# 스프링 핵심 원리 이해2 - 객체 지향

- AppConfig : 배우를 교체하는 느낌
- 이제 할인 정책을 변경해도, 애플리케이션 구성 역할을 담당하는 AppConfig만 변경하면 됨.
- 클라이언트 코드인 OrderServiceImpl을 포험해서 **사용 영역**의 어떤 코드도 변경할 필요 X
- **구성 영역**의 코드만 변경하면 됨.

## 전체 흐름 정리

### 새로운 할인 정책 적용의 문제점

- 새로운 할인 정책을 적용하려고 하니 “클라이언트 코드”인 주문 서비스 구현체도 함께 변경해야 함. (OCP 위반)
- 주문 서비스 클라이언트가 인터페이스인 ‘DiscountPolicy’ 뿐만 아니라, 구체 클래스인 ‘FixDiscountPolicy’도 함께 의존. (DIP 위반)

### 관심사의 분리

- 공연 기획자인 **AppConfig**가 등장! (구성 : configuration)
- AppConfig는 애플리케이션의 전체 동작 방식을 구성하기 위해 “구현 객체를 생성”하고 “연결”하는 책임
-

추가 작성 필요

---

# IoC, DI, 그리고 컨테이너

## 제어의 역전

- 내가 호출하는 게 아니라 프레임워크가 대신 호출해 주는 것
- 말 그대로 제어권이 뒤바뀐다는 뜻
- 프로그램 제어 흐름을 직접 제어하는 것이 아니라 외부에서 관리하는 것을 의미

  ex) 프로그램에 대한 제어권은 AppConfig가 가지고 있으며, 구현체는 묵묵히 자신의 로직만 실행

## 프레임워크 vs 라이브러리

- 프레임워크는 내가 작성한 코드를 제어하고 대신 실행 (ex. JUnit)
- 반면 내가 작성한 코드가 직접 제어의 흐름을 담당한다면 그것은 라이브러리

## 의존관계 주입 DI

---

# 스프링으로 전환하기

@Configuration 설정정보

@Bean 각 메서드에 달아 줌 → 그럼 스프링 컨테이너에 등록

```
public class MemberApp {

    public static void main(String[] args) {
//        AppConfig appConfig = new AppConfig();
//        MemberService memberService = appConfig.memberServi

        ApplicationContext applicationContext = new Annotatio
    }
}
```

- AppConfig에 있는 환경 정보를 보고 골뱅이 붙은 애들을 스프링컨테이너에서 관리하겠다는 의미.

```
MemberService memberService = applicationContext.getBean("mem
```

- memberService라는 객체를 찾을 거고, 반환 타입은 MemberService