50 Java Spring Boot Interview Questions: Concept Differences Explained

Your Guide to Acing Java Spring Boot Interviews

Compiled by Puneet Joshi for Software Developers Preparing for Technical Interviews

May 27, 2025

Contents

1 Introduction

This document provides detailed answers to 50 Java Spring Boot interview questions, each focusing on the differences between key concepts. These questions are designed to help you articulate your understanding clearly and confidently in interviews. Each answer explains the concepts, their differences, and practical implications.

2 Questions and Answers

2.1 @RestController vs. @Controller

@Controller is a Spring MVC annotation for creating controllers that handle HTTP requests and return views (e.g., JSP, Thymeleaf). **@RestController** combines @Controller and @ResponseBody, designed for RESTful APIs, returning data (e.g., JSON, XML) directly. *Difference*: @RestController eliminates the need for @ResponseBody on methods, simplifying REST API development.

2.2 Spring MVC vs. Spring Boot

Spring MVC is a framework for building web applications with a Model-View-Controller pattern, requiring manual configuration for dependencies and servers. **Spring Boot** is a Spring module that simplifies setup with auto-configuration, embedded servers, and starters. *Difference*: Spring Boot reduces boilerplate, enabling faster development compared to Spring MVCs manual setup.

2.3 @Bean vs. @Component

@Component is a class-level annotation marking a class as a Spring-managed bean, auto-detected via component scanning. **@Bean** is a method-level annotation in a @Configuration class, explicitly defining a bean. *Difference*: @Component is for auto-detected classes; @Bean allows programmatic bean creation with custom logic.

2.4 Singleton vs. Prototype

Singleton scope (default) creates one bean instance per Spring container. **Prototype** scope creates a new instance each time the bean is requested. *Difference*: Singleton is shared across the application; Prototype provides unique instances, useful for stateful objects.

2.5 JPA vs. Hibernate

JPA (Java Persistence API) is a specification for ORM, defining standards for data persistence. **Hibernate** is an implementation of JPA, providing additional features like caching. *Difference*: JPA is an interface; Hibernate is a concrete ORM framework with extended functionality.

2.6 @Autowired vs. @Inject

@Autowired is a Spring-specific annotation for dependency injection, supporting @Qualifier for disambiguation. **@Inject** is a Java CDI annotation, more generic and JSR-330 compliant. *Difference*: @Autowired is Spring-only with advanced features; @Inject is portable across frameworks.

2.7 REST vs. SOAP

REST (Representational State Transfer) is an architectural style using HTTP methods, stateless, and lightweight (e.g., JSON). **SOAP** (Simple Object Access Protocol) is a protocol with strict standards, XML-based, and supports stateful operations. *Difference*: REST is simpler and web-friendly; SOAP is rigid and secure for enterprise use.

2.8 @GetMapping vs. @PostMapping

@GetMapping handles HTTP GET requests, typically for retrieving data. **@PostMapping** handles HTTP POST requests, used for creating or submitting data. *Difference*: @GetMapping is read-only; @PostMapping modifies server state.

2.9 @RequestParam vs. @PathVariable

@RequestParam extracts query parameters from the URL (e.g., ?id=1). **@PathVariable** extracts values from the URL path (e.g., /users/id). *Difference*: @RequestParam uses query strings; @PathVariable uses URL segments.

2.10 Spring Security vs. Shiro

Spring Security is a comprehensive security framework for Spring applications, supporting authentication, authorization, and more. **Shiro** is a lightweight security framework, simpler but less integrated with Spring. *Difference*: Spring Security is feature-rich for Spring ecosystems; Shiro is easier for smaller projects.

2.11 @ComponentScan vs. @EnableAutoConfiguration

@ComponentScan enables scanning for Spring components (e.g., @Component, @Service). **@EnableAutoConfiguration** auto-configures Spring Boot based on dependencies. *Difference*: @ComponentScan finds beans; @EnableAutoConfiguration sets up the application context.

2.12 @Transactional vs. @TransactionManagementConfigurer

@Transactional marks a method or class for transaction management. **@TransactionManagementConfigurer** customizes the transaction manager. *Difference*: @Transactional applies transactions; @TransactionManagementConfigurer configures transaction behavior.

2.13 ApplicationContext vs. BeanFactory

BeanFactory is a basic Spring container managing beans. **ApplicationContext** extends BeanFactory, adding features like event publishing and internationalization. *Difference*: ApplicationContext is more feature-rich for enterprise applications.

