

31 to 1 march Video:-

Tuesday, May 24, 2022 12:58 PM

Constructor injection:- Whenever we injecting dependency throw constructor then it is called constructor injection.

Setter Method Base injection:- Whenever we are injecting dependency throw setter method then it is called setter method based injection.

When one object depend upon another object then if we want to add one more property into item then we have to be also change into the order object we have to set that property data into order object so that is called hardcoded java program but this thing is not good that if we change one object data then we have to be change manually into dependent object so spring core not allow us to do so in spring core it will be happen automatically.

If we do not want to hard code data then we should use beans.xml file. Because in future if we want to change price of item we have to change into only one file that is called beans.xml file.

For more information go into the spring core file in shared location on git ..

IOC Container:- our order class have dependency on item class so initialize the order throw constructor or setter method so this is the wrong this is the hardcoded. So better way we should adapted we configure this object into a xml file and xml file specify <property> dependency. When we apply this dependency it will automatically inject into the initializer automatically injected so whenever you injected the dependency throw generic programs this is called as IOC Container.

IOC is says we should write a xml file there we mention our beans and give it to xml file I will find out what interdependency in this xml file we have and write such generic code using reflection (Class.forName()) we should injecting the item into the order automatically And give us injected code hence I can say this program is called IOC Container.

Note:- spring framewrok provides us a IOC Container. And our jobs only write xml file.

History :- Spring framework is an open source Java platform and it was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Business logic -> EJB -> Application Server (costly) -> weblogic/webspare.

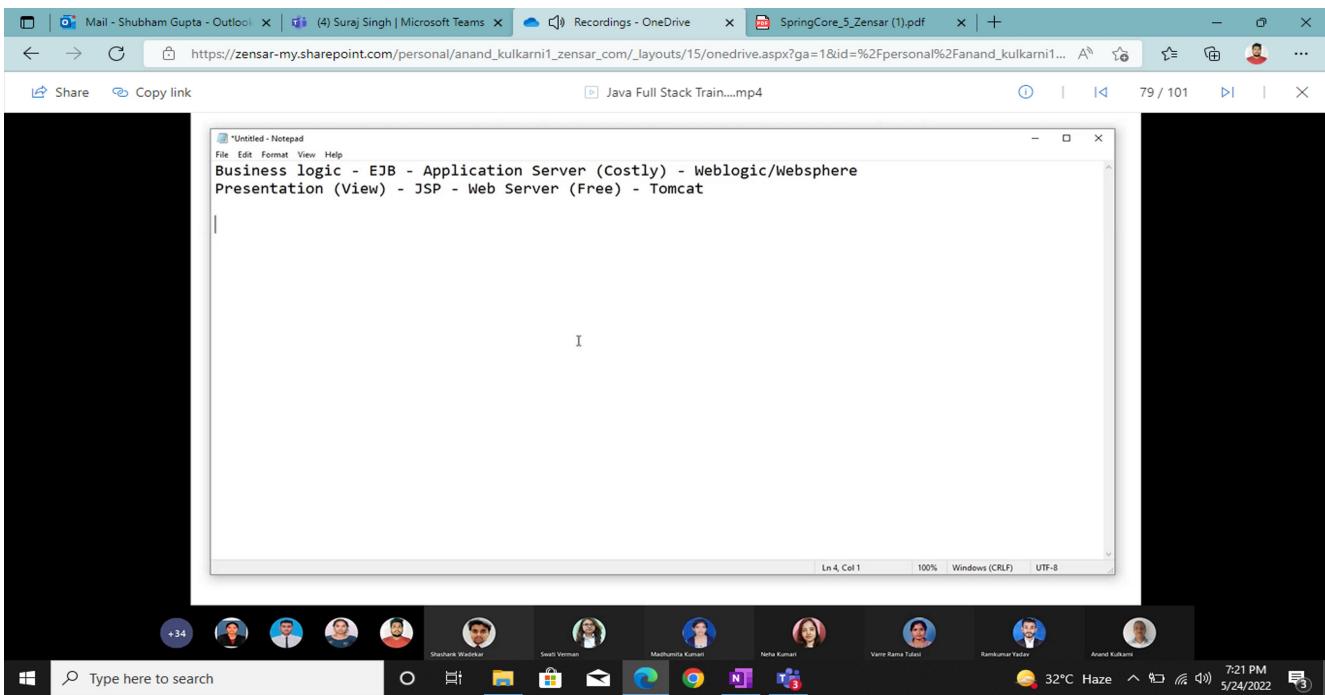
Presentation (View) -> JSP ->Web Server -> Tomcat.

In earlier day when spring framework had not come then when we had to written complex query or business logic using EJB(Enterprise Java Bean) and then we had to used Application Server and Application server cost is very high so this was the major drawback small company to use Application server but after spring framework have come into picture then we can write complex business logic into the JSP(Java Server Pages) and we can also use.

Web Server like Tomcat.

Application Server:- it is having web server feature as well as lots of built in services like collection pulling services, Jindia services.. and it very costly..

web server:- whenever any normal server we will take which will handle http request coming from the browser and send the http response to the client this is called simple web server. And web server is commercially free.



Benefits of Using Spring Framework :----

- 1) Spring enables developers to develop enterprise-class applications using POJOs. No need of EJB container.
- 2) It truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.
- 3) Spring's web framework is a well-designed web MVC framework.
- 4) Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
- 5) Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
- 6) Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Let see how to create spring project let see the steps click on

File -> New -> maven project -> catlog(intarnal) -> choose artifact id -> and give the project name and group id

Add below dependency into POM file

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.9</version>
</dependency>
```

After created project right click on project change jdk int 1.8

And create a "resource" folder into src -> main folder and inside the resource folder we have to create xml file

Why we are putting xml file inside the main folder because spring core take it automatically from src-> main folder
Hame batane ki jarurat ni padti h ki xml file kha h...

Go to the main app.java file and mention the IOC container ClassPathXmlApplicationContext("beans.xml");

This is the loc container it is return reference

```
AbstractApplicationContext iocContainer = new ClassPathXmlApplicationContext("beans.xml");
Item item =(Item) iocContainer.getBean("itemBeans");
System.out.println("Item:- "+item);
```

```
<Bean id="itemBean" class="com.zensar.Item">
<Property name="name" value="Laptop"></Property>
<Property name="price" value="7000"></Property>
</Bean>
```

Type of IOC Container:-----

Spring IoC containers:-

- 1) BeanFactory container This is the simplest container providing basic support for DI and defined by the org.springframework.beans.factory.BeanFactory interface.
- 2) ApplicationContext container This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the org.springframework.context.ApplicationContext interface. The ApplicationContext container includes all functionality of the BeanFactory container.

BeanFactory IoC container:--

```
XmlBeanFactory factory = new XmlBeanFactory (new ClassPathResource("Beans.xml"));
HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
obj.getMessage();
```

ApplicationContext IoC container:---

The Application Context is spring's more advanced container. The ApplicationContext includes all functionality of the BeanFactory, it is generally recommended over the BeanFactory. The most commonly used ApplicationContext implementations are. It is an interface and it has many implementation classes like AbstractApplicationContext and AbstractApplicationContext is not a normal class it is also an Abstract class.

Any class which is implementing ApplicationContext is called IOC Container.

These three are famous implementation of ApplicationContext.

- 1) FileSystemXmlApplicationContext (this is used when our XML file is present at a different location not present inside the project dir).
- 2) ClassPathXmlApplicationContext (& this is used when our XML file is present inside the project dir)
- 3) WebXmlApplicationContext (this is used when we would create dynamic web applications)

FileSystemXmlApplicationContext and WebXmlApplicationContext are not used mostly.

ApplicationContext Implementations :--

- 1) FileSystemXmlApplicationContext: This container loads the definitions of the beans from an XML file. ApplicationContext context = new FileSystemXmlApplicationContext ("c:/Beans.xml");
- 2) ClassPathXmlApplicationContext: This container loads the definitions of the beans from an XML file using CLASSPATH. ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
- 3) XmlWebApplicationContext: This container loads the XML file with definitions of all beans from within a web application. WebApplicationContext ctx = WebApplicationContextUtils.getWebApplicationContext(servletContext);

<Property name="name" value="Laptop"></Property> this is called setter method injection.

<constructor-args name="name" value="Laptop"></constructor-args> this is called constructor based injection.

Lazy-Initialization: jab hm getBean() method ko call krte h to us time pe agar bean initialize hota h to usse hm lazy-Initialization khete h. If we want to make any bean lazy than we can write like below in bean tag. If we use lazy-init ="true" it's mean that

