🥈

라이브 세션-2022.12.23(금)-Spring Data JDBC를 통한 데이터 액세스 계층 구현

✅ 들어온 질문

1️⃣ AggregateReference를 같은 애그리거트내에서 Set 같은 컬렉션 대신에 사용할 수 없나요?

- 시도해보진 않았지만 **AggregateReference의 사용 목적**이 **다른 애그리거트 간의 참조**에 사용되는 것입니다.

2️⃣ DDD의 설계가 어려운 이유가 Aggregate를 나누는 부분이 어렵기 때문인가요?

- 네, 도메인의 경계를 잘 정의하는게 생각보다 까다롭습니다.
- 도메인 객체를 테이블과 연관짓는 것 역시 까다롭습니다.
- **DIP(Dependency Inversion Principle)**를 적절하게 정의하는게 조금 까다롭습니다.
  - DIP란?
    - 고수준의 기능(단일 기능)이 저수준의 기능(하위 기능)을 직접적으로 사용하지 않는 것
    - 예)
      - 커피 할인 기능(고수준의 단일 기능)
      - **어떤 기술**을 통한 고객 정보 조회 기능(하위 기능)
      - **어떤 기술**을 통한 할인 규칙 계산(하위 기능)
    - 고수준은 고수준 끼리만 의존하도록한다.
      - Java의 인터페이스와 연관이 있음

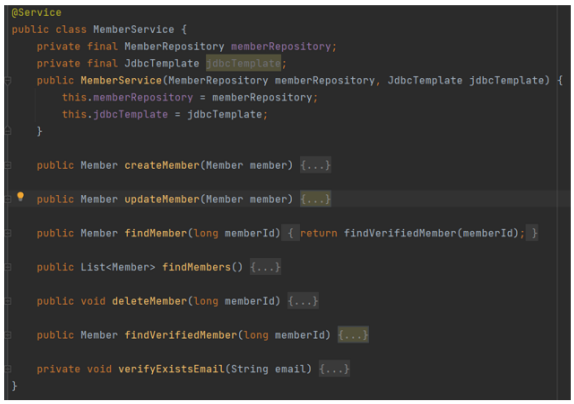
✅ 도메인 엔티티 클래스와 테이블 설계

1️⃣ DDD란?

- 도메인 주도 설계(Domain Driven Design)
- 한마디로 모든 기능을 도메인 모델 위주로 돌아가는 설계 기법
- 도메인(Domain)이란?
  - **비즈니스적인 어떤 업무 영역**
  - **우리가 실제로 현실 세계에서 접하는 업무의 한 영역**
- 코드로 이해

2️⃣ 빈약한 도메인 모델

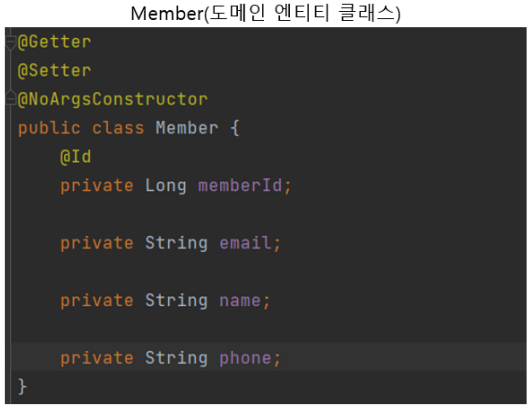
MemberService(서비스 클래스)



```
@Service
public class MemberService {
    private final MemberRepository memberRepository;
    private final JdbcTemplate jdbcTemplate;
    public MemberService(MemberRepository memberRepository, JdbcTemplate jdbcTemplate) {
        this.memberRepository = memberRepository;
        this.jdbcTemplate = jdbcTemplate;
    }

    public Member createMember(Member member) {...}

    public Member updateMember(Member member) {...}

    public Member findMember(long memberId) { return findVerifiedMember(memberId); }

    public List<Member> findMembers() {...}

    public void deleteMember(long memberId) {...}

    public Member findVerifiedMember(long memberId) {...}

    private void verifyExistsEmail(String email) {...}
}
```

서비스 클래스에 기능 집중

Member(도메인 엔티티 클래스)



```
@Getter
@Setter
@NoArgsConstructor
public class Member {
    @Id
    private Long memberId;

    private String email;

    private String name;

    private String phone;
}
```

