07. 도메인 서비스

version 2018.35

여러 애그리거트가 필요한 기능

도메인 영역의 코드를 작성하다 보면 한 애그리거트로 기능을 구현할 수 없을 때가 있습니다. 대표적인 예가 결제 금액 계산 로직입니다. 실제로 결제 금액을 계산할 때 다음과 같은 애그리거트가 필요할 것입니다.

- 상품 애그리거트
- 주문 애그리거트
- 할인 쿠폰 애그리거트
- 회원 애그리거트

만약 결제 금액 계산에 필요한 로직을 주문 애그리거트가 구현한다고 가정해봅시다. 이 경우 애그리거트는 자신의 책임 범위를 넘어서는 기능을 구현 하기 때문에 코드가 길어지고 외부에 대한 의존이 높아지게 됩니다. 이는 코드를 복잡하게 만들고 애그리거트의 범위를 넘어서는 도메인 개념이 들어오므로써 기존 애그리거트의 의미를 잃어버리게 만듭니다.

이러한 문제를 해결하는 방법은 바로 **도메인 서비스**를 별도로 구현하는 것입니다.

도메인 서비스

할인 금액 규칙 계산처럼 한 애그리거트에 넣기 애매한 도메인 개념을 구현하려면 애그리거트에 억지로 넣기보다는 도메인 서비스를 이용해서 도메인 개념을 명시적으로 드러내면 됩니다. 응용 영역의 서비스가 응용 로직을 다룬다면 도메인 서비스는 도메인 로직을 다룹니다.

도메인 서비스가 도메인 영역의 애그리거트나 밸류와 같은 다른 구성요소와 비교할때 다른 점이 있다면 그것은 바로 상태 없이 로직만 구현한다는 점입니다. 도메인 서비스를 구현하는 데 필요한 상태는 애그리거트나 다른 방법으로 전달받습니다. 할인 금액 규칙 계산에 대한 도메인 서비스를 구현한 코드입니다.

```
public class DiscountCalculationService {

    public Money calculateDiscountAmounts(List<OrderLine> orderLines,
List<Coupon> coupons, MemberGrade grade) {
        Money couponDiscount = coupons
                                .stream()
                                .map(coupon -> calculateDiscount(coupon))
                                .reduce(Money(0), (v1, v2) -> v1.add(v2));

        Money membershipDiscount =
caculateDiscount(orderer.getMember().getGrade());

        return couponDiscount.add(membershipDiscount);
    }

    private Money calculateDiscount(Coupon coupon) {
        ...
    }

    private Money calculateDiscount(MemberGrade grade) {
        ...
    }
}
```

할인 계산 서비스를 사용하는 주체는 애그리거트가 될 수도 있고 응용 서비스가 될 수도 있습니다. DiscountCalculation-Service를 다음과 같이 애그리거트의 결제 금액 계산 기능에 전달하면 사용 주체는 애그리거트가 됩니다.

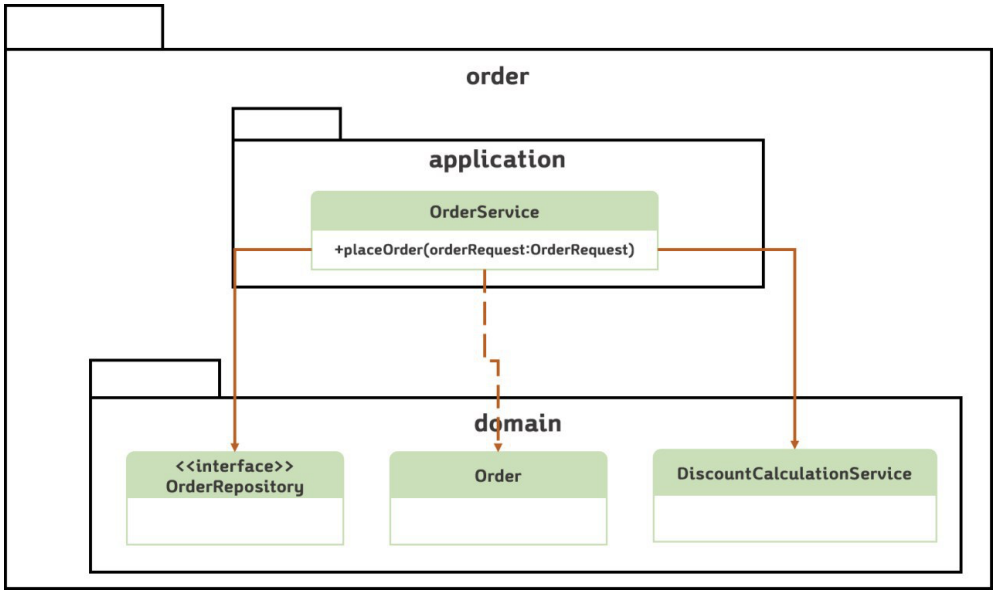
```
public class Order {

    public void calculateAmounts(DiscountCalculationService disCalSvc,
MemberGrade grade) {
        Money totalAmounts = getTotalAmounts();
        Money discountAmounts =
disCalSvc.calculateDiscountAmounts(this.orderLines, this.coupons, grade);
        this.paymentAmounts = totalAmounts.minus(discountAmounts);
    }
    ...
}
```

