Câu 1: Trình bày hiểu biết về Spring

Giới thiệu qua về Spring:

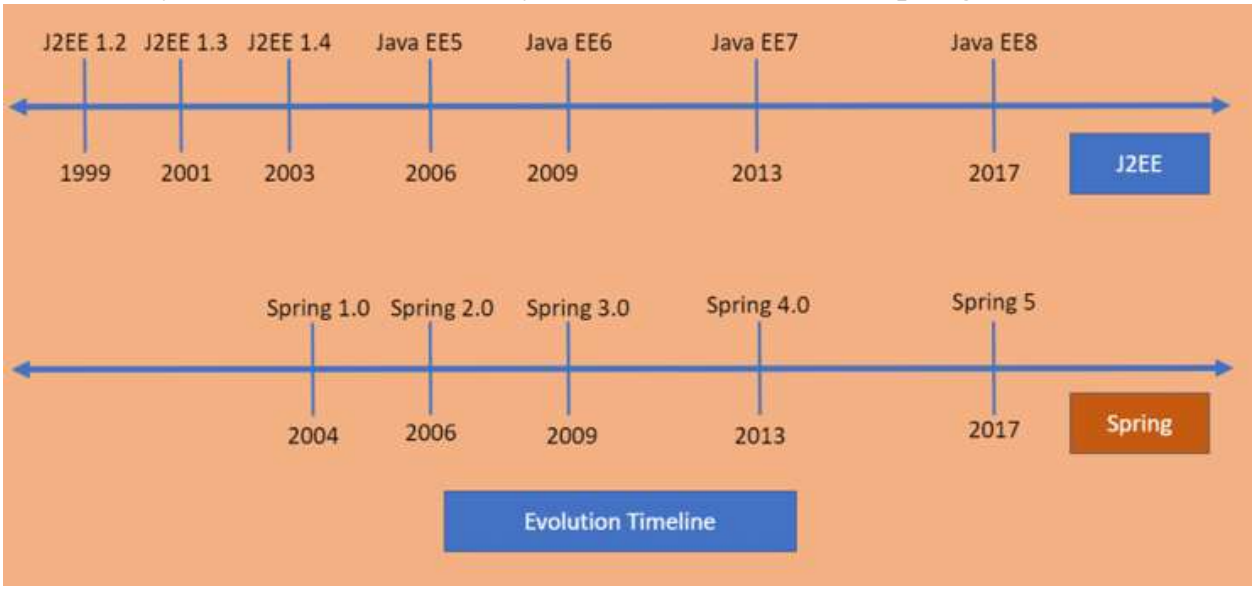
- Spring Framework là một framework mã nguồn mở được viết bằng Java, cung cấp một giải pháp toàn diện cho việc phát triển ứng dụng Java. Spring Framework được thiết kế để giảm nhẹ công việc kỹ thuật cho lập trình viên Java, giúp họ tập trung sâu vào các công việc nghiệp vụ của ứng dụng.
- Spring Framework được xây dựng dựa trên hai nguyên tắc chính:
  + Dependency Injection (DI): DI là một kỹ thuật trong đó các đối tượng phụ thuộc được truyền vào một đối tượng thay vì đối tượng đó tạo ra các đối tượng phụ thuộc của mình. DI giúp cho việc phát triển ứng dụng trở nên linh hoạt và dễ dàng bảo trì hơn.
  + Aspect Oriented Programming (AOP): AOP là một kỹ thuật trong đó các chức năng được chia thành các khía cạnh (aspect) và được áp dụng cho các đối tượng (object) mà không cần thay đổi mã của các đối tượng đó. AOP giúp cho việc phát triển ứng dụng trở nên dễ dàng hơn và giảm thiểu sự lặp lại mã.
- Những tính năng cốt lõi của Spring Framework:
  + Spring Core: Cung cấp các tính năng cơ bản của Spring Framework, bao gồm:
    - DI
    - AOP
    - Bean Factory
    - IoC Container
  + Spring MVC: Cung cấp một framework phát triển ứng dụng web dựa trên mô hình MVC.
  + Spring Data: Cung cấp một framework truy cập dữ liệu cho các cơ sở dữ liệu khác nhau.
  + Spring Boot: Là một framework giúp việc triển khai ứng dụng Spring trở nên dễ dàng hơn.
- Mục tiêu của Spring Framework: Spring Framework được thiết kế với mục tiêu giúp phát triển các ứng dụng Java J2EE một cách dễ dàng hơn dựa trên mô hình sử dụng POJO (Plain Old Java Object).
- Lợi ích của việc sử dụng Spring Framework:
  + Giảm nhẹ công việc kỹ thuật cho lập trình viên Java: Spring Framework giúp lập trình viên Java tập trung vào các công việc nghiệp vụ của ứng

dụng, thay vì phải lo lắng về các công việc kỹ thuật như cấu hình, triển khai, bảo trì,...

+ Tăng cường khả năng tái sử dụng code: Spring Framework cung cấp nhiều thành phần và tính năng có thể được tái sử dụng trong các ứng dụng khác nhau.
+ Tăng cường khả năng bảo trì và mở rộng ứng dụng: Spring Framework giúp ứng dụng trở nên linh hoạt và dễ dàng bảo trì, mở rộng hơn.

Lịch sử phát triển của Spring:

- Rod Johnson, người sáng lập Spring, bắt đầu cuộc hành trình này để tạo ra một sự thay thế cho J2EE. Điều này dẫn đến sự ra đời của Spring Framework.



- Theo dòng thời gian trên, Spring bắt đầu vào năm 2004. Ngay trước đó, 2001-2003 là khoảng thời gian mà mọi người đều chán ngấy với EJB. Sự ra mắt của Spring trùng với giai đoạn tồi tệ nhất trong việc thực hiện EJB. Nó thực sự chiếm lĩnh cộng đồng nhà phát triển và mọi người bắt đầu chuyển sang Spring.
- Tuy nhiên, những người tạo ra Java và J2EE tại Sun microsystems đã lưu ý đến sự trỗi dậy của Spring. Họ đã cố gắng để hiểu lý do đằng sau sự phổ biến ngày càng giảm của J2EE. Và họ cũng nhận ra rằng việc thực hiện EJB là vấn đề chính.
- Vì vậy, họ thực sự đã quay trở lại và tái thiết kế EJB. Nói cách khác, họ làm cho nó đơn giản hơn để sử dụng. Điều này trùng với phiên bản Java EE5 vào năm 2006. Kể từ đó, J2EE đã dần cải thiện với mọi phiên bản mới.

- Thực tế mà nói, tại thời điểm này, cả Spring và J2EE đều khá ngang nhau về các tính năng. Về cơ bản bạn có thể làm điều tương tự với cả hai framework. Tuy nhiên J2EE hơi chậm để giải quyết các vấn đề và những vấn đề trước đó với EJB đã làm mờ hình ảnh của J2EE mãi mãi. Trong thời gian này, Spring đã đi từ sức mạnh này sang sức mạnh khác. Nó đã thu thập được một lượng lớn động lực trong ngành công nghiệp và cộng đồng nhà phát triển.

