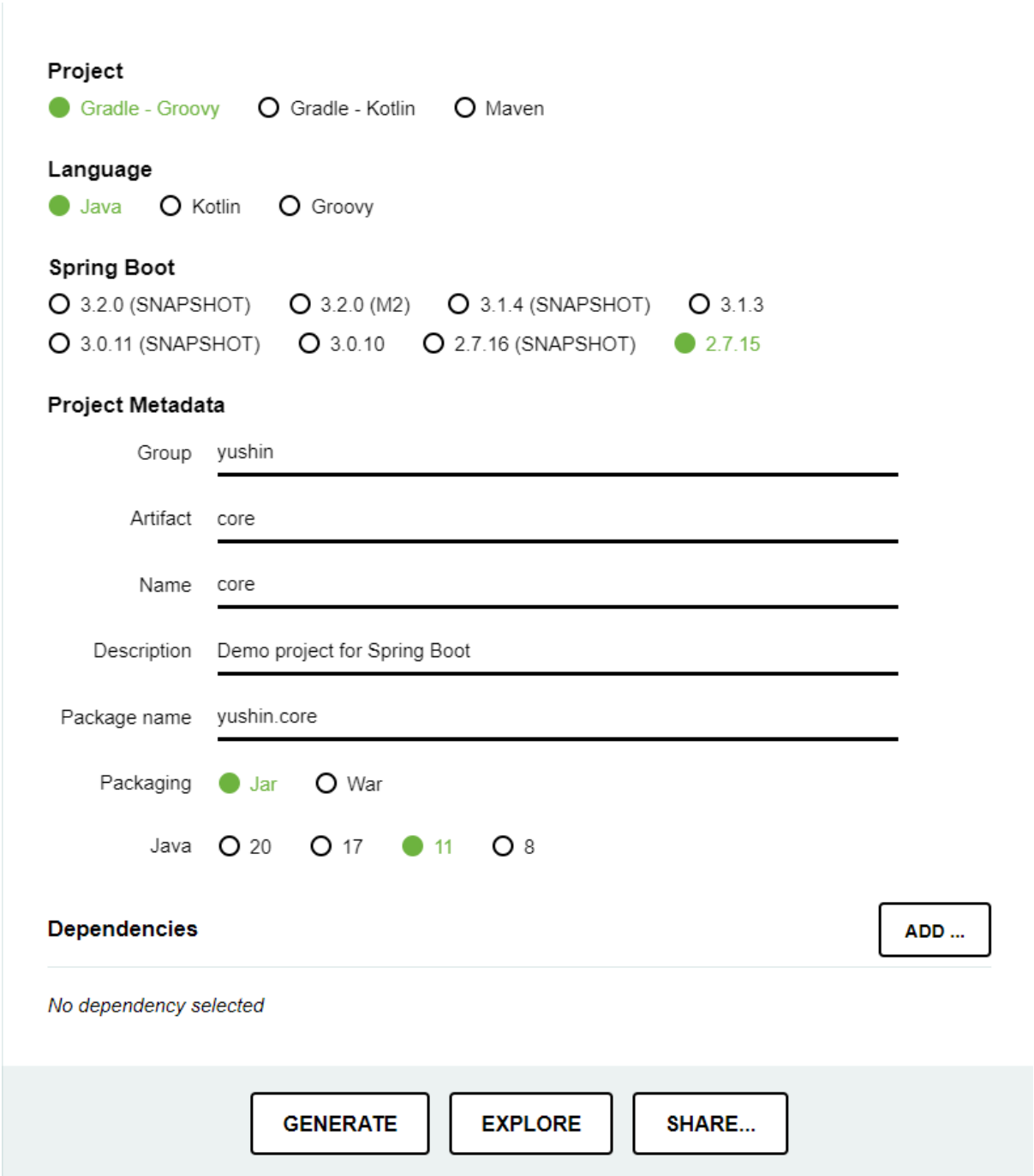
섹션 2 강의 내용 요약

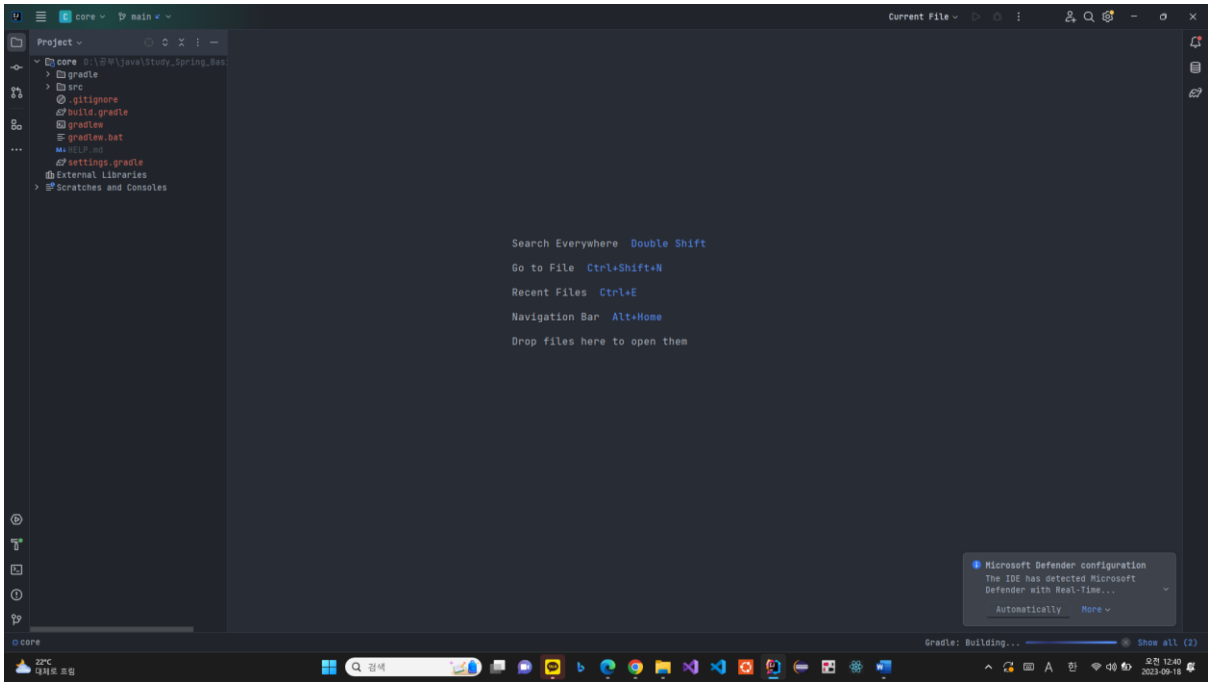
스프링 부트 프로젝트 생성 방법

