VO사용법에 대한 의문점들

개발몬스터
2015. 4. 28. 17:20

그 동안 많은 프로젝트를 하면서 80프로 이상의 프로젝트에서 VO를 사용했습니다.
VO를 사용할 때 특정 경우에는 더 비효율적인 코드가 나올 것 같다는 생각을 많이 했었고, 그로 인해 다른 분들과 많은 토론을 했었습니다. 마침 이번에 기회가 생겨 그 동안 가지고 있던 생각과 책을 보며 새로 생긴 생각에 대해 적어보고자 합니다. 주관적인 내용이 많이 개입되어있지만, 그냥 저런 생각도 있구나 하면서 봐주시면 감사하겠습니다.

ORM(+DDD)을 쓰지 않고 국내에서 많이 쓰는 SQL Mapper를 이용한 개발을 하는 경우, VO를 요즘 정의된 VO의 개념(불변적인)이 아닌 전송객체와 Entity 객체의 개념을 더한 구현을 주로 사용합니다. 이로 인해 개념의 혼동이 일어나 상황에 맞지 않는 사용을 하는 경우가 있는 것 같은 생각을 많이 하게 됩니다.

1. getter/setter의 의미란?

OOP에서 객체는 행위와 상태를 가진 그룹이라 정의되고, VO역시 행위(getXXX(),setXXX())와 상태(변수)를 가지게 됩니다.
다만 아래와 같은 VO는 객체의 행위가 아닌 단순히 상태의 행위가 됩니다.
이 무슨 전설의 레전드 같은 말이냐고 하시겠지만 제 표현력의 한계가..T.T

```
private String attr;
public String getAttr(){...};
public VOid setAttr(String attr){....};
```

OOP의 특징 중 하나는 캡슐화/정보은닉입니다.

● **정보 은닉**: 내부 동작을 외부로부터 숨기는 것. 각각의 미래의 변경은 하나의 특정모듈에 국한된다. 왜냐하면 본래의 구현 결정의 세부사항은 다른 모듈들에는 보이지 않으므로 설계를 변경하는 것은 어떤 다른 모듈에게 영향을 미치지 않는다.

● **캡슐화** : 관련 있는 개념을 한곳에 무리 짓는, 그리고 단일 이름으로 불릴 수 있어야 캡슐화가 되었다고 한다.

-Meilir Page-Jones, Fundamental Of Object Oriented Design in UML

VO(DTO)는 정보은닉이란 이유로 private 변수를 사용하고 getter/setter를 사용하면서도 오퍼레이션이 있는 객체와 분리되거나 set/get 함수를 그대로 입/출력하게 만들어 특정 동작에 대하여 정보은닉이라 못하고 제대로 캡슐화되지 못하는 단지 자료구조 개념의 코드가 되는 경우가 많습니다.

예로, 클린코드에서는 이와 같은 코드는 추천할만한 코드는 아니라고 나옵니다.

- 변수사이에 함수라는 계층을 넣는다고 구현이 저절로 감춰지지는 않는다.구현을 감추려면 추상화가 필요하다 ! 그저(형식 논리에 치우쳐)조회함수와 설정함수로 변수를 다룬다고 클래스가 되지는 않는다..... 아무생각 없이 조회/설정 함수를 추가하는 방법이 가장 나쁘다.

- 클린코드 로버트C.마틴

* 그 외 객체를 자료구조로 사용하는 것에 대한 글 : http://lacti.me/2011/10/03/getter-and-setter-at-java/

2. 변수 형 및 값 Validation이 VO의 장점일까?

"하지만 VO가 형 체크와 검증에 쓰기 좋지 않나요?"라는 말을 많이 들었습니다.

Java의 형에 맞춰 숫자, 문자, 날짜 형 등의 체크 가능하지만 실제 값 검증에 대해서는 고개가 갸우뚱?합니다. name은 string이고 age는 number형(int형)인 user 테이블이 있다고 했을 때, age에는 "홍길동"이란 값이 들어올 수 없기 때문에 가벼운 형 체크가 가능한 것은 맞습니다.

