DDD (도메인 주도 설계)

주차	30주차
스터디 일자	@2024/06/06

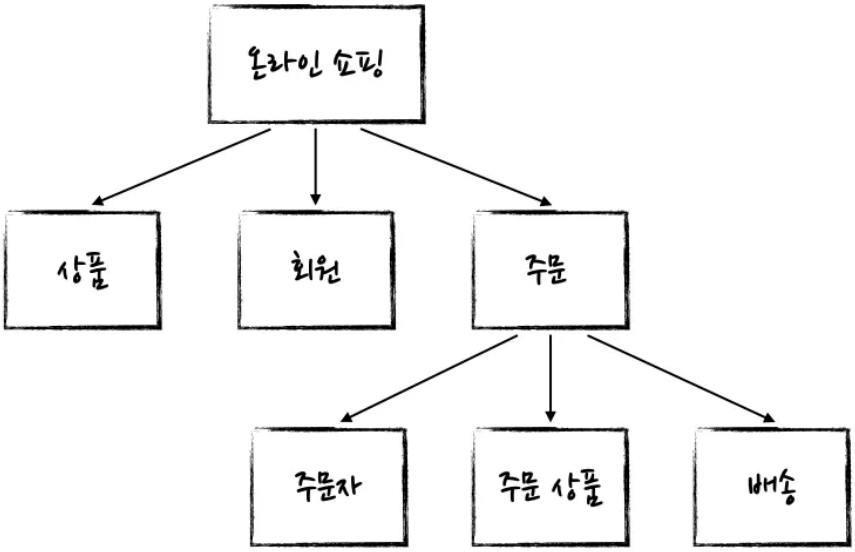
DDD (Domain Driven Design)

> 도메인 주도 설계(DDD) 란, **"도메인 중심으로 소프트웨어를 설계하는 방법"** 이라고 말할 수 있다.

⇒ 여기서 **도메인**이란 무엇인가??

도메인

> 도메인 주도 설계에서 도메인이란 우리가 소프트웨어로 **"해결하고자 하는 문제 영역"** 을 의미한다.



위처럼 도메인은 다시 하위의 여러 도메인으로 나누어지곤 한다.

도메인을 나누는 기준에는 절대적인 기준이 있는 것은 아니고

상위 수준에서는 **좀 더 추상화된 도메인**으로,

하위 수준에서는 **좀 더 세분화된 도메인**으로 나눌 수 있다.

도메인 특징

도메인 모델은 도메인 객체들로 표현할 수 있는데, 이러한 도메인 객체는 기본적으로 **엔티티(Entity)**와 **밸류(Value)**로 구분할 수 있다.

엔티티 (Entity)

엔티티는 식별성과 연속성을 가지는 객체로,

고유한 식별자로 식별할 수 있으며 자신의 상태와 라이프사이클을 가지는 도메인 객체다.

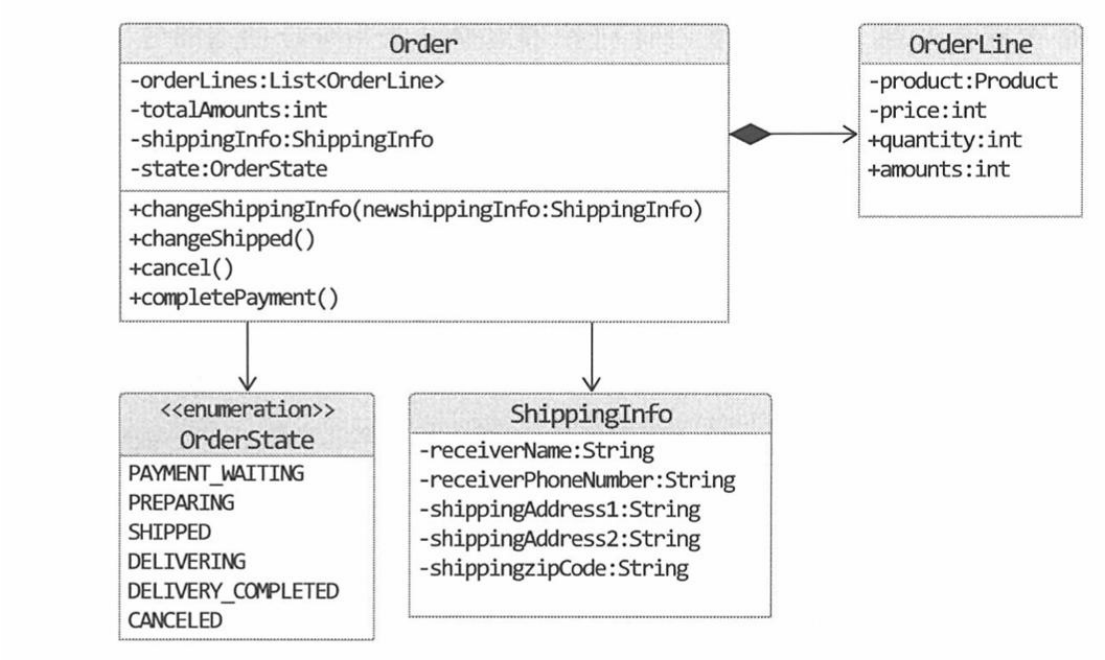
밸류 (Value)