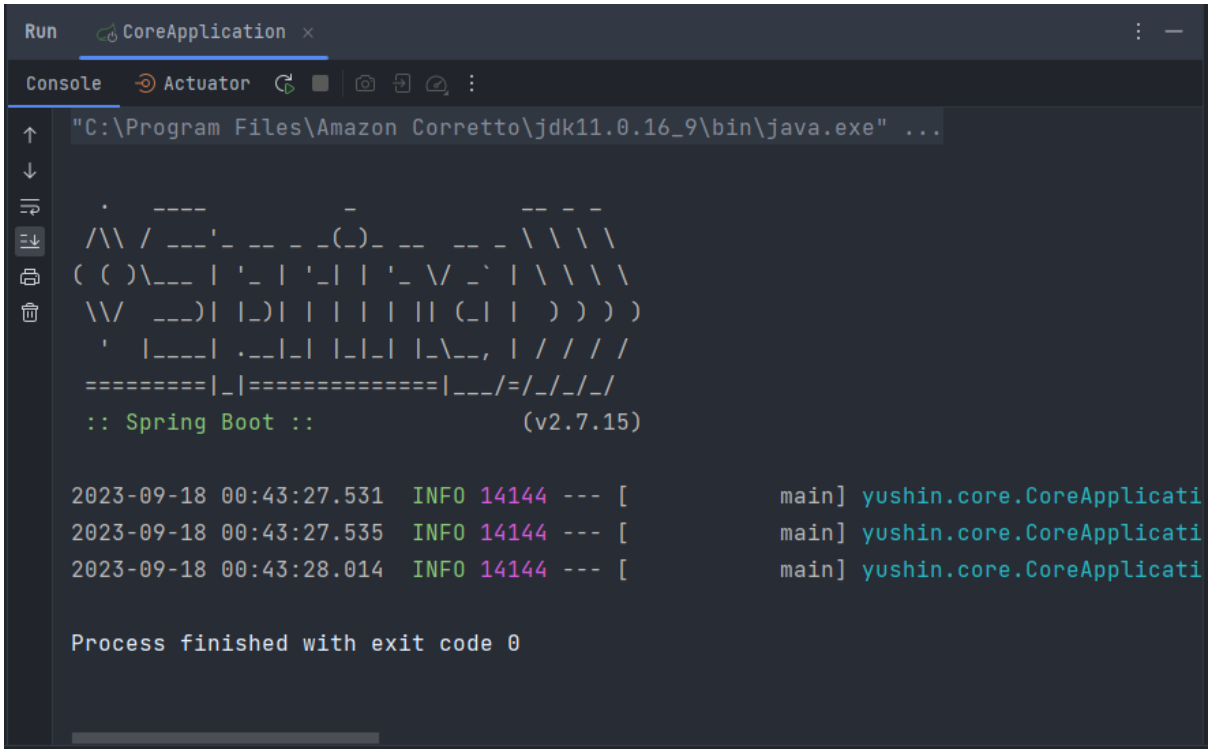
1. 스프링 부트 스타터 사이트로 이동 https://start.spring.io
2. 아래와 같은 옵션으로 프로젝트 생성 및 다운로드



