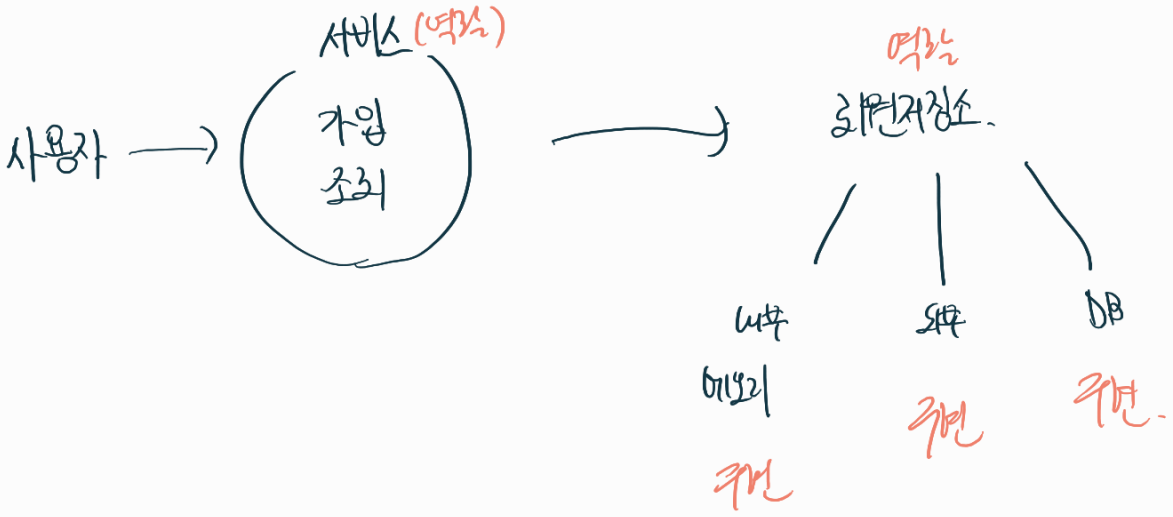
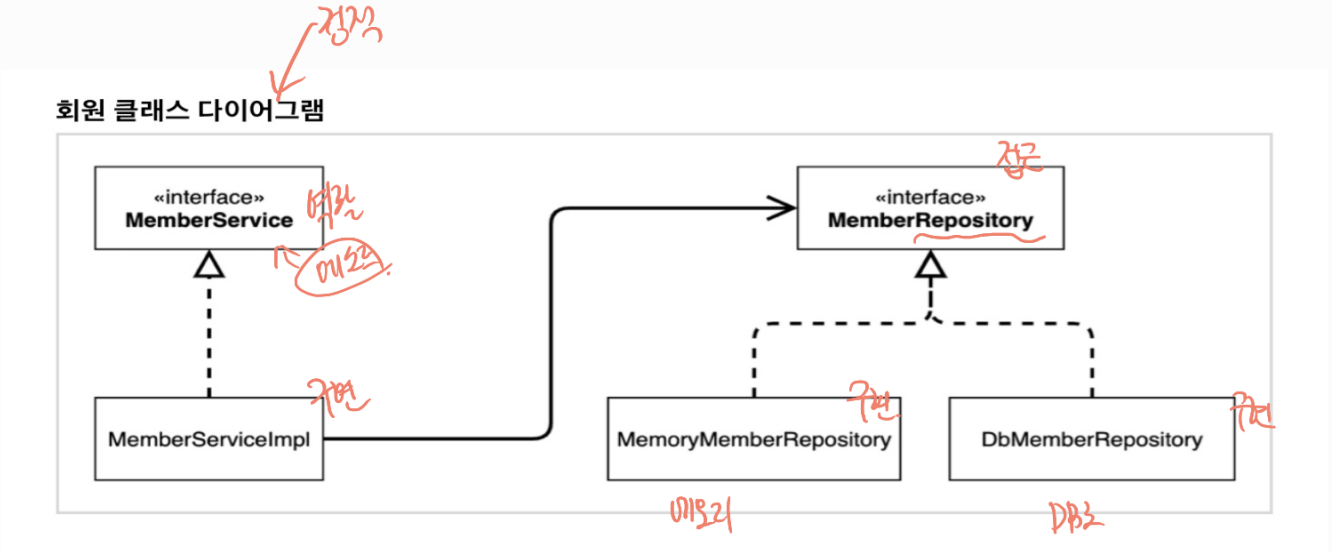
요구사항설계

- 회원
  - 회원을 가입하고 조회할 수 있다.
  - 회원은 일반과 VIP 두 가지 등급이 있다.
  - 회원 데이터는 자체 DB를 구축할 수 있고, 외부 시스템과 연동할 수 있다. (미확정)
- 주문과 할인 정책
  - 회원은 상품을 주문할 수 있다.
  - 회원 등급에 따라 할인 정책을 적용할 수 있다.
  - 할인 정책은 모든 VIP는 1000원을 할인해주는 고정 금액 할인을 적용해달라. (나중에 변경 될 수 있다.)
  - 할인 정책은 변경 가능성이 높다. 회사의 기본 할인 정책을 아직 정하지 못했고, 오픈 직전까지 고민을 미루고 싶다. 최악의 경우 할인을 적용하지 않을 수 도 있다. (미확정)

