

# Spring Study - 스프링 핵심 원리 이해

스프링 핵심 원리 이해 2/3

# Spring이 해주는 것을 알아보기 전에 직접 구축해보자

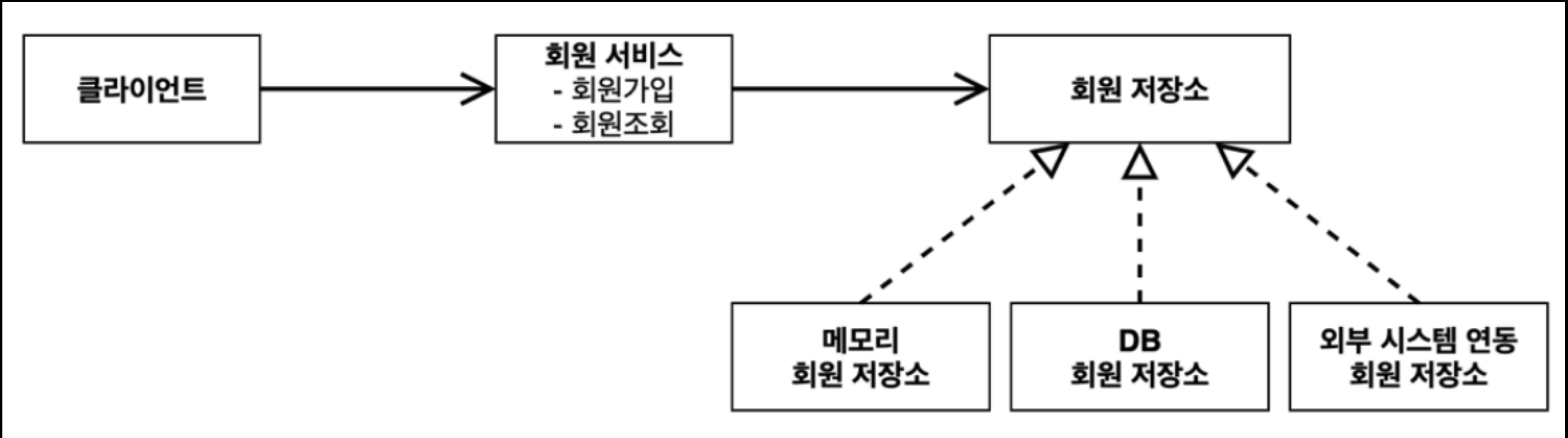
## 개발 전에 계획을 먼저 해보자

도메인 협력 관계

클래스 다이어그램

객체 다이어그램

계획을 통해 도메인 간의 관계를 정리



# Spring이 해주는 것을 알아보기 전에 직접 구축해보자

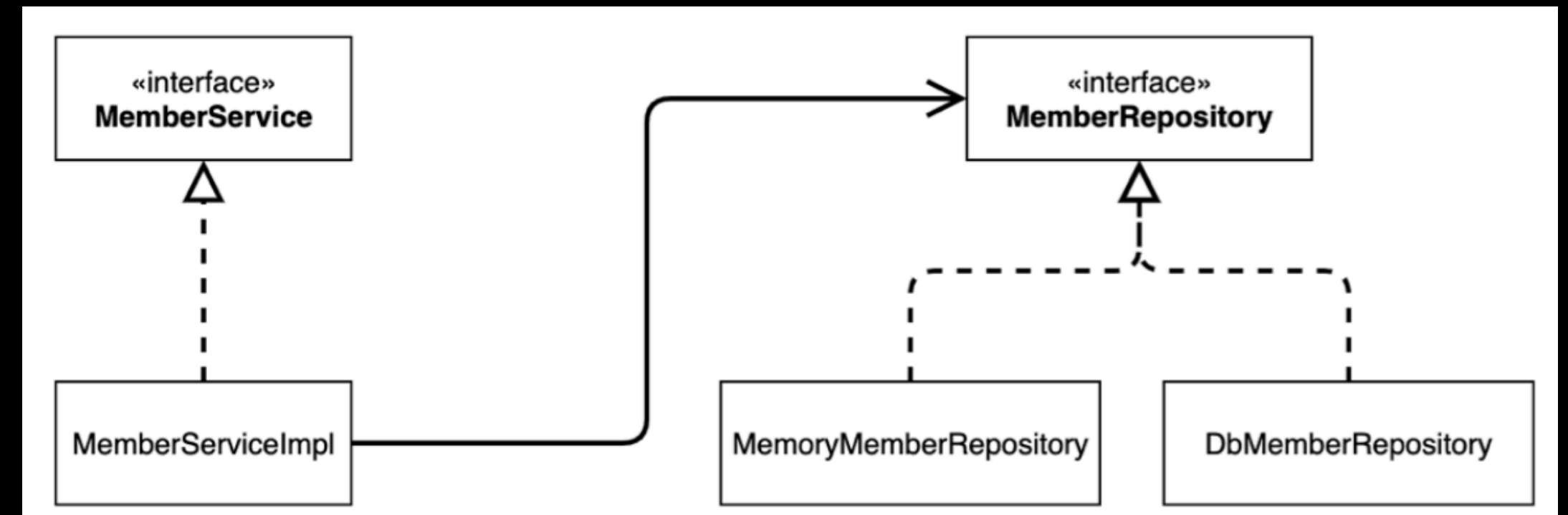
## 개발 전에 계획을 먼저 해보자

도메인 협력 관계

