DAOUTECH

DDD

#1. 개요

플랫폼서비스팀

김설한

2021.10.13

• 목차

- 도메인 모델
- 아키텍처
- 도메인 영역의 구성요소
- 모듈 구성/ 패키지 구조

01

도메인 모델

도메인 모델

- DDD (Domain Driven Design) : 애플리케이션을 비즈니스 도메인 별로 나누어 설계/개발 하는 방법론.
- 특징

  ① 비즈니스 도메인 그 자체와 로직에 초점을 맞춘다.

  ② 도메인 용어와 코드 용어를 일치시켜 가독성을 높이고 퍼포먼스를 향상시킨다.

  즉, 보편적인(ubiquitous) 언어를 사용한다.

  ① 도메인 모델부터 코드까지 항상 함께인 모델을 지향한다.

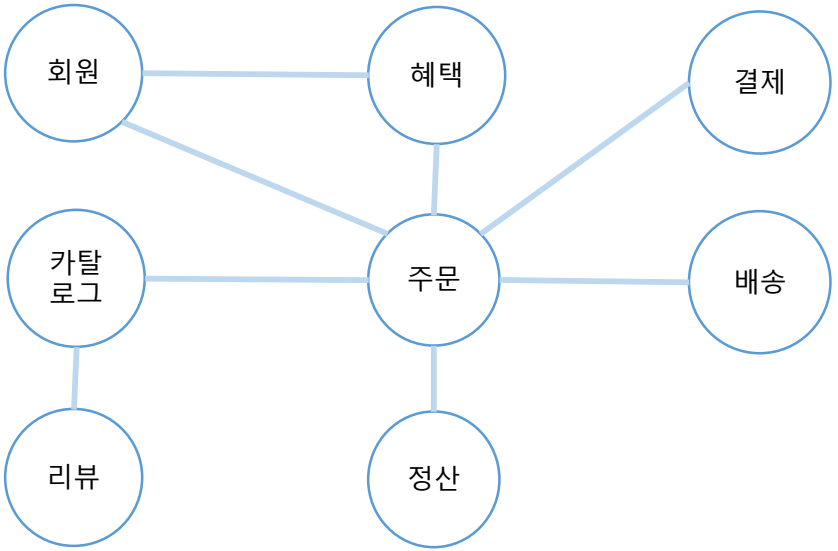
- 도메인 : 유사한 업무의 집합. DDD에서 말하는 도메인은 비즈니스 도메인을 의미.

  소프트웨어로 해결하고자 하는 문제 영역 (일반적인 요구 사항).

도메인 모델

- 도메인 모델
  - 특정 도메인을 개념적으로 표현한 것.
  - 단순히 구현 수준의 모델이 아닌, 여러 이해 당사자가 이해할 수 있는 모델로 작성하는 것에 의미가 있다.
  - 표현하는 방법은 도메인의 상황에 따라 달라질 수 있다. (UML, 상태 그래프 등 알맞은 표현 사용)
  - 도메인은 여러 하위 도메인으로 구성될 수 있으며, 하위 도메인들이 상호작용해 기능을 제공한다.

[상위 도메인]



[하위 도메인]

도메인 모델

- 도메인의 종류는 엔티티와 벨류로 나눌 수 있다.
  - 엔티티
    - ✓ 고유한 식별자를 갖는다.
    - ✓ 식별자 생성 시점은 도메인의 특징/사용 기술에 따라 달라질 수 있다.
      - ① UUID
      - ② 값을 직접 입력 (이메일/아이디 등)
      - ③ 일련 번호 사용 (DB 시퀀스 혹은 자동 증가 값)
  - 벨류
    - ✓ 값을 표현하는 클래스/객체로서 개념적으로 완전한 하나를 표현.
    - ✓ 벨류 객체 데이터 변경은 기존 데이터를 변경하는 방식이 아닌,

      새로운 벨류 객체를 생성하는 방식을 권장한다. (Immutable)

도메인 모델

- 도메인 모델에 Setter 사용하지 않기
  - Setter를 사용하면 도메인의 핵심 개념이나 의도를 명확히 표현하기 어렵다.
  - 도메인 객체를 생성하는 시점에 완벽한 상태를 보장할 수 없으므로, 잠정적 에러 가능성을 갖게 된다.
  - 이를 막기 위해, 객체를 생성하는 시점에 완벽한 상태 값을 주입해줘야 한다.
    - ➢ **생성자** 혹은 **팩토리 메서드** 등을 활용해 필요한 객체/ 데이터를 모두 받는다.