Lý do Spring framework được sử dụng rộng rãi:

- Cấu hình Spring Framework dễ dàng, nhanh chóng:
  + Một trong những khía cạnh quan trọng trong sự phổ biến của bất kỳ frameworknào là việc nhà phát triển sử dụng nó dễ dàng hay không. Spring Framework thông qua nhiều quy ước về cấu hình giúp lập trình viên dễ dàng bắt đầu và sau đó thay đổi cấu hình theo chính xác những gì họ cần.
  + Những dự án như Spring Boot đã làm cho việc khởi động một dự án Spring phức tạp gần như trở nên dễ dàng. Chưa kể, nó có tài liệu và hướng dẫn tuyệt vời để giúp mọi người tham gia.
- Java Spring Framework phù hợp với mọi lập trình viên - lập trình tăng năng suất:
  + Spring Framework được biết đến là khung phần mềm của ngôn ngữ Java. Nói đến Java, hầu hết lập trình viên đều đã từng một lần sử dụng nó. Java là một ngôn ngữ phổ biến và lâu đời, được thiết kế theo phong cách hướng đối tượng - OOP. Java cũng là một môn học tại nhiều trường đại học về công nghệ. Vậy nên có thể cho rằng, Java là một ngôn ngữ có số lượng lập trình viên nhiều nhất. Đó là lý do vì sao doanh nghiệp sản xuất không gặp trở ngại về việc thiếu nhân lực phát triển trong dự án sử dụng Java.
- Spring Framework sở hữu tính chất Module linh hoạt
  + Một khía cạnh quan trọng khác trong sự phổ biến của Spring là tính chất mô-đun cao của nó. Spring Framework cho phép lựa chọn sử dụng toàn bộ hệ sinh thái của Spring, hoặc chỉ sử dụng mô-đun cần thiết. Hơn nữa, lập trình viên có thể tùy chọn đưa vào một hoặc nhiều dự án Spring tùy theo nhu cầu.
- Khả năng Kiểm thử mà bất kỳ framework nào cũng cần
  + Việc lựa chọn framework phần lớn phụ thuộc vào thực tế là việc kiểm thử phần mềm có diễn ra dễ dàng hay không.
  + Với Spring, nó đáp ứng nhu cầu này bằng việc hỗ trợ Phát triển ứng dụng theo hướng thử nghiệm - Test Driven Development (TDD).

    + Ứng dụng Spring chủ yếu bao gồm các POJO, điều này đương nhiên làm cho việc kiểm thử đơn vị (Unit Testing) trở nên đơn giản hơn nhiều. Tuy nhiên, Spring cung cấp Mock Objects cho các tình huống như MVC, nơi mà việc kiểm thử đơn vị trở nên phức tạp hơn.
- Sự trưởng thành của Spring
    + Spring Framework có một lịch sử lâu dài về đổi mới, áp dụng và tiêu chuẩn hóa.Qua nhiều năm, nó đã đủ trưởng thành để trở thành giải pháp mặc định cho hầu hết các vấn đề phổ biến gặp phải trong quá trình phát triển các ứng dụng doanh nghiệp quy mô từ nhỏ đến lớn.
    + Điều thú vị hơn nữa là Spring đang được phát triển và duy trì tích cực. Hỗ trợ cho các tính năng ngôn ngữ mới và các giải pháp tích hợp doanh nghiệp đang được phát triển mỗi ngày.
- Sự đóng góp của cộng đồng Spring
    + Cuối cùng nhưng không kém phần quan trọng, bất kỳ framework nào hoặc ngôn ngữ lập trình nào cũng tồn tại trong ngành thông qua sự đổi mới và không có nơi nào tốt hơn cho sự đổi mới ngoài cộng đồng.
    + Spring là một mã nguồn mở do Pivotal Software dẫn đầu và được hỗ trợ bởi một nhóm lớn những tổ chức và nhà phát triển cá nhân. Điều này có nghĩa là nó vẫn tồn tại theo ngữ cảnh và thường mang tính tương lai, thể hiện rõ qua số lượng dự án dưới sự hỗ trợ của nó ngày một tăng.

Các thành phần chính của Spring framework:

1. Dependency Injection (DI) và Inversion of Control (IoC):

a, IoC:

- Nguyên tắc Inversion of Control (IoC) là một nguyên tắc trong kỹ thuật phần mềm, nó chuyển quyền kiểm soát của các đối tượng hoặc một phần của chương trình đến một container hoặc framework. Thường thường, chúng ta sử dụng nó trong ngữ cảnh của lập trình hướng đối tượng.
- Khác với lập trình truyền thống, trong đó mã tùy chỉnh của chúng ta gọi đến một thư viện, IoC cho phép một framework tiếp quản luồng của một chương trình và gọi đến mã tùy chỉnh của chúng ta. Để thực hiện điều này, các framework sử dụng các trừu tượng với hành vi bổ sung được tích hợp sẵn. Nếu chúng ta muốn thêm hành vi của riêng mình, chúng ta cần mở rộng các lớp của framework hoặc gắn thêm các lớp của riêng mình.
- Các lợi ích của kiến trúc này bao gồm:

1. Tách biệt việc thực thi một nhiệm vụ với việc triển khai nó.
2. Dễ dàng chuyển đổi giữa các triển khai khác nhau.
3. Tính mô đun cao hơn của chương trình.
4. Dễ dàng kiểm tra chương trình bằng cách cô lập một thành phần hoặc mô phỏng các phụ thuộc của nó và cho phép các thành phần giao tiếp thông qua các hợp đồng.

- Chúng ta có thể đạt được Inversion of Control thông qua các cơ chế khác nhau như: mẫu thiết kế Strategy, mẫu Service Locator, mẫu Factory, và Dependency Injection(DI).

b, DI

- Dependency injection là một mẫu thiết kế mà chúng ta có thể sử dụng để thực hiện Inversion of Control (IoC), trong đó quyền kiểm soát đang được đảo ngược là việc thiết lập các phụ thuộc của một đối tượng.
- Kết nối đối tượng với các đối tượng khác, hoặc "inject" đối tượng vào các đối tượng khác, được thực hiện bởi một bộ lắp (assembler) thay vì bởi các đối tượng chính chúng.
- Dưới đây là cách chúng ta sẽ tạo một phụ thuộc đối tượng trong lập trình truyền thống:

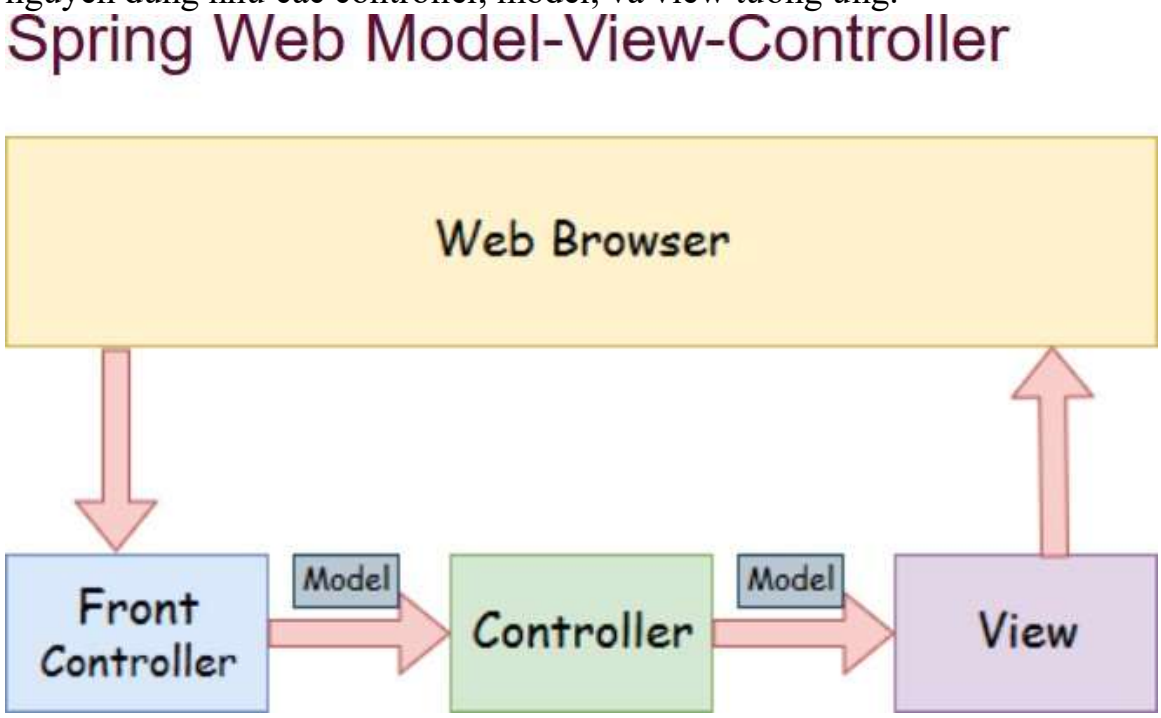
```
public class Store {
    private Item item;

    public Store() {
        item = new ItemImpl1();
    }
}
```

Trong ví dụ ở trên, chúng ta cần phải khởi tạo một phiên bản của giao diện Item bên trong lớp Store.
Bằng cách sử dụng Dependency Injection (DI), chúng ta có thể viết lại ví dụ mà không cần xác định cụ thể việc triển khai của giao diện Item mà chúng ta muốn:

```
public class Store {
    private Item item;
    public Store(Item item) {
        this.item = item;
    }
}
```

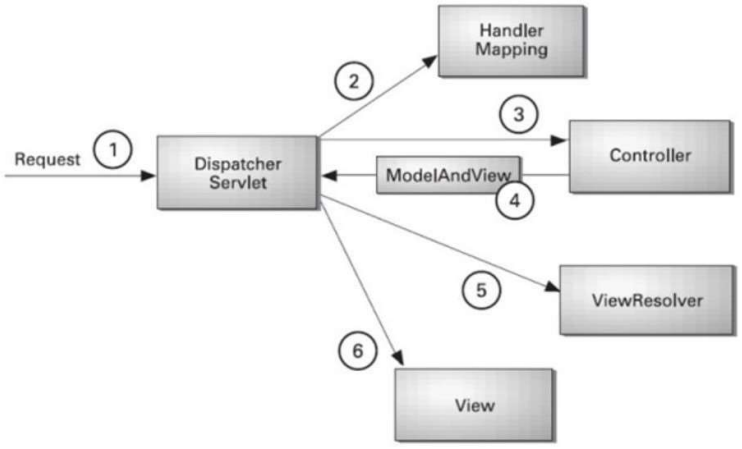
2. Spring MVC:

- Spring MVC là một framework Java được sử dụng để xây dựng các ứng dụng web. Nó tuân theo mô hình thiết kế Model-View-Controller (MVC). Spring MVC thực hiện tất cả các tính năng cơ bản của Spring Framework như Inversion of Control (IoC) và Dependency Injection (DI).
- Spring MVC cung cấp một giải pháp thanh lịch để sử dụng mô hình MVC trong Spring Framework thông qua sự hỗ trợ của DispatcherServlet. Ở đây, DispatcherServlet là một lớp nhận các yêu cầu đến và ánh xạ chúng đến tài nguyên đúng như các controller, model, và view tương ứng.



+ Model - Một model chứa dữ liệu của ứng dụng. Dữ liệu có thể là một đối tượng đơn lẻ hoặc một tập hợp các đối tượng.
+ Controller - Một controller chứa logic kinh doanh của ứng dụng. Ở đây, chúng ta sử dụng chú thích @Controller để đánh dấu lớp là một controller.
+ View - Một view đại diện cho thông tin được cung cấp dưới một định dạng cụ thể. Thông thường, chúng ta sử dụng JSP+JSTL để tạo trang view. Tuy nhiên, Spring cũng hỗ trợ các công nghệ view khác như Apache Velocity, Thymeleaf và FreeMarker.
+ Front Controller - Trong Spring Web MVC, lớp DispatcherServlet hoạt động như một front controller. Nó chịu trách nhiệm quản lý luồng của ứng dụng Spring MVC.

- Luồng xử lý của spring web MVC:



+ Như được hiển thị trong hình vẽ, tất cả các yêu cầu đến đều được Intercepted (chặn) bởi DispatcherServlet, làm nhiệm vụ của front controller.
+ DispatcherServlet lấy thông tin từ tệp XML về việc ánh xạ (handler mapping) và chuyển tiếp yêu cầu đến controller (bộ điều khiển).
+ Controller trả về một đối tượng ModelAndView (một đối tượng chứa thông tin về dữ liệu và view).
+ DispatcherServlet kiểm tra thông tin về xử lý view được cấu hình trong tệp XML và gọi thành phần view được chỉ định để hiển thị kết quả.

– Ưu điểm của Spring MVC:

+ Tách biệt các vai trò - Spring MVC tách biệt mỗi vai trò, trong đó đối tượng model, controller, đối tượng command, trình giải quyết view, DispatcherServlet, trình kiểm tra hợp lệ (validator), và nhiều vai trò khác có thể được thực hiện bởi đối tượng chuyên biệt.
+ Nhẹ và nhẹ - Nó sử dụng một máy chủ servlet nhẹ để phát triển và triển khai ứng dụng của bạn.
+ Cấu hình mạnh mẽ - Nó cung cấp cấu hình mạnh mẽ cho cả framework và các lớp ứng dụng, bao gồm việc tham chiếu dễ dàng giữa các ngữ cảnh, chẳng hạn từ các web controller đến các đối tượng kinh doanh và trình kiểm tra hợp lệ.

