1) 애플리케이션 아키텍처에서 애플리케이션 서버 티어를 구성하는 3가지 레이어에 대해 설명하고 결합도를 줄이기 위한 방법을 설명하시오.

1) – ①: 애플리케이션 아키텍처에서 애플리케이션 서버 티어를 구성하는 3가지 레이어

웹 애플리케이션의 아키텍처는 크게 티어라고 하는 물리층과 레이어라고 하는 논리층으로 나뉜다.

티어는 클라이언트 층, 중간층(애플리케이션 서버), EIS(Enterprise Information System)층이 기본이다. 기본적으로 웹 어플리케이션에서 고려할 것은 중간층이다.

레이어는 클라이언트 층에도 일부 있지만, 기본적으로는 중간층에 있는 웹 애플리케이션을 논리적으로 분류한 것이다. 레이어를 약한 결합으로 유지해서 변경이나 기능 추가에 강한 웹 애플리케이션을 만든다. 레이어는 원래 아키텍처 패턴 중 하나이며, 서로 인접한 레이어끼리 만 단방향 액세스를 할 수 있다. 일반적인 레이어는 다음 3개 층으로 나누고 각각 다른 역할을 부여한다.

- 프레젠테이션 층(Presentation)

프레젠테이션 층의 주된 역할은 사용자 인터페이스와 컨트롤러를 제공하는 것이다. 사용자 인터페이스란 사용자가 직접 조작하는 화면이나 장표를 말한다. 컨트롤러는 사용자 인터페이스를 통해 사용자의 입력을 받아 적절한 비즈니스 로직을 호출하고, 그 결과를 사용자 인터페이스로 반환하는 작업을 한다. 컨트롤러의 또 한가지 중요한 작업은 웹 애플리케이션의 상태(세션)를 저장해 이용하는 데이터를 관리하는 것이다. 컨트롤러는 일반적으로 MVC2라고 불리는 JSP 모델의 컨트롤러로 알려져 있다.

- 비즈니스 로직 층(Business Logic

비즈니스 로직 층은 서비스나 도메인 같은 비즈니스 로직을 구현하는 웹 애플리케이션의 중심이다. 유스 케이스로 표현되는 특정 업무나 특정 부서 처리의 통합인 서비스 및 도메인으로 구성된다. 도메인은 서비스에서 시작되는 비즈니스 실행에서 필요한 고객이나 주문 등의 처리를 구현하는 클래스의 집합이다. 이 서비스와 도메인은 각각 비즈니스 층에 만들어진 서비스 패키지의 클래스와 도메인 패키지의 클래스로 구현한다.

- 데이터 액세스 층(Data Access)

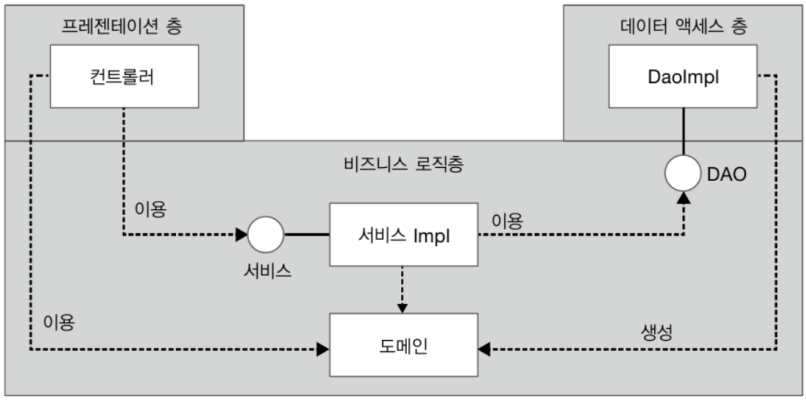
데이터 액세스 층은 기본적으로 RDB(테이블) 액세스를 비즈니스 로직에서 숨기고, 비즈니스 로직에 필요한 데이터를 테이블에서 취득해서 오브젝트에 매핑하는 것이다. 이렇게 오브젝트와 RDB를 매핑하는 것을 O/R 매핑이라고 한다(O는 Object, R은 Relational).

