**Specifying Dependencies**

• There are mainly two ways to define dependencies in the XML

configuration file:

• Explicit definition through constructor and property injection

• Implicit definition through **autowire** attribute of <**bean**/> element

**Constructor & Property Injection**

• In dependency injection all collaborators are injected into the bean on which it depends on.

• In XML-based configuration there are mainly two types of explicit injection:

• **Constructor injection**

• **Property or setter injection**

• Both ways use either **ref** attribute to refer to another bean or **value**

attribute for a primitive value or a **String**.

XML bazlı configurasyonda temelde iki tip injection vardır.

• **Constructor injection**

• **Property or setter injection**

Eğer depend olunan sey bir bean ise **ref** attribute kullanırlar değilse Bir primitive değer ya da string geçeceksek bu durumda **value** attribute kullanırlar..

• For constructor injection **<constructor-arg/>** and for setter (or

property) injection **<property/>** is used.

• In case of injecting a bean using **ref** attribute either **id** or one of the

**names** or **aliases** of the collaborator bean can be used inside

**<constructor-arg/>** **or <property/>** element.

**• A nested <ref bean='...'/> can also be used.**

• To inject a literal value such as a primitive or a String **value** attribute is

used inside **<constructor-arg/> or <property/>** element.

• To inject a **null** value for beans **<null/>** element is used in

**<constructor-arg/> and <property/>** elements.

Constructor inject icin **<constructor-arg/>** ya da setter(or property) icin **<property/>** elementi kullanılır.

**Ref** kullanacaksak ya **id’si** ya da **name** veya **aliaslarından** birisi **refe** **değer** olarak geçilmesi lazım.

**Name ile alias aynı sey.**

**Constructor Injection**

• For constructor injection **<constructor-arg/>** is used.

• Collaborator bean is referred by the **ref** element inside **<constructorarg/>** which receives the **id, name** or **alias** of the bean.

**• Or nested <ref bean='...'/> is used.**

• The bean must have a constructor that receives an instance of the collaborator bean as an argument otherwise **Spring** throws

**org.springframework.beans.factory.UnsatisfiedDependency Exception.**

**<constructor-arg/>** kullanılmalıdır.

**Ref** elemanı ile **id,name ya da alias ile constructor-arg’a** gecilir.

Resolution etmesi ile alakalı aynı tipte birden fazla injectleri **type,index,name attribute** vererek gecebiliriz.

**Primitive’ler** icin ya da **stringler** icin **value** attribute kullanılır.

• Resolution of the constructor argument occurs basically by using the

argument’s type.

• In case of more than one argument **index**, **name** and **type** attributes

can be used to help to resolve dependencies:

• **index** shows the index of the argument and takes an integer starting from

0

• **name** takes a string as the name of the parameter the constructor receives

• **type** takes a string as the type of the parameter the constructor receives

Using **index** would be the simplest way to help **Spring** IoC in resolving the arguments.

**Property Injection**

• For property injection <**property**/> is used.

• name attribute of the <**property**/> shows the name of the property.

• Collaborator bean is referred by the **ref** element inside <**property**/>

which receives the **id**, **name** or **alias** of the bean.

• Or nested **<ref bean='...'/> is** used.

• Resolution of the argument occurs basically by JavaBeans naming

convention in the setter method.

Constructor ile aynı durumlar gecerli. Sadece <**property**/> olarak kullanıyoruz.

• The **constructor** injection over the property injection can be preferred

when the bean should be **immutable**.

• On the other hand constructor **injection** should be used for **mandatory**

**dependencies** and to avoid **NullPointerException** exceptions

Constructor injection tercih etme sebebi beanlerin immutable olmasının sebebi. Immutable önemli ise constructor tercih etmemiz lazım.

Zorunlu olan fiedlar varsa inject yine constructor olması lazım.

**Inner Bean**

**•** Sometimes a dependency is defined directly using **<bean/>** tag

without an **id** or **name** in which case the bean is called **inner bean.**

• In this case the class attribute of **the <bean/>** element inside the

**<property/> or <constructor-arg/>** elements takes as value the

class of the bean.

**Specifying Dependencies Autowire**

• **autowire** attribute of the **<bean/>** element allows **Spring IoC**

container to find and satisfy dependencies automatically.

• Depended beans must still be defined in the XML file.

• This way provides cleaner XML configuration files because

**<constructor-arg>** and **<property>** tags are not needed.

• All dependencies are resolved through the conventions specified by

**autowire** attribute value.

Spring IOC containerinin dependencyleri otomatik olarak bulup yerine getirmesini sağlar.

