[수정본]

우아한 객체지향

의존성을 이용해 설계 진화시키기

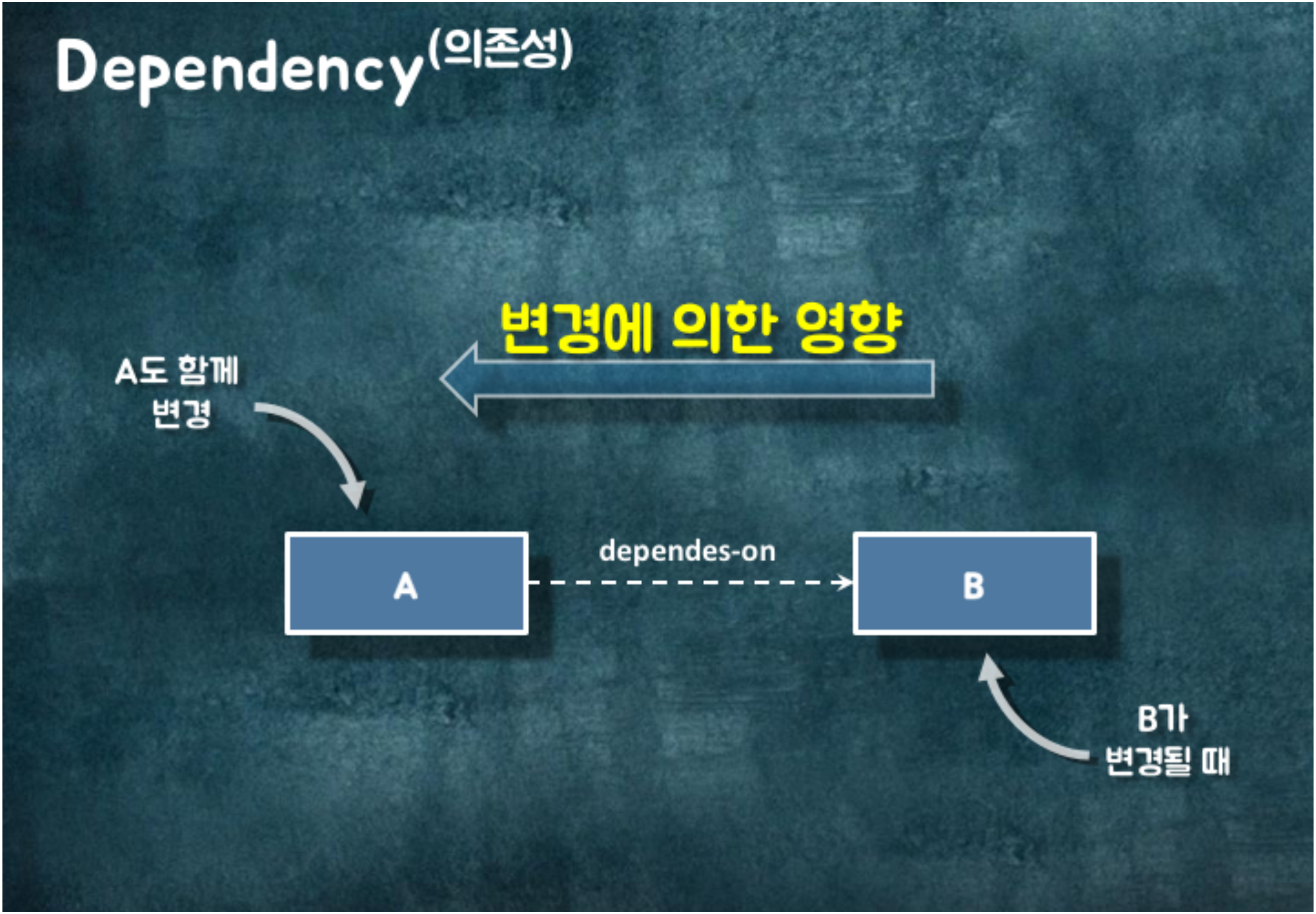
https://github.com/eternity-oop

우아한 형제들

Part - 1

의존성

Dependency(의존성)



클래스 의존성의 종류

Association 연관관계

```
class A {
  private B b;
}
```

클래스 의존성의 종류

Association 연관관계

Dependency 의존 관계

```
class A {
  public B method(B b) {
    return new B();
  }
}
```

클래스 의존성의 종류

Association 연관관계

Dependency 의존 관계

Inheritance 상속 관계

```
class A extends B {
}
```

클래스 의존성의 종류

Association 연관관계

Dependency 의존 관계

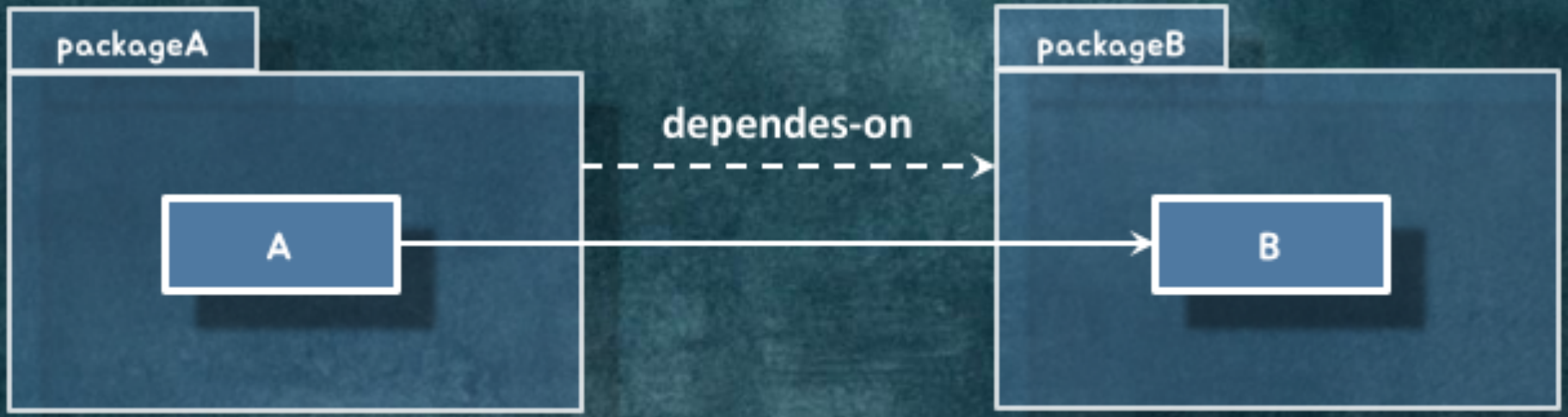
Inheritance 상속 관계

Realization 실체화 관계

```
class A implements B {
}
```

패키지 의존성

패키지에 포함된
클래스 사이의 의존성



양방향 의존성을 피하라

Bi-Directional양방향


```
class A {
  private B b;

  public void setA(B b) {
    this.b = b;
    this.b.setA(this);
  }
}

class B {
  private A a;

  public void setA(A a) {
    this.a = a;
  }
}
```

Uni-Directional단방향


```
class A {
  private B b;

  public void setA(B b) {
    this.b = b;
  }
}


class B {
}
```

다중성이 적은 방향을 선택하라

One-To-Many 일대다


```
class A {
  private Collection<B> bs;
}

class B {
}
```

Many-To-One 다대일


```
class A {
}

class B {
 private A a;
}
```

의존성이 필요없다면 제거하라

Uni-Directional 단방향


```
class A {
  private B b;
}


class B {
}
```

None 없음


```
class A {
}


class B {
}
```

패키지 사이의 의존성 사이클을 제거하라

Bi-Directional양방향

Uni-Directional단방향

Part - 2

예제 살펴보기

배달앱

주문 플로우

가게선택


9:12
따르릉직화삼겹 & 불고기
4.9
최소주문금액 14,000원
결제방법 바로결제, 만나서결제
전화주문
찜 410
공유
메뉴
정보
리뷰
대표메뉴
고기만!+쌈장 내맘대로 세트
· 10,000원 ~ 25,000원
직화불고기 도시락(국내산)
· 14,000원

