### **PHẦN 1**

### **1. Dependency Injection (DI) là gì?**

* **Dependency Injection (DI)** là một mẫu thiết kế trong đó sự phụ thuộc của một lớp được truyền từ bên ngoài vào thay vì tạo bên trong lớp đó.
* DI giúp tách biệt giữa khởi tạo đối tượng và việc sử dụng chúng, giúp mã nguồn dễ bảo trì và kiểm thử hơn.

### **2. Có những loại DI nào? Lợi ích của DI?**

* **Các loại DI trong Spring:**
  + **Constructor Injection** – Tiêm phụ thuộc qua constructor.
  + **Setter Injection** – Tiêm phụ thuộc qua setter method.
  + **Field Injection** – Tiêm trực tiếp vào biến bằng @Autowired.
* **Lợi ích của DI:**
  + Giảm phụ thuộc chặt chẽ (tight coupling).
  + Dễ kiểm thử (unit test).
  + Tăng tính module hóa.

### **3. Bạn thích loại DI nào hơn?**

* **Constructor Injection** thường được ưa chuộng hơn do:
  + Giúp class trở nên bất biến.
  + Dễ kiểm thử.
  + Tránh lỗi NullPointerException do thiếu dependency.

### **4. Cơ chế Inversion of Control (IoC) trong Spring Container hoạt động thế nào?**

* **IoC** là nguyên tắc mà Spring Container chịu trách nhiệm quản lý vòng đời của bean.
* Container sẽ khởi tạo, cấu hình và quản lý các dependency thay vì lập trình viên làm thủ công.

### **5. Sự khác nhau giữa BeanFactory và ApplicationContext?**

* BeanFactory: Cơ bản, lazy loading.
* ApplicationContext: Mở rộng BeanFactory, hỗ trợ event, AOP, và eager loading.

### **6. Sự khác nhau giữa ApplicationContext và BeanContext?**

* ApplicationContext quản lý toàn bộ lifecycle của bean.
* BeanContext không phải là khái niệm trong Spring, có thể đề cập đến BeanFactory.

### **7. Vòng đời của một bean trong Spring?**

* **Instantiation** → **Dependency Injection** → **PostConstruct (@PostConstruct)** → **Ready to use** → **PreDestroy (@PreDestroy)** → **Destroyed**.

### **8. Các phạm vi (scope) của bean?**

* singleton – Mặc định, một instance duy nhất.
* prototype – Mỗi lần gọi tạo một instance mới.
* request – Mỗi request HTTP có một instance.
* session – Mỗi session có một instance.
* application – Dùng chung trong toàn bộ ServletContext.

### **9. Stateless bean trong Spring là gì?**

* Stateless bean không lưu trữ trạng thái, thường được dùng trong REST API (ví dụ: @Service).

### **10. Làm sao để xử lý dependency vòng tròn (Cyclic Dependency)?**

* Dùng **@Lazy** hoặc tách dependency bằng cách sử dụng **EventListener** hoặc **Factory Bean**.

### **11. Xử lý ngoại lệ trong Spring?**

* **Cách phổ biến:**
  + @ExceptionHandler
  + @ControllerAdvice
  + @ResponseStatus
  + HandlerExceptionResolver

### **12. Bộ lọc (Filter) trong Spring hoạt động như thế nào?**

* Spring Filter chạy trước Servlet xử lý request.
* Có thể dùng @Component hoặc cấu hình trong FilterRegistrationBean.

### **13. Singleton bean có xử lý được nhiều request song song không?**

* **Có**, nhưng cần đảm bảo **thread-safety**.

### **14. Các design pattern trong Spring Framework?**

* **Dependency Injection** – Giảm coupling.
* **Factory Pattern** – Tạo bean.
* **Singleton Pattern** – Quản lý instance bean.
* **Proxy Pattern** – Dùng trong AOP.
* **Template Method** – Trong JDBC Template.

### **15. Singleton bean có thread-safe không?**

* **Không**, trừ khi ta sử dụng synchronized hoặc các công cụ hỗ trợ thread-safe như ConcurrentHashMap.

### **16. Factory Design Pattern hoạt động như thế nào trong Spring?**

* Spring sử dụng Factory Pattern để tạo và quản lý bean.

### **17. Nếu gọi singleton bean từ prototype bean thì có bao nhiêu instance?**

* Singleton bean **luôn có một instance**.
* Prototype bean **luôn tạo một instance mới**.

