**• Annotation-Based Configuration**

• @Autowired

**• Defining Beans Outside of XML**

• @Component and ComponentScan

• Qualifying Beans: @Primary,

@Qualifier & Custom Qualifiers

• @Scope

• Value Injection using @Value

• Life-Cycle Events

• @Order & @DependsOn

**• XML vs. Annotation Configuration**

• Configuring all of the beans and their dependencies using constructor and

property arguments in XML is cumbersome and error-prone.

• So Spring developed along the way easier ways to do the same thing.

• Spring provides annotations for the specification of both

• beans and

• dependencies.

• We’ll start with dependency configuration through annotations and later

proceed to bean annotations.

**Autowiring**

• Spring container can figure out and satisfy dependencies between beans using annotations since version 2.5.

• Spring can resolve collaborators automatically by inspecting the contents of the **ApplicationContext.**

• This is called autowiring.

• Autowiring allows cleaner DI management.

**DI’ları daha basit hale getiren, XML’den kurtaran ve belli annotationları kullanarak iliskileri daha rahat betimlemizi sağlar.**

Bu annotationları kullanarak **ApplicationContext** icinden beanleri sonra onların DI’larını bulup halledecek sekilde spring ayağa kalkar.

• To have annotation-based configuration **<context:annotationconfig/>**

element in **<beans/>** element must exist in XML file.

• Context schema is available at **http://www.springframework.org/schema/**

**context/spring-context-4.3.xsd**

• Eventually **Spring** will let us to get rid of all XML files and depend on

annotations only

Bu xml icerisinde beans elementi icinde context semasında var olan annotation configi kullanmak lazım. (Yukarıda schema’nın linki verilmistir.)

**Packages And Annotations**

• Spring has its annotations for DI mainly in following two packages:

**• org.springframework.beans.factory.annotation has**

**Autowired, Configurable, Lookup, Qualifier, Required, and**

**Value.**

• **org.springframework.context.annotation has mainly Bean,**

**ComponentScan, Conditional, Configuration, DependsOn,**

**Import, Lazy, Primary, PropertySource(s), and Scope.**

**@Autowired**

• **org.springframework.beans.factory.annotation.Autowired**

annotation is used to specify dependencies in Java source code.

• **Autowired** annotation became available with **Spring** 2.5.

• **<context:annotation-config/>** element in **<beans/>** is needed.

• **Spring** figures out dependencies through **Autowired** annotations.

• But beans must still be defined in XML configuration without any info on their dependencies.

Burada beanlerin inject durumu hala XML icerisindedir, Ama DI’lar değildir. Burada beanler XML icinde olmalı ama dependencies’lere gerek yoktur. Cünkü onu Autowired ile hallediyoruz.

• **Autowired** annotation is applied to following places and put before:

• instance variable

• constructor

• setter method

• any configuration method with any number of parameters

• Of course any element that is annotated by **Autowired** should be

injectable with a bean defined in **<bean/> in** the XML file.

**Autowired** instance variable, constructor, setter method, herhangi bir methoda uygulanır.

Xml file’da var olan beanlerdan bir tanesini inject edecek sekilde bu yapılar yazılmıs olmalı.

• **Autowired** annotation has only one attribute of type **boolean**, **required** which is **true** in default.

• So **Spring** tries to inject dependencies into every single point that is annotated by **Autowired**.

• If it can not satisfy any required dependency it throws

**org.springframework.beans.factory.UnsatisfiedDependencyException.**

• Giving **required** attribute false makes the dependency optional which may cause **NullPointerException**.

• In a class only one constructor can be annotated with **Autowired**.

• A class can have many other constructors without annotation but Spring

always try to call the one with annotation to satisfy all dependencies.

• If there is only one constructor in a bean with an injectable dependency

there is no need even to use **@Autowired** for that constructor.

• **Spring** automatically wires the dependencies by passing it to the

constructor of the bean.

• This feature became available with 4.3.

• If the dependency is for a value instead of a bean then values must be

specified for injection in XML file using **value** attribute of **</bean>.**

• In this case **constructor-arg** and **property** attributes are used

only for values, there is still no need to use **ref** for beans because they

are defined in XML and **Spring** automatically finds and autowires them.



• **Spring** injects into any method that accepts a parameter of type of the

collaborator bean.

• **Spring** calls **them config methods**.

• Property setter methods are special case of config methods due to their

proper names.

• **Spring** does not put forward any other rule on the config methods.

• Config methods can return values for example.

• Beans can be excluded from autowiring by setting the **autowire** **candidate**

attribute of **the <bean/>** element to false.

• The IoC container makes that specific bean definition unavailable to the

autowiring infrastructure including annotation style configurations such

**as @Autowired.**

• **If @Autowired**. is used on a method that doesn’t receive any

dependency IoC raises a log at run-time and says INFO: Autowired

annotation should only be used on methods with parameters:…

**Inject anlamında constructor injection bu anlamda en iyisi onu kullanmak gerekir. Çünkü unit testlerde daha rahat kullanabiliriz. Dolayısıyla immutable de yapabiliriz objelerimizi. Hem zorunluluk sağlıyor.**

Setter methodlarda immutable olmuyor, nesne run time da degisebilir.