1) - ②: 그것의 결합도를 줄이기 위한 방법

- 오목형 레이어

웹 애플리케이션의 레이어는 크게 비즈니스와 관련된 부분과 비즈니스 로직의 결과를 어떻게

표현할지 구현하는 두 부분으로 나눌 수 있다. 비즈니스 로직은 애플리케이션에서 가장 중요하며, 비즈니스 로직의 결과를 어떻게 다룰지 구현하는 기술이 비즈니스 로직에 영향을 미치지 않는 것이 좋은 설계이다. 그러므로 기존의 세로형 레이어를 버리고 새로운 형태의 레이어를 생각해야 한다. 그것이 바로 오목형 레이어이다.



위 그림 안의 동그라미는 UML의 인터페이스, 사각형은 클래스이다. 오목형 레이어는 안정 의존 원칙(Stable Dependencies Principle)과 의존 관계 역전 원칙(Dependency Inversion Principle) 등에 뒷받침된 꽤 건실한 구조이다.

오목형 레이어에서 중요한 점은 프레젠테이션 층의 이용 기술이 브라우저를 사용하는 기술에서 스마트폰을 사용하는 기술로 변경됐을 경우와 데이터 액세스 층의 사용 기술이 RDB에서 NoSQL로 변경된 경우 등에서도 비즈니스 층에 영향을 미치지 않는다는 점이다. 즉, 비즈니스 층이야말로 시스템의 핵심이나 기반이므로 표시 메커니즘이나 영속화 메커니즘이 바뀌어도 영향을 받지 않게 만드는 것이 중요하다.

비즈니스 로직 층을 다른 층의 변경과 분리하려면 레이어를 형식으로만 분류할 것이 아니라 레이어의 결합 부분에 인터페이스를 도입한 약한 결합 설계나 구현을 고려해야 한다.

2) 스프링 프레임워크의 특징 3가지(POJO, IoC, DIxAOP)를 설명하시오.

- POJO(Plain Old Java Object) 관리

특정한 인터페이스를 구현하거나 상속을 받을 필요가 없는 가벼운 객체를 관리한다. DIxAOP 컨테이너에 실을 수 있는 일반 자바 오브젝트이다. EJB 컨테이너에 의존하는 EJB 컴포넌트는 단위 테스트를 수행하기 어렵다는 문제점이 있지만, DIxAOP 컨테이너로 관리되는 POJO는 DI 컨테이너에 의존하지 않는다는 특징 덕분에 단위 테스트를 쉽게 수행할 수 있다.

- IoC(Inversion of Control) 경량 컨테이너(light-weight Container)

필요에 따라 스프링 프레임워크에서 사용자의 코드를 호출한다. 일반 오브젝트의 생애 주기 관리나 오브젝트 간의 의존관계를 해결하는 아키텍처를 구현한다.

- DIxAOP(Dependency Injection, Aspect Oriented Programming)

DI는 각각의 계층이나 서비스들 간에 의존성이 존재할 경우 프레임워크가 서로 연결시켜준다. 변경과 품질관리가 용이하고, 확장성이 증가한다.

AOP는 트랜잭션이나 로깅, 보안과 같이 여러 모듈에서 공통적으로 사용하는 기능의 경우 해당 기능을 분리하여 관리한다. 프로그램 가독성이 높아지고, 기술을 은닉에 용이하다.

DIxAOP 컨테이너는 POJO라고 부르는, 컨테이너와 프레임워크 등에 의존하지 않는 일반 오브젝트의 생명 주기 관리나 오브젝트 간의 의존 관계를 해결하는 아키텍처를 구현한 컨테이너를 말한다. 스프링으로 대표되는 고성능 DIxAOP 컨테이너는 EJB의 장점인 선언적 트랜잭션 관리를 POJO로 구현할 수 있다. EJB 컨테이너 대신 DIxAOP 컨테이너를 이용하는 가장 큰 이유는 DIxAOP 컨테이너에 실을 수 있는 오브젝트가 POJO라고 부르는 일반 자바 오브젝트이기 때문이다. EJB컨테이너에 의존하는 EJB 컴포넌트는 단위 테스트를 수행하기 어렵다는 문제점이 있지만, DIxAOP 컨테이너로 관리되는 POJO는 DI 컨테이너에 의존하지 않는다는 특징 덕분에 단위 테스트를 쉽게 수행할 수 있다.

3) 모델 2방식의 첨부 예제를 참고하여 실습하고 결과 출력 페이지를 PrintScreen하여 첨부하시오.