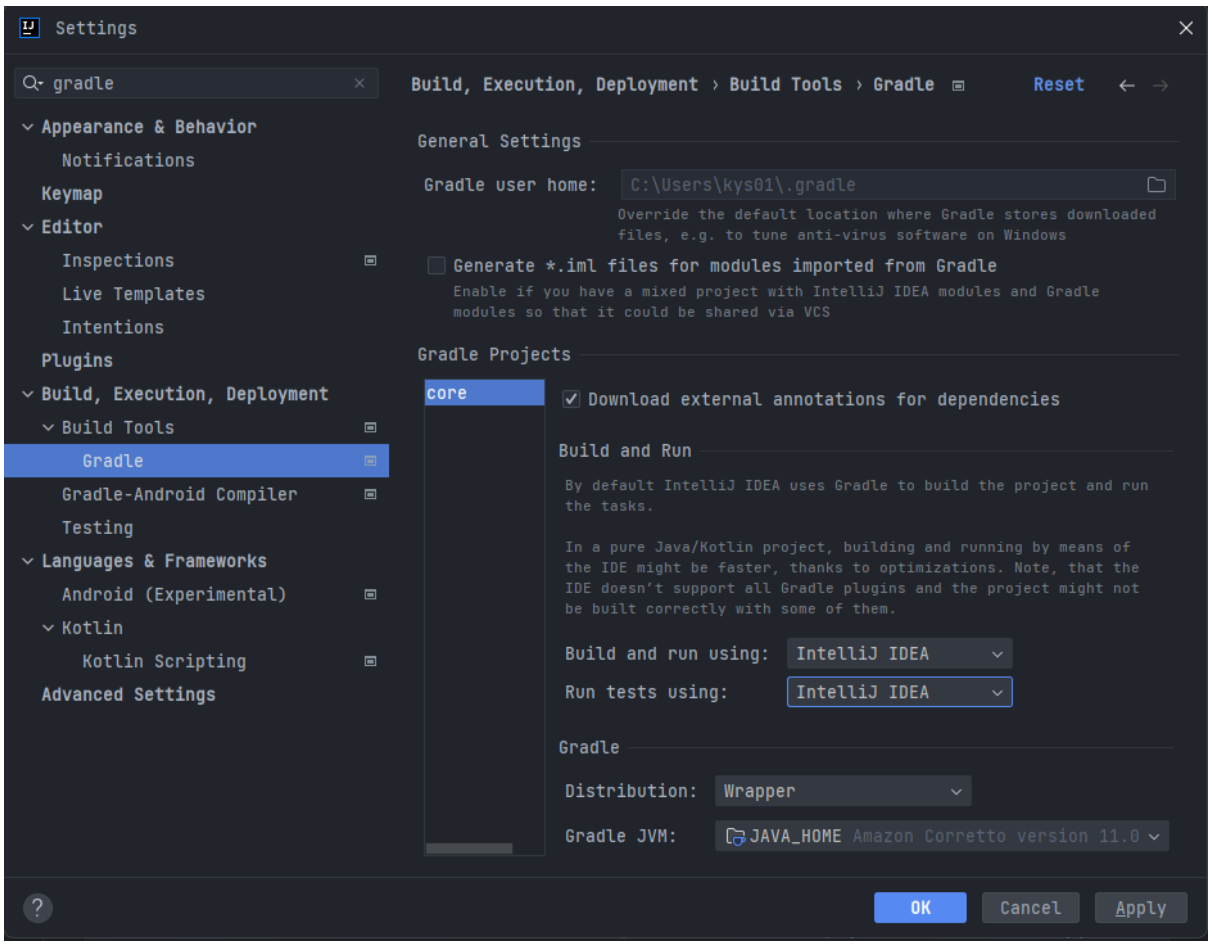
강의 코드가 스프링 부트 버전 2를 기준으로 작성되었기 때문에 2.7.15 버전을 사용한다.