클래스 다이어그램

객체 다이어그램

코드 레벨로 도메인을 가져와 클래스로 정리



# Spring이 해주는 것을 알아보기 전에 직접 구축해보자

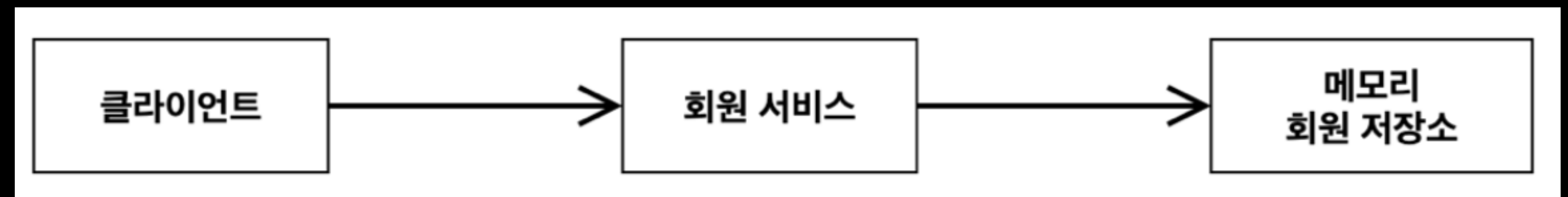
## 개발 전에 계획을 먼저 해보자

도메인 협력 관계

클래스 다이어그램

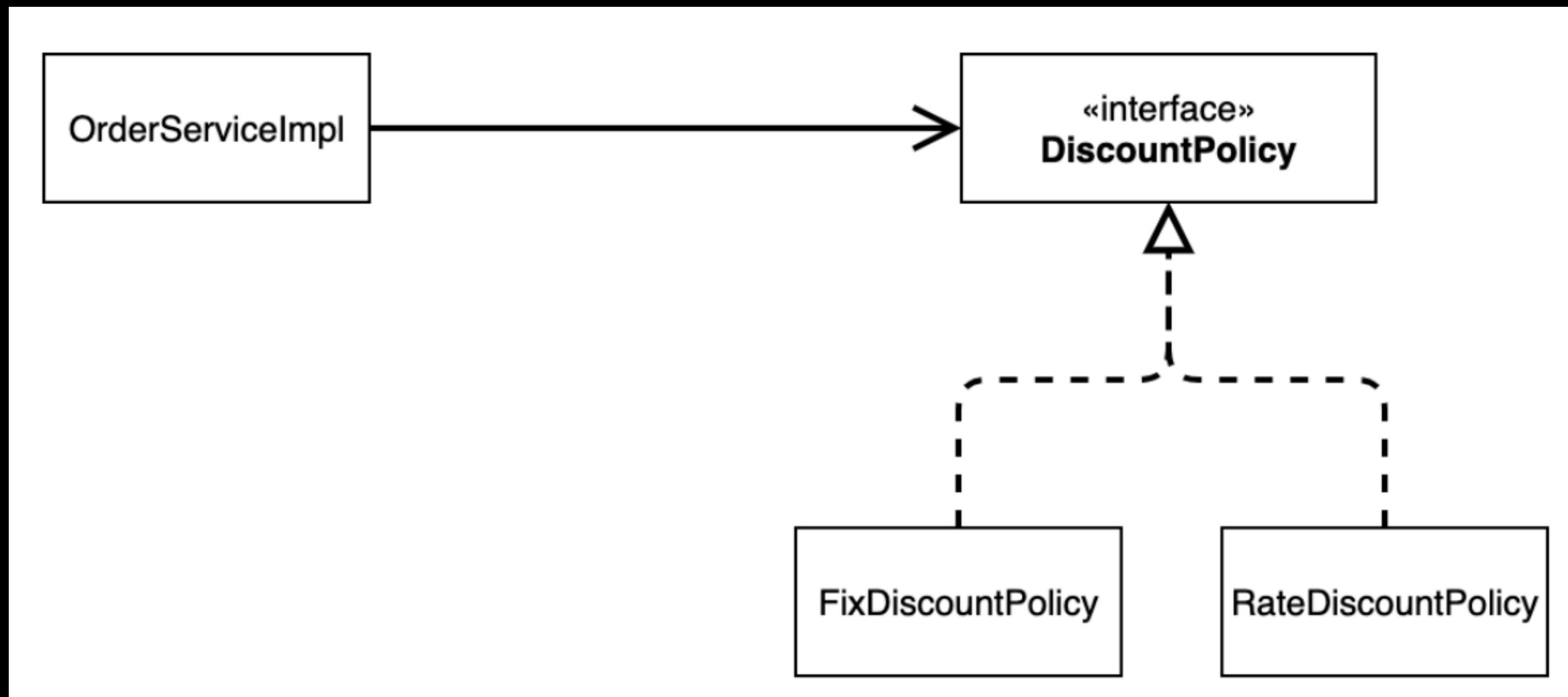
객체 다이어그램

객체간의 참조 관계를 정리



# Spring이 해주는 것을 알아보기 전에 직접 구축해보자

## 할인 정책 - 클래스 다이어그램



```
1
2
3 public class OrderServiceImpl implements OrderService {
4     // private final DiscountPolicy discountPolicy = new FixDiscountPolicy();
5     private final DiscountPolicy discountPolicy = new RateDiscountPolicy();
6 }
```

Line 5, Column 59      UTF-8      Unix      Spaces: 4      Java

역할과 구현 분리

다형성 활용



OCP ? DIP ?

