**Spring Boot Annotations**

Spring Boot provides numerous annotations to simplify application development. Here’s a categorized list of commonly used Spring Boot annotations:

**Core Spring Boot Annotations**

1. **@SpringBootApplication**
   * Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
   * Marks the main class of a Spring Boot application.
2. **@EnableAutoConfiguration**
   * Enables Spring Boot’s auto-configuration feature.
3. **@ComponentScan**
   * Automatically scans for components, configurations, and services in specified packages.
4. **@Configuration**
   * Marks a class as a source of bean definitions.
5. **@Bean**
   * Indicates that a method produces a bean to be managed by the Spring container.

**Controller and REST API Annotations**

1. **@RestController**
   * Combines @Controller and @ResponseBody.
   * Indicates a class that handles REST API requests.
2. **@Controller**
   * Marks a class as a Spring MVC controller.
3. **@RequestMapping**
   * Maps HTTP requests to handler methods.
4. **@GetMapping** / **@PostMapping** / **@PutMapping** / **@DeleteMapping**
   * Specific HTTP method mappings.
5. **@PathVariable**
   * Binds a method parameter to a URI template variable.
6. **@RequestParam**
   * Extracts query parameters from the URL.
7. **@RequestBody**
   * Binds the body of a POST or PUT request to a method parameter.
8. **@ResponseBody**
   * Binds the method return value to the web response body.

**Dependency Injection Annotations**

1. **@Autowired**
   * Automatically wires beans by type.
2. **@Qualifier**
   * Specifies which bean should be wired when multiple candidates are available.
3. **@Primary**
   * Indicates the preferred bean when multiple candidates exist.
4. **@Lazy**
   * Marks a bean to be initialized lazily.

**Database and JPA Annotations**

1. **@Entity**
   * Marks a class as a JPA entity.
2. **@Table**
   * Specifies the table name for an entity.
3. **@Id**
   * Marks a field as the primary key.
4. **@GeneratedValue**
   * Specifies the generation strategy for primary keys.
5. **@Column**
   * Maps a field to a specific database column.
6. **@ManyToOne**, **@OneToMany**, **@OneToOne**, **@ManyToMany**
   * Define relationships between entities.
7. **@Transactional**
   * Indicates that a method or class is transactional.
8. **@Repository**
   * Marks a class as a DAO (Data Access Object).

**Validation Annotations**

1. **@Valid**
   * Enables validation for method parameters or entities.
2. **@NotNull**, **@NotEmpty**, **@Size**, **@Pattern**, etc.
   * Hibernate Validator annotations for input validation.

**Security Annotations**

1. **@EnableWebSecurity**
   * Enables Spring Security configuration.
2. **@PreAuthorize** / **@PostAuthorize**
   * Applies authorization checks before or after method execution.
3. **@Secured**
   * Secures methods with roles.
4. **@RolesAllowed**
   * Specifies roles allowed to access a method or resource.

**Spring Boot Testing Annotations**

1. **@SpringBootTest**
   * Loads the full application context for integration tests.
2. **@WebMvcTest**
   * Tests only the web layer of the application.
3. **@DataJpaTest**
   * Tests only the JPA layer.
4. **@MockBean**
   * Creates a mock of a bean for testing.
5. **@Test**
   * Marks a method as a test case (JUnit 5).

**Hibernate Annotations**

Hibernate is a popular ORM framework. Here are its key annotations:

**Core Annotations**

1. **@Entity**
   * Marks a class as a Hibernate entity.
2. **@Table**
   * Specifies the table name for the entity.
3. **@Id**
   * Marks a field as the primary key.
4. **@GeneratedValue**
   * Specifies the primary key generation strategy:
     + GenerationType.AUTO
     + GenerationType.IDENTITY
     + GenerationType.SEQUENCE
     + GenerationType.TABLE
5. **@Column**
   * Maps a field to a database column.
6. **@Transient**
   * Excludes a field from being persisted.
7. **@Embedded**
   * Embeds a class as part of an entity.
8. **@Embeddable**
   * Marks a class as embeddable.

**Relationship Annotations**

1. **@OneToOne**
   * Defines a one-to-one relationship.
2. **@OneToMany**
   * Defines a one-to-many relationship.
3. **@ManyToOne**
   * Defines a many-to-one relationship.
4. **@ManyToMany**
   * Defines a many-to-many relationship.
5. **@JoinColumn**
   * Specifies the foreign key column for a relationship.
6. **@JoinTable**
   * Defines the join table for many-to-many relationships.

**Lifecycle Annotations**

1. **@PrePersist**
   * Executes before saving an entity.
2. **@PostPersist**
   * Executes after saving an entity.
3. **@PreUpdate**
   * Executes before updating an entity.
4. **@PostUpdate**
   * Executes after updating an entity.
5. **@PreRemove**
   * Executes before deleting an entity.
6. **@PostRemove**
   * Executes after deleting an entity.

**Other Annotations**

1. **@Lob**
   * Maps a field to a Large Object (LOB) in the database.
2. **@Enumerated**
   * Maps an enum to a database column.
3. **@Version**
   * Implements optimistic locking.
4. **@NamedQuery**
   * Defines a named JPQL query.
5. **@Cache**
   * Configures second-level caching for an entity.