3. IntelliJ에서 프로젝트 불러오기





CoreApplication 실행 시 정상 종료되는 것을 확인한다.



Windows OS 기준 File - Settings에서 gradle을 검색한 후 Build and run using, Run tests using 옵션을 모두 IntelliJ IDEA로 변경한다. -> Gradle을 통해서가 아닌 자바를 통해서 직접 실행하기 때문에 속도가 조금 더 빠르다.

가상의 비즈니스 요구 사항과 설계

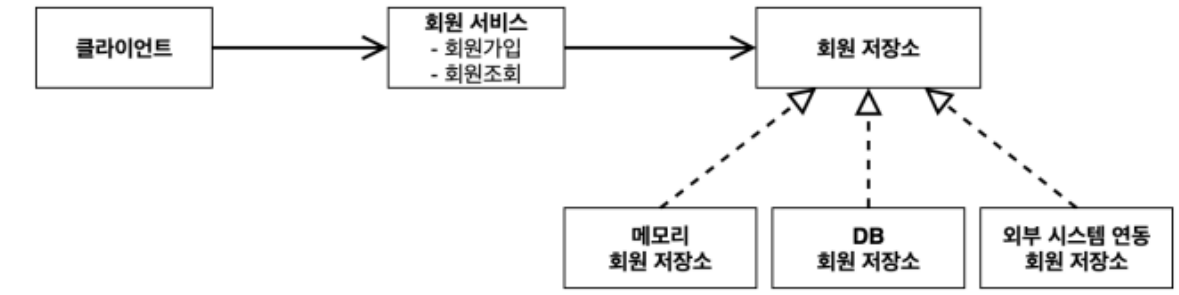
- 회원
  - 회원을 가입하고 조회할 수 있다.
  - 회원은 일반과 VIP 두 가지 등급이 있다.
  - 회원 데이터는 자체 DB를 구축할 수 있고, 외부 시스템과 연동할 수 있다. (미확정)
- 주문과 할인 정책
  - 회원은 상품을 주문할 수 있다.
  - 회원 등급에 따라 할인 정책을 적용할 수 있다.
  - 할인 정책은 모든 VIP는 1000원을 할인해주는 고정 금액 할인을 적용해달라. (나중에 변경 될 수 있다.)
  - 할인 정책은 변경 가능성이 높다. 회사의 기본 할인 정책을 아직 정하지 못했고, 오픈 직전까지 고민을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

변경 사항이 생길 수 있는 요소는 인터페이스를 이용하여 구현한다.

회원 도메인 설계

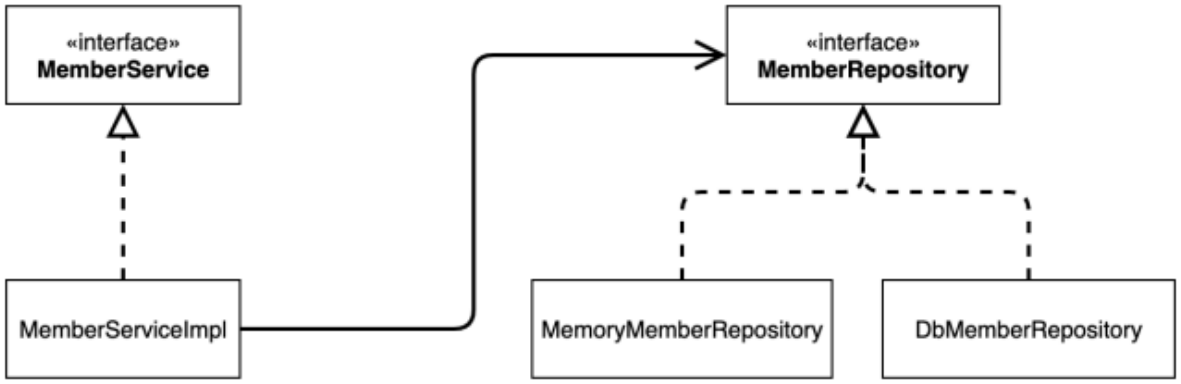
1. 회원을 가입하고 조회할 수 있다.
2. 회원 등급은 일반, VIP 두 가지가 있다.
3. 회원 데이터는 자체 DB를 구축할 수도 있고, 외부 시스템과 연동할 수도 있다.