주문 플로우

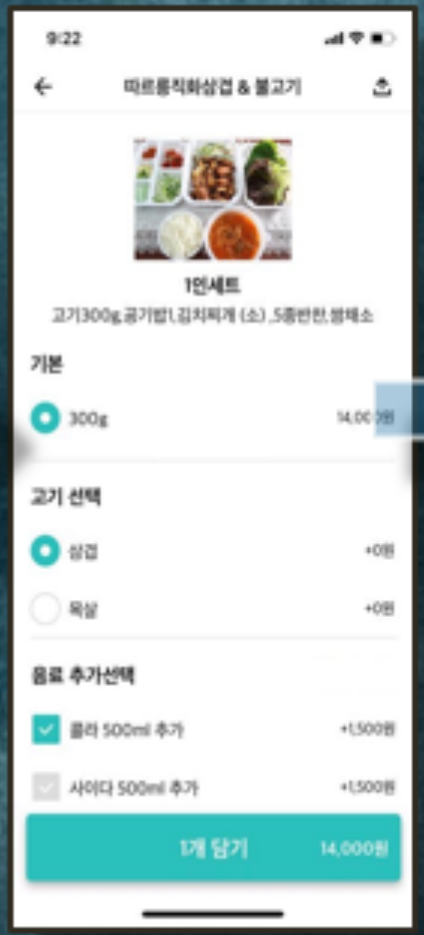
가게선택

메뉴선택

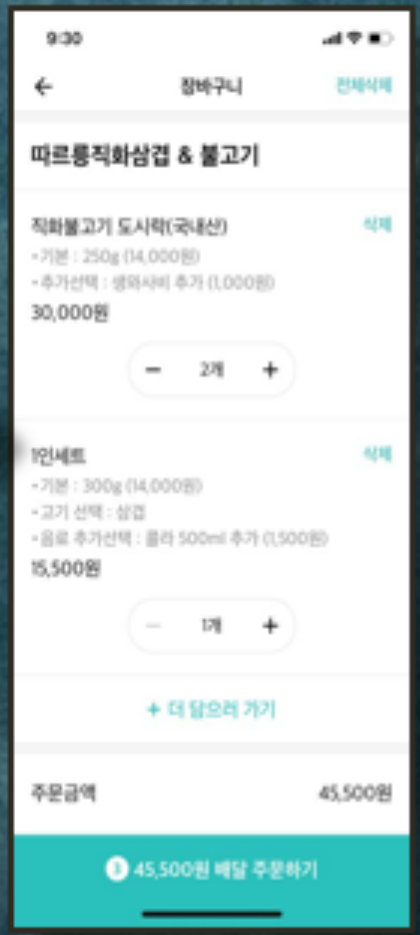
주문 플로우

가게선택

메뉴선택



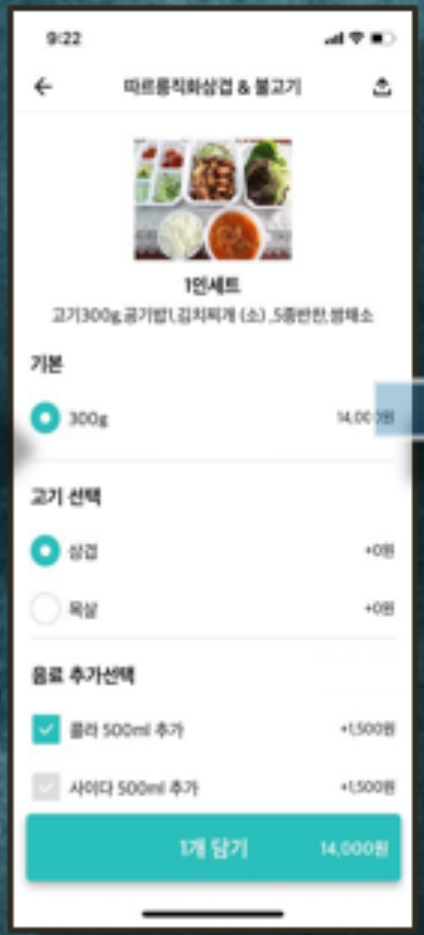
장바구니담기



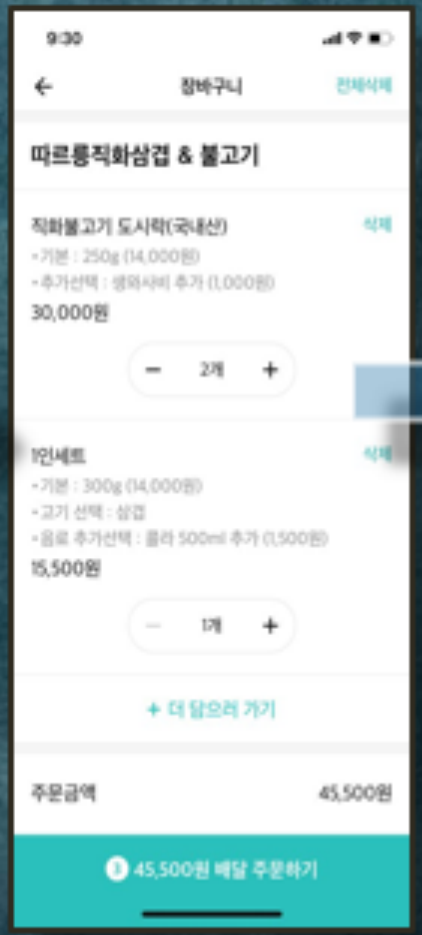
주문 플로우

가게선택

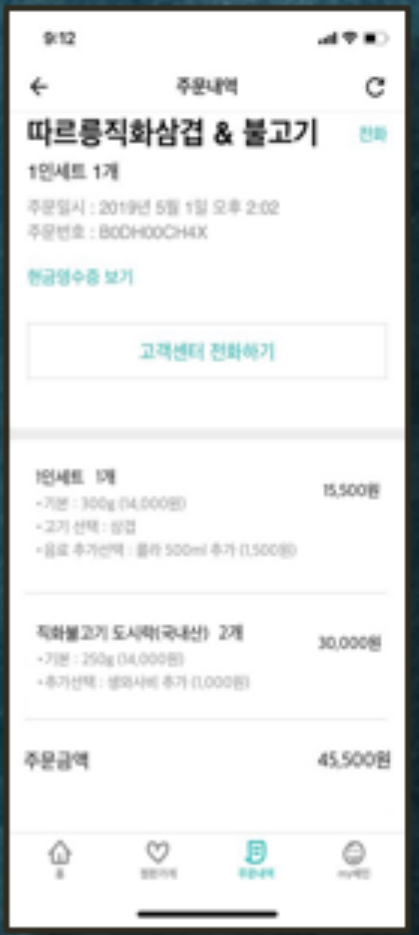
메뉴선택



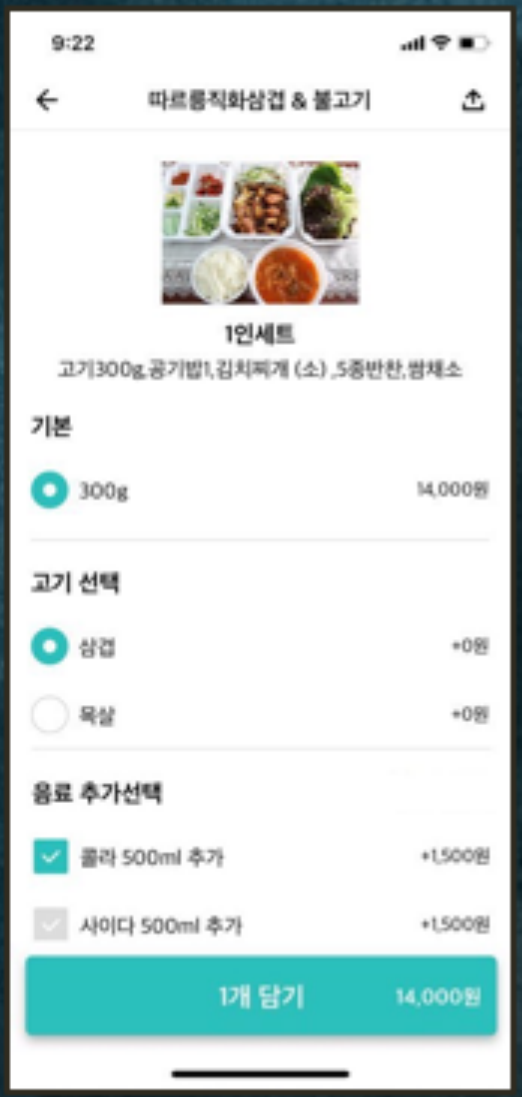
장바구니담기

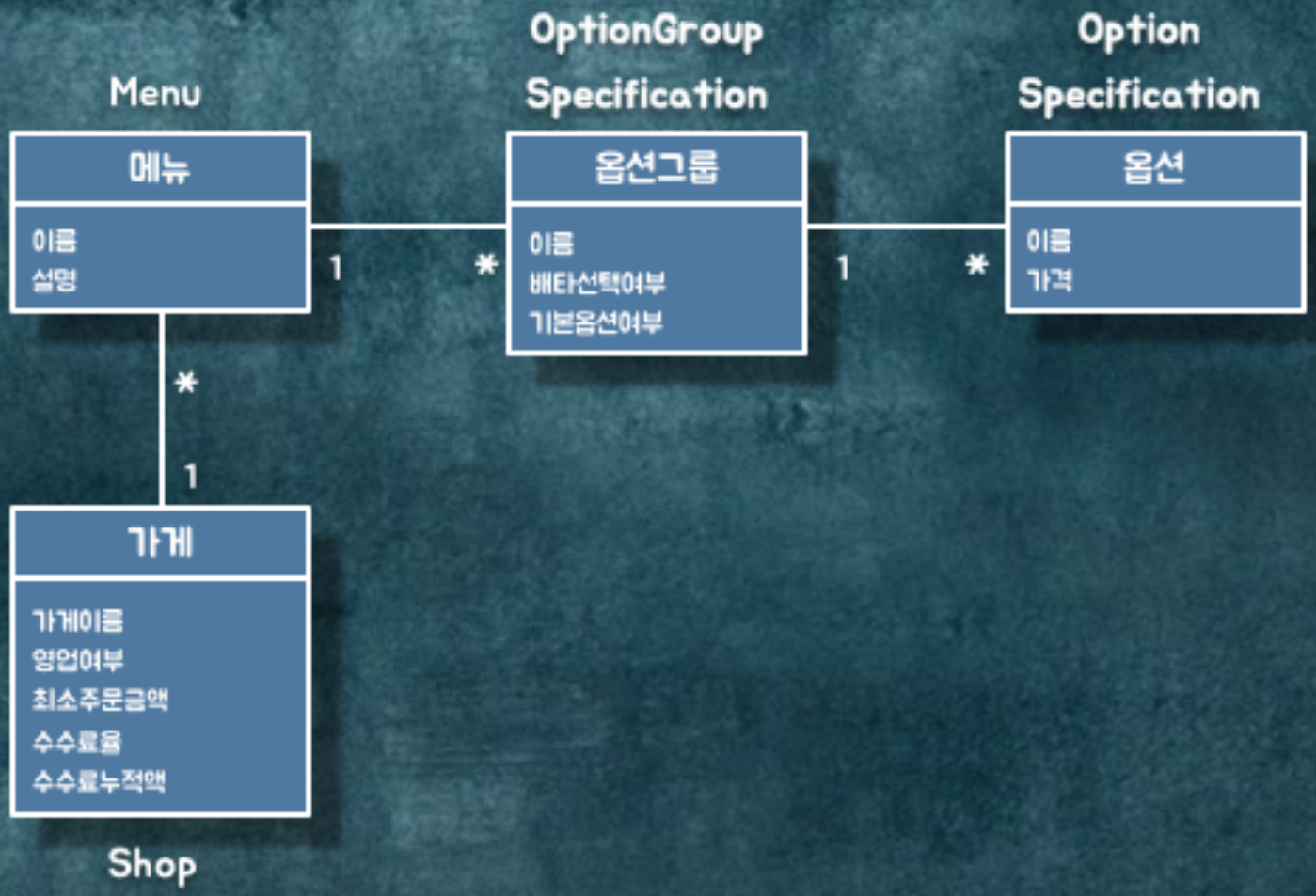


주문완료

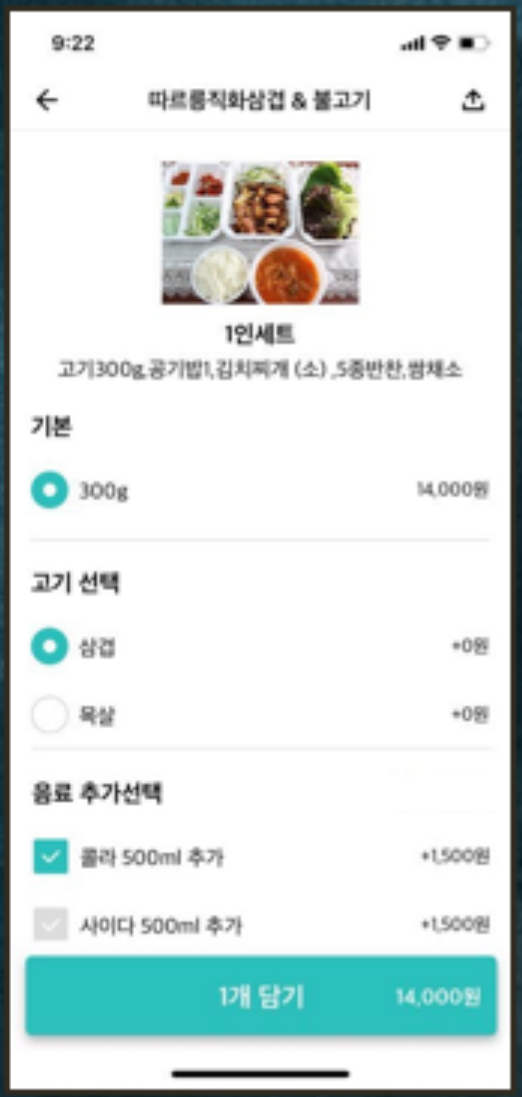


Domain Concept - 가게 & 메뉴





Domain Objects - 가게 & 메뉴



:메뉴

이름 = '1인세트'
설명 = '고기300g...'

:가게

가게이름 = '따르릉직화삼겹&불고기'
영업여부 = TRUE
최소주문금액 = 14,000원
수수료율 = 1%
수수료누적액 = 150원

:옵션그룹

이름 = '기본'
기본옵션여부 = TRUE
배타선택여부 = TRUE

:옵션

이름 = '300g'
가격 = 14,000원

:옵션그룹

이름 = '고기선택'
기본옵션여부 = FALSE
배타선택여부 = TRUE

:옵션

이름 = '삼겹'
가격 = 0원

:옵션

이름 = '목살'
가격 = 0원

:옵션그룹

이름 = '음료추가선택'
기본옵션여부 = FALSE
배타선택여부 = FALSE

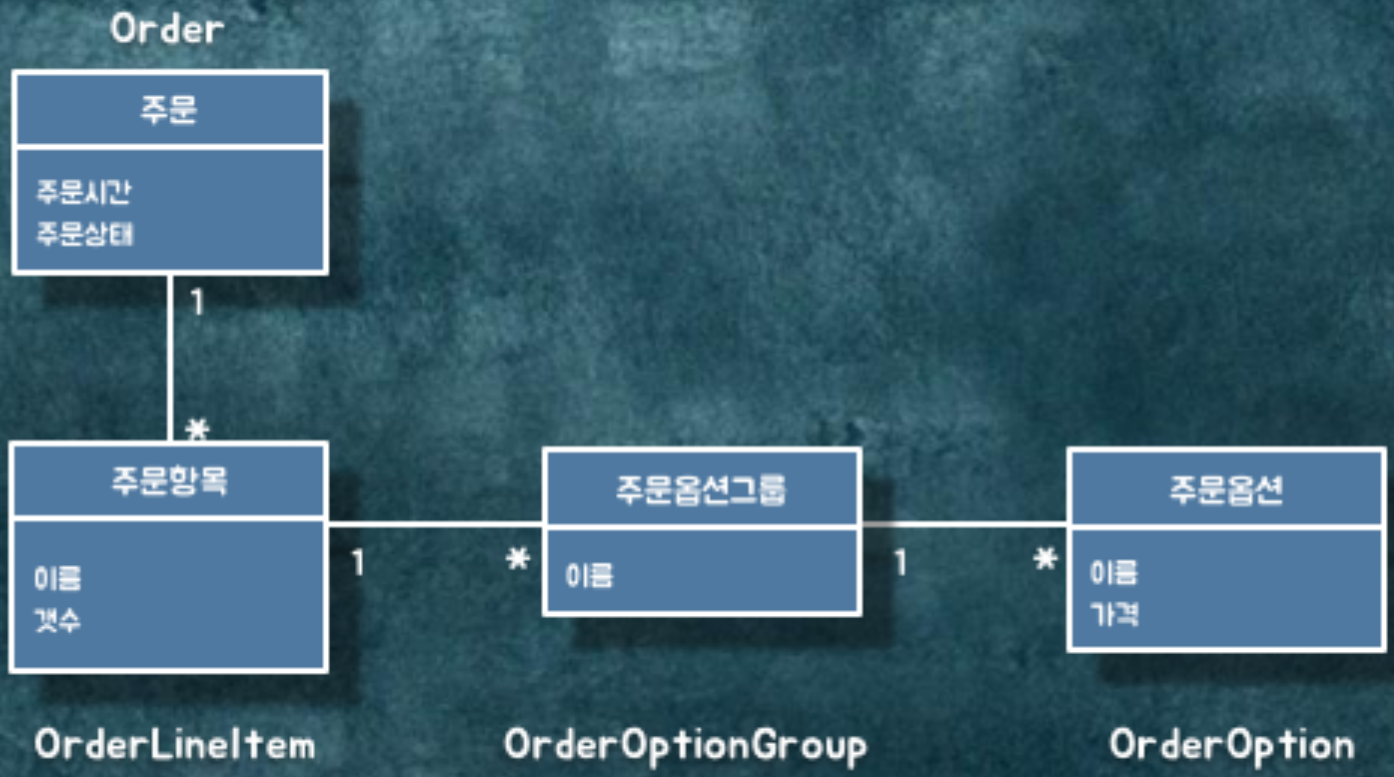
:옵션

이름 = '콜라500ml추가'
가격 = 1,500원

:옵션

이름 = '사이다500ml추가'
가격 = 1,500원

Domain Concept - 주문



Domain Object - 주문

:주문

주문시간 = 2019-05-01 14:02
주문상태 = ORDERED

:주문항목

이름 = '1인세트'
갯수 = 1

:주문항목

이름 = '직화불고기 도시락(국내산)'
갯수 = 2

:주문옵션그룹

이름 = '기본'

:주문옵션

이름 = '300g'
가격 = 14,000원

:주문옵션그룹

이름 = '고기선택'

:주문옵션

이름 = '삼겹'
가격 = 0원

:주문옵션그룹

이름 = '음료추가선택'

:주문옵션

이름 = '콜라500ml추가'
가격 = 1,500원

:주문옵션그룹

이름 = '기본'

:주문옵션

이름 = '250g'
가격 = 14,000원

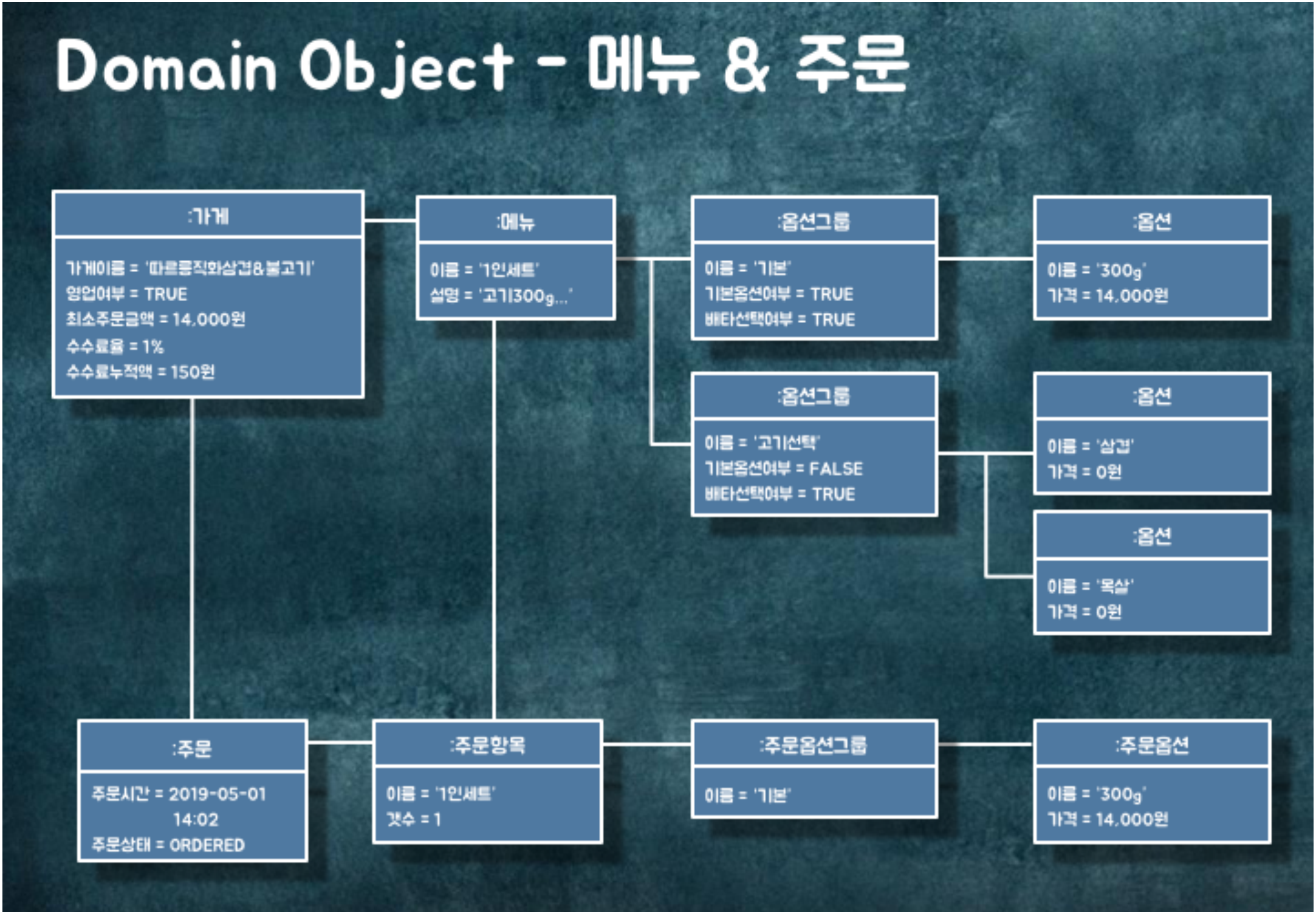
:주문옵션그룹

이름 = '추가선택'

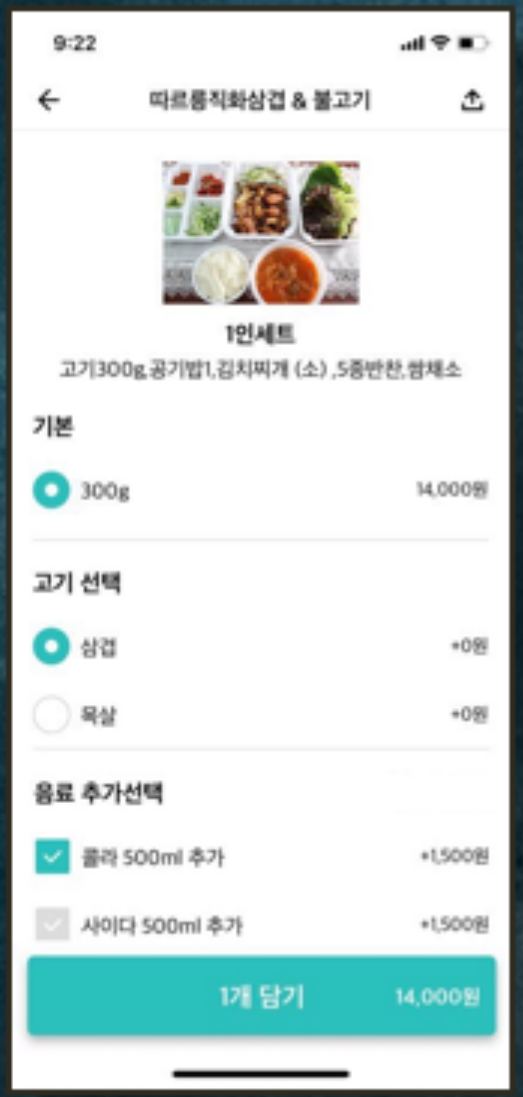
:주문옵션

이름 = '생와사비 추가'
가격 = 1,000원

Domain Object - 메뉴 & 주문



(문제점) 메뉴 선택



:메뉴
이름 = '1인세트'
설명 = '고기300g...'

