**Spring and Spring Boot Framework**

**https://github.com/navinreddy20**

**We go through on - Spring framework, Spring Boot Framework, Spring ORM, Spring AOP, Spring Security and Microservices.**

1. **Spring Framework**: What is Spring Framework?

* If we want to build enterprise-level application, want to build any big application in java we will be use some framework that is spring.
* One of the best frameworks available for developing java applications is spring. Everything will get from one framework like hibernate frameworks, Security frameworks, Testing frameworks, Messaging frameworks, build tool, logging frameworks etc. that is spring.
* Spring is an eco-system and lightweight framework.
* Spring makes java productive, reactive, simple, and modern etc.
* Using spring, we can build microservices applications, reactive, cloud, web applications, serverless applications, event driven and batch applications etc.
* We go through the <https://spring.io/projects> we can see the project which are able to use using spring.
* This is the link of official documentation of spring framework:  
  <https://docs.spring.io/spring-framework/reference/core/beans/introduction.html>

In Eclipse by default does not get the spring support, by default -> Help -> Eclipse Marketplace -> Search for spring tool 4 -> click on install -> Click on confirm - > finish we added the spring framework library to the eclipse.

The same way we add the library in to the VS code as well.

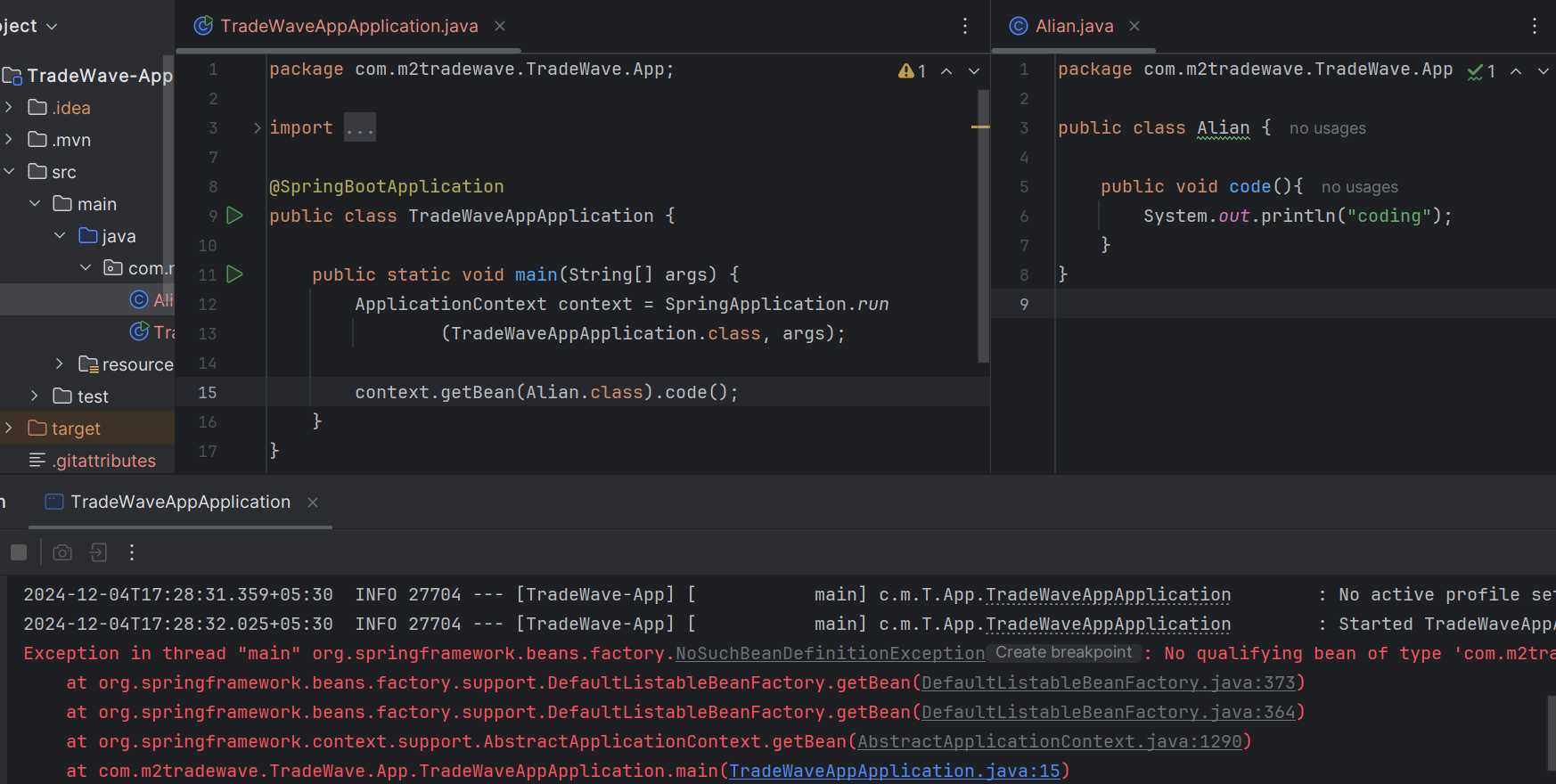
**IoC And DI (Inversion of control and dependency injection):**