> 애그리거트 객체에 도메인 서비스를 전달하는 것은 응용 서비스의 책임입니다.

도메인 서비스의 패키지 위치

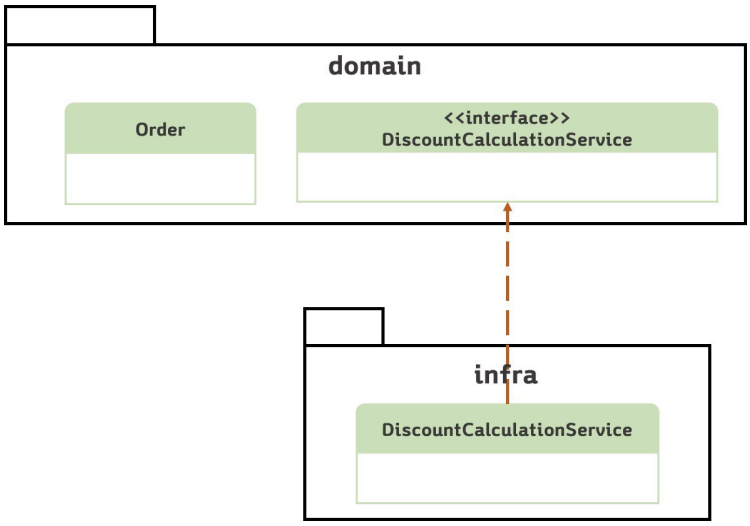
도메인 서비스는 도메인 로직을 실행하므로 도메인 서비스의 위치는 다른 도메인 구성 요소와 동일한 패키지에 위치합니다. 예를 들어, 주문 금액 계산을 위한 도메인 서비스는 다음 그림과 같이 주문 애그리거트와 동일 패키지에 위치합니다..



도메인 서비스의 개수가 많거나 엔티티나 밸류와 같은 다른 구성요소와 명시적으로 구분하고 싶다면 `domain` 패키지 밑에 `domain.model`, `domain.service`, `domain.repository`와 같이 하위 패키지를 구분해서 위치시켜도 됩니다.

도메인 서비스의 인터페이스와 클래스

도메인 서비스의 로직이 고정되어 있지 않은 경우 도메인 서비스 자체를 인터페이스로 구현하고 이를 구현한 클래스를 둘 수도 있습니다. 특히 도메인 로직을 외부 시스템이나 별도 엔진을 이용해서 구현해야 할 경우에 인터페이스와 클래스를 분리하게 됩니다. 예를 들어, 할인 금액 계산 로직을 룰 엔진을 이용해서 구현한다면 아래의 그림처럼 도메인 영역에는 도메인 서비스 인터페이스가 위치하고 실체 구현은 인프라스트럭처 영역에 위치 시킬수 있습니다.



이 처럼 도메인 서비스의 구현이 특정 구현 기술에 의존적이거나 외부 시스템의 API를 실행한다면 도메인 영역의 도메인 서비스는 인터페이스로 추상화해야 합니다.