**Spring Core – Spring Boot – Micro Services**

**By Mr. Raghu**

**Spring Framework introduction:**

**Spring framework** is a RAD framework for Java based enterprise application.

**RAD**: Rapid Application Development.(Faster & less lines of code)

**Framework** 🡪 **Technologies** + **Design Patterns** (Solution to common problems with high performance)

**Case-1:** Web Application 🡨 Technologies (Servlet/JSP), Database(JDBC), dependencies, container/server…etc

🡨 3 months and 10k lines of code

🡨 Less performance

**Case-2:** Web Application 🡨 Framework

🡨 1 month and 3k line of code

🡨 High performance

**Spring Core:** In simple, it provides all rules and guidelines to work with Spring Framework and Spring Boot based applications.

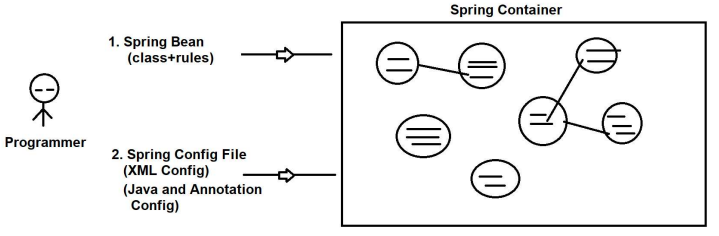
**Spring Container:** It is a Pre-defined Application that does

1. Scan/Find classes. Called Component Scanning
2. Create Object to detected class.
3. Provide data to variables and link objects (HAS-A) called Dependency Injection
4. Finally destroy the objects (When we stop Application/Server)

**Programmer has to provide two inputs to Spring Container:**

1. **Spring Bean**: class + Rules given by Spring container
2. **Spring Configuration**: (objName and link details)

**XML Config** | **Java Config** | \*\*\***Annotation Config**



**Spring Bean:**  is a class that follows the below rules given by spring framework.

* Package statement must be given.
* Class must be public with any name
* private variable (if exist/optional)
* Provide zero-param constructor **and/or** N-params constructor [sometimes both are used]
* Provide Mutators (setters, getters) 1 variable = 1 set / 1 get methods
* We can override 3 methods from object class mainly. **toString()**, **hashCode() +** **equals()** methods which are

[non-private, non-static and non-final]

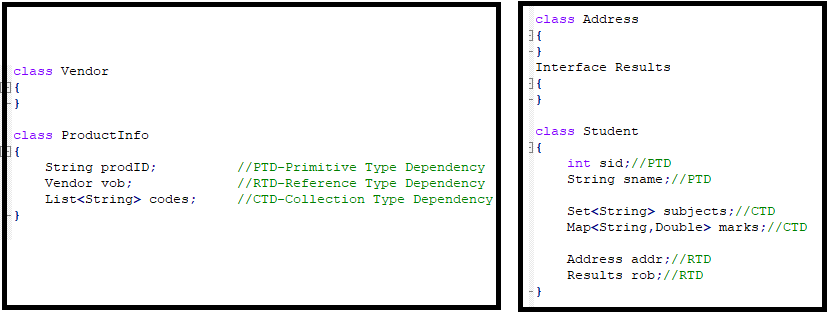
* Inheritance Rule: One Spring bean can extend another Spring f/w or our spring classes, but not other classes. Only Serializable(I) is allowed.
* Annotations: Only Spring Annotations allowed (some integrations we use :spring framework, integration such as Lombok, JPA[Hibernate], AOP ...etc)

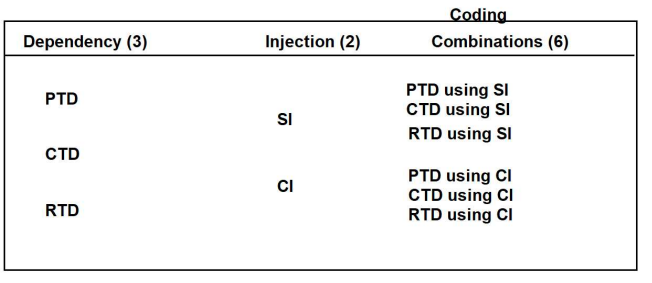
|  |  |
| --- | --- |
| **Theory** | **Programming** |
| OOPs | Core Java |
| ORM | JPA/Hibernate |
| DI | SpringF/W( Container/IoC) |

**Dependency Injection [Theory]**

**Dependency / Dependencies** (3): A variable created inside class (Spring Bean) is called Dependency.

* **Primitive Type Dependency(PTD)[8+1] :** If a variable is created using the following data types such as byte, short, int, long, float, double, boolean, char + String [Even Wrapper classes]
* **Collection Type Dependency (CTD)[4] :** If a variable is created using : List, Set, Map or Properties(C) [all are from java.util package]
* **Reference Type Dependency(RTD)[no count] :**If a variable is created using class or interface (another Spring bean) ( HAS-A Relation)

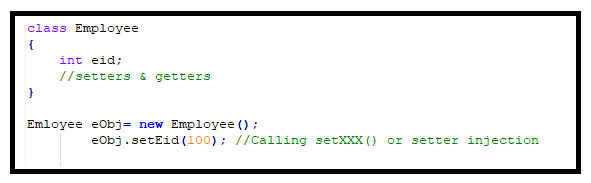




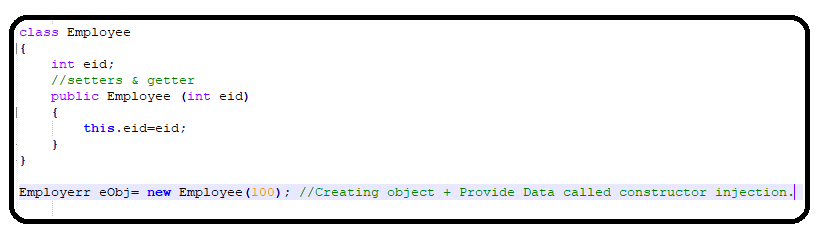
**Injection (4)/ (3):** Provide data to variable after creating object.