인터페이스/구현체 분리

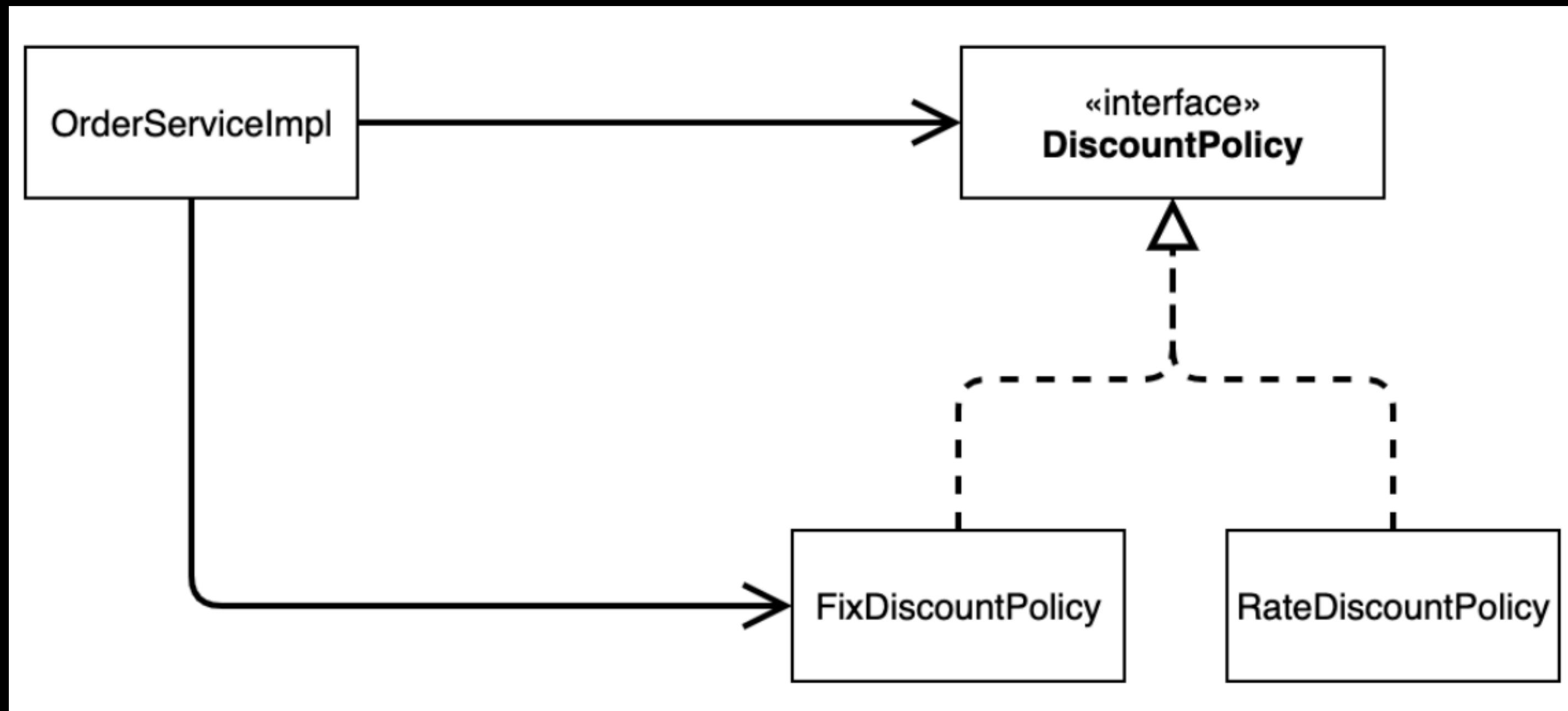
# REMIND

## DIP

의존역전원칙 - “추상화에 의존해야지, 구체화에 의존하면 안된다.”