* It simply means that we are inverting the control, but where and what control we are inverting?
* As we are developers **Create the object**, **Maintain the object**, and **destroy the object**. The main thing is business logic, every application has different business logics.
* As a programmer our focus should be **business logic** not on the **object creation**. We want to someone else to control the object creation. That means we are inverting a control. So, this is called **Inversion of control (IOC).**
* To achieve this in spring we have **IOC container** basically spring works in container, we don’t have to create an object someone create that is **spring framework**.
* Spring will help us to create an object, and it will keep inside the **IOC container** we will be having objects this is **IOC** and container name is **IOC container**.
* This concept is basically a principle to follow and achieve this. But how exactly we will make it work that’s why we are using design pattern called **Dependency Injection.**
* Where we have the objects in IOC container, but someone must inject into our application. So, here the **Dependency Injection** design which we use to implement the IOC principle.

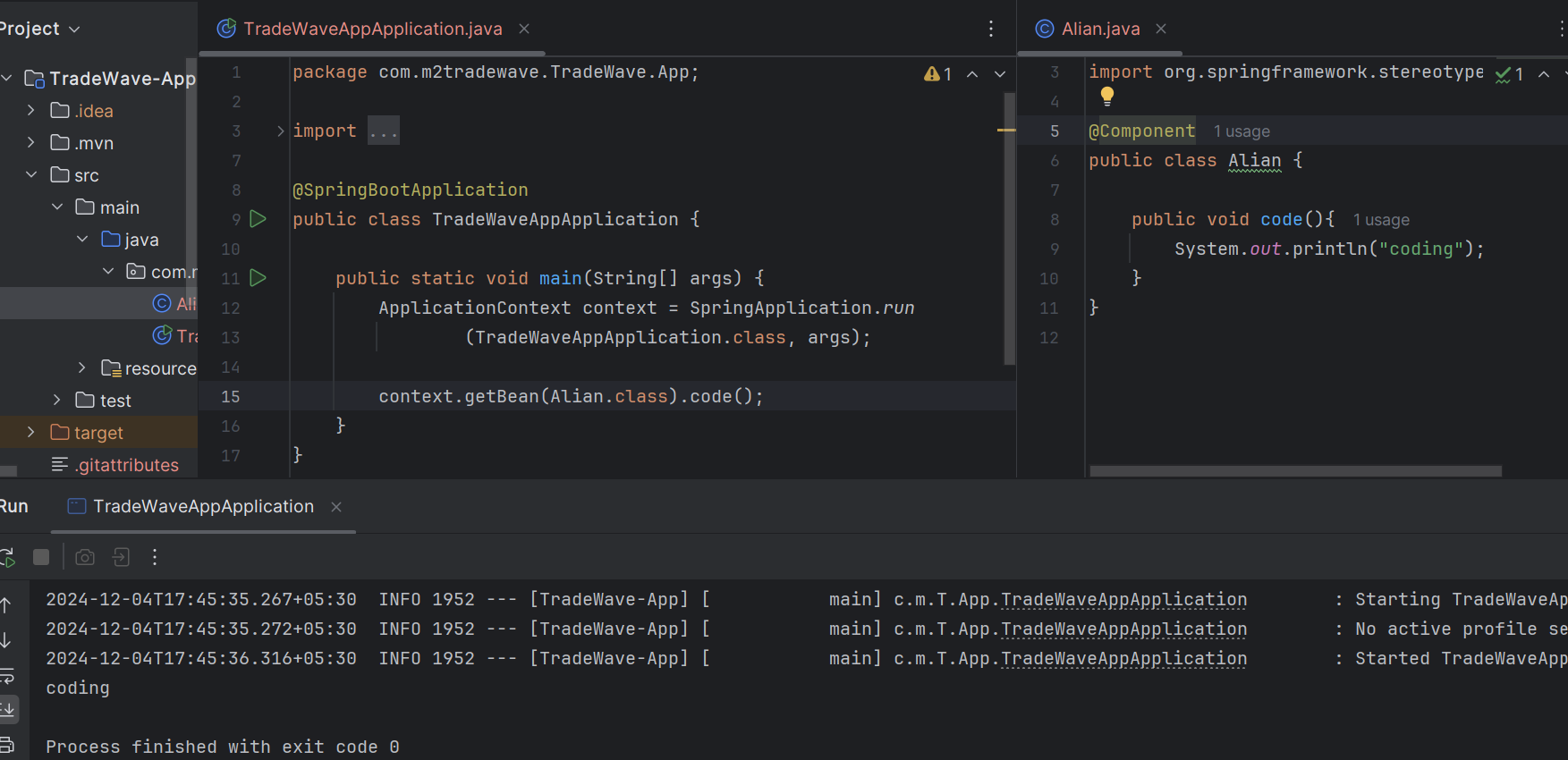
So, IOC is a principle and Dependency Injection is a design pattern using which we can do that.