기능이 없는 빈약한 도메인 모델

3️⃣ 풍부한 도메인 모델

MemberService(서비스 클래스)

```java
@Service
public class MemberService {
    private final MemberRepository memberRepository;
    private final JdbcTemplate jdbcTemplate;
    public MemberService(MemberRepository memberRepository, JdbcTemplate jdbcTemplate) {
        this.memberRepository = memberRepository;
        this.jdbcTemplate = jdbcTemplate;
    }
    ...
    ...
}
```

서비스 클래스의 기능 축소

기능 이전

Member(도메인 엔티티 클래스)

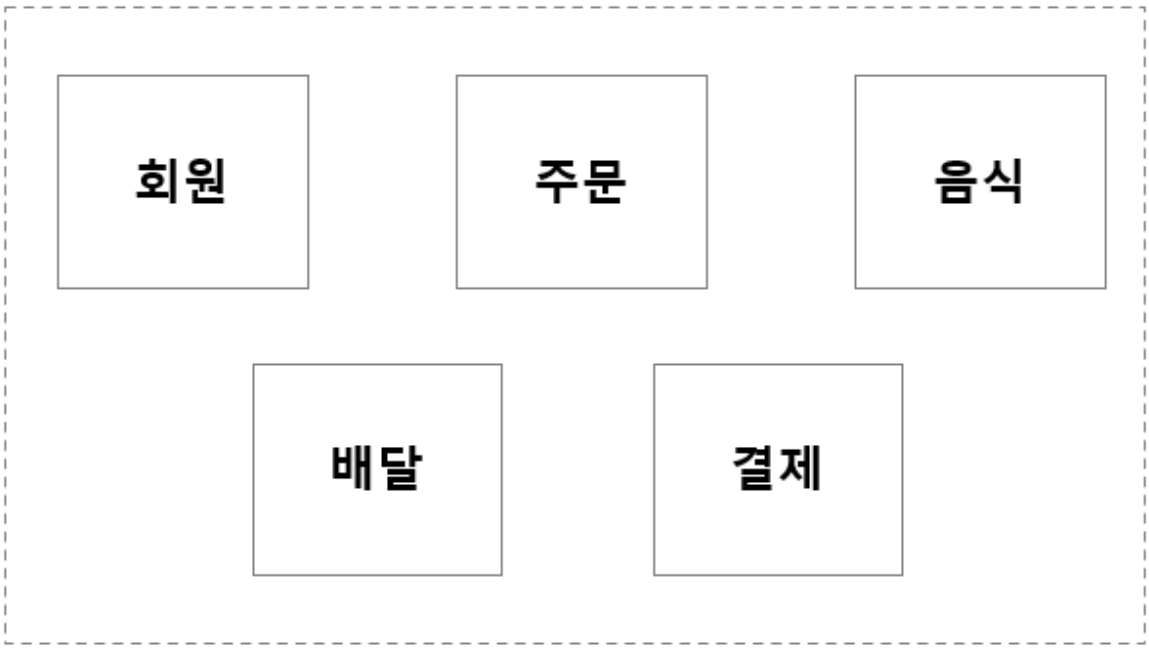
```java
@Getter
@Setter
@NoArgsConstructor
public class Member {
    @Id
    private Long memberId;

    private String email;

    private String name;

    private String phone;

    public Member createMember(Member member) {...}

    public Member updateMember(Member member) {...}

    public Member findMember(long memberId) { return findVerifiedMember(memberId); }

    public List<Member> findMembers() {...}

    public void deleteMember(long memberId) {...}

    public Member findVerifiedMember(long memberId) {...}

    private void verifyExistsEmail(String email) {...}
}
```

기능이 많은 풍부한 도메인 모델(Rich Domain)