## OCP

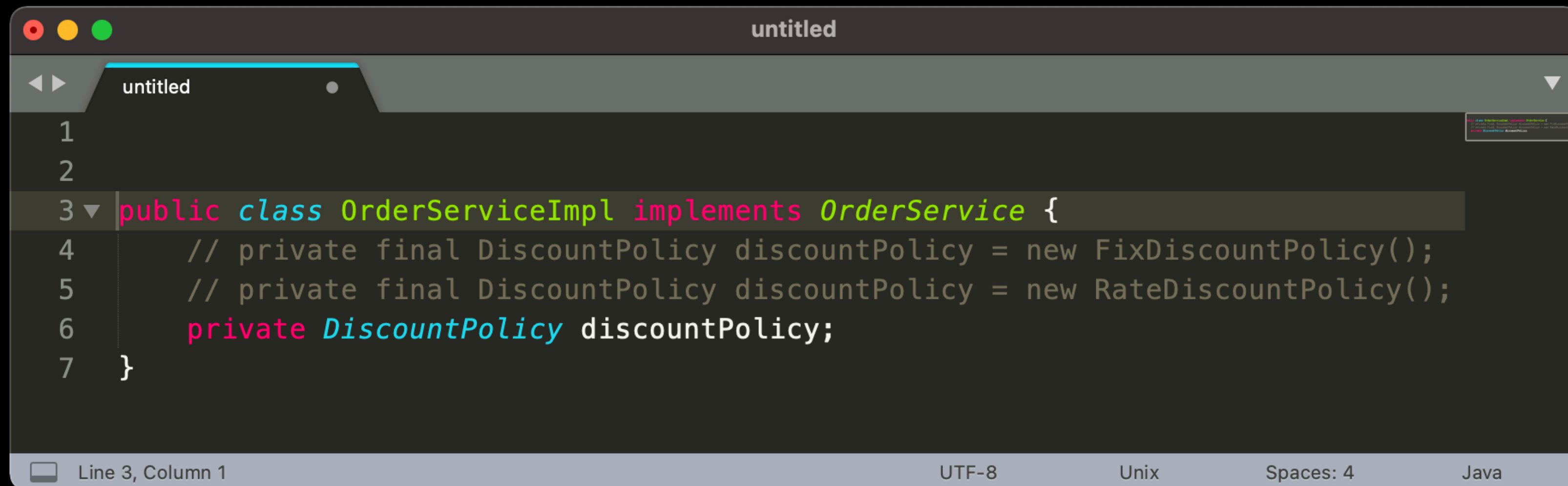
개방폐쇄원칙 - “소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다”



추상 뿐 아니라 구현체에도 의존하고 있다.  
기능을 변경하려면, 클라이언트 코드도 수정해야한다.

# HOW ?

어떻게 하면 추상(인터페이스)에만 의존할 것인가?



The screenshot shows a code editor window titled 'untitled'. The code is as follows:

```
1  
2  
3 public class OrderServiceImpl implements OrderService {  
4     // private final DiscountPolicy discountPolicy = new FixDiscountPolicy();  
5     // private final DiscountPolicy discountPolicy = new RateDiscountPolicy();  
6     private DiscountPolicy discountPolicy;  
7 }
```

The status bar at the bottom indicates 'Line 3, Column 1', 'UTF-8', 'Unix', 'Spaces: 4', and 'Java'.

구현체가 없는데 ? ( NPE 🙌 ? )