:옵션그룹
이름 = '기본'
기본옵션여부 = TRUE
배타선택여부 = TRUE

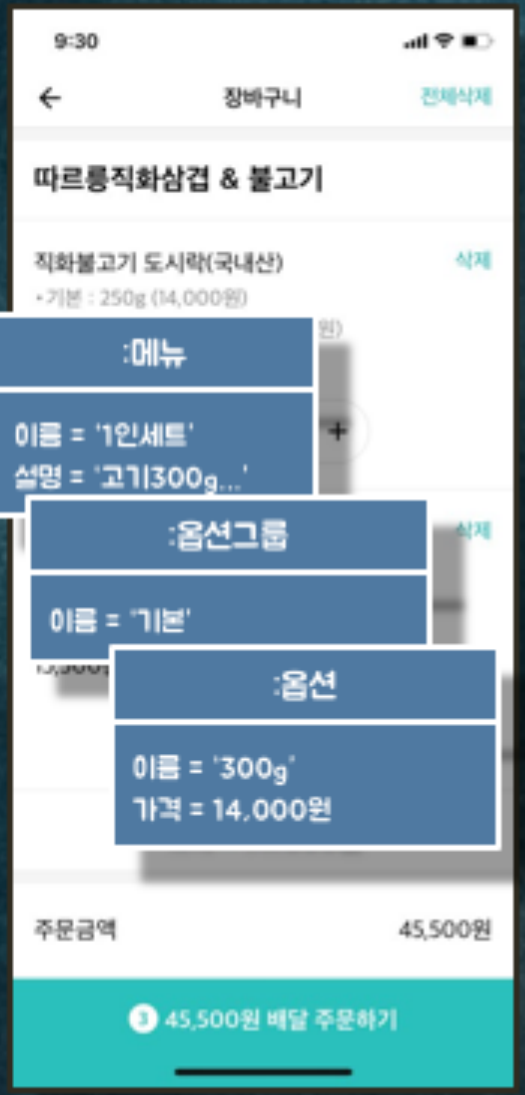
:옵션
이름 = '300g'
가격 = 7,000원

:옵션그룹
이름 = '고기선택'
기본옵션여부 = FALSE
배타선택여부 = TRUE

:옵션
이름 = '삼겹'
가격 = 0원

:옵션
이름 = '목살'
가격 = 0원

(문제점) 장바구니 담기



:메뉴
이름 = '1인세트'
설명 = '고기300g...'

:옵션그룹
이름 = '기본'
기본옵션여부 = TRUE
베타선택여부 = TRUE

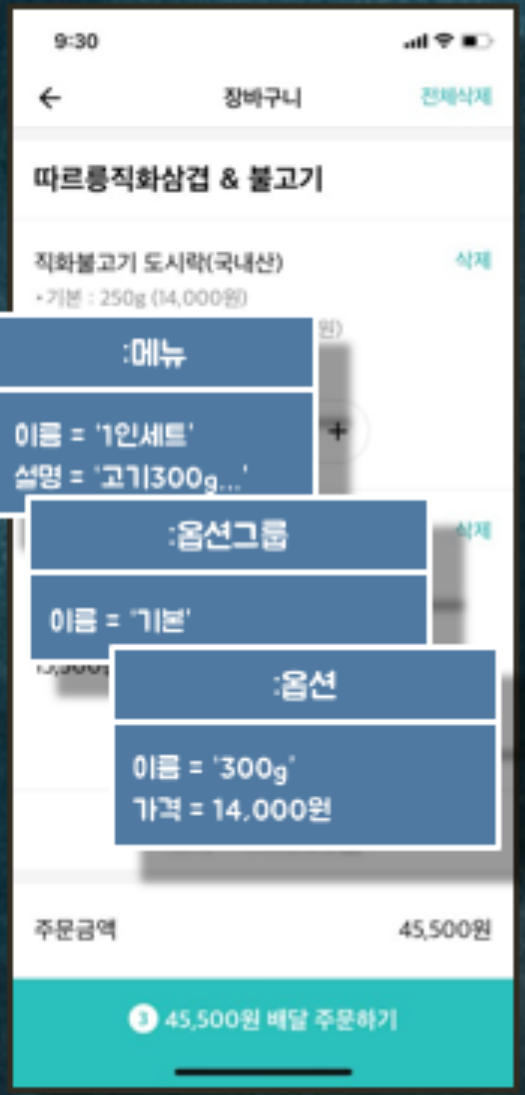
:옵션
이름 = '300g'
가격 = 7,000원

:옵션그룹
이름 = '고기선택'
기본옵션여부 = FALSE
베타선택여부 = TRUE

:옵션
이름 = '삼겹'
가격 = 0원

:옵션
이름 = '목살'
가격 = 0원

(문제점) 업소 메뉴 변경



:메뉴

이름 = '0.5인세트'
설명 = '고기150g...'

:옵션그룹

이름 = '기본'
기본옵션여부 = TRUE
베타선택여부 = TRUE

:옵션

이름 = '150g'
가격 = 7,000원

:옵션그룹

이름 = '고기선택'
기본옵션여부 = FALSE
베타선택여부 = TRUE

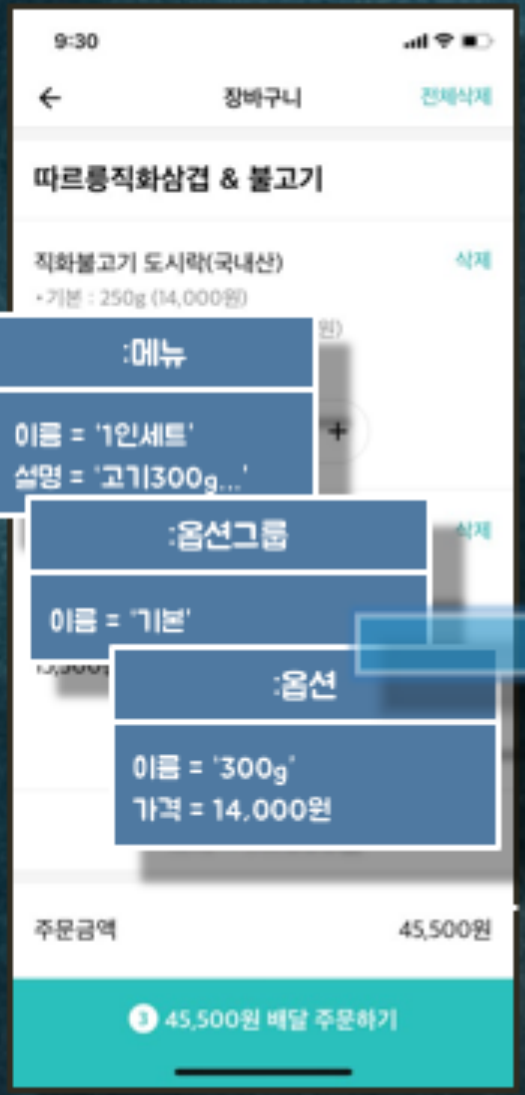
:옵션

이름 = '삼겹'
가격 = 0원

:옵션

이름 = '목살'
가격 = 0원

(문제점) 메뉴 불일치



:메뉴
이름 = '1인세트'
설명 = '고기300g...'

:옵션그룹
이름 = '기본'

:옵션
이름 = '300g'
가격 = 14,000원

:주문
주문시간 = 2019-05-01 14:02
주문상태 = ORDERED

:주문항목
이름 = '1인세트'
갯수 = 1

:주문옵션그룹
이름 = '기본'

:주문옵션
이름 = '300g'
가격 = 14,000원

:메뉴
이름 = '0.5인세트'
설명 = '고기150g...'