**Ama depended olunan kendisine bağımlı olunan beanler XML file icinde belirtilmeli.**

• **autowire** attribute can have one of three values:

• **constructor**: Allows to inject into constructor.

• **byName**: Allows to inject into a property resolving by name of a setter

method.

• **byType**: Allows to inject into a property resolving by type of the argument

that is received by a method.

• **byName** and **byType** are property injections but the first one looks for a

proper setter method while the second one looks for any method with a

proper type.

Autowire’n üc değeri vardır. **Constructor, byName, byType**

**Default değeri “no” dur.**

Constructor constructor ile inject eder.

byName isim ile inject eder.

byType constructorına name ile inject eder.

• Using autowire attribute has some limitations:

• Primitives and String objects can not be injected,

• Autowiring may create ambuguities.

• So explicitly specifying dependencies in the XML file would avoid these problems.

Primitive ve stringler böyle inject edilemez.

**Autowire-candidate**

• Beans can be excluded from autowiring by setting the autowirecandidate

attribute of the <bean/> element to false in XML

configuration file.

• The container doesn’t makes that specific bean definition available to the autowiring

• But note that autowire-candidate attribute only affects type-based autowiring.

• It doesn’t affect explicit references by name, which get resolved even if

the specified bean is not marked as an autowire candidate.

• So autowiring by name nevertheless injects a bean if the name matches.

* Injectionlarda istersek bean yerine null gecebiliriz, NullPointerException fırlatabilir bize. <null> elementi ile.

**Value Injection**

• Values like beans can be injected into constructors and setter methods.

• For this purpose instead of **ref** attribute **or <ref bean=“…”>** element

to refer to a bean, **value** attribute is used in **<constructor-arg/>**

**or </property>** element.

• A **nested <value>…<value/>** can also be used.

• value attribute is used to pass primitives, **String** literals and

collections (array, list, set and map) of those types.

• All type conversions are made automatically by **Spring**.

• Injecting collections will be handled soon.

Refe doğrudan bean’in adını geciyoruz, veya id,name, aliasda olur.

Value injectionda ise ref yerine value kullanırız. Temelde primitive, String literals ve collections icin kullanılır.

Property olarak doğrudan value olarak gecebiliyoruz.

• Of course proper constructor and setter methods must be provided

otherwise **Spring** throws

**org.springframework.beans.factory.UnsatisfiedDependency**

**Exception.**



Burada best-practics index kullanmak.

**Collection Injection**

• Collections of beans or values can also be injected into constructors

and setter methods.

• As a collection **array, List, Set and Map** implementations can be used.

**• To specify collections in XML file </array>, </list>, </set>, and </map> are used.**

• In case of values all type conversions are made automatically by Spring.

• While array, **List, Set** takes values or references to beans Map

receives key-value pairs which is called entry.

• **So </array>, </list>, </set> elements use nested <value>… <value/> for values or nested <ref bean=‘...'/> for beans.**

**• </map> uses </entry> element to pass key-value pairs.**

**Array, list, set’in icerisinde value kullanırız..**

**Value icin ref bean kullanırız beanlere referans vermek icin.**

**Map icerisinde dogrudan value kullanmayız, cünkü key-value deger tutuyor. Onun icin de entry kullanmamız gerekiyor. Asagıda belirtildigi gibi.**

• **</entry>** element in **</map>** has several attributes:

• **key**: Used for key of the entry if it is value

• **key-ref:** Used for key or value of entry if it is another bean

• **value**: Used for value of entry if it is value

• **value-ref**: Used for key or value of entry if it is another bean

• Nested **<ref bean=‘...'/>** can also be used for beans.



• For bean references **<null/>** can be used

• In **</entry>** element **of </map>** key can be given null value using **<key> <null/> </key>** and value can be given null value using only <**null/>**

• Don’t forget **</entry>** element doesn’t take two **<null/>** elements such as **<entry> <null/>** **<null/></entry>**. for both key and value.

• Use instead: **<entry> <key> <null/> </key> <null/></entry>**

Map’in icerisinde entry elemanında key’e null verebiliriz.

Value’ya null vermek icin ayrı bir sekilde yazmalıyız.

**UYGULAMA**

**• org.javaturk.spring.di.ch03.ex.calculator.collection**

**.Test**

**• Create an XML configuration file for the Test.**

**Test kısmının calısabilecek düzeyde mapinjection sekidle configurasyon edilmesi lazım. Ödev.**

**Singleton bir classtan**