```
<Bean id="itemBean" class="com.zensar.Item" lazy-init="true">
<Property name="name" value="Laptop"></Property>
<Property name="price" value="7000"></Property>
```

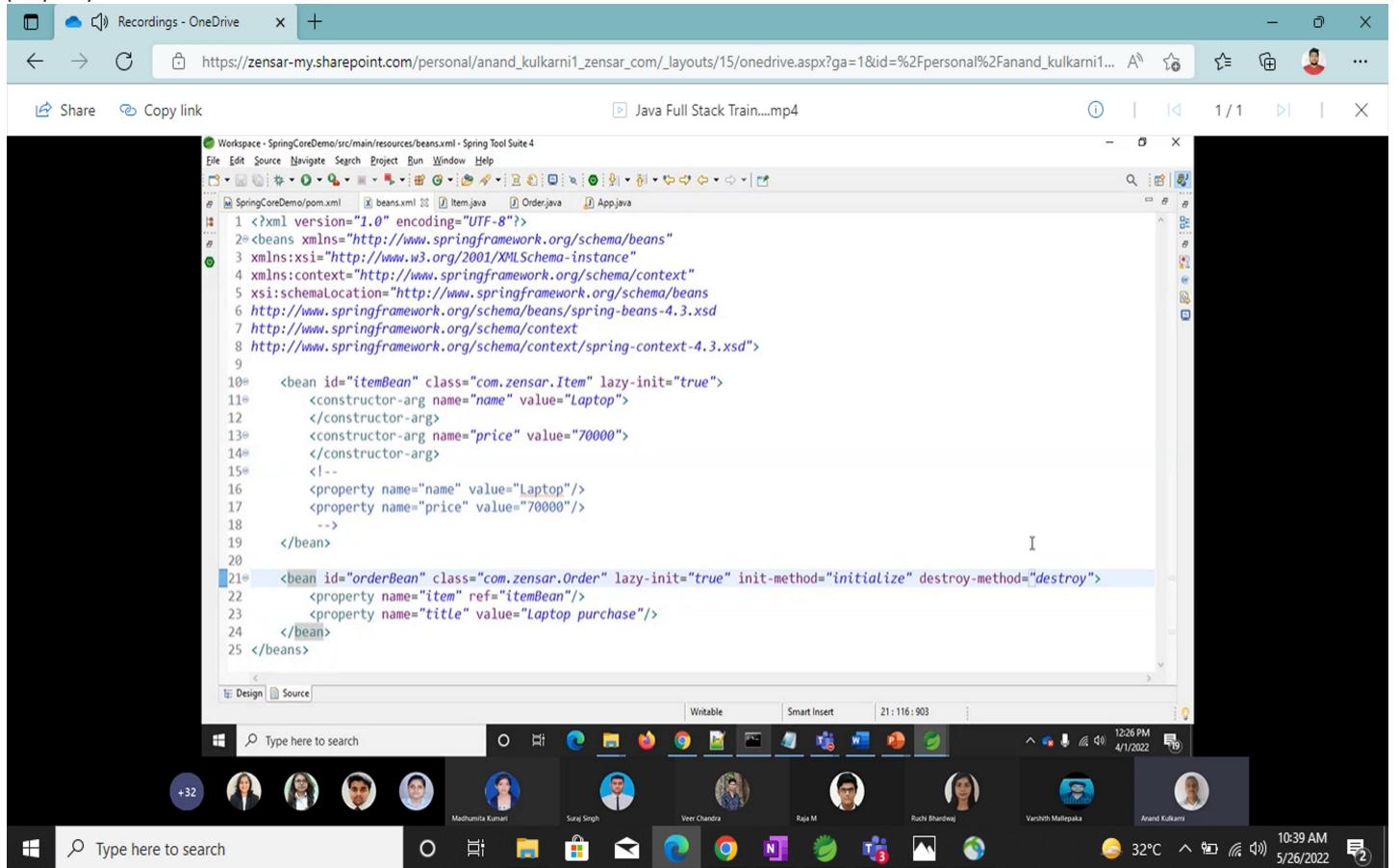
```
</Bean>
```

Eager-Initialization:- jaise hi hm beans.xml file classPathXmlApplication me paas krte h to ussi time pe initialization ho jata h to usse Eager-Initialization khte h. and by default eager-initialization hi hota h.

Initialize method:- if we make any bean init-method="initialize" like this it's mean initialize function have in that bean exist and that will be run when that bean all property would be set then it will automatically called like below images.

Any business logic we want to perform only once then we can go for init-method="add" . Like we can use it for getConnection from database.

Destroy method :- if we want to close databases connections then we may go for destroy if we make bean destroy-method="destroy" then we should declare that destroy method into our class and again we have to call one more fucntion to call destroy function that is called via iocContainer.registerShutdownHook(); then destroy function will call. It call at last of the all function or property call. Like below ...



The screenshot shows a Microsoft Edge browser window displaying a Spring Core Demo application. The URL is https://zensar-my.sharepoint.com/personal/anand_kulkarni1_zensar_com/_layouts/15/onedrive.aspx?ga=1&id=%2Fpersonal%2Fanand_kulkarni1... The page content is a Java Full Stack Train....mp4 video player. The browser toolbar includes Back, Forward, Stop, Refresh, and a search bar. Below the browser is a taskbar with various pinned icons and user profiles. The main content area shows a code editor with XML configuration for a Spring application. The XML code defines two beans: itemBean and orderBean. The itemBean is a com.zensar.Item object with properties name="laptop" and price="70000". The orderBean is a com.zensar.Order object with properties item=itemBean, title="Laptop purchase", and a destroy-method="destroy" attribute. The code editor interface includes tabs for Design and Source, and a status bar at the bottom showing the date and time.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:context="http://www.springframework.org/schema/context"
5   xsi:schemaLocation="http://www.springframework.org/schema/beans
6   http://www.springframework.org/schema/beans-4.3.xsd
7   http://www.springframework.org/schema/context
8   http://www.springframework.org/schema/context/spring-context-4.3.xsd">
9
10<bean id="itemBean" class="com.zensar.Item" lazy-init="true">
11  <constructor-arg name="name" value="laptop">
12  </constructor-arg>
13  <constructor-arg name="price" value="70000">
14  </constructor-arg>
15<!--
16  <property name="name" value="laptop"/>
17  <property name="price" value="70000"/>
18-->
19</bean>
20
21<bean id="orderBean" class="com.zensar.Order" lazy-init="true" init-method="initialize" destroy-method="destroy">
22  <property name="item" ref="itemBean"/>
23  <property name="title" value="Laptop purchase"/>
24</bean>
25</beans>
```

Recordings - OneDrive

https://zensar-my.sharepoint.com/personal/anand_kulkarni1_zensar_com/_layouts/15/onedrive.aspx?ga=1&id=%2Fpersonal%2Fanand_kulkarni1...

Share Copy link

Java Full Stack Train....mp4

Workspace - SpringCoreDemo/src/main/java/com/zensar/Order.java - Spring Tool Suite 4

```
1 package com.zensar;
2
3 public class Order {
4     private Item item;
5     private String title;
6     public Order() {
7         System.out.println("Inside Order default constructor");
8     }
9     public Order(Item item, String title) {
10        this.item = item;
11        this.title = title;
12        System.out.println("Inside Order parameterized constructor..");
13    }
14    public void initialize() {
15        System.out.println("Inside initialize()");
16    }
17    public void destroy() {
18        System.out.println("Inside destroy()");
19    }
20    public Item getItem() {
21        return item;
22    }
23    public void setItem(Item item) {
24        this.item = item;
25        System.out.println("Inside setItem()");
26    }
27    public String printTitle() {
28        System.out.println("Inside printTitle()");
29    }
30}
```

Type here to search

12:26 PM 4/1/2022

+32 Madhumita Kumar Suraj Singh Veer Chandra Raja M Ruchi Bhandewar Vanshith Mallepaka Anand Kulkarni

32°C 10:40 AM 5/26/2022

Recordings - OneDrive

https://zensar-my.sharepoint.com/personal/anand_kulkarni1_zensar_com/_layouts/15/onedrive.aspx?ga=1&id=%2Fpersonal%2Fanand_kulkarni1...

Share Copy link

Java Full Stack Train....mp4

Workspace - SpringCoreDemo/src/main/java/com/zensar/App.java - Spring Tool Suite 4

```
1 package com.zensar;
2
3 import org.springframework.context.support.AbstractApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6 public class App {
7
8     public static void main( String[] args )
9     {
10        //IoC container provided by Spring framework
11        AbstractApplicationContext iocContainer =
12            new ClassPathXmlApplicationContext("beans.xml");
13
14        Order order = (Order)iocContainer.getBean("orderBean");
15        System.out.println("Order: " + order);
16
17        iocContainer.registerShutdownHook();
18    }
19 }
```

Type here to search

12:27 PM 4/1/2022

+32 Madhumita Kumar Suraj Singh Veer Chandra Raja M Ruchi Bhandewar Vanshith Mallepaka Anand Kulkarni

32°C 10:41 AM 5/26/2022

The screenshot shows an IDE interface with a code editor on the left containing Java code, and a console window on the right displaying the execution of a program. The code in the editor includes imports and a class definition. The console output shows the execution path through constructor, setter, and initialize methods, followed by the final output of the Order object.

```
nContext;
actionContext;
"");

```

```
<terminated> App [Java Application] C:\Program Files\Java\jdk1.8.0_121\bin\javaw
Inside Order default constructor
Inside setItem()
Inside initialize()
Order: Laptop purchase - Laptop - 70000.0
Inside destroy()|
```

The screenshot shows a PDF document titled "Spring Bean Attributes". It contains a table with 9 rows, each listing a bean attribute, its description, and a brief note about its usage. The table has three columns: Sr.No., Bean Property, and Description.

