

**Explore More**

Subscription : Premium CDAC NOTES & MATERIAL @99



Contact to Join  
Premium Group



Click to Join  
Telegram Group

<CODEWITHARRAY'S/>

**For More E-Notes**

Join Our Community to stay Updated

**TAP ON THE ICONS TO JOIN!**

	<b>codewitharrays.in freelance project available to buy contact on 8007592194</b>	
<b>SR.NO</b>	<b>Project NAME</b>	<b>Technology</b>
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

freelance_Project available to buy contact on 8007592194		
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql



41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

## Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	<a href="https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW">https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW</a>
2	PG Mate / Room sharing/Flat sharing	<a href="https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp">https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp</a>
3	Tour and Travel System Project Version 1.0	<a href="https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12">https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12</a>
4	Marriage Hall Booking	<a href="https://youtu.be/VXz0kZQi5to?si=ILOS-QG3TpAFP5k7">https://youtu.be/VXz0kZQi5to?si=ILOS-QG3TpAFP5k7</a>
5	Ecommerce Shopping project	<a href="https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq">https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq</a>
6	Bike Rental System Project	<a href="https://youtu.be/FlzsAmIBCbk?si=7ujQTJqEgkQ8ju2H">https://youtu.be/FlzsAmIBCbk?si=7ujQTJqEgkQ8ju2H</a>
7	Multi-Restaurant management system	<a href="https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB">https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB</a>
8	Hospital management system Project	<a href="https://youtu.be/lynlouBZvY4?si=CXzQs3BsRkjKhZCw">https://youtu.be/lynlouBZvY4?si=CXzQs3BsRkjKhZCw</a>
9	Municipal Corporation system Project	<a href="https://youtu.be/cVMx9NVyl4I?si=qX0oQt-GT-LR_5jF">https://youtu.be/cVMx9NVyl4I?si=qX0oQt-GT-LR_5jF</a>
10	Tour and Travel System Project version 2.0	<a href="https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ">https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ</a>

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	<a href="https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug">https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug</a>
12	Gym Management system Project	<a href="https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX">https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX</a>
13	Online Driving License system Project	<a href="https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn">https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn</a>
14	Online Flight Booking system Project	<a href="https://youtu.be/m755rOwdk8U?si=HURvAY2VnizlyJlh">https://youtu.be/m755rOwdk8U?si=HURvAY2VnizlyJlh</a>
15	Employee management system project	<a href="https://youtu.be/ID1iE3W_GRw?si=Y_jv1xV_BljhrD0H">https://youtu.be/ID1iE3W_GRw?si=Y_jv1xV_BljhrD0H</a>
16	Online student school or college portal	<a href="https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD">https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD</a>
17	Online movie booking system project	<a href="https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSIsm">https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSIsm</a>
18	Online Pizza Delivery system project	<a href="https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM">https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM</a>
19	Online Crime Reporting system Project	<a href="https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO">https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO</a>
20	Online Children Adoption Project	<a href="https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N">https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N</a>

# TOP 50 SPRING BOOT ANNOTATIONS FOR INTERVIEW

## 1. @SpringBootApplication

1. Combines @ Configuration, @ EnableAutoConfiguration, and @ ComponentScan.
2. Marks the main class as the entry point of a Spring Boot application.
3. Enables auto-configuration for configuring Spring beans.
4. Scans components within the base package of the annotated class.
5. Simplifies Spring application setup and bootstrapping.

Example:

java

Code :

```
@SpringBootApplication
public class MyApp {
    public static void main(String[] args) { SpringApplication.run(MyApp.class,
        args);
    }
}
```

## 2. @RestController

1. Combines @Controller and @ResponseBody.
2. Defines a controller for REST APIs.
3. Automatically serializes returned objects into HTTP responses.
4. Simplifies API creation by eliminating explicit @ResponseBody
5. Works well with @GetMapping, @PostMapping, etc.

**Example:**

java

Code :

@RestController

```
public class UserController { @  
    GetMapping("/user") public  
        String getUser() { return  
            "User data";  
        }  
}
```

## 3. @GetMapping

1. Maps HTTP GET requests to handler methods.
2. A shortcut for @RequestMapping(method = RequestMethod.GET).
3. Used for fetching or querying data.
4. Supports dynamic path variables and query parameters.
5. Helps build RESTful GET APIs.