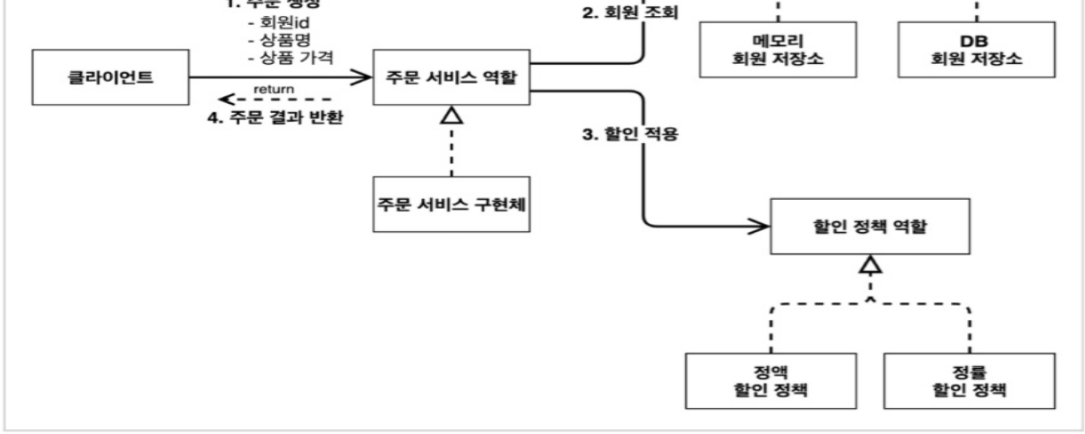
역할과 구현을 나누자.





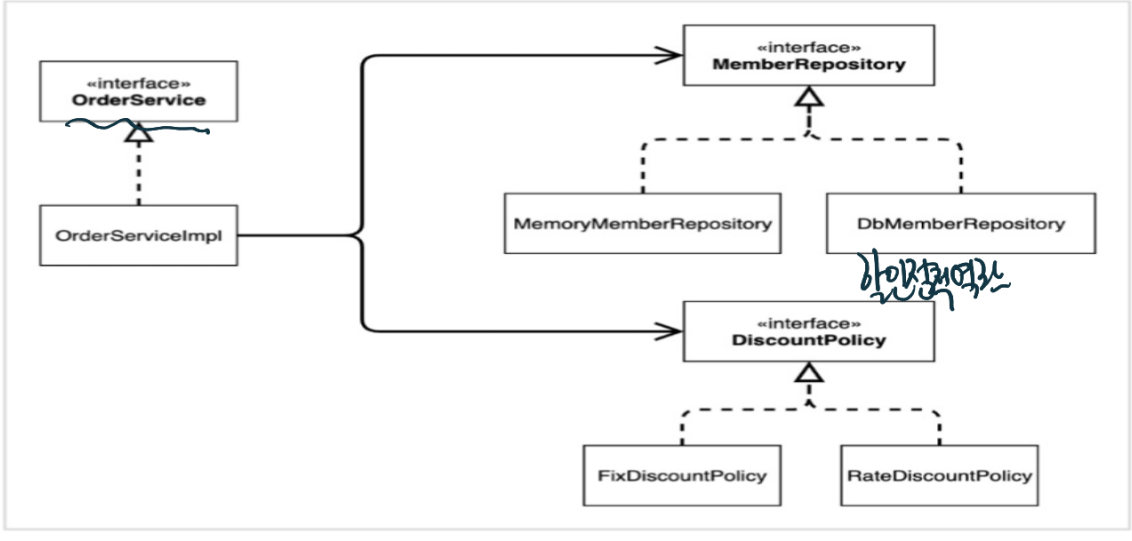
객체 다이어그램 (동적)

Client → Service → Repository



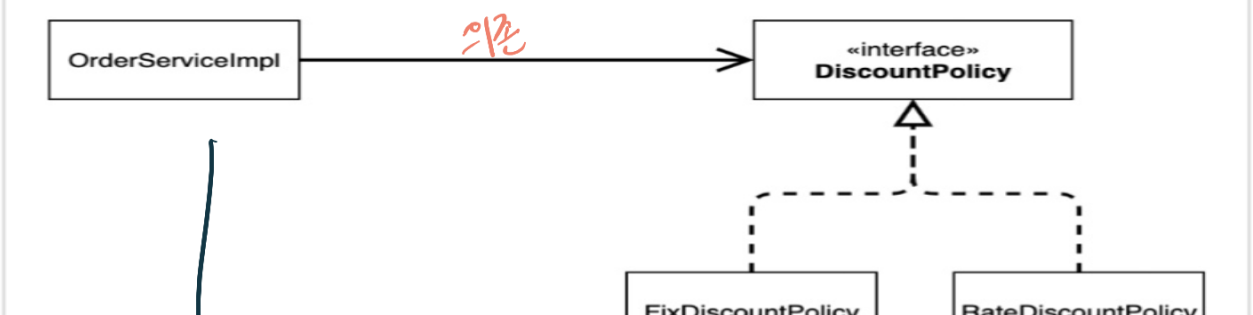
역할과 구현을 분리해서 자유롭게 구현 객체를 조립할 수 있게 설계했다. 덕분에 회원 저장소는 물론이고, 할인 정책도 유연하게 변경할 수 있다.

