**Autowiring-**

**Why?**

Suppose I have two classes Employee and address both are the dependent on each other so in that case we should go for autowiring concepts.

**What is Autowiring?**

Autowiring feature of spring framework enables you to inject the object dependency implicitly.

Note-

1. We can inject only secondary types.
2. Autowiring can not apply to primitives types. Spring container will take care of automatic dependency injection.
3. You can apply autowiring through constructor or setter method.

There are several types of autowiring are as follows.

* ByType-(internally it uses setter base injection)
* ByName--(internally it uses setter base injection)
* Constructor-(internally it uses constructor base injection)
* autodetect-(it uses both setter and constructor base injection.)

**ByType-**

In this case, spring framework attempts to find out a bean in the configuration file, whose bean id (policy) is matching with the property type (categories.java private Policy policy) to be wired.  If a bean found with class as property type then that class object will be injected into that property by calling setter injection.

If no class found then that property remains un-wired, but never throws any exception just like before.

**Policy.java**

**package** com.spring.auto;

**public** **class** Policy {

**private** String planName;

**private** **int** planAmount;

**public** String getPlanName() {

**return** planName;

}

**public** **void** setPlanName(String planName) {

**this**.planName = planName;

}

**public** **int** getPlanAmount() {

**return** planAmount;

}

**public** **void** setPlanAmount(**int** planAmount) {

**this**.planAmount = planAmount;

}

}

**Categories.java**

**package** com.spring.auto;

**public** **class** Categories {

**private** String name;

**private** Policy policy;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Policy getPolicy() {

**return** policy;

}

**public** **void** setPolicy(Policy policy) {

**this**.policy = policy;

}

**public** **void** show() {

System.***out***.println("Categories name>>" + name);

System.***out***.println("Policy Name>>" + policy.getPlanName());

System.***out***.println("Policy Amount>>" + policy.getPlanAmount());

}

}

**Test.java**

package com.test;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Test {

public static void main(String[] args) {

Resource res = new ClassPathResource("spring.xml");

BeanFactory factory = new XmlBeanFactory(res);

Categories categories = (Categories) factory.getBean("categories");

categories.show();

}

}

**Spring.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>

<bean id=*"categories"* class=*"com.spring.auto.Categories"* autowire=*"byType"*>

<property name=*"name"* value=*"Life Insurance"* />

</bean>

<bean id=*"policy"* class=*"com.spring.auto.Policy"*>

<property name=*"planName"* value=*"Term Plan"* />

<property name=*"planAmount"* value=*"8500"* />

</bean>

</beans>

**Pom.xml** (same as last program).

**Output**-

Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500

**Internal working of byType**

We called categories from Test.java [line number 14], then it will navigate to spring.xml file and check the bean id as categories, also read the autowire=byType, so spring container will checks for the bean id with class attribute policy in xml file then navigate to Categories.java and check the policy type whether it is exactly matching then inserts policy objects into categories.

Test.java we used to type cast to get our output.

Note- if we change bean id=p in spring.xml and select autowire=”byType” then wont get any error but in case of byName it will get error.

**ByName**

In this case, spring framework attempts to find out a bean in the configuration file, whose bean id (policy) is matching with the property name (categories.java private Policy policy) to be wired.  If a bean found with class as property name then that class object will be injected into that property by calling setter injection.

If no class found then that property remains un-wired, but never throws any exception just like before.

**Policy.java**

**package** com.spring.auto;

**public** **class** Policy {

**private** String planName;

**private** **int** planAmount;

**public** String getPlanName() {

**return** planName;

}

**public** **void** setPlanName(String planName) {

**this**.planName = planName;

}

**public** **int** getPlanAmount() {

**return** planAmount;

}

**public** **void** setPlanAmount(**int** planAmount) {

**this**.planAmount = planAmount;

}

}

**Categories.java**

**package** com.spring.auto;

**public** **class** Categories {

**private** String name;

**private** Policy policy;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Policy getPolicy() {

**return** policy;

}

**public** **void** setPolicy(Policy policy) {

**this**.policy = policy;

}

**public** **void** show() {

System.***out***.println("Categories name>>" + name);