**Example:**

java

Code :

@GetMapping("/user/{id}")

```
public String getUserById(@PathVariable int id) { return
```

```
"User with ID: " + id;  
}
```

#### 4. @PostMapping

1. Maps HTTP POST requests to methods.
2. A shortcut for `@RequestMapping(method = RequestMethod.POST)`.
3. Handles data creation or input submission.
4. Often paired with `@RequestBody` for accepting JSON data.
5. Simplifies form or API POST handling.

**Example:**

java

Code :

```
@PostMapping("/user")
```

```
public String createUser(@RequestBody String user) {  
    return "User created: " + user;  
}
```

#### 5. @PutMapping

1. Maps HTTP PUT requests to methods.
2. Used for updating resources.
3. Suitable for idempotent update operations.
4. Works well with `@RequestBody` for accepting input.
5. Often used for updating existing records.

**Example:**

java

Code :

```
@PutMapping("/user/{id}")
```

```
public String updateUser(@PathVariable int id, @RequestBody String user) {
```



```
    return "User " + id + " updated with data: " + user;
}
```

## 6. @DeleteMapping

1. Maps HTTP DELETE requests to method
2. Handles resource deletion on the server.
3. Simplifies handling of HTTP DELETE requests.
4. Typically includes @PathVariable to identify the resource.
5. Useful for RESTful DELETE APIs.

**Example:**

java

Code :

```
@DeleteMapping("/user/{id}")
public String deleteUser(@PathVariable int id) {
    return "User " + id + " deleted";
}
```

## 7. @RequestMapping

1. General-purpose annotation for mapping HTTP requests.
2. Can handle all HTTP methods (GET, POST, etc.).
3. Used at both class and method levels.
4. Offers flexibility in request handling.
5. Replaced by @GetMapping, @PostMapping, etc., for specific methods.

### Example:

java :

```
@RequestMapping("/api")
public class ApiController {
    @RequestMapping(value = "/users", method = RequestMethod.GET) public
    String getUsers() {
        return "All users";
    }
}
```



## 8. @PathVariable

1. Binds method parameters to URI variables.
2. Extracts values from the URL path.
3. Works with @GetMapping and @DeleteMapping.
4. Supports type conversion for values.
5. Useful for dynamic endpoints.

### Example:

java

Code :

```
@GetMapping("/user/{id}")
public String getUser(@PathVariable int id) { return
    "User ID: " + id;
}
```

## 9. @RequestParam

1. Binds query parameters to method arguments.
2. Supports default values for missing parameters.
3. Helps handle HTTP GET query parameters.
4. Works well with form submission.
5. Provides optional and required parameter settings.

### Example:

java

Code :

```
@GetMapping("/search")
public String search(@RequestParam String keyword) { return
    "Search keyword: " + keyword;
}
```

## 10. @RequestBody

1. Maps HTTP request body to method parameters.
2. Supports JSON, XML, and other formats.
3. Used in POST and PUT methods.
4. Requires a compatible deserialization library.
5. Simplifies handling of input data.

### Example:

java

Code :

```
@PostMapping("/add")
```

```
public String addUser(@RequestBody User user) { return  
    "User added: " + user.getName();  
}
```

---

## 11. @ResponseBody

1. Indicates that a method's return value should be serialized into the HTTP response body.
2. Converts objects into JSON or XML for RESTful responses.
3. Automatically included in @RestController.
4. Can be used on individual controller methods.
5. Useful for creating non-view-based responses.

### Example:

java

Code :

```
@GetMapping("/status") @
```

```
ResponseBody
```

```
public String getStatus() {
```

```
    return "Application is running";
```

```
}
```



## 12. @Controller

1. Marks a class as a Spring MVC controller.
2. Used to define traditional web controllers that return views (e.g., HTML).
3. Often paired with @RequestMapping for route handling.
4. Returns a ModelAndView or a logical view name.
5. Unlike @RestController, it does not include @ResponseBody by default.

Example:

java

Code :

@Controller

```
public class HomeController { @  
    GetMapping("/home") public  
    String home() {  
        return "home"; // Refers to home.html in templates  
    }  
}
```

### 13. @Service

1. Marks a class as a service layer component.
2. Indicates that it holds business logic.
3. Automatically detected and registered as a Spring bean.
4. Promotes separation of concerns between layers.
5. Works well with dependency injection.