2.14 @Configuration vs. @Component

@Configuration marks a class for defining beans via @Bean methods. **@Component** marks a class as a general Spring-managed bean. *Difference*: @Configuration is for bean configuration; @Component is for general components.

2.15 @Service vs. @Repository

@Service marks a class for business logic. **@Repository** marks a class for data access, with exception translation. *Difference*: @Repository handles persistence exceptions; @Service is for service-layer logic.

2.16 @EnableWebMvc vs. @SpringBootApplication

@EnableWebMvc enables Spring MVC configuration, requiring manual setup. **@SpringBootApplication** includes auto-configuration for Spring Boot, including MVC. *Difference*: @SpringBootApplication simplifies setup; @EnableWebMvc is for custom MVC.

2.17 @Qualifier vs. @Primary

@Qualifier specifies a bean when multiple candidates exist. **@Primary** sets a default bean among multiple candidates. *Difference*: @Qualifier is explicit; @Primary is implicit.

2.18 @RequestMapping vs. @GetMapping

@RequestMapping maps HTTP requests with any method. **@GetMapping** is specific to GET requests. *Difference*: @GetMapping is a shortcut for @RequestMapping(method = RequestMethod.GET).

2.19 Spring Boot Starter vs. Actuator

Spring Boot Starter provides pre-configured dependencies for specific functionalities. **Actuator** provides monitoring and management endpoints. *Difference*: Starters simplify dependency setup; Actuator enables application monitoring.

2.20 @Profile vs. @Conditional

@Profile activates beans based on the active profile (e.g., dev, prod). **@Conditional** activates beans based on custom conditions. *Difference*: @Profile is profile-specific; @Conditional is more flexible.

2.21 @Async vs. @Scheduled

@Async runs methods asynchronously. **@Scheduled** runs methods on a fixed schedule. *Difference*: @Async is for one-off async tasks; @Scheduled is for recurring tasks.

2.22 @Entity vs. @Table

@Entity marks a class as a JPA entity for persistence. **@Table** specifies the database table for the entity. *Difference*: @Entity defines the entity; @Table customizes its table mapping.

2.23 @Column vs. @Transient

@Column maps a field to a database column. **@Transient** excludes a field from persistence. *Difference*: @Column includes fields in the database; @Transient excludes them.

2.24 @Lazy vs. @Eager

@Lazy delays bean initialization until needed. **@Eager** initializes beans at startup. *Difference*: @Lazy saves resources; @Eager ensures immediate availability.

2.25 @Scope vs. @Bean

@Scope sets the scope of a bean (e.g., singleton, prototype). **@Bean** defines a bean in a @Configuration class. *Difference*: @Scope configures bean lifecycle; @Bean creates the bean.

2.26 Spring AOP vs. AspectJ

Spring AOP is a proxy-based AOP framework for Spring applications. **AspectJ** is a full-fledged AOP framework using bytecode weaving. *Difference*: Spring AOP is simpler; AspectJ is more powerful.

2.27 @RequestBody vs. @ResponseBody

@RequestBody binds HTTP request body to a method parameter. **@ResponseBody** serializes the methods return value to the response body. *Difference*: @RequestBody handles input; @ResponseBody handles output.

2.28 @Valid vs. @Validated

@Valid triggers validation on a method parameter (JSR-303). **@Validated** enables Spring-specific validation on a class or method. *Difference*: @Validated supports group validation; @Valid is standard.

2.29 @PropertySource vs. @Value

@PropertySource loads properties from a file. **@Value** injects property values into fields or parameters. *Difference*: @PropertySource defines the source; @Value uses the values.

2.30 Spring Data JPA vs. Spring JDBC

Spring Data JPA simplifies JPA-based data access with repositories. **Spring JDBC** provides direct JDBC access with templates. *Difference*: Spring Data JPA is higher-level; Spring JDBC is lower-level.

2.31 @RestControllerAdvice vs. @ControllerAdvice

@ControllerAdvice handles exceptions and advice for controllers. **@RestControllerAdvice** combines @ControllerAdvice and @ResponseBody for REST APIs. *Difference*: @RestControllerAdvice returns JSON/XML responses.

2.32 @EnableCaching vs. @Cacheable

@EnableCaching enables caching support in Spring. **@Cacheable** caches a methods result. *Difference*: @EnableCaching activates caching; @Cacheable applies it to methods.