:옵션그룹
이름 = '기본'
기본옵션여부 = TRUE
베타선택여부 = TRUE

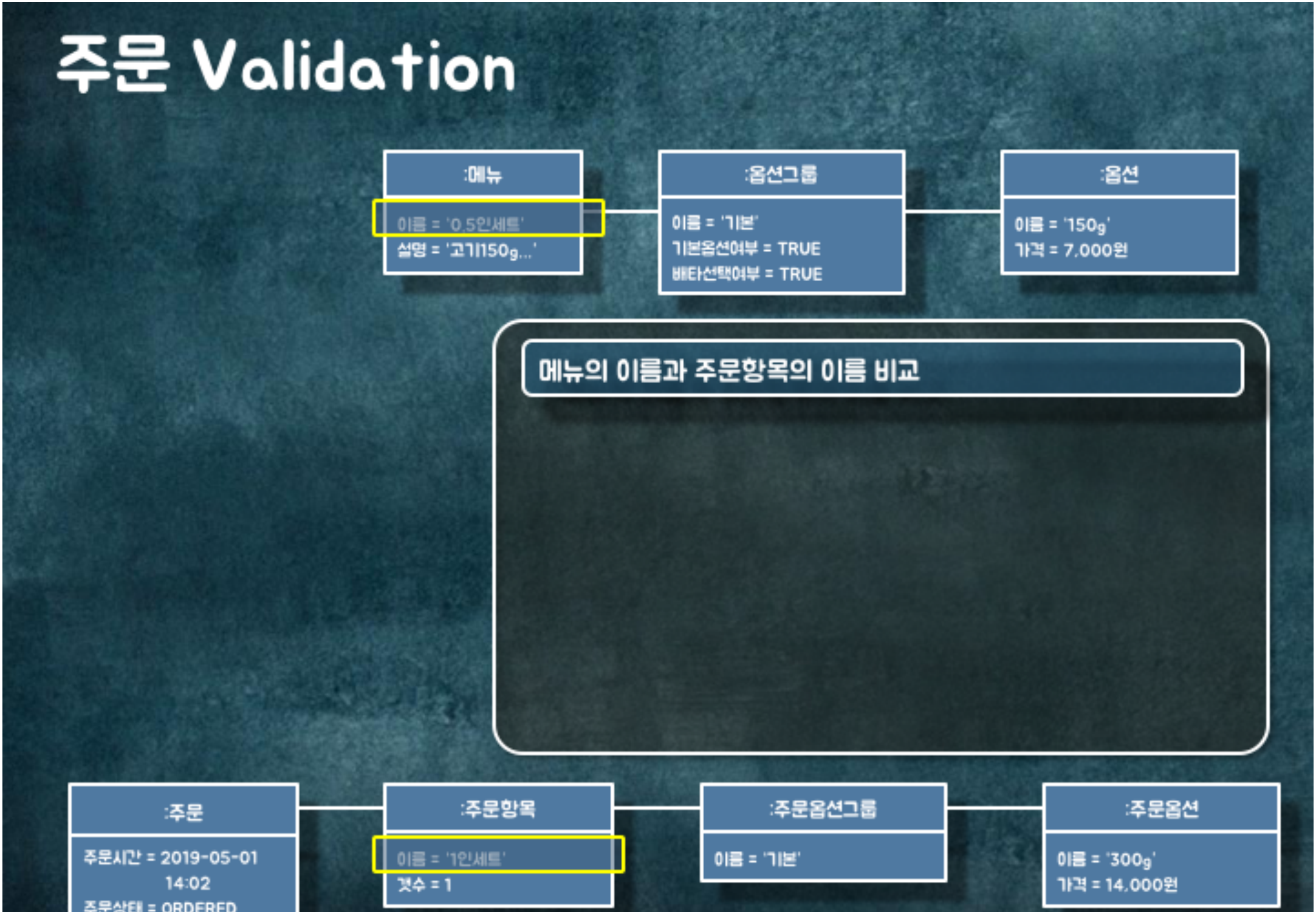
:옵션
이름 = '150g'
가격 = 7,000원

:옵션그룹
이름 = '고기선택'
기본옵션여부 = FALSE
베타선택여부 = TRUE

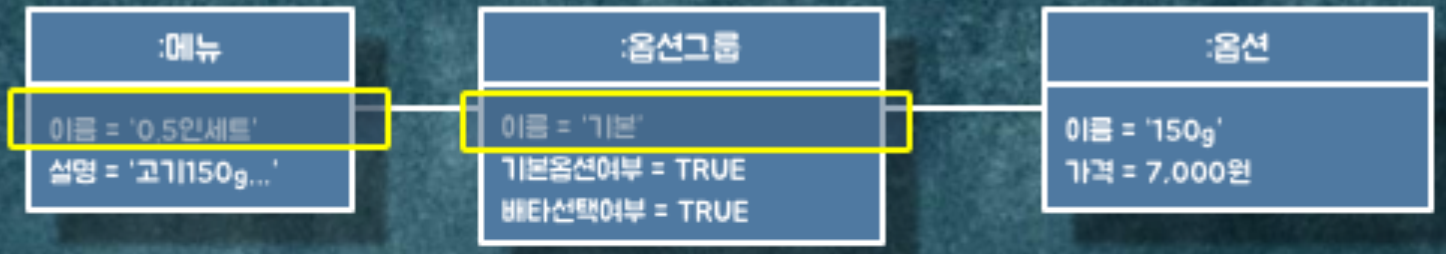
:옵션
이름 = '삼겹'
가격 = 0원

:옵션
이름 = '목살'
가격 = 0원

주문 Validation



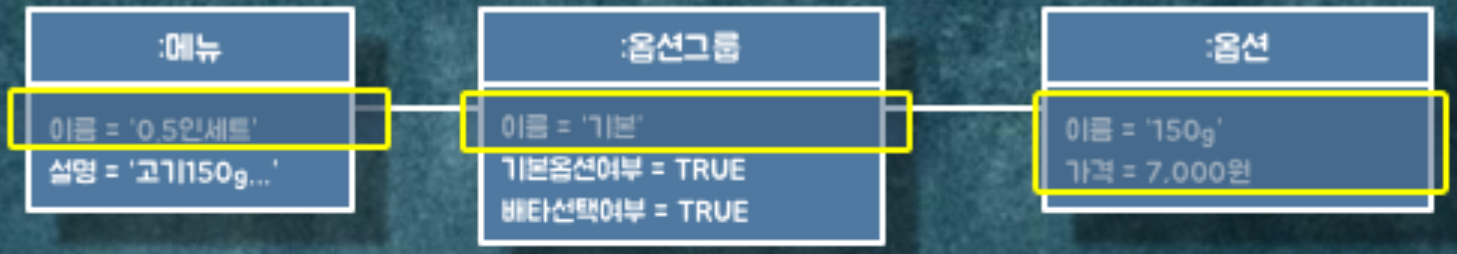
주문 Validation



메뉴의 이름과 주문항목의 이름 비교

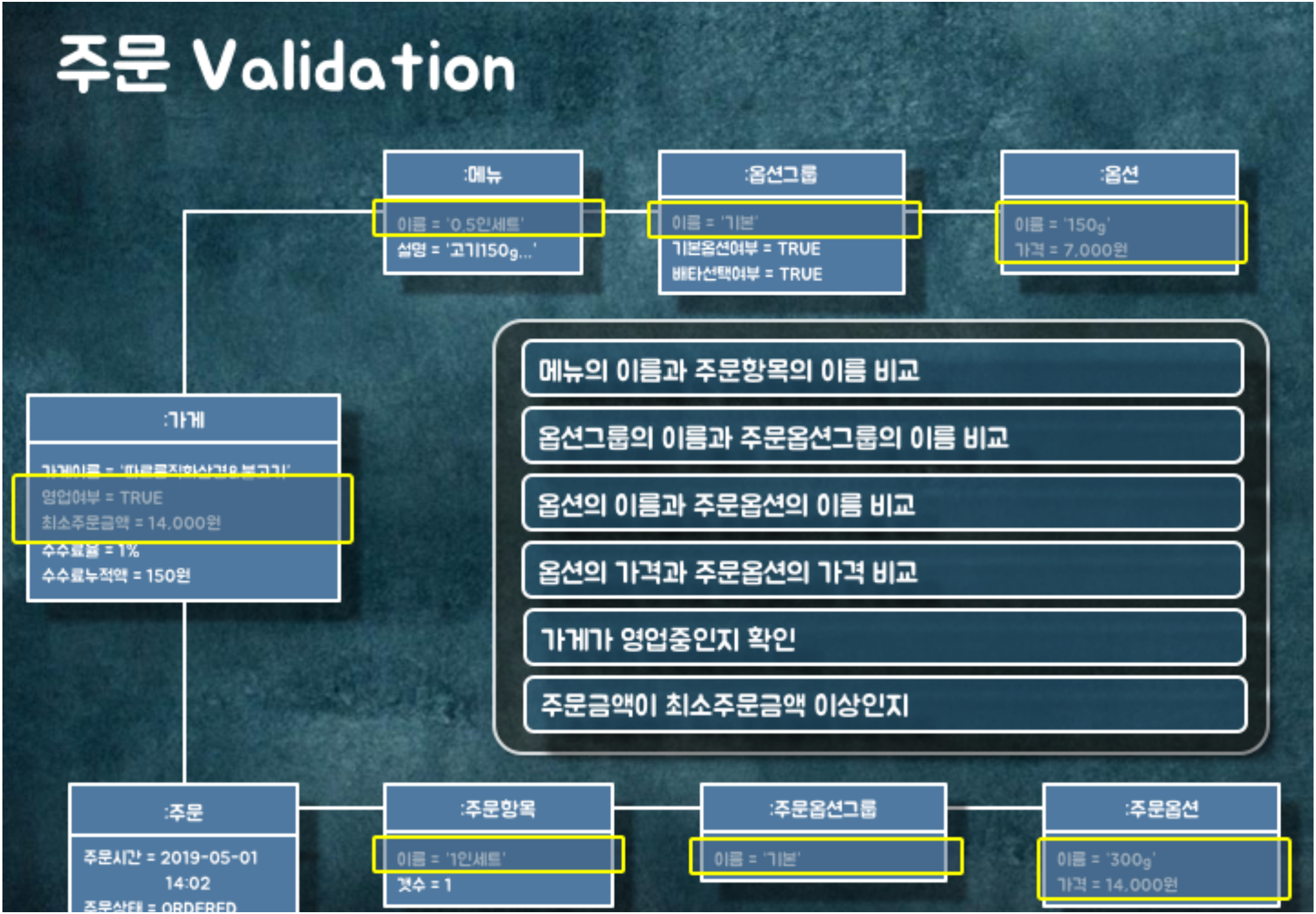
옵션그룹의 이름과 주문옵션그룹의 이름 비교

주문 Validation



- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교

주문 Validation



협력 설계하기

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

협력 설계하기

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

협력 설계하기

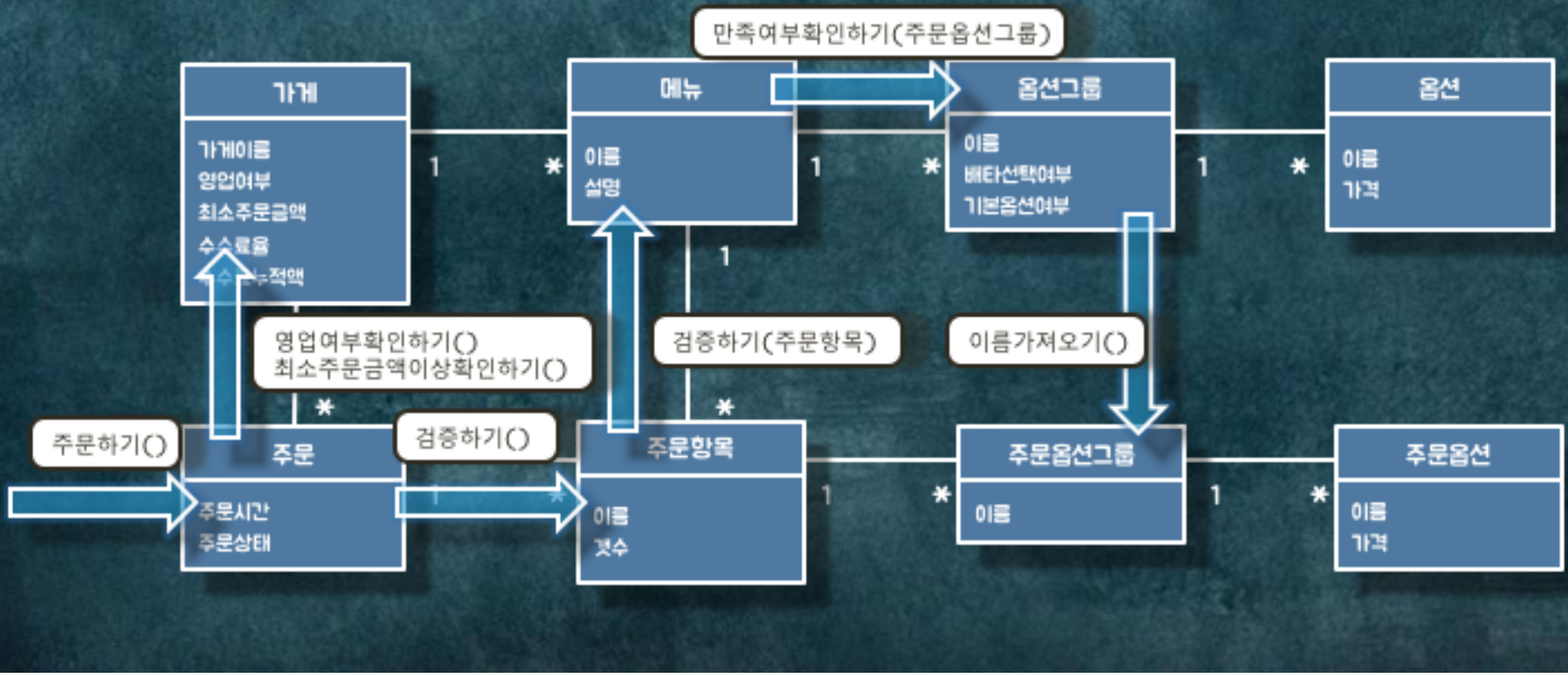
- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

협력 설계하기

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

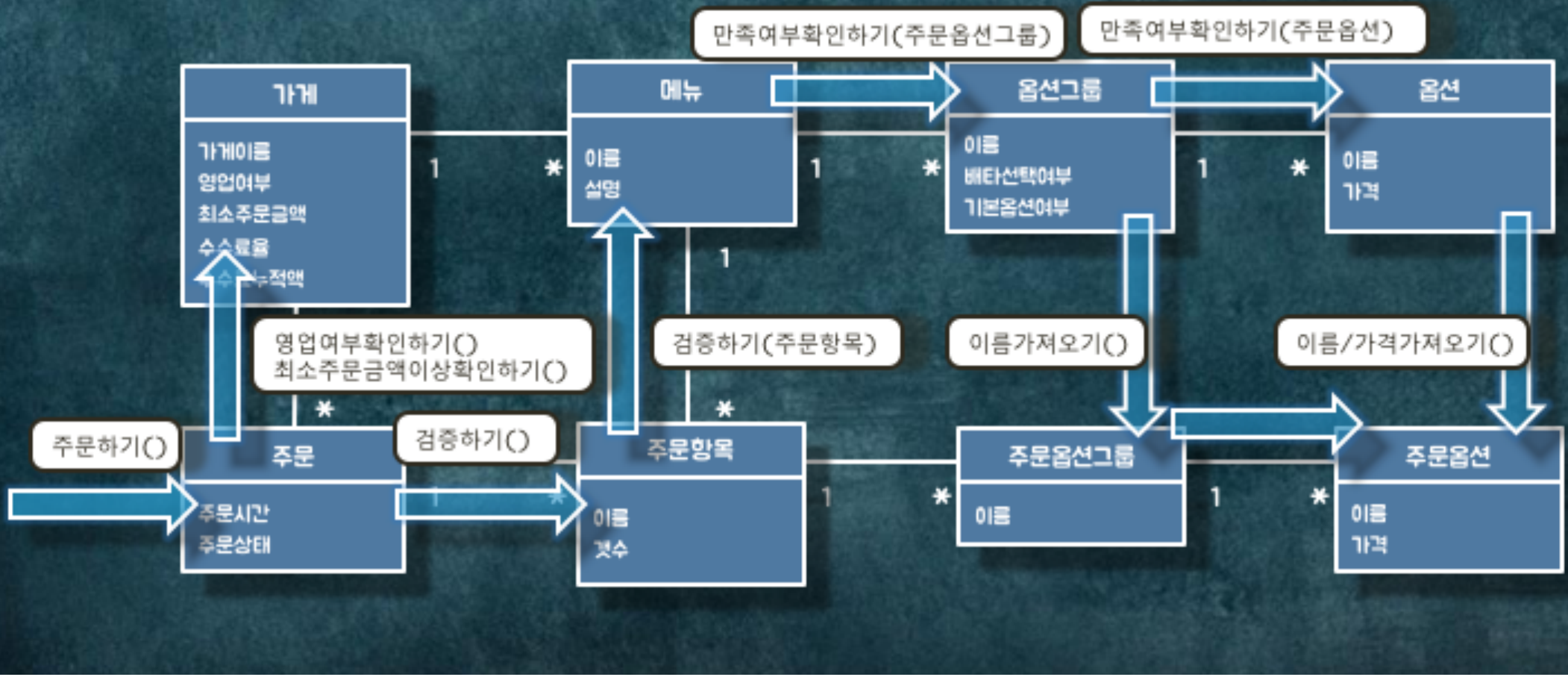
협력 설계하기

- 메뉴의 이름과 주문항목의 이름 비교
- **옵션그룹의 이름과 주문옵션그룹의 이름 비교**
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

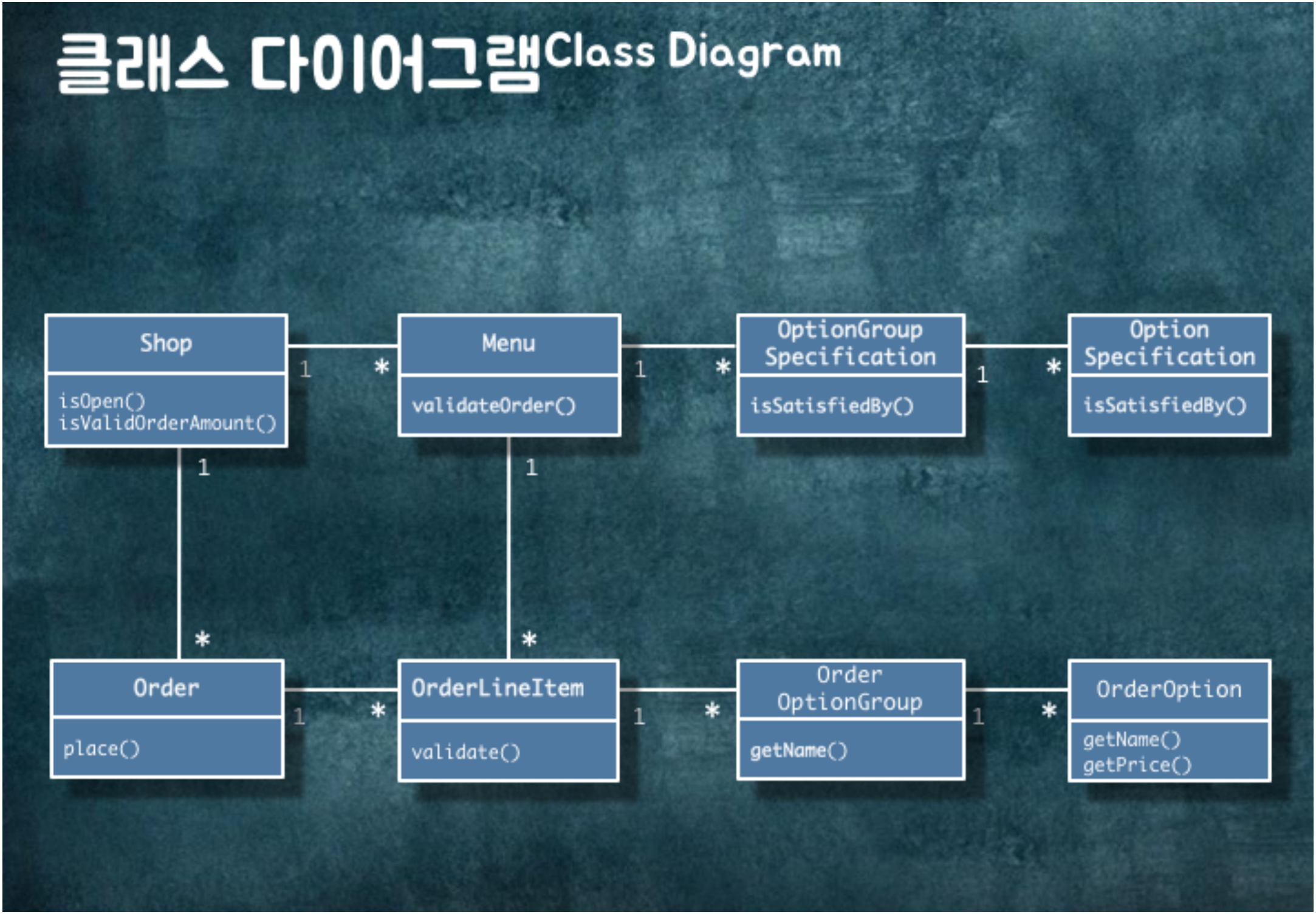


협력 설계하기

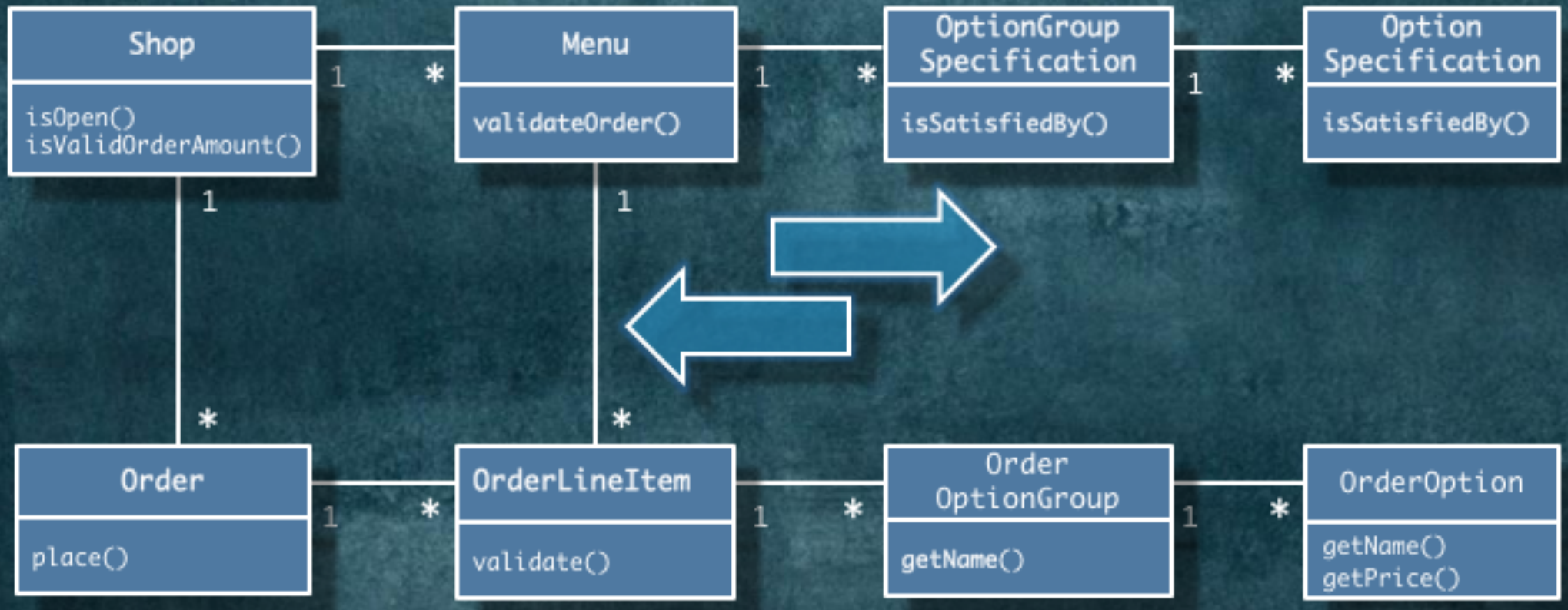
- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지



