**🌱 Spring Interview Questions and Answers:**

**✅ 1. What is Spring Framework?**

**Answer:**  
Spring is an open-source Java platform that provides comprehensive infrastructure support for developing Java applications. It is primarily known for **Dependency Injection (DI)** and **Aspect-Oriented Programming (AOP)**.

**✅ 2. What are the advantages of Spring Framework?**

* Loose coupling using Dependency Injection
* Easy integration with other frameworks (Hibernate, JPA, etc.)
* Simplified JDBC with Spring JDBC
* Built-in support for AOP
* Powerful MVC web framework (Spring MVC)
* Easy testing and maintainability

**✅ 3. What is a Spring Bean?**

**Answer:**  
A Spring Bean is an object that is instantiated, assembled, and managed by the Spring container. Beans are defined in the Spring configuration file or via annotations.

**✅ 4. What is the Spring Container?**

**Answer:**  
The container is the core of the Spring Framework. It creates and manages the lifecycle and configuration of application objects (Beans). It uses **Dependency Injection** to manage the components.

**✅ 5. What are the different types of Spring Containers?**

* BeanFactory – Basic container, lazy initialization
* ApplicationContext – Advanced container, supports AOP, internationalization, etc.

**✅ 6. What is Dependency Injection (DI)?**

**Answer:**  
DI is a design pattern where the dependencies of an object are injected by a framework (Spring) rather than the object creating them itself.

**✅ 7. What are the ways to inject dependencies in Spring?**

* **Constructor Injection**
* **Setter Injection**
* **Field Injection** (via @Autowired)

**✅ 8. What is the difference between @Component, @Controller, @Service, and @Repository?**

| **Annotation** | **Description** |
| --- | --- |
| @Component | Generic stereotype for any bean |
| @Controller | Used in web MVC layer |
| @Service | Used for service or business logic |
| @Repository | Used for database/DAO layer |

**✅ 9. What is @Autowired in Spring?**

**Answer:**  
It is used for automatic dependency injection by type. Spring automatically injects the appropriate bean into the class.

**✅ 10. What is the Bean lifecycle in Spring?**

1. Instantiation
2. Populate properties
3. Set bean name (aware)
4. Set bean factory (aware)
5. Pre-initialization (BeanPostProcessor)
6. Initialize bean
7. Post-initialization (BeanPostProcessor)
8. Ready to use
9. Destruction

**✅ 11. What are the different bean scopes in Spring?**

| **Scope** | **Description** |
| --- | --- |
| singleton | One instance per Spring container |
| prototype | New instance per injection |
| request | One per HTTP request (web only) |
| session | One per HTTP session (web only) |
| application | One per ServletContext |

**✅ 12. What is the difference between ApplicationContext and BeanFactory?**

* BeanFactory loads beans lazily
* ApplicationContext loads all beans at startup and provides more features like AOP, internationalization, event propagation

**✅ 13. What is the use of @Qualifier?**

**Answer:**  
When multiple beans of the same type exist, @Qualifier helps Spring choose the correct one to inject.

@Autowired

@Qualifier("myBean")

private MyService service;

**✅ 14. What is the difference between @ComponentScan and @Bean?**

* @ComponentScan automatically scans and registers beans with annotations like @Component, @Service, etc.
* @Bean is used to manually define a bean in a @Configuration class.

**✅ 15. How do you define beans in XML vs Annotation?**

**XML:**

<bean id="myBean" class="com.example.MyClass" />

**Annotation:**

@Component

public class MyClass { }

**✅ 16. What is the role of @Configuration and @Bean?**

* @Configuration: Marks a class as a source of bean definitions.
* @Bean: Used inside @Configuration to define beans manually.

**Spring Boot Interview Questions & Answers:**

**✅ 1. What is Spring Boot?**

**Answer:**  
Spring Boot is a framework that simplifies Spring application development by eliminating boilerplate code and configuration. It provides:

* Auto-configuration
* Embedded server (like Tomcat/Jetty)
* Production-ready features (Actuator, Metrics)
* Opinionated defaults

**✅ 2. What are the advantages of using Spring Boot?**

* Auto-configuration reduces configuration overhead
* Standalone and embedded server support
* Microservice-ready
* Quick development with minimal setup
* Actuator for monitoring
* Easy integration with databases, security, messaging systems, etc.

