**---------------------------------------------------------------------------**

**XML CONFIGURATION**

ClassPathXmlApplicationContext

***>> PROPERTIES***

<**context:property-placeholder location="classpath:sport.properties"**/>

***>> BEAN DEFINITION***

<**bean id="myCoach" class="com.jezh.springdemoone.entityImpl.BaseballCoach**

**init-method="init" destroy-method="destroy"**>

<**constructor-arg ref="myFortuneService"**/>

<**property name="emailAddress" value="emailAddress is**

***>> property injection from file***

**${cricketCoach.email}"**/>

<**property name="fortuneService" ref="randomFortuneService"**/>

</**bean**>

***>> NAME - ALIAS***

<**bean name="tracker, trackCo" class="com.jezh.springdemoone... "**/>  
<**alias name="trackCo" alias="trackChief"**/>

**---------------------------------------------------------------------------**

**ANNOTATION CONFIGURATION**

ClassPathXmlApplicationContext

<context:annotation-config/> only works on beans registered within the application context… because has no "targets" to work on. ----- Т.е. так все равно придется создавать бины в XML, но вот автовайрить их уже можно будет через @Autowired

<**context:component-scan base-package="com.jezh..., scopeTest"**/>

<**context:property-placeholder  
 location="classpath:testEntityOneProps.properties,  
 classpath:something.properties,...**

**ignore-unresolvable="true"**/>

**---------------------------------------------------------------------------**

**JAVA CONFIGURATION**

AnnotationConfigApplicationContext

-------------------------------------

Перед классом:

@Component*/\*("swimCoach")\*/*  
@Scope(**"singleton"**)

-------------------------------------

Перед классом:

@Configuration

@ComponentScan(**"com.jezh.spring\_demo\_two\_annotations, scopeTest"**)

@PropertySources({@PropertySource(**"classpath:testEntityOneProps.properties"**),  
 @PropertySource(**"classpath:something.properties"**),

.................  
 @PropertySource(**"classpath:"**)})

-----

Spring 4.2 and lower – to return Value from .properties file:

В классе @Configuration добавить следующий код:

*// add support to resolve ${...} properties*

*//* ***todo: NB: STATIC bean***

**@Bean  
    public static PropertySourcesPlaceholderConfigurer  
                    propertySourcesPlaceHolderConfigurer() {  
          
        return new PropertySourcesPlaceholderConfigurer();  
    }**

@Bean*/\*("swimCoach")\*/***public** Coach swimCoach() {} *// либо даем название методу такое же, как имя бина, либо устанавливаем value в @Bean*

**@Bean** в классе, помеченном **@Configuration** – это замена **@Component** перед классом

-------------------------------------

@Value(**"${testEntityOne.email}"**) **private** String **email**; *// работает, поскольку перед конфигурационным классом @Configuration есть аннотация @PropertySource с указанием расположения пропертей, и больше уже нигде это указывать не нужно*

-------------------------------------

***>> НАПОЛНЕНИЕ КОЛЛЕКЦИИ ИЗ ПРОПЕРТЕЙ:*** *// https://stackoverflow.com/questions/12576156/reading-a-list-from-properties-file-and-load-with-spring-annotation-value  
 //****todo: NB: работает только с чистым типом - с List, а не ArrayList!!!!*** *//* ***todo: Using Spring EL (Spring Data Elasticsearch ):*** @Value(**"#{'${randomFortunes.list}'.split(',')}"**)  
 **private** List<String> **fortunes**;

***>> АВТОВАЙРИТЬ КОЛЛЕКЦИЮ:***

*//* ***todo: NB: когда автовайрим массив или коллекцию, она автоматически наполняется экземплярами ВСЕХ компонентов (бинов)******указанного типа***@Autowired  
**private** FortuneService[] **fortunesToChooseFrom**;