# Spring and Spring Boot

## Spring Architecture

Spring Framework is a comprehensive framework for Java applications, providing infrastructure support at multiple layers. The core components of Spring architecture include:

1. **Core Container**
   - **BeanFactory:** Manages beans and dependencies.
   - **ApplicationContext:** Extends BeanFactory with additional features.
2. **Spring AOP (Aspect-Oriented Programming)**
   - Enables modularization of cross-cutting concerns (logging, security, etc.).
3. **Spring JDBC**
   - Simplifies database interaction with templates.
4. **Spring ORM**
   - Integration with ORM frameworks like Hibernate, JPA.
5. **Spring Web**
   - Supports web applications using MVC pattern.
6. **Spring Security**
   - Provides authentication and authorization features.
7. **Spring Test**
   - Supports unit and integration testing.

### How Spring Helps Us

**Spring provides multiple benefits, including:**

- **Dependency Injection (DI):** Reduces tight coupling between components.
- **Aspect-Oriented Programming (AOP):** Enables cleaner code by separating cross-cutting concerns.

- **Declarative Transaction Management:** Simplifies handling transactions.
- **Integration with Third-Party Frameworks:** Works seamlessly with Hibernate, JPA, etc.
- **Testability:** Enhances unit testing capabilities with dependency injection.

## Spring Features with Examples and Diagrams

### 1. Dependency Injection (DI)

- Allows injection of dependencies at runtime.

```
@Component
public class Car {
    private Engine engine;
    @Autowired
    public Car(Engine engine) {
        this.engine = engine;
    }
}
```

### 2. Aspect-Oriented Programming (AOP)

- **Example of logging aspect:**

```
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*(..))")
    public void logBefore() {
        System.out.println("Logging before method execution");
    }
}
```

### 3. Spring MVC

- Implements Model-View-Controller pattern.

```
@Controller
public class HomeController {
    @GetMapping("/")
    public String home() {
        return "home";
    }
}
```

### 4. Spring Security

- **Securing a web application:**

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
    @Override
    protected void configure(HttpSecurity http) throws Exception
{
        http.authorizeRequests().antMatchers("/
admin").authenticated().and().formLogin();
    }
}
```

## Spring vs Spring Boot

### How Spring Boot Overcomes Spring

| Feature | Spring | Spring Boot |
|---|---|---|
| Configuration | Requires XML/Java-based configuration | Auto-configuration simplifies setup |
| Dependency Management | Manual dependency resolution | Comes with pre-configured dependencies |
| Embedded Server | Requires external Tomcat, Jetty | Embedded Tomcat, Jetty, Undertow |
| Microservices | Requires additional configurations | Built-in support for microservices |
| Production-Ready | Needs extra setup for monitoring | Includes Actuator for monitoring |

## Spring Boot Features

1. **Auto-Configuration:** Eliminates manual configuration.
2. **Embedded Web Server:** Uses Tomcat, Jetty, or Undertow.
3. **Spring Boot Starter Dependencies:** Pre-configured dependencies simplify setup.
4. **Spring Boot Actuator:** Provides built-in monitoring and health checks.
5. **Microservices Support:** Built-in tools for developing microservices applications.

## Spring Boot Annotations with Examples

Spring Boot provides various annotations to simplify development. Below are the most commonly used annotations:

### 1. @SpringBootApplication

- Entry point for a Spring Boot application.

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

### 2. @RestController

- Combines `@Controller` and `@ResponseBody`.

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

### 3. @RequestMapping

- Maps HTTP requests to handler methods.

```
@RestController
@RequestMapping("/api")
public class ApiController {
    @GetMapping("/users")
    public List getUsers() {
        return Arrays.asList("User1", "User2");
    }
}
```

## 4. @Component, @service, @Repository

- Used for component scanning and dependency injection.

```
@Component
public class MyComponent {
    public void doSomething() {
        System.out.println("Component doing work");
    }
}
```

## 5. @Autowired

- Injects dependencies automatically.

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

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

## 6. @Value

- Injects values from properties files.

```
@Component
public class ConfigReader {
    @Value("${app.name}")
    private String appName;
}
```

### 7. @Configuration & @Bean

- Used to define beans manually.

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

### 8. @EnableAutoConfiguration

- Enables Spring Boot's auto-configuration feature.

```
@EnableAutoConfiguration
public class MyApplication {}
```

### 9. @ConditionalOnProperty

- Enables configuration based on property values.

```
@Configuration
@ConditionalOnProperty(name = "feature.enabled", havingValue =
"true")
public class FeatureConfig {}
```

### 10. @SpringBootTest

- Used for testing Spring Boot applications.