Sr.No.	Bean Property	Description
1.	class	This attribute is mandatory and specify the bean class to be used to create the bean.
2.	id	This attribute specifies the bean identifier uniquely.
3.	scope	Specifies the scope of the objects created.
4.	constructor-arg	This is used to inject constructor level dependencies.
5.	properties	This is used to inject the dependencies at bean property level.
6.	autowiring	This is used to inject the dependencies using bean autowiring.
7.	lazy-initialization	Creates a bean instance when it is first requested.
8.	initialization method	A callback to be called just after all necessary properties on the bean have been set by the container.
9.	destruction method	A callback to be used when the container containing the bean is destroyed.

Spring bean configuration :-

We can configure beans in two different ways:

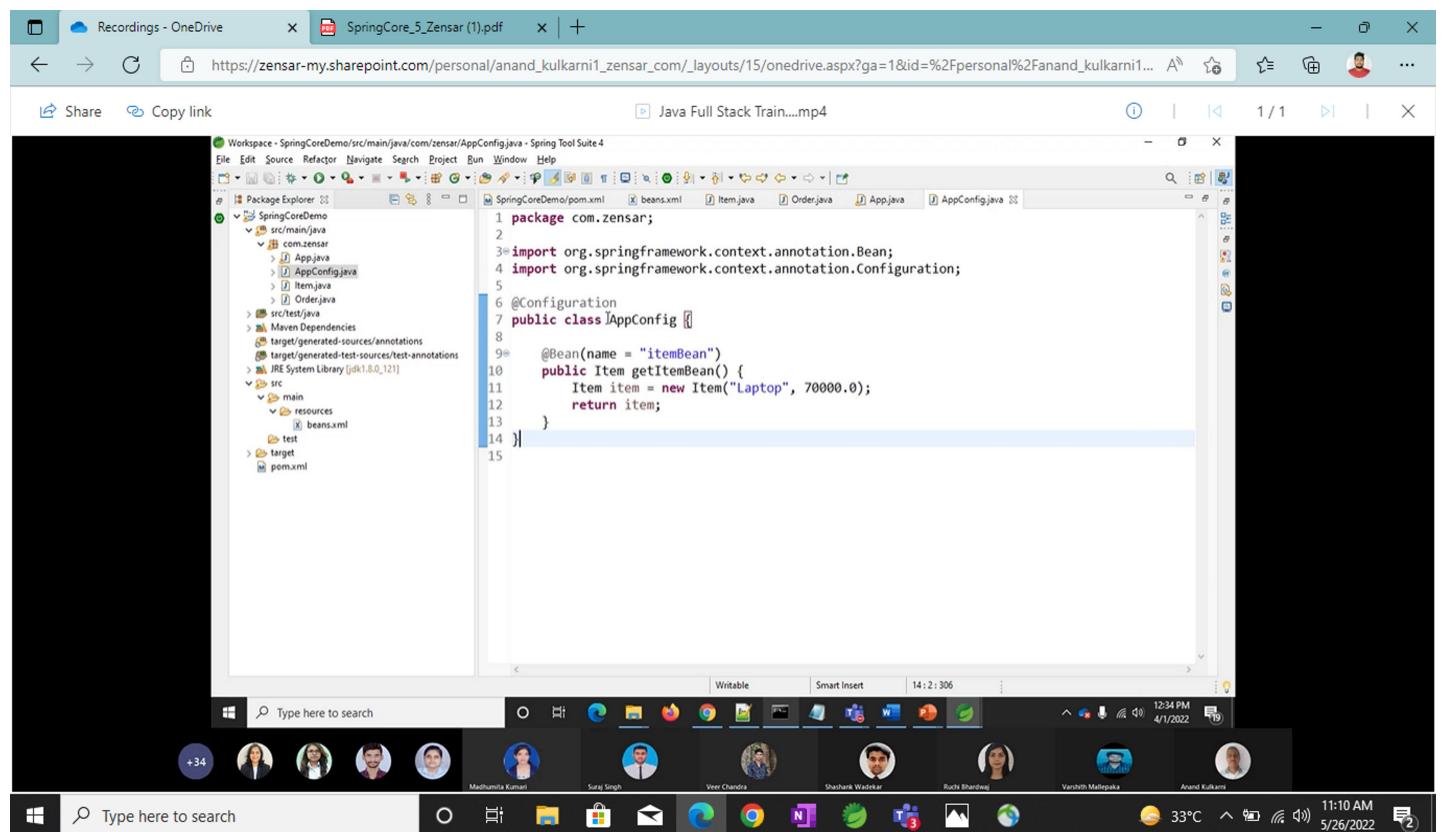
1. XML based configuration file.
2. Java Annotation based configuration.

XML based configuration :- we have seen xml based configuration.

```
<Bean id="itemBean" class="com.zensar.Item" lazy-init="true">
<Property name="name" value="Laptop"></Property>
<Property name="price" value="7000"></Property>
</Bean>
```

Java Annotation based configuration:- if we use java Annotation based configuration then we no need to use beans.xml file.

```
@Configuration  
public class AppConfig {  
  
    @Bean(name="messagePrinter")  
    public MessagePrinterBean getMessagePrinterBean()  
    {  
        return new MessagePrinterBean();  
    }  
}
```



The screenshot shows a Microsoft Edge browser window with two tabs open: 'Recordings - OneDrive' and 'SpringCore_5_Zensar (1).pdf'. The PDF page displays a table titled 'Bean scopes' with five rows. The table has three columns: 'Sr.No.', 'Bean Scope', and 'Description'. The rows are numbered 1 through 5.

Sr.No.	Bean Scope	Description
1.	singleton	Creates single instance of bean within IOC container. (default)
2.	prototype	Creates new bean whenever request to the bean is made.
3.	request	Bean instance is valid until request is going on.
4.	session	Bean instance is valid until client's session is going on.
5.	global-session	Bean scope will be assigned to HTTP session & applicable for portlets.

Request:- When the client makes request to your bean whichever components is trying to access our order bean all of them is going to share The same bean instance.

Session and global-session rarely use and also request onlyb two scope is important singleton and prototype.

Aware interfaces:-

- Spring offers a range of Aware interfaces that allow beans to indicate to the container that they require a certain infrastructure dependency.
- Each interface will require you to implement a method to inject the dependency in bean.
- For example:

```
public class MessagePrinterBean implements BeanNameAware {  
    public void setBeanName(String beanName) {  
        System.out.println("Bean Name: " + beanName);  
    }  
}
```

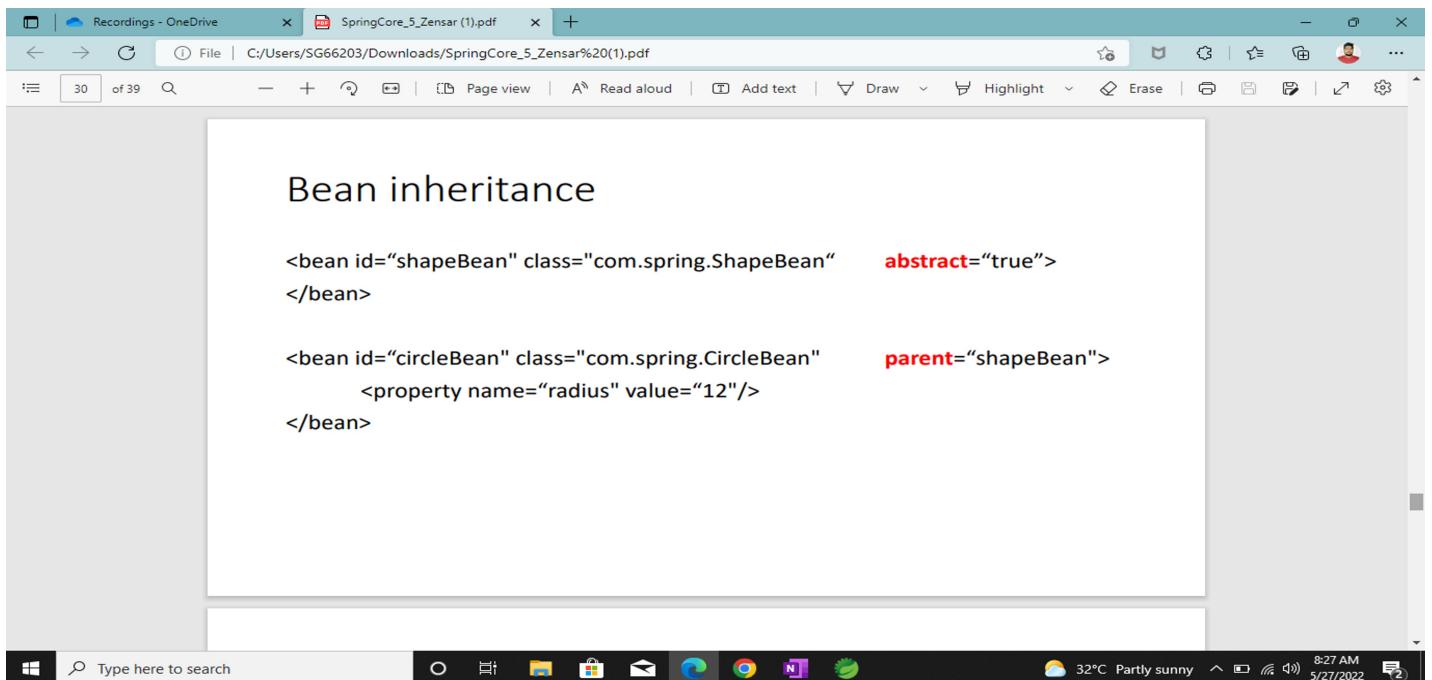
The screenshot shows a Microsoft Edge browser window with two tabs open: 'Recordings - OneDrive' and 'SpringCore_5_Zensar (1).pdf'. The current tab displays a table titled 'List of aware interfaces'.