System.***out***.println("Policy Name>>" + policy.getPlanName());

System.***out***.println("Policy Amount>>" + policy.getPlanAmount());

}

}

**Test.java**

**package** com.spring.auto;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Resource res = **new** ClassPathResource("spring.xml");

BeanFactory factory = **new** ~~XmlBeanFactory~~(res);

Categories categories = (Categories) factory.getBean("categories");

categories.show();

}

}

**Note-**

1. In byName,constructor and autodetect, Categories object name(categories) and getBean(“categories)” must be same and also bean id=”categories” should be same.
2. In spring.xml, bean id=”policy” and Categories.java(private Policy policy) must be same.

**Spring.xml**

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*><bean id=*"policy"* class=*"com.spring.auto.Policy"*>

<property name=*"planName"* value=*"Term Plan"* />

<property name=*"planAmount"* value=*"8500"* />

</bean>

<bean id=*"categories"* class=*"com.spring.auto.Categories"*

autowire=*"byName"*>

<property name=*"name"* value=*"Life Insurance"* />

</bean>

</beans>

Internal working byName

We called categories from Test.java [line number 14], then it will navigate to spring.xml file and check the bean id as categories, also read the autowire=byName, so spring container will checks for the bean id with class attribute policy in xml file then navigate to Categories.java and check the policy type whether it is exactly matching then inserts policy objects into categories,

and then injects value “Life Insurance” into name property of Categories class.

Test.java we used to type cast to get our output.

**Pom.xml** (same as last program).

**Output-**

Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500

**Constructor-**

Spring Autowiring by constructor is similar to spring autowiring byType [ internally it will considers as byType only ]  but with little difference, in byType we used setter injection here we have to use constructor injection.

**Policy.java**

**package** com.spring.auto;

**public** **class** Policy {

**private** String planName;

**private** String planAmount;

**public** Policy(String planName, String planAmount) {

**this**.planName = planName;

**this**.planAmount = planAmount;

}

**public** String getPlanName() {

**return** planName;

}

**public** String getPlanAmount() {

**return** planAmount;

}

}

**Categories.java**

**package** com.spring.auto;

**public** **class** Categories {

**private** String name;

**private** Policy policy;

**public** Categories(String name, Policy policy) {

**this**.name = name;

**this**.policy = policy;

}

**public** **void** show() {

System.***out***.println("categories name>>" + name);

System.***out***.println("Policy name>>" + policy.getPlanName());

System.***out***.println("Policy Amount>>" + policy.getPlanAmount());

}

}

**Test.java**

**package** com.spring.auto;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Resource res = **new** ClassPathResource("spring.xml");

BeanFactory factory = **new** ~~XmlBeanFactory~~(res);

Categories categories = (Categories) factory.getBean("categories");

categories.show();

}

}

**Spring.xml**

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>

<bean id=*"categories"* class=*"com.example.Categories"*

autowire=*"constructor"*>

<constructor-arg value=*"Life Insurance"*></constructor-arg>

</bean>

<bean id=*"policy"* class=*"com.example.Policy"*>

<constructor-arg value=*"Term Plan"*></constructor-arg>

<constructor-arg value=*"8500"*></constructor-arg>

</bean>

</beans>

**Output-**

Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500

**autodetect**

Actually spring autowire=”autodetect” first will works as Spring Autowiring constructor if not then works as Spring Autowiring byType, byType means setter injection.