1. Setter Injection (SI)
2. Constructor Injection (CI)
3. Lookup Method Injection (LMI)
4. Interface Injection (not exist in Spring/Boot)

**1. Setter Injection (SI):** Provide data to variable after creating object by using setters.



**2. Constructor Injection (CI):**  Provide data to variable using PARAMETERIZED CONSTRUCTOR.

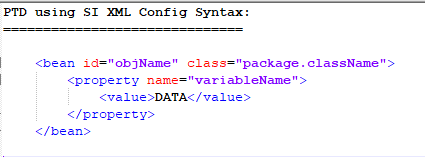


**3. Lookup Method Injection (LMI):**  It depends on spring bean scopes. If parent bean is singleton and child bean is prototype then we use this injection.

**4. Interface Injection:**  (not exist in Spring/Boot). Creating object to interface and link to variable.

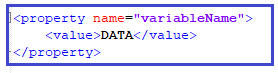
**Sring XML Configuration using DI Theory**

|  |  |
| --- | --- |
| **<bean>** | Container creates object |
| **<property>** | Calling set method(Setter Injection) |
| **<constructor-arg>** | Calling parameterized constructor(Constructor Injection) |
| **<value>** | Primitive type dependency(PTD) |
| **<list> , <set>, <map> , <props>** | Collection type dependency(CTD) |
| **<ref/>** | Reference type dependency(RTD) |

****

**We can write XML Config for Primitive in 3 different ways:**

**1. Value as Tag:**

****

**2. Value as Attribute**

****

**3. p:schema or p:namespace 🡪\*** Here, p is short format of property tag (setter)

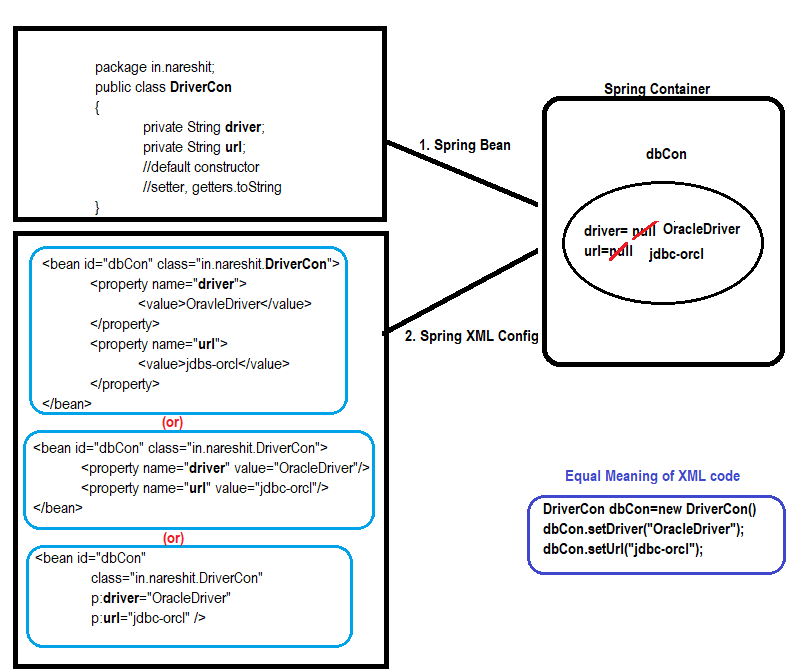
****

* If any “**p:**variableName” unidentified error then need to add the below schemaLocation in xml config file.

xmlns:c=*"http://www.springframework.org/schema/c"*

xmlns:p=*"http://www.springframework.org/schema/p"*>

**Example#1**

****

**Spring Container Types :**

**1. Old Container**

(BeanFactory) [I]

**XmlBeanFactory** (C) [It supports only XML Configuration]

**2. New Container**

(ApplicationContext) [I]

**ClasPathXmlApplicationContext**(C)

**AnnotationConfigApplicationContext**(C)

**ServletWebServerApplicationContext**(C)

..etc

* [It supports XML, Java and Annotation Configuration]

**New Container:**

**ApplicationContext(I)**

**IS-A**

**ClassPathXmlApplicationContext(C)**

ClassPath = src/main/resources folder

Xml = XML File

ApplicationContext = Spring Container

ClassPathXmlApplicationContext = Create Spring Container, by reading XML file which exist in location src/main/resources folder

**getBean("objName")**: Object This method is used to read one object from container.

**Setup:**

1. Download and install JDK : 8/11/Latest
2. Download and install STS(Spring tool suite)
3. Basics of Maven(Learn)
4. Basics of Lombok(Learn)

**ScApp1\_XMLConfig\_SI Project (Ref)**

1. Create maven project
2. Add compiler plugin, spring-context in pom.xml(Search from web)
3. Create Spring bean
4. Create SpringConfig.xml file & SpringConfig2.xml update accordingly. (Search spring xml configuration from web)
5. Create Test class: to check bean is created by Spring Container or not?

ApplicationContext ac = new ClassPathXmlApplicationContext("SpringConfig.xml","SpringConfig2.xml");

**Spring Configuration- Annotation**

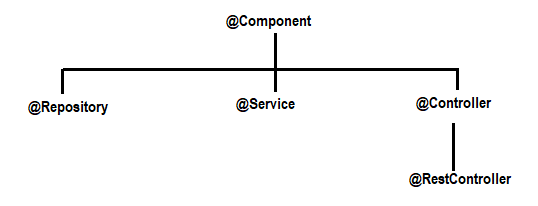
**Annotation Configuration:**

* Annotation configurationis fast execution comparing to xml/ java based configuration.
* It applicable for only programmer defined spring classes and should apply in source files(\*.java).

