**Interview Question🡪**

# Spring Boot🡪

## What is Spring Boot?

Spring Boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework.

It is used to create standalone spring based application that you can just run because it needs very little spring configuration.

Spring boot is used to develop the REST API.

**Advantage🡪**

* Create stand-alone Spring applications that can be started using java -jar.
* Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.
* It provides opinionated 'starter' POMs to simplify your Maven configuration.
* It automatically configure Spring whenever possible.

## Spring boot Feature

* Web Development
* SpringApplication
* Application events and listeners
* Admin features

## 1.3 @SpringbootApplication

**@SpringBootApplication=@ComponentScan+@EnableAutoConfiguration+@Configuration**

## Spring Boot Annotations

Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide supplemental information about a program. It is not a part of the application that we develop. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

#### **@SpringBootApplication:**

 It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.

#### **@EnableAutoConfiguration:**

 The @EnableAutoConfiguration annotation enables Spring Boot's auto-configuration feature. This feature automatically configures the application by scanning the classpath components and registering the beans.

#### **@Configuration:**

It indicates that the annotated class declares one or more bean definitions and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

#### @ComponentScan

It enables component scanning to automatically discover and register Spring-managed components like @Component, @Service, @Repository, and @Controller in the specified base package and its sub-packages.

#### **@Required:**

It applies to the **bean** setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception **BeanInitilizationException**.

#### **@Autowired:**

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

It internally uses setter or constructor injection.

#### **@Component:**

@Component is a stereotype annotation used to indicate that a class is a Spring-managed component. It's a generic stereotype for any Spring-managed component, whereas more specific stereotype annotations like @Repository, @Service, and @Controller are used to indicate specific types of components.

#### **@Service:**

It is also used at class level. It tells the Spring that class contains the **business logic**.

#### **@Repository:**

It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

#### **@RequestMapping**

 It allows to define how incoming requests should be handled and which methods should process them. also define the behaviour of your application's endpoints.

## REST API Annotation

##### @Controller

* @Controller is a general annotation used to define a class as a Spring MVC controller.
* It is typically used for traditional web applications where the response is often a view (HTML page) rendered by a templating engine like Thymeleaf or JSP.
* Methods annotated with @RequestMapping, @GetMapping, @PostMapping, etc., inside a @Controller class handle HTTP requests and return a view.

##### @RestController

* @RestController is a specialized version of @Controller that is used to define RESTful web services.
* It combines @Controller and @ResponseBody, meaning that every method annotated with @RequestMapping, @GetMapping, @PostMapping, etc., returns the response directly as JSON or XML data without rendering a view.
* @RestController is commonly used in modern web applications to build RESTful APIs.

##### @GetMapping:

It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**

##### @PostMapping:

It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**

##### @PutMapping:

It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**

##### @DeleteMapping:

It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**

##### @PatchMapping:

It maps the **HTTP PATCH**requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**

##### @RequestBody:

 It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.

##### @ResponseBody:

It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

##### @PathVariable:

It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

##### @RequestParam:

It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

##### @RequestHeader:

 It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.**For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

##### @RequestAttribute:

 It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

## What is Spring boot Actuator?

Spring Boot provides actuator to monitor and manage our application. Actuator is a tool which has HTTP endpoints. when application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

## 1.6 **Cross-cutting concerns**

The responsibility of each layer is different, but there are a few common aspects that apply to all layers are **Logging, Security, validation, caching,** etc. These common aspects are called **cross-cutting concerns.**

If we implement these concerns in each layer separately, the code becomes more difficult to maintain. To overcome this problem, **Aspect-Oriented Programming** (AOP) provides a solution to implement cross-cutting concerns.

## AOP (Aspect object Programming)

AOP **(Aspect-Oriented Programming)** is a programming pattern that increases modularity by allowing the separation of the **cross-cutting concern**. These cross-cutting concerns are different from the main business logic. We can add additional behaviour to existing code without modification of the code itself.

For more check <https://www.javatpoint.com/spring-boot-aop>

## Autowiring🡪 @Autowired

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

It internally uses setter or constructor injection.

## 1.8. What is dependency Injection?

The process of injecting dependent bean objects into target bean objects is called dependency injection.

* Setter Injection: The IOC container will inject the dependent bean object into the target bean object by calling the setter method.
* Constructor Injection: The IOC container will inject the dependent bean object into the target bean object by calling the target bean constructor.
* Field Injection: The IOC container will inject the dependent bean object into the target bean object by Reflection API.

## 1.9. What is an IOC container?

IoC Container is a framework for implementing automatic dependency injection. It manages object creation and its life-time and also injects dependencies into the class

## 1.10 what is beans

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.

**Configuration**: It indicates that the annotated class declares one or more bean definitions and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

Auto-Configuration: It enables Spring Boot's automatic configuration mechanism, which automatically configures beans based on the dependencies and the content of the application context.

Component Scanning: It enables component scanning to automatically discover and register Spring-managed components like @Component, @Service, @Repository, and @Controller in the specified base package and its sub-packages.

@Component: is a stereotype annotation used to indicate that a class is a Spring-managed component. It's a generic stereotype for any Spring-managed component, whereas more specific stereotype annotations like @Repository, @Service, and @Controller are used to indicate specific types of components.

@RequestMapping: It allows to define how incoming requests should be handled and which methods should process them. also define the behavior of your application's endpoints.

@Transactional is an annotation used in Spring Boot to declare transactional behavior on methods or classes. When you annotate a method or class with @Transactional, Spring Boot manages transactions for the annotated components, ensuring that they are executed within a transactional context.