# Spring Boot Annotations

## Introduction

Spring Boot annotations are essential tools in simplifying configuration, reducing boilerplate code, and providing powerful functionality in Java applications. This guide highlights the key annotations across different categories, explaining their purpose, usage, and best practices.

---

## Core Stereotype Annotations

### `@Component`

```
@Component
public class UserService {
    // Class implementation
}
```

- **Purpose:** Generic annotation for Spring-managed components
- **Usage:** Marks the class as a Spring bean
- **Best Practice:** Use for general-purpose classes

---

### `@Service`

```
@Service
public class UserService {
    // Business logic implementation
}
```

- **Purpose:** Specializes `@Component` for service layer classes
- **Usage:** Indicates the class contains business logic
- **Best Practice:** Enhances code readability and architectural clarity

---

### `@Repository`

```
@Repository
public class UserRepository {
    // Data access logic
}
```

- **Purpose:** Specializes `@Component` for data access
- **Usage:** Marks classes that handle database operations
- **Best Practice:** Enables Spring's exception translation mechanism

---

### `@Controller`

```
@Controller
public class UserController {
    // Web layer logic
}
```

- **Purpose:** Specializes `@Component` for Spring MVC controllers
- **Usage:** Handles HTTP requests in web applications
- **Best Practice:** Use with `@RequestMapping` for routing

---

### `@RestController`

```
@RestController
public class UserController {
    @GetMapping("/users")
    public List<User> getAllUsers() {
        // Return users
    }
}
```

- **Purpose:** Combines `@Controller` and `@ResponseBody`
- **Usage:** Simplifies creation of RESTful web services
- **Best Practice:** Automatically serializes responses to JSON/XML

---

## Dependency Injection Annotations

## @Autowired

```
@Service
public class UserService {
    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}
```

- **Purpose:** Enables automatic dependency injection
- **Usage:** Can be used on constructors, methods, or fields
- **Best Practice:** Prefer constructor injection for immutability

---

## @Qualifier

```
@Autowired
@Qualifier("specificUserRepository")
private UserRepository userRepository;
```

- **Purpose:** Disambiguates when multiple beans of the same type exist
- **Usage:** Specifies which bean to inject when there are multiple options
- **Best Practice:** Use for precise dependency injection

---

## @Primary

```
@Repository
@Primary
public class PrimaryUserRepository implements UserRepository {
    // Primary implementation
}
```

- **Purpose:** Marks a bean as the default when multiple candidates exist
- **Usage:** Resolves conflicts between beans of the same type
- **Best Practice:** Useful when one implementation should be preferred

---

# Configuration Annotations

## `@Configuration`

```
@Configuration
public class AppConfig {
    @Bean
    public DataSource dataSource() {
        // Configure and return DataSource
    }
}
```

- **Purpose:** Indicates a class contains bean definitions
- **Usage:** Replaces XML configuration, allows programmatic configuration
- **Best Practice:** Use for organizing bean configuration in classes

---

## `@Bean`

```
@Configuration
public class CacheConfig {
    @Bean
    public CacheManager cacheManager() {
        return new ConcurrentMapCacheManager("users");
    }
}
```

- **Purpose:** Declares a bean to be managed by Spring context
- **Usage:** Defines custom beans with configuration logic
- **Best Practice:** Use for complex bean creation

---

## `@ConfigurationProperties`

```
@Configuration
@ConfigurationProperties(prefix = "app.database")
public class DatabaseProperties {
    private String url;
    private String username;
    // Getters and setters
}
```

- **Purpose:** Binds external configuration properties to a class
- **Usage:** Enables type-safe configuration management
- **Best Practice:** Use with properties files like `application.properties`

---

# Request Handling Annotations

### `@RequestMapping`

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        // Retrieve user
    }
}
```

- **Purpose:** Maps web requests to handler methods
- **Usage:** Specifies the routing of HTTP requests
- **Best Practice:** Use at both the class and method levels for clean routing

---

### `@GetMapping`, `@PostMapping`, etc.

```
@RestController
public class UserController {
    @GetMapping("/users")
    public List<User> listUsers() {}

    @PostMapping("/users")
    public User createUser(@RequestBody User user) {}
}
```

- **Purpose:** Shortcuts for HTTP method-specific mapping
- **Usage:** Provides cleaner and more readable routes for GET, POST, PUT, DELETE
- **Best Practice:** Use method-specific annotations for clarity

---

### `@PathVariable`

```
@GetMapping("/users/{id}")
public User getUser(@PathVariable Long id) {
    return userService.findById(id);
}
```

- **Purpose:** Extracts values from the URL path
- **Usage:** Binds method parameters to path variables
- **Best Practice:** Essential for RESTful API design

---

### `@RequestBody`

```
@PostMapping("/users")
public User createUser(@RequestBody UserDTO userDTO) {
    return userService.create(userDTO);
}
```

- **Purpose:** Binds HTTP request body to method parameter
- **Usage:** Automatically deserializes incoming JSON/XML
- **Best Practice:** Commonly used with POST/PUT requests

---

## Validation Annotations

### `@Valid`