```
@SpringBootTest
public class MyApplicationTests {
    @Test
    public void contextLoads() {}
}
```

# Spring Boot:Important Interview Questions and Answers

## 1. What is Spring Boot?

Spring Boot is an extension of the Spring framework that simplifies application development by offering convention-over-configuration, embedded servers, and a range of production-ready features like monitoring and health checks.

## 2. What are the key features of Spring Boot?

- **Auto-Configuration:** Reduces manual configuration by automatically setting up application context.
- **Standalone Applications:** Runs without requiring an external web server like Tomcat.
- **Spring Boot Starters:** Provide dependencies to simplify development.
- **Spring Boot Actuator:** Enables monitoring and management.
- **Embedded Servers:** Supports Tomcat, Jetty, and Undertow.
- **Microservices Support:** Facilitates building microservices-based architectures.

## 3. What are Spring Boot Starters?

Spring Boot Starters are dependency descriptors that simplify adding dependencies in a Spring Boot project. Example:

- spring-boot-starter-web for web applications
- spring-boot-starter-data-jpa for database interactions

## 4. What is Spring Boot Auto-Configuration?

Spring Boot auto-configures beans based on the project's classpath settings. It reduces the need for manual configurations by using @EnableAutoConfiguration.

## 5. How does Spring Boot work internally?

1. Reads spring-boot-starter dependencies.
2. Uses @SpringBootApplication (combines @Configuration,

@EnableAutoConfiguration, and @ComponentScan).

3. Checks for available beans and auto-configures them.
4. Runs the application via an embedded web server.

## 6. What is @SpringBootApplication?

It is a combination of:

- @Configuration: Allows defining beans.
- @EnableAutoConfiguration: Enables auto-configuration.
- @ComponentScan: Scans components in the package.

## 7. How to create a Spring Boot application?

1. Use Spring Initializr (https://start.spring.io/).
2. Add dependencies (spring-boot-starter-web, spring-boot-starter- data-jpa).
3. Create the @SpringBootApplication class.
4. Run using SpringApplication.run(Application.class, args).

## 8. What is Spring Boot CLI?

Spring Boot CLI is a command-line tool for quickly developing Spring applications using Groovy scripts.

## 9. What is Spring Boot Actuator?

Spring Boot Actuator provides production-ready features like health checks, metrics, and environment details. Key endpoints:

- /actuator/health – Shows application health.
- /actuator/info – Displays app information.
- /actuator/metrics – Provides metrics.

## 10. What is the difference between Spring Boot and Spring MVC?

| Feature | Spring MVC | Spring Boot |
|---|---|---|
| Setup | Requires configuration | Auto-configured |
| Server | Needs external server | Uses embedded server |
| Dependencies | Manually managed | Uses starters |

## 11. What are the different ways to run a Spring Boot application?

- Using mvn spring-boot:run.

- Running the main class with SpringApplication.run().
- Packaging as a JAR and running java -jar app.jar.

## 12. What is application.properties?

A configuration file used to define database connection settings, server port, logging, etc. Example:

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
```

## 13. How to handle exceptions in Spring Boot?

Using @ControllerAdvice and @ExceptionHandler: @ControllerAdvice

```
public class GlobalExceptionHandler {
@ExceptionHandler(Exception.class)
public ResponseEntity<String> handleException(Exception ex)
{
return new ResponseEntity<>(ex.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
}
}
```

## 14. What is Spring Boot DevTools?

DevTools enables live reloading, property changes without restarting, and enhances the development experience.

## 15. What are Profiles in Spring Boot?

Profiles allow defining different configurations for different environments.

```
spring.profiles.active=dev
```

## 16. How does Spring Boot support Microservices?

- Embedded server for lightweight deployment.
- spring-cloud for service discovery (Eureka), configuration (Config Server), and API gateways (Zuul).

## 17. What is Spring Boot Security?

Spring Security provides authentication and authorization using filters, OAuth2, JWT, and session management.

## 18. What is Spring Boot Data JPA?

Spring Data JPA simplifies database access by eliminating boilerplate code and using

JpaRepository.

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

## 19. What is the use of @RestController?

Combines @Controller and @ResponseBody to create RESTful web services.

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

## 20. What is Spring Boot Kafka?

Spring Boot provides Kafka integration using spring-kafka dependency for producing and consuming messages.

## 21. How to implement Logging in Spring Boot?

Using SLF4J and Logback:

```
private static final Logger logger =
LoggerFactory.getLogger(MyClass.class); logger.info("Application
started");
```

## 22. What is Circuit Breaker in Spring Boot?

A fault-tolerance mechanism using Resilience4j or Hystrix to prevent cascading failures.

## 23. How to handle transactions in Spring Boot?

Using @Transactional annotation.

```
@Transactional
public void transferMoney() { accountService.debit(accountId,
amount); accountService.credit(accountId, amount);
}
```

## 24. What is Swagger in Spring Boot?

Swagger provides API documentation for RESTful services using springdoc- openapi-ui dependency.

## 25. What are the differences between @Component, @Service, and @Repository?

| Annotation | Purpose |
|---|---|
| @Component | Generic Spring-managed component |
| @Service | Business logic layer component |
| @Repository | DAO (Data Access Object) component |