엔티티(Entity)와 값 객체(Value Object)의 차이

Feb 6, 2016

자바 웹 개발 워크북을 공부하면서 값 객체(Value Object)라는 개념을 처음 보게 되었다. 책에서는 값 객체를 값을 전달하는 역할을 하는 객체라고 설명한다. 개념을 소개하는 부분에서는 그런 개념이구나 하고 넘어갔는데 책의 중반부로 가니 예제에서 DB 테이블과 매핑되는 모든 객체들을 값 객체로 다루고 있었다. MVC에서 Model에 해당되는 부분이 단지 DB에서 값만 읽어 오는 역할 밖에 못하나 하는 생각이 들었다. 예전에 .NET Framework 기반에서 개발을 할 때는 ORM 프레임워크에서 테이블과 매핑해주는 객체를 엔티티(Entity)라고 불렀다. 같은 개념을 Java에서는 값 객체라 하고, .NET 에서는 엔티티라고 부르는 것일까? 이런 의문들이 생겨 구글에 검색해보니 명쾌하게 설명해주는 글이 있었다. 이 글이 신빙성이 있는지 확신은 할 수 없지만 구글에서 'difference between entity and value object'라고 검색했을 때 가장 위에 위치한 것을 보아 충분히 읽어볼 만한 글이라 생각한다. 그리고 내가 공부삼아 굳이 번역을 하긴 했지만 원문이 어렵지 않으므로 되도록 원문을 읽어보길 바란다.

원문 보기

여러분은 컴퓨터 프로그래밍을 하면 할 수록 엄청나게 많은 새로운 이론과 개념들을 맞닥뜨리게 된다. 그 중에서 직관적으로 이해되지 않는 하나는 바로 값 객체(Value Object)이다. 값 객체는 도메인 주도 설계(Domain Driven Design)에서 중요한 개념이다.당장 도메인 주도 설계에 대해 잘 모른다 하더라도 이 글을 이해하는 데 문제는 없다. 이 글은 값 객체를 설명하는 데 초점을 두고 있다. 하지만 이 글이 여러분이 추후 도메인 주도 설계를 이해하는 첫 걸음이 되기를 희망한다.

참고: 이 글은 여러분이 객체 지향 프로그래밍(Object Oriented Programming)에 대해 잘 알고 있다고 가정한다. 그렇지 않다면 객체 지향 프로그래밍에 대해 먼저 학습하고 오길 바란다.

엔티티 vs 값 객체

객체 지향 프로그래밍에서는 관련된 속성과 메서드를 객체로 표현한다. 예를 들어, 어플리케이션 상에서 사람은 객체가 될 수 있다. 사람은 이름, 이메일, 비밀번호 등 많은 속성을 지닌다. 데이터베이스에서는 이 사람을 `id` 로 식별한다. 이는 사람이 자신의 이름, 이메일, 비밀번호를 바꾸더라도 `id` 가 바뀌지 않으면 같은 사람임을 의미한다. 이렇게 속성이 바뀌더라도 여전히 같은 것으로 인식되는 객체를 엔티티라고 한다. 엔티티는 속성이 바뀔 수 있기 때문에 가변(mutable) 객체다. 엔티티의 정체성은 `id` 로 표현된다.

우리 어플리케이션이 사람의 현재 위치를 기록할 수 있다고 상상해보자. 그 사람이 성공적으로 인터넷에 연결하고 우리 어플리케이션에 인증을 하면 새로운 위치 객체가 생성된다. 위치 객체는 위도와 경도 값을 가진다. 우리는 위치 객체가 어떤 위치 정보를 담고 있는지에만 관심이 있기 때문에 위치 객체는 값 객체가 된다.

어떤 사람의 위치가 바뀌었을 때 우리는 기존의 위치 객체를 갱신하지 않는다. 단순히 새로운 위치 객체를 생성한다. 위치 객체는 생성된 이후로 소멸될 때까지 자신의 속성을 변경하지 않는다. 객체의 속성이 바뀔 수 없을 때, 그 객체를 불변(immutable) 객체라고 한다.

또한 값 객체는 식별자에 의해 같음(equality)이 결정되지 않는다. 예를 들어 당신이 같은 위도와 경도를 가진 위치 객체를 두 개 생성했다면 그 두 객체는 같은 객체이다. 반면에 사람 객체는 `id` 라는 고유한 식별자로 같음을 판단한다. 같은 이름을 가진 두 개의 사람 객체가 있다고 해도 두 객체는 같지 않을 수 있다.

어떻게 값 객체를 식별할까?

이제 우리는 어떤 객체가 `id` 로 식별된다면 엔티티로, 그렇지 않다면 값 객체라고 구분할 수 있다. 엔티티의 속성은 바뀔 수 있다. 하지만 엔티티는 고유한 식별자로 표현되기 때문에 우리 시스템 상에서 같은 객체로 인식된다. 반면 값 객체는 갖고 있는 값으로표현되는 객체이다. 한번 생성되면 그 값을 바꿀 수 없으며, 우리는 그것이 어떤 개체인지 신경쓰지 않는다.

언제 엔티티를 쓰고, 값 객체를 써야할지 어떻게 알 수 있을까? 그것은 어플리케이션의 상황에 따라 결정된다. 처음 예시로 돌아가보자. 우리 어플리케이션은 더이상 단순한 소셜 서비스가 아니다. 포스퀘어와 동일 다고 가정해보자. 모든 위치 객체는 고유한 식별자를 지닌다. 많은 사람들이 같은 위치에서 체크인을 할 수 있기 때문이다. 이제 위치 객체는 더이상 값 객체가 아닌 엔티티가 된다.