```
@PostMapping("/users")
public User createUser(@Valid @RequestBody UserDTO userDTO) {
    return userService.create(userDTO);
}
```

- **Purpose:** Triggers validation of the request body
- **Usage:** Works with Java Bean Validation annotations
- **Best Practice:** Ensures input data integrity before processing

---

### Validation Annotations Examples

```
public class UserDTO {
    @NotBlank(message = "Username cannot be blank")
```

```
    private String username;

    @Email(message = "Invalid email format")
    private String email;

    @Size(min = 8, max = 50, message = "Password must be between 8 and 50
characters")
    private String password;
}
```

- **Common Annotations:**
  - `@NotNull`: Ensures the value is not null
  - `@NotBlank`: Ensures a string is neither null nor empty
  - `@Size`: Validates string length constraints
  - `@Email`: Validates email format
  - `@Pattern`: Validates with a regular expression

---

# Transaction Management

### `@Transactional`

```
@Service
public class UserService {
    @Transactional
    public void transferFunds(Account from, Account to, BigDecimal amount) {
        // Transactional method
    }
}
```

- **Purpose:** Manages transactions declaratively
- **Usage:** Ensures transactions are committed or rolled back based on method execution
- **Best Practice:** Use for critical business operations requiring transaction management

---

# Exception Handling

### `@ControllerAdvice`

```
@ControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler(ResourceNotFoundException.class)
   @ResponseStatus(HttpStatus.NOT_FOUND)
   public ErrorResponse handleResourceNotFound(ResourceNotFoundException ex) {
      return new ErrorResponse(ex.getMessage());
   }
}
```

- **Purpose:** Global exception handling across controllers
- **Usage:** Centralizes error management
- **Best Practice:** Define custom error responses for a consistent API

---

## Spring Boot Core Annotations

### `@SpringBootApplication`

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

- **Purpose:** A composite annotation that includes `@SpringBootConfiguration`, `@EnableAutoConfiguration`, and `@ComponentScan`.
- **Usage:** Marks the main class of a Spring Boot application.
- **Best Practice:** Always use as the entry point for Spring Boot apps.

### `@SpringBootConfiguration`

```
@SpringBootConfiguration
public class AppConfig {
```

```
    // Bean configurations
}
```

- **Purpose:** Indicates a class that contains Spring configuration.
- **Usage:** Used when defining custom configuration classes.
- **Best Practice:** Primarily used internally when combined with `@SpringBootApplication`.

### `@EnableAutoConfiguration`

```
@EnableAutoConfiguration
public class AutoConfig {
    // Configuration for auto configurations
}
```

- **Purpose:** Tells Spring Boot to automatically configure application beans based on classpath and other settings.
- **Usage:** Allows Spring Boot to configure beans based on the environment.

### `@ComponentScan`

```
@ComponentScan(basePackages = "com.example")
public class AppConfig {
    // Custom scanning logic
}
```

- **Purpose:** Instructs Spring to scan the specified packages for components, configurations, and services.
- **Usage:** Defines the base packages to scan for Spring beans.

---

## Auto-Configuration Annotations

### `@ConditionalOnClass`

```
@Configuration
@ConditionalOnClass(DataSource.class)
public class DataSourceConfig {
    // Beans are created if DataSource is on the classpath
}
```

- **Purpose:** Indicates that a bean should be created if a specified class is on the classpath.
- **Usage:** Conditional bean creation based on the presence of a class.

### `@ConditionalOnMissingClass`

```
@Configuration
@ConditionalOnMissingClass("org.springframework.jdbc.datasource.DataSource")
public class MyDataSourceConfig {
    // Beans created if DataSource class is missing
}
```

- **Purpose:** Indicates that a bean should be created if a specified class is missing from the classpath.
- **Usage:** Conditional bean creation when a class is absent.

### `@ConditionalOnBean`

```
@Configuration
@ConditionalOnBean(DataSource.class)
public class DatabaseConfig {
    // Bean created only if DataSource bean exists
}
```

- **Purpose:** Conditional bean creation based on the presence of a specified bean.
- **Usage:** Ensures that a bean is only created when another bean is available.

### `@ConditionalOnMissingBean`

```
@Configuration
@ConditionalOnMissingBean(DataSource.class)
public class MyDatabaseConfig {
    // Bean created if DataSource bean is not available
}
```

- **Purpose:** Creates a bean only if a specified bean is not already defined in the context.
- **Usage:** Useful for creating fallback or default beans.

### `@ConditionalOnProperty`

```
@Configuration
@ConditionalOnProperty(name = "app.feature.enabled", havingValue = "true")
```

```
public class FeatureConfig {
    // Beans are created if the property is true
}
```

- **Purpose:** Creates a bean only if a specific property is set to a particular value.
- **Usage:** Conditional configuration based on application properties.

### `@ConditionalOnResource`