- 도메인 용어
  - 실제 비즈니스 도메인에 맞는 유의미한 단어를 선택해, 코드 해석 과정을 줄이는 것이 중요하다.

```
emum class OrderState {
    STEP1, STPE2, STEP3 ...
}
```

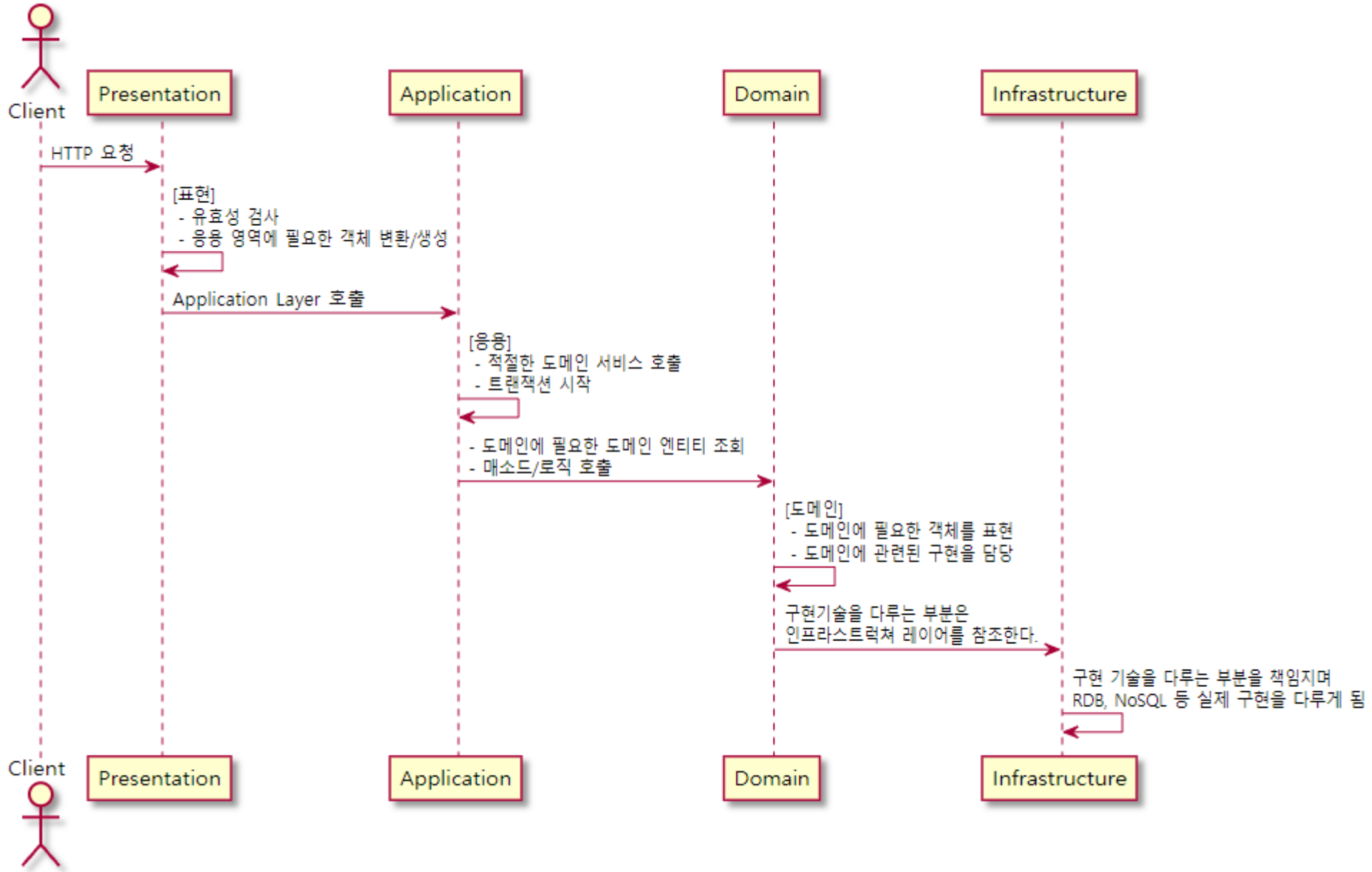
Better

```
emum class OrderState(val statement: String) {
    PAYMENT_WAITING("입금대기"),
    PREPARING("배송준비"),
    SHIPPED("출고"),
    DELIVERING("배송중"),
    DELIVERY_COMPLETED("배송완료")
}
```