하지만, age에 -99,210000,-123 이란 값이 들어왔을 때 해당 값이 정확한 값이라고 할 수는 없습니다. 또한 name에 "null"이라던가 "00191" 같은 값이 들어올 때 틀리진 않지만 개발자가 의도한 값이라고 할 수 없습니다. 오히려 이 문제는 Generic Programming 개념이 컬렉션 형에 구현되는(자바에 Generic의 확장같은) 방식으로 푸는 것이 더 나은 방안이 되지 않을까란 생각을 하게 됩니다.

3. 명시적 인터페이스의 장점과 그로 인해 생기는 세밀함의 불일치

VO를 사용할 때 속성이 코드 내에 투영됨으로 생기는 명시적 인터페이스는 확연한 장점입니다.
하지만 ORM이 아닌 SQL Mapper를 사용할 때, 테이블 쿼리의 디자인을 그대로 VO에 투영하면 클래스가 Entity보다 늘어나게 되어 전체 코드의 복잡도가 증가하기도 합니다.

Hibernate 패러다임 불일치 중 하나인 problem of granularity (세밀함의 불일치) 문제라는 것이 있습니다.
SQL Mapper 사용 시, 입력변수와 출력변수가 매번 달리 정의되고 로직이 두 개의 계층으로 분리되는 현상이 자주 일어나는데, 과연 ORM만큼 세밀함의 불일치 문제를 해결할 수 있을지 또한 중복과 클래스의 증가를 해결할 수 있을지에 대한 생각을 많이 하게 됩니다.

<세밀함의 불일치 문제>

1개의 테이블에 저장된 데이터를 1개 이상의 클래스에 표현할 수 있다.
user 테이블을 아래와 같이 정의 할 때,

Create table users(
Id varchar(15) not null,
Name varchar(10),
Address_street varchar(10),

Address_city varchar(10),
Address_state varchar(10),
Address_zipcode varchar(10),
Address_country varchar(10),
Age number(3)
)

도메인 모델의 클래스는 User와 같이 구성단위가 큰 엔티티 클래스에서부터 Address와 같은 더 세밀한 클래스, 그리고 zipcode와 같은 String 타입의 프로퍼티(property)에 이르기까지 다양하게 세밀함을 정할 수 있다. 반면 SQL데이터베이스는 User같은 테이블과 Address_zipcode같은 열처럼 두 단계의 세밀함만 지원한다.

단순한 영속화 기술은 상당수 이러한 불일치를 반영하지 못해 결국 객체 모델에 덜 유연한 SQL표현법을 쓰게끔 한다.

<u>이러한 예로 우리는 zipcode라는 프로퍼티를 포함하는 User클래스를 무수히 볼수있다.</u>

- *하이버네이트 완벽 가이드(크리스찬 바우어, 개빈킹/박찬욱,백기선,이대엽)*

4. 거대 서비스 아키텍처에서 VO(DTO)의 설계관점의 문제

'토비의 스프링'이라는 책을 보면 흔히 사용되는 아키텍처 방식들 중 거대 서비스 계층 방식의 아키텍처가 있습니다(아래 그림 참조). 보통은 이것과 토비 스프링의 DB/SQL아키텍처를 구현 레벨에서 적절히 섞어 사용합니다
.