**Detailed Explanation of Spring Boot Annotations**

Spring Boot annotations simplify configuration and development by reducing boilerplate code. Below is a detailed explanation of the major annotations in Spring Boot, categorized based on their functionality:

**1. Core Spring Boot Annotations**

**@SpringBootApplication**

* **Purpose:** Marks the main class of a Spring Boot application.
* **Combines:**
  1. **@Configuration**: Indicates that the class has Spring beans defined via methods annotated with @Bean.
  2. **@EnableAutoConfiguration**: Automatically configures Spring application based on the classpath dependencies.
  3. **@ComponentScan**: Scans the package and its sub-packages for Spring components (e.g., @Controller, @Service, @Repository, etc.).
* **Example:**

java

Copy code

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

**@Configuration**

* **Purpose:** Marks a class as a source of bean definitions for the Spring container.
* **Example:**

java

Copy code

@Configuration

public class AppConfig {

@Bean

public MyService myService() {

return new MyService();

}

}

**@EnableAutoConfiguration**

* **Purpose:** Enables Spring Boot's auto-configuration mechanism to configure beans automatically based on dependencies in the classpath.
* **Note:** Generally used indirectly via @SpringBootApplication.

**@ComponentScan**

* **Purpose:** Enables component scanning to automatically detect and register beans (classes annotated with @Component, @Service, @Repository, etc.) in the application context.
* **Example:**

java

Copy code

@ComponentScan(basePackages = "com.example.services")

**@Bean**

* **Purpose:** Indicates that a method produces a Spring bean to be managed by the Spring container.
* **Example:**

java

Copy code

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

**2. Web and Controller Annotations**

**@RestController**

* **Purpose:** Marks a class as a RESTful controller where each method returns a JSON or XML response.
* **Combines:** @Controller and @ResponseBody.
* **Example:**

java

Copy code

@RestController

public class MyController {

@GetMapping("/hello")

public String hello() {

return "Hello, World!";

}

}

**@Controller**

* **Purpose:** Marks a class as a Spring MVC controller. Use this for non-RESTful web applications where views (e.g., Thymeleaf, JSP) are returned.
* **Example:**

java

Copy code

@Controller

public class MyWebController {

@GetMapping("/home")

public String home(Model model) {

model.addAttribute("message", "Welcome to Spring Boot!");

return "home";

}

}

**@RequestMapping**

* **Purpose:** Maps HTTP requests to handler methods of MVC and REST controllers.
* **Usage:** Can be applied to classes and methods.
* **Example:**

java

Copy code

@RequestMapping("/api")

public class ApiController {

@RequestMapping("/greet")

public String greet() {

return "Hello, API!";

}

}

**@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping**

* **Purpose:** Shortcut annotations for HTTP method-specific mappings.
* **Examples:**

java

Copy code

@GetMapping("/users")

public List<User> getUsers() {

return userService.getAllUsers();

}

@PostMapping("/users")

public User createUser(@RequestBody User user) {

return userService.saveUser(user);

}

**@PathVariable**

* **Purpose:** Binds a method parameter to a URI template variable.
* **Example:**

java

Copy code

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

public User getUser(@PathVariable Long id) {

return userService.getUserById(id);

}

**@RequestParam**

* **Purpose:** Extracts query parameters from the request URL.
* **Example:**

java

Copy code

@GetMapping("/search")

public List<User> searchUsers(@RequestParam String name) {

return userService.searchByName(name);

}

**@RequestBody**

* **Purpose:** Binds the request body to a method parameter.
* **Example:**

java

Copy code

@PostMapping("/users")

public User createUser(@RequestBody User user) {

return userService.saveUser(user);

}

**@ResponseBody**

* **Purpose:** Binds a method's return value to the HTTP response body.
* **Example:**

java

Copy code

@GetMapping("/message")

@ResponseBody

public String message() {

return "Direct response body";

}

**3. Dependency Injection Annotations**

**@Autowired**

* **Purpose:** Automatically wires a bean by type into the dependent component.
* **Example:**

java

Copy code

@Service

public class MyService {

@Autowired

private UserRepository userRepository;

}

**@Qualifier**

* **Purpose:** Specifies which bean to inject when multiple beans of the same type exist.
* **Example:**

java

Copy code

@Service

public class MyService {

@Autowired

@Qualifier("specialUserRepository")

private UserRepository userRepository;

}

**@Primary**

* **Purpose:** Marks a bean as the primary candidate for autowiring.
* **Example:**

java

Copy code

@Bean

@Primary

public DataSource mainDataSource() {

return new DataSource();

}

**@Lazy**

* **Purpose:** Marks a bean to be initialized lazily (when first accessed).
* **Example:**

java

Copy code

@Lazy

@Service

public class HeavyService {

// Loaded lazily

}

**4. Database and JPA Annotations**

**@Entity**

* Marks a class as a JPA entity.

**@Table**

* Maps an entity to a database table.

**@Id**

* Marks a field as the primary key.

**@GeneratedValue**

* Specifies the generation strategy for primary keys.

**@Repository**

* Marks a class as a DAO (Data Access Object) for database operations.

**5. Security Annotations**

**@EnableWebSecurity**

* Enables Spring Security for the application.

**@PreAuthorize**

* Applies authorization checks before a method is executed.

**6. Testing Annotations**

**@SpringBootTest**

* Loads the full application context for integration tests.

**@WebMvcTest**

* Loads only the web layer of the application for testing.

**@DataJpaTest**

* Configures an in-memory database for JPA-related tests.