**Types of Annotation Configuration:**

**1. Stereo Type Annotations:** This type of annotations is used for object creation.

1. **@Component :** Creating object to our class
2. **@Repository :** Creating objects to our class + Database operation
3. **@Service :** Creating objects to our class + Transaction management, logics, calculation, sorting…. etc
4. **@Controller :** Creating objects to our class + Web Application – MVC (http request)
5. **@RestController :** Creating objects to our class + Web Services (http) + Data conversions(Json,xml)

****

**2. Basic /Support Annotation:** These are used to provide data and link other objects.

1. **@Value :** Called as Field Injection, no need of setters/constructor)

Provides default/hardcoded values**|** reads data from properties files**|** Provides SpEL data to variables in spring bean.

1. **@Autowired**, **@Qualifier**, **@Primary**
2. **@PropertySource**
3. **@ComponentScan:**  It is used to provide package[s] information to spring container for finding/scanning spring beans.

Here, Spring Container scans beans in specified package and its all sub packages.

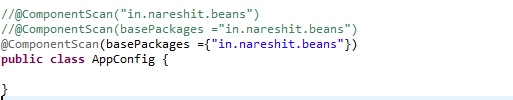
1. **@Scope**

..etc

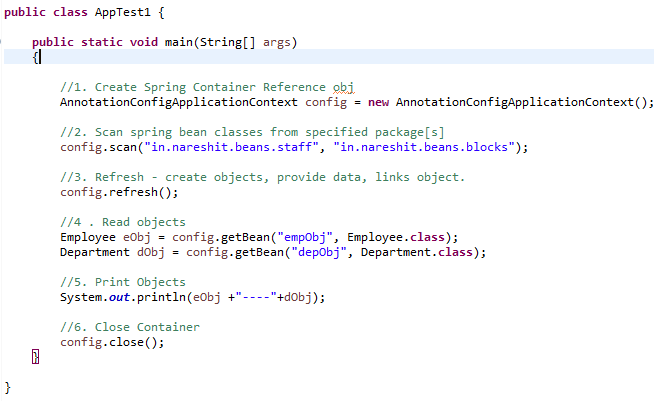
* Spring Boot has not given support for XML Config (Zero XML Config)
* In Spring Boot, we use mostly Annotation Config (99%) and Java Config (1%).

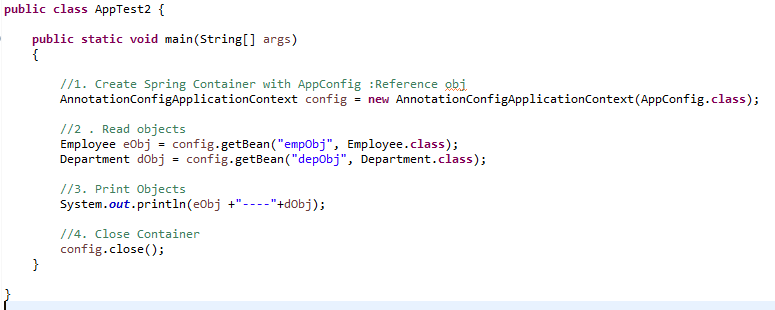
**ScApp2\_AnnotationConfig Project (Ref)**

1. Create maven project
2. Add compiler plugin, spring-context dependency in pom.xml(Search from web)
3. Create Spring beans with @Component and @Value annotations inside.
4. Create AppConfig call to provide basePackage information



1. Create Test classes





**basePackage:** This is one package name, given to Spring container find/scan classes from given package and subpackages.

**Setup Data:** Data comes at runtime from end-user (Form data such as login form, register form….)

Even developer/testing team provide some data before running/starting application is called setup data.

**Ex:** Database connection, connection pooling, security configuration, Web MVC prefix/suffix, Email properties, Logging properties, ORM properties…. Etc.

Such data is given form

1. Properties files
2. YAML files
3. VM/System Args
4. Cmd line Args

**Properties file:**

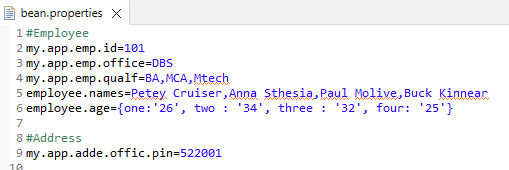
* <anyFileName>**.properies** and it should be created in **src/main/resources** [classpath]
* Data must be **key=value** format. Key can have dot (.), dash(-) , underscore(\_)

**Ex:** my.app-driver\_class=ABCD

* **Keys** are case-sensitive & **Value** is string type by default , auto-parsing supported.
* **‘#’** indicates comment.

**Working with Properties file(steps):**

1. Create properties files and have data in key=value format.

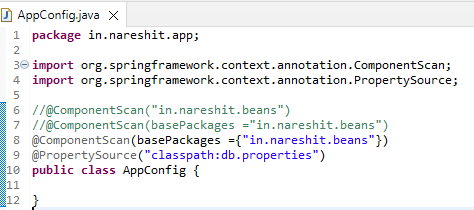


1. Load properties files in Spring Container by **@PropertySource**(“**classpath:**abcd.properties”)

Container loads properties file and created object to **Environment (I)** 🡪 **StandardEnvironment(C)**

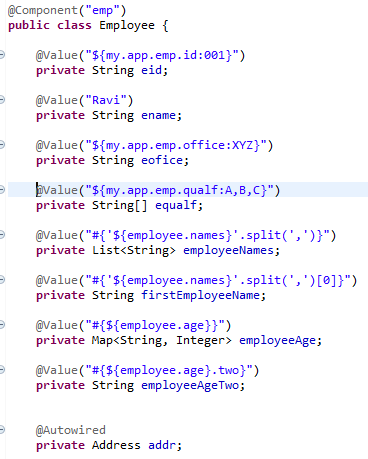
Stores all key=values

Note: word **classpath:** is optional for in spring non-web applications.