밸류는 개념적으로 묶을 수 있는 데이터 집합을 표현한다.

밸류를 사용하면 각각의 데이터를 단일로 취급할 때보다 **표현력을 향상시킬 수 있다.**

👉 예를 들면,

`주문` 객체에 배송 관련 정보인 `배송지 주소`, `배송지 우편번호`, `수령자 이름`과 같이 개별 속성으로 취급하는 것보다는, `배송 정보`라는 **밸류로 모아서 관리**하면 **표현력이 좀 더 향상**되는 것을 느낄 수 있다.



setter/getter 메서드 지양

위에 나온 엔티티와 밸류의 **메서드(행위)로 기능과 제약을 표현**하고, 습관적으로 사용하는 **setter와 getter 메서드를 지양**해라.

why? ↴️

주문의 배송을 완료시키는 기능의 간단한 코드 예

```
public class Order() {
    private long id;
    private OrderStatus status;

    public OrderStatus getStatus() {
        return this.status;
    }

    public void setStatus(OrderStatus status) {
        this.status = status;
    }

    public boolean isDeliveryCompletable() {
        return this.status == OrderStatus.DELIVERY_IN_PROGRESS;
    }

    public void completeDelivery() {
        this.status = OrderStatus.DELIVERED;
    }
}
```

```
public class DeliveryCompleteService() {
    public void completeDelivery(final Order order) {

        // case 1
        if (order.getStatus() == OrderStatus.DELIVERY_IN_PROGRES
            order.setStatus(OrderStatus.DELIVERED);
        }

        // case 2
        if (order.isDeliveryCompletable()) {
            order.completeDelivery();
        }
```

```
    }
}
```

`case 1`보다 `case 2` 가 더 무엇을 하고자 하는지 **의도가 분명**하게 느껴진다.

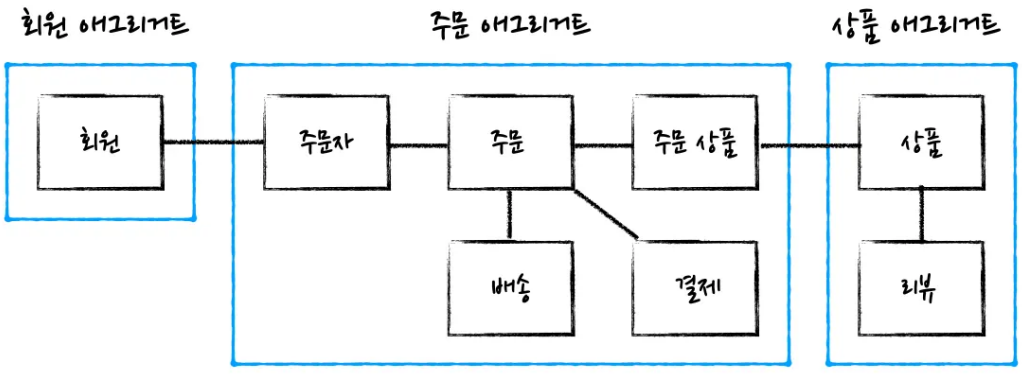
고로, `case 2` 처럼 만든 객체가 **도메인 기능을 잘 표현하고 있는 도메인 객체**이다.

애그리거트(Aggregate)

서비스가 자랄수록 도메인 역시 함께 자라기 때문에 도메인 모델은 점차 복잡해진다.

이러한 복잡도를 관리하기 위해 **도메인 객체들의 묶음이자 집합체**인 **애그리거트**가 필요하다.