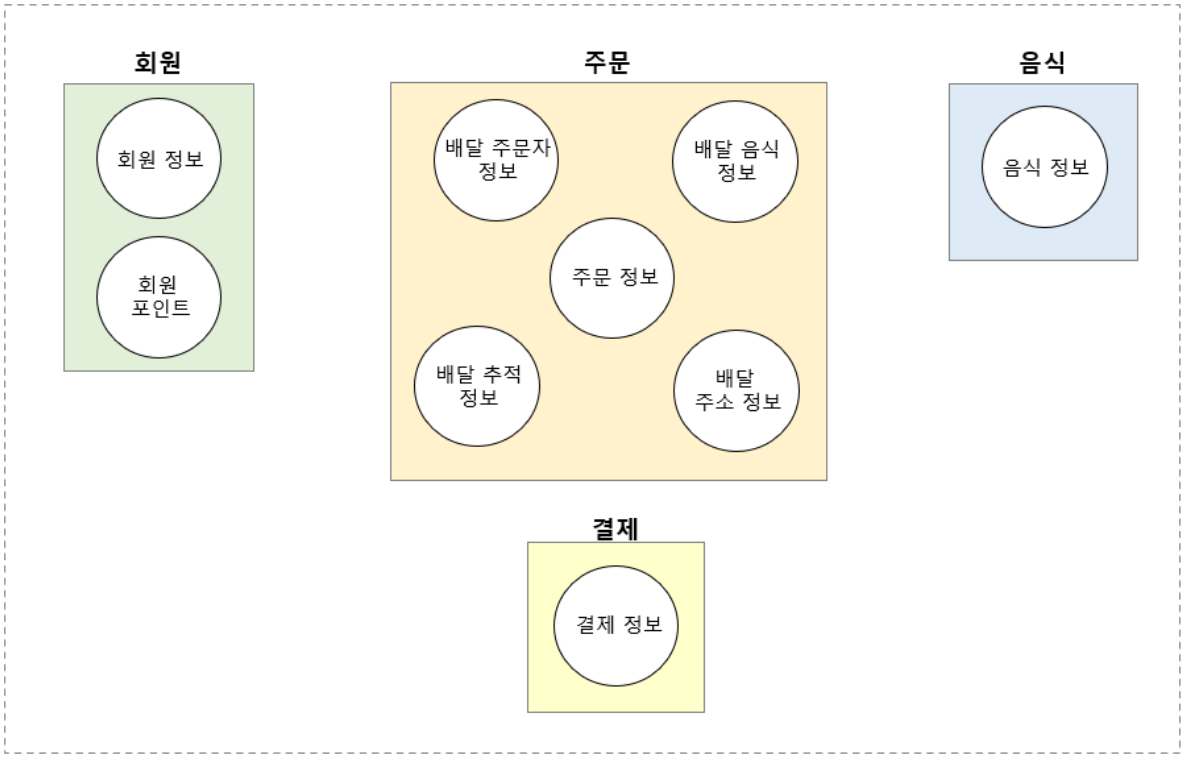
✅ 애그리거트(Aggregate)란?

비슷한 업무 도메인들의 묶음

1️⃣ 배달 주문 앱의 도메인 모델 예



2️⃣ 배달 주문 앱 도메인에서의 애그리거트(Aggregate)



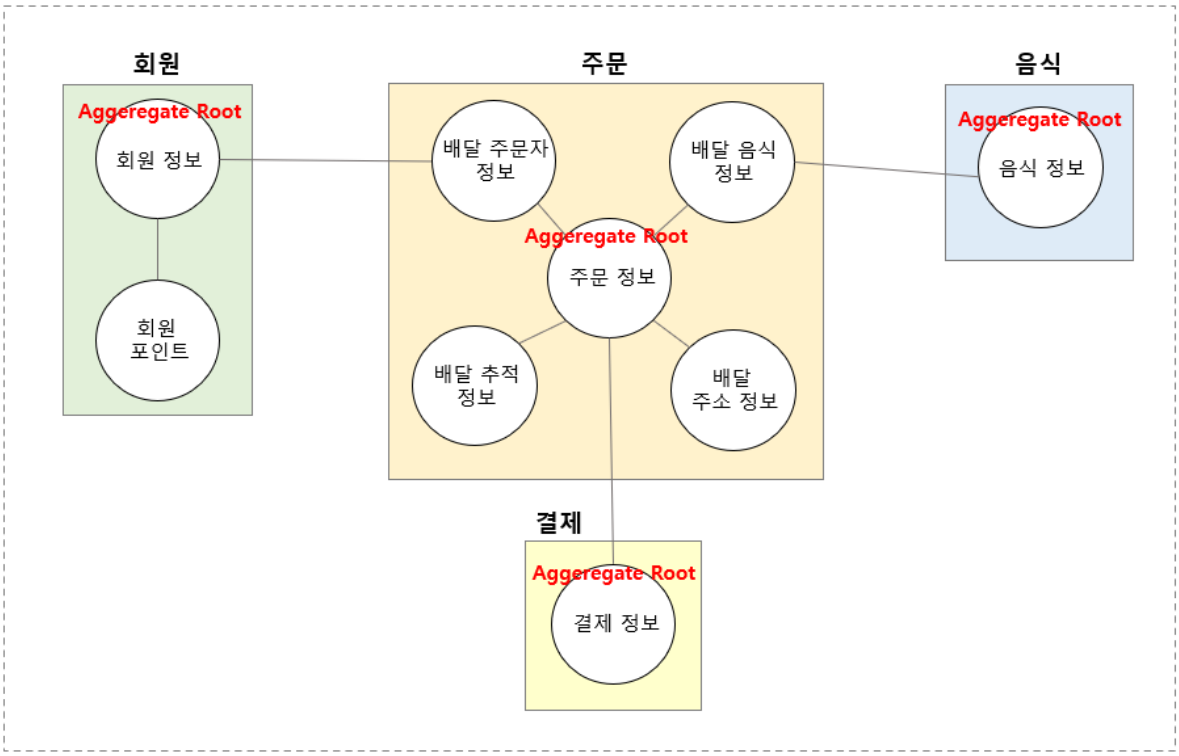
3️⃣ 애그리거트 루트(Aggregate Root)