1. Read data from properties files.

**@Value( “ ${keyName} ”)**

****

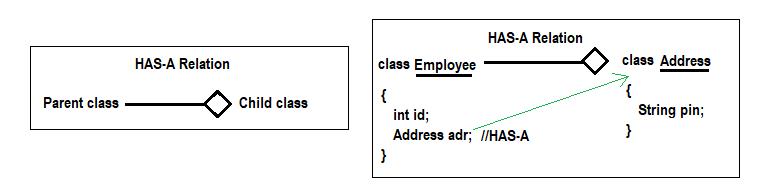
**Association (HAS-A) Relation/Mapping:** Using one class(child) object(object reference) inside another class(parent) is called as HAS-A relation.

Using child as a Data type inside parent and creating one variable is called as Association/HAS-A relation.

* **Wiring :** creating link between two objects which are in spring container.
* **Autowired:** Link between two objects is created by spring container.
* **@Autowired** annotation is used on top of HAS-A relation.
* If **@Autowired** annotation is not applied on top HAS-A relation then the link will not be created and holds the default value ‘null’.
* If **@Autowired** is applied on HAS-A variable but child object is not created(i.e no **@Component** on child class) then the spring container throws exception ‘NoSuchBeanDefinitionException…..’, Parent object will not be created. Application itself stops.
* **@Autowired(required=false):** If object is there link it else set ‘null.

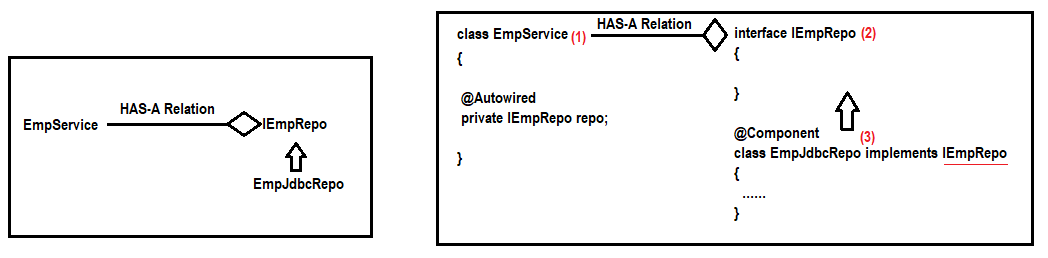
By default @Autowired internally works with ‘required=true’. It means child object must be provided, else Exception.

**Ref: ScApp3\_Autowired (project)**

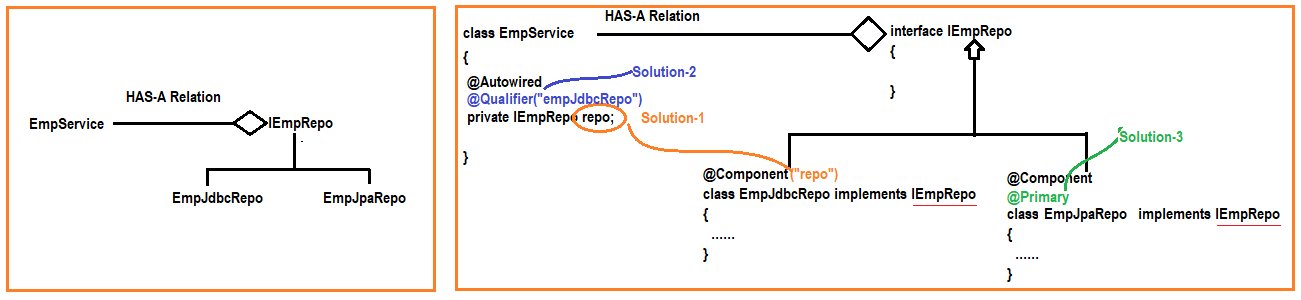


**Autowired using child interface:** If child is an interface in HAS-A relation then Autowired will search for implemented class object for child interface. Same injected into parent object.

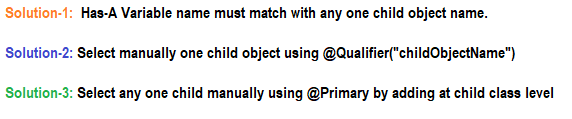
* We must not write @Component on top of interface. It will give runtime error Object can not create for interface.
* If Child interface has not implemented class or no @Component on top of child implemented class level then Spring Container throws **NoSuchBeanDefinitaionException**.



**Child Interface having multiple implementations class and objects:**



If child interface is having multiple Implementation classes and their objects are created Spring Container throws **NoUniqueBeanDefinitaionException.**

****

* Use **@Qualifier** annotation at HAS-A variable level to select one child object in-case of multiple found. [ Use this if child are predefined we have jar files or .class files. i.e no .java file]
* Use **@Primary** annotation at any one child Implementation class level, that says if multiple objects are exist then choose current class only.
* **@Qualifier** compares given objectname with every object in container(bit slow), **@Primary** says to Autowire “Do not search any other, Link it this”.

