SPRING FRAMEWORK….

Spring boot is work on Spring framework.

Previously we use **STRUTS** -> To build Web application and use **EJB** (Enterprise Java Beans) and if we want to work with database with the help of ORM we use HIBERNATE.

What is **Spring** **Framework** -> if we want to build enterprise level application or any big application in java. Spring is a **light weight framework.**

Spring works with **POJOs** (Plain old Java Object)

When spring is launched a way back, it was only for dependency injection(1 module)

Spring is a Ecosystem in which we can many things.

Any framework need **features, good community, and good documentation**

**Servlet : ,but now we spring MVC**

When we want to work with spring MVC, it required some server to runs, let say **Tomcat is Servlet container.**

**IOC(Inversion of Control) -> It simply means you are inverting the control of object creation, As a programmer your focus should be on business logic rather than object creation(Creation of object, Maintain that object and destroy of object).**

To achieve this in spring we have **IOC** **container** imagine container as Box where we get all object, if I say you don’t have to create the object someone else have to do that and that’s’ your **spring** **framework**.

Spring will create object for you and where it keeps it inside a container which is the IoC container.

And this concept basically is principle, IoC is the principle which you follow to achieve this.

To work with this principle (IoC) we have to use **design** **pattern** called **Dependency Injection.**

Spring is not just creating the object but also **injecting** **that into your application that is your dependency injection design pattern which you use to implement the IoC principle.**

On top spring framework we got spring boot.

When spring came into existing way back it was good for the industry and also for the programmers.

But we have to do lot of configuration to even print “Hello World”. Lot of configuration means creating the project, Configuring XML file, Creating the beans and it needs lot of effort and time to create your first spring project

To ease of that process we got spring boot, which is opinionated framework, which will give you project which will run in a first go or we can say it will structure for you which you run directly. You don’t have to do much configuration.

**First Spring BOOT APP using IntelliJ**

In Eclipse we have that plugin installed using which you can easily create sprint boot project.

<https://start.spring.io/> -> Spring Initializer -> You can **create** the **structure** of spring boot in IntelliJ as in IntelliJ there is no plugin in community version.

**Let’s try to implement dependency Injection in Spring:**

SpringApplication.*run*(DemoApplication.class, args);  
// it activates your spring framework, it also has something called a container where you can create your spring beans

**Spring is responsible to create the objects and these objects are basically called beans.**

Any object which is created or managed by spring is called **beans.**

It will create object inside container. So you have to find that way to get the container and the way you can do that by using **Application** **Context**.

Spring by default say I will not create any object for you only when you mention that you want the object then only I will create. How you will tell spring to create object by using **Annotations.**

By using Annotation @**Component** -> Making your class component you are making sure that your spring knows that spring has to manage this particular object.

Those objects will create where you have @**component** on top to your class and that’s basically your dependency injection is happening. So Spring is carrying this object in the container and it is also getting injected here because we are saying getBean.

**Autowiring in Spring Boot:**

What if my class is dependent on any other class? -> By using @**Autowired** annotation we are creating wiring between two classes.

**Autowiring enables you to inject the object dependency implicitly. It internally uses setter and constructor injections.**

For Spring 6 we have at least JDK1 17.

We can ask spring to manage certain object or beans for us but for that we have to use some annotations.

When you are maven project irrespective of which IDE you are use, the project structure will remain the same.

**SPRING 6:**

**Pom.xml** help maven to understand what you need

Spring boot will have its own container which is the IOC container and Every time you ask Spring to create the objects for you, it will create the object and keep in the container.

To create container we want to use something called **Application Context.**

**ApplicationContext** is responsible to work with container

There is also one **bean** **factory** which is old one for creation of container and get the object.

ApplicatinContext provides all the features of bean factory plus some additional features.

ApplicationContext is basically a Interface.

ApplicationContext is a superset of bean factory.

// ApplicationContext is not a part of java, this is the part of Spring  
// So we have to some packages that means we have to add spring dependencies...