**Spring and Spring Boot framework:**

* Spring is a framework on top of that we got spring boot framework.
* In spring boot main method, we have run method, basically it activates our spring framework, it also has something called a container where we can create a spring bean.
* As we discussed before spring is responsible to create the object and these objects are basically called as beans. So, any objects created and managed by spring called beans.
* Every time our spring create an object that will be available in the container. This is we must find the to get the object into that container. The way we find the solution is **Application.context**.
* Ex: we can see the inside run method returns -> object of **ConfigurableApplicationContext** -> we get inside this object -> it extends **ApplicationContext** -> This means we can simply use run method; return value we got application context object.
* This means, now we have a way communicate with IOC container.
* How will we communicate? We communicate hey container give me the object; object means in spring bean. So, we have to say getBean (here we mentioned which class object we want).
* The IOC container will be able to give this object only if the object is there in the container



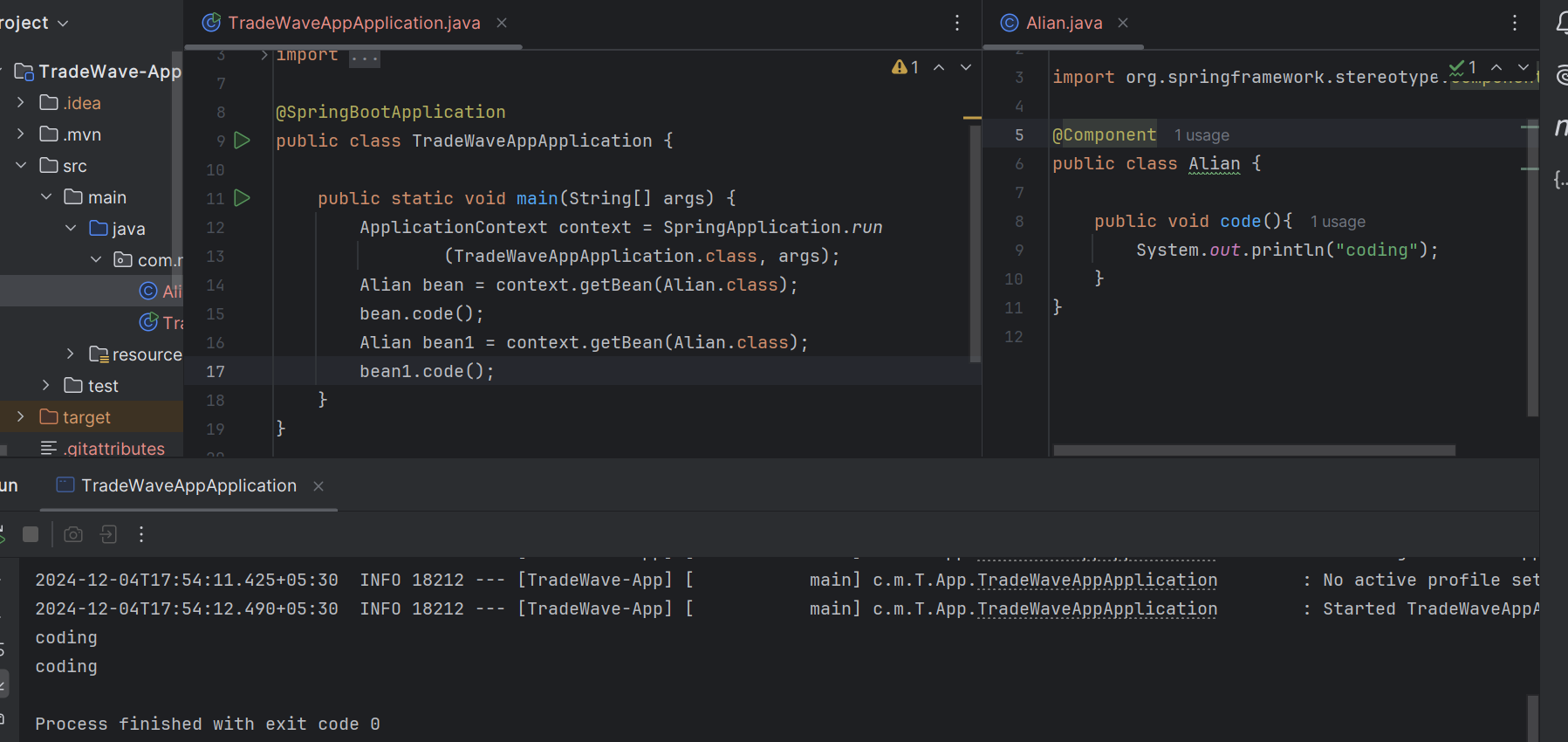
We are facing an **NoSuchBeanDefinitionException** we handle this exception we have add an annotation in the Alian class, we are making this **@Component** annotation this object fetched.