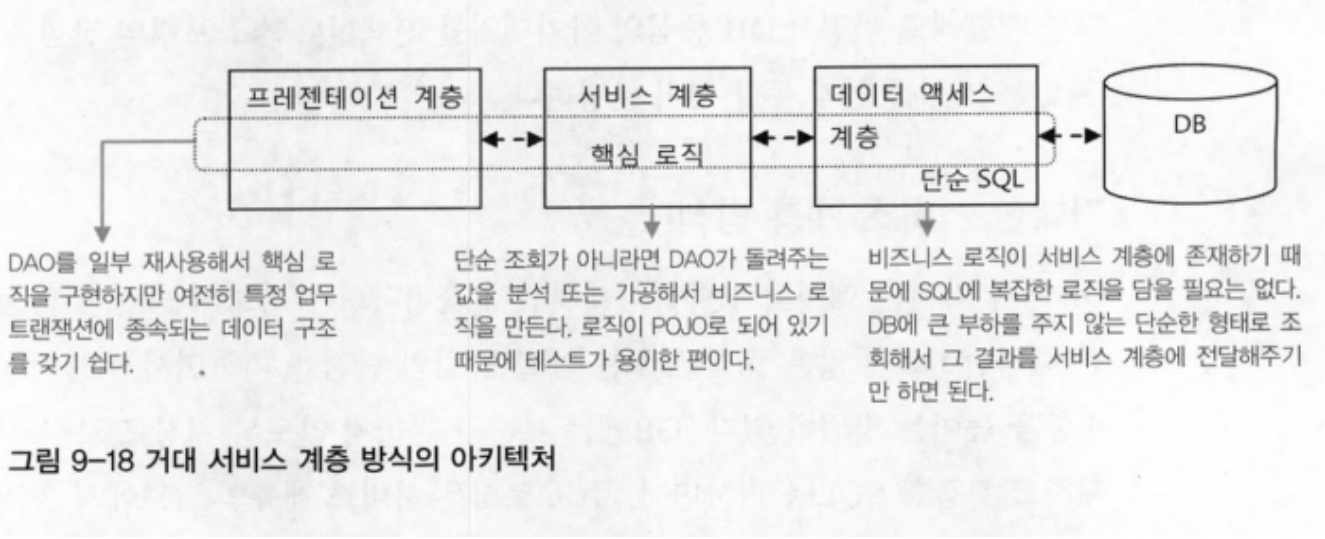


그림 9-18 거대 서비스 계층 방식의 아키텍처

- 비즈니스 로직을 DB나 SQL에 담는 경우에는 항상 최종 결과만 DAO에서 서비스 계층으로 전달된다. 반면 거대 서비스 계층 방식에서는 DAO에서 좀 더 단순한 결과를 돌려준다. 돌려준 정보를 분석/가공하며 비즈니스 로직을 적용하는 것은 서비스 계층 코드의 책임이 된다..... 비즈니스 로직의 대부분을 서비스 계층에 집중하는 이런 접근 방법은 결국 거대한 서비스 계층을 만들게 된다.

장점은 자바 언어의 장점을 활용해 로직을 구현할 수 있고 테스트 하기도 수월하다. 일부 DAO코드는 비즈니스 로직에서 공유해서 사용할 수 있다.
단점은 계층간의 결합도가 크고 업무 트랜잭션 단위로 만들어지기 때문에 비슷한 기능의 코드가 여러 메소드에 중복돼서 나타나기 쉽다. 이유는 DAO가 제공해주는 값의 포맷에 따라 이를 취급하는 방법이 달라지기 때문이다.
.....개발능력이 떨어지는 경우 자바 코드로 구현한 비즈니스 로직이 복잡한 SQL보다 더 이해하기 힘들 수도 있다.
....개발이 진행되면서 구현할 비즈니스 로직이나 설계에 변경이 생기거나 유지 보수 중에 수정할 필요가 있을 경우 코드를 손대기가 쉽지 않을 수 있다. 테스트가 불충분하거나 아예 없다면 오히려 SQL보다 더 다루기 힘든 스파게티 코드로 전락할 위험도 있다.

- 토비의 스프링 3.1 - 이일민

위에 나와 있듯, 거대 서비스 아키텍처 방식을 사용할 때에는 DAO와 연관된 입출력 변수인 DTO나 VO가 중복되는 것을 많이 봤습니다. 또한 데이터 액세스, 서비스, 프레젠테이션 계층들 사이에 VO(DTO)를 사용하는데 이로 인해 결합도가 높아지는 경우가 많습니다.

여기서 결합도(coupling)란 두 모듈간의 상호작용을 나타내는 것입니다.
그리고 모듈이란 [Stevens,Myers,Constantine,1974]등이 정의한 "시스템의 다른 부분이 자신을 호출할 수 있게 이름을 갖고 있는 그리고 자기 자신의 고유한 변수명을 갖고 있는 하나 또는 그 이상의 연속된 프로그램 문들의 집합"을 말합니다.