**Q) Can we use @Value for Object linking in Spring F/W?**

**A)** Yes. @Value(“#{objectName}”) on top of HAS-A variable

**Q) What is the difference between @Autowired and @Value?**

**ScApp3\_Autowired Project (Ref)**

**ScApp3a\_Autowired\_childIntrface Project (Ref)**

**ScApp3b\_Autowired\_multi\_Child Project (Ref)**

**Spring Bean – Lifecycle Method**

**Life Cycle Methods:** A method is called by container in between object creation and destroys.

=> In Servlet API 3 Life Cycle methods exist:

**init()** : Can be called only once after creating object.

**service()** : Can be called for every request

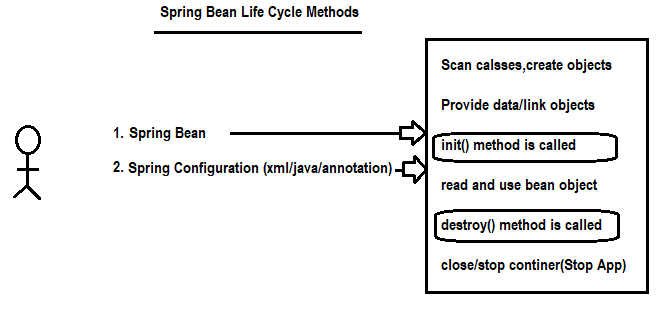
**destroy()** : Can be called only once before deleting the object by container

=> In Spring container 2 Life Cycle methods exist: (Optional to use)

init ()

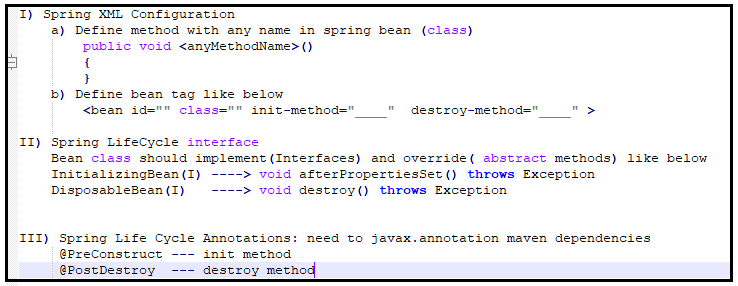
destroy()

\* destroy method will not destroy the object. It’s logic should be closes connection/files/streams.. etc.



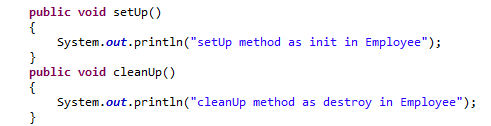
Life Cycle Method Coding:

1. Spring XML Configuration
2. \*\*\* Spring Life Cycle Interfaces
3. Spring Life Cycle Annotation(JEE)

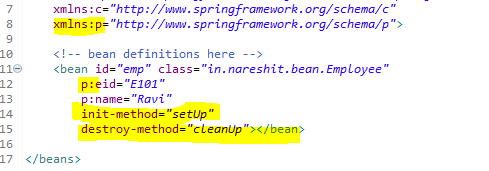


**ScApp4a\_LifeCycle\_XML Project (Ref)**

1. Create Maven project and add javasource compiler plugin, spring-contect dependencies from Web.
2. Create Spring Bean with two public void methods with any name(setup, cleanUp).



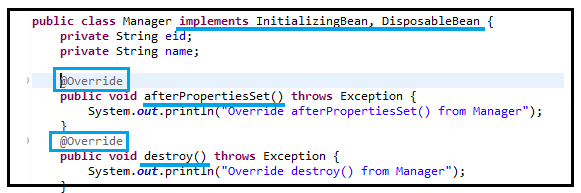
1. Create Spring XML config file and copy the structure from web and add **init-method, destroy-method** properties for bean tag



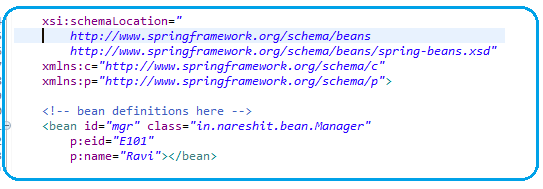
1. Create Test class then check the bean lifecycle methods flow.

**ScApp4b\_LifeCycle\_Interface Project (Ref)**

1. Create Maven project and add javasource compiler plugin, spring-contect dependencies from Web.
2. Create Spring Bean by implementing InitializingBean, DisposableBean interfaces.



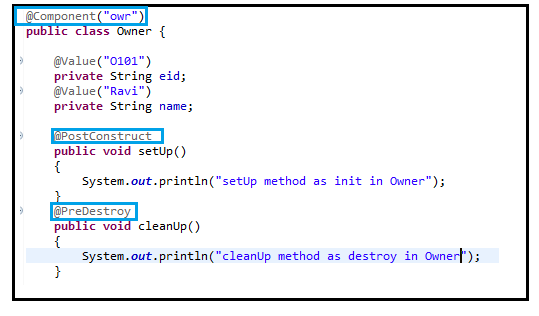
1. Create Spring XML config file and copy the structure from web and add bean tag



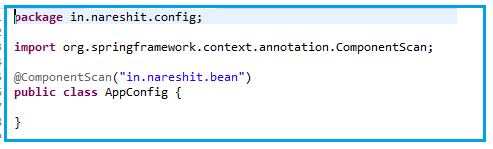
1. Create Test class then check the bean lifecycle methods flow.

**ScApp4c\_LifeCycle\_Annotation Project (Ref)**

1. Create Maven project and add javasource compiler plugin, spring-contect, **javax.annottion** dependencies from Web.
2. Create Spring Bean as follow with annotation based config.



1. Create Spring annotation based config to scan components



1. Create Test class then check the bean lifecycle methods flow.

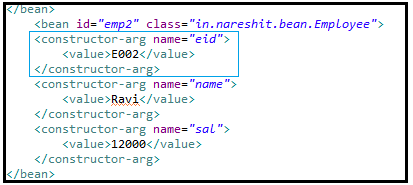
**Spring XML Config using Constructor Injection**

When we use <bean>, <properties> tags then Spring container uses default constructor to create object to class then provides data by using setter methods.