회원 도메인 협력 관계



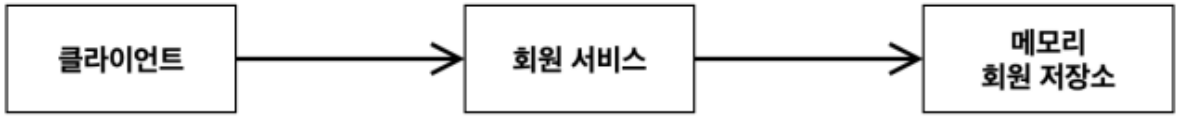
➔ 즉, 회원 저장소는 변경 사항이 생길 수 있으므로 인터페이스를 활용한다.

“회원 클래스 다이어그램”



인터페이스와 그 클래스 구현은 위와 같이 이루어질 것이다. 클래스 다이어그램은 정적이다.

“회원 객체 다이어그램”



객체 간의 참조 양상은 위와 같을 것이다. 객체 다이어그램은 동적이다.

회원 도메인 개발

1. 회원 엔티티
    A. 회원 등급

```java
package yushin.core.member;

no usages  new *
public enum Grade {
    no usages
    BASIC,
    no usages
    VIP
}
```

B. 회원 엔티티

```java
package yushin.core.member;

no usages  new *
public class Member {

    3 usages
    private Long id;
    3 usages
    private String name;
    3 usages
    private Grade grade;

    no usages  new *
    public Member(Long id, String name, Grade grade) {
        this.id = id;
        this.name = name;
        this.grade = grade;
    }
```

```
    no usages  new *
    public Long getId() {
        return id;
    }

    no usages  new *
    public void setId(Long id) {
        this.id = id;
    }

    no usages  new *
    public String getName() {
        return name;
    }

    no usages  new *
    public void setName(String name) {
        this.name = name;
    }

    no usages  new *
    public Grade getGrade() {
        return grade;
    }

    no usages  new *
    public void setGrade(Grade grade) {
        this.grade = grade;
    }
}
```

ALT + INSERT를 이용해 간편하게 생성자와 게터&세터를 정의할 수 있다.

2. 회원 저장소

A. 인터페이스

```java
package yushin.core.member;

no usages  new *
public interface MemberRepository {

    no usages  new *
    void save(Member member);

    no usages  new *
    Member findById(Long memberId);
}
```

간단하게 저장, 검색(ID 사용) 기능만 정의

B. 구현체: 데이터베이스가 정해지지 않았으므로 테스트를 위해 메모리 기반 저장소를 먼저 구현한다.

```java
package yushin.core.member;

import java.util.HashMap;
import java.util.Map;

no usages  new *
public class MemoryMemberRepository implements MemberRepository {

    2 usages
    private static Map<Long, Member> store = new HashMap<>();

    no usages  new *
    @Override
    public void save(Member member) {
        store.put(member.getId(), member);
    }

    no usages  new *
    @Override
    public Member findById(Long memberId) {
        return store.get(memberId);
    }
}
```

실무에서는 동시성 문제 해결을 위해 HashMap 대신 ConcurrentHashMap을 활용해야 한다.

3. 회원 서비스

   A. 인터페이스

```java
package yushin.core.member;

1 usage  1 implementation  new *  1 related problem
public interface MemberService {

    no usages  new *
    void join(Member member);

    no usages  new *
    Member findMember(Long memberId);
}
```

간단하게 가입, 검색 기능만 정의

B. 구현체

```java
package yushin.core.member;

no usages  new *
public class MemberSericeImpl implements MemberService {

    2 usages
    private final MemberRepository memberRepository = new MemoryMemberRepository();

    no usages  new *
    @Override
    public void join(Member member) {
        memberRepository.save(member);
    }

    no usages  new *
    @Override
    public Member findMember(Long memberId) {
        return memberRepository.findById(memberId);
    }
}
```

구현체가 하나일 경우엔 뒤에 Impl이라는 접미사를 붙여 클래스를 만들기도 한다.

회원 도메인 실행과 테스트

1. 회원 가입 main

```java
package yushin.core;

import yushin.core.member.Grade;
import yushin.core.member.Member;
import yushin.core.member.MemberService;
import yushin.core.member.MemberServiceImpl;

new *
public class MemberApp {

    new *
    public static void main(String[] args) {
        MemberService memberService = new MemberServiceImpl();
        Member member = new Member( id: 1L,  name: "memberA", Grade.VIP);
        memberService.join(member);

        Member findMember = memberService.findMember( memberId: 1L);

        System.out.println("new member = " + member.getName());
        System.out.println("found member = " + findMember.getName());
    }
}
```

순수한 자바 코드로 구현한 애플리케이션, 그러나 main 메소드로 테스트를 진행하는 것엔 한계가 있다. -> jUnit 테스트를 사용한다.