**✅ 3. What is the difference between Spring and Spring Boot?**

| **Feature** | **Spring** | **Spring Boot** |
| --- | --- | --- |
| Setup | Manual Configuration | Auto-Configuration |
| Deployment | WAR, External Server | Embedded Server (JAR) |
| Dependency Mgmt | Manual | Starter POMs (pre-defined sets) |
| Monitoring | Not built-in | Built-in via Actuator |

**✅ 4. What are Spring Boot Starters?**

**Answer:**  
Starters are pre-configured Maven or Gradle dependencies that bundle commonly used libraries for specific functionalities.

Examples:

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-security

**✅ 5. What is Spring Boot Actuator?**

**Answer:**  
A module that provides production-ready features like monitoring, health checks, metrics, etc.

Example endpoints:

* /actuator/health
* /actuator/metrics
* /actuator/env

**✅ 6. How does Spring Boot handle application configuration?**

**Answer:**

* Uses application.properties or application.yml files
* Supports environment-specific configs like application-dev.properties
* Can use @Value, @ConfigurationProperties to bind configs

**✅ 7. How to create an executable JAR in Spring Boot?**

**Answer:**  
Spring Boot uses the Maven/Gradle plugin to package the app into an executable JAR:

mvn clean package

java -jar target/app-name.jar

**✅ 8. What is @SpringBootApplication?**

**Answer:**  
It's a combination of:

* @Configuration
* @EnableAutoConfiguration
* @ComponentScan

Used as the main class annotation to bootstrap a Spring Boot application.

**✅ 9. What is Auto-Configuration in Spring Boot?**

**Answer:**  
Auto-configuration automatically sets up Spring Beans based on the libraries in the class path and your project structure.

It reduces manual bean declarations.

**✅ 10. How do you connect Spring Boot with a database?**

Add dependencies like:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

And configure DB in application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/db

spring.datasource.username=root

spring.datasource.password=pass

spring.jpa.hibernate.ddl-auto=update

**✅ 11. What is the use of Spring Initializr?**

**Answer:**  
Spring Initializr (<https://start.spring.io>) helps generate a Spring Boot project structure quickly with chosen dependencies.

**✅ 12. What is the use of @RestController in Spring Boot?**

**Answer:**  
@RestController = @Controller + @ResponseBody. It’s used to create RESTful APIs by returning JSON or XML responses directly.

**✅ 13. How do you handle exceptions in Spring Boot?**

Using:

* @ControllerAdvice
* @ExceptionHandler
* Global exception handlers for cleaner error responses

**✅ 14. How do you secure Spring Boot REST APIs?**

Common ways:

* Spring Security with JWT
* OAuth2
* API key or basic auth
* Secure endpoints using roles (@PreAuthorize, @Secured)

**✅ 15. What are Spring Boot DevTools?**

**Answer:**  
DevTools enhance development experience:

* Auto-restart on code change
* Live reload of static resources
* Better logging and cache disabling in dev mode

**✅ 16. What is CommandLineRunner and ApplicationRunner?**

They are interfaces used to run logic at the time of application startup.

@Component

public class StartupRunner implements CommandLineRunner {

public void run(String... args) {

}

}

**✅ 17. How do you monitor a Spring Boot application in production?**

Using:

* Spring Boot Actuator : for basic
* ELK Stack (Elasticsearch, Logstash, Kibana) : for centralized logging
* Prometheus + Grafana : for metrics and dashboard visualization
* Micrometer metrics

**✅ 18. How do you define custom properties in Spring Boot?**

app.name=MyApp

@Value("${app.name}")

private String appName;

**19.What is Spring AOP?**

**Answer:** Aspect-Oriented Programming allows separation of cross-cutting concerns like logging, security, transactions. It uses concepts like:

* Aspect
* Join Point
* Advice
* Pointcut
* Weaving

**✅ 20. How do you handle transactions in Spring?**

Use @Transactional to mark methods as transactional. Spring manages the transaction boundaries.