### **18. Làm thế nào để tạo prototype bean?**

* Đánh dấu bean bằng @Scope("prototype").

### **19. Sự khác nhau giữa Method Overloading và Method Overriding?**

* **Overloading**: Cùng tên khác tham số.
* **Overriding**: Ghi đè method từ superclass.

### **20. AOP là gì? Annotation nào được dùng?**

* **Aspect-Oriented Programming (AOP)** giúp tách biệt các logic chung.
* **Annotation AOP:**
  + @Aspect
  + @Before
  + @After
  + @Around
  + @Pointcut

### **21. @Primary và @Qualifier là gì?**

* @Primary: Chỉ định bean mặc định khi có nhiều bean cùng loại.
* @Qualifier: Xác định bean cụ thể khi tiêm dependency.

### **22. CDI (Contexts and Dependency Injection) là gì?**

* **CDI** là công nghệ DI trong Java EE.
* **Spring hỗ trợ CDI**, nhưng Spring DI linh hoạt hơn.

**PHẦN 2**

### **1. Phương thức gọi trước khi Spring Boot application bắt đầu tải?**

* **@PostConstruct**: Đánh dấu phương thức chạy sau khi bean được khởi tạo.
* **CommandLineRunner & ApplicationRunner**: Chạy code sau khi ứng dụng khởi động.

### **2. Sự khác biệt giữa Spring Boot và Spring?**

| **Tiêu chí** | **Spring Framework** | **Spring Boot** |
| --- | --- | --- |
| Cấu hình | Phải cấu hình thủ công | Cấu hình tự động |
| Server | Phải cài đặt Tomcat, Jetty | Hỗ trợ embedded server |
| Dependency | Quản lý phụ thuộc thủ công | Dùng Spring Boot Starter |
| Phát triển nhanh | Cần nhiều cấu hình XML hoặc Java | Tối giản hóa quy trình |

### **3. Điểm entry của Spring Boot và cách hoạt động của @SpringBootApplication?**

**Entry Point**: Lớp chứa phương thức main(), ví dụ:

@SpringBootApplication

public class MyApp {

public static void main(String[] args) {

SpringApplication.run(MyApp.class, args);

}

}

* @SpringBootApplication = @Configuration + @EnableAutoConfiguration + @ComponentScan

### **4. Một số annotation trong Spring Boot?**

* @RestController: Kết hợp @Controller và @ResponseBody
* @Service: Đánh dấu class là service
* @Repository: Đánh dấu class thao tác với DB
* @Component: Annotation cơ bản của Spring

### **5. Khác biệt giữa @Component, @Service, @Repository, @Controller?**

* @Component: Annotation chung nhất
* @Service: Đánh dấu lớp service
* @Repository: Làm việc với database, hỗ trợ exception translation
* @Controller: Xử lý request trong MVC

### **6. Component Scan là gì?**

* Tự động phát hiện bean và đăng ký vào Spring Context.
* Dùng @ComponentScan để chỉ định packages quét.

### **7. Cách Spring Boot tự động phát hiện Bean?**

* Dựa trên @ComponentScan và @EnableAutoConfiguration

### **8. Làm thế nào để loại trừ một cấu hình?**

* Dùng @SpringBootApplication(exclude = {DataSourceAutoConfiguration.class})
* Hoặc spring.autoconfigure.exclude trong application.properties

### **9. Làm idempotent method trong Spring Boot?**

* Sử dụng PUT thay vì POST
* Kiểm tra request trùng lặp bằng Idempotency Key
* Dùng synchronized hoặc database locks

### **10. Spring Boot Profile là gì?**

* Quản lý cấu hình cho nhiều môi trường (dev, test, prod).

Ví dụ:  
# application-dev.properties

server.port=8081

* Chạy với spring.profiles.active=dev

### **11. Làm thế nào để thiết lập properties cho các môi trường khác nhau (Dev, QA, PROD)?**

**Sử dụng application-{profile}.properties hoặc application-{profile}.yml**# application-dev.properties

server.port=8081

# application-prod.properties

server.port=8082

* Kích hoạt profile khi chạy ứng dụng:  
   bash  
    
  java -jar myapp.jar --spring.profiles.active=dev

### **12. Spring Transaction Management là gì?**

* **Spring Transaction Management** giúp quản lý giao dịch (commit/rollback) khi thao tác với database.