02

아키텍처

아키텍처


Client
Presentation
Application
Domain
Infrastructure
HTTP 요청
[표현]
- 유효성 검사
- 응용 영역에 필요한 객체 변환/생성
Application Layer 호출
[응용]
- 적절한 도메인 서비스 호출
- 트랜잭션 시작
- 도메인에 필요한 도메인 엔티티 조회
- 매소드/로직 호출
[도메인]
- 도메인에 필요한 객체를 표현
- 도메인에 관련된 구현을 담당
구현기술을 다루는 부분은
인프라스트럭쳐 레이어를 참조한다.
구현 기술을 다루는 부분을 책임지며
RDB, NoSQL 등 실제 구현을 다루게 됨
Client
Presentation
Application
Domain
Infrastructure

아키텍처

- 표현 (Presentation) : 사용자(사용자/외부 시스템) 요청을 처리. 요청에 대한 유효성 검사/ 응용 레이어 호출을 위한 객체 생성/변환 작업이 주로 이루어 진다.
- 응용 (Application) : 응용 서비스에 대한 대부분의 트랜잭션이 시작되는 지점. 도메인 서비스를 호출해 도메인에 필요한 구현을 도메인 영역 객체에 위임한다.
- 도메인 (Domain) : 도메인에 필요한 데이터를 표현하고, 관련 기능을 구현한다.
- 인프라 스트럭처 (Infrastructure) : 실제 구현을 담당한다. 응용/도메인 레이어가 의존하는 구현 기술을 해당 레이어가 담당한다 .

각 계층은 보통 표현 → 응용 → 도메인 → 인프라 스트럭처 순으로 의존하지만,

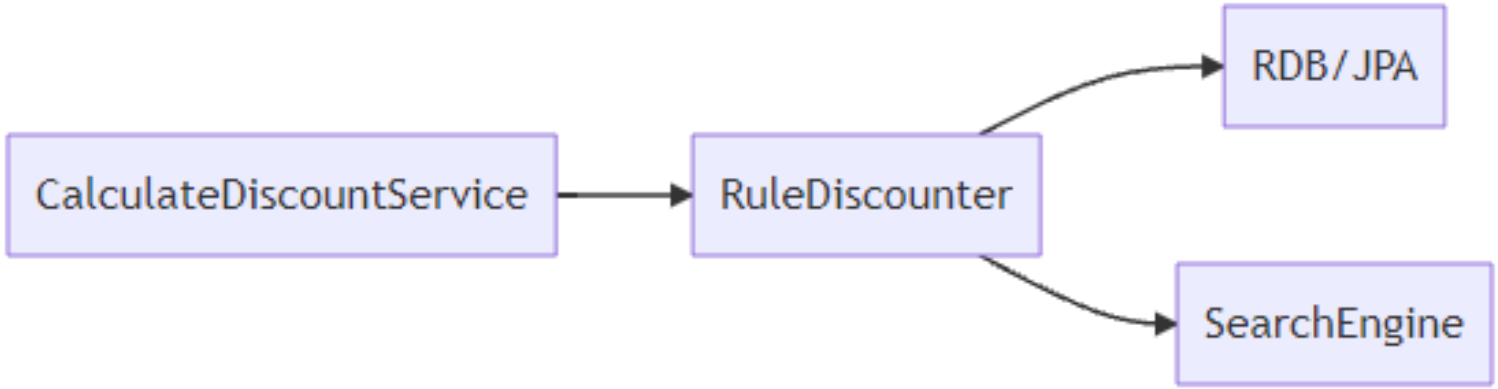
상황에 따라 유연하게, 응용 → 인프라 스트럭처 로 직접 참조하기도 한다.