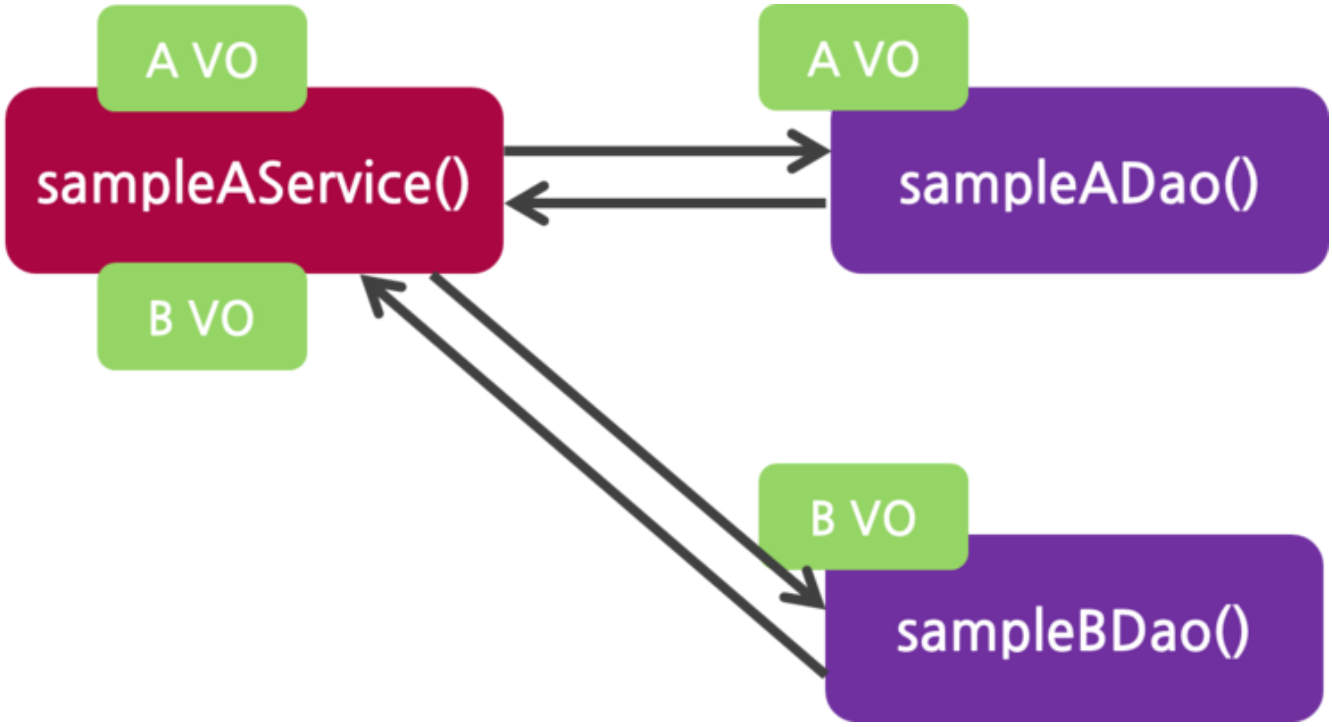
–객체지향 소프트웨어 공학

우리회사의 (이름을 말해선 안 되는)높은 분이 항상 말하는 느슨한 결합(loosely coupling)이 있습니다. 이는 결국 프로그램의 객체(메소드)간의, 혹은 티어(프로그램서버)간의 상호 작용이 적을수록 좋다는 뜻입니다.
예를 들어, 티어 간 이동 시(혹은 웹 서비스)에 RPC와 DOCUMENT방식에 대하여(예로 자바 직렬화를 이용한 교환방식과 JSON, XML 규약을 이용한 방식) 어느 것이 결합도가 낮은지는 쉽게 알 수 있습니다.

