Spring Boot Annotations

Spring Boot Annotations and Their Uses

Simplifying Application Development with Annotations

Name: Rama Shanker

Date: 28/01/2025

Core Annotations: @SpringBootApplication

Combines: **@Configuration @EnableAutoConfiguration, and @ComponentScan**

Entry point for Spring Boot apps.

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

Configuration & Bean Management

Combines three annotations:

@Configuration : Defines beans and configuration.

@EnableAutoConfiguration : Enables Spring Boot auto-configuration feature.

@ComponentScan : Scans for components, services, and repositories.

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

@RestController and @RequestMapping

@RestController: Combines @Controller and @ResponseBody. Used for REST APIs.

@RequestMapping: Maps HTTP requests to handler methods.

@GetMapping/@PostMapping: Shortcuts for specific HTTP methods.

```
@RestController
@RequestMapping("/api")
public class MyController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, Spring Boot!";
    }
}
```

Property Management

@**Value:** Injects values from properties files.

@**ConfigurationProperties:** Binds properties to POJOs.

```
@Value("${app.api.key}")
private String apiKey;

@ConfigurationProperties(prefix = "app")
public class AppConfig {
  private String apiKey;
  // getters/setters
}
```

@Service and @Repository

@Service:

Marks a class as a service that contains business logic.

@Repository:

Indicates a class as a data access layer.

Handles database operations and exception translation.

```
@Service
public class UserService {
    public String getUserById(Long id) {
        return "User: " + id;
    }
}

@Repository
public interface UserRepository extends JpaRepository<User, Long> {}
```

@Entity and @Table

Used in the model layer to map Java objects to database tables.

@Entity: Marks a class as a JPA entity.

@Table: Specifies table name (optional).

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

    // Getters and Setters
}
```

@Autowired

Used for dependency injection.

Automatically wires beans by type.

```
@Service
public class OrderService {
    @Autowired
    private UserRepository userRepository;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }
}
```

@Configuration and @Bean

@Configuration:

Marks a class for defining beans.

@Bean:

Indicates that a method creates a bean.

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

Spring Boot Actuator Annotations

@Endpoint: Exposes a custom actuator endpoint.

@ReadOperation: Marks a read operation in the endpoint.

```
@Component
@Endpoint(id = "customEndpoint")
public class CustomEndpoint {
    @ReadOperation
    public String customStatus() {
        return "Custom Actuator Endpoint: UP";
    }
}
```

@Enable… Annotations

@EnableAutoConfiguration:

Enables Spring Boot's auto-configuration.

@EnableScheduling:

Enables scheduled tasks.

@EnableCaching:

Enables caching.

```
@SpringBootApplication
@EnableScheduling
public class SchedulerApp {
    @Scheduled(fixedRate = 5000)
    public void scheduledTask() {
        System.out.println("Task executed at: " + new Date());
    }
}
```