**Constructor Injection:** If spring container uses param-contructor to create object to class then it is called CI.

* 1 <constructor-arg> tag 🡪 1 parameter in constructor
* If class has 3 variables/3 setters, we can call them in any order (or) we can set even few values (need not to set all). But not in constructors, must pass all params in given order.
* **In Spring Boot, We use Autowiring concept with runtime data(properties & @value) there is no constructor injection in Spring Boot.**

**ScApp5\_XMLConfig\_CI Project (Ref)**

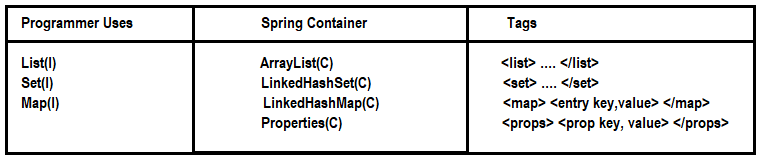
****

**Dependency – Collction Type(LIST,SET,MAP & Properties)**

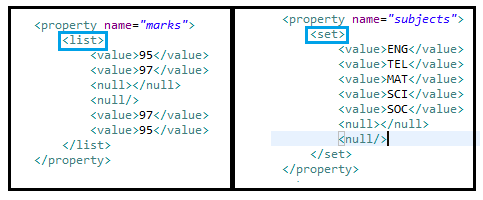
Dependencies:

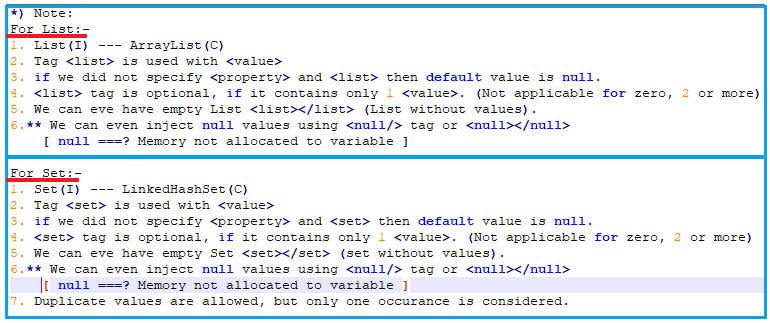
1. Primitive Dependency (8+1): byte, short, int, long, float, double, char, Boolean & String…
2. Collection Type Dependency(4): List(I), Set(I), Map(I) & Properties(C)
3. Reference Type Dependency:

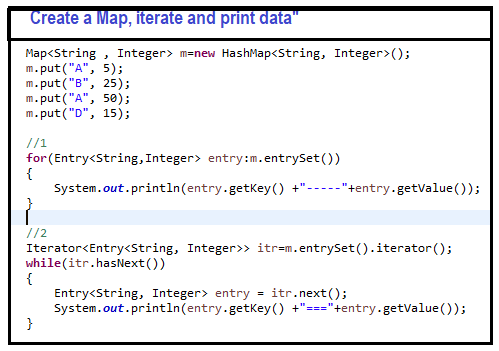
* Java/Spring says WORK WITH INTERFACES, I’LL CHOOSE IMPL CLASSES. Tomorrow if better IMPL class comes your code remains same.

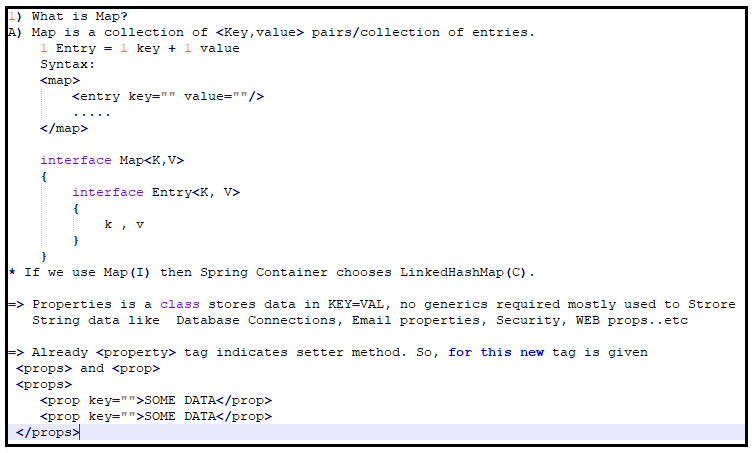


XML Configuration for List and Set:

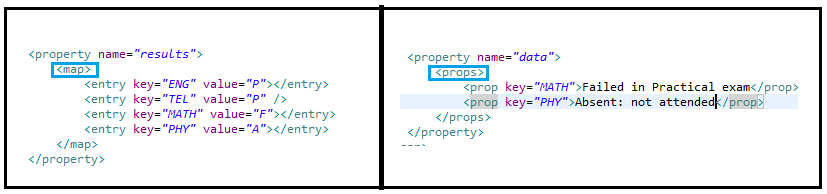


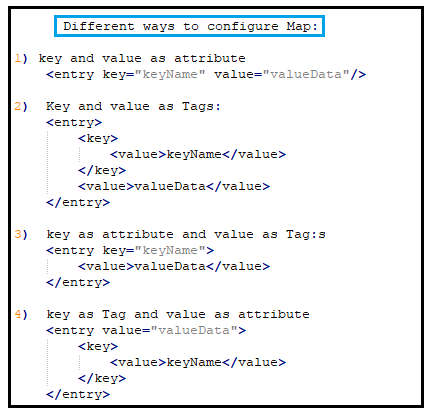






XML Configuration for Map and Properties:







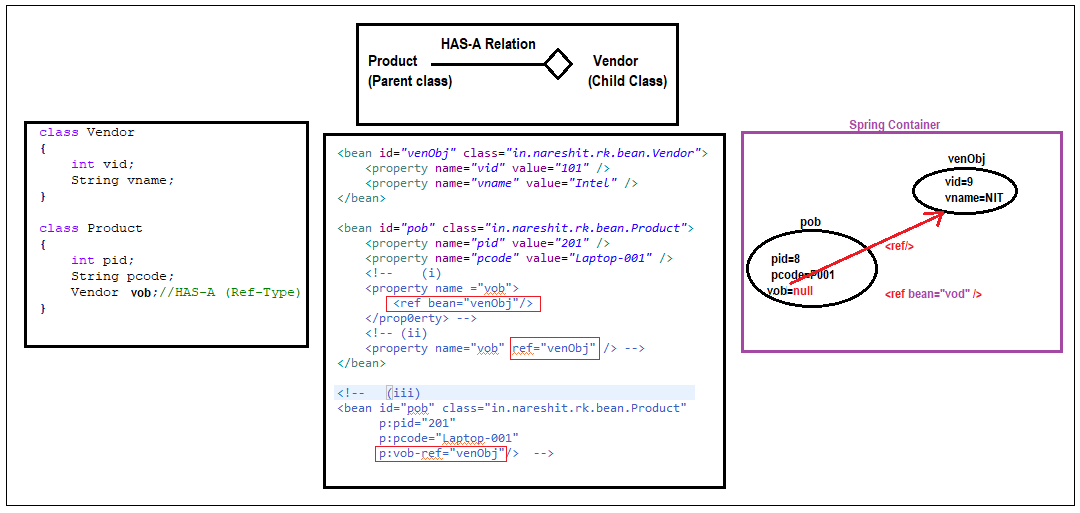
Ref: **ScApp6\_XmlConfig\_SI\_List\_Set\_Map\_Properties**

**Spring Core – Reference Tag - <ref/>**

HAS-A or Association Mapping: Creating a variable inside a class by using class/interface as data type.

* If two classes are connected using HAS-A relation then their objects are linked using **<ref/>** tag.
* The process of linking by using **<ref/>** Tag is called **wiring.** If **<ref/>** Tag is not used then default value is null(assigned). i.e no link between two objects.
* **3 Ways to define <ref/> Tag:** as marked red in diagram

1. **Ref as a Tag**
2. **Ref as attribute**
3. **p:schema-ref**

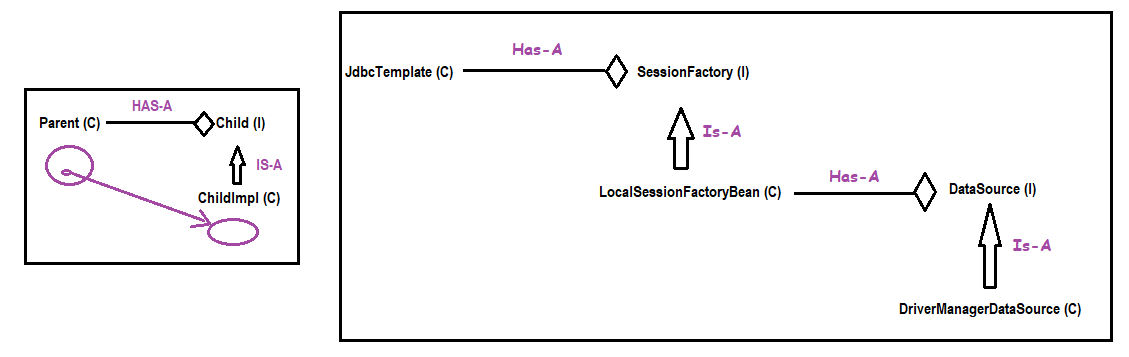


Ref: ScApp7\_XmlConfig\_RefTag

**Ref Tag using interface**

If Child is an interface then parent class object links to Child’s implementation class by using <ref/> tag.

If Two classes are connected using interface reference is called loose-coupling.



**Spring Bean -Scope**

**Scope**: lifetime/period of object in memory. \* When it is created, and How long data exist in memory

**Core Java Scopes**

1. local scope --> local variables

2. object scope --> instance variables

3. global scopre --> static variables

**Servlets Scopes in Web application)**

1. request scope : Data stored at server until response given

2. session scope : Data stored at server from login to logout

3. context scope : Data stored at server from app start to stop.

**Q) When bean/object is created in Container and how long it exists?**

A) It depends on the scope is used in programming.

Spring Bean scopes

1. singleton –default scope
2. prototype
3. request [web application]
4. session [web application]
5. global context [Removed]
6. **singleton** : For every configuration only one object is created by spring container when app started or container is created and exist until app stopped or container destroyed.

1 <bean> 1 object is created inside container.

1 @Component 1 object is created inside container.

1 @Bean 1 object is created inside container.

**Ex:** DataBase connection, Service/logic objects, ViewResolver…etc

**Realtime:** Controller, Service, DAO are recommended to be singleton

1. **prototype :** For every access/use one new object is required. Object is created by container for every request.

**Ex:** Token

1. **request:** Object is created when request comes and destroy once response committed.

**Ex:** Form Data(Register,Login,Payment….etc)

1. **session:** When we login is successfully a new object is created and stored until logout.

**Ex:** User data like role, name,id…. Etc

Coding syntax:

1. **XML Configuration :**

*<****bean*** id=”…” class=”…” ***scope*=”…”** />

1. **Annotation Configuration:**

***@Component(“…”)***

***@Scope(“…”)***

public class ClassName

{

…

}

1. **Java Configuration:**

***@Configuration***

public class AppConfig

{

**@Bean**

***@Scope(“..”)***

public <ClassName> <objName>()

{

….