+ Phát triển nhanh - Spring MVC giúp việc phát triển nhanh chóng và song song.
+ Mã kinh doanh có thể tái sử dụng - Thay vì tạo ra các đối tượng mới, nó cho phép chúng ta sử dụng lại các đối tượng kinh doanh hiện có.
+ Dễ kiểm tra - Trong Spring, chúng ta thường tạo các lớp JavaBeans cho phép chèn dữ liệu kiểm tra bằng cách sử dụng các phương thức setter.
+ Phân định linh hoạt - Nó cung cấp các chú thích cụ thể giúp dễ dàng chuyển hướng trang.

3. Spring Data JPA:

- Spring Data là một module của Spring Framework. Mục đích của Spring

Data JPA là giảm thiểu việc thực hiện quá nhiều bước để có thể implement được JPA. Spring Data JPA là một phần của Spirng Data và nó hỗ trợ Hibernate 5, OpenJPA 2.4 và EclipseLink 2.6.1

- Spring Data Generated DAO – No More DAO Implementations:

Như chúng ta đã biết thì class DAO thường bao gồm rất nhiều code được viết sẵn và cần phải đơn giản hóa nó đi. Những lợi thế của việc đơn giản hóa như vậy rất nhiều: giảm số lượng tương tác mà chúng ta cần xác định và duy trì, tính nhất quán của mẫu truy cập dữ liệu và cấu hình.

Spring Data đưa sự đơn giản hóa này lên một bước nữa và có thể loại bỏ hoàn toàn việc triển khai DAO. Interface DAO hiện là các thao tác duy nhất mà chúng ta cần define rõ ràng.

Để bắt đầu tận dụng Spring Data programming model với JPA, một interface DAO cần extends JPA specific Repository interface, JpaRepository. Điều này cho phép Spring Data tìm thấy interface này và tự động tạo 1 implementation cho nó.

Bằng cách extends interface, chúng ta nhận được các method CRUD phù hợp nhất để truy cập dữ liệu tiêu chuẩn có sẵn trong một DAO tiêu chuẩn.

- Custom Access Method và Queries:

Như đã thảo luận, bằng cách implement một trong các Repository interfaces, DAO sẽ có một số method CRUD cơ bản (và truy vấn) được define và implement.

Để define các method truy cập cụ thể hơn, Spring JPA hỗ trợ khá nhiều tùy chọn:

+ chỉ cần define một method mới trong interface.

+ cung cấp query JPQL bằng cách sử dụng annotation @Query

+ sử dụng Specification nâng cao và Querydsl hỗ trợ trong Spring Data

+ define custom queries thông qua JPA Named Queries

- Transaction Configuration:

Spring và Spring Data hỗ trợ việc quản lý transaction khiến cho việc này cực kỳ đơn giản, tất cả những gì chúng ta cần làm là chú thích một class hay một method với @Transactional annotation.

Transaction quản lý những thay đổi mà bạn thực hiện trong một hoặc nhiều hệ thống. Nó có thể database, message brokers, hoặc bất kỳ loại hệ thống phần mềm nào khác.

Mục tiêu chính của giao dịch là cung cấp các đặc điểm ACID để đảm bảo tính nhất quán và hợp lệ của dữ liệu của bạn.

- Spring Data JPA Repository Configuration

Để kích hoạt hỗ trợ Spring JPA repository, chúng ta có thể sử dụng annotation @EnableJpaRepositories.

```
@EnableJpaRepositories(basePackages = "com.hdd.repository")
public class PersistenceConfig {
    ...
}
```

Chúng ta có thể làm tương tự với cấu hình XML:

```
<jpa:repositories base-package="com.hdd.repository"/>
```

- Java hoặc XML Configuration

Chúng ta đã thảo luận rất chi tiết về cách cấu hình JPA trong Spring trong một bài viết trước. Spring Data cũng tận dụng sự hỗ trợ của Spring cho annotation JPA @PersistenceContext. Nó sử dụng điều này để chuyển EntityManager

vào Spring factory bean. JpaRepositoryFactoryBean sẽ chịu trách nhiệm tạo các implement DAO.

Ngoài cấu hình đã được thảo luận, chúng ta cần include Spring Data XML Config nếu sử dụng XML:

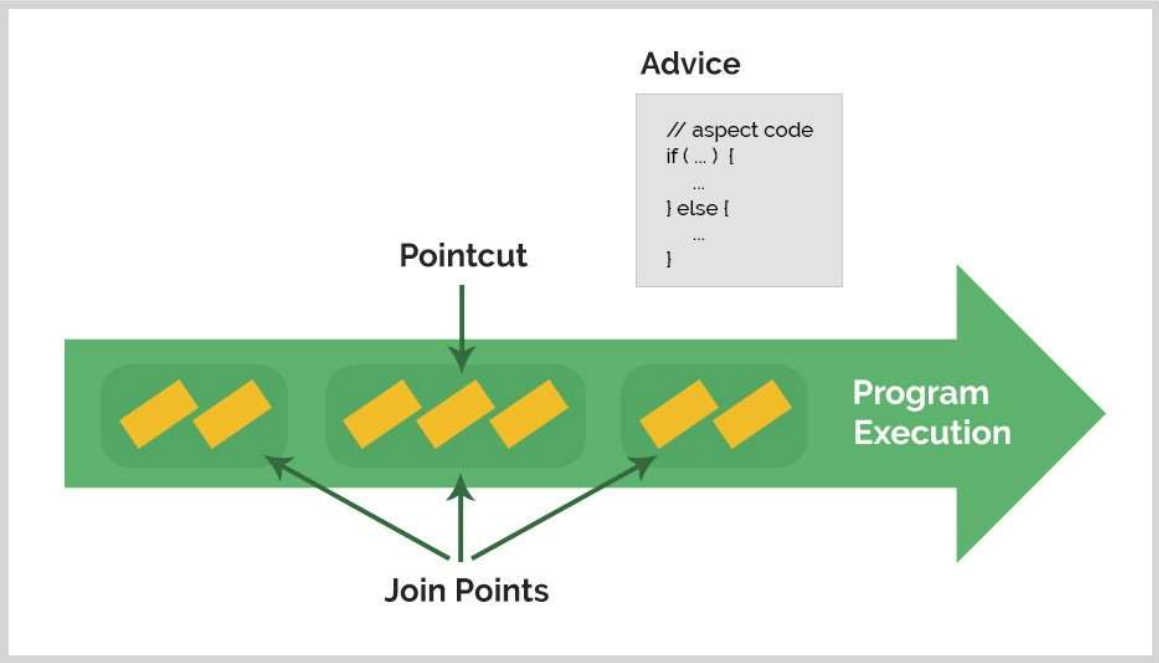
```
@Configuration
@EnableTransactionManagement
@ImportResource("classpath*:*springDataConfig.xml")
public class PersistenceJPAConfig {
    ...
}
```

Các tính năng nổi bật của Spring Framework:

1. Aspect-Oriented Programming (AOP):

- AOP (Aspect-Oriented Programming) là một mô hình lập trình mục tiêu tăng tính module bằng cách cho phép tách biệt các vấn đề xuyên tuyến (cross-cutting concerns). Nó làm điều này bằng cách thêm hành vi bổ sung vào mã nguồn hiện có mà không cần sửa đổi mã nguồn đó. Thay vì vậy, chúng ta có thể khai báo mã mới và các hành vi mới một cách độc lập. Framework AOP của Spring giúp chúng ta thực hiện các vấn đề xuyên tuyến này.