Sr.No.	Aware Interface	Purpose
1.	ApplicationContextAware	Bean that wishes to be notified of the ApplicationContext that it runs in.
2.	ApplicationEventPublisher Aware	Set the ApplicationEventPublisher that this object runs in.
3.	BeanClassLoaderAware	Supplies the bean class loader to a bean instance.
4.	BeanFactoryAware	Supplies the owning factory to a bean instance.
5.	BeanNameAware	Supplies the bean name to a bean instance.
6.	BootstrapContextAware	Supplies the BootstrapContext to a bean instance.
7.	MessageSourceAware	Supplies the MessageSource to a bean instance.
8.	ServletConfigAware	Supplies servlet config to a bean instance.
9.	ServletContextAware	Supplies servlet context to a bean instance.

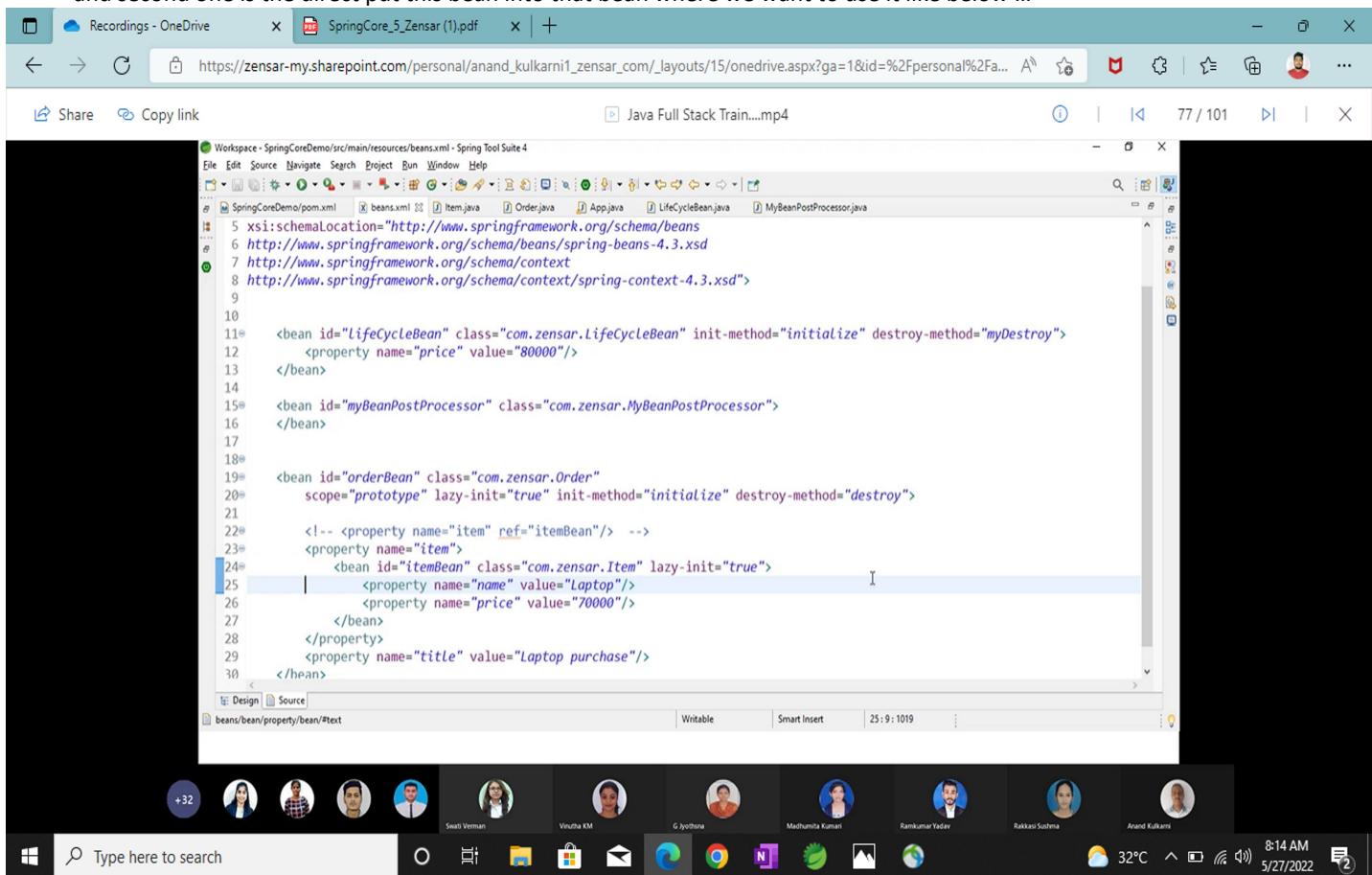
Bean life cycle

- Bean life cycle**
1. Reads bean configuration from xml file.
 2. Creates instance of bean class.
 3. Calls setter methods on bean to set the specified properties.
 4. Calls aware interface methods on bean if it implements any.
 5. If BeanPostProcessor is associated then calls postProcessBeforeInitialization() method on implementing class.
 6. If bean implements InitializingBean interface then calls afterPropertiesSet() method on bean.
 7. If bean configuration has "init-method" or @PostConstruct specified then it calls the defined init method on bean.
 8. If BeanPostProcessor is associated then calls postProcessAfterInitialization() method on implementing class.
 9. If bean implements DisposableBean interface then it calls destroy() method on bean.
 10. If bean configuration has "destroy-method" or @PreDestroy specified then it calls the defined destroy method on bean.

Bean inheritance :- if we want inheritance then we can do also...



Note:- if we not want to use another class object bean into other class bean then we can use one way is to using ref variable and second one is the direct put this bean into that bean where we want to use it like below ...



In above image we can see we have used item bean into order bean directly.....

Using collections

Spring allows us to configure the following collections inside xml:

1. <list> : Ordered collection with duplicates.
2. <set> : Collection without any duplicate.
3. <map> : Key-value based where key & value can be of any type.
4. <props> : Key-value based but key & value both are strings.

Configure list & set inside xml

```
<bean id="collectionBean" class="com.spring.beans.CollectionBean">
    <property name="cityList">
        <list>
            <value>Pune</value>
            <value>Mumbai</value>
            <value>Kolkatta</value>
            <value>Mumbai</value>
        </list>
    </property>
    <property name="countrySet">
        <set>
            <value>India</value>
            <value>USA</value>
            <value>Austria</value>
            <value>India</value>
        </set>
    </property>

```

Configure map & props inside xml

Recordings - OneDrive

File | C:/Users/SG66203/Downloads/SpringCore_5_Zensar(1).pdf

34 of 39

Configure map & props inside xml

```
<bean id="collectionBean" class="com.spring.beans.CollectionBean">
    <property name="zipCityMap">
        <map>
            <entry key="411001" value="Pune" />
            <entry key="411002" value="Mumbai" />
            <entry key="411003" value="Kolkatta" />
            <entry key="411004" value="Chennai" />
        </map>
    </property>
    <property name="cityCountryProps">
        <props>
            <prop key="Pune">INDIA</prop>
            <prop key="Washington">USA</prop>
            <prop key="Stockholm">SWEDEN</prop>
        </props>
    </property>
```

Type here to search

31°C Haze 8:31 AM 5/27/2022

Autowiring

Share Copy link

Java Full Stack Train....mp4

83 / 101

Workspace - SpringCoreDemo/src/main/resources/beans.xml - Spring Tool Suite 4

File Edit Source Navigate Search Project Run Window Help

Package Explorer

SpringCoreDemo

src/main/java

com.zensar

App.java

AppConfig.java

ChatApplication.java

Item.java

LifeCycleBean.java

MyBeanPostProcessor.java

Order.java

src/test/java

Maven Dependencies

target/generated-sources/annotations

target/generated-test-sources/test-annotations

JRE System Library [jdk1.8.0_121]

src

main

resources

beans.xml

beans.xml

Design Source

```
11<bean id="chatApp" class="com.zensar.ChatApplication">
12    <property name="usersMap">
13        <map>
14            <entry key="Sports">
15                <set>
16                    <value>Anand</value>
17                    <value>Jerry</value>
18                    <value>Tom</value>
19                </set>
20            </entry>
21            <entry key="Music">
22                <set>
23                    <value>Bipin</value>
24                    <value>Tony</value>
25                    <value>Ivan</value>
26                </set>
27            </entry>
28        </map>
29    </property>
30    <property name="messagesMap">
31        <map>
32            <entry key="Sports">
33                <list>
34                    <value>Hello</value>
35                    <value>Hi</value>
36                </list>
37            </entry>
38        </map>
39    </property>
40</bean>
```