**Example:**

java

Code :

@Service

```
public class UserService {  
    public String getUser() {  
        return "User service called";  
    }  
}
```

### 14. @Repository

1. Marks a class as a DAO (Data Access Object).
2. Indicates that it interacts with the database.
3. Automatically detected and registered as a Spring bean.
4. Provides exception translation for persistence-related errors.
5. Promotes separation of concerns in the data layer.

**Example:**

java

Code :

@Repository

```
public class UserRepository {
```

```
public String findUserById(int id) { return
    "User found with ID: " + id;
}
}
```

## 15. @Component

1. Marks a class as a Spring-managed bean.
2. Acts as a generic stereotype for any Spring component.
3. Automatically detected during component scanning.
4. Can be used as a parent annotation for custom stereotype
5. Works across all layers of the application.

**Example:**

java

Code :

```
@Component public
class Utility {
    public String formatText(String text) {
        return text.toUpperCase();
    }
}
```

---

## 16. @Autowired

1. Injects dependencies into Spring-managed beans.
2. Can be applied to constructors, setters, or fields.
3. Automatically resolves and injects a matching bean.
4. Reduces boilerplate code compared to manual bean wiring.
5. Requires a matching bean defined in the Spring context.

**Example:**

java

Code :

@Service

```
public class UserService { @  
    Autowired  
    private UserRepository userRepository;  
}
```

## 17. @Qualifier

1. Used with @Autowired to specify which bean to inject when multiple beans match.
2. Helps resolve ambiguity in dependency injection.
3. Works with bean names or custom qualifiers.
4. Ensures precise bean injection.
5. Useful for applications with multiple implementations of an interface.

Example:

java

Code :

@Service

```
public class UserService { @  
    Autowired  
    @Qualifier("adminRepository")  
    private UserRepository userRepository;  
}
```

## 18. @Primary

1. Marks a bean as the primary candidate for autowiring.
2. Used when multiple beans of the same type are present.
3. Eliminates the need for @Qualifier in some cases.



4. Ensures a default preference for bean injection.

5. Simplifies dependency management.

**Example:**

java

Code :

@Configuration

public class AppConfig { @

Bean

@Primary



```
public UserRepository userRepository() { return  
    new UserRepository();  
}  
}
```

## 19. @Bean

1. Marks a method as a bean definition in Java-based configuration.
2. Defines Spring beans manually in @Configuration classes.
3. Used to configure third-party libraries or non-Spring classes.
4. Provides fine-grained control over bean creation.
5. Supports dependency injection in method parameters.

**Example:**

java

Code :

@Configuration

```
public class AppConfig { @  
    Bean  
    public Utility utility() {  
        return new Utility();  
    }  
}
```

## 20. @Configuration

1. Marks a class as a source of Spring bean definitions.
2. Typically used for Java-based configuration.
3. Replaces XML configuration files.
4. Works well with @Bean for explicit bean creation.
5. Scanned automatically if in the component scan path.

**Example:**

java

Code :

@Configuration

```
public class AppConfig { @  
    Bean  
    public UserRepository userRepository() { return  
        new UserRepository();  
    }  
}
```

## 21. @Scope

1. Defines the scope of a Spring bean (e.g., singleton, prototype).
2. Works with @Component, @Service, and other bean-defining annotations.
3. Defaults to singleton scope.
4. Useful for creating new instances per request in prototype scope.
5. Enhances bean lifecycle management.

Example:

java

Code :

@Component

@Scope("prototype")

```
public class PrototypeBean {  
    public PrototypeBean() {  
        System.out.println("Prototype instance created");  
    }  
}
```

## 22. @Lazy

1. Indicates that a bean should be lazily initialized.
2. Defers bean creation until it is first requested.
3. Useful for optimizing application startup time.
4. Works with @Component, @Service, and @Bean.
5. Particularly effective in large, complex applications.

**Example:**

java

Code :

@Component @

Lazy

public class LazyBean { public

LazyBean() {

System.out.println("Lazy bean initialized");

}

}

---

## 23. @Value

1. Injects values into fields, methods, or constructor parameters.
2. Supports property placeholders (e.g., \${property.name}).
3. Allows default values using : syntax (e.g., \${property:defaultValue}).
4. Can inject primitive types, strings, or arrays.
5. Often used to read values from application.properties.

