

의존관계 자동 주입 방식 실습

| | |
|------|---|
| ◎ 미션 | [백엔드] 스프링 핵심 원리 - 기본 |
| ※ 상태 | 완료 |
| ≡ 설명 | @Autowired, @Qualifier, @Primary를 사용하여 의존관계 자동 주입 방식을 각각 적용해 보고, 각 방식의 차이점을 정리합니다. 코드 스크린샷과 주입 방식 비교 표를 PDF 문서로 만들어 결과물로 제출합니다. |

The screenshot shows a Java code editor with the following code:

```
@Component
@RequiredArgsConstructor
public class OrderServiceImpl implements OrderService {

    private final MemberRepository memberRepository;

    @Autowired
    private DiscountPolicy discountPolicy;

    @Override
    public Order createOrder(Long memberId, String itemName, int itemPrice) {
        Member member = memberRepository.findById(memberId);
        int discountPrice = discountPolicy.discount(itemName);

        return new Order(memberId, itemName, itemPrice - discountPrice);
    }
}
```

A tooltip is displayed over the `@Autowired` annotation on the `discountPolicy` field, stating: "자동 주입을 할 수 없습니다. 'DiscountPolicy' 타입의 bean이 두 개 이상 있습니다." (Automatic injection is not possible. There are two or more beans of type 'DiscountPolicy'.). It also lists the beans: `fixDiscountPolicy` (`FixDiscountPolicy.java`) and `rateDiscountPolicy` (`RateDiscountPolicy.java`). Below the tooltip, there are buttons for "한정자 추가" (Add Qualifier), "추가 액션..." (More Actions...), and a file icon.

다음과 같이 하나의 타입에 두 개의 bean이 조회가 될 경우 의존관계를 자동으로 주입하는데 문제가 발생한다.

@Autowired 필드 명 매칭

```
@Autowired  
private DiscountPolicy rateDiscountPolicy;  
  
@Autowired  
private DiscountPolicy fixDiscountPolicy;
```

다음과 같이 필드 명을 빈 이름으로 변경하여 명확하게 명시하도록 한다.

필드 명 매칭은 먼저 타입 매칭을 시도하고 결과에 두 개 이상의 빈이 존재할 경우 필드 명과 일치하는 빈이 주입되도록 추가 동작하는 기능이다.

요약

1. 타입 매칭
2. 타입 매칭의 결과가 2개 이상일 경우, 필드 명으로 빈 매칭

@Qualifier 사용

추가 구분자를 붙여 활용하는 방식. 주입 시 추가적인 방법을 제공하는 것이기 때문에 빈 이름을 변경하는 것은 아니다!

```
@Component & Noh  
@Qualifier("rateDiscountPolicy")  
public class RateDiscountPolicy implements DiscountPolicy {
```

```
@Component & Noh  
@Qualifier("fixDiscountPolicy")  
public class FixDiscountPolicy implements DiscountPolicy {
```

다음과 같이 빈 주입시 `@Qualifier` 를 붙여주고 등록할 이름을 적어준다.

```
@Autowired & Noh *
public OrderServiceImpl(MemberRepository memberRepository, @Qualifier("rateDiscountPolicy") DiscountPolicy discountPolicy) {
    this.memberRepository = memberRepository;
    this.discountPolicy = discountPolicy;
}
```

생성자 자동 주입 시에 다음과 같이 사용할 수 있다.

요약

1. `@Qualifier` 끼리 매칭
2. 빈 이름 매칭

@Primary 사용

우선순위를 지정하는 방법. `@Autowired` 시에 여러 빈이 매칭되면 `@Primary` 가 우선권을 가진다.

```
@Component
@Primary
public class RateDiscountPolicy implements DiscountPolicy {
}
```

`@Primary` 를 통해 `rateDiscountPolicy`가 우선순위를 가지도록 지정하였다.

```

class RateDiscountPolicyTest {
    RateDiscountPolicy discountPolicy = new RateDiscountPolicy(); 2개 사용 위치

    @Test & Noh *
    @DisplayName("VIP는 10%할인이 적용되어야 한다.")
    void vip_o() {
        //given
        Member member = new Member(id: 1L, name: "memberVIP", Grade.VIP);

        //when
        int discount = discountPolicy.discount(member, price: 20000);

        //then
        Assertions.assertThat(discount).isEqualTo(expected: 2000);
    }

    @Test & Noh
    @DisplayName("VIP가 아니면 할인 적용 X.")
    void vip_x() {
        Member member = new Member(id: 1L, name: "memberVIP", Grade.BASIC);
        int discount = discountPolicy.discount(member, price: 10000);
        Assertions.assertThat(discount).isEqualTo(expected: 0);
    }
}

```

실행 RateDiscountPolicyTest ×

2 테스트 통과 총 2개의 테스트, 27ms

/Users/noh/Library/Java/JavaVirtualMachines/ms-17.0.17/Contents/Home/bin/java ...

종료 코드 0(으)로 완료된 프로세스

테스트 해보면 문제없이 동작하는 것을 알 수 있다.

@Qualifier 와 @Primary 활용

메인 DB의 Connection을 획득하는 스프링 빈과, 서브 DB의 Connection을 획득하는 스프링 빈 두 개가 있을때, 우리는 메인 DB의 Connection을 획득하는 빈은 `@Primary` 를 지정하여 우선순위로 가져오고, 서브 DB의 커넥션을 획득할 때에는 `@Qualifier` 를 이용해 명시적으로 획득하도록 코드를 깔끔하게 작성할 수 있다.

우선순위

`@Primary` 는 기본값처럼 동작하고, `@Qualifier` 는 매우 상세히 동작한다. 스프링은 자동보다는 수동이, 넓은 범위보다 좁은 범위의 선택권이 우선 순위가 높다. 따라서 `@Qualifier` 의 우선권이 더 높다.