2. 회원 가입 테스트

```java
package yushin.core.member;

import org.assertj.core.api.Assertions;
import org.junit.jupiter.api.Test;

new *
public class MemberServiceTest {

    2 usages
    MemberService memberService = new MemberServiceImpl();
    new *
    @Test
    void join() {
        // given: 이러한 환경이 주어졌을 때
        Member member = new Member( id: 1L,  name: "memberA", Grade.VIP);

        // when: 이렇게 하면
        memberService.join(member);
        Member findMember = memberService.findMember( memberId: 1L);

        // then: 이렇게 된다.
        Assertions.assertThat(member).isEqualTo(findMember);
    }
}
```

given(조건) - when(행동) - then(결과)의 절차로 테스트를 설계한다.

- **문제점**

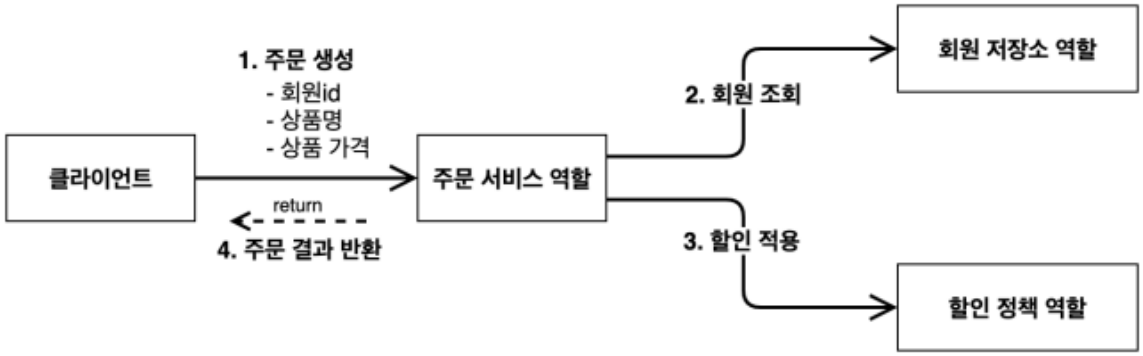
    1. 다른 저장소로 변경할 때 MemberServiceImpl의 코드를 변경해야 한다. -> 추상화 & 구체화에 의존하는 상태이다. -> OCP, DIP 원칙을 위반한다.

주문과 할인 도메인 설계

1. 회원은 상품을 주문할 수 있다.

2. 회원 등급에 따라 할인 정책을 적용할 수 있다.
3. 모든 VIP는 1000원을 할인해 주는 고정 금액 할인 정책을 적용한다. (추후 변경될 수 있다.)
4. 할인 정책은 변경 가능성이 높다. 기본 할인 정책을 아직 정하지 못한 상황이고, 오픈 직전까지 고민을 미루고자 한다. 최악의 경우 할인을 적용하지 않을 수도 있다. (미확정 상태)