애그리거트를 사용하면 도메인 객체를 좀 더 **상위 수준으로 추상화** 할 수 있다.



아래 내용을 참고해서 도메인 객체들을 애그리거트로 구성할 수 있다.

- 하나의 애그리거트에 포함된 객체들은 도메인 규칙과 요구사항에 따라 **함께 취급된다.**
- 하나의 애그리거트에 포함된 객체들은 **동일하거나 유사한 라이프사이클(함께 생성되거나, 함께 제거되거나, 함께 변경되는 등)을 가져야 한다.**
- 하나의 애그리거트에 포함된 객체는 **다른 애그리거트에 포함되지 않아야 한다.**
- 애그리거트는 자기 자신을 관리할 수 있지만 **다른 애그리거트를 직접 관리하지는 않는다.**

애그리거트 루트(Aggregate root)

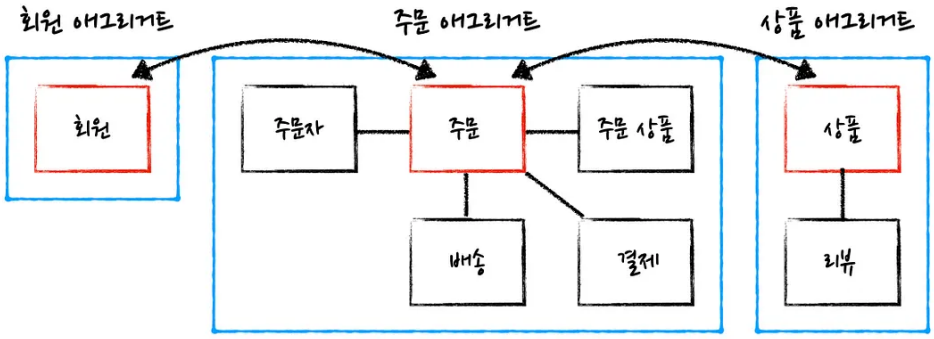
애그리거트에는 포함된 객체들의 **대표**가 되는 **애그리거트 루트** 가 필요하다.

애그리거트에는 다수의 객체들이 포함되어 있고 이들은 함께 움직이면서 일관성을 유지해야 하는데,

바깥에서 애그리거트 내부의 객체들에게 직접 접근해서 상태나 속성을 변경해버리면 일관성이 깨져버리기 때문에

애그리거트 바깥에서 애그리거트에 직접 접근할 수 있는 곳은 오직 애그리거트 루트 뿐이어야 한다.

→ 이러한 애그리거트 루트가 이러한 **창구 역할**을 하면 애그리거트에 포함된 객체들의 **일관성을 유지**할 수 있다.



위의 이미지로 예를 들면,

`주문 애그리거트`의 애그리거트 루트인 `주문 객체`를 통해서 `주문의 배송을 완료한다` 라는 기능을 제공하고, 이에 따라 주문 애그리거트 내부에서는 여러 객체들의 상태나 속성이 알맞게 변경될 수 있다.

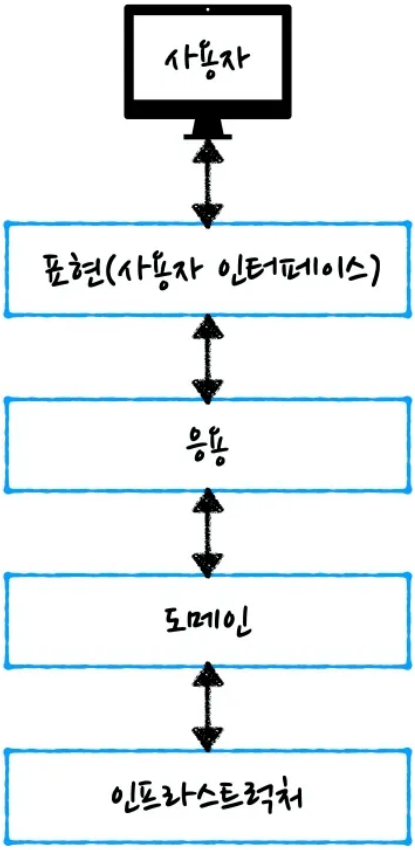
만약 애그리거트 루트를 접근하지 않고 애그리거트 내부에 있는 객체에 접근하게 한다면 어떻게 될까?