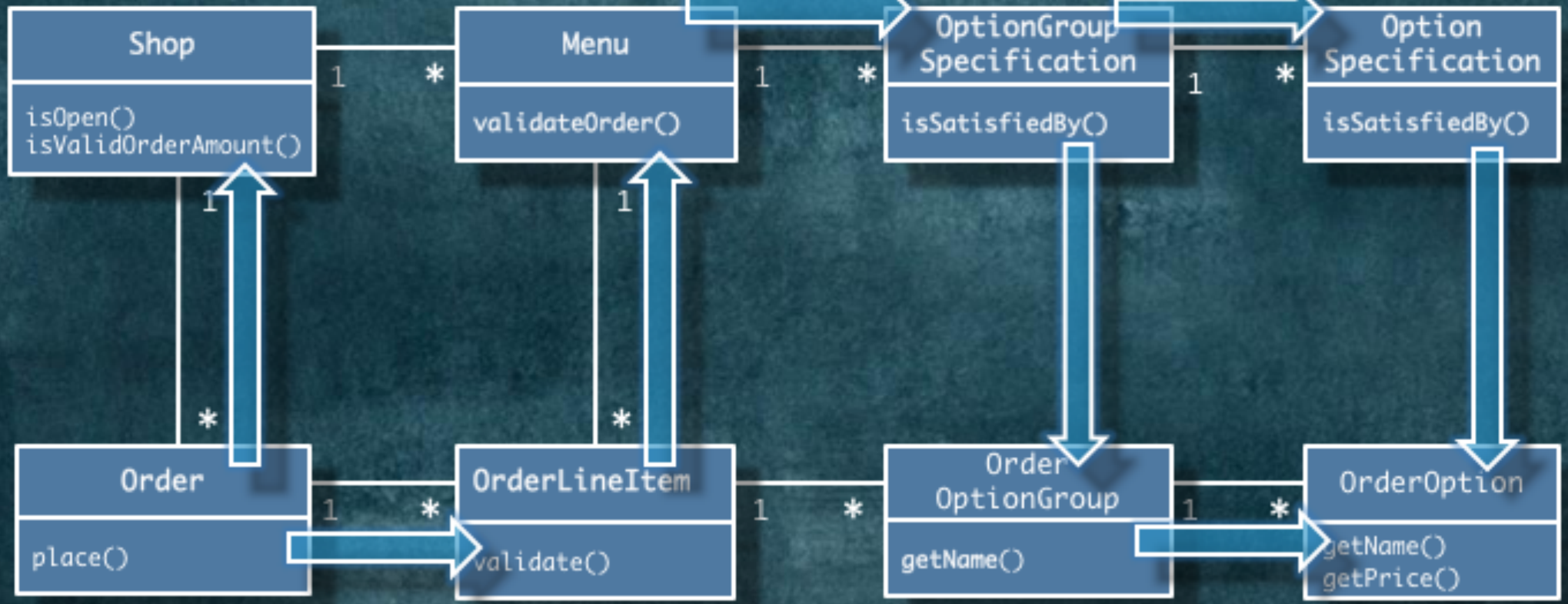
클래스 다이어그램Class Diagram



관계에는 방향성이 필요


Shop
isOpen()
isValidOrderAmount()
Menu
validateOrder()
OptionGroup Specification
isSatisfiedBy()
Option Specification
isSatisfiedBy()
Order
place()
OrderLineItem
validate()
Order OptionGroup
getName()
OrderOption
getName()
getPrice()
1
*

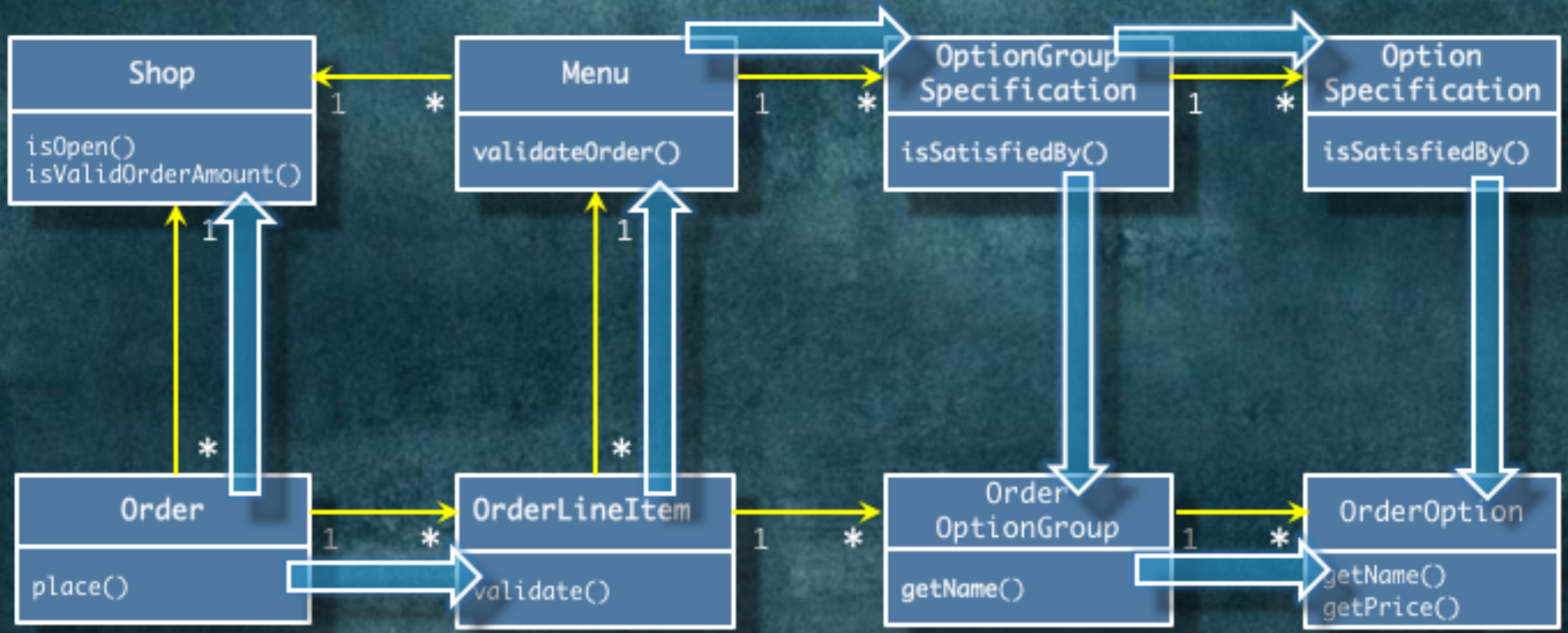
관계의 방향 = 협력의 방향
= 의존성의 방향



관계의 종류 결정하기

연관관계

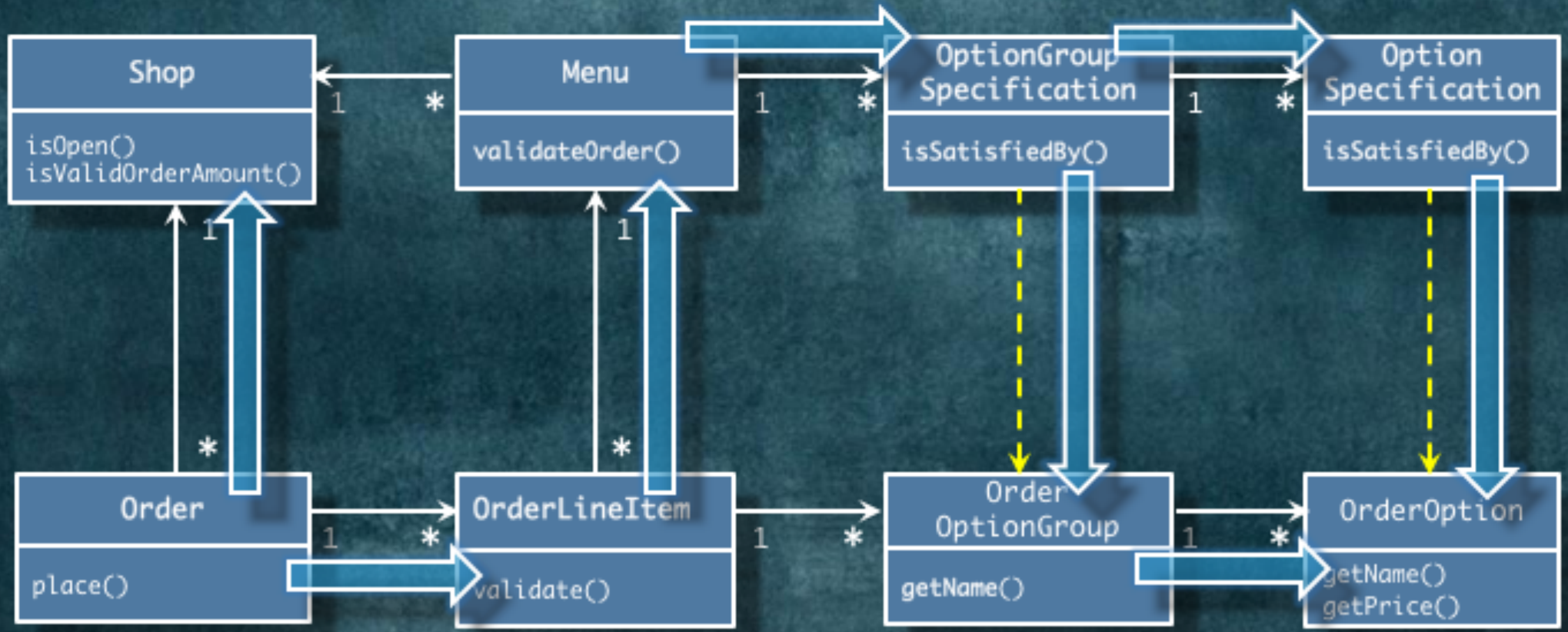
협력을 위해 필요한 영구적인 탐색 구조



관계의 종류 결정하기

의존관계

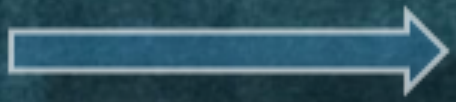
협력을 위해 **일시적으로** 필요한 의존성
(파라미터, 리턴타입, 지연변수)



연관관계 = 탐색가능성(navigability)

Order에서
OrderLineItem으로
탐색가능

Order가
뭔지 알면



Order를 통해 원하는
OrderLineItem을
찾을 수있다.

연관관계와 협력

두 객체 사이에 협력이 필요하고
두 객체의 관계가 영구적이라면
연관관계를 이용해 탐색 경로 구현

객체 참조를 이용한 연관관계 구현

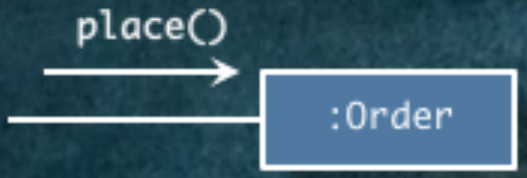

```
class Order {
  private List<OrderLineItem> orderLineItems;

  public void place() {
    validate();
    ordered();
  }

  private void validate() {
    ...

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }
}
```

연관관계를 통해 협력

구현 시작하기

```
public class Order {
  public void place() {
    validate();
    ordered();
  }

  private void validate() {
  }

  private void ordered() {
  }
}
```

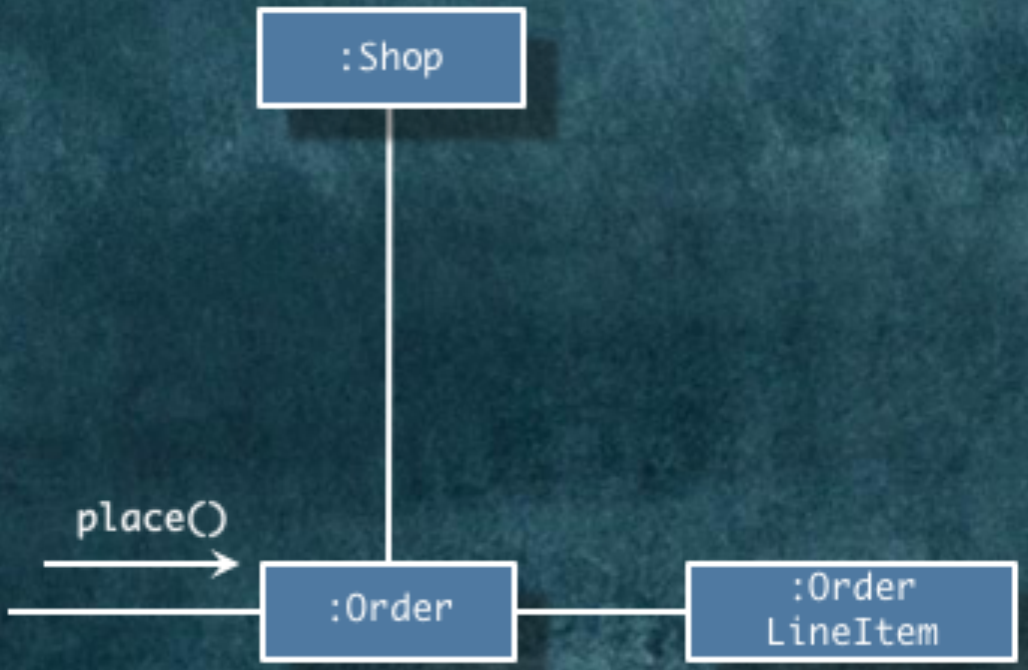
객체 협력



Shop & OrderLineItem 연관관계 연결

```
@Entity
@Table(name="ORDERS")
public class Order {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name="ORDER_ID")
  private Long id;

  @ManyToOne
  @JoinColumn(name="SHOP_ID")
  private Shop shop;

  @OneToMany(cascade = CascadeType.ALL)
  @JoinColumn(name="ORDER_ID")
  private List<OrderLineItem> orderLineItems = new ArrayList<>();

  ...

  public void place() {
    validate();
    ordered();
  }

  private void validate() {
  }

  private void ordered() {
  }
}
```

객체 협력



validate() 메서드

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

```
public class Order {
  public void place() {
    validate();
    ordered();
  }

  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format(
                "최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }

  private void ordered() {
    this.orderStatus = OrderStatus.ORDERED;
  }
}
```

