Spring-boot

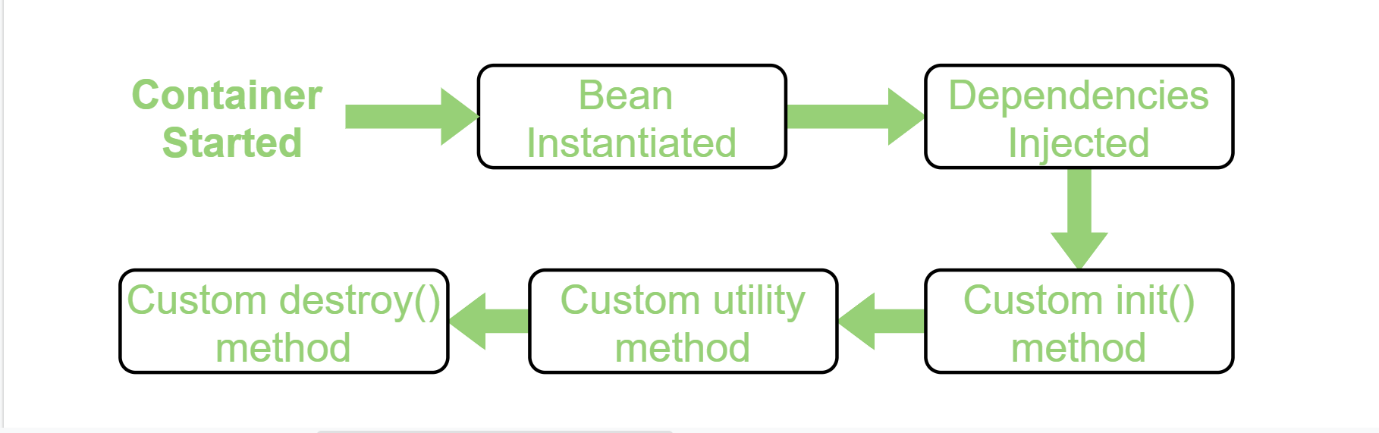
\*what is spring-boot?

Spring Boot is a Java framework that makes it easier to create and run Java applications. It simplifies the configuration and setup process, allowing developers to focus more on writing code for their applications.

\*what is bean?

\*what is beanlife cycle? The lifecycle of any object means when & how it is born, how it behaves throughout its life, and when & how it dies.

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started.



\*What is MVC architecture?

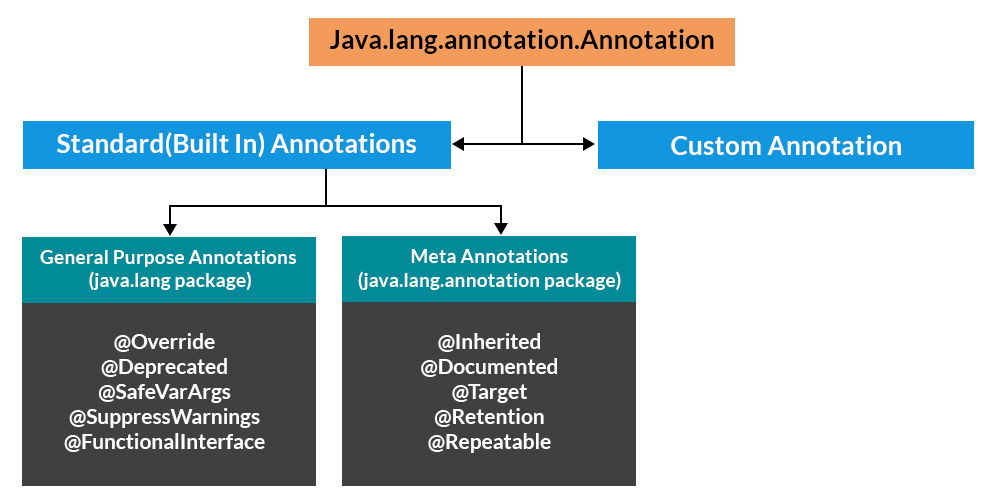
Model-View-Controller (MVC) is a design pattern used to separate concerns within an application, allowing for a more modular and maintainable codebase. It divides an application into three interconnected components:

1. **Model:** Represents the data and business logic of the application. It directly manages the data, logic, and rules of the application. The model is responsible for retrieving data from the database and performing operations on it.
2. **View:** Represents the UI (User Interface) of the application. It is responsible for displaying the data provided by the model in a specific format. The view interacts with the model to display data to the user and also receives user input.
3. **Controller:** Acts as an intermediary between the model and the view. It receives input from the user through the view, processes it (often by calling the appropriate method on the model), and then returns the output display to the view.

\*what are annotations?

Annotations are used to provide supplemental information about a program.

* Annotations start with ‘**@**’.
* Annotations do not change the action of a compiled program.
* Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.



There are broadly 5 categories of annotations as listed:

1. Marker Annotations
2. Single value Annotations
3. Full Annotations
4. Type Annotations
5. Repeating Annotation

\*What are types of annotations?

Annotations in Spring and Java are metadata that provide additional information about the code to the compiler and runtime. Here are some common annotations and their purposes:

**1. @Component**

* **Purpose:** Marks a Java class as a component or bean in the Spring context.
* **Usage:** It is a general-purpose stereotype annotation, often used to denote a class that can be automatically detected via classpath scanning.
* **Example:** @Component public class MyComponent { }

**2. @Service**

* **Purpose:** A specialized version of @Component, used to annotate service layer classes.
* **Usage:** Indicates that a class performs some business logic or service-related tasks.
* **Example:** @Service public class MyService { }

**3. @Repository**

* **Purpose:** Another specialization of @Component, used to annotate Data Access Object (DAO) classes.
* **Usage:** It provides additional functionalities like exception translation.
* **Example:** @Repository public class MyRepository { }

**4. @Controller**

* **Purpose:** A specialization of @Component, used to define a controller class in the Spring MVC framework.
* **Usage:** It is used to handle HTTP requests.
* **Example:** @Controller public class MyController { }

**5. @RestController**

* **Purpose:** Combines @Controller and @ResponseBody.
* **Usage:** Used to create RESTful web services by returning data instead of views.
* **Example:** @RestController public class MyRestController { }

**6. @Entity**

* **Purpose:** Specifies that a class is an entity and is mapped to a database table.
* **Usage:** Used in JPA (Java Persistence API) to define database entities.
* **Example:** @Entity public class MyEntity { @Id private Long id; }

**7. @Table**

* **Purpose:** Specifies the table name in the database that the entity is mapped to.
* **Usage:** Used in conjunction with @Entity to map an entity to a specific table.
* **Example:** @Entity @Table(name = "my\_table") public class MyEntity { }

**8. @Autowired**

* **Purpose:** Allows Spring to resolve and inject collaborating beans into your bean.
* **Usage:** Used for dependency injection.
* **Example:** @Autowired private MyService myService;

**9. @Qualifier**

* **Purpose:** Used along with @Autowired to avoid ambiguity when multiple beans of the same type exist.
* **Usage:** Specifies which bean to inject when there are multiple candidates.
* **Example:** @Autowired @Qualifier("myBean") private MyService myService;

**10. @RequestMapping**

* **Purpose:** Used to map web requests to specific handler classes or methods.
* **Usage:** Defines the URL pattern for a controller method.
* **Example:** @RequestMapping("/home") public String home() { return "home"; }

**11. @GetMapping, @PostMapping, @PutMapping, @DeleteMapping**

* **Purpose:** Specialized versions of @RequestMapping for specific HTTP methods.
* **Usage:** These annotations are shortcuts for @RequestMapping with method attribute.
* **Examples:**
  + @GetMapping("/items")
  + @PostMapping("/items")
  + @PutMapping("/items/{id}")
  + @DeleteMapping("/items/{id}")