### **13. Cách sử dụng transaction trong Spring Boot?**

Dùng @Transactional trong Service:  
@Service

public class UserService {

@Autowired

private UserRepository userRepository;

@Transactional

public void updateUser(Long id) {

User user = userRepository.findById(id).orElseThrow();

user.setName("Updated Name");

userRepository.save(user);

}

}

### **4. Cách xử lý transaction và các mức độ isolation?**

* Các mức độ isolation phổ biến:

| **Isolation Level** | **Ý nghĩa** |
| --- | --- |
| READ\_COMMITTED | Chỉ đọc dữ liệu đã commit |
| READ\_UNCOMMITTED | Đọc cả dữ liệu chưa commit (có thể gây lỗi "dirty read") |
| REPEATABLE\_READ | Giữ nguyên dữ liệu đọc trong suốt transaction |
| SERIALIZABLE | Transaction chạy tuần tự, tránh xung đột |

* Ví dụ:  
  @Transactional(isolation = Isolation.REPEATABLE\_READ)

### **5. Spring Boot là gì? Các mục tiêu và tính năng chính?**

* **Spring Boot** là framework giúp tạo ứng dụng Spring nhanh hơn bằng cách cung cấp:
  + Cấu hình tự động (@EnableAutoConfiguration)
  + Embedded server (Tomcat, Jetty, Undertow)
  + Ít cấu hình XML hơn
  + Tích hợp Spring Security, Spring Data, Spring Cloud dễ dàng

### **6. So sánh Spring Boot vs Spring Framework vs Spring MVC**

| **Tiêu chí** | **Spring Framework** | **Spring MVC** | **Spring Boot** |
| --- | --- | --- | --- |
| Cấu hình | Cấu hình XML nhiều | MVC-based, phải tự config | Ít cấu hình, tự động |
| Web Server | Phải cài đặt | Dùng Tomcat hoặc Jetty | Embedded Server |
| Dependency | Phải quản lý thủ công | Có hỗ trợ | Dùng Spring Boot Starter |

### **7. Embedded Server là gì? Spring Boot dùng server nào mặc định?**

* **Embedded Server** là server chạy bên trong ứng dụng Spring Boot mà không cần deploy WAR.
* **Mặc định:** Tomcat
* **Hỗ trợ:** Jetty, Undertow

### **8. Starter Projects là gì? Một số ví dụ?**

* **Starter Projects** là các dependency giúp tích hợp nhanh.
  + spring-boot-starter-web: Web app (Tomcat, Spring MVC)
  + spring-boot-starter-data-jpa: JPA/Hibernate
  + spring-boot-starter-security: Spring Security

### **9. Spring Initializr là gì?**

* **Spring Initializr** là công cụ tạo nhanh project Spring Boot:  
   👉<https://start.spring.io/>

### **10. Spring Boot Actuator là gì? Cách giám sát ứng dụng bằng Actuator?**

* **Spring Boot Actuator** cung cấp endpoints để giám sát ứng dụng.
* Kích hoạt bằng cách thêm dependency:  
  <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

* Một số endpoints hữu ích:
  + /actuator/health → Kiểm tra trạng thái
  + /actuator/info → Xem thông tin app

**PHẦN 3**

### **1. Spring MVC Flow là gì?**

* **Spring MVC Flow** bao gồm các bước:
  1. **Client gửi request** → Tới Front Controller (DispatcherServlet)
  2. **DispatcherServlet gọi HandlerMapping** → Tìm Controller phù hợp
  3. **Controller xử lý request** → Trả về ModelAndView
  4. **ViewResolver xác định View** (JSP, Thymeleaf,...)
  5. **View hiển thị dữ liệu** đến client

### **2. Sự khác biệt giữa @Controller và @RestController?**

* @Controller: Trả về **view name** (JSP, Thymeleaf,...).
* @RestController: Kết hợp @Controller và @ResponseBody, trả về JSON hoặc XML.

### **3. @ResponseBody có ý nghĩa gì?**

Dùng để **trả về dữ liệu JSON/XML** thay vì một view.  
@GetMapping("/hello")

@ResponseBody

public String sayHello() {

return "Hello World";

}

### **4. Các phương thức HTTP trong REST?**

| **Method** | **Chức năng** |
| --- | --- |
| GET | Lấy dữ liệu |
| POST | Tạo mới tài nguyên |
| PUT | Cập nhật toàn bộ tài nguyên |
| PATCH | Cập nhật một phần tài nguyên |
| DELETE | Xóa tài nguyên |

### **5. Phương thức Idempotent trong REST là gì?**

* **Idempotent**: Gửi nhiều lần vẫn cho kết quả như nhau (GET, PUT, DELETE).
* **Non-Idempotent**: Gửi nhiều lần có thể thay đổi dữ liệu (POST).