- **하나의 애그리거트를 대표하는 도메인**
- ⭐ **애그리거트 루트를 통해서만 나머지 엔티티의 상태를 변경하도록 하나?**
  - 도메인 규칙에 대한 일관성을 유지한다.
  - 도메인 규칙을 준수함으로써 데이터 상태에 대한 정합성이 유지된다.

4️⃣ 배달 주문 앱 도메인에서의 애그리거트 루트(Aggregate Root)

- 애그리거트에서 대장 격인 도메인
- 다른 도메인과 직간접적으로 연결되는 도메인

- 데이터베이스의 테이블 간 관계에서는 부모 테이블이 애그리거트 루트, 자식 테이블은 애그리거트 루트가 아닌 다른 도메인이 된다.



💡 애그리거트 루트(Aggregate Root)의 핵심 역할

- 애그리거트의 일관성이 깨지지 않도록 하는 것.
  - DDD에서는 애그리거트 루트를 통해서만 나머지 도메인에 접근할 수 있도록 한다.
    - 즉, 애그리거트 루트에서 나머지 도메인의 상태를 바꿔도 되는지에 대한 검증 과정을 일관되게 거칠 수 있게 한다.
      - 예)
        - 주문이라는 애그리거트 루트를 거치지 않고, 배달 주소지 정보를 마음대로 바꾸는 행위
        - 주문한 음식이 이미 **배송중인 상태일 경우에는 배달 주소를 바꿀 수 있으면 안된다**(**도메인 규칙**).

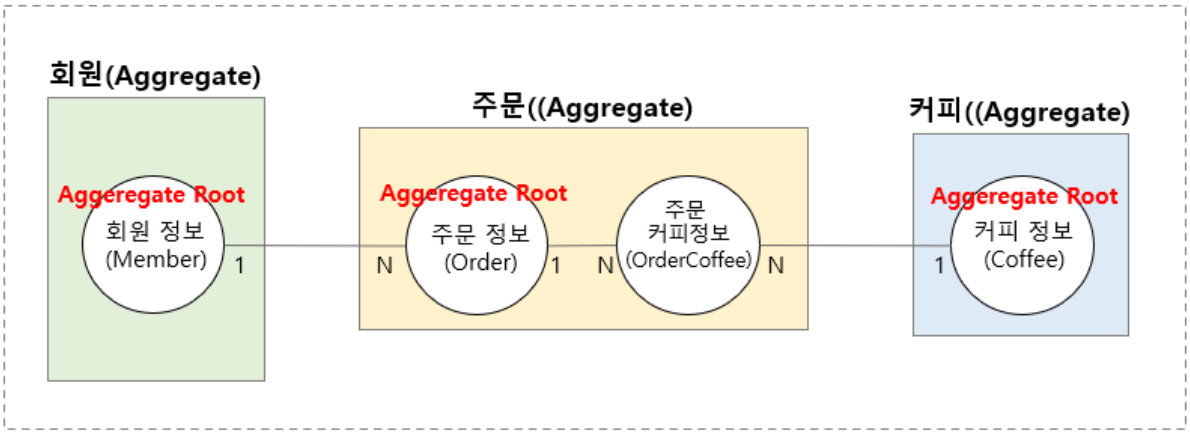
    - 애그리거트 루트인 주문 도메인에서 이런 규칙에 대한 검증을 담당한다.