**12. @Transactional**

* **Purpose:** Manages transaction boundaries.
* **Usage:** Used to ensure that a method or class has transactional semantics.
* **Example:** @Transactional public void performTransaction() { }

**13. @Configuration**

* **Purpose:** Indicates that a class is a source of bean definitions.
* **Usage:** Used to define beans and Spring configurations.
* **Example:** @Configuration public class MyConfig { }

**14. @Bean**

* **Purpose:** Indicates that a method produces a bean to be managed by Spring.
* **Usage:** Used within @Configuration classes to define beans.
* **Example:** @Bean public MyBean myBean() { return new MyBean(); }

**15. @Value**

* **Purpose:** Injects values from a property file into a Spring bean.
* **Usage:** Used to read values from configuration properties.
* **Example:** @Value("${my.property}") private String myProperty;

**16. @Scope**

* **Purpose:** Specifies the scope of a bean.
* **Usage:** Defines how a bean is created and managed.
* **Example:** @Scope("singleton") public class MyBean { }

These annotations are fundamental in Spring-based applications for defining and managing the various components and behaviors.

\*what are annotations needed in department model to provide string methods,gettersetters,allconstructors,noconstructors with detailed explanation?

 **@Entity**

* Marks this class as a JPA entity, meaning it represents a table in a relational database.
* The Department class will be mapped to a database table, with each instance of the class representing a row in that table.

 **@Data**

* A Lombok annotation that automatically generates boilerplate code like getters, setters, toString(), equals(), and hashCode() methods.
* This simplifies your code by removing the need to manually write these methods.

 **@AllArgsConstructor**

* Another Lombok annotation that generates a constructor with parameters for all fields in the class.
* Useful when you want to create an instance of the Department class and set all of its fields in one step.

 **@NoArgsConstructor**

* Lombok annotation that generates a no-argument constructor.
* This is required by JPA, which needs a default constructor to create an instance of the entity when fetching data from the database.

 **@Builder**

* Lombok annotation that provides a builder pattern for your class.
* This allows you to create instances of the Department class in a more readable and flexible way, especially useful when the class has many fields.

**@Id and @GeneratedValue(strategy = GenerationType.IDENTITY)**:

* @Id: Marks the field as the primary key.
* @GeneratedValue(strategy = GenerationType.IDENTITY): Specifies that the primary key should be generated automatically by the database.

### \* what is dependency injection? Key Concepts of Dependency Injection

1. **Dependency**: A dependency is any object that another object requires to function. For example, if a Car class needs an Engine to operate, the Engine is a dependency of the Car.

### Injection: Instead of the Car class creating its own Engine object, an external source (often a DI framework) provides the Engine to the Car. This "injecting" of dependencies can happen through constructors, setters, or directly into fields. Types of Dependency Injection

1. **Constructor Injection**: Dependencies are provided through a class's constructor.

java

Copy code

public class Car {

private Engine engine;

public Car(Engine engine) {

this.engine = engine;

}

public void start() {

engine.run();

}

}

Here, the Engine dependency is injected into the Car class via its constructor.

1. **Setter Injection**: Dependencies are provided through setter methods.

java

Copy code

public class Car {

private Engine engine;

public void setEngine(Engine engine) {

this.engine = engine;

}

public void start() {

engine.run();

}

}

The Engine is injected into the Car class using the setEngine method.

1. **Field Injection**: Dependencies are injected directly into fields. In frameworks like Spring, this is often done using annotations.

java

Copy code

public class Car {

@Autowired

private Engine engine;

public void start() {

engine.run();

}

}

In this example, the Engine dependency is injected directly into the engine field by the DI container.

### \*  What does the @SpringBootApplication annotation do internally?

The **@SpringBootApplication**annotation combines three annotations. Those three annotations are: **@Configuration, @EnableAutoConfiguration,**and **@ComponentScan**.