아키텍처

- 응용/도메인 계층이 인프라 스트럭처 계층에 의존하게 되면,

  (1) **실제 구현 부분에 대한 변경이 어려워**지고 (2) **테스트가 어려워**지는 문제점이 발생한다.

- DIP를 이용해 저 수준의 구현 모듈이 고 수준 모듈에 의존하도록하면 문제를 해결할 수 있다.
- **DIP** (Dependency Inversion Principle, 의존 역전 원칙)
  - 추상 인터페이스로 메시지를 주고받아(method 호출) 관계를 최대한 느슨하게 만드는 방법.
  - 인터페이스를 참조하도록 해, 실제 구현 기술에 관계 없이 필요한 기능 자체에만 집중한 표현이 가능하다.



아키텍처

```java
public class CalculateDiscountService {
  private RuleDiscounter ruleDiscounter;

  public CalculateDiscaountService(RuleDiscounter ruleDiscounter){
    this.ruleDiscounter = ruleDiscounter;
  }

  public Money calculateDiscount(List<OrderLine> orderLines, String customerId){
    Customer customer = findCustomer(customerId);
    return ruleDiscounter.applyRules(customer, orderLines);
  }
  // ...
}
```

참조

```java
public interface RuleDiscounter {
  public Money applyRules(Customer customer, List<OrderLine> orderLines);
}
```

상속

```java
public class DroolsRuleDiscounter implements RuleDiscounter {
  private KieContainer kContainer;

  public DroolsRuleDiscounter(){
    //...
  }

  @Override
  public Money applyRule(Customer customer, List<OrderLine> orderLines){
    KieSession kSession = kContainer.newKieSession("discountSession");
    try {
      // ...
      kSession.fireAllRules();
    } finally{
      kSession.dispose();
    }

    return money.toImmutableMoney();
  }
}
```

실제 구현 부분

(1) **실제 구현 부분의 변경이 용이**해지며,

(2) 실제 구현 부분의 구현이 덜 되었더라도

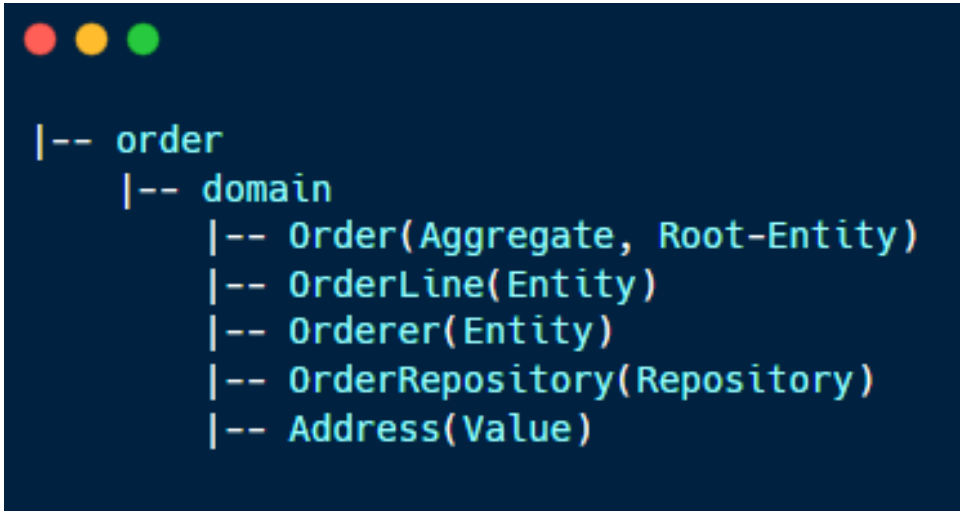
(상위 레벨인) **서비스 층 테스트가 가능**해 진다.