💡 애그리거트를 잘 나누는 방법에 대한 팁

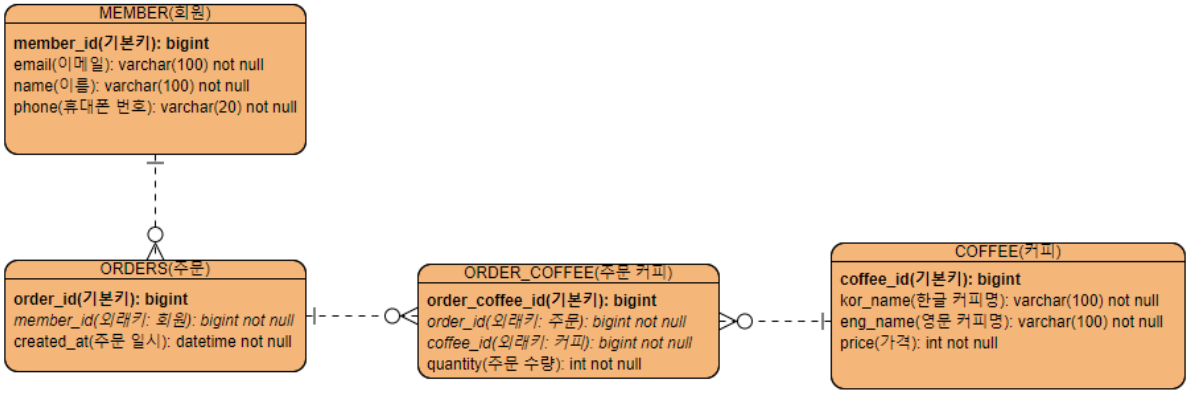
- **같은 애그리거트에 포함되는 도메인은 일반적으로 대부분 다 함께 생성되고, 함께 제거된다.**
    - 따라서 함께 생성되어야 하는 도메인들은 같은 애그리거트에 속할 가능성이 높다.
- **도메인의 변경 주체가 다르다면 각각 다른 애그리거트에 속한다.**
    - 예)
        - **음식**과 음식에 대한 **리뷰**의 관계
            - 음식 정보나 상태는 식당에서 관리
            - 리뷰의 상태는 고객이 관리

✅ 커피 주문 샘플 애플리케이션 테이블 및 도메인 엔티티 설계

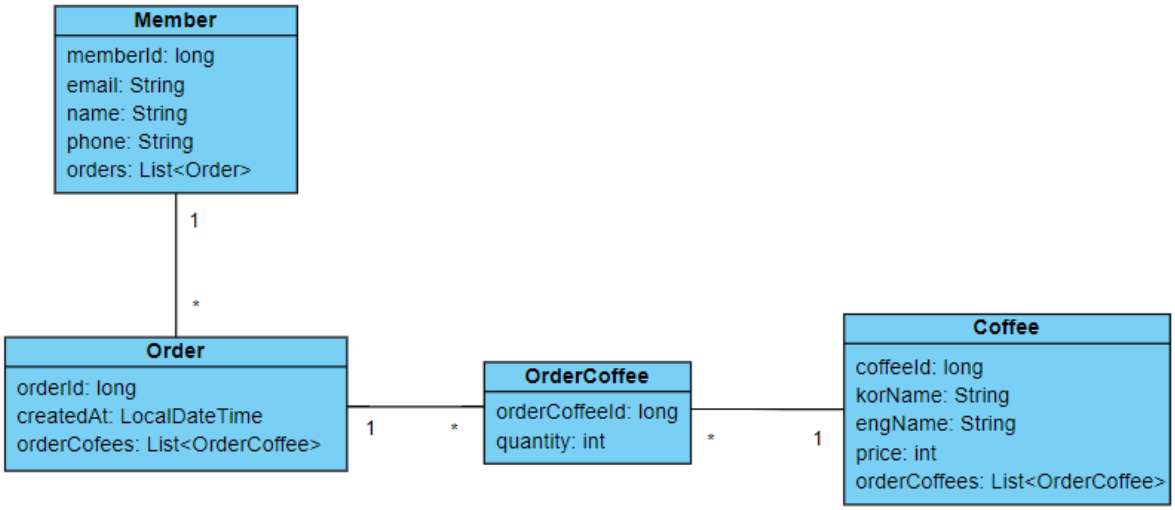
1️⃣ 커피 주문 샘플 애플리케이션의 애그리거트 루트(Aggregate Root) 찾기