- AOP Concepts and Terminology:



2. Spring Boot

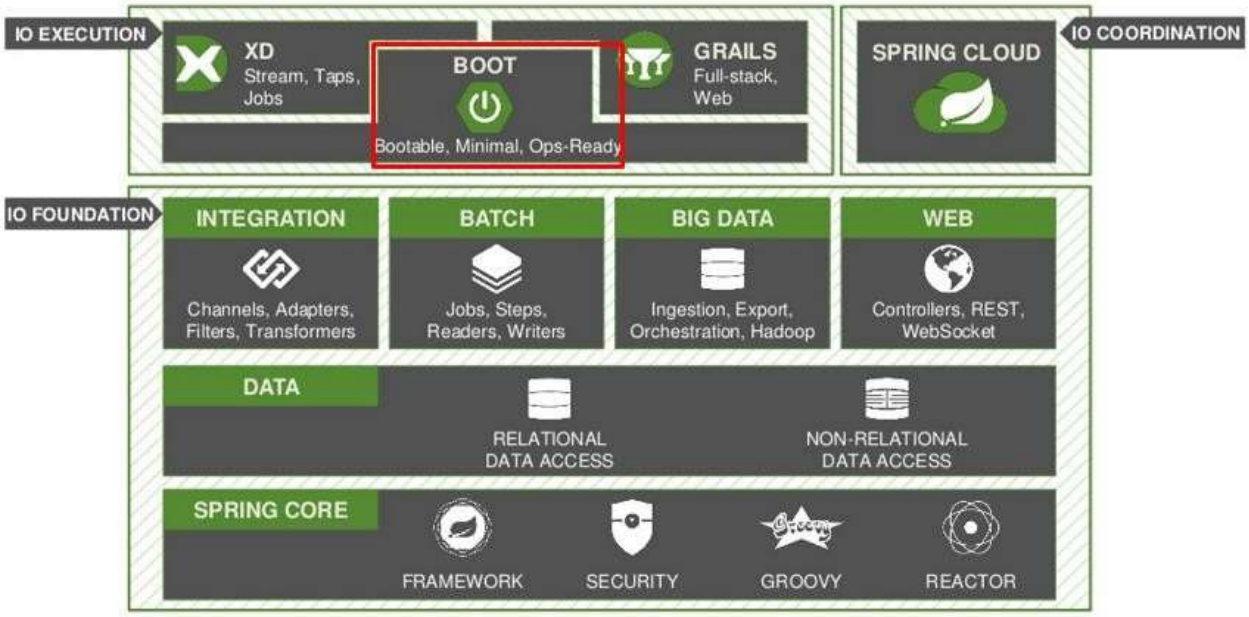
- Spring Boot là gì?

+ Spring Boot là một module của Spring Framework, cung cấp tính năng RAD (Rapid Application Development) – Phát triển ứng dụng nhanh.

+ Spring Boot được dùng để tạo các ứng dụng độc lập dựa trên Spring.

+ Spring Boot không yêu cầu cấu hình XML

+ Nó là một chuẩn cho cấu hình thiết kế phần mềm, tăng cao năng suất cho developer.



- Ưu điểm của Spring Boot.

  + Có các tính năng của Spring Framework.

  + Tạo ứng dụng độc lập, có thể chạy bằng java -jar (cho cả java web)

  + Nhúng trực tiếp các ứng dụng server (Tomcat, Jetty…) do đó không cần phải triển khai file WAR

  + Cấu hình ít, tự động cậu hình bất kì khi nào có thể (Giảm thời gian viết code, tăng năng suất)

  + Không yêu cầu XML config…

  + Cung cấp nhiều plugin

  + Chuẩn cho Microservices (Cloud support; giảm việc setup, config; các thư viện hỗ trợ…)

- Các features (tính năng, đặc điểm) của Spring Boot:

  + SpringApplication: SpringApplication là một class cung cấp cách thuận tiện để khởi chạy ứng dụng từ hàm main(). Để start ứng dụng, chỉ cần gọi method run().

  Ví dụ:

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

+ Externalized Configuration:

Spring Boot cho phép chúng ta cấu hình từ bên ngoài (externalize), do đó một ứng dụng có thể chạy trên nhiều môi trường khác nhau.

Chúng ta có thể sử dụng file YAML, file properties, các biến môi trường và tham số command-line để thực hiện externalize configuration.

Các thuộc tính cấu hình có thể inject trực tiếp vào bean bằng cách sử dụng annotation @Value hoặc thông qua object với @ConfigurationProperites…

Ví dụ:

```
import org.springframework.stereotype.*
import org.springframework.beans.factory.annotation.*

@Component
public class MyBean {

    @Value("${name}")
    private String name;

    // ...

}
```

+ Profiles: Spring Boot Profiles cung cấp một cách để phân chia các cấu hình cho mỗi môi trường. Các annotation @Component hoặc@Configuration có thể được đánh dấu @Profile để giới hạn khi nào được tải lên.

Ví dụ chỉ tải trong môi trường product…

```
@Configuration
@Profile("production")
public class ProductionConfiguration {

  // ...

}
```

- Logging: Spring Boot sử dụng common logging cho tất cả các chức năng log nội bộ.

Các depencdeny logging được quản lý mặc định, chúng ta không nên / cần sửa dependency logging nếu không có yêu cầu tùy biến (customization) thực sự cần.

