Spring Boot Annotations

Spring Boot Annotations – Complete Notes with Usage

Part 1: Core Spring Annotations

1. @Component

Usage: Marks a class as a Spring-managed bean. Generic stereotype for any component. Spring auto-detects it during component scanning.

```
@Component
class MyBean { }
```

Custom bean name:

```
@Component("myTestBean")
class MyBean { }
```

2. @Controller

Usage: Marks a class as a Spring MVC controller (handles web requests in MVC projects).

```
@Controller
public class MyController { }
```

3. @RestController

Usage: Marks a class as a REST controller (combines @Controller + @ResponseBody). Returns JSON/XML responses.

```
@RestController
public class MyRestController { }
```

4. @Service

Usage: Marks a class as a service layer component. Holds business logic.

```
@Service
public class MyService { }
```

---

5. @Repository

Usage: Marks a class as a data access layer component. Handles database operations. Also enables exception translation.

```
@Repository
public class MyRepository { }
```

---

6. @Configuration

Usage: Marks a class as a configuration class (alternative to XML). Can declare beans with @Bean.

```
@Configuration
public class AppConfig {
   @Bean
   public RestTemplate restTemplate() {
      return new RestTemplate();
   }
}
```

---

7. @Bean

Usage: Declares a bean inside a @Configuration class. The method name is the bean ID by default.

```
@Bean
public Employee employee() {
   return new Employee();
}
```

---

8. @ComponentScan

Usage: Tells Spring which packages to scan for components (beans, services, controllers, etc.).

```
@Configuration
@ComponentScan(basePackages = {"com.example.package1", "com.exam
ple.package2"})
public class AppConfig { }
```

9. @Import

Usage: Imports multiple @Configuration classes, grouping configurations.

```
@Configuration
@Import({DataSourceConfig.class, MyCustomConfig.class})
public class AppConfig { }
```

10. @PropertySource / @PropertySources

Usage: Loads external properties files into Spring Environment.

```
@Configuration
@PropertySource("classpath:app.properties")
public class MyClass { }

@Configuration
@PropertySources({
   @PropertySource("classpath:app1.properties"),
   @PropertySource("classpath:app2.properties")
})
public class MyClass { }
```

11. @Value

Usage: Injects property values from property files or environment variables. Can provide defaults.

```
@Value("${server.ip}")
private String serverIP;

@Value("${emp.department:Admin}")
private String empDepartment;

@Value("${columnNames}")
private String[] columnNames;
```

Part 2: Spring Boot Specific Annotations

1. @SpringBootApplication

Usage: Main class annotation. Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. Boots the application.

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

2. @EnableAutoConfiguration

Usage: Automatically configures Spring context based on dependencies on the classpath.

```
@EnableAutoConfiguration
public class MyApp { }
```

Excluding classes:

```
@EnableAutoConfiguration(exclude = {WebSocketMessagingAutoConfiguration.class})
```

```
@EnableAutoConfiguration(excludeName = {"org.springframework.boot.au
toconfigure.websocket.servlet.WebSocketMessagingAutoConfiguration"})
```

3. @SpringBootConfiguration

Usage: Alternative to @Configuration. Automatically discovered. Useful in tests.

```
@SpringBootConfiguration
public class MyApp {
    @Bean
    public EmployeeService employeeService() {
        return new EmployeeServiceImpl();
    }
}
```

4. @ConfigurationProperties

Usage: Binds properties (from application.properties or YAML) to a bean class.

```
@ConfigurationProperties(prefix="dev")
public class MyDevAppProperties {
    private String name;
    private int port;
    private String dburl;
    private String dbname;
    private String dbuser;
    private String dbpassword;
    // getters & setters
}
```

5. @EnableConfigurationProperties

Usage: Registers a @ConfigurationProperties bean in the Spring context.

```
@Configuration
@EnableConfigurationProperties(MyDevAppProperties.class)
```

```
public class MySpringBootDevApp { }
```

---

6. @EnableConfigurationPropertiesScan

Usage: Scans packages for all @ConfigurationProperties beans.

```
@SpringBootApplication
@EnableConfigurationPropertiesScan("com.dev.spring.test.annotation")
public class MyApplication { }
```

---

7. @EntityScan & @EnableJpaRepositories

Usage: Specifies packages to scan for JPA entity classes and repositories.

```
@EntityScan(basePackages = "com.dev.springboot.examples.entity")
@EnableJpaRepositories(basePackages = "com.dev.springboot.examples.j
pa.repositories")
```

Part 3: Spring Data JPA Annotations

1. @Entity

Marks a class as a JPA entity.

```
@Entity
public class User { }
```

---

2. @Table

Specifies table name.

```
@Table(name="users")
```

---

3. @Id

Marks a field as primary key.

```
@Id
private Long id;
```

4. @GeneratedValue

Auto-generates primary key values.

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

5. @Column

Specifies column mapping.

```
@Column(name="username", nullable=false)
private String username;
```

6. Relationship Annotations

- **@OneToOne, @OneToMany, @ManyToOne, @ManyToMany**
- **@JoinColumn, @JoinTable**
- **@Embeddable, @Embedded**
- **@Transient, @Lob**
- **@Enumerated**

Examples:

```
@OneToMany(mappedBy="user")
private List<Order> orders;

@Embedded
private Address address;

@Enumerated(EnumType.STRING)
private Status status;
```

Part 4: Spring Security Annotations

- @EnableWebSecurity – Enable security config
- @EnableMethodSecurity – Enable method-level security
- @Secured("ROLE_ADMIN") – Restrict access by role
- @RolesAllowed – Define allowed roles
- @PreAuthorize / @PostAuthorize – Expression-based security
- @AuthenticationPrincipal – Inject currently authenticated user
- @WithMockUser – Mock a user for testing
- @PermitAll / @DenyAll – Allow or deny all access

Example:

```
@Secured("ROLE_ADMIN")
public void deleteUser() { }

@PreAuthorize("#username == authentication.name")
public void updateProfile(String username) { }

@GetMapping("/profile")
public String profile(@AuthenticationPrincipal UserDetails user) {
    return user.getUsername();
}
```

Part 5: Validation Annotations

- **@NotNull, @NotEmpty, @NotBlank** – Non-null/empty validation
- **@Email, @Pattern** – Validate email or regex
- **@Min, @Max, @Positive, @Negative** – Numeric constraints
- **@Size** – String/Collection size
- **@Past, @Future** – Date validation
- **@Valid** – Nested object validation
- **@AssertTrue, @AssertFalse** – Boolean validation

- **@Digits, @Length** – Numeric and string length

Example:

```
@NotBlank
private String username;

@Email
private String email;

@Size(min=3, max=20)
private String password;
```

Part 6: Scheduling Annotations

- @EnableScheduling – Enable scheduling support
- @Scheduled – Define scheduled tasks

Examples:

```
@Scheduled(fixedRate = 5000)
public void runTask() { }

@Scheduled(cron = "0 0 10 * * *")
public void runDailyTask() { }
```

Part 7: Caching Annotations

- @EnableCaching – Enable caching support
- @Cacheable – Cache method result
- @CachePut – Update cache without skipping method execution
- @CacheEvict – Remove cache entry
- @CacheConfig – Class-level cache configuration

Examples:

```
@Cacheable("users")
public User getUser(int id) { return repo.findById(id).get(); }

@CachePut("users")
public User updateUser(User user) { ... }

@CacheEvict(value="users", allEntries=true)
public void clearCache() { }
```