주문 도메인 클래스 다이어그램



주문 도메인 객체 다이어그램1

RateDiscountPolicy 추가



DIP도 위반함 정책이나 고정 클래스에 의해서도 의존함으로.

역할에 의존 O

구현에는 의존 X

인터페이스에만 의존하도록 코드 변경

```
public class OrderServiceImpl implements OrderService {
    //private final DiscountPolicy discountPolicy = new RateDiscountPolicy();
    private DiscountPolicy discountPolicy;
}
```

ㄴ> 구현체가 없음 → Null Point Exception!

⇒ How?

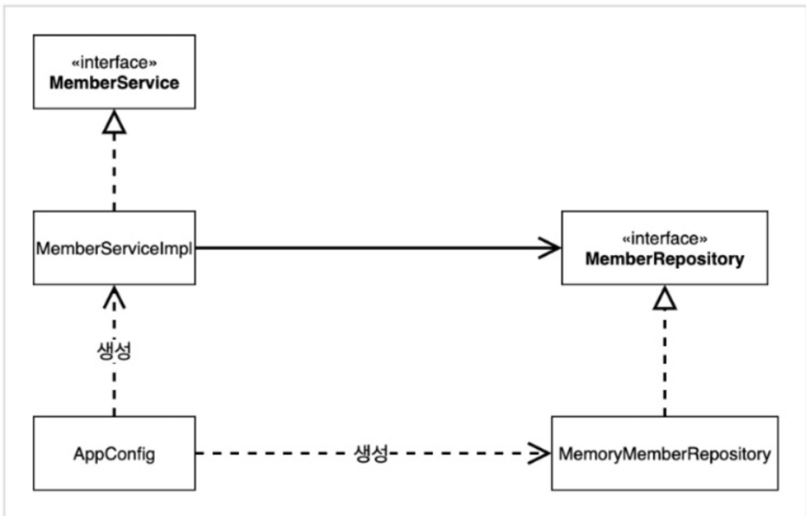
누군가 OrderServiceImpl에 DiscountPolicy의 객체를 생성하고 주입해 주어야한다.

→ 책임과 역할이 들어나야함.

AppConfig 등장

구현객체를 생성하고 연결하는 별도의 클래스!

그림 - 클래스 다이어그램

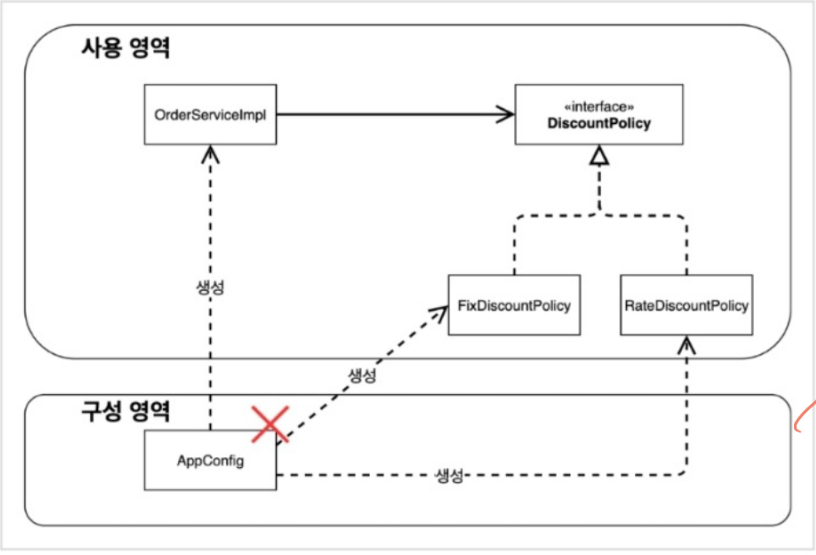


- 객체의 생성과 연결은 `AppConfig` 가 담당한다.
- **DIP 완성:** `MemberServiceImpl` 은 `MemberRepository` 인 추상에만 의존하면 된다. 이제 구체 클래스를 몰라도 된다.

- **관심사의 분리:** 객체를 생성하고 연결하는 역할과 실행하는 역할이 명확히 분리되었다.

그림 - 회원 객체 인스턴스 다이어그램

그림 - 할인 정책의 변경



AppConfig가 생성
이제 사용영역의
코드변경없이 구성역할
만의 책임지중.