Note- same as above example except one change in spring.xml file

<bean id="id1" class="com.test.Categories" autowire="autodetect">

**Policy.java**

**package** com.spring.auto;

**public** **class** Policy {

**private** String planName;

**private** **int** planAmount;

**public** String getPlanName() {

**return** planName;

}

**public** **void** setPlanName(String planName) {

**this**.planName = planName;

}

**public** **int** getPlanAmount() {

**return** planAmount;

}

**public** **void** setPlanAmount(**int** planAmount) {

**this**.planAmount = planAmount;

}

}

**Categories.java**

**package** com.spring.auto;

**public** **class** Categories {

**private** String name;

**private** Policy policy;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Policy getPolicy() {

**return** policy;

}

**public** **void** setPolicy(Policy policy) {

**this**.policy = policy;

}

**public** **void** show() {

System.***out***.println("Categories name>>" + name);

System.***out***.println("Policy Name>>" + policy.getPlanName());

System.***out***.println("Policy Amount>>" + policy.getPlanAmount());

}

}

**Test.java**

**package** com.spring.auto;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.~~XmlBeanFactory~~;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Resource res = **new** ClassPathResource("spring.xml");

BeanFactory factory = **new** ~~XmlBeanFactory~~(res);

Categories categories = (Categories) factory.getBean("categories");

categories.show();

}

}

**Spring.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>

<bean id=*"categories"* class=*"com.spring.auto.Categories"* autowire=*"autodetect"*>

<property name=*"name"* value=*"Life Insurance"* />

</bean>

<bean id=*"policy"* class=*"com.spring.auto.Policy"*>

<property name=*"planName"* value=*"Term Plan"* />

<property name=*"planAmount"* value=*"8500"* />

</bean>

</beans>

**Output-**

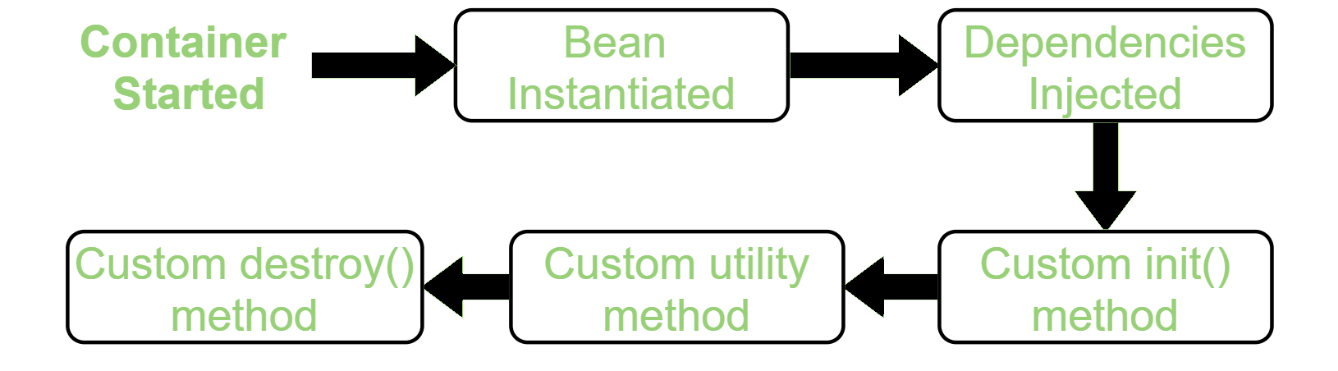
Categories name>>Life Insurance

Policy Name>>Term Plan

Policy Amount>>8500

Bean Life Cycle

* Bean life cycle is managed by the spring container.
* When we run the program then, first of all, the spring container gets started.
* After that, the container creates the instance of a bean as per the request, and then dependencies are injected.
* And finally, the bean is destroyed when the spring container is closed.
* Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.



**Student.java**

**package** com.test;

**public** **class** Student {

**private** String city;

**public** **void** setCity(String city) {

**this**.city = city;

}

**public** **void** getMessage(String name) {

System.***out***.println("Hello,"+name+","+city);

}

**public** **void** init() {

System.***out***.println("bean student has bean created");

}

**public** **void** destroy() {

System.***out***.println("Bean student has been destroyed");

}

}

**Test.java**

**package** com.test;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

ConfigurableApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("Spring.xml");

applicationContext.close();

}

}

**Spring.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"*>

<bean id=*"s"* class=*"com.test.Student"* init-method=*"init"*

destroy-method=*"destroy"*>

<property name=*"city"* value=*"pune"*/>

</bean>

</beans>