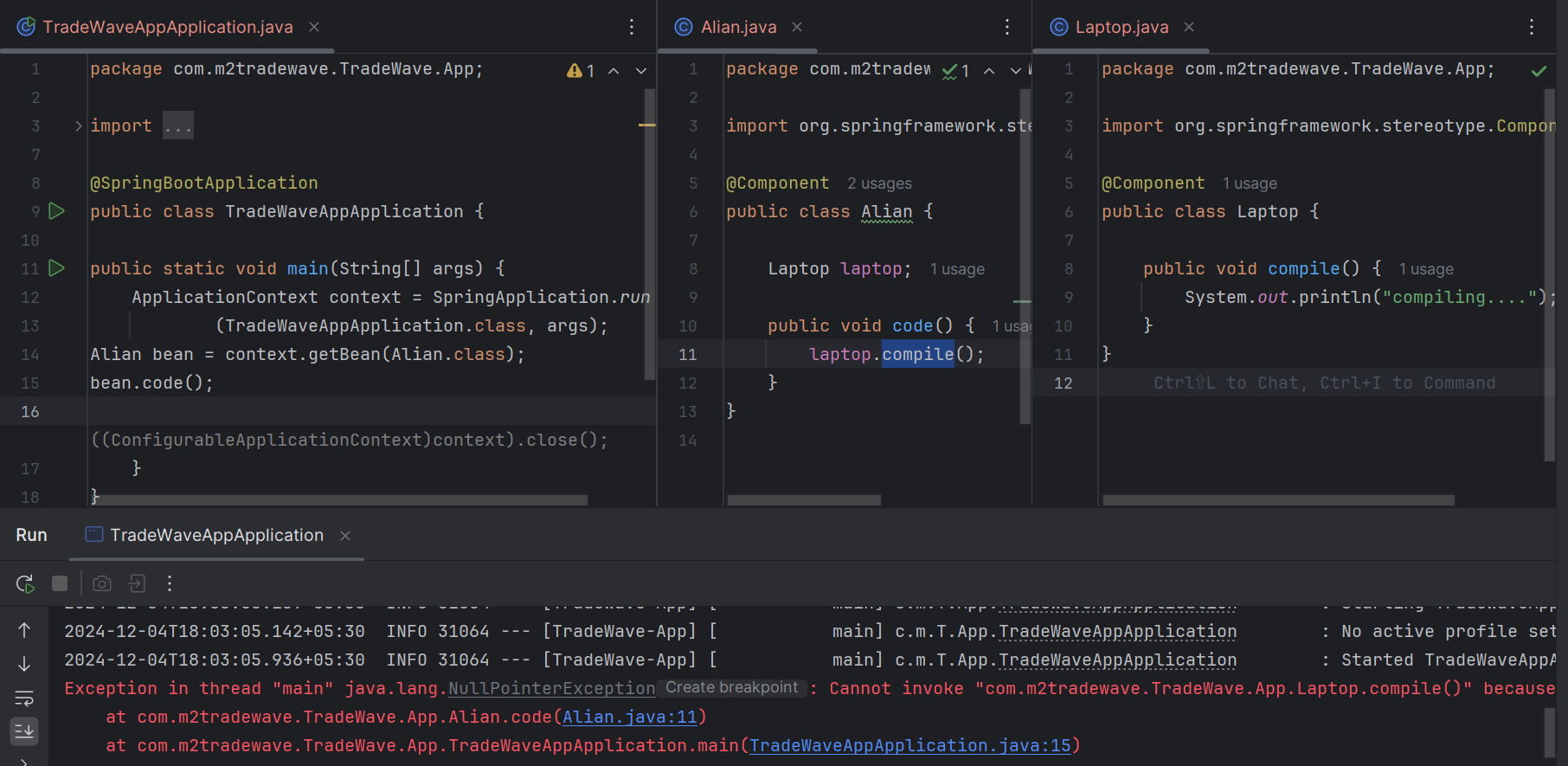
In this example we are not creating object spring is doing it just we add @Component annotation. Then spring see the component annotation and understand need to create an object, finally it creates.

This is the example our dependency injection is happening. Spring is taking the object into the container and injected to the main class where we are used getBean () method.

Can we create this object multiple times? Yes, but we are getting same or different objects.



In this example we are getting an **NullPointerException** in laptop class. We have configured the is coming from the spring framework, by default spring boot will not do everything for us we have to mention something.



That’s where we have a concept called **Wiring**. Just we are adding a @Autowired annotation we are getting wiring between Alian and Laptop class, now our spring boot framework knows it is there responsible to search for the object inside the container.

When we are using Alian in the main class, we can use **Context** directly because we have the access but apart from the main whenever we want to use this object creation or object accessing, we must use @Component and @Autowired.

**Spring Boot This is for another course**

1. **What are the two different spring bean scopes?**
2. Singleton
3. Prototype.

* **Singleton**: It is where only one instance of a bean will be created for the entire IOC container and the same instance will be injected wherever required. It is a default scope.
* **Prototype**: If we configure the prototype bean then multiple instances of the beans will be created and injected wherever required.

1. **when we use Singleton and when prototype.**

* If our application is needs statelessness, or if our application classes are being stateless, we can go with the Singleton. Like controllers DAOs etc.
* which do not have any state within them and if they have state, then will have to go with the prototype scope. Because if we have state with singleton that datacan be corrupted across multiple threads, but we don't have to worry multiple threads in prototype because each thread will it gets own instance of a bean.