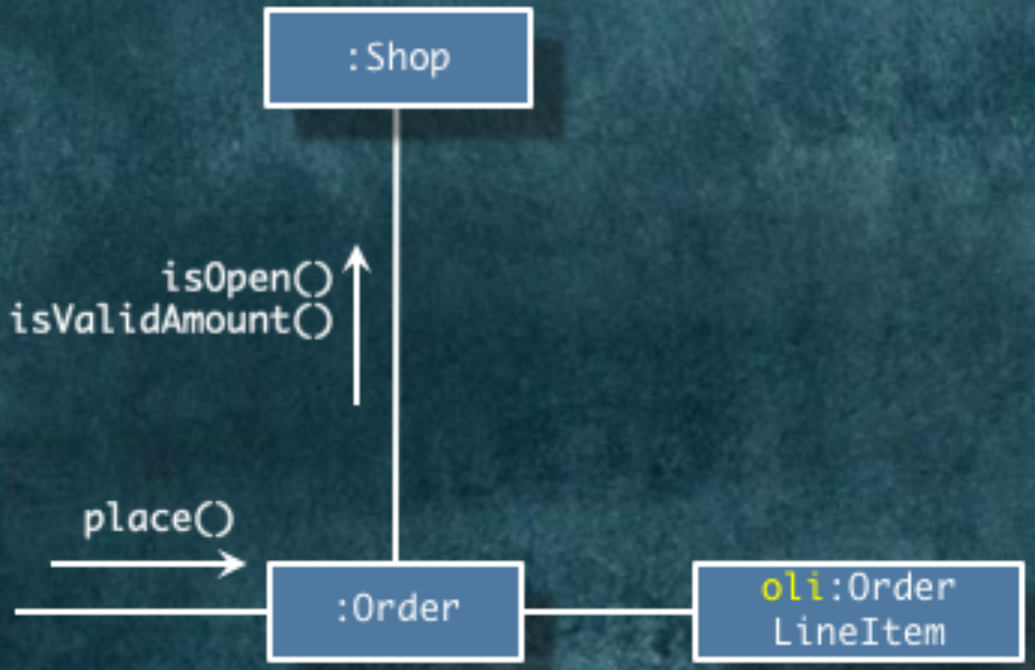
Shop validation

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

```
public class Order {
  public void place() {
    validate();
    ordered();
  }

  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format(
                "최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : or
      orderLineItem.validate();
    }
  }

  private void ordered() {
    this.orderStatus = OrderStatus.ORDERE
  }
}
```

```
public class Shop {
  public boolean isOpen() {
      return this.open;
  }

  public boolean isValidOrderAmount(Money amount) {
      return amount.isGreaterThanOrEqual(minOrderAmount);
  }
}
```

객체 협력



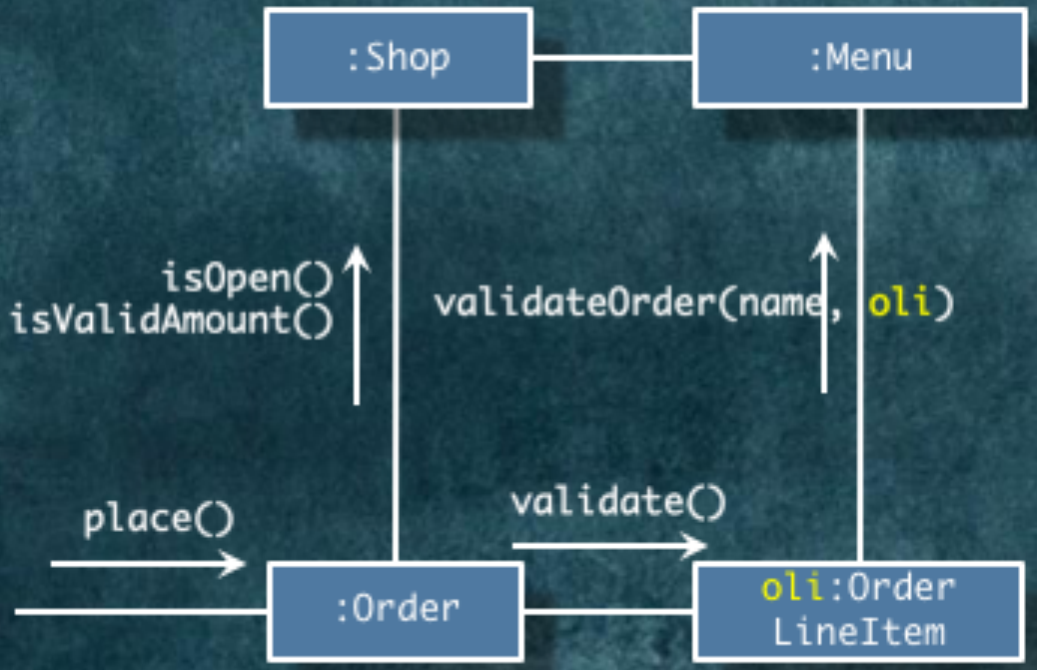
구현하기

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

```
public class Order {
  public void place() {
    validate();
    ordered();
  }

  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format(
                "최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }

  private voi
    this.orde
  }
}
```

```
public class OrderLineItem {
  public void validate() {
    menu.validateOrder(name, this.orderOptionGroups);
  }
}
```

객체 협력

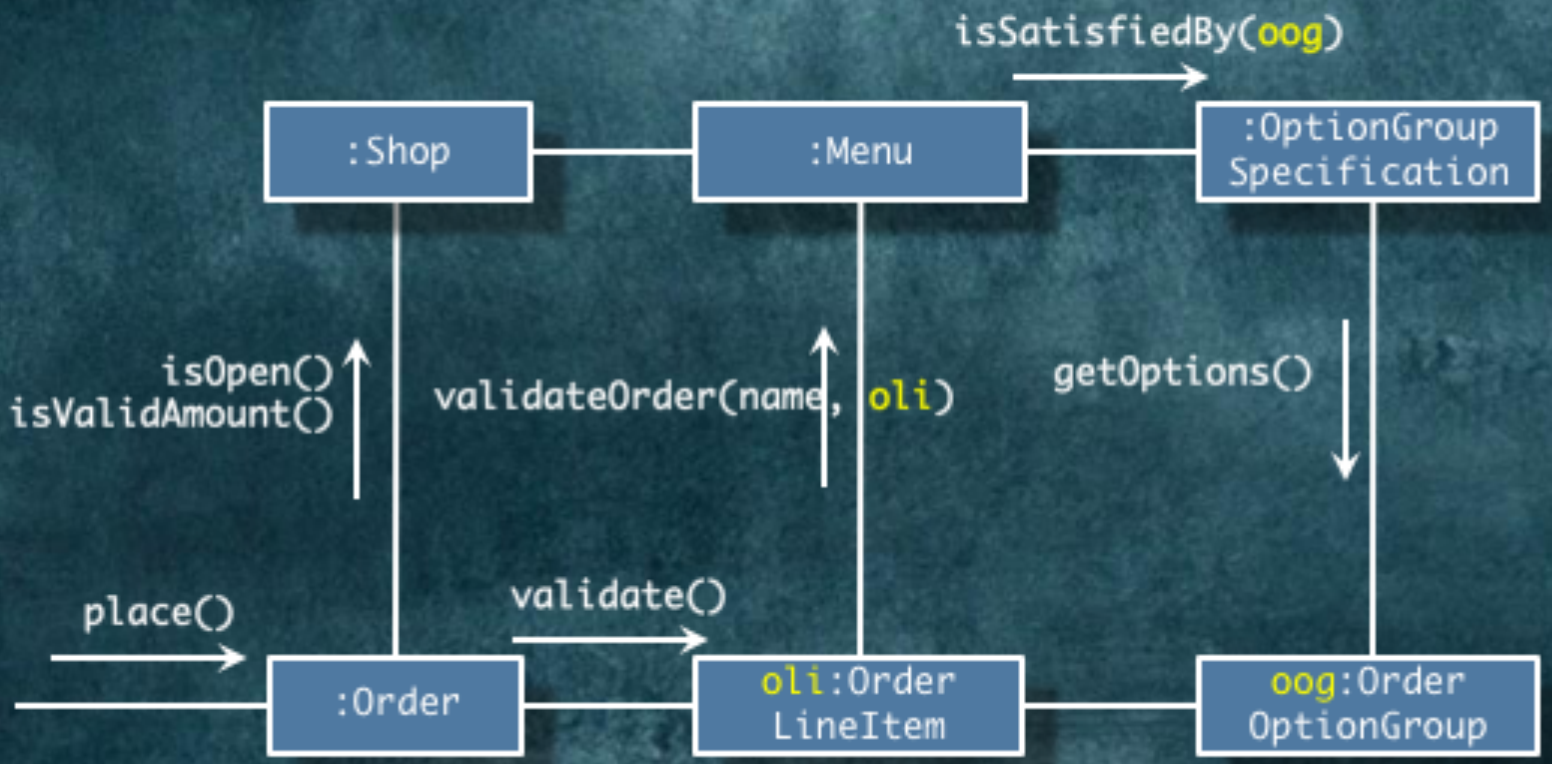

:Shop
:Menu
isOpen()
isValidAmount()
validateOrder(name, oli)
place()
validate()
:Order
oli:Order
LineItem

구현하기

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

```
public class Menu {
  public void validateOrder(String menuName,
                            List<OrderOptionGroup> groups) {
    if (!this.name.equals(menuName)) {
        throw new IllegalArgumentException("기본 상품이 변경됐습니다.");
    }

    if (!isSatisfiedBy(groups)) {
        throw new IllegalArgumentException("메뉴가 변경됐습니다.");
    }
  }

  private boolean isSatisfiedBy(List<OrderOptionGroup> groups) {
    return cartOptionGroups.stream().anyMatch(this::isSatisfiedBy);
  }

  private boolean isSatisfiedBy(OrderOptionGroup group) {
    return optionGroupSpecs.stream().anyMatch(spec -> spec.isSatisfiedBy(group));
  }
}
```

객체 협력


isSatisfiedBy(oog)
:Shop
:Menu
:OptionGroup Specification
isOpen()
isValidAmount()
validateOrder(name, oli)
getOptions()
place()
validate()
:Order
oli:Order LineItem
oog:Order OptionGroup

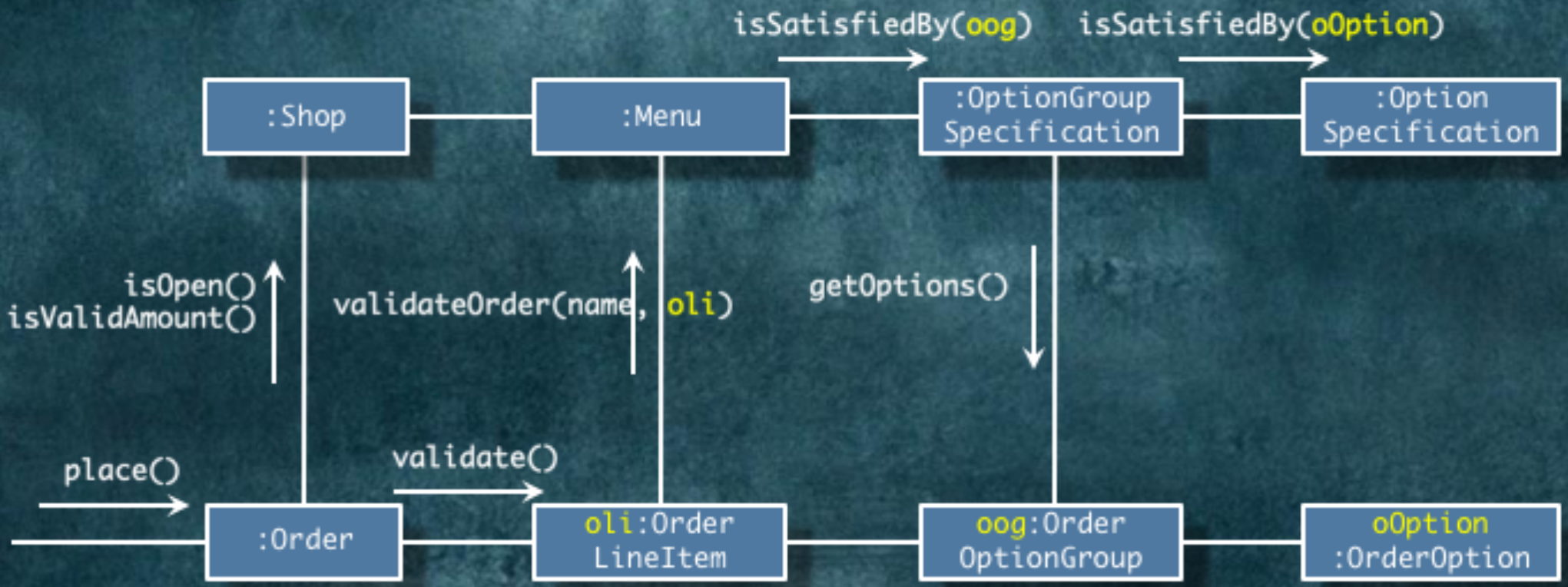
구현하기

- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

```
public class Menu {
  public void validateOrder(String menuName,
                            List<OrderOptionGroup> groups) {
    if (!this.name.equals(menuName)) {

    }

    if

    }
  }

  priv
    re
  }

  priv
    re
  }
}
```

```
public class OptionGroupSpecification {
  public boolean isSatisfiedBy(OrderOptionGroup group) {
    return !isSatisfied(group.getName(), satisfied(group.getOptions()));
  }

  private boolean isSatisfied(String groupName, List<OrderOption> satisfied) {
    if (!name.equals(groupName) ||
        satisfied.isEmpty() ||
        (exclusive && satisfied.size() > 1)) {
        return false;
    }

    return true;
  }

  private List<Option> satisfied(List<OrderOption> options) {
    return optionSpecs
              .stream()
              .flatMap(spec -> options.stream().filter(spec::isSatisfiedBy))
              .collect(toList());
  }
}
```

객체 협력


isSatisfiedBy(oog)
isSatisfiedBy(oOption)
:Shop
:Menu
:OptionGroup Specification
:Option Specification
isOpen()
isValidAmount()
validateOrder(name, oli)
getOptions()
place()
validate()
:Order
oli:Order LineItem
oog:Order OptionGroup
oOption :OrderOption

구현하기

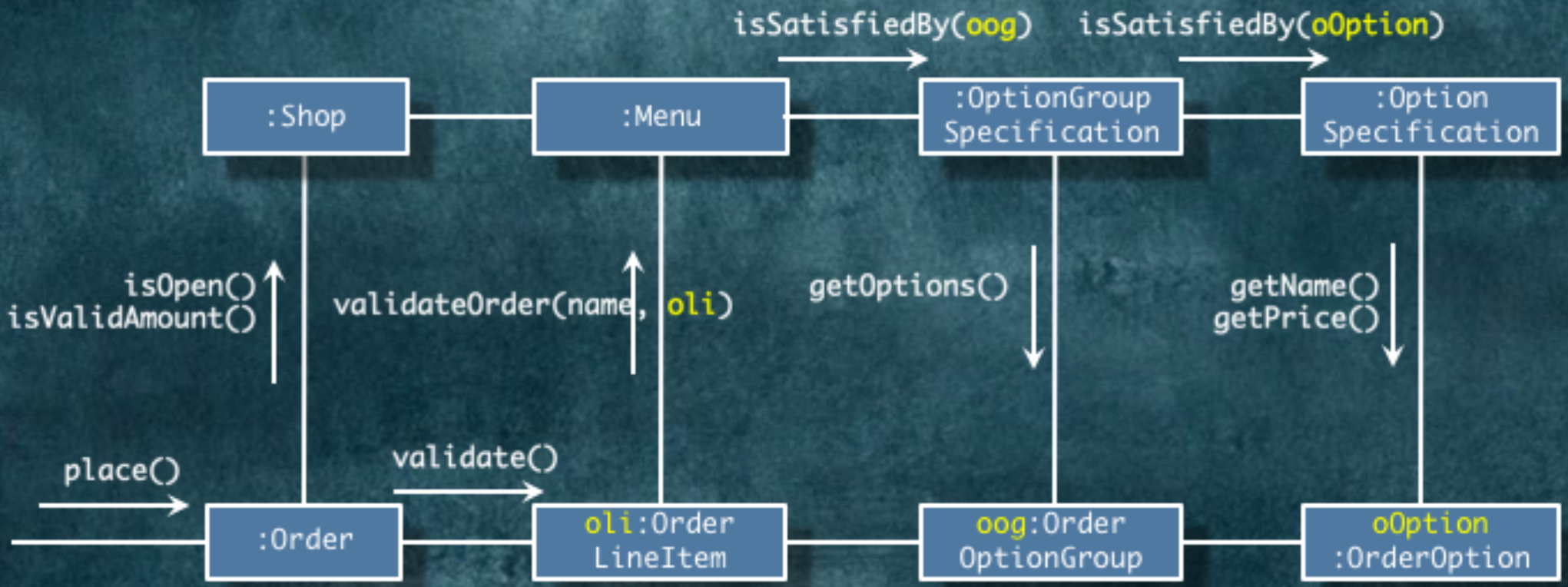
- 메뉴의 이름과 주문항목의 이름 비교
- 옵션그룹의 이름과 주문옵션그룹의 이름 비교
- 옵션의 이름과 주문옵션의 이름 비교
- 옵션의 가격과 주문옵션의 가격 비교
- 가게가 영업중인지 확인
- 주문금액이 최소주문금액 이상인지

```
public class Menu {
  public void validateOrder(String menuName,
                            List<OrderOptionGroup> groups) {
    if (!this.name.equals(menuName)) {

    }

    if

    }
  }

  priv
    re
  }

  priv
    re
  }
}
```