```
@Configuration
@ConditionalOnResource(resources = "classpath:/somefile.txt")
public class ResourceBasedConfig {
    // Bean created if the specified resource exists
}
```

- **Purpose:** Conditionally configures a bean based on the presence of a specific resource (e.g., file, classpath resource).
- **Usage:** Useful for setting up configurations based on available resources.

### `@ConditionalOnWebApplication` and `@ConditionalOnNotWebApplication`

```
@Configuration
@ConditionalOnWebApplication
public class WebConfig {
    // Beans for web applications only
}
```

- **Purpose:** Configures beans based on whether the application is a web application.
- **Usage:** Helps in conditionally creating beans in web vs non-web environments.

### `@ConditionalExpression`

```
@Configuration
@ConditionalExpression("system.properties['env'] == 'prod'")
public class ProductionConfig {
    // Beans only loaded in production
}
```

- **Purpose:** Provides a way to load beans based on a SpEL (Spring Expression Language) expression.

- **Usage:** Conditional bean creation based on evaluated expressions.

### `@Conditional`

```
@Configuration
@Conditional(MyCustomCondition.class)
public class ConditionalConfig {
    // Beans created based on custom condition logic
}
```

- **Purpose:** Allows the creation of beans based on a custom condition.
- **Usage:** Custom conditional logic for bean creation.

---

## Controller Annotations for HTTP Request Handling

### `@Controller`

```
@Controller
public class UserController {
    // Handles web requests for user-related operations
}
```

- **Purpose:** Marks a class as a Spring MVC controller for handling web requests.
- **Usage:** Typically used in web-based applications to map requests to methods.

### `@RestController`

```
@RestController
public class UserRestController {
    @GetMapping("/users")
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }
}
```

- **Purpose:** Combines `@Controller` and `@ResponseBody` to simplify RESTful API development.
- **Usage:** Used for creating REST controllers that automatically serialize return values to JSON/XML.

## @RequestMapping

```
@RequestMapping("/users")
public class UserController {
    // Maps HTTP requests to methods
}
```

- **Purpose:** General mapping for HTTP requests.
- **Usage:** Used to specify a route for a method or class, supports all HTTP methods.

## @RequestParam

```
@GetMapping("/users")
public User getUser(@RequestParam("id") Long userId) {
    return userService.getUserById(userId);
}
```

- **Purpose:** Extracts query parameters from the request URL.
- **Usage:** Used for retrieving parameters passed in the URL query string.

## @PathVariable

```
@GetMapping("/users/{id}")
public User getUser(@PathVariable Long id) {
    return userService.getUserById(id);
}
```

- **Purpose:** Extracts values from the URI path.
- **Usage:** Common in RESTful APIs to capture dynamic path segments.

## @RequestBody

```
@PostMapping("/users")
public User createUser(@RequestBody UserDTO userDTO) {
    return userService.createUser(userDTO);
}
```

- **Purpose:** Binds the request body to a method parameter.
- **Usage:** Automatically deserializes the incoming JSON or XML to an object.

## @ResponseBody

```
@RequestMapping("/users")
@ResponseBody
public List<User> getUsers() {
    return userService.getAllUsers();
}
```

- **Purpose:** Tells Spring to write the method's return value directly to the HTTP response body.
- **Usage:** Typically used in REST APIs for sending JSON or XML responses.

### `@ModelAttribute`

```
@ModelAttribute("user")
public User getUserModel() {
    return new User();
}
```

- **Purpose:** Binds a method parameter or return value to a model attribute.
- **Usage:** Commonly used in forms or web pages for binding request parameters to model objects.

---

## Specialized HTTP Method Annotations

### `@GetMapping`

```
@GetMapping("/users")
public List<User> getUsers() {
    return userService.getAllUsers();
}
```

- **Purpose:** Shortcut for `@RequestMapping(method = RequestMethod.GET)`.
- **Usage:** Use when the HTTP method is GET.

### `@PutMapping`

```
@PutMapping("/users/{id}")
public User updateUser(@PathVariable Long id, @RequestBody UserDTO userDTO) {
    return userService.updateUser(id, userDTO);
}
```

- **Purpose:** Shortcut for `@RequestMapping(method = RequestMethod.PUT)`.
- **Usage:** Use for updating existing resources.

### `@PostMapping`

```
@PostMapping("/users")
public User createUser(@RequestBody UserDTO userDTO) {
    return userService.createUser(userDTO);
}
```

- **Purpose:** Shortcut for `@RequestMapping(method = RequestMethod.POST)`.
- **Usage:** Used for creating new resources.

### `@PatchMapping`

```
@PatchMapping("/users/{id}")
public User partiallyUpdateUser(@PathVariable Long id, @RequestBody UserDTO
userDTO) {
    return userService.updateUser(id, userDTO);
}
```

- **Purpose:** Shortcut for `@RequestMapping(method = RequestMethod.PATCH)`.
- **Usage:** Used for partial updates to resources.

### `@DeleteMapping`

```
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable Long id) {
    userService.deleteUser(id);
}
```

- **Purpose:** Shortcut for `@RequestMapping(method = RequestMethod.DELETE)`.
- **Usage:** Used to delete resources.

---