03

도메인 영역의 구성요소

도메인 영역의 구성요소

- 엔티티 : 식별자를 가지는 객체. 라이프 사이클을 갖는다.
- 벨류 : 완전한 하나의 값을 표현. 도메인 속성을 표현하는 목적으로 주로 사용된다.
- 애그리거트 : 연관 객체의 집합. 관련된 엔티티와 벨류 객체를 개념적으로 묶어 놓은 것이다.
- 레파지토리 : 도메인 모델의 영속성을 처리한다.
- 도메인 서비스 : 특정 엔티티에 종속적이지 않은 도메인의 로직을 제공한다.



도메인 영역의 구성요소

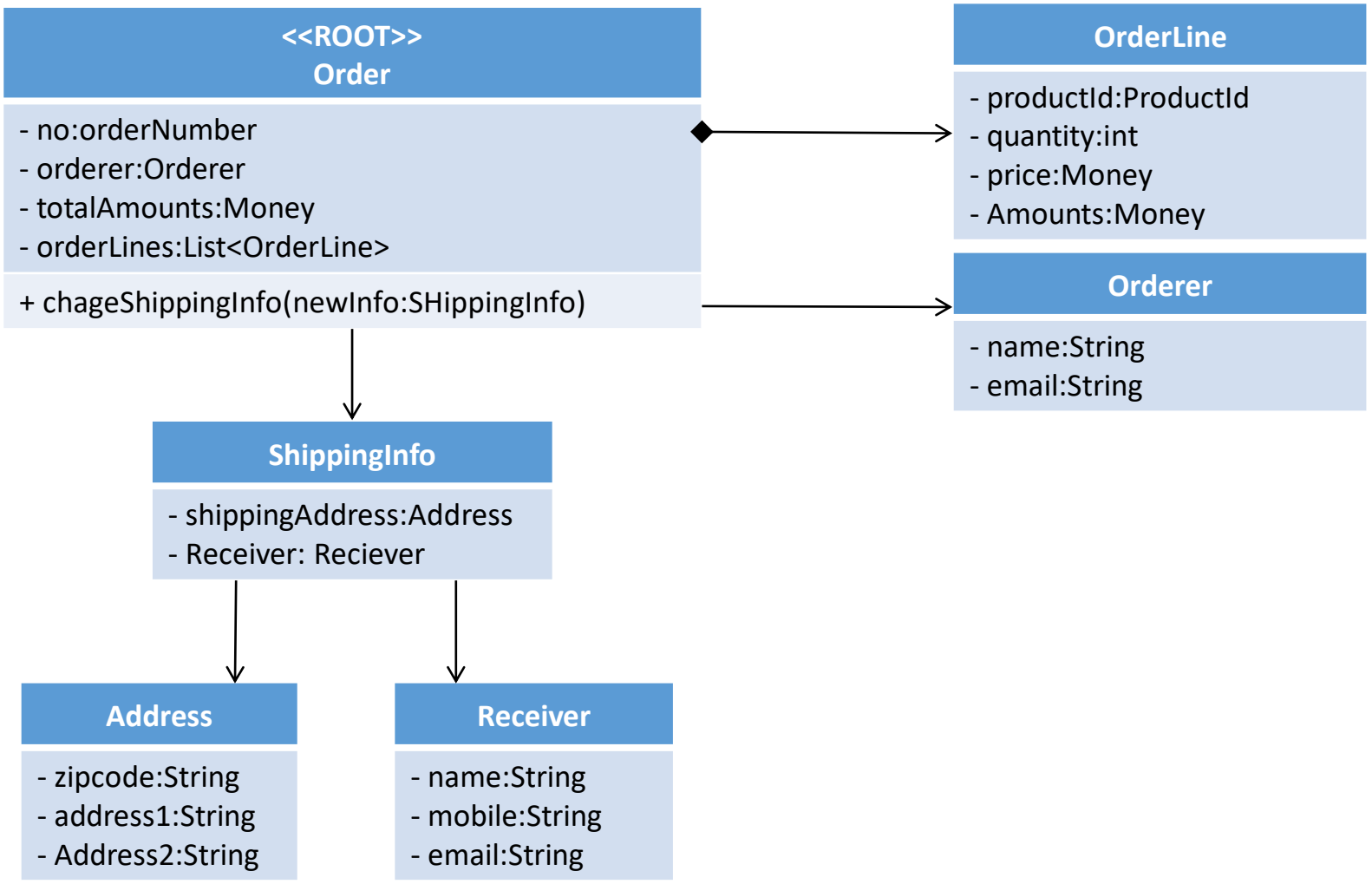
- 엔티티 (Entity)
  - 세터(Setter) 지양 : 무분별한 상태 변경 방지. 세터를 만들더라도 private/protected 접근 자 사용.
  - 도메인 객체 생성은 완전한 상태로 이뤄지는 것을 지향한다.
  - 개념적으로 하나를 표현하는 값이라면 밸류 객체를 적극적으로 활용한다.