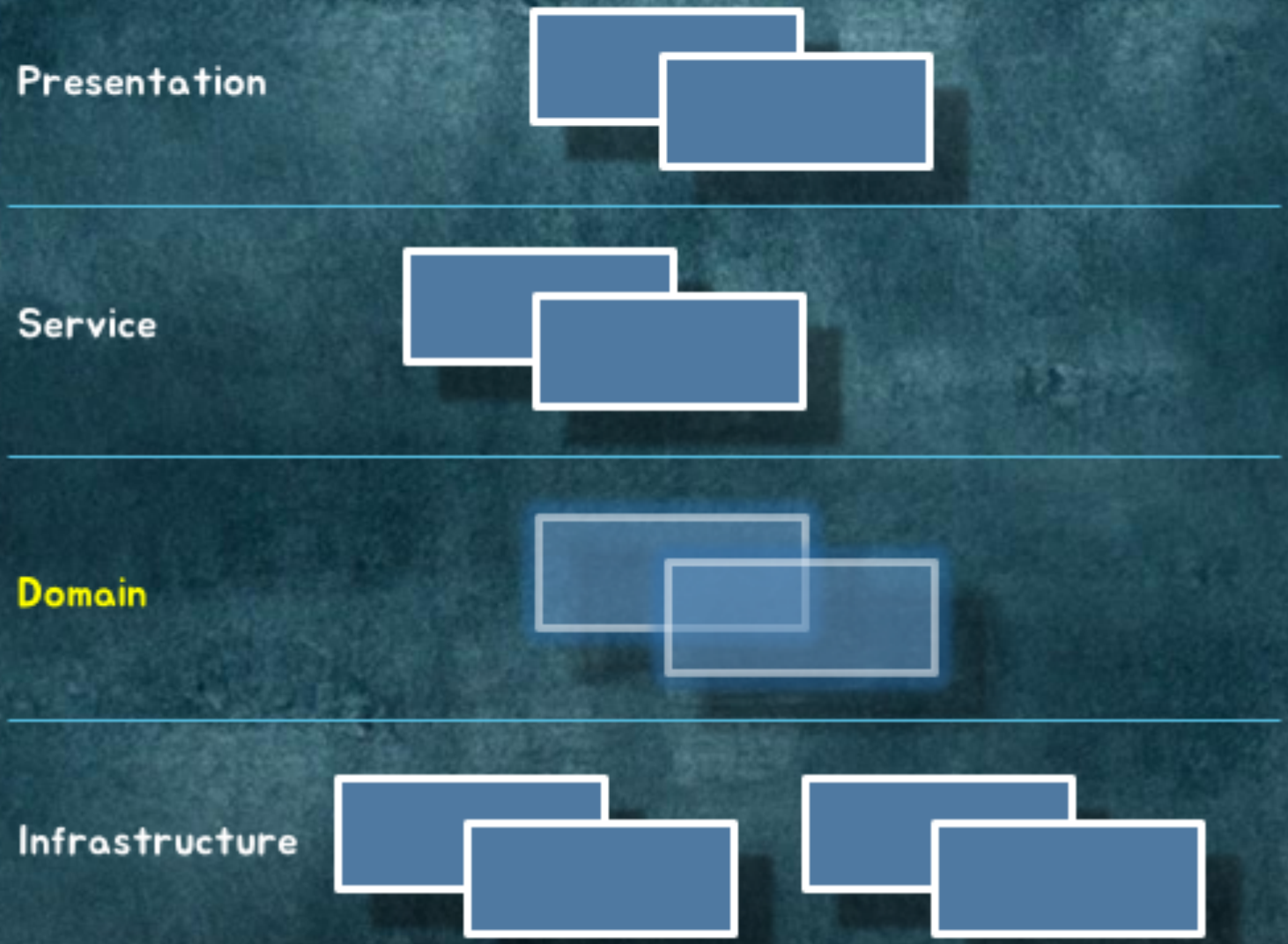
```
public class OptionGroupSpecification {
  public boolean isSatisfiedBy(OrderOptionGroup group) {
    return !isSatisfied(group.getName(), satisfied(group.getOptions()));
  }

  private boolean isSatisfied(String groupName, List<OrderOption> satisfied) {
    if (!name.equals(groupName) ||
        satisfied.isEmpty() ||
        (exclusive && satisfied.size() > 1)) {
        return false;
    }

    return true;
  }

  private List<Option> satisfied(List<OrderOption> options) {
    return optionSpecs
              .stream()
              .flatMap(spec -> options.stream().filter(spec::isSatisfiedBy))
              .collect(toList());
  }
}
```

```
public class OptionSpecification {
  public boolean isSatisfiedBy(OrderOption option) {
    return Objects.equals(name, option.getName()) &&
           Objects.equals(price, option.getPrice());
  }
}
```

객체 협력


isSatisfiedBy(oog)
isSatisfiedBy(oOption)
:Shop
:Menu
:OptionGroup Specification
:Option Specification
isOpen()
isValidAmount()
validateOrder(name, oli)
getOptions()
getName()
getPrice()
place()
validate()
:Order
oli:Order LineItem
oog:Order OptionGroup
oOption :OrderOption

레이어 아키텍처(Layered Architecture)



레이어 아키텍처(Layered Architecture)

Presentation

Service

Domain

Infrastructure

```java
@Service
public class OrderService {
  @Transactional
  public void placeOrder(Cart cart) {
    Order order = orderMapper.mapFrom(cart);
    order.place();
    orderRepository.save(order);
  }
}
```

```java
@Entity
@Table(name="ORDERS")
public class Order {
}
```

```java
public interface OrderRepository
    extends JpaRepository<Order, Long>
{
}
```

```java
@Repository
public class OrderRepositoryImpl
    implements OrderRepository {
}
```