이 티어 간 이동을 레이어 간 이동으로 생각한다면 VO역시 Map등의 공용 객체에 비해 자바 직렬화와 같이 결합도가 높아집니다. VO는 각 모듈간에는 스탬프 결합도(Stamp coupling)를 가지고 있고 공용 객체는 (모듈간) 데이터 결합도를 가지고 있다고 볼 수 있는 것입니다. (VO는 공유 결합도로 보고 Map등을 스탬프 결합도로 볼 수도 있겠지만, 어느 방식이든 VO의 결합도가 높다는 것은 맞습니다)

결합도가 높다는 것은 응집도가 낮다는 반증일수 있습니다. 한 모듈이 다른 모듈에 대한 "의존성이 적어 진다"와 "독립성이 높아진다"는 연관이 있는 이야기입니다. 1개의 VO와 서비스 계층의 메소드(혹은 DAO의 Query 로직)는 메소드간 결합도만 높아지고 응집도, 캡슐화 등에 문제가 생깁니다.

그리고 좀 더 넓게 본다면 각 계층간의 결합도가 높아지는 것 입니다.



- 각 계층엔 반드시 VO가 구현되어있어야 한다. 그로 인하여 결합도 증가.
- VO를 객체간 메시지로 볼 때 기본변수, 자료구조라도 상관없다.

Domain Model(+ORM)에서는 위와 같은 상황에서 Entity를 반영하고 DO처럼 특정 행위를 객체에 투영하여 사용하지만 SQL Mapper를 사용하는 경우 위와 같이 사용할 수 없는 한계가 있다고 생각합니다. 그렇기 때문에, VO(DTO)를 사용할 때엔 응집도와 결합도를 고려하여 컬렉션 객체 혹은 다른 Data Holder 개념의 공용 객체를 사용하는 방안을 우선적으로 생각하는 것이 좋습니다.

5. 결국....

VO를 쓴다고 유지보수가 편해지고 Map을 쓴다고 유지보수가 불편해진다기 보다는 어떤 아키텍처를 어떻게 적용할 것인가 그리고 그 아키텍처에 맞는 방식은 무엇인가 생각하고 타성에 젖어 쓰지 말자가 제가 말하고 싶은 주제입니다.

원래 '토비 스프링'에서 소개된 DDD와, 거대 서비스 아키텍쳐, DB/SQL 아키텍처에 대해 제 나름대로 다시 설명하고 VO와 Map을 어느 경우에 적용하는 것이 최선인지를, DTO의 개념에선 왜 제 생각에 VO보다 컬렉션이 더 유리한지 자세히 적고 싶었지만, 실제로 써보니 너무 길고 장황해지는 것 같아(사실 너무 글 솜씨가 너무 없어서..) 세 번이나 수정한 끝에 GG칩니다...

제 글에 분명 틀린 부분도 많고 오해가 있을 수도 있을 것입니다. 때리지는 마시고 살포시 볼만 한 책 추천해주시면 감사하겠습니다^^

추가적으로, 아래는 VO나 DTO 혹은 Entity의 개념을 혼합하여 이야기 하는 분들이 많아 정리해보았습니다. 우선 Map과 VO에 대하여 이야기를 하려면 어느 정도 용어에 대한 개념을 구분할 필요가 있어, 각각의 정의나 개념에 대해 각 서적 및 인터넷에서 발췌한 내용이니 참고하시기 바랍니다.

[용어 정의]

A. VO

l 엔터프라이즈 애플리케이션 아키텍처 패턴.마틴파울러

"그 동등함은 고유성에 기반을 두지 않는 작고 단순한 객체"
ex: new aVO("1") == new bVO("1") 이 성립되는 객체
가능하면 생성시의 데이터가 불변인 객체라 정의
- http://martinfowler.com/bliki/ValueObject.html

l J2EE Design Patterns Applied.나디아나쉬

"Value Object 패턴은 처음 Entity EJB를 사용할 때 필요한 원격객체 호출횟수를 최소화하는 ..(생략).. 만들어졌다."라고 정의
데이터의 논리적인 View를 표현하며 티어간 데이터 전달을 할 수 있는 패턴

l 도메인모델을 이용한 J2EE(POJO) 컴포넌트 만들기.김형준

"멀티티어 환경에서 티어간 데이터이동이 필요한 경우 주로 사용되는 패턴이다.
VO의 경우 최초 VO 생성한 곳에서만 멤버변수 값을 설정할 수 있고..(생략)..
다른 티어에서는 값을 변경할 수 없도록 구성한다.
여기서는 VO보다는 DTO용어를 사용한다.이유는..(생략).."