To add dependencies we have to go to pom.xml. <https://mvnrepository.com/>

There’s is different ways to configuring yours Spring Project and one of them is **XML** then **java** based and last one **Annotations**.

ApplicationContext context = new ClassPathXmlApplicationContext();

**This above basically create container for you**. (then we have talk with spring it your responsibility to create object)

**Spring Bean XML Config:**

ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

**This above line is not only creating the container but also creating the object referring that spring.xml file. (Telling him go to that file create object which are mentioned with bean tag)**

**And the configuration for that container is mentioned in the spring.xml.**

Now you spring framework knows what are the classes objects this spring has to create in the container.

getbean() will give the object from container.

By default getbean will give the type of **object** as object so we have **typecaste**.

To talk with spring we can use XML, Java based Config, Annotations.

**Make sure to “spring.xml” in resources folder and resources folder be part of main folder. ->(XML based Config)**

Beans are basically object which is managed by spring framework.

Spring.xml where you mention bean tag.

**To create bean we have to use bean tag. In bean tag we have mention two things first name of bean or ID of bean and the class name (fully qualify class name means with the package as well).**

**In XML file we have to use those tag which spring understand and that you have to define the definition of beans. (The tag which you added earlier)**

The definition is four lines for spring core. You can find from Internet -> <https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html>

<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.xsd>">

<bean id = "alien" class="org.example.Alien">  
  
</bean>  
</beans>

spring.xml

So, it depends how many beans you mention here those many object it will create in spring.xml

How many times you mention this bean in your file So if you mention two beans with two different ID’s for the same class it will create two objects.

**Scopes:**

When we talk about bean in Spring it has different scope like Singleton, prototype, Request, session.

In Spring core we only Singleton and prototype.

By default it follow **singleton** means if you say getBeans() thousand of times , it will refer the **same** **object**.

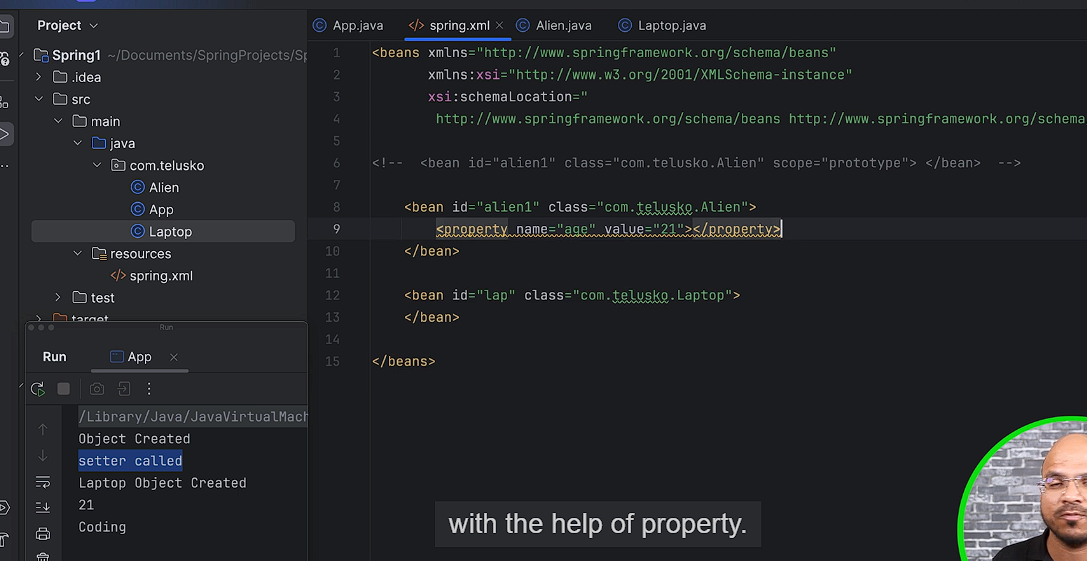
When you say **prototype** it will create new object every time you say getBean()

When the scope is **singleton** it will create the object by default so the moment you load the application it will the container, and it will get the object because singleton object will be created when the container is loading or container is loaded and then you can use any number of times.