### **6. Tiêu chuẩn xây dựng REST API tốt?**

* **Stateless**: Không lưu trạng thái request trước đó.
* **Proper HTTP Methods**: Dùng đúng phương thức HTTP.
* **Use Proper Status Codes**: 200 OK, 201 Created, 400 Bad Request,...
* **Security**: Dùng OAuth2, JWT, HTTPS.

### **7. Sự khác biệt giữa POST và PUT?**

| **Tiêu chí** | **POST** | **PUT** |
| --- | --- | --- |
| Chức năng | Tạo mới | Cập nhật |
| Idempotent | ❌ Không | ✅ Có |
| Khi không có tài nguyên | Tạo mới | Có thể tạo mới |

### **8. Header trong HTTP chứa gì? Có thể intercept header không?**

* **Header có thể chứa**: Authorization (JWT), Content-Type, Accept,...
* **Intercept Header** bằng HandlerInterceptor:  
    
  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {

String token = request.getHeader("Authorization");

// Xử lý token

return true;

}

### **9. Cách bảo mật REST API?**

* **Dùng HTTPS**
* **JWT Authentication**
* **OAuth2 Authorization**
* **Rate Limiting** để tránh DDoS

### **10. Cách truyền tham số trong request (URL vs JSON)?**

* **URL Parameter** (GET):  
  @GetMapping("/user/{id}")

public User getUser(@PathVariable Long id) { ... }

* **JSON Body** (POST):  
  @PostMapping("/user")

public User createUser(@RequestBody User user) { ... }

### **11. Model 1 vs Model 2 Architecture trong MVC?**

* **Model 1**: Không có Controller, View (JSP) xử lý cả logic và UI.
* **Model 2 (MVC)**: Có Controller xử lý logic, View chỉ hiển thị UI.

### **12. Ví dụ Controller trong Spring MVC**

@Controller

public class HomeController {

@RequestMapping("/home")

public String home(Model model) {

model.addAttribute("message", "Hello Spring MVC");

return "home";

}

}

### **13. PathVariable, Model Attribute, Session Attribute là gì?**

* @PathVariable: Lấy tham số từ URL.
* @ModelAttribute: Lưu đối tượng vào Model.
* @SessionAttribute: Lưu dữ liệu vào session.

### **14. Controller Advice và @ExceptionHandler?**

**Dùng để xử lý lỗi tập trung**:  
@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleException(Exception e) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(e.getMessage());

}

}

### **15. Cách xử lý Exception trong Web App?**

* Dùng @ExceptionHandler
* Dùng @ControllerAdvice
* Trả về ResponseEntity với mã lỗi HTTP

### **16. Ví dụ xử lý lỗi cụ thể trong Spring MVC?**

@ControllerAdvice

public class CustomExceptionHandler {

@ExceptionHandler(UserNotFoundException.class)

public ResponseEntity<String> handleUserNotFound(UserNotFoundException e) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User Not Found");

}

}

### **17. Ví dụ PostResource trong REST API dùng ResponseEntity**

@RestController

@RequestMapping("/posts")

public class PostController {

@PostMapping

public ResponseEntity<Post> createPost(@RequestBody Post post) {

return ResponseEntity.status(HttpStatus.CREATED).body(post);

}

}

### **18. HATEOAS là gì?**

* HATEOAS (**Hypermedia as the Engine of Application State**) giúp API cung cấp **liên kết động**.
* **Ví dụ Response HATEOAS:**{

"id": 1,

"name": "John Doe",

"\_links": {

"self": { "href": "/users/1" },

"orders": { "href": "/users/1/orders" }

}

}

### **19. Swagger là gì?**

* **Swagger** giúp tạo **tài liệu API tự động**.
* Thêm dependency:  
  <dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-boot-starter</artifactId>

<version>3.0.0</version>

</dependency>

* Truy cập tài liệu API:  
   🔗 http://localhost:8080/swagger-ui/

### **20. Tóm tắt Swagger UI**

* Cung cấp giao diện UI cho REST API.
* Hỗ trợ test API trực tiếp trên trình duyệt.
* Truy cập qua /swagger-ui/.