**✅ 21. What is the difference between @Autowired and @Inject?**

| **Annotation** | **Provided by** | **Notes** |
| --- | --- | --- |
| @Autowired | Spring | Allows required = false |
| @Inject | Javax (JSR-330) | Standard, fewer features |

**✅ 22. How do you secure a Spring Boot REST API?**

* Use **Spring Security**
* Add @EnableWebSecurity
* Configure HttpSecurity for endpoints
* Use **JWT** or **OAuth2** for token-based security

**✅ 23. What are some common annotations in Spring Boot?**

* @SpringBootApplication
* @RestController
* @GetMapping, @PostMapping
* @Autowired
* @Service, @Repository, @Component
* @EnableAutoConfiguration

**✅ 24. How does Spring Data JPA simplify database access?**

* Provides CRUD repository interfaces
* Auto-generates queries from method names
* Custom query support using @Query

**✅ 16. What is the use of application.properties or application.yml?**

Stores configuration like:

* Server port
* DB credentials
* Logging level
* JWT secrets

**Mock Spring Boot Interview Questions & Sample Answers:**

**✅ 1. Scenario: API Endpoint Security**

**Q:** You have an endpoint /getCustomerDetails. How will you secure it using Spring Security and JWT?

**A:**

* Add spring-boot-starter-security dependency.
* Configure security with a SecurityConfig class using HttpSecurity.
* Create JWT utility for token generation and validation.
* Use filters to extract and validate JWT on incoming requests.
* Annotate controller with @PreAuthorize("hasRole('USER')") for role-based access.

**✅ 2. Scenario: Custom Error Response**

**Q:** A client calls your API and gets a stack trace. How will you return a proper JSON error response?

**A:**

* Use @ControllerAdvice with @ExceptionHandler.
* Create a custom response object like ApiError with message, timestamp, status.
* Return meaningful error info in ResponseEntity.

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<ApiError> handleNotFound(ResourceNotFoundException ex) {

ApiError error = new ApiError(LocalDateTime.now(), ex.getMessage(), HttpStatus.NOT\_FOUND);

return new ResponseEntity<>(error, HttpStatus.NOT\_FOUND);

}

**✅ 3. Scenario: Application Slowness**

**Q:** Your Spring Boot app is slow. How do you diagnose and fix it?

**A:**

* Use Spring Actuator’s /actuator/metrics and /actuator/health endpoints.
* Enable and analyze HTTP traces, GC stats.
* Check for:
  + N+1 query problems (solve with FetchType.LAZY or @EntityGraph)
  + Unoptimized DB queries (check logs)
  + Large objects in memory
* Add caching with @Cacheable.

**✅ 4. Scenario: Environment-specific Config**

**Q:** How will you maintain different configs for dev and prod?

**A:**

* Create application-dev.properties and application-prod.properties.
* Specify active profile in application.properties or as an environment variable:

Properties

spring.profiles.active=dev

**✅ 5. Scenario: REST API with Pagination**

**Q:** How would you implement pagination in Spring Boot?

**A:**  
Use Spring Data JPA’s PagingAndSortingRepository or JpaRepository with Pageable.

@GetMapping("/customers")

public Page<Customer> getCustomers(Pageable pageable) {

return customerRepo.findAll(pageable);

}

Call with: /customers?page=0&size=5&sort=name,asc

**✅ 6. Scenario: Custom Starter Creation**

**Q:** Have you ever created a custom Spring Boot starter?

**A:** *(If not, you can say this)*  
I haven't created one from scratch, but I understand the concept — it involves packaging reusable configurations and auto-configuration classes, then exposing it as a dependency for other projects to consume.

**✅ 7. Scenario: CI/CD Integration**

**Q:** How do you deploy your Spring Boot application in CI/CD?

**A:**

* Package with Maven (.jar)
* Jenkins to build & test
* Use Docker to containerize
* Deploy to cloud (AWS EC2/ECS, Azure, GCP, or Kubernetes)
* Use Actuator for post-deploy health checks

**✅ 8. Scenario: Database Connection Issue**

**Q:** If DB credentials are incorrect, how does Spring Boot react?

**A:**

* The app throws CannotConnectException or DataSourceLookupFailureException.
* We can handle this gracefully using @PostConstruct validation or Actuator's /health/db check.

**✅ 9. Scenario: DTO vs Entity Confusion**

**Q:** Why should we use DTOs instead of directly exposing entities?

**A:**

* Hides internal structure and sensitive fields
* Prevents lazy loading exceptions during JSON serialization
* Helps shape response for API consumers
* Decouples DB model from REST API

**✅ 10. Scenario: You need to expose metrics**

**Q:** How would you expose custom metrics in Spring Boot?

**A:**

* Use Micrometer with Spring Boot Actuator
* Inject MeterRegistry and create custom counters or timers

Counter counter = meterRegistry.counter("custom\_requests");

counter.increment();