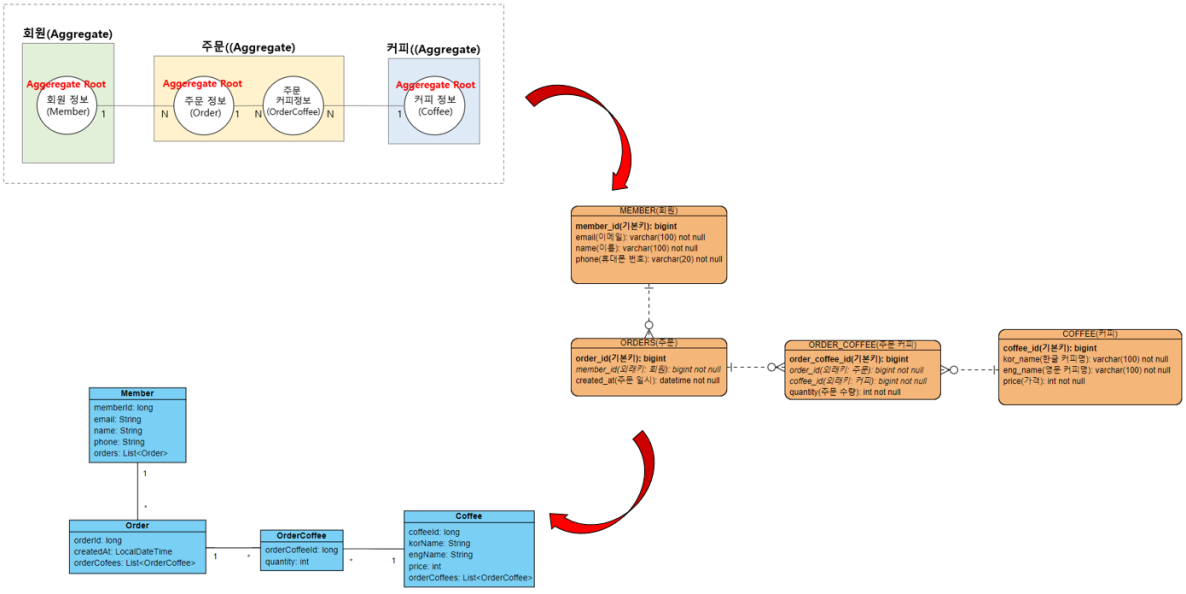
2️⃣ 테이블 설계



3️⃣ 도메인 엔티티 설계(객체 관점에서의 일반적인 엔티티 관계, DDD 적용 전)



4️⃣ 단계별 엔티티 설계 완성



설계 끝났으니 이제 구현입니다.

✅ 도메인 엔티티 클래스 연관관계 매핑

1️⃣ Member와 Order의 관계

- 1 대 N의 관계
- 둘 다 Aggregate Root
  - **ID 참조**

2️⃣Order와 Coffee의 관계

- N 대 N의 관계
- 1 대 N, N 대 1의 관계로 재 설계
- ⭐ **두 가지 관점에서 생각해야 한다.**
  - CoffeeRef(ORDER_COFFEE)는 Order와 Coffee 사이에서 **AggregateRef 같은 역할**을 한다.
    - Order가 Coffee를 참조하기 위해 coffeeId를 가지는 참조 클래스의 역할을 한다.
  - **동일한 Aggregate 내에서 1 대 N의 관계**도 생각해야 됨.

✅ 서비스, 리포지토리 구현

1️⃣ 리포지토리(Repository) 인터페이스

- **쿼리 메서드 작성 방법**
- **네이티브 쿼리 작성 방법**

2️⃣ 서비스 클래스

- 등록, 수정 시 verifyxxxx() 부분 설명
- Stream API 사용 설명
- Optional 사용 설명
  - `Optional.ofNullable()`
  - `Optional.orElseThrow()`

3️⃣ 기타 주문 기능 수정으로 변경된 클래스

- 주문에 대한 설계가 대폭 수정되었으므로..

✔️ OrderController

✔️ OrderPostDto

✔️ OrderCoffeeDto

✔️ OrderCoffeeResponseDto

✔️ OrderResponseDto

✔️ OrderMapper

✔️ ExceptionCode