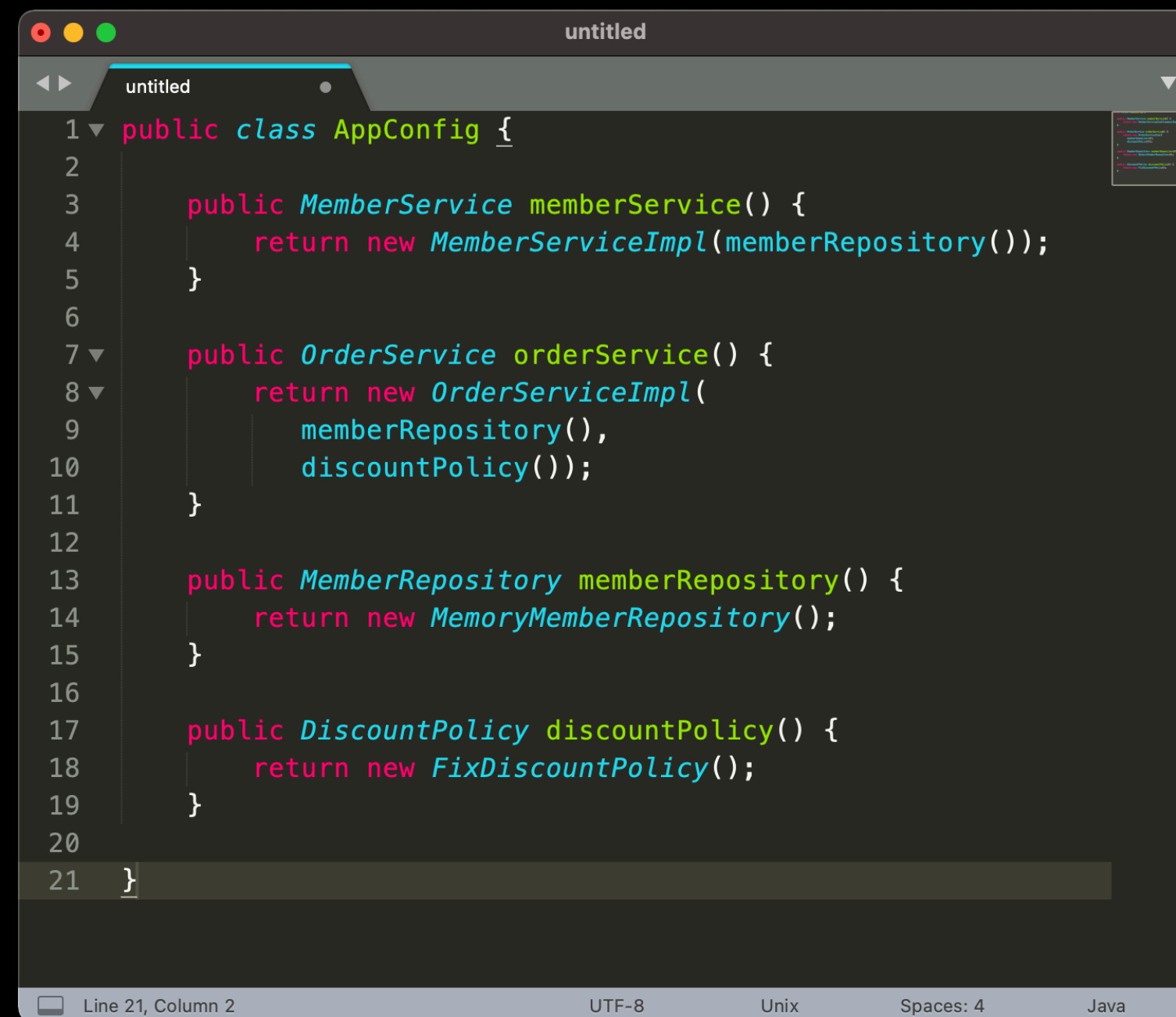
# HOW ?

어떻게 하면 추상(인터페이스)에만 의존할 것인가?