* **@AutoConfiguration**: This annotation automatically configuring beans in the class path and automatically scans the dependencies according to the application need.
* **@ComponentScan**: This annotation scans the components (@Component, @Service, etc.) in the package of annotated class and its sub-packages.
* **@Configuration:**This annotation configures the beans and packages in the class path.

@SpringBootApplication automatically configures the application based on the dependencies added during project creation and bootstraps the application by using run() method inside the main class of an application.

*@SpringBootApplication = @Configuration +*

*@EnableAutoConfiguration + @ComponentScan*

### *\** What is Spring Initializr?

**Spring Initializer**is a tool that helps us to create skeleton of spring boot project or project structure by providing a maven or gradle file to build the application. It set up the framework from scratch.

### \* ****How does a spring application get started?****

A Spring application gets started by calling the **main()**method with **@SpringBootApplication**annotation in the **SpringApplication**class. This method takes a SpringApplicationBuilder object as a parameter, which is used to configure the application.

### \*  What are the Spring Boot Starter Dependencies?

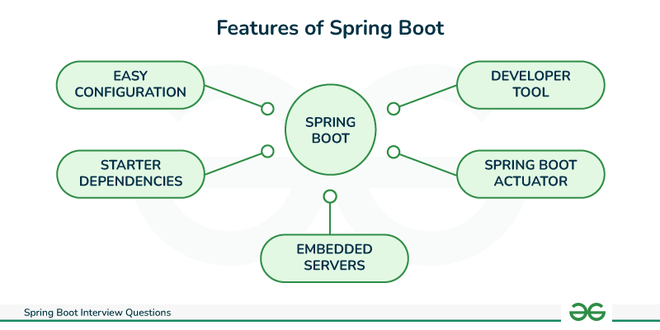
Spring Boot provides many starter dependencies. Some of them which are used the most in the Spring Boot application are listed below:

* Data JPA starter
* Web starter
* Security starter
* Test Starter
* Thymeleaf starter

### What are the Features of Spring Boot?

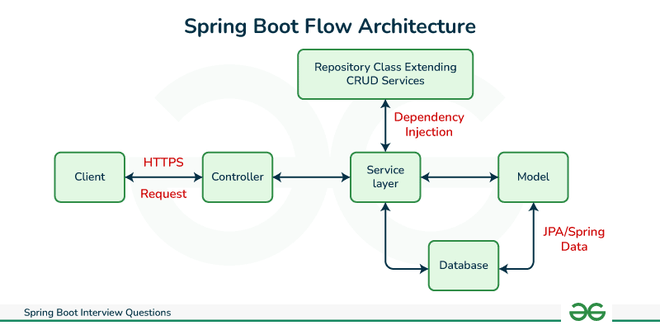
There are many useful features of Spring Boot. Some of them are mentioned below:

* **Auto-configuration –**Spring Boot automatically configures dependencies by using **@EnableAutoconfiguration**annotation and reduces boilerplate code.
* **Spring Boot Starter POM**– These Starter POMs are pre-configured dependencies for functions like database, security, maven configuration etc.
* **Spring Boot CLI (Command Line Interface)**– This command line tool is generally for managing dependencies, creating projects and running the applications.
* **Actuator –**Spring Boot Actuator provides health check, metrics and monitors the endpoints of the application. It also simplifies the troubleshooting management.
* **Embedded Servers –**Spring Boot contains embedded servers like Tomcat and Jetty for quick application run. No need of external servers.



### \* Describe the flow of HTTPS requests through the Spring Boot application.

The flow of HTTPS requests through a Spring Boot application is as follows:



* First client makes an **HTTP request**( **GET, POST, PUT, DELETE**) to the browser.
* After that the request will go to the controller, where all the requests will be mapped and handled.
* After this in Service layer, all the **business logic**will be performed. It performs the business logic on the data that is mapped to **JPA (Java Persistence API)**using model classes.
* In repository layer, all the **CRUD**operations are being done for the **REST APIs**.
* A **JSP page**is returned to the end users if no errors are there.