**PHẦN 4**

### **1. Spring JDBC là gì? Khác gì so với JDBC? JdbcTemplate là gì? RowMapper là gì?**

* **Spring JDBC**: Là module giúp đơn giản hóa thao tác với cơ sở dữ liệu trong Spring.
* **Khác JDBC**:
  + JDBC yêu cầu viết nhiều code để quản lý kết nối, thực thi query.
  + Spring JDBC tự động quản lý connection, xử lý exceptions.
* **JdbcTemplate**: Lớp trong Spring hỗ trợ thao tác JDBC dễ dàng hơn.
* **RowMapper**: Interface giúp ánh xạ dữ liệu từ ResultSet thành đối tượng Java.

### **2. JPA là gì? Hibernate là gì? Cách định nghĩa Entity trong JPA? Entity Manager? Persistence Context?**

* **JPA (Java Persistence API)**: Chuẩn ORM trong Java giúp ánh xạ CSDL thành đối tượng.
* **Hibernate**: Là một ORM framework phổ biến dựa trên JPA.

**Định nghĩa Entity trong JPA**:  
@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

* **Entity Manager**: Quản lý lifecycle của entity trong JPA.
* **Persistence Context**: Lưu trữ các entity đang được quản lý bởi Entity Manager.

### **3. Spring Data là gì? Tại sao cần nó? Spring Data JPA là gì?**

* **Spring Data**: Là module giúp thao tác với cơ sở dữ liệu dễ dàng hơn.
* **Lý do cần Spring Data**:
  + Giảm boilerplate code khi làm việc với CSDL.
  + Hỗ trợ repository patterns.
* **Spring Data JPA**: Là một phần của Spring Data giúp thao tác với JPA đơn giản hơn.

### **4. CrudRepository, PagingAndSortingRepository là gì?**

* **CrudRepository**: Interface hỗ trợ các thao tác CRUD.
* **PagingAndSortingRepository**: Mở rộng CrudRepository, hỗ trợ phân trang và sắp xếp.

### **5. JdbcTemplate là gì?**

* **JdbcTemplate** giúp thao tác với JDBC thuận tiện hơn.
* **Một số phương thức quan trọng:**
  + **Query đơn giản**: queryForObject, queryForList.
  + **Truy vấn danh sách**: queryForList với Map.
  + **SQLRowSet**: Kết quả không cần ánh xạ.
  + **Batch Operations**: Thực hiện nhiều câu lệnh một lúc.
  + **NamedParameterJdbcTemplate**: Hỗ trợ tham số có tên.

### **6. Java Persistence API (JPA) là gì?**

* Là API chuẩn trong Java để làm việc với cơ sở dữ liệu thông qua ORM.

### **7. Object-Relational Mapping (ORM) là gì?**

* Kỹ thuật ánh xạ giữa bảng CSDL với các lớp Java.

### **8. Lợi ích của JPA? Một số ORM frameworks phổ biến?**

* **Lợi ích:**
  + Giảm code SQL.
  + Dễ bảo trì.
  + Hỗ trợ transaction.
* **Một số ORM phổ biến:**
  + Hibernate
  + EclipseLink
  + OpenJPA

### **9. JPQL là gì? Tính năng của JPQL?**

* **JPQL (Java Persistence Query Language)**: Ngôn ngữ truy vấn của JPA, tương tự SQL nhưng hoạt động trên entity.
* **Tính năng:**
  + Hỗ trợ truy vấn động.
  + Truy vấn đối tượng thay vì bảng.

### **10. Lifecycle của một object trong JPA?**

* **New (Transient)** → **Managed (Persistent)** → **Detached** → **Removed**.

### **11. Các kiểu Identifier Generation?**

* AUTO
* IDENTITY
* SEQUENCE
* TABLE

### **12. Thuộc tính của Entity?**

* Entity có các thuộc tính ánh xạ với cột trong bảng CSDL.

### **13. Entity Manager trong JPA đóng vai trò gì?**

* Quản lý lifecycle của entity.

### **14. Ràng buộc trong Entity Class là gì?**

* Ví dụ: @NotNull, @Size, @Column(unique = true).

### **15. Các loại collections trong JPA?**

* List, Set, Map.

### **16. Kiểu Cascade trong JPA?**

* PERSIST, MERGE, REMOVE, REFRESH, ALL.

### **17. Criteria API là gì?**

* API cho phép tạo truy vấn động mà không cần viết JPQL.