관심사의 분리 - 구현 객체를 생성하고 연결하는 책임을 가지는 클래스

## Configuration

- 구현 객체 생성
- 생성자를 통한 주입



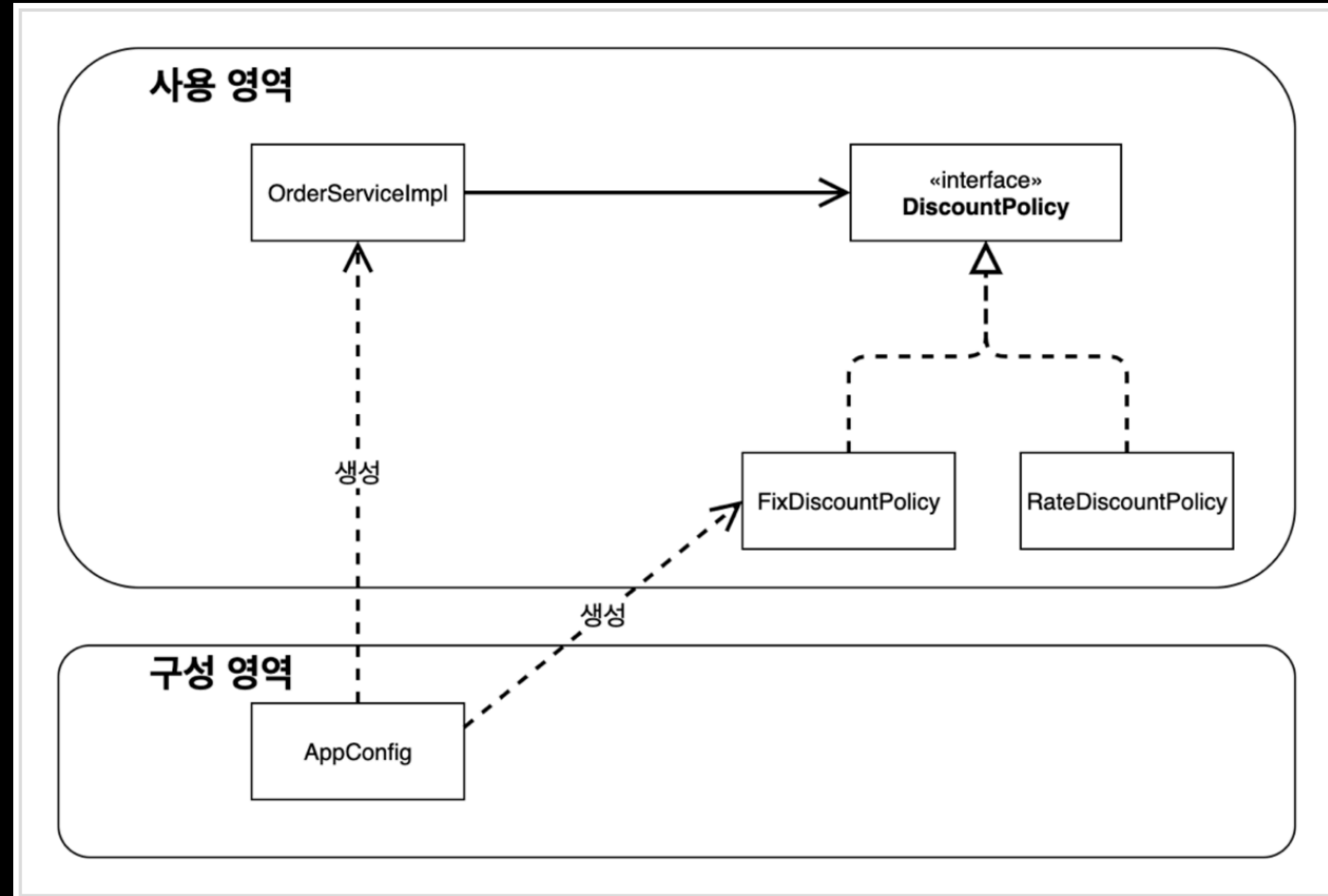
```
1 public class AppConfig {
2
3     public MemberService memberService() {
4         return new MemberServiceImpl(memberRepository());
5     }
6
7     public OrderService orderService() {
8         return new OrderServiceImpl(
9             memberRepository(),
10            discountPolicy());
11     }
12
13     public MemberRepository memberRepository() {
14         return new MemoryMemberRepository();
15     }
16
17     public DiscountPolicy discountPolicy() {
18         return new FixDiscountPolicy();
19     }
20
21 }
```



Dependency Injection

# HOW ?

어떻게 하면 추상(인터페이스)에만 의존할 것인가?



# DI ? 이거 어떻게 관리해요 ?

코드가 많아짐에 따라

- 필요한 기능
- 기능을 구현한 클래스
- 그 클래스를 의존하는 클래스
- 또 그 클래스를 의존하는 클래스
- 또 또 그 클래스를 의존하는 클래스



@Configuration, @Bean

=> **ApplicationContext**

=> **Spring container**

-> **Spring Bean으로 관리**