- 애그리거트 (Aggregate)
  - 비슷한 도메인 모델을 상위 수준에서 접근. 연관/관련된 객체를 하나로 묶은 군집.
  - 애그리거트를 관리하는 루트 엔티티를 갖는다.

    루트트 엔티티는 연관 객체, 벨류를 이용해 도메인 주 기능을 제공한다.
  - 하위 객체/밸류에 대한 접근은 루트 엔티티를 통해 간접적으로 이뤄져야 하며,

    실제 구현을 내부로 숨겨 애그리거트 단위의 캡슐화가 가능해 진다.

도메인 영역의 구성요소



도메인 영역의 구성요소

- 레파지토리 (Repository)
  - 구현을 위한 도메인 모델. 도메인 객체를 영속성에서 사용하기 위한 영역.
  - 주로, 애그리거트 단위로 객체를 저장하고 조회하는 기능을 구현한다.
  - 레파지토리 인터페이스는 도메인 영역, 인터페이스 구현체는 인프라 스트럭처 영역에 속한다.

04

모듈 구성/ 패키지 구조

모듈 구성/ 패키지 구조

- 영역(Layer)별로 구성
  - 모듈 내에서 연관 서비스/엔티티 끼리 다시 패키징

```
|-- presentation
|-- domain
    |-- order
    |-- member
|-- application
    |-- order
    |-- member
|-- infrastructure
```

- 도메인(Domain)별로 구성
  - 도메인이 크다면 서브 도메인 별로 다시 나누는 전략

```
|-- order
    |-- presentation
    |-- domain
    |-- application
    |-- infrastructure
|-- member
    |-- presentation
    |-- domain
    |-- application
    |-- infrastructure
```

모듈 구성/ 패키지 구조 (spring 프로젝트)

- 영역(Layer)별로 구성
  - 레이어 단위로 구성하는 방법
  - 관심사가 다른 모듈 간 무분별한 DI가 발생할 수 있어,

    주의해서 사용해야 한다.

- 도메인(Domain)별로 구성
  - 관심 도메인 별로 패키지 구조를 작성
  - 개별 도메인 별 타입 개수가 늘어나거나,

    관심사 별로 다시 분리 해야 할 경우 도메인 패키지 내 재분리.

모듈 구성/ 패키지 구조 (spring 프로젝트)

도메인 별 상호 참조의 유무나 요구사항, 이해도 등에 따라

레이어 단위/ 도메인 단위 구조를 결정할 수 있다.

DDD Start! 도메인 주도 설계 구현과 핵심 개념 익히기 (최범균 저) 를 읽고 정리한 내용입니다.

감사합니다 !