Ngoài ra còn rất nhiều các tính năng khác:

- Developing Web Applications
- Security
- Working with SQL Databases: Spring Boot + Hibernate ...
- Working with NoSQL Technologies: Redis, MongoDB, Neo4j, LDAP, Solr ...
- Caching: EhCache, JCache …
- Messaging
- Calling REST Services with RestTemplate
- Calling REST Services with WebClient
- Validation
- Sending Email
- Distributed Transactions with JTA
- Hazelcast
- Quartz Scheduler
- Spring Integration
- Spring Session
- Monitoring and Management over JMX
- Testing
- WebSockets
- Web Services
- Creating Your Own Auto-configuration

Câu 2: Kiến trúc MVC

Khái niệm MVC:

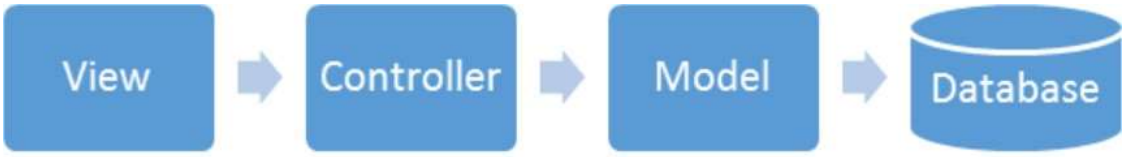
- MVC là một kiến trúc phân tách logic kinh doanh, giao diện và dữ liệu. Trong MVC:

+ M là Model (Mô hình)

+ V là View (Giao diện)

+ C là Controller (Bộ điều khiển)

MVC là một cách hệ thống để sử dụng ứng dụng, nơi luồng làm việc bắt đầu từ tầng giao diện (View), nơi yêu cầu được gửi và xử lý trong tầng điều khiển (Controller) và sau đó được gửi đến tầng mô hình (Model) để chèn dữ liệu và nhận lại thông báo về thành công hoặc thất bại. Biểu đồ kiến trúc MVC được mô tả như sau:



MVC Architecture Diagram

- Tầng Model:

+ Đây là tầng dữ liệu chứa logic kinh doanh của hệ thống.

+ Bao gồm tất cả dữ liệu của ứng dụng.

+ Đại diện cho trạng thái của ứng dụng.

+ Bao gồm các lớp có kết nối với cơ sở dữ liệu.

+ Bộ điều khiển (Controller) kết nối với tầng này để truy vấn dữ liệu và gửi đến tầng giao diện (View).

+ Mô hình (Model) cũng kết nối với cơ sở dữ liệu và lưu trữ dữ liệu vào cơ sở dữ liệu mà nó kết nối.

- Tầng View:

+ Đây là tầng trình bày giao diện.

+ Bao gồm các công nghệ như HTML, JSP, v.v.

+ Thường làm giao diện người dùng (UI) của ứng dụng.

+ Dùng để hiển thị dữ liệu được truy vấn từ tầng điều khiển (Controller), mà trong trường hợp này đang lấy dữ liệu từ các lớp ở tầng Model.

+ Tầng View này hiển thị dữ liệu trên giao diện người dùng của ứng dụng.

- Tầng Controller:

  + Tầng này hoạt động như một giao diện giữa tầng View và tầng Model.

  + Intercept (bắt và xử lý) tất cả các yêu cầu đến từ tầng View.

  + Nhận các yêu cầu từ tầng View, xử lý các yêu cầu và thực hiện các kiểm tra cần thiết cho yêu cầu.

  + Yêu cầu này sau đó được gửi tiếp đến tầng Model để xử lý dữ liệu, và khi yêu cầu được xử lý, nó gửi trở lại cho tầng Controller với thông tin cần thiết và được hiển thị tương ứng bởi tầng View.

Các lợi ích của kiến trúc MVC là:

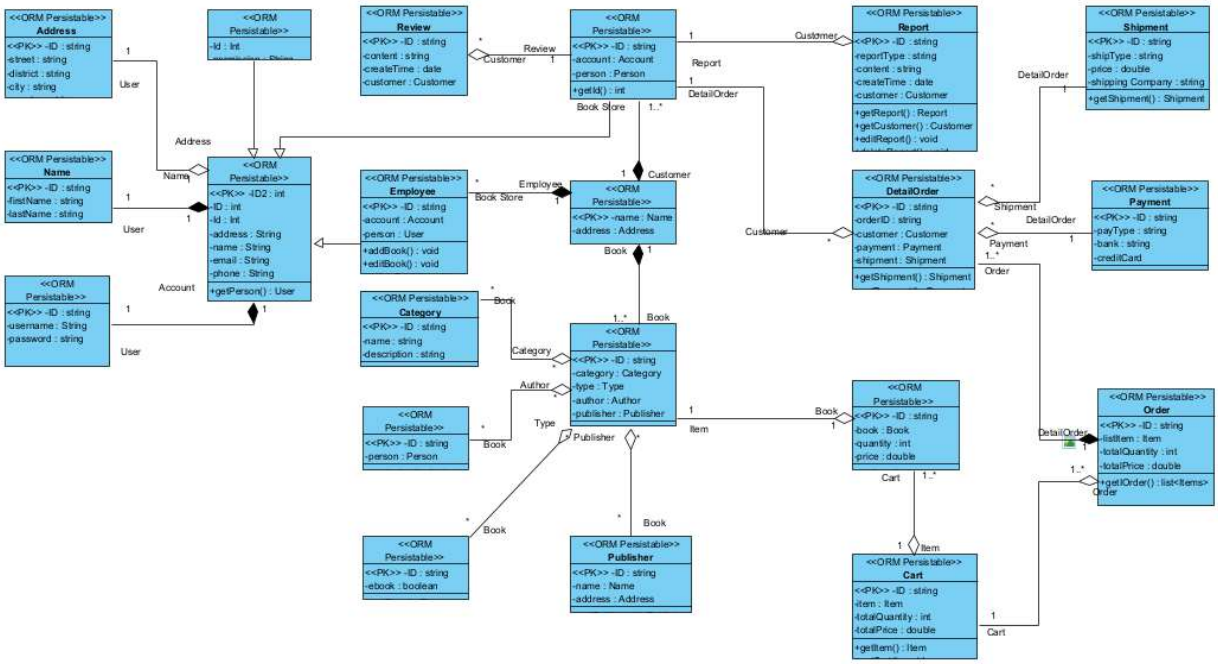
1. Dễ dàng bảo trì: Kiến trúc MVC giúp tách biệt rõ ràng giữa các tầng khác nhau, giúp làm cho mã nguồn dễ bảo trì hơn. Bạn có thể tìm và sửa lỗi trong một phần của ứng dụng mà không cần ảnh hưởng đến các phần khác.