1. **Can we prototype bean will be injected into a singleton bean?**

yes, Once the prototype bean is injected into the singleton bean at runtime, the same instance of the prototype will be used by the singleton bean.

1. **What are different spring bean HTTP context scopes?**

* request
* session and
* global

1. **request**: A request scope is where a new bean instance will be created for every HTTP request coming in.
2. **session**: A session scope is where there will be only one bean instance used across the session.
3. **global**: A global or global session makes only if we are using potlets within our application and this scope applied across the potlets like a global session and same bean will be used across the global sessions or potlets.
4. **Why using spring boot? what are advantages of using the spring boot**

* Auto configuration: This means we no longer need to write a lot of xml or java-based configuration to configure spring MVC to create the web layer or ORM using hibernate spring data jpa etc.
* Once we add the required dependency the Dispatcher servlet will be configured for our web layer a data source will create for our ORM layer if we use the spring data jpa and Transaction Manager is created and injected as well. without as writing code.
* We don't need to worry about Module Availability and version compatible across these library or modules Spring Boot Starters provided by the spring development team these starters are ready to be used in our projects are:
* Spring-Boot-Starter-parent
* spring-boot-starter-web
* spring-boot-starter-data-jpa
* we are deploying our application, once it is ready to production to any other container, we are usable for Embedded-servlet-container like: Tomcat, Jetty, Undertow using this server we can easily launch our application.
* Spring boot also offers Spring Boot Actuators which allow us to do health checks we are adding the dependency of Spring Boot Actuators it will gives us to
* autoconfigure
* mappings
* info
* health and
* metrics

1. **What is the spring boot actuator?**

* It provides insights about the spring boot applications once it’s up and running, it uses application context information the autoconfiguration that were enabled, the MVC mapping and several other metrics
* **Actuator** - It exposes various end points starting with, /actuator when we hit this endpoint this is like a discoverypage for all the other rest endpoints provided by actuator.
* **Autoconfigure** - It will give us all the automatic configurations that spring boot as enabled for our application.
* **Congigprops** - It will give us all the configurations’ properties we have configured for our applications.
* **Beans** - It will show us and all the beans that are created by spring.
* **Dump** - Thread dump of our application.
* **Health** - It shows the status is the application is up or down.
* **Info** - will give us all the information that we have provided for the application for others to understand what this application does.
* **logfile** - will help us retrieve the logfile at any point provided. we set a property called log.file in the application.properties and point it to the log file.
* **metrics** - will give us various metrics such as memory that our application is using the disc space etc.
* **caches** - will give us the caching information of our application and ther few other endpoints etc.

**AOP (Aspect oriented programming)**

We are looking at one enterprise application is typically divided into layers are:

* UI Layer
* Business Logic Layer
* Data Access Layer and many more

All this layers typically have some non-functional requirements like:

* Security - securing our application
* Profiling - To see how our application is performing
* logging - write our errors or information’s to the log files
* transaction management - used for database management

These are called cross-cutting concerns, these are required across this layer as well as across the application. These are all applied to our application at the runtime doing that will have several advantages are:

* **Cross Cutting Concerns** - we can address all the requirements which are common across the enterprise or even enterprises.
* **Reuse** - allowing us to reuse.
* **Quick Development** - We can focus on our business logic without worring about the non-functional requirements.
* Focus on one aspect
* Enable/Disabled at the runtime.

1. What are the four important terminologies of aspect-oriented programming?

* **Aspect** - It is a plan java class that can contains a number of advice and pointcuts, this class is where we address a particular cross cutting concerns for our application this could be security, transaction management, logging etc.
* **Advice** - Aspect is made up number of advises advice is a method that address a part of the concern
* **Pointcut** - It provides an expressional language it’s like a regular expression language to match a particular join point, it gives as syntax to express join points.
* **Joinpoints** - It is a point in the java program where the advice needs to be applied this could be a method or field or a constructor once we develop an aspect a lot of methods, we need to apply those methods to the other classes and then methods in our java application that is where joinpoints come in.