**Example:**

java

Code :

@Component public

class Config {



@Value("\${app.name:DefaultApp}")



```
private String appName;

public String getAppName() {
    return appName;
}
}
```

---

## 24. @PropertySource

1. Loads properties files into the Spring environment.
2. Can load files from the classpath or file system.
3. Works with @Value for property injection.
4. Supports multiple property files.
5. Simplifies externalized configuration.

### Example:

java

Code :

@Configuration

@PropertySource("classpath:application.properties")

```
public class AppConfig {
}
```

---

## 25. @EnableConfigurationProperties

1. Enables the use of configuration properties in a Spring Boot app.
2. Typically used with @ConfigurationProperties.
3. Simplifies binding of properties to Java objects.
4. Automatically registers annotated classes as beans.
5. Enhances type-safe configuration.

### Example:

java

Code :

```
@EnableConfigurationProperties(AppProperties.class) @
Configuration
public class Config {
}
```

---

## 26. @ConfigurationProperties

1. Binds external configuration properties to a Java bean.
2. Supports nested properties and data structures.
3. Works with @EnableConfigurationProperties.
4. Enables type-safe access to configuration values.
5. Useful for managing complex configurations.

**Example:**

java

Code :

```
@ConfigurationProperties(prefix = "app")
public class AppProperties {
    private String name;
    private String version;

    // Getters and setters
}
```

---

## 27. @Conditional

1. Enables conditional bean registration based on custom logic.
2. Can be combined with @Bean, @Component, etc.
3. Useful for creating beans only when certain conditions are met.
4. Supports both built-in and custom conditions.
5. Commonly used for environment-specific configurations.

### Example:

java

Code :

@Configuration

```
public class AppConfig { @  
    Bean  
    @Conditional(MyCondition.class) public  
    MyBean myBean() {  
        return new MyBean();  
    }  
}
```

---

### 28. @Profile

1. Activates beans only in specific environments or profiles.
2. Works with @Component, @Service, and @Configuration.
3. Commonly used for dev, test, and prod configurations.
4. Profiles are activated using spring.profiles.active.
5. Simplifies environment-specific bean management.

### Example:

java

Code :

@Component

@Profile("dev")

```
public class DevBean { public  
    DevBean() {  
        System.out.println("DevBean initialized");  
    }  
}
```

---



## 29. @EventListener

1. Registers a method to listen for application events.
2. Simplifies event-driven programming.
3. Supports both custom and predefined events.
4. Works with asynchronous event handling.
5. Replaces traditional ApplicationListener implementation.

### Example:

java

Code :

@Component

```
public class MyEventListener { @  
    EventListener  
    public void handleEvent(ApplicationReadyEvent event) {  
        System.out.println("Application is ready!");  
    }  
}
```

---

## 30. @EnableAsync

1. Enables asynchronous method execution.
2. Works with @Async annotation for methods.
3. Requires a task executor bean configuration.
4. Improves performance for non-blocking operations.
5. Often used for background tasks.

### Example:

java

Code :

@Configuration

@EnableAsync

```
public class AppConfig {  
}
```

@Service

```
public class AsyncService { @  
    Async  
    public void asyncMethod() {  
        System.out.println("Running asynchronously!");  
    }  
}
```

---

### 31. @Async

1. Marks a method for asynchronous execution.
2. Requires @EnableAsync in the configuration.
3. Methods annotated with @Async will execute in a separate thread.
4. Used for background or time-consuming tasks.
5. Supports custom task executors for fine control.

**Example:**

java

Code :

@Service

```
public class AsyncService { @  
    Async  
    public void executeAsyncTask() {  
        System.out.println("Executing in a separate thread");  
    }  
}
```

---

### 32. @EnableScheduling

1. Enables Spring's scheduled task execution capability.
2. Works with @Scheduled annotation.
3. Often used in service-level components.
4. Requires no additional configuration for basic scheduling.
5. Simplifies implementation of periodic or delayed tasks.

#### Example:

java

Code :

@Configuration

@EnableScheduling

```
public class SchedulerConfig {  
}
```

@Component

```
public class TaskScheduler {  
    @Scheduled(fixedRate = 5000)  
    public void scheduledTask() {  
        System.out.println("Task executed every 5 seconds");  
    }  
}
```

---

### 33. @Scheduled

1. Configures scheduled tasks for periodic execution.
2. Supports fixed rate, fixed delay, and cron expressions.
3. Requires @EnableScheduling in the configuration.
4. Can run tasks in the background at defined intervals.
5. Simplifies periodic task implementation.