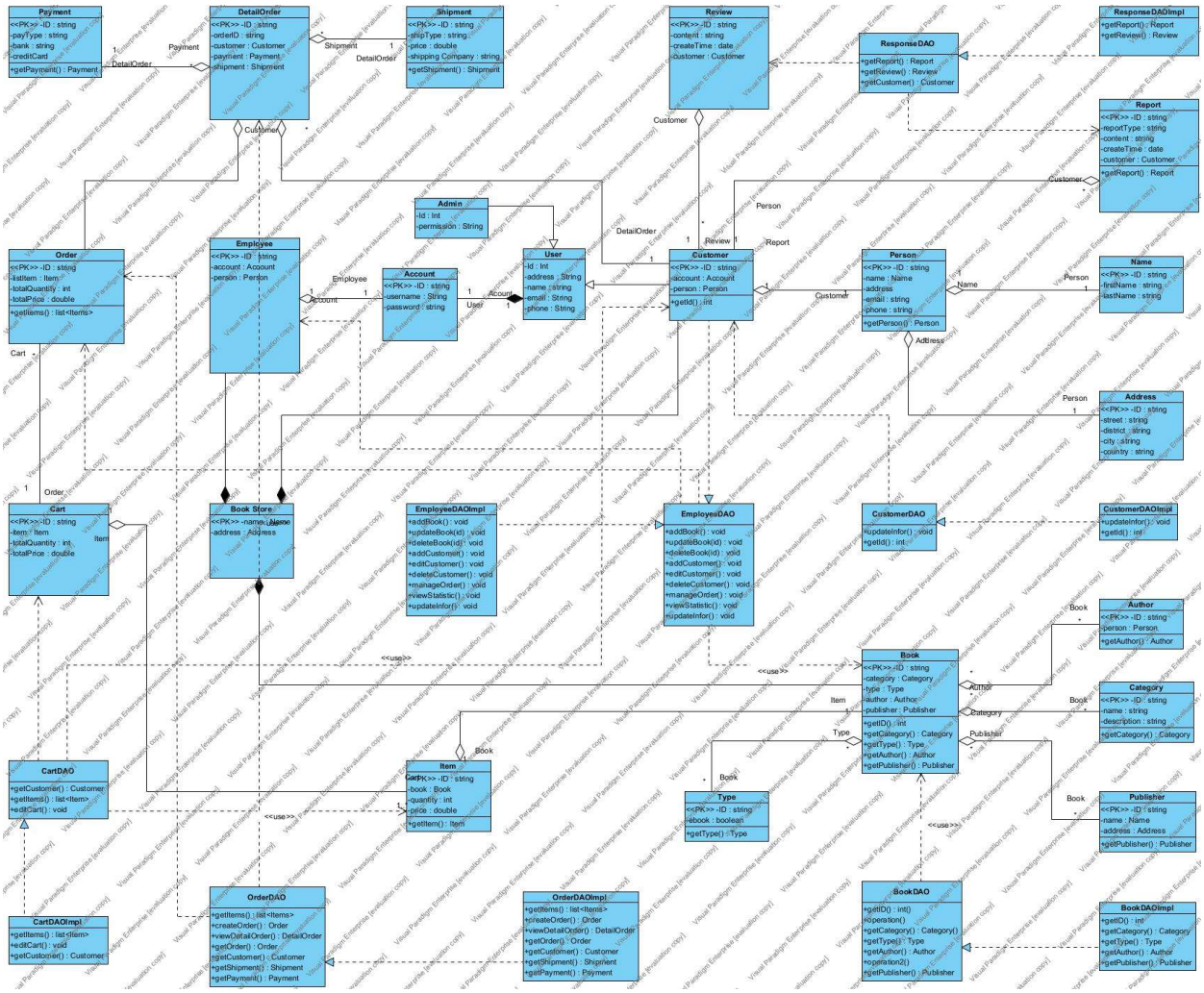
2. Dễ dàng mở rộng: Kiến trúc MVC cho phép bạn thêm mới các tính năng và chức năng vào ứng dụng một cách dễ dàng mà không cần phải thay đổi quá nhiều mã nguồn hiện có. Điều này giúp cho việc phát triển ứng dụng trở nên linh hoạt và nhanh chóng.

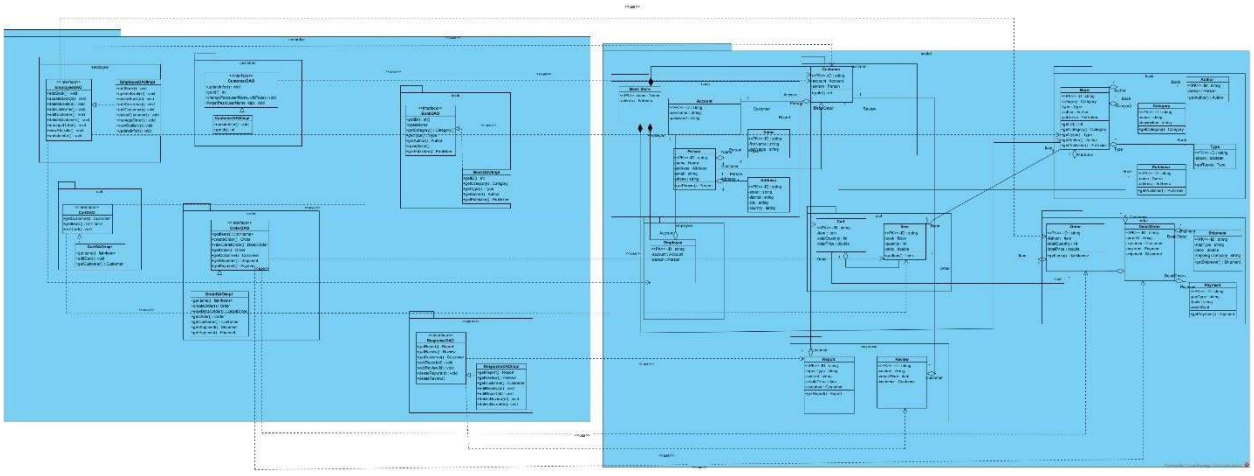
3. Dễ dàng kiểm thử: Với việc tách biệt rõ ràng giữa các tầng, bạn có thể kiểm tra từng phần riêng lẻ của ứng dụng một cách dễ dàng. Điều này giúp tạo ra các bộ kiểm thử (test cases) hiệu quả để đảm bảo tính ổn định và đúng đắn của ứng dụng.

4. Quản lý điều hướng được tập trung: Kiến trúc MVC thường tập trung quản lý điều hướng của ứng dụng trong tầng Controller. Điều này làm cho việc quản lý và theo dõi các tác động và chuyển đổi giữa các trang hoặc chức năng trở nên dễ dàng hơn.

