Putem configura un Container si fara fisier XML, doar cu Java code.

Putem configura un Spring Container prin 3 metode:

1. Folsind in totalitate un XML file
2. XML Component scan
3. Java configuration class

**Java Configuration class**

1. Cream o clasa si punem anotatia @Configuration
2. Adaugam component scanning suport, ca <context:component-scan.../>

Prin **@ComponentScan(“package”)** (optional)

1. Citim configuration class cu AnnotationConfigApplicationContext

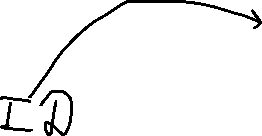
AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(SportConfig.class);

1. Obtinem beanurile ca deobicei

**Adaugarea beanurilor manual in classa config**

Putem adauga beanuri in clasa config exact ca in fisierul XML. Avem nevoie doar de anotatia @Bean. Totusi, nu vom crea obiecte ca membri! Cream mereu metode ce returneaza obiecte care sunt create cu new. Asa, obiectele create in clasa vor trece sub controlul Containerului Spring.

@Configuration  
public class SportConfig {  
  
 @Bean  
 public Coach swimCoach(){  
 SwimCoach mySwimCoach = new SwimCoach();  
 return mySwimCoach;  
 }  
}



**Numele metodei este ID al beanului!**

**Totusi, putem sa il modificam cu @Bean(“altId”);**

**Injectarea in Config Class**

Pentru a face injectia intr-un config class, cream o metoda bean, si in ea cand cream un obiect, pur si simplu i dam la constructor metoda care va returna obiectul de injectat.

@Configuration  
public class SportConfig {  
  
 @Bean  
 public FortuneService happyFortuneService(){  
 return new HappyFortuneService();  
 }  
 @Bean  
 public Coach swimCoach(){  
 SwimCoach mySwimCoach = new SwimCoach(happyFortuneService());  
 return mySwimCoach;  
 }  
}

**Cum functioneaza @Bean in Spring**

**Ca si in cazul la contextApplication.xml, spring creaza toate beanurile singleton, si in cazul clasei notata cu @Configuration, la rularea programului.**

Pentru prototype, Spring nu creaza nimic prin Container, ci doar returneaza noi beanuri, si nu pastreaza referinte la ele caci nu le inregitreaza in Contaiener.

Putem combina XML cu @AutoWired si alte anotatii liber.

**ADD Property files cu anotatii**

Pentru a adauga un property file la o clasa configuration, dupa anotatia @Configuration adaugam:

@PropertySource(“classpath:propertyFile”);

Apoi folosim @Value(“${foo.team}”)

* **Anotatiile ca @PropertySource pot accesa liber ce e si in webapp!Deci, intai cauta in resources si apoi in webapp, dar asta doar daca aplicatia e rulata de TomCat**

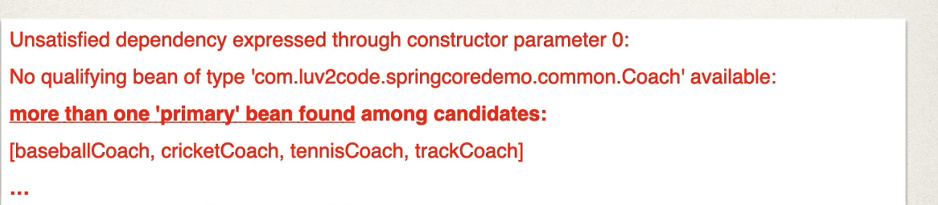
**@Primary**

* **@Qualifier** este pentru a specifica care e id la bean ce vrem sa il injectam
* Dar, putem folosi si @Primary. Aceasta anotatie se pune cu @Component, si spune ca daca sunt mai multe beanuri de un tip, se va alege mereu cel cu @Primary
* De ex, daca avem o interfata Animal, si 3 clase ce o exstind: Cat, Dog, Horse
* Acum, sa zicem ca avem o clasa Zoo, ce e @Component, si avem un @Autowired la un field de tip Animal
* In loc de Qualifier(), putem pune, de ex, la Cat

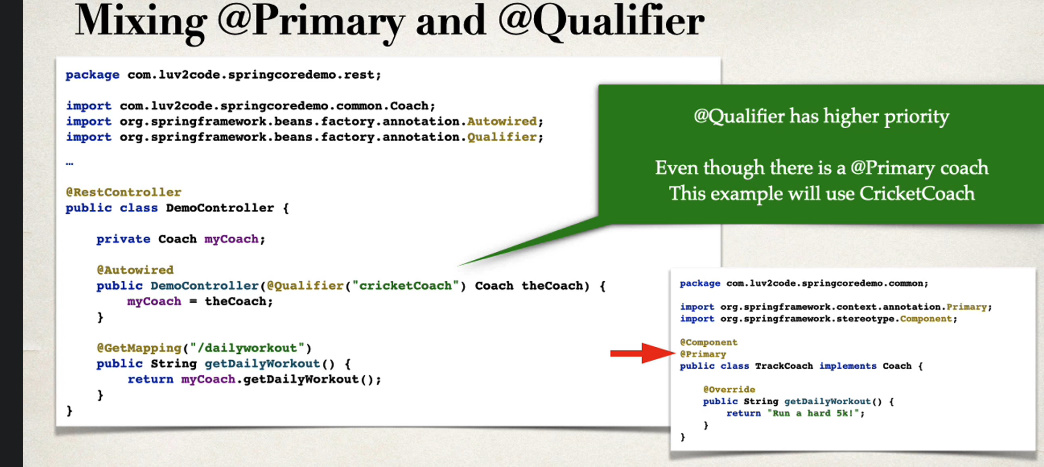
@Component

@Primary

* **Acum, se va lua mereu Cat cand apare asa conflict**
* Daca vom mai pune @Primary de ex la Dog, iar apare eroare



* **Putem si amesteca @Qualifier cu @Primary, dar nu e o idee prea buna**
* In asa caz, e logic ca @Qualifier are mereu prioritate

****

**@Qualifier e mai recomandat de folosit**

**Lazy Initialization**

* Toate beanurile sunt create in container in mod eager
* Putem sa setam un bean ca lazy, si atunci el va fi creat in urmatoarele cazuri:

1. este necesar pentru a face vreo injectie
2. il cerem noi

* Un bean e creat lazy prin @Lazy, si fie il punem la clasa, fie la metoda cu @Bean
* Putem seta o proprietate ca sa facem toate beanurile lazy:

spring.main.lazy-initialization=true