But when you talk about **prototype** the object will created only when you say getBean().

Every time you say getBean(), it will create new object.

**Setter Injections: How to inject property?**



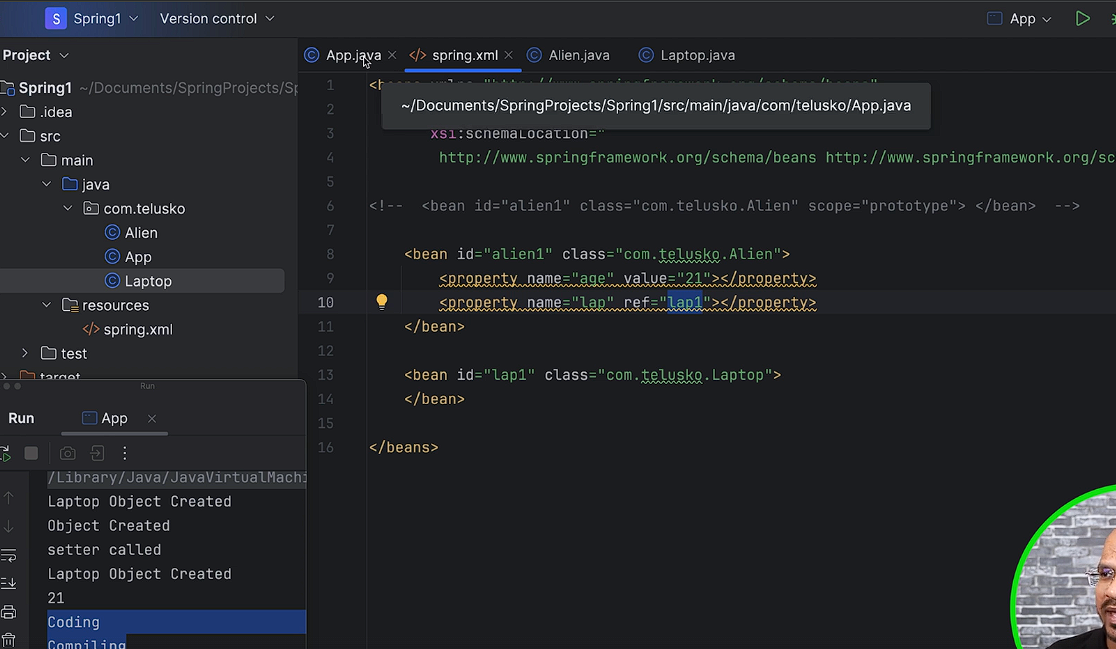
Set the value of property like age with the help of property tag. This is basically called setter injections. Because you are calling setter method to assign the value.

Value is used for primitive data type.

**Ref Attribute: What if property is reference?**

If you want to assign primitive value you can go for **value** but if you want to assign reference you can go for **ref.**

**property tag basically setter.**

****

Kind of autowiring….