l MVVM 패턴을 이용한 엔터프라이즈 애플리케이션 – 라파엘레 가로팔로

도메인 내의 어떤 개념을 나타내기 위해 사용하는 객체로, 상태가 변경되지 않으며 도메인 내에서 유일성을 지니지 않는다.

l 도메인 주도 설계 – 에릭 에반스

개념적 식별성을 갖지 않으면서 도메인의 서술적 측면을 나타내는 객체를 VALUE OBJECT라 한다. 모델에 포함된 어떤 요소의 속성에만 관심 있다면 그것을 VO로 분류하라..또한 VO는 불변적으로 다뤄라. VO에는 아무 식별성을 부여하지 말고 엔티티를 유지하는데 필요한 설계상의 복잡성을 피하라.

각 서적이나 사람마다 정의가 조금씩 틀리지만 각 개발자 블로그 및 이 이외의 서적들에서 일반적인 개념은 불변적인 값을 가진 자바 개체로 여겨지고 또한 DTO와는 약간 다른 개념으로 보고 있습니다. 하지만 필드에서 주로 이야기하는 VO의 개념은 EJB의 Entity Bean과 DTO를 혼합하거나 DDD의 Domain Object 개념에 DTO개념을 가진 자바빈즈 관례를 따르는 객체를 말하는 것 같습니다.

사실 VO정의가 서적마저도 조금씩 달라서 정답은 없는 것 같습니다.
VO가 불변적인 객체가 아닌 자바빈즈 규약과 같은 관례로 쓰이게 된 이유를 찾던 중 흥미로운 블로그 글을 찾아 링크 합니다.
- http://faildev.blogspot.kr/2012/05/bean.html ([java] object generator library는 immutable class를 만들 수 없다)

B. 자바빈즈

l 자바빈즈 관례 (위키디피아)

클래스는 직렬화되어야 한다.
클래스는 기본 생성자를 가지고 있어야 한다.

클래스의 속성들은 getter, setter 혹은 표준 명명법을 따르는 메서드들을 사용해 접근할 수 있어야 한다.

C. Entity

l 하이버네이트 완벽 가이드(크리스찬 바우어, 개빈킹/박찬욱, 백기선, 이대엽)

엔티티는 기본적인 비즈니스 관련 주체를 나타내는 영속 타입이다. 바꿔 말하면 애플리케이션에서 다뤄야 할 어떤 클래스나 타입이 다른 것에 비해 더 중요하게 부각된다는 뜻이다.
엔티티 타입 객체는 자신만의 데이터베이스 동일성을 가진다.
엔티티는 저마다 자체적인 생명주기가 있으며 따라서 다른 엔티티와 독립적으로 존재할 수 있다.

l 객체지향 소프트웨어 공학 8판 –STEPHEN R. SCHACH

엔티티 클래스는 오래 유지될 정보를 모델화 한 것이다. MSG Foundation 소프트웨어 프로덕트인 경우 Investment Class는 엔티티 클래스이다. 왜냐하면 투자들에 대한 정보는 오래 유지되어야 한다.

l 엔터프라이즈 자바 어플리케이션 구축 –브렛 맥래프린지음/임백준역

엔티티 빈은 데이터를 둘러싸고 있는 비즈니스 로직을 제거했을 때 남게 되는 순수한 데이터를 자바 내부에 표현하기 위한 목적으로 탄생하였다.

l 도메인 주도 설계 – 에릭 에반스

어떤 객체를 일차적으로 해당 객체의 식별성으로 정의할 경우 그 객체를 ENTITY라 한다. 엔티티에는 모델링과 설계상의 특수한 고려사항이 포함돼 있다.