Part - 3

설계 개선하기

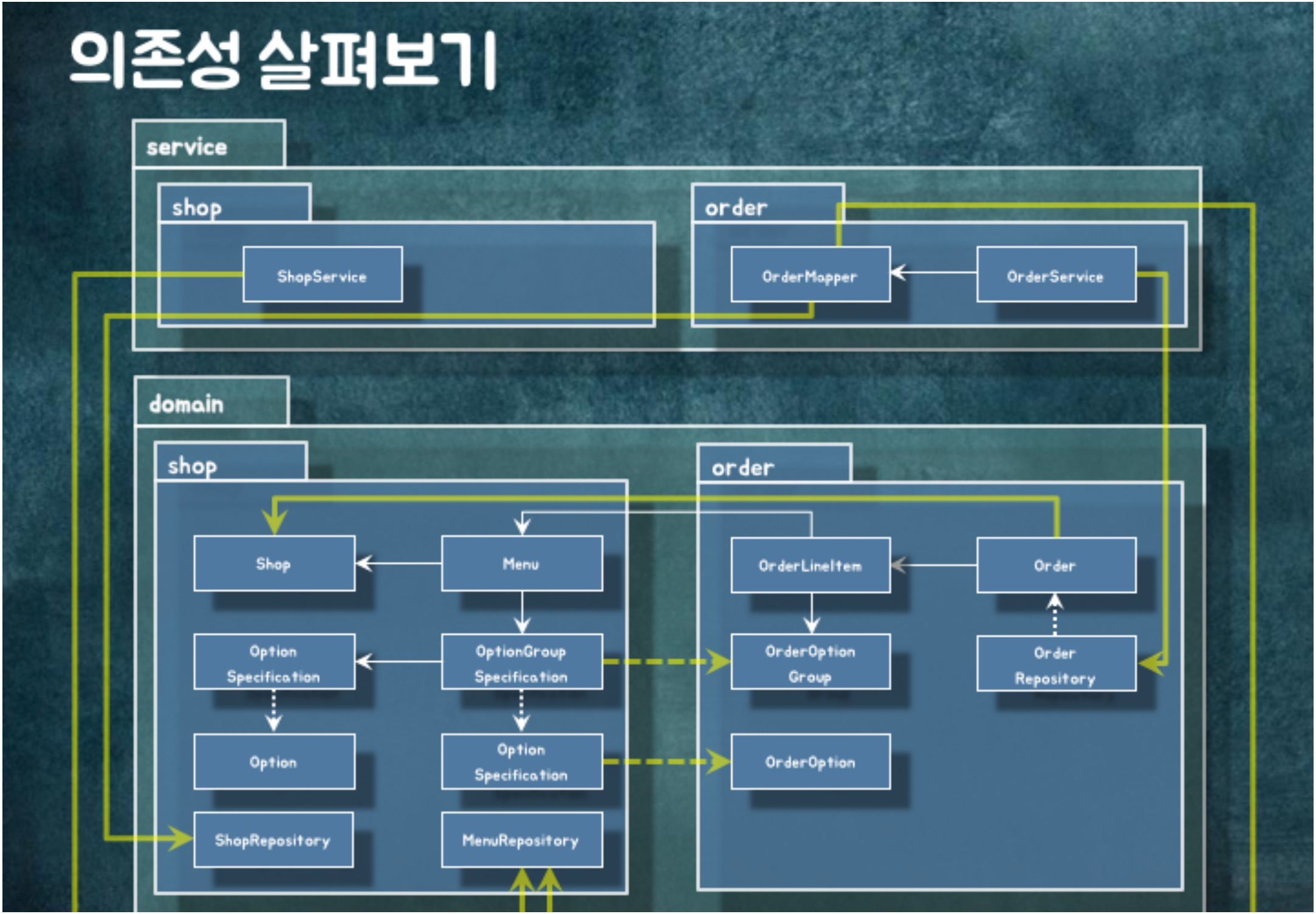

설계를 진화 시키기
위한 출발점
코드 작성 후
의존성 관점에서
설계 검토

두 가지 문제

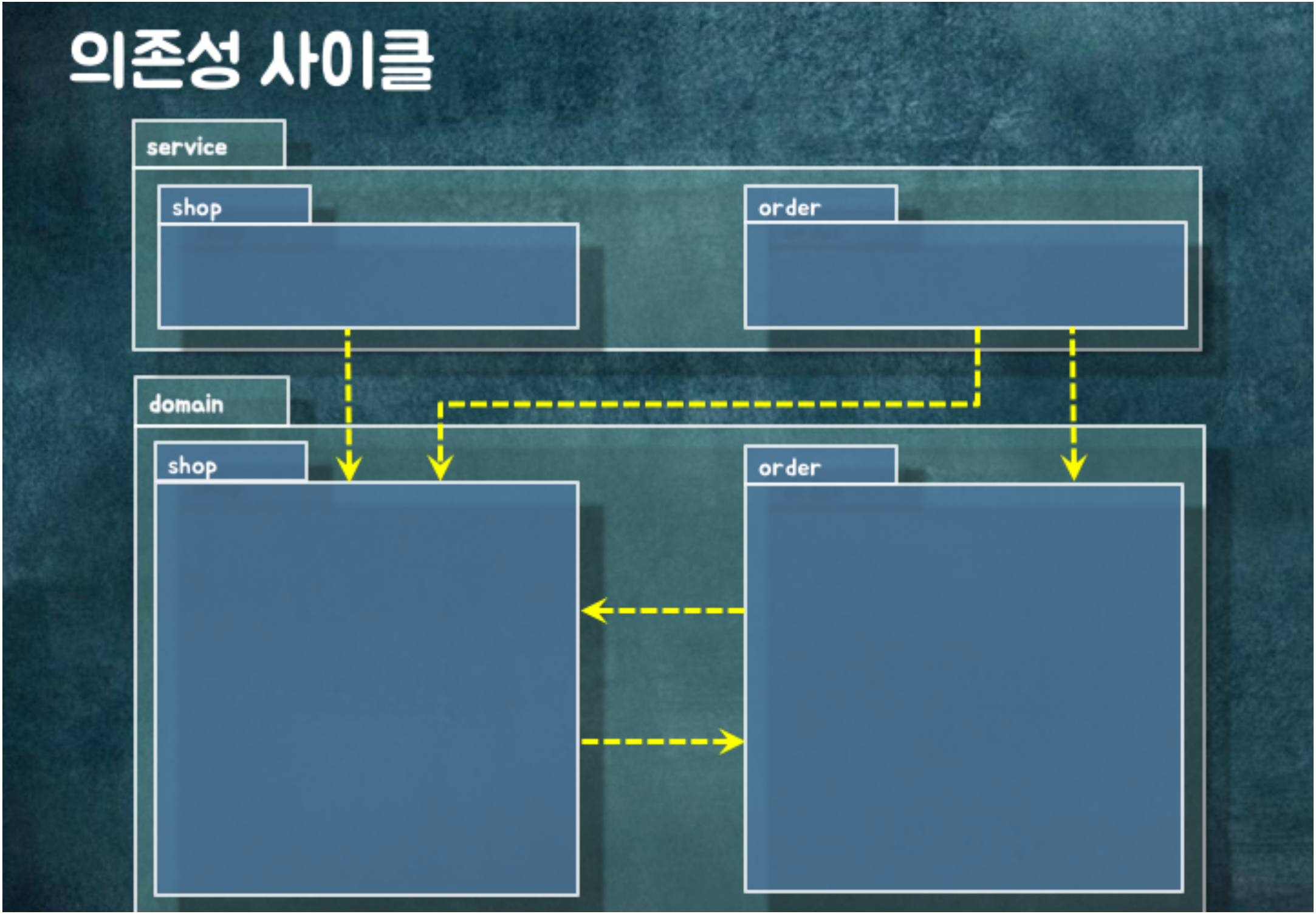
객체 참조로 인한 결합도 상승

패키지 의존성 사이클

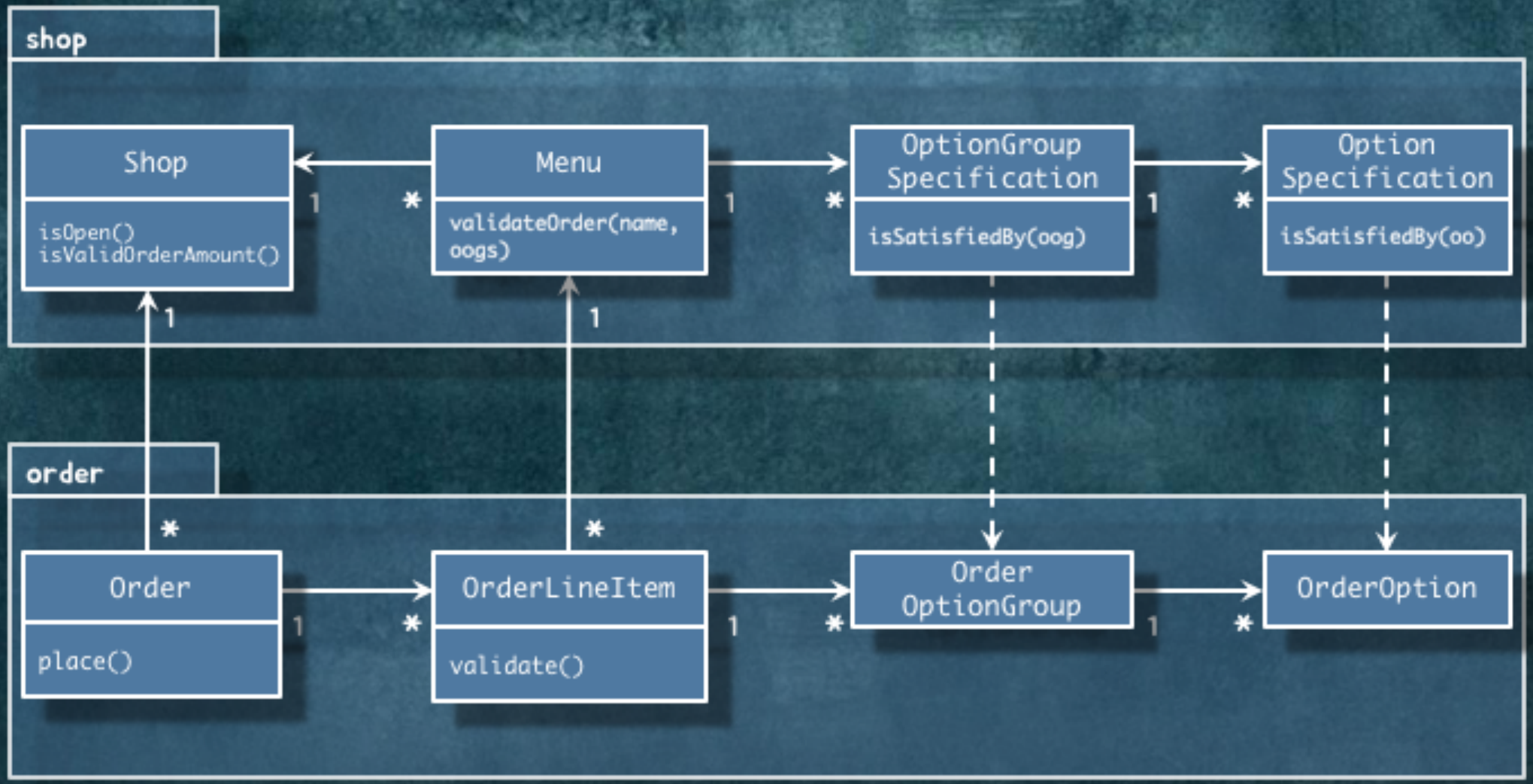
의존성 살펴보기


service
shop
ShopService
order
OrderMapper
OrderService
domain
shop
Shop
Menu
Option Specification
OptionGroup Specification
Option
Option Specification
ShopRepository
MenuRepository
order
OrderLineItem
Order
OrderOption Group
Order Repository
OrderOption

의존성 사이클



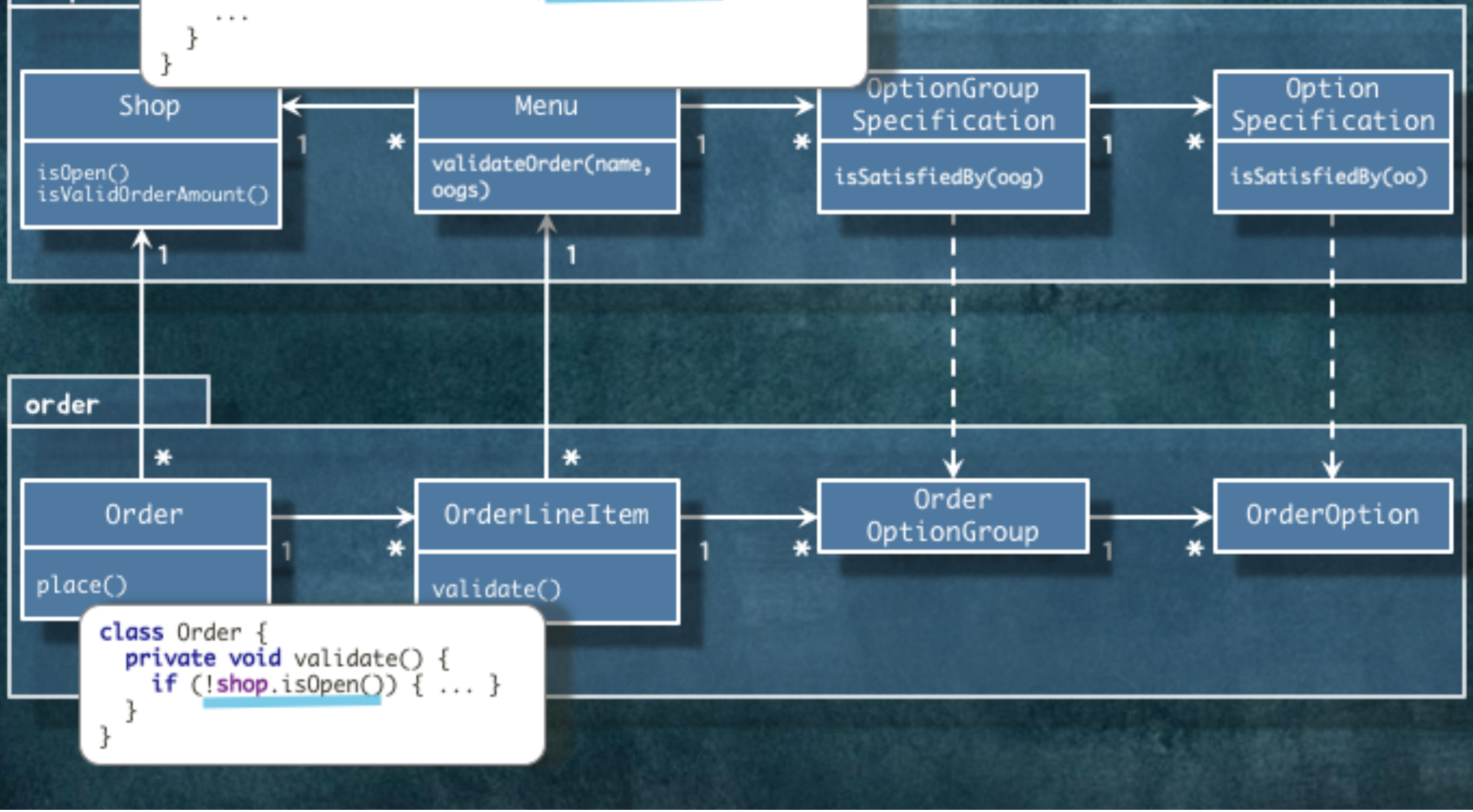
무엇이 문제인가?


shop
Shop
isOpen()
isValidOrderAmount()
Menu
validateOrder(name, oogs)
OptionGroup Specification
isSatisfiedBy(oog)
Option Specification
isSatisfiedBy(oo)
order
Order
place()
OrderLineItem
validate()
Order OptionGroup
OrderOption
1
*

무엇이 문제인가?

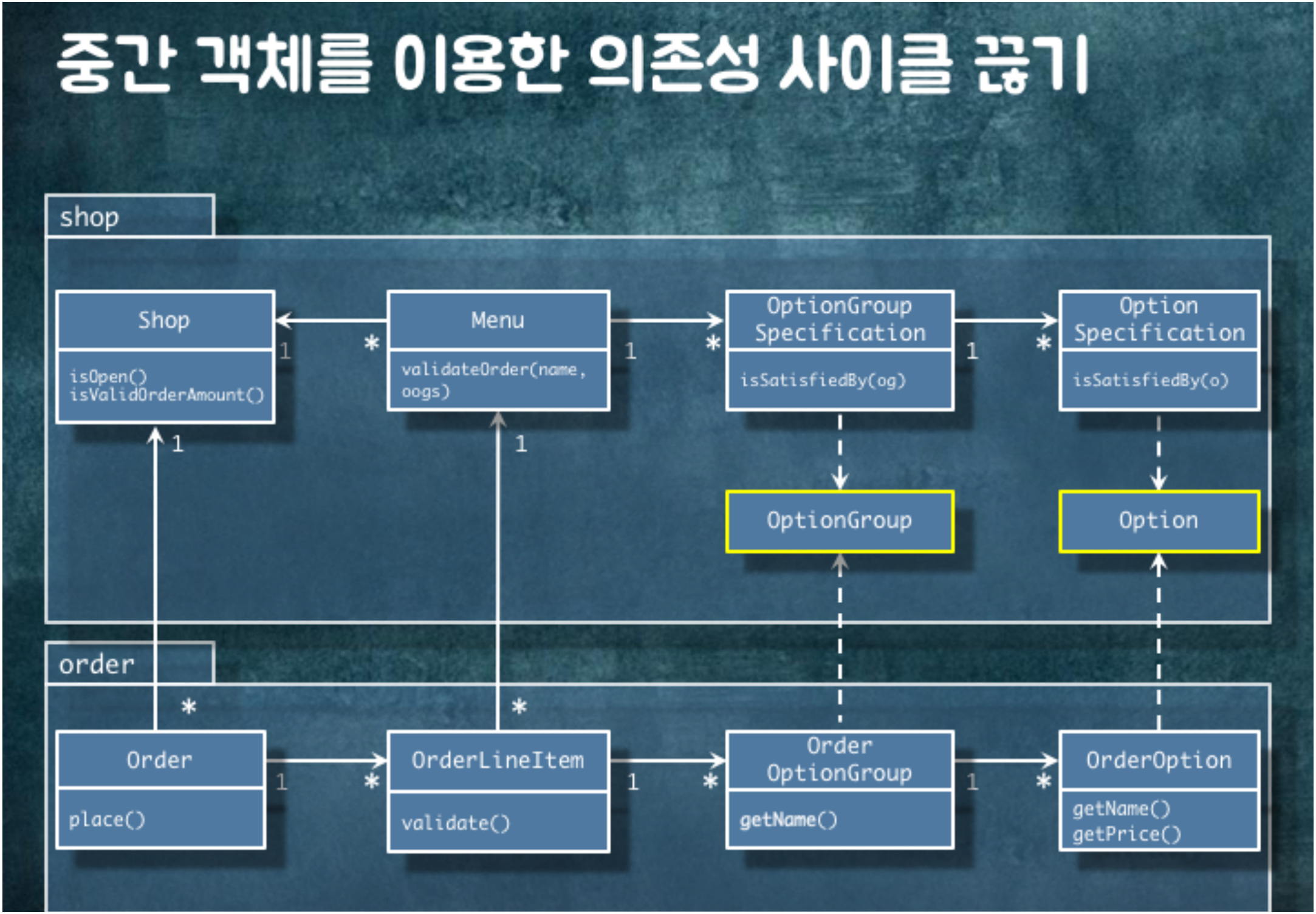
```
class OptionGroupSpecification {
  public boolean isSatisfiedBy(OrderOptionGroup group) {
    ...
  }
}
```

```
class OptionGroupSpecification {
  public boolean isSatisfiedBy(OrderOption option) {
    ...
  }
}
```

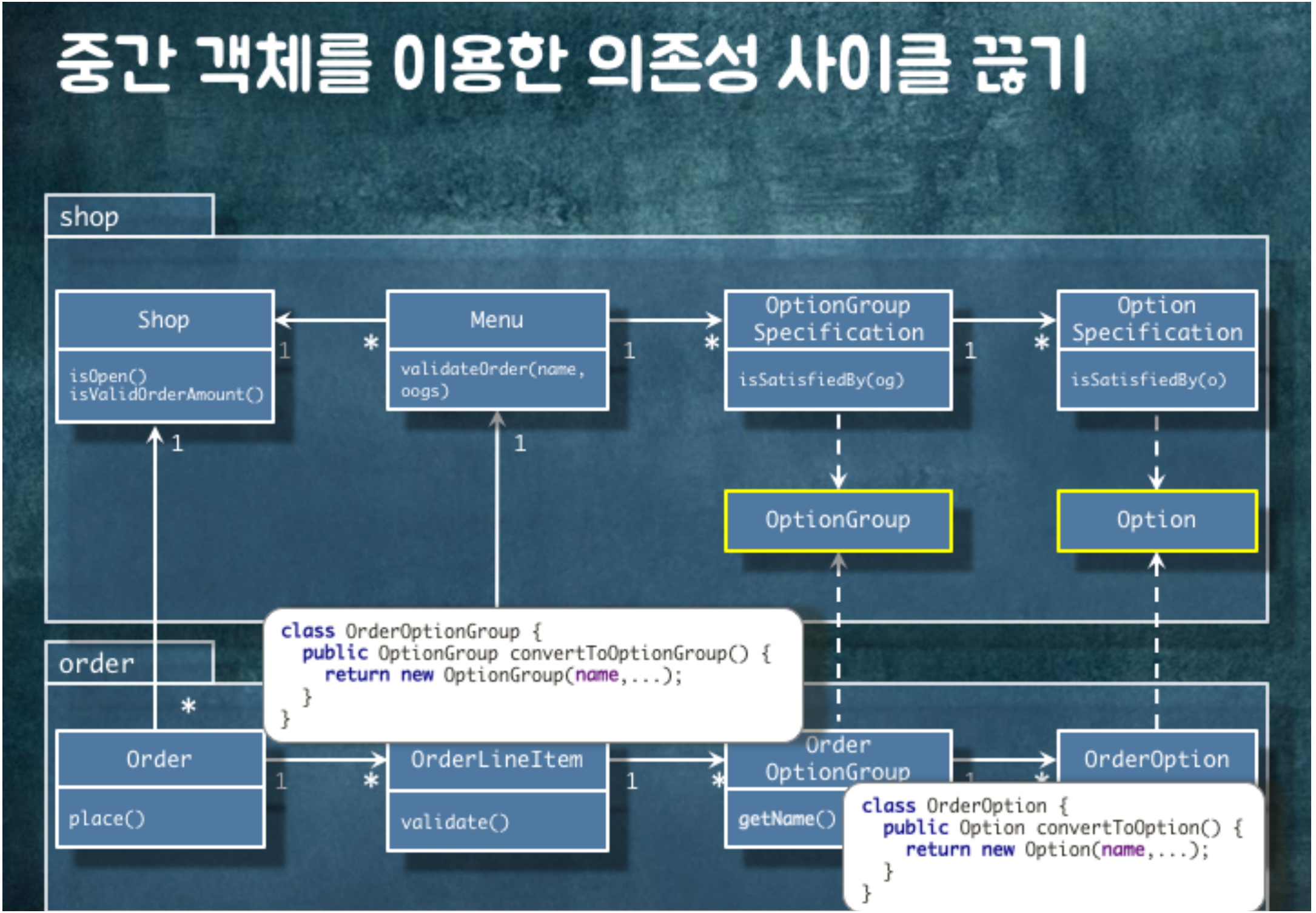



```
class Order {
  private void validate() {
    if (!shop.isOpen()) { ... }
  }
}
```

중간 객체를 이용한 의존성 사이클 끊기



중간 객체를 이용한 의존성 사이클 끊기



중간 객체를 이용한 의존성 사이클 끊기

```
class OptionGroupSpecification {
  public boolean isSatisfiedBy(~~OrderOptionGroup group~~
                               OptinGroup group) {
    ...
  }
}
```

```
class OptionGroupSpecification {
  public boolean isSatisfiedBy(~~OrderOption option~~
                               Option option) {
    ...
  }
}
```

isOpen()
isValidOrderAmount()

validateOrder(name, oogs)

OptionGroup Specification
isSatisfiedBy(og)

1 * 1

Option Specification
isSatisfiedBy(o)

1 1

OptionGroup

Option

```
class OrderOptionGroup {
  public OptionGroup convertToOptionGroup() {
    return new OptionGroup(name,...);
  }
}
```

order

* *

Order
place()

1 * 1 *

OrderLineItem
validate()

Order OptionGroup
getName()

OrderOption

```
class OrderOption {
  public Option convertToOption() {
    return new Option(name,...);
  }
}
```

연관관계 다시 살펴보기

Order에서
OrderLineItem으로
탐색가능

Order가
뭔지 알면