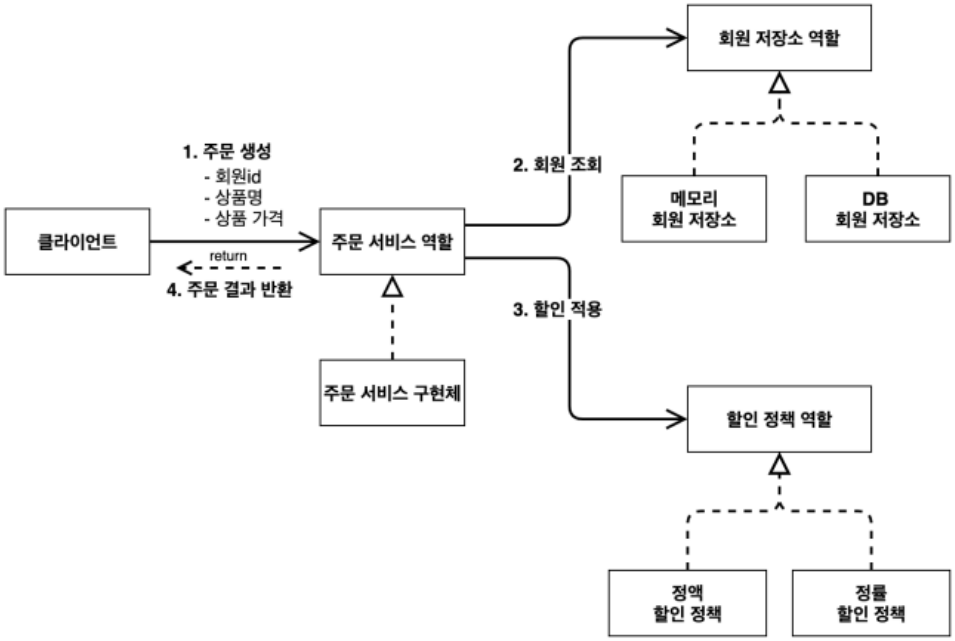
주문 도메인 협력, 역할, 책임



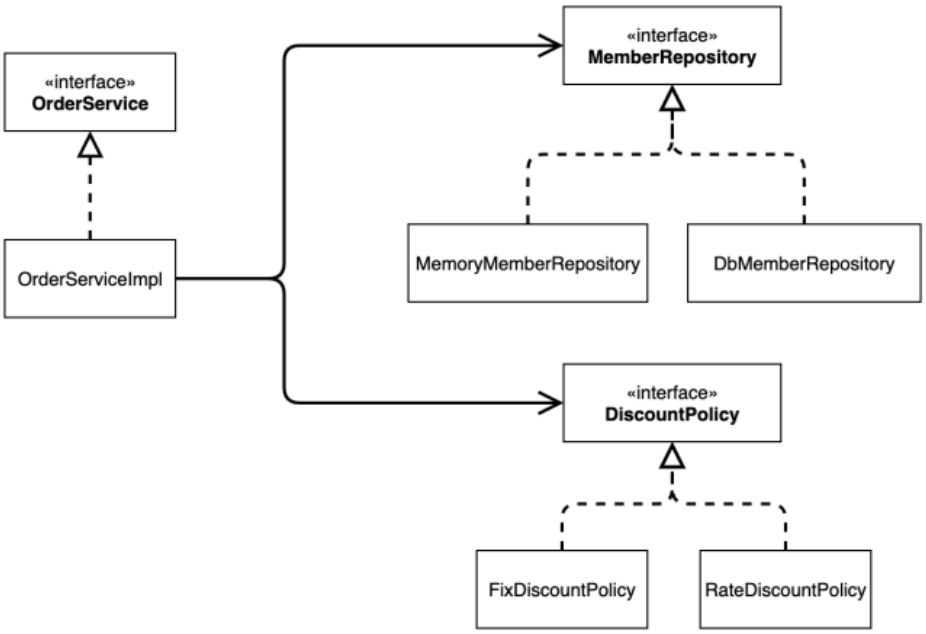
1. **주문 생성:** 클라이언트는 주문 서비스에 주문 생성을 요청한다.
2. **회원 조회:** 할인을 위해서는 회원 등급이 필요하다. 회원 저장소에서 회원(등급)을 조회한다.
3. **할인 적용:** 주문 서비스는 회원 등급에 따른 할인 여부를 할인 정책에 위임한다.
4. **주문 결과 반환:** 주문 서비스는 할인 결과를 포함한 주문 결과를 반환한다.

참고: 실제로는 주문 데이터를 DB에 저장하겠지만, 예제가 너무 복잡해질 수 있어 생략하고 단순히 주문 결과를 반환하게끔 구현한다.

“주문 도메인 전체”: 역할과 구현을 분리해서 구현 객체를 조립할 수 있게 설계했다. 덕분에 회원 저장소는 물론이고, 할인 정책도 유연하게 변경할 수 있다.

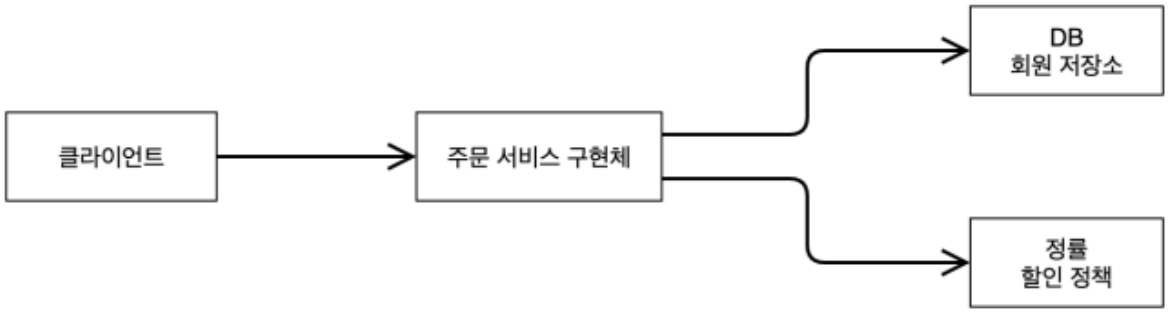


“주문 도메인 클래스 다이어그램”



“주문 도메인 객체 다이어그램(메모리 저장소 ver.)”: 회원을 메모리에서 조회하고, 정액 할인 정책(고정 금액)을 지원해도 주문 서비스에는 변경이 생기지 않는다. -> 역할들의 협력 관계를 재사용할 수 있다.