Autowiring :-

Spring framework provides facility to inject a bean into another bean implicitly which is known as 'Autowiring'. Thus, Autowiring allows Spring to resolve and inject collaborating beans into our bean using IoC Container. Suppose, UserController bean

has dependency on UserService bean then autowiring is done as below:

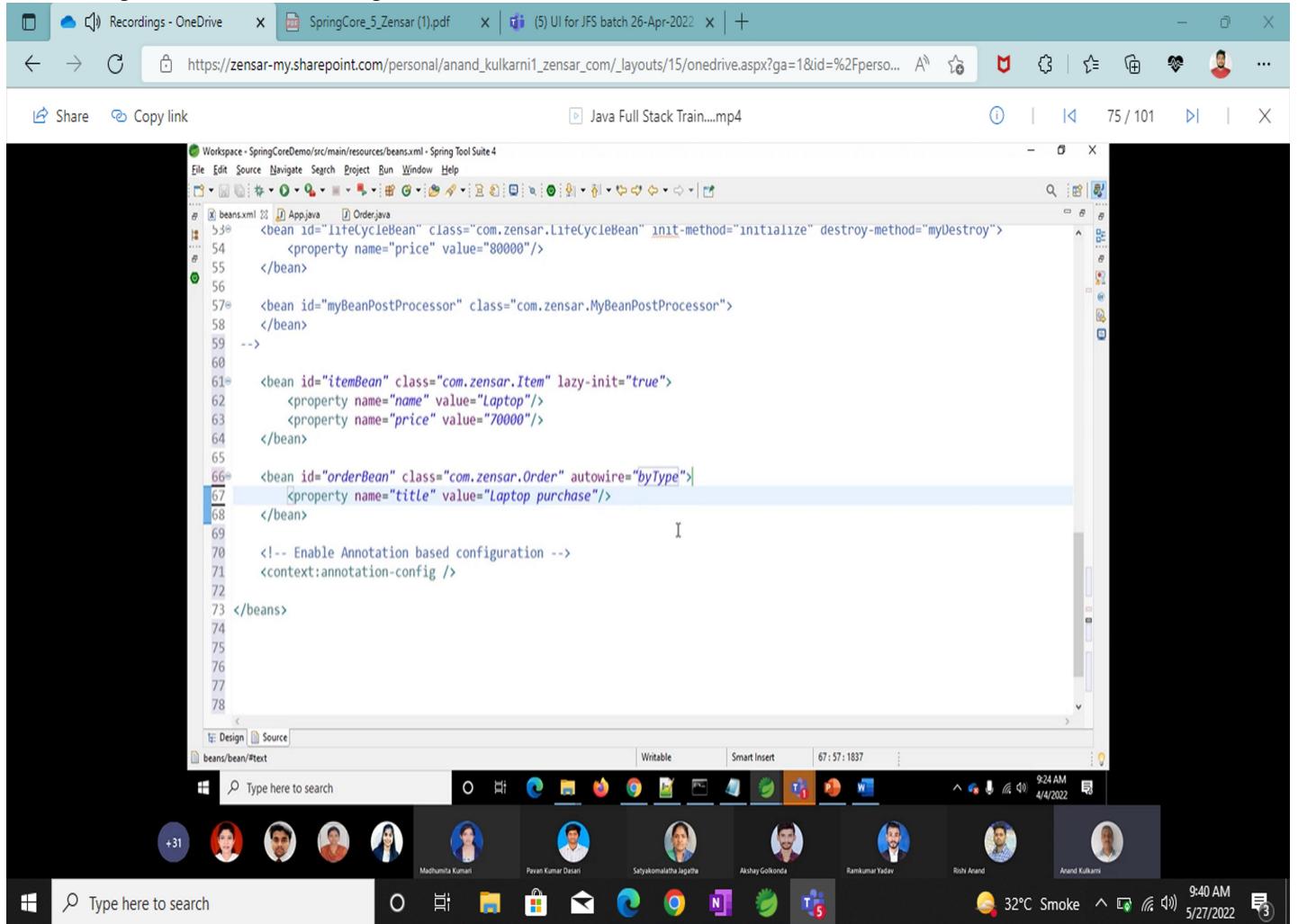
```
public class UserController {  
  
    @Autowired  
    UserService userService;  
  
}
```

Types of Autowiring :-

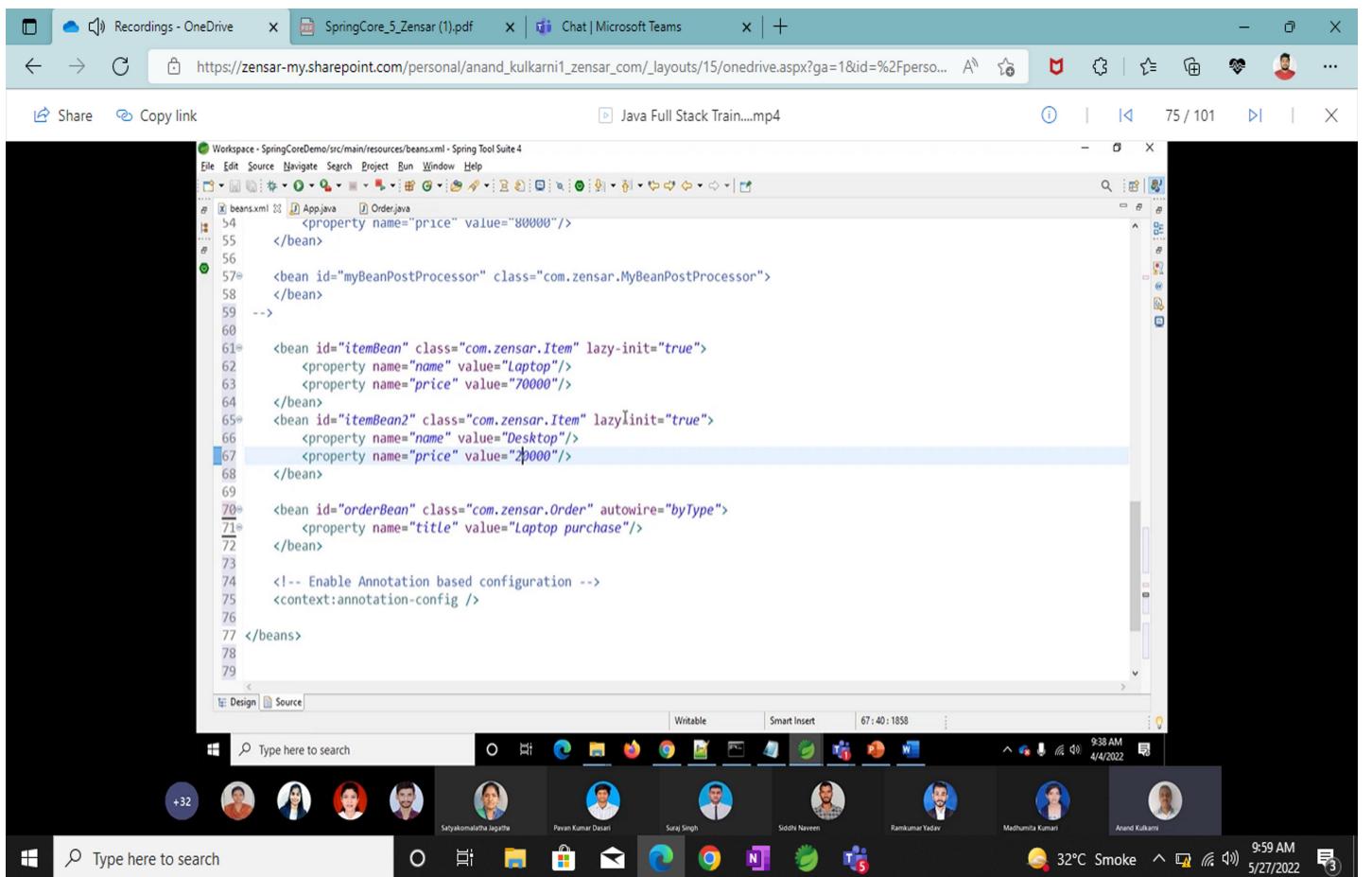
Autowiring is divided into following types:

1. autowire byType
2. autowire byName
3. autowire by constructor
4. @Autowired annotation
5. @Qualifier annotation

