Name: Shreyas Salunke

1. **What is Dependency Injection in Spring, and how is it different from regular object creation using new?**

Dependency Injection (DI) is a design pattern where the control of creating dependencies (objects) is handed over to the framework (Spring IoC container), instead of creating them manually using the new keyword.

Real-world example: Think of a car manufacturing unit. Instead of the car assembling its own engine, the factory (Spring container) provides the engine. The car just uses it.

1. **You have two beans of the same type. What will happen if you use @Autowired without**

@Qualifier or @Primary?

Spring will throw a NoUniqueBeanDefinitionException.

Code:

@Component

public class EngineA implements Engine {}

@Component

public class EngineB implements Engine {}

@Component

public class Car {

@Autowired

private Engine engine; // Ambiguity here

}

1. **Explain how the Spring IoC container works behind the scenes when your application starts.**

*Mention key steps like bean scanning, creation, injection, etc.*

***Steps:***

1. ***Component Scanning:*** *Spring scans packages defined by @ComponentScan for beans marked with @Component, @Service, etc.*
2. ***Bean Definition:*** *Metadata for beans is created and stored in the application context.*
3. ***Bean Creation:*** *Spring instantiates beans (singleton by default).*
4. ***Dependency Injection:*** *It injects dependencies into these beans (via constructor/setter/field).*
5. ***Lifecycle Callbacks:*** *Beans with @PostConstruct, InitializingBean, etc., are triggered.*
6. ***Ready to Use:*** *The application context is fully initialized and beans are available.*
7. **You have a service class with multiple dependencies. Which injection method would you prefer (constructor, field, setter) and why?**

Preferred: Constructor Injection

Why?

* Ensures immutability and mandatory dependencies.
* Helps with unit testing (easy to mock).
* Encouraged by Spring and supported by tools like Lombok (@RequiredArgsConstructor).

1. **What is the role of @Component, @Service, @Repository, and @Controller?**

*Are they all the same? If not, how are they different?*

* @Component: Generic bean (base for all others).
* @Service: Used for business logic classes.
* @Repository: Data access layer, adds exception translation.
* @Controller: MVC controller (Web layer).

1. **Write a small Spring Boot configuration class that defines a custom bean of type**

RestTemplate using @Bean.

*(No need to import everything – focus on core logic.)*

@Configuration

public class AppConfig {

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

}

1. **What would happen if you forget to annotate your custom service class with @Component or any stereotype annotation?**

*Explain and give a short code example to demonstrate.*

The bean won't be created or managed by Spring, leading to a NullPointerException when trying to autowire it.

**Example:**

public class MyService {

public String greet() {

return "Hello!";

}

}

@Component

public class MyController {

@Autowired

private MyService myService; // Will be null

}

**Fix:** Add @Component or @Service on MyService.

1. **In application.properties, you define app.env=dev. How can you use this to conditionally load a bean only in dev environment?**

*Write a short code using @Profile.*

application.properties:

app.env=dev

spring.profiles.active=dev

*Bean Class:*

@Configuration

@Profile("dev")

public class DevConfig {

@Bean

public DataSource devDataSource() {

return new H2DataSource();

}

}

This bean will only be loaded in the dev environment.

1. **What’s the difference between @ComponentScan and @EnableAutoConfiguration in Spring Boot?**

*Explain with one practical use case where both are useful.*

* *@ComponentScan: Tells Spring where to look for @Component, @Service, etc.*
* *@EnableAutoConfiguration: Enables Spring Boot’s auto-configurations based on classpath and properties.*

*Use case:*

*@SpringBootApplication // includes both*

*public class App {}*

*@ComponentScan(basePackages = "com.my.custom.package")*

*@EnableAutoConfiguration*

*public class AppConfig {}*