiCE

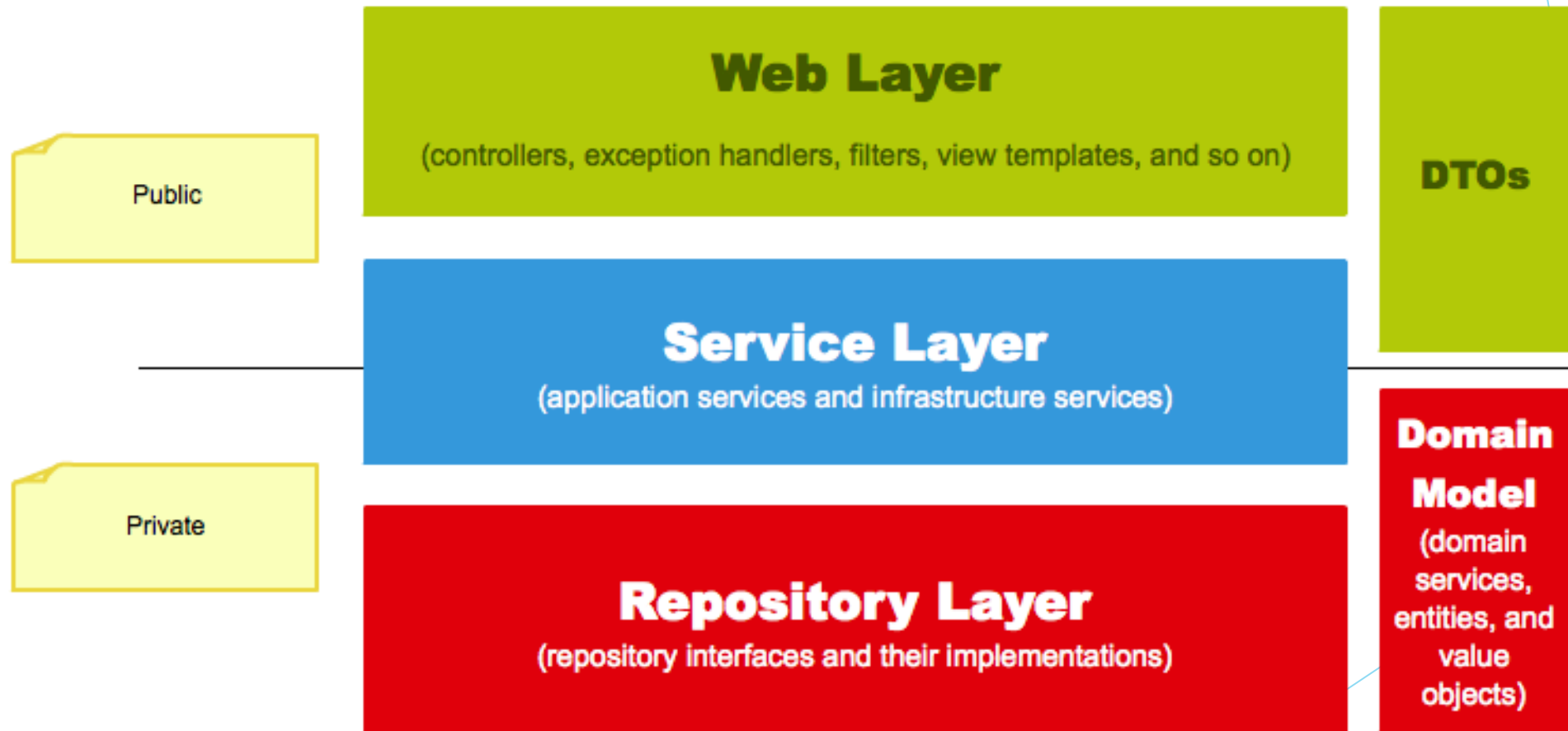
Uygulama

- ▶ 2 class tanımlayın:
 - ▶ Biri ComponentTest, diğeri BeanTest
 - ▶ ComponentTest class'ını @Component ile annotate edin ve 'Ben @Component kullanıyorum' yazan bir print function'ı yazın
 - ▶ Benzer durumu BeanTest ile de yapın ama @Component koymayın
- ▶ Configuration adlı bir class açın, @Configuration ile annotate edin ve @Bean kullanarak BeanTest class'ını bir spring bean'i olarak tanımlayın
- ▶ InjectionTest diye bir class oluşturup constructor injection ile yukarıdaki iki bean'i oluşturun ve bir print function'ı yazarak diğer iki class'ın print function'larını çağırın
- ▶ Main fonksiyonunda InjectionTest'i getBean ile alıp print function'ını çağırın

Stereotype annotations

- ▶ Domain-Driven Design-Eric J. Evans
- ▶ @Controller: Dışarıdan gelen request'leri karşılayan class'ların başına konur
- ▶ @Repository: Veri tabanı erişimi sağlayan class'ların başına konur
- ▶ @Service: Bussiness Logic(İş kuramı) içeren class'ların başına konur

Layered Architecture



External Properties

```
spring.h2.console.enabled=true  
server.port=8090  
spring.h2.console.path=/h2-console  
spring.data.cassandra.keyspace-name=reactive
```

External Properties'e erişme

```
@Component
public class DenemeService {

    @Autowired
    private Environment environment;

    @Value("${spring.data.cassandra.keyspace-name}")
    private String keyspaceName;

    public void printProperties() {
        System.out.println(environment.getProperty("server.port"));
    }
}
```