→ `배송 객체`에만 접근해서 상태를 배송 완료 상태로 바꿔버린다면?

다른 객체들은 배송 완료에 대한 어떤 기능을 수행하지 않았지만 배송 객체 혼자 배송 완료 상태가 되어 **애그리거트 내의 일관성이 깨지게 될 것이다.**

⚠ 이런 상황을 피하기 위해 애그리거트 바깥에서는 **반드시 애그리거트 루트를 통해서만 접근해야 한다 !**

DIP (Dependency Inversion Principle)



보통의 계층형 아키텍처 구조

계층형 아키텍처에서는 **일반적으로 상위 계층이 하위 계층에 의존한다**.

표현 계층 → 응용 계층에 의존, 응용 계층 → 도메인 계층에 의존

우리가 다루는 모듈은 `고수준 모듈`과 `저수준 모듈` 로 나눌 수 있다.

고수준 모듈 : 의미 있는 단일 기능을 제공하는 모듈 (= 응용 계층, 도메인 계층)

저수준 모듈 : 고수준 모듈의 기능을 구현하기 위해 필요한 하위 기능의 실제 구현인 모듈 (= 인프라스트럭처 계층)

흔히 고수준 모듈이 저수준 모듈에 의존하도록 구현하는데, 이럴 경우 **저수준 모듈의 변경이 곧 고수준 모듈의 변경**으로 이뤄지곤 한다.

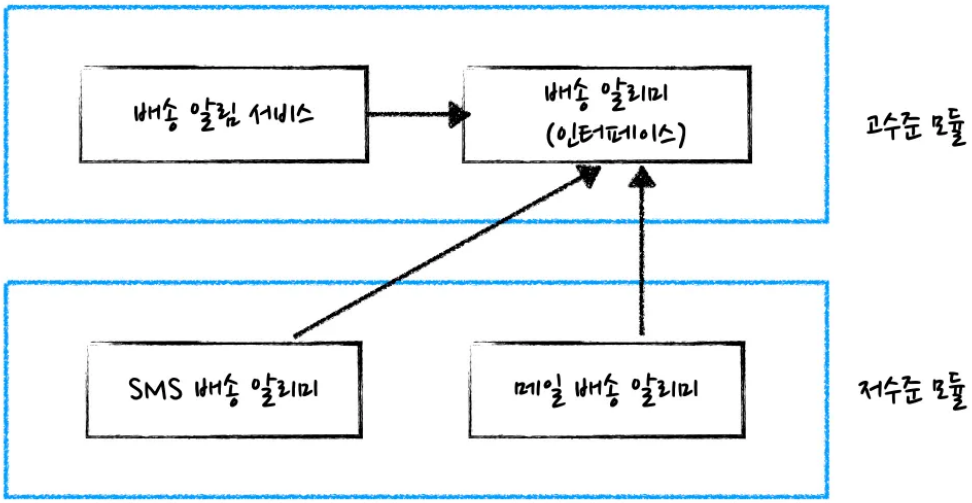
고수준에서의 `배송 알림` 자체에 변경이 없어도,

저수준인 `메일로 배송 알림 메일을 전송한다` 라는 저수준의 기능이

`SMS로 배송 알림 메시지를 전송한다` 로 바뀐다면

고수준 모듈에서도 변경이 발생하는 것.

⇒ 이러한 단점을 극복하기 위해 의존관계를 역전시켜서 **저수준 모듈이 고수준 모듈에 의존하도록 구현하는 것을 DIP** 라고 한다.



위처럼 `배송 알림` 기능을 정의한 `배송 알리미` 인터페이스를 만들고,

저수준 모듈에서 `배송 알리미` 인터페이스를 구현한 `메일 배송 알리미`나 `SMS 배송 알리미`를 만드는 식 !

⇒ 이렇게 DIP를 만족시키면 저수준 기능이 바뀌더라도 고수준 모듈에서의 변경은 최소화할 수 있다.

✨ 이러한 특징 외에도 응용 서비스, 리포지토리, 도메인 서비스, 도메인 이벤트, CQRS, Bounded Context, Context Map 등등등 ...

더 많은 내용이 있지만 다음에 각자 알아보는 것으로 합시다.. ~ ^^ 😉

출처

https://medium.com/myrealtrip-product/what-is-domain-driven-design-f6fd54051590

https://yoonbing9.tistory.com/121

https://tech.kakao.com/posts/555