Câu 3: Bổ sung thuộc tính


<<ORM Persistable>>
Address
<<PK>> -ID : string
-street : string
-district : string
-city : string
User
Address
Name
<<ORM Persistable>>
Name
<<PK>> -ID : string
-firstName : string
-lastName : string
User
<<ORM Persistable>>
<<PK>> -ID2 : int
-ID : int
-Id : Int
-address : String
-name : String
-email : String
-phone : String
+getPerson() : User
Account
<<ORM Persistable>>
<<PK>> -ID : string
-username : String
-password : string
User
<<ORM Persistable>>
Review
<<PK>> -ID : string
-content : string
-createTime : date
-customer : Customer
Review
Customer
<<ORM Persistable>>
<<PK>> -ID : string
-account : Account
-person : Person
+getId() : int
Book Store
Customer
Report
DetailOrder
<<ORM Persistable>>
Employee
-account : Account
-person : User
+addBook() : void
+editBook() : void
Employee
Book Store
<<ORM Persistable>>
<<PK>> -name : Name
-address : Address
Customer
Book
<<ORM Persistable>>
Category
<<PK>> -ID : string
-name : string
-description : string
Book
Category
<<ORM Persistable>>
<<PK>> -ID : string
-category : Category
-type : Type
-author : Author
-publisher : Publisher
Book
Author
Type
Publisher
Item
<<ORM Persistable>>
<<PK>> -ID : string
-person : Person
Book
Author
<<ORM Persistable>>
<<PK>> -ID : string
-ebook : boolean
Book
Type
<<ORM Persistable>>
Publisher
<<PK>> -ID : string
-name : Name
-address : Address
Book
Publisher
<<ORM Persistable>>
Report
<<PK>> -ID : string
-reportType : string
-content : string
-createTime : date
-customer : Customer
+getReport() : Report
+getCustomer() : Customer
+editReport() : void
Customer
Report
<<ORM Persistable>>
DetailOrder
<<PK>> -ID : string
-orderID : string
-customer : Customer
-payment : Payment
-shipment : Shipment
+getShipment() : Shipment
Customer
DetailOrder
<<ORM Persistable>>
Book
<<PK>> -ID : string
-book : Book
-quantity : int
-price : double
Book
Item
<<ORM Persistable>>
Cart
<<PK>> -ID : string
-item : Item
-totalQuantity : int
-totalPrice : double
+getItem() : Item
Item
Cart
<<ORM Persistable>>
Shipment
<<PK>> -ID : string
-shipType : string
-price : double
-shipping Company : string
+getShipment() : Shipment
DetailOrder
Shipment
<<ORM Persistable>>
Payment
<<PK>> -ID : string
-payType : string
-bank : string
-creditCard
DetailOrder
Payment
<<ORM Persistable>>
Order
<<PK>> -ID : string
-listItem : Item
-totalQuantity : int
-totalPrice : double
+getIOrder() : list<Items>
DetailOrder
Order
Order
Cart





Câu 4:

Model:

```java
import jakarta.persistence.Entity;
@Entity
public class Account {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
    public Account() {
        super();
    }
    public Account(int id, String username, String password) {
        super();
        this.id = id;
        this.username = username;
        this.password = password;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
```

```java
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;
    private Double price;
    private String description;
    @ManyToMany
    @JoinTable(name = "book_cate",
               joinColumns = @JoinColumn(name = "bookId"),
               inverseJoinColumns = @JoinColumn(name = "cateId"))
    private Set<Category> categories = new HashSet<Category>();
    @ManyToMany
    @JoinTable(name = "book_author",
               joinColumns = @JoinColumn(name = "bookId"),
               inverseJoinColumns = @JoinColumn(name = "authorId"))
    private Set<Author> authors = new HashSet<Author>();
    @ManyToMany
    @JoinTable(name = "book_publisher",
               joinColumns = @JoinColumn(name = "bookId"),
               inverseJoinColumns = @JoinColumn(name = "publisherId"))
    private Set<Publisher> publishers = new HashSet<Publisher>();
    public Book() {
        super();
    }


    public Book(int id, String name, Double price, String description, Set<Category> categories, Set<Author> authors,
            Set<Publisher> publishers) {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
```

```java
@Entity
public class Category {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;
    private String description;
    @ManyToMany
    @JoinTable(name = "book_cate",
              joinColumns = @JoinColumn(name = "cateId"),
              inverseJoinColumns = @JoinColumn(name = "bookId"))
    private Set<Book> books = new HashSet<Book>();
    public Category() {
        super();
    }
    public Category(int id, String name, String description, Set<Book> books) {
        super();
        this.id = id;
        this.name = name;
        this.description = description;
        this.books = books;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```java
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;
    @ManyToMany
    @JoinTable(name = "book_author",
              joinColumns = @JoinColumn(name = "authorId"),
              inverseJoinColumns = @JoinColumn(name = "bookId"))
    private Set<Book> books = new HashSet<Book>();
    public Author() {
        super();
    }
    public Author(int id, String name, Set<Book> books) {
        super();
        this.id = id;
        this.name = name;
        this.books = books;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```java
@Entity
public class Item {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "bookId", referencedColumnName = "id")
    private Book book;
    private int quantity;
    private Double totalPrice;
    public Item() {
        super();
    }
    public Item(int id, Book book, int quantity, Double totalPrice) {
        super();
        this.id = id;
        this.book = book;
        this.quantity = quantity;
        this.totalPrice = totalPrice;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public Book getBook() {
        return book;
    }
    public void setBook(Book book) {
        this.book = book;
    }
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
```

```java

@Entity
public class Publisher {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;
    private String address;
    @ManyToMany
    @JoinTable(name = "book_publisher",
               joinColumns = @JoinColumn(name = "publisherId"),
               inverseJoinColumns = @JoinColumn(name = "bookId"))
    private Set<Book> books = new HashSet<Book>();
    public Publisher() {
        super();
    }
    public Publisher(int id, String name, String address, Set<Book> books) {
        super();
        this.id = id;
        this.name = name;
        this.address = address;
        this.books = books;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
```

Controller:

```java
public interface BookDAO {
    public List<Book> getAllBook();
    public List<Category> getAllCategories();
    public List<Book> getBookByCateId(int id);
    public Book getBookById(int id);
}
```

```java
@Repository
public class BookDAOImpl implements BookDAO{
    @Autowired
    public BookRepo bRepo;
    @Autowired
    public CategoryRepo categoryRepo;
    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public List<Book> getAllBook() {

        return bRepo.findAll();
    }

    @Override
    public List<Category> getAllCategories() {
        return categoryRepo.findAll();
    }

    @Override
    public List<Book> getBookByCateId(int id) {
        String jpql = "SELECT b FROM Book b JOIN b.categories c WHERE c.id = :cateId";
        return entityManager.createQuery(jpql, Book.class)
                .setParameter("cateId", id)
                .getResultList();
    }

    @Override
    public Book getBookById(int id) {
        return bRepo.getById(id);
    }
```

```java
public interface CustomerDAO {
    public Account getAccountByUP(String username, String pass);
}
```

```java
@Repository
public class CustomerDAOImpl implements CustomerDAO{
    @Autowired
    AccountRepo accountRepo;
    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public Account getAccountByUP(String username, String pass) {
        String jpql = "SELECT a FROM Account a WHERE a.username = :username AND a.password = :pass";
        List<Account> resultList = entityManager.createQuery(jpql, Account.class)
            .setParameter("username", username)
            .setParameter("pass", pass)
            .getResultList();

        if (resultList.isEmpty()) {
            return null;
        }
        return resultList.get(0);
    }


}
```