### \* Explain @RestController annotation in Spring Boot.

**@RestController**annotation is like a shortcut to building RESTful services. It combines two annotations:

* **@Controller**: Marks the class as a request handler in the Spring MVC framework.
* **@ResponseBody**: Tells Spring to convert method return values (objects, data) directly into HTTP responses instead of rendering views.

### \* Difference between @Controller and @RestController

| **Features** | **@Controller** | **@RestController** |
| --- | --- | --- |
| **Usage** | It marks a class as a controller class. | It combines two annotations i.e. @Controller and @ResponseBody. |

### \* What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?

| **Features** | **@SpringBootApplication** | **@EnableAutoConfiguration** |
| --- | --- | --- |
| **When to use** | When we want to use auto-configuration | When we want to customize auto-configuration |
| **Entry point** | Typically used on the main class of a Spring Boot application, serving as the entry point. | Can be used on any configuration class or in conjunction with @SpringBootApplication. |
| **Component Scanning** | Includes **@ComponentScan**annotation to enable component scanning. | Does not perform component scanning by itself. |
| **Example** | @SpringBootApplication public class MyApplication { public static void main(String[] args) { SpringApplication.run(MyApplication.class, args); } } |  |

### \* What are Profiles in Spring?

**Spring Profiles**are like different scenarios for the application depending on the environment.

* You define sets of configurations (like database URLs) for different situations (development, testing, production).
* Use the **@Profile**annotation to clarify which config belongs to where.
* Activate profiles with **environment variables**or **command-line**options

### \* Mention the differences between WAR and embedded containers.

| **Feature** | **WAR** | **Embedded containers** |
| --- | --- | --- |
| **Packaging** | Contains all of the files needed to deploy a web application to a web server. | It is a web application server included in the same JAR file as the application code. |

### \*  What is Spring Boot Actuator?

**Spring Boot Actuator**is a component of the Spring Boot framework that provides production-ready operational monitoring and management capabilities. We can manage and monitor your Spring Boot application while it is running.

### \* What is the purpose of using @ComponentScan in the class files?

**@ComponentScan**annotation is used to tell Spring to scan a package and automatically detect Spring components, configurations, and services to configure. The @ComponentScan annotation can be used in the following ways:

* **Without arguments**
* **With basePackageClasses**
* **With basePackages**

### \* How to get the list of all the beans in your Spring boot application?

* Using the **ApplicationContext**object in Spring Boot, we can retrieve a list of all the beans in our application.
* The ApplicationContext is responsible for managing the beans and their dependencies.

### \* What is an IOC container?

An **IoC (Inversion of Control)**Container in Spring Boot is essentially a central manager for the application objects that controls the creation, configuration, and management of dependency injection of objects (often referred to as beans), also referred to as a DI (Dependency Injection) container.

### \* What is the difference between Constructor and Setter Injection?

| **Features** | **Constructor Injection** | **Setter Injection** |
| --- | --- | --- |
| **Dependency** | Dependencies are provided through constructor parameters. | Dependencies are set through setter methods after object creation. |
| **Immutability** | Promotes immutability as dependencies are set at creation. | Dependencies can be changed dynamically after object creation. |

### \* Explain Spring Data and What is Data JPA?

**Spring Data**is a powerful framework that can be used to develop data-oriented applications. It aims to simplify the development of data-centric applications by offering abstractions, utilities, and integration with various data sources.

* **Spring Data JPA:**This project provides support for accessing data from relational databases using JPA.

### \* Explain Spring MVC

**MVC**stands for **Model, View,**and **Controller. Spring MVC**is a web MVC framework built on top of the Spring Framework. It provides a comprehensive programming model for building web applications.

### \* What is Bean Wiring?

**Bean wiring**is a mechanism in Spring that is used to manage the dependencies between beans. It allows Spring to inject collaborating beans into each other. There are two types of Bean Wiring:

* Autowiring
* Manual wiring

### \* What is Spring Bean?

An object that is managed by the Spring IoC container is referred to as a spring bean. A Spring bean can be any Java object.