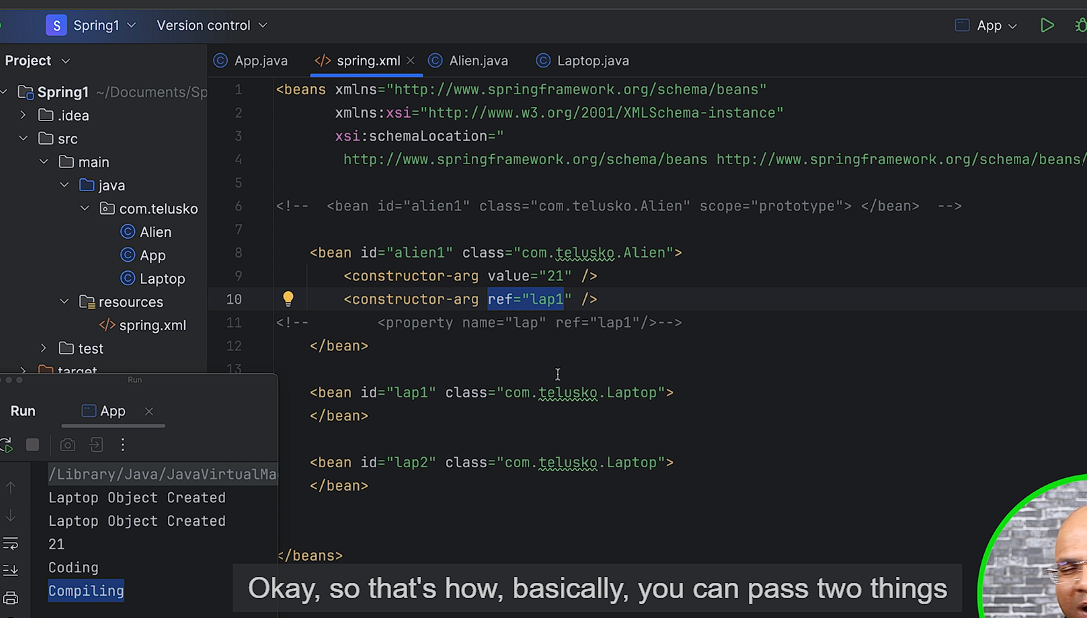
We have injected a reference with the help of **ref** attribute. This is also setter injection with the help of **ref** attribute.

**Constructor Injections:**

When we want to assign the value initially when the object is created, we prefer **constructor** **injection**. When you want to assign value later **setter** **injections** make sense.

The way **property** tag is used in setter injection, In Constructor Injection we use **constructor-arg**

Here we are injecting value with the help of Constructor injections.



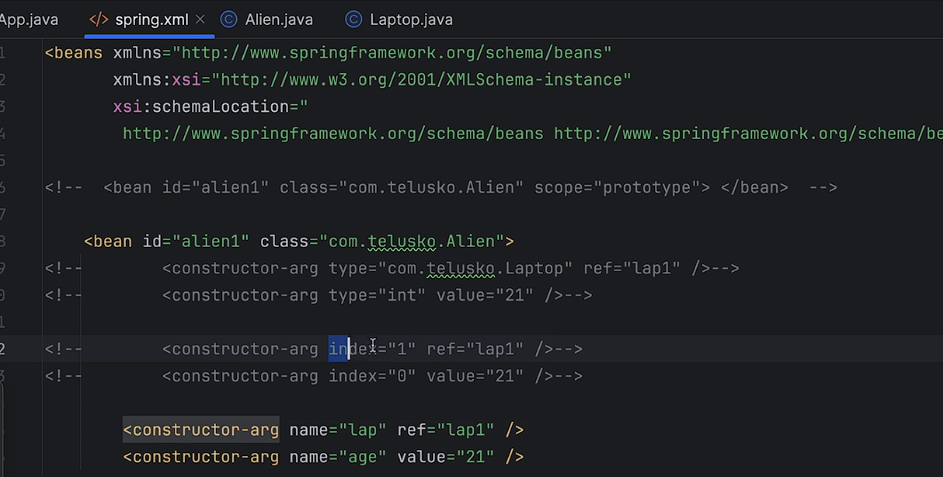
Here it follows sequence at line number 9 and 10. (It was not checking the type it was checking the sequence.)

You can also mention the type to clarify the confusion. If you don’t mention the type it will go for sequence.

But what if we have same type of data then we can go for **index and we can also name attribute.**

**The ideal solution is to use index number.**

If we want to work with **name** attribute, we have specify **@ConstructorProperties** annotations.



**Creating Interface:**

2:24

Interfaces enable polymorphism in Java**, allowing objects of different classes to be treated as if they have the same behavior**. This makes it possible to write more generic code that can be reused with different objects. We create two objects, rectangle and circle, both of which are declared as type Shape.