### Example:

java

Code :

@Component

```
public class MyTask {  
    @Scheduled(cron = "0 0 * * * ?")  
    public void runTask() {  
        System.out.println("Task runs at the start of every hour");  
    }  
}
```

---

### 34. @EnableTransactionManagement

1. Enables annotation-driven transaction management.
2. Works with @Transactional for declarative transactions.
3. Automatically manages commit, rollback, and propagation.
4. Often used in data-access layers.
5. Reduces boilerplate code for transaction handling.

### Example:

java

Code :

@Configuration

@EnableTransactionManagement

```
public class AppConfig {  
}
```

---

### 35. @Transactional

1. Manages database transactions at the method or class level.
2. Supports rollback for exceptions by default.
3. Configures transaction propagation and isolation levels.

4. Often used in repository or service layers.
5. Simplifies error handling in database operations.

**Example:**

java

Code :

@Service

```
public class TransactionService { @Transactional
    public void performTransactionalTask() {
        // Database operations
    }
}
```

---

### 36. @RestControllerAdvice

1. Handles exceptions for REST APIs globally.
2. Combines @ControllerAdvice and @ResponseBody.
3. Provides centralized error handling for REST controllers.
4. Can define custom exception-handling logic.
5. Simplifies the management of API error responses.

**Example:**

java

Code :

@RestControllerAdvice

```
public class GlobalExceptionHandler {
    @ExceptionHandler(RuntimeException.class)
    public String handleRuntimeException(RuntimeException ex) { return
        "Error: " + ex.getMessage();
    }
}
```

}

---

### 37. @SessionAttributes

1. Specifies attributes to store in the session scope.
2. Used in Spring MVC controllers.
3. Helps share attributes between handler methods.
4. Works with @ModelAttribute.
5. Useful for managing user sessions or temporary data.

**Example:**

java

Code :

@Controller

@SessionAttributes("user") public

class SessionController {

    @ModelAttribute("user")

    public User user() {

        return new User();

    }

}

---

### 38. @RequestAttribute

1. Binds a request attribute to a method parameter.
2. Useful for passing data across filters and controllers.
3. Simplifies accessing request attributes.
4. Reduces boilerplate code compared to manual extraction.
5. Supports type conversion for parameters.

**Example:**

java

Code :



@Controller

```
public class MyController { @  
    GetMapping("/greet")  
    public String greet(@ RequestAttribute("name") String name) { return  
        "Hello, " + name;  
    }  
}
```

---

### 39. @EnableJpaRepositories

1. Enables JPA repositories in a Spring Boot application.
2. Automatically detects interfaces extending JpaRepository.
3. Simplifies database interaction with JPA.
4. Configures the base package for scanning repository interfaces.
5. Reduces boilerplate for data access layers.

Example:

java

Code :

@Configuration

```
@ EnableJpaRepositories(basePackages = "com.example.repositories") public class  
JpaConfig {  
}
```

---

### 40. @MappedSuperclass

1. Marks a class as a JPA mapped superclass.
2. Provides a base class for JPA entities.
3. Fields in the superclass are mapped to database columns.
4. Cannot be directly instantiated or queried.
5. Simplifies code reuse in JPA entities.

Example:

java

Code :

@MappedSuperclass

```
public abstract class BaseEntity { @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private LocalDateTime createdAt;
}
```

---

#### 41. @Embedded

1. Marks an attribute in an entity class as an embeddable type.
2. Used for embedding reusable value objects in entities.
3. Simplifies modeling of composite fields like addresses or measurements.
4. Works with the @Embeddable annotation in the reusable class.
5. Reduces redundancy by reusing components in multiple entities.

Example:

java

Code :

@Entity

```
public class Employee { @
    Id
    @GeneratedValue
    private Long id;
    @Embedded
    private Address address;
```

```
}
```

@Embeddable

```
public class Address { private  
    String street; private String  
    city; private String zip;  
}
```

---

#### 42. @Embeddable

1. Marks a class as embeddable for JPA entities.
2. Used with the @Embedded annotation in the parent entity.
3. Fields in the class are mapped as part of the containing entity.
4. Makes code modular and reusable.
5. Useful for representing composite attributes.