예제를 바꿔보자. 우리가 파워 플랜트의 소유자라고 상상해보라. 파워 플랜트는 철책(securify fence) 주변에서 일어나는 모든 활동을 기록한다. 감시의 목적으로 철책 주변의 많은 위치에서 활동이 기록된다. 이 때 우리는 특정 위치에서 일어나는 활동에 대해 관심이 있기 때문에 철책 주변의 위치는 엔티티가 된다. 수상한 사람이 우리가 감시하는 위치로 들어오게 되면 그 사건은 데이터베이스에 저장된다. 이 예제에서 사람은 값 객체이다. 우리는 감시중인 위치에 들어 온 사람이 어떤 사람인지 관심이 없기 때문이다. 단지 그 사람이 감시중인 위치에 걸려들었을 뿐이다.

위의 예제와 보았듯 어떤 객체가 엔티티인지 값 객체인지는 여러분의 어플리케이션이 그것을 어떻게 사

용할 것인지에 달려있다. 일반적으로 원칙, 날짜, 숫자, 금액은 값 객체인 경우가 많다. 그리고 사람, 제품, 파일, 판매는 엔티티인 경우가 대부분이다.

왜 값 객체와 엔티티를 구분하는 것이 중요할까?

여러분은 **왜 값 객체와 엔티티의 구분이 중요할까?** 하고 생각할지 모른다. 이것은 다음과 같은 이유로 매우 중요하다.

첫 번째로, 여러분이 같은 속성을 가진 두 개의 엔티티를 갖고 있을 때, 두 객체는 다른 `id` 를 갖고 있기 때문에 같지 않다. 하지만 같은 속성을 가진 두 값 객체를 갖고 있는 경우에 이 두 객체는 같은 객체이므로 두 객체를 자유롭게 바꿔서 쓸 수 있다. 한 객체를 다른 객체로 대체할 수 있을 때 그 객체는 값 객체이다. 반면 엔티티는 다른 것으로 교체할 경우 원치않는 부작용이 발생하게 된다.

두 번째로, 오랜 시간동안 엔티티의 속성은 변하지만 여전히 그것은 같은 엔티티이다. 어떤 사용자가 자신의 이메일 주소를 변경하는 것이 그 예이다. 하지만 어플리케이션이 값 객체의 속성을 변경하려고 할 때, 그 객체는 사라지고 새로운 객체가 대신하게 된다. 결제를 할 때 당신이 냈던 돈이 거스름돈이 되어 다시 돌아오지 않는다. 당신은 낮은 금액의 새로운 돈 객체를 받는다.

결론

엔티티와 값 객체의 차이는 도메인 주도 설계에서 중요한 개념이다. 우리가 어플리케이션에서 실제 세계를 모델링할 때 이 차이는 아주 중요하다. 이 글에서 언급했듯이 여러분이 어떤 것을 개발하는지 이해하는 것이 중요하다. 어떤 어플리케이션을 만드냐에 따라 어떤 객체가 엔티티가 될지 값 객체가 될지 결정된다. 언뜻 보기에 어떤 객체가 식별자를 가져야 할 것 같다고 해서 무작정 그것을 엔티티로 만들면 안된다. 불변 객체가 되어야 할 개념을 엔티티로 모델링하면 원치 않는 부작용이 발생할 수 있다.

불변 객체는 어플리케이션이 의도한 대로 동작하게 하는 중요한 요소이다. 돈과 같은 것을 모델링할 때 값 객체를 사용하면 그것이 시간에 따라 변하지 않는다는 것을 보장해준다.

엔티티와 값 객체의 차이가 항상 뚜렷하지는 않다. 때로 그 차이를 구분하는 것은 여러분이 만들고자 하는 어플리케이션에 대한 완벽한 이해를 요구한다. 하지만 실제 세계를 여러분의 코드로 모델링하는 데 있어 엔티티와 값 객체의 차이를 아는 것은 매우 중요하다.

자바 웹 개발 워크북은 Spring 프레임워크를 이해하는 데 아주 좋은 책이라고 생각한다. 하지만 이런 사소해보이지만 중요한 개념을 설명하는 데는 다소 소홀하지 않았나 하는 생각이 든다. 저자가 그런 것 까지 설명하면 책의 내용이 방대해질까 걱정되어 어물쩍 넘어 간 것일 수도 있겠다.

하지만 이런 내용을 잘못 이해하면 엉뚱한 방향으로 개발할 수 있다. 책의 예제를 보고 모든 DB와 매핑된 객체들을 값 객체로 오인하면 한 속성만 바꾸어 저장해도 될 일을 매번 객체를 생성하게 되고, 그 객체를 관리하는 다른 클래스를 만들게 될 수도 있다.

PREV NEXT

수 없을때에만 VO를 만들어서 사용하자!" 입니다.
혹 이것에 대해서는 어떻게 생각하시는지요?? 실제로 개발할때는 두가지 형태가 혼재 되어있고 문제는 없습니다만.. ㅎ

Reply • Share ›

Hanghee, Yi ➚ 김성남 • a year ago

질문내용은 이 글이 설명하는 것과는 전혀 관련없는 내용같네요; 애초에 이 글에서의 Entity와 JPA의 Entity가 같은 것을 의미하지 않습니다.

질문 내용에 대해서는 질문자님이 말씀하신 Entity라는 걸 JPA 의 테이블로 본다고 했을 때, 보통은 이렇게 사용할 수 있는 경우가 크게 많지 않고(단순한 View 성의 프로그램이라면 가능할지도 모르겠네요) 나중에 결국 비즈니스 등이 확장되면 결국 Entity 를 그대로 쓸 수 없는 상황이 많이 오게 됩니다. MSA 라면 그 상황은 더 빨리 오게 될거구요. JPA OSIV 라도 활성화 되어있다면 DB에 직접적인 타격이 갈 수도 있습니다.

2 Reply • Share ›