**Autowiring:** Refer CODE

**Primary Bean:** Refer CODE

When there is confusion between beans when we are using byType().

**Lazy Init Beans:** Lazy Initialization of the bean

When you made it lazy-init=”true”, which means the object will not created by default, only when you want to use it, it will be created.

When you have non-lazy bean dependent on a lazy bean, still it will create the object of lazy bean because someone wants it.

You can make your bean lazy it will speed up your application.

**Getbean By Type: How do we specify class type?**

**Inner Bean:** What if I want to limit particular bean only for one class.

**Java Based Configuration:**

You can also create spring project with the help of java based configurations.

If we want to configure your application with the help of java based configuration we make our config class with annotation called add **configuration**.

In XML we are using bean tag here we are using bean annotations.

We are creating the help **new** keyword in java-based configurations but we are not injecting the object Spring is injecting it. Who will call this particular method, who is managed this object spring will.

If you want spring will create the object for you, you have use **@Bean** Annotations.

**Creating, Injecting and managed** by Spring framework, we simply write the code here.

The default name of bean is **method** name.

What if we want two different objects for two different objects? [HINT: PROTOTYPE]

By default every bean will be singleton that means the moment you load the application, it will create the container and in that container the object will be available

**Component Stereotype Annotations:**

In this you can talk to your spring framework with the class metadata itself. You can mention in class that hey spring you are responsible to manage these beans.

**@Component** is stereotype annotations in Spring which helps your Spring framework to understand that this alien is class where Spring has to manage the object for it.

Your bean name is your class name but make sure your first letter is small.

You can also change your bean name just specify name next to your @Component.

If we talked about @Autowired this is basically injecting the object, so the injection is done at three levels here -> **Field injection, constructor injection and setter injection.**

When we are using @Autowired this is basically field injection

It is always prefer to write @Autowired where you have setter.

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

@Qualifier is ahead of @Primary.

<https://www.geeksforgeeks.org/spring-dependency-injection-with-example/>

**Scope and value Annotation:**

If we want to change the scope you can write @Scope() Annotations

When we want to inject value we can use @Value() Annotations.

**Spring BOOT:**

In Spring boot, we get property file So either we can use a **YAML** file or **property** files and you can specify some external values if you want.

**Different Layers:**

If we want spring to manage our class or object we have to write @**Component** on top of it.

If we full fledge application we have multiple layers.

**Client->Server->DB**

Ultimately it is the server who is responsible to accept and return data.

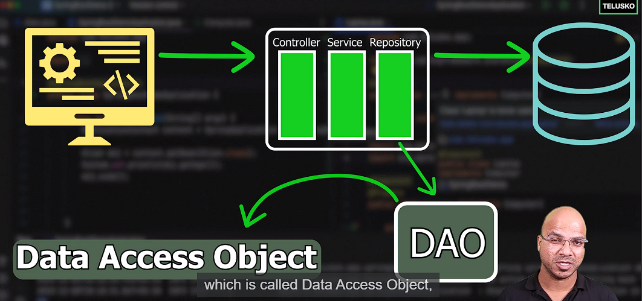
Even Server have multiple layers like **Controller, Service, Repository**

**Controller job is only work with the request.**

**Controller Job was to accept the request and send data back. (Processing job is not comes under this)**

**Service is layer who is responsible to do any kind of processing. (LOGIC)**

**DAO and Repository -> This is layer which is responsible to interact with the database. Get the data give to the Service.**

****

The job of Service layer is to do only processing part.

Make sure to use different packages for different things.

Model -> The actual entity which you are going to store in database.

Service has certain **methods** where you do some processing part.

**@Service and @Component** does the same things they both say now spring will manage the beans.

Using @Service much more sense that it is service class.

**What if you want to store data in DATABASE ?**

You should write your JDBC code in a separate class called **repository class. Where you have all the database connections. The only job of that class is to work with database.**

**Repository Layer: You have all the CRUD operations here.**