

SPRING CORE:

• Recap

- ApplicationContext creates the IOC container.
- We specify .xml file in path to refer the info about what bean to create and manage.
- If Spring is managing the object then it is called 'Bean'.
- **id** is reference and **class** contains info of which class's bean to be made.
- Using **property** tag we perform setter injection.
- Using **constructor-arg** tag we perform constructor injection.

```
<bean id="sb" class="services.SpringBoot"/>
<bean id="java" class="services.Java"/>
<bean id="ts" class="services.TShapedSkills">

  <property ref="java" name="course"></property>
  <constructor-arg></constructor-arg>
</bean>

</beans>
```

• Auto Wiring:

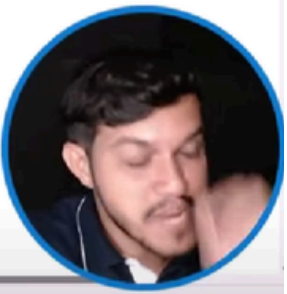
- Using **property** and **constructor-arg** tag we are performing manual dependency injection.
- If we want to automate the DI then we can use autowire.

```
<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-

<bean id="sb" class="services.SpringBoot"/>
<bean id="java" class="services.Java"/>
<bean id="ts" class="services.TShapedSkills" autowire="byType">

<!-- <property ref="java" name="course"></property> -->
<!-- <constructor-arg ref="java" name="course"></constructor-arg> -->
</bean>

</beans>
```



No qualifying bean of type 'services.ICourse' available: expected single matching bean but found 2: sb, java
veNotUnique(DependencyDescriptor.java:218)
y.doResolveDependency(DefaultListableBeanFactory.java:1420)
y.resolveDependency(DefaultListableBeanFactory.java:1353)
anFactory.autowireByType(AbstractAutowireCapableBeanFactory.java:1521)

- Here Java and SpringBoot are both implementing ICourse interface so they are of same type.
- “byType” states same type, so both Java and SpringBoot are eligible for bean creation, due to which there is ambiguity.

```
<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-

<bean id="sb" class="services.SpringBoot" primary="true"/>
<bean id="java" class="services.Java"/>
<bean id="ts" class="services.TShapedSkills" autowire="byType">

<!-- <property ref="java" name="course"></property> -->
<!-- <constructor-arg ref="java" name="course"></constructor-arg> -->
</bean>

</beans>
```



SpringBoot Bean created
Java Bean created
TShaped skills bean created
Setter is called for setter injection
SpringBoot Course is Purchased successfully and fees paid is 454.4
Course purchased successfully

- Here we have removed the ambiguity, **primary = “true”**.

- SpringBoot's object will be used in DI.
- Also we can see that we haven't specified Spring that what type of injection to use → it uses setter injection as default.
- For constructor injection → `autowire = "constructor"`.

• Bean Factory

```
public static void main(String[] args)
{
    //ApplicationContext
    //BeanFactory

    DefaultListableBeanFactory container= new DefaultListableBeanFactory();
    XmlBeanDefinitionReader read=new XmlBeanDefinitionReader(container);
    read.loadBeanDefinitions("applicationconfig.xml");
}
```

- This is how we initialize BeanFactory container.

• Eager Initialization :

```
12 public static void main(String[] args)
13 {
14
15     //ApplicationContext
16     //BeanFactory
17
18
19     ApplicationContext container=new ClassPathXmlApplicationContext("applicationconfig.xml");
20
21     /*
22     * TShapedSkills t = container.getBean(TShapedSkills.class); Boolean
23     * status=t.buyTheCourse(454.4); if(status)
24     * System.out.println("Course purchahsed successfully"); else
25     * System.out.println("Failed to get the course");
26     */
27 }
28
29 }
30
```

SpringBoot Bean created
Java Bean created
TShaped skills bean created

- Here Application context initializes the beans even if they currently not in use.
- If you go with Application context, whether you are going to use that bean, doesn't matter, Spring will create the beans of all the beans you have specified in application.xml (configuration file).

• Lazy Initialization :

- But if you go with Bean Factory then in that case, it will create bean only if you are using.
- If you use the getBean method after creation of bean then that bean will be initialized.

• Difference b/w ApplicationContext and BeanFctory:

```
ApplicationContext container=new ClassPathXmlApplicationContext("applicationconfig.xml");
```

- If we want to make beans from more than one .xml file then in ApplicationContext we would have to create one more instance of it to specify in its constructor.

```
DefaultListableBeanFactory container= new DefaultListableBeanFactory();
```

```
XmlBeanDefinitionReader read=new XmlBeanDefinitionReader(container);  
read.loadBeanDefinitions("applicationconfig.xml");  
read.loadBeanDefinitions("applicationconfig2.xml");
```

I

- But in BeanFactory, we don't have to make more instances.