Entity란 어떤 비즈니스의 관련 있는 영속적인 속성들의 집합이라고 생각하시면 됩니다. EJB에서 Entity란 단어를 쓰긴 했는데(3번째 예), 개념은 비슷하지만 Entity Bean과 Entity는 다른 내용이며, Entity Bean은 또 CMP/BMP등으로 나뉘어 집니다.
참고로, Entity 개념을 데이터베이스모델에 적용하면 테이블과 매우 유사하므로 데이터 베이스의 테이블의 개념을 애플리케이션으로 가져온다면 Entity라고 불리기도 합니다.

* Domain Model(Object)

l 코어 J2EE 패턴.디팍알루어

전에 관련 내용을 봤는데 책을 지금 빌릴 수 없어 발췌는 생략합니다..OTL

l 엔터프라이즈 애플리케이션 아키텍쳐 패턴

간단한 도메인 모델은 각 데이터 베이스당 대부분 하나의 도메인 객체를 가진 데이터베이스 설계와 매우 비슷해 보인다. 풍부한 도메인 모델은 상속, 전략,패턴, 내부적으로 연결된 작은 객체들의 복잡한 망을 가지고 있어 데이터베이스 설계와는 사뭇 달라 보일 수 있다.

l 하이버네이트 완벽 가이드(크리스찬 바우어, 개빈킹/박찬욱, 백기선, 이대엽)

도메인모델 구현은 다른 직교적인 자바 API에 의존해서는 안 되는 중요코드다.
도메인모델은 비즈니스 도메인 모델링하고만 관계를 맺어야 한다. 도메인 모델을 구현하는 클래스에는 횡단관심사에 해당하는 코드를 넣어서는 안 된다.
하이버네이트의 주된 목표는 풍부한 도메인 모델의 가장 중요한 요구사항으로 간주하여 분리했던, 구성단위가 세밀한 도메인 모델을 지원하는 것이다. 그것이 바로 POJO를 이용하는 한가지 이유다.

D. DTO

Transfer Object 패턴 혹은 Data Transfer Object 패턴.

l 코어 J2EE 패턴.디팍알루어

티어 사이의 다중 데이터 요소를 전송해야 합니다.

J2EE 애플리케이션은 Session Facade와 Business Object(BO)로 서버 측 비즈니스 컴포넌트를 구현합니다....(생략)..그러나 엔터프라이즈 빈 메소드를 매번 호출하는 것은 그 성격상 원격 호출이기 때문에 효율성 문제를 야기할 수 있습니다...(생략)..심지어 원격컴포넌트에 접근하는 것이 아니더라도...(생략)..데이터를 전송하고 받기 위해서는 큰 단위 인터페이스로 접근 해야 합니다

l 엔터프라이즈 애플리케이션 아키텍쳐 패턴

"메쏘드 호출 횟수를 줄이기 위해 프로세스간 데이터를 옮기는 객체"

"원격 인테페이스로 작업할 때 각각의 호출은 비용이 많이 든다...(생략)...이에 대한 해결책은 호출을 위한 모든 데이터를 유지하는 데이터전송 객체를 생각하는 것이다..(생략)...이것은 원격 연결을 지나갈 수 있도록 직렬화 가능해야 한다."

l MVVM 패턴을 이용한 엔터프라이즈 애플리케이션 – 라파엘레 가로팔로

DTO는 평면적인 객체다. 직렬화가 가능하며 레이어/티어/객체 간 데이터를 전달하기 위해 사용된다. 비즈니스 로직을 포함하지 않으며 일반적으로 부모/자식 객체에 대한 순환참조를 하지 않는다.

DTO는 연결자체가 큰 비용을 가진 티어 간의 구성에서 데이터 전송의 비용을 줄이기 위하여 데이터를 모아서 보내는 패턴 혹은 모아놓은 Data Holder개념의 객체입니다. 예전 EJB의 구성을 보면 분산환경에서 객체를 호출하므로 주로 사용하던 것인데 티어간 구성이 아닌 현재의 통합 티어 환경에서 계층 간의 데이터 이동 시에도 동일한 개념으로 주로 사용되고 있습니다.