2.33 @ModelAttribute vs. @RequestBody

@ModelAttribute binds form data or query parameters to an object. **@RequestBody** binds the HTTP request body to an object. *Difference*: @ModelAttribute is for form data; @RequestBody is for JSON/XML payloads.

2.34 @SessionAttributes vs. @SessionAttribute

@SessionAttributes stores model attributes in the session at the class level. **@SessionAttribute** retrieves a session attribute in a method. *Difference*: @SessionAttributes manages session storage; @SessionAttribute accesses it.

2.35 @EnableJpaRepositories vs. @EnableTransactionManagement

@EnableJpaRepositories enables JPA repository scanning. **@EnableTransactionManagement** enables transaction support. *Difference*: @EnableJpaRepositories is for repositories; @EnableTransactionManagement is for transactions.

2.36 @NamedQuery vs. @Query

@NamedQuery defines a static JPQL query in the entity class. **@Query** defines a query directly in the repository method. *Difference*: @NamedQuery is reusable; @Query is method-specific.

2.37 @Embeddable vs. @Embedded

@Embeddable marks a class as a reusable component for entities. **@Embedded** includes an @Embeddable object in an entity. *Difference*: @Embeddable defines the component; @Embedded uses it.

2.38 @OneToMany vs. @ManyToOne

@OneToMany maps one entity to many related entities. **@ManyToOne** maps many entities to one related entity. *Difference*: @OneToMany is the owning side; @ManyToOne is the inverse.

2.39 @MappedSuperclass vs. @Inheritance

@MappedSuperclass shares common mappings with subclasses without a table. **@Inheritance** defines a table-per-class strategy for entities. *Difference*: @MappedSuperclass is for non-entity inheritance; @Inheritance is for entities.

2.40 @EnableScheduling vs. @Scheduled

@EnableScheduling enables scheduling in Spring. **@Scheduled** marks a method to run on a schedule. *Difference*: @EnableScheduling activates scheduling; @Scheduled defines tasks.

2.41 @ResponseStatus vs. @ExceptionHandler

@ResponseStatus sets the HTTP status for a response or exception. **@ExceptionHandler** handles specific exceptions in a controller. *Difference*: @ResponseStatus sets status; @ExceptionHandler customizes exception handling.

2.42 @EnableWebSecurity vs. @EnableGlobalMethodSecurity

@EnableWebSecurity enables Spring Security configuration. **@EnableGlobalMethodSecurity** enables method-level security (e.g., @PreAuthorize). *Difference*: @EnableWebSecurity is for global security; @EnableGlobalMethodSecurity is for method security.

2.43 @DataJpaTest vs. @SpringBootTest

@DataJpaTest tests JPA components with an in-memory database. **@SpringBootTest** tests the full application context. *Difference*: @DataJpaTest is lightweight for JPA; @SpringBootTest is comprehensive.

2.44 @MockBean vs. @SpyBean

@MockBean creates a mock for a bean in tests. **@SpyBean** wraps a real bean, allowing partial mocking. *Difference*: @MockBean replaces the bean; @SpyBean augments it.

2.45 @ConditionalOnProperty vs. @ConditionalOnClass

@ConditionalOnProperty enables a bean based on a property value. **@ConditionalOnClass** enables a bean if a class is present. *Difference*: @ConditionalOnProperty checks properties; @ConditionalOnClass checks classpath.

2.46 @SpringBootConfiguration vs. @Configuration

@SpringBootConfiguration is a specialized @Configuration for Spring Boot. **@Configuration** defines beans in any Spring application. *Difference*: @SpringBootConfiguration is Spring Boot-specific.

2.47 @ActiveProfiles vs. @Profile

@ActiveProfiles sets active profiles in tests. **@Profile** restricts a bean to specific profiles. *Difference*: @ActiveProfiles is for testing; @Profile is for bean activation.

2.48 @Import vs. @ImportResource

@Import imports @Configuration classes. **@ImportResource** imports XML configuration files. *Difference*: @Import is for Java config; @ImportResource is for XML.

2.49 @EnableAspectJAutoProxy vs. @EnableTransactionManagement

@EnableAspectJAutoProxy enables AOP proxying. **@EnableTransactionManagement** enables transaction management. *Difference*: @EnableAspectJAutoProxy is for AOP; @EnableTransactionManagement is for transactions.

2.50 @PersistenceContext vs. @Autowired for EntityManager

@PersistenceContext injects an EntityManager with JPA-specific handling. **@Autowired** injects an EntityManager as a regular bean. *Difference*: @PersistenceContext ensures proper JPA lifecycle management.