Example:

java

Code :

@Embeddable

```
public class Address { private  
    String street; private String  
    city; private String zip;  
}
```

---

#### 43. @ElementCollection

1. Maps a collection of basic or embeddable types to a database table.
2. Used for lists, sets, or maps of values in entities.

3. Requires no additional table relationships.
4. Automatically maps collections to a join table.
5. Simplifies modeling of multi-valued attributes.

**Example:**

java

Code :

@Entity

```
public class Employee { @  
    Id  
    @GeneratedValue  
    private Long id;  
    @ElementCollection  
    private List<String> skills;  
}
```

---

**44. @Enumerated**

1. Specifies how an enumeration should be mapped to the database.
2. Supports EnumType.ORDINAL and EnumType.STRING.
3. Ensures type-safe handling of enums in JPA.
4. Used with entity fields representing enums.
5. Avoids errors by explicitly defining mapping.

**Example:**

java

Code :

@Entity

```
public class Task { @  
    Id  
    @GeneratedValue
```

```
private Long id;

@Enumerated(EnumType.STRING) private
TaskStatus status;
}
```

```
public enum TaskStatus {
    PENDING, COMPLETED
}
```

---

#### 45. @Query

1. Defines custom JPQL or SQL queries for JPA repositories.
2. Enhances flexibility for complex queries.
3. Used with repository interface methods.
4. Supports parameterized queries with @Param.
5. Simplifies custom data access logic.

**Example:**

java

Code :

@Repository

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

    @Query("SELECT e FROM Employee e WHERE e.name = :name") List<Employee>
    findByName(@Param("name") String name);
}
```

---

#### 46. @Modifying

1. Indicates a repository method that performs a modifying query.
2. Works with @Query for update or delete operations.
3. Requires transactional support with @Transactional.

4. Enhances data manipulation capabilities in repositories.

5. Ensures proper handling of non-select queries.

**Example:**

java

Code :

@Repository

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
    @Modifying  
    @Query("UPDATE Employee e SET e.salary = :salary WHERE e.id = :id") void  
    updateSalary(@Param("id") Long id, @Param("salary") double  
    salary);  
}
```

---

#### 47. @GeneratedValue

1. Specifies the generation strategy for primary keys.
2. Supports strategies like AUTO, IDENTITY, SEQUENCE, and TABLE.
3. Works with the @Id annotation in entities.
4. Automatically assigns unique identifiers to entities.
5. Simplifies key generation for database tables.

**Example:**

java

Code :

@Entity

```
public class Employee { @  
    Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
}
```



---

#### 48. @Lob

1. Maps a field to a large object (LOB) in the database.
2. Supports storing large data like text or binary files.
3. Can be used for CLOB (Character LOB) or BLOB (Binary LOB).
4. Works with @Column for additional configurations.
5. Simplifies handling of large data in entities.

##### Example:

java

Code :

@Entity

```
public class Document { @
```

```
    Id
```

```
    @GeneratedValue
```

```
    private Long id;
```

```
    @Lob
```

```
    private byte[] content;
```

```
}
```

---

#### 49. @Temporal

1. Maps date/time fields to appropriate SQL types.
2. Supports DATE, TIME, and TIMESTAMP.
3. Works with java.util.Date or java.util.Calendar.
4. Ensures accurate handling of date/time data.
5. Avoids incorrect type mapping in database schema.

##### Example:

java

Code : @

Entity

```
public class Event { @  
    Id  
    @GeneratedValue  
    private Long id;  
    @Temporal(TemporalType.TIMESTAMP) private  
    Date eventDate;  
}
```

---

## 50. @JsonIgnore

1. Prevents serialization or deserialization of a field in JSON.
2. Used in classes processed by Jackson.
3. Helps exclude sensitive or unnecessary fields from JSON responses.
4. Can be used with bidirectional relationships to avoid infinite loops.
5. Simplifies control over JSON output.

### Example:

java

Code :

@Entity

```
public class User {  
    @Id  
    @GeneratedValue  
    private Long id;  
    private String username;  
  
    @JsonIgnore  
    private String password;}
```



<https://www.youtube.com/@codewitharrays>



<https://www.instagram.com/codewitharrays/>



<https://t.me/codewitharrays> Group Link: <https://t.me/ccee2025notes>



[+91 8007592194](tel:+918007592194) [+91 9284926333](tel:+919284926333)



[codewitharrays@gmail.com](mailto:codewitharrays@gmail.com)



<https://codewitharrays.in/project>