“주문 도메인 객체 다이어그램(DB 저장소 ver.)”: 회원을 실제 DB에서 조회하고 정률 할인 정책(주문 금액에 따라 % 할인)을 지원해도 주문 서비스를 변경하지 않아도 된다. -> 역할들의 협력 관계를 재사용할 수 있다.



주문과 할인 도메인 개발

1. 할인 정책 인터페이스

```java
package yushin.core.discount;

import yushin.core.member.Member;

1 usage  1 implementation  new *
public interface DiscountPolicy {

    /*
    * @return 할인 대상 금액
    * */
    no usages  1 implementation  new *
    int discount(Member member, int price);

}
```

2. 정액 할인 정책 구현체

```java
package yushin.core.discount;

import yushin.core.member.Grade;
import yushin.core.member.Member;

no usages  new *
public class FixedDiscountPolicy implements DiscountPolicy {

    1 usage
    private int discountFixedAmount = 1000;     // 1000원 할인

    no usages  new *
    @Override
    public int discount(Member member, int price) {
        if (member.getGrade() == Grade.VIP) {
            return discountFixedAmount;
        } else {
            return 0;
        }
    }
}
```

3. 주문 엔티티

```java
package yushin.core.order;

no usages  new *
public class Order {

    3 usages
    private Long memberId;
    3 usages
    private String itemName;
    4 usages
    private int itemPrice;
    4 usages
    private int discountPrice;

    no usages  new *
    public Order(Long memberId, String itemName, int itemPrice, int discountPrice) {
        this.memberId = memberId;
        this.itemName = itemName;
        this.itemPrice = itemPrice;
        this.discountPrice = discountPrice;
    }

    no usages  new *
    public int calculatePrice() {
        return itemPrice - discountPrice;
    }
```

아래 게터 & 세터 코드는 생략. 할인된 금액을 계산하는 메소드와 오버라이딩된 toString 메소드를 추가적으로 정의한다.

4. 주문 서비스 인터페이스

```java
package yushin.core.order;

no usages  new *
public interface OrderService {
    no usages  new *
    Order createOrder(Long memberId, String itemName, int itemPrice);
}
```

5. 주문 서비스 구현체

```java
package yushin.core.order;

import yushin.core.discount.DiscountPolicy;
import yushin.core.discount.FixDiscountPolicy;
import yushin.core.member.Member;
import yushin.core.member.MemberRepository;
import yushin.core.member.MemoryMemberRepository;

no usages  new *
public class OrderServiceImpl implements OrderService {

    1 usage
    private final MemberRepository memberRepository = new MemoryMemberRepository();
    1 usage
    private final DiscountPolicy discountPolicy = new FixDiscountPolicy();

    no usages  new *
    @Override
    public Order createOrder(Long memberId, String itemName, int itemPrice) {
        Member member = memberRepository.findById(memberId);
        // SRP(단일 책임 원칙)가 잘 지켜진 예시이다. 할인 정책에 변경점이 있어도 받는 영향이 적다.
        int discountPrice = discountPolicy.discount(member, itemPrice);

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

```

할인 정책을 별도의 역할로 분리하여 단일 책임 원칙이 잘 지켜지고 있다.

주문과 할인 도메인 실행과 테스트

1. 주문과 할인 정책 실행

```java
new *
public class OrderApp {

    new *
    public static void main(String[] args) {
        MemberService memberService = new MemberServiceImpl();
        OrderService orderService = new OrderServiceImpl();

        Long memberId = 1L;
        Member member = new Member(memberId, name: "memberA", Grade.VIP);
        memberService.join(member);

        Order order = orderService.createOrder(memberId, itemName: "itemA", itemPrice: 10000);

        System.out.println("order = " + order);
        System.out.println("price = " + order.calculatePrice());
    }
}
```

2. 주문과 할인 정책 테스트

```java
new *
public class OrderServiceTest {

    1 usage
    MemberService memberService = new MemberServiceImpl();
    1 usage
    OrderService orderService = new OrderServiceImpl();

    new *
    @Test
    void createOrder() {
        Long memberId = 1L;
        Member member = new Member(memberId, name: "memberA", Grade.VIP);
        memberService.join(member);

        Order order = orderService.createOrder(memberId, itemName: "itemA", itemPrice: 10000);
        Assertions.assertThat(order.getDiscountPrice()).isEqualTo( expected: 1000);

    }
}
```