Spring을 통한 객체 지향 원리 적용 실습

미션	[백엔드] 스프링 핵심 원리 - 기본
상태	완료
설명	다형성을 활용한 서비스 구현 예제를 만들고, 스프링 컨테이너가 다형성을 지원하는 방식을 확인합니다. 예제에서는 여러 종류의 결제 방식을 다형성을 통해 구현합니다. 구현된 코드 파일과 다형성 활용 방법 설명을 문서로 정리하여 결과물로 제출합니다.

DiscountPolicy.java

```
public interface DiscountPolicy {
    int discount(Member member, int price);
}
```

FixDiscountPolicy.java

```
public class FixDiscountPolicy implements DiscountPolicy {

    private int discountFixAmount = 1000;

    @Override
    public int discount(Member member, int price) {
        if (member.getGrade() == Grade.VIP) {
            return discountFixAmount;
        }
        else {
            return 0;
        }
```

```
    }
}
```

RateDiscountPolicy.java

```
public class RateDiscountPolicy implements DiscountPolicy {

    private int discountRate = 10;

    @Override
    public int discount(Member member, int price) {
        if (member.getGrade() == Grade.VIP) {
            return price * discountRate / 100;
        }
        else {
            return 0;
        }
    }

}
```

AppConfig.java

```
@Configuration
public class AppConfig {

    @Bean
    public MemberService memberService() {
        return new MemberServiceImpl(memoryMemberRepository());
    }

    @Bean
    public OrderService orderService() {
        return new OrderServiceImpl(
            memoryMemberRepository(),
```

```
            discountPolicy()
        );
    }

    @Bean
    public MemoryMemberRepository memoryMemberRepository() {
        return new MemoryMemberRepository();
    }

    @Bean
    public DiscountPolicy discountPolicy() {
        // return new FixDiscountPolicy();
        return new RateDiscountPolicy();
    }
}
```

MemberApp.java

```
public class MemberApp {
    public static void main(String[] args) {
            ...

        ApplicationContext applicationContext = new AnnotationConfigApplica
tionContext(AppConfig.class);
        MemberService memberService = applicationContext.getBean(Memb
erService.class);

        ...
    }
}
```

스프링 컨테이너 동작

1. `DiscountPolicy` 인터페이스를 통해 구현체(`FixDiscountPolicy`, `RateDiscountPolicy`)를 추상화하여 사용한다. 따라서 클라이언트 코드는 구체 클래스가 아닌 인터페이스에만 의존한다.

(DI)

2. `AppConfig`가 실제 구현체 생성과 주입을 담당한다.
    - `FixDiscountPolicy` → `RateDiscountPolicy` 변경 시 클라이언트 코드 수정 없음
3. 스프링 컨테이너의 IoC

```
ApplicationContext applicationContext = new AnnotationConfigApplication
Context(AppConfig.class);
MemberService memberService = applicationContext.getBean(MemberSer
vice.class);
```

객체 생성 및 의존 관계 설정을 스프링 컨테이너가 대신 한다.

4. `DiscountPolicy` 타입으로 요청이 들어오면 등록된 빈 `RateDiscountPolicy`를 반환한다