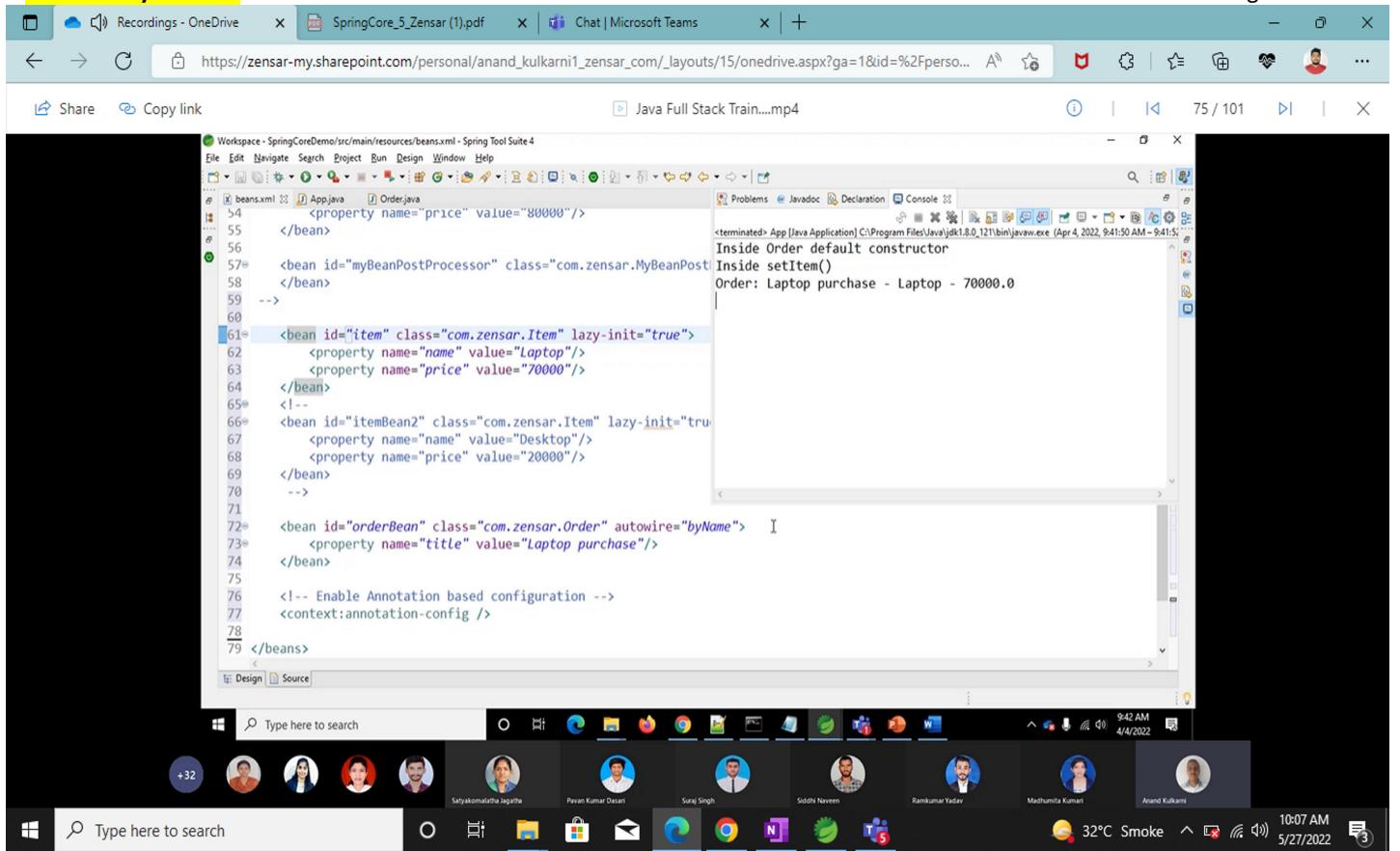
1. **autowire byType** :- if we want to inject item object into order object then we can use autowire="byType" into bean tag and this will search that type into the xml file and if ioc container get the same type of object name then it will put it into the The bean tag and run it like below image



But in byType has a problem when we are using two or more item bean then it will not distinct that which one it has to insert into order bean because both bean is same type like below image have two bean it name is itemBean and itemBean2 it will give error

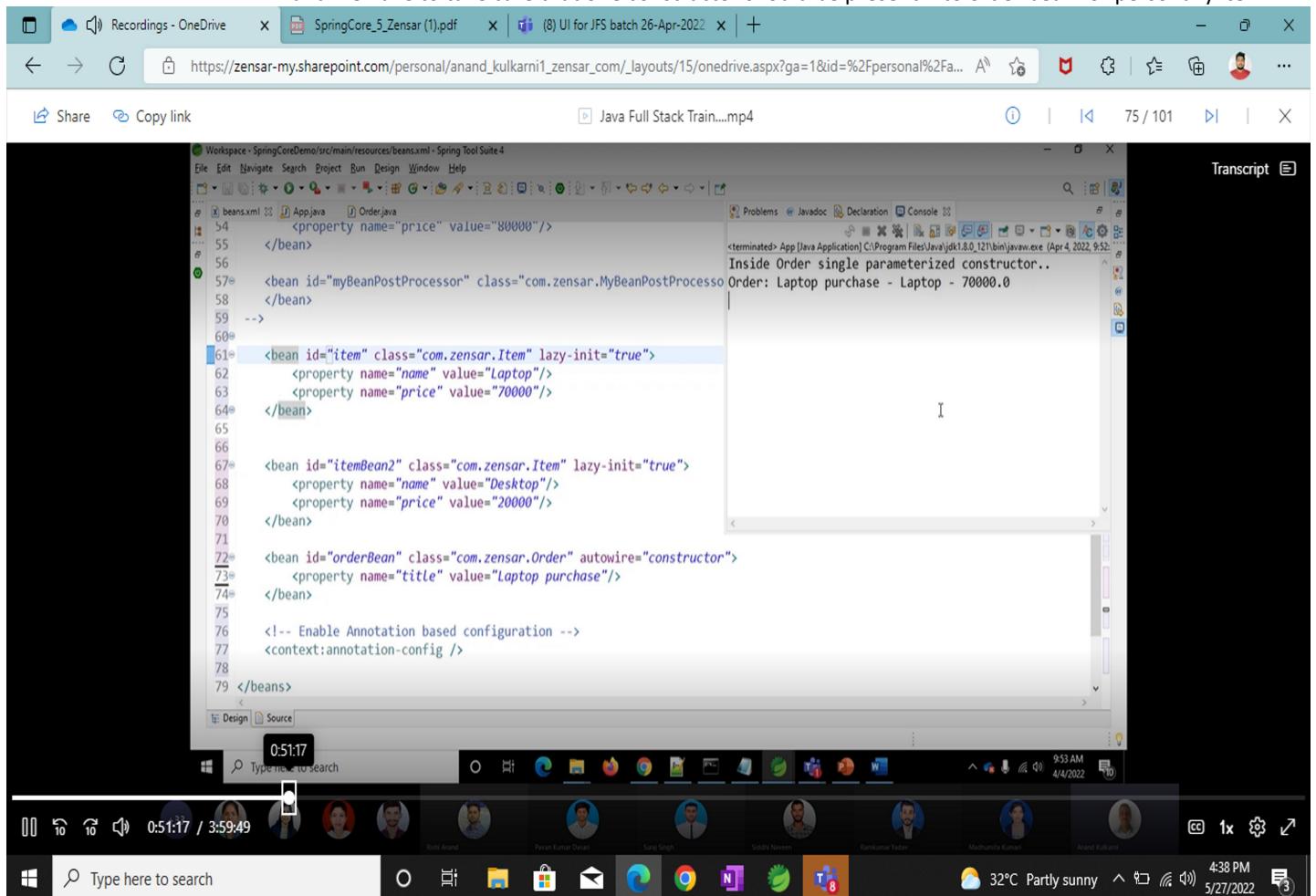


2. **autowire byName :-** in this variable name of the class item should be same as id name of the item bean like below image



In the above image we can see that id name is same of the class variable name as a item then we can use autowire="byName".

3. **autowire by constructor :-** by constructor is combination of both byType or byName in this use simply autowire="constructor" and we have to take care that one constructor should be present into order bean for personally item.



The screenshot shows the Spring Tool Suite 4 interface with the beans.xml file open. The XML code defines several beans:

```
<beans>
    <bean id="item" class="com.zensar.Item" lazy-init="true">
        <property name="name" value="Laptop"/>
        <property name="price" value="70000"/>
    </bean>

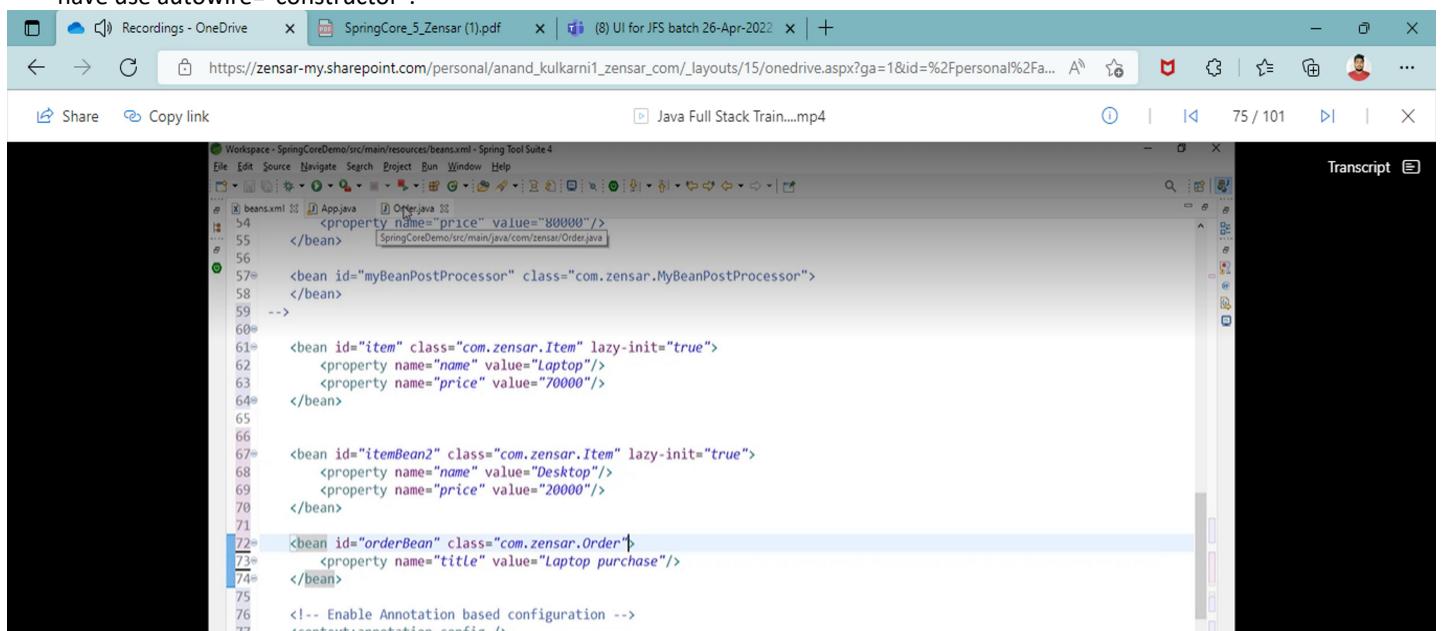
    <bean id="itemBean2" class="com.zensar.Item" lazy-init="true">
        <property name="name" value="Desktop"/>
        <property name="price" value="20000"/>
    </bean>

    <bean id="orderBean" class="com.zensar.Order" autowire="constructor">
        <property name="title" value="Laptop purchase"/>
    </bean>

    <!-- Enable Annotation based configuration -->
<context:annotation-config />
</beans>
```

A message in the Problems view says: "Inside Order single parameterized constructor.. Order: Laptop purchase - Laptop - 70000.0". The status bar at the bottom shows the time as 05:17.

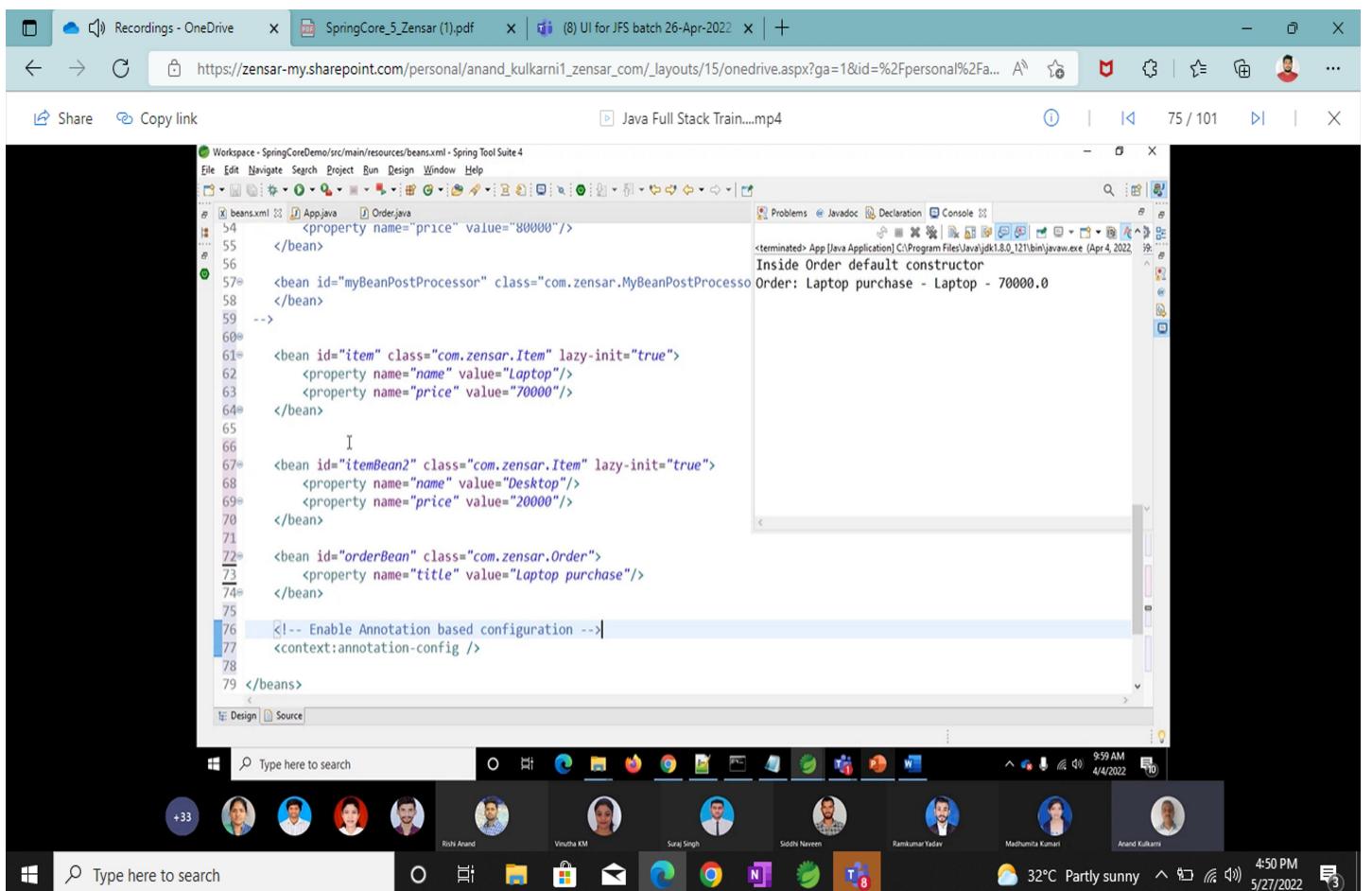
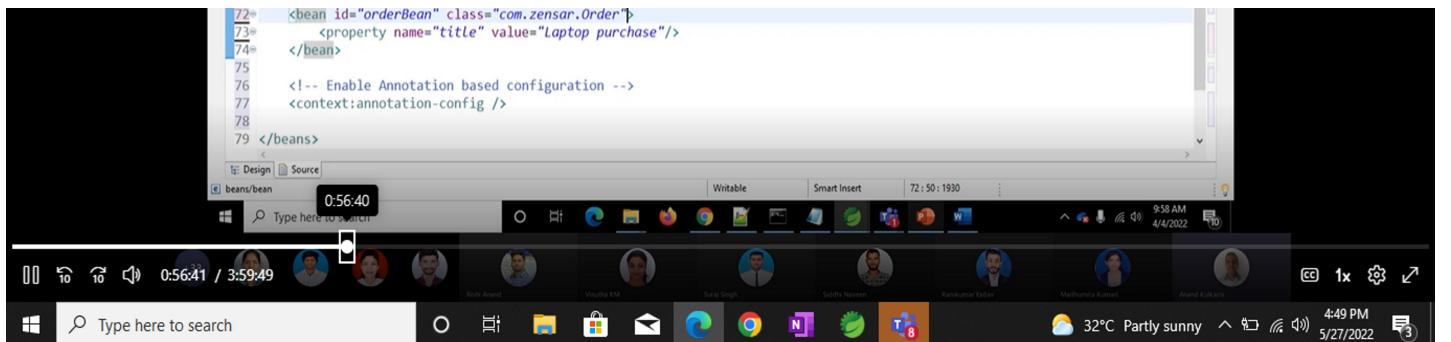
1. **@Autowired annotation:-** in this we no need to define autowire annotation into order bean we have to use this annotation into order class above to the declaration of item variable and it working is that first ioc container goes byType check that any item bean is present in xml file or not if it take to or more bean or same class then ioc container again go for byName and this annotation is applicable for each order bean by constructor if we use autowire="constructor" then it is only applicable for that bean there we have use autowire="constructor".

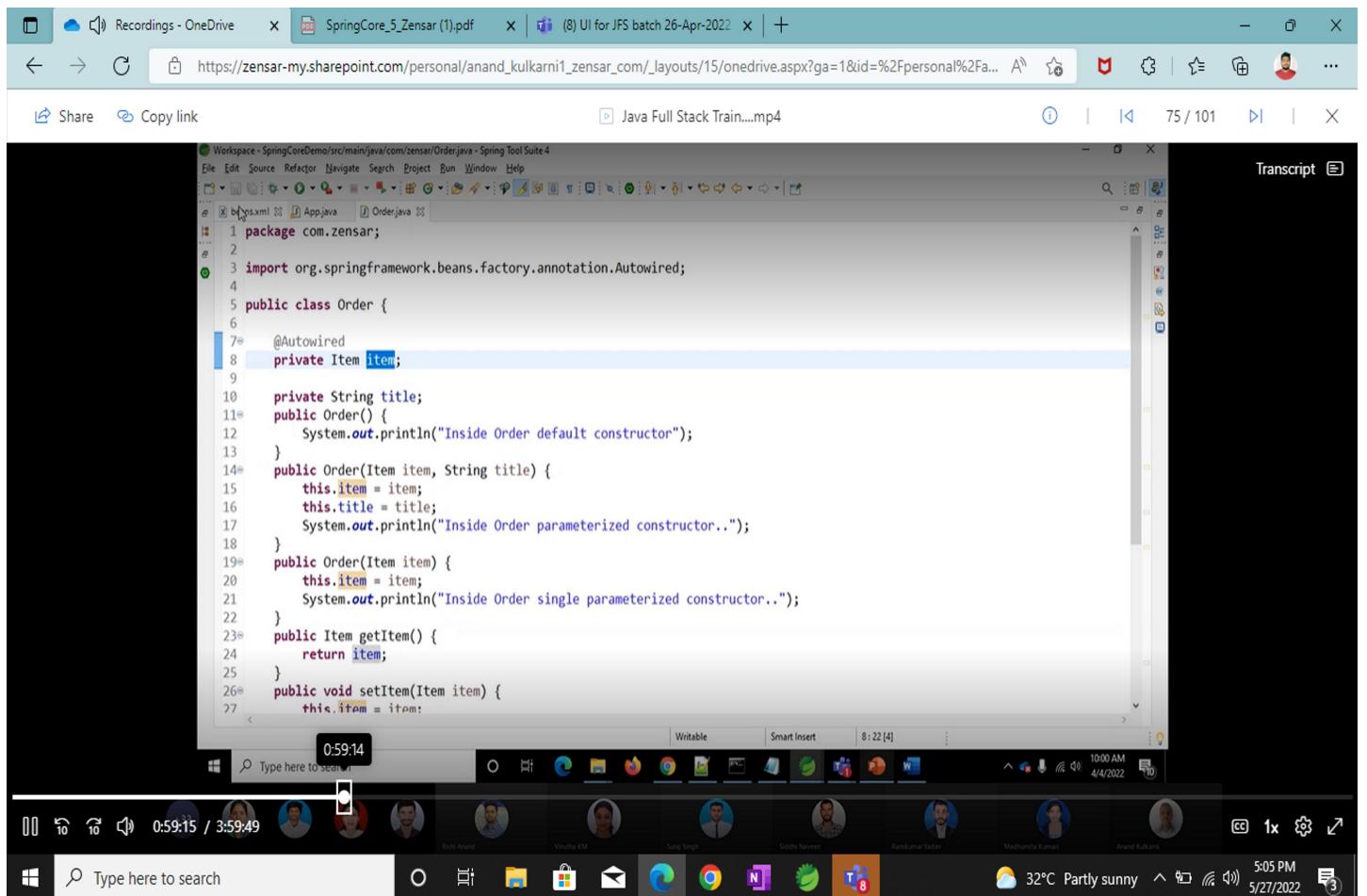


The screenshot shows the Spring Tool Suite 4 interface with the beans.xml file open. The XML code is identical to the previous one, but the Order.java class now contains an annotation:

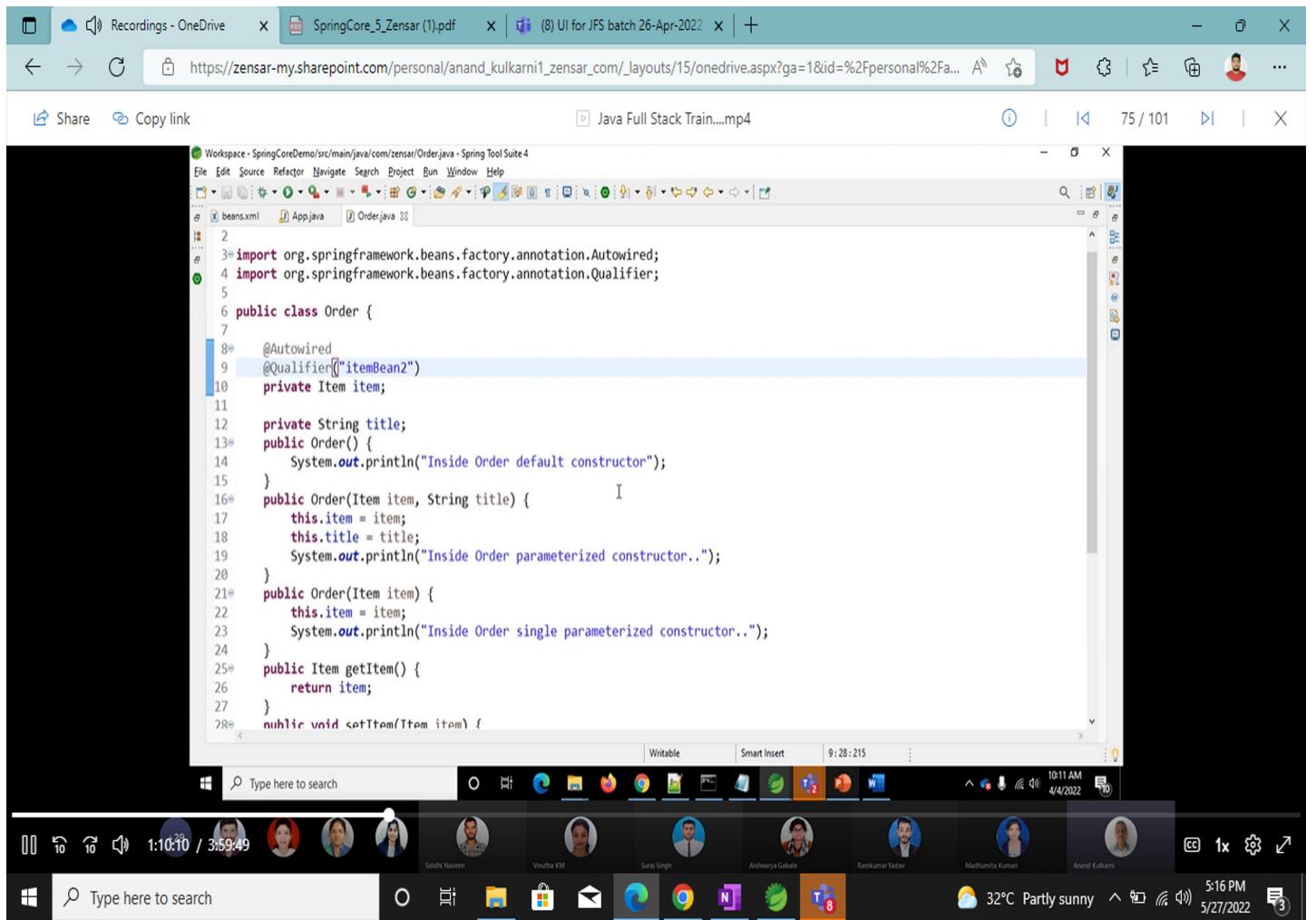
```
public class Order {
    @Autowired
    private Item item;
}
```

The status bar at the bottom shows the time as 05:17.





5. **@Qualifier annotation** :- we have seen if we use @autowire annotation into the order bean then if we have two bean of item type then we have got error that not suitable bean is find then we can go for Qualifier annotation like below ..



Whenever there is ambiguity which bean should be injected using `@Qualifier("itemBean1")` we can specify ioc container out of multiple beans which exactly we need to inject.

Recordings - OneDrive | SpringCore_5_Zensar (1).pdf | (8) UI for JFS batch 26-Apr-2022 | +

File | C:/Users/SG66203/Downloads/SpringCore_5_Zensar%20(1).pdf

38 of 39 | Page view | Read aloud | Add text | Draw | Highlight | Erase | |

Types of Autowiring

Sr. No.	Autowire type	Description
1	byName	This option enables the dependency injection based on bean names.
2	byType	This option enables the dependency injection based on bean types.
3	Constructor	Autowiring by constructor is similar to byType, but applies to constructor arguments.
4	@Autowired	autowiring can be specified in bean classes also using @Autowired annotation. To use @Autowired annotation in bean classes, you must first enable the annotation in beans.xml using <context:annotation-config />
5	@Qualifier	During 'byType' autowiring, IoC container may find more than one bean which leads to an exception. In order to specify which bean you wish to inject, we use @Qualifier(value="beanName") annotation.

Type here to search | O | G | M | E | C | F | N | P | Zensar | 32°C Partly sunny | 5:22 PM | 5/27/2022 |

If we do not want to use xml file then we can do configure that bean in other java file like AppConfig is a file like below but if in this file have two or more item bean then we can use @Primary Annotation with @Autowired annotation then conflict will not occurred.

Share | Copy link | Java Full Stack Train....mp4 | 75 / 101 |

Workspace - SpringCoreDemo/src/main/java/com/zensar/AppConfig.java - Spring Tool Suite 4

```

package com.zensar;
import org.springframework.context.annotation.Bean;
@Configuration
public class AppConfig {
    @Bean(name = "itemBean")
    public Item getItemBean() {
        Item item = new Item("Laptop", 70000.0);
        return item;
    }
    @Bean(name = "itemBean2")
    public Item getItemBean2() {
        Item item = new Item("Desktop", 20000.0);
        return item;
    }
    @Bean(name="orderBean")
    public Order getOrderBean() {
        Order order = new Order();
        order.setTitle("IT Order");
        return order;
    }
}

```

Type here to search | O | G | M | E | C | F | N | P | 11:44 AM | 4/4/2022 |

+24 | Akshay Golkonda | Satyakalatha Jagath | Madhumita Kumar | Rituja Pawar | Chirayati Avadipali | Rakkar Sutma | Anand Kulkarni | 32°C Partly sunny | 6:28 PM | 5/27/2022 |