}

}

**Ref: ScApp8\_SpringBeanScopes project**

**Lookup Method: Method level Injection**

Spring could inject dependencies between the beans in the application by following way of injection types:

1) Setter Injection

2) Constructor Injection

3) Field Injection (@Autowired at field)

4) Method Injection

Spring Beans have mainly two types of scope

1. Singleton (Instantiate only one object)

2. Prototype (Instantiate a new object every time).

In Spring application there are many beans injected to each other for a goal.

There is no problem when injected beans have same scope of beans like singleton bean injected with other singleton beans.

Sometimes in Spring, Problems arise when you need to **inject a prototype-scoped bean in a singleton-scoped bean**.

Since singletons are created (and then injected) during context creation it’s the only time the Spring context is accessed and thus prototype-scoped beans are injected only once.

**Lookup or Method Injection**

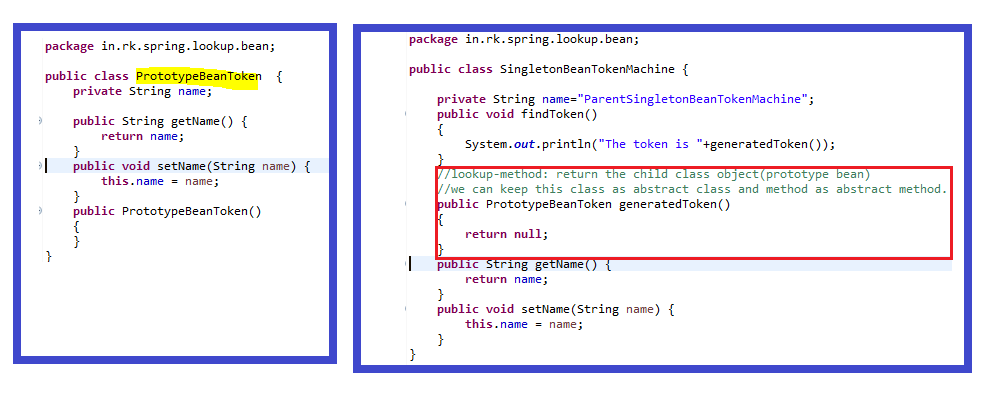
Spring provides another way for injection of beans, It is called method injection.

It is solution of above problem in injecting different scoped beans.

It works as that since singleton beans are instantiated at context creation, but it changes the way prototype-scoped are handled, from injection to create by an abstract method.

**ScApp8a\_LookupMethodInjection Project (Ref)**

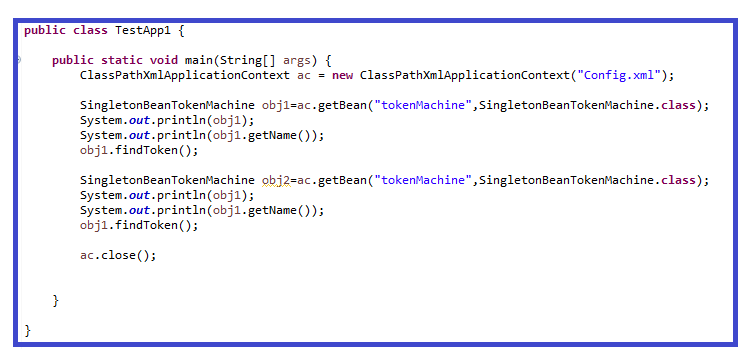
1. Create Maven project and add javasource compiler plugin, spring-contect dependencies from Web.
2. Create Spring bean classes **child(prototype)** and **parent(singleton)**



1. Create **config.xml** xml based spring configuration

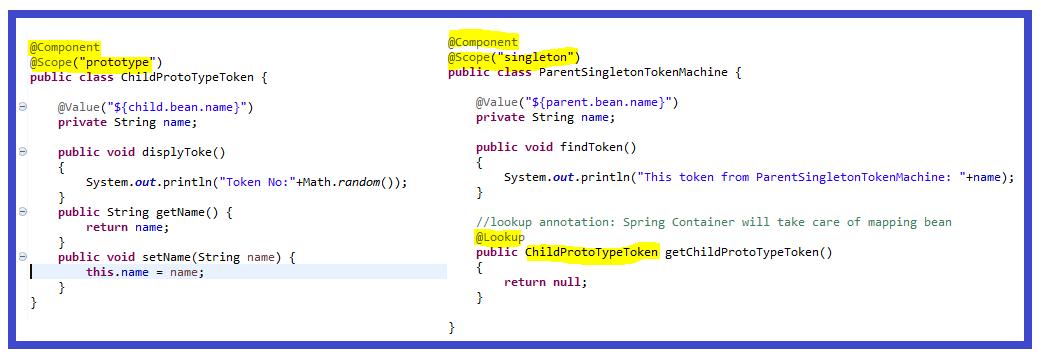


1. Create Test class then check the lookup methods flow.

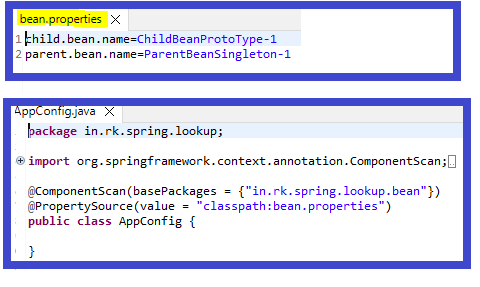


**ScApp8a\_LookupMethodInjectionAnnotation Project (Ref)**

1. Create Maven project and add javasource compiler plugin, spring-contect dependencies from Web.
2. Create Spring bean classes **child(prototype)** and **parent(singleton)**



1. Create Spring bean properties files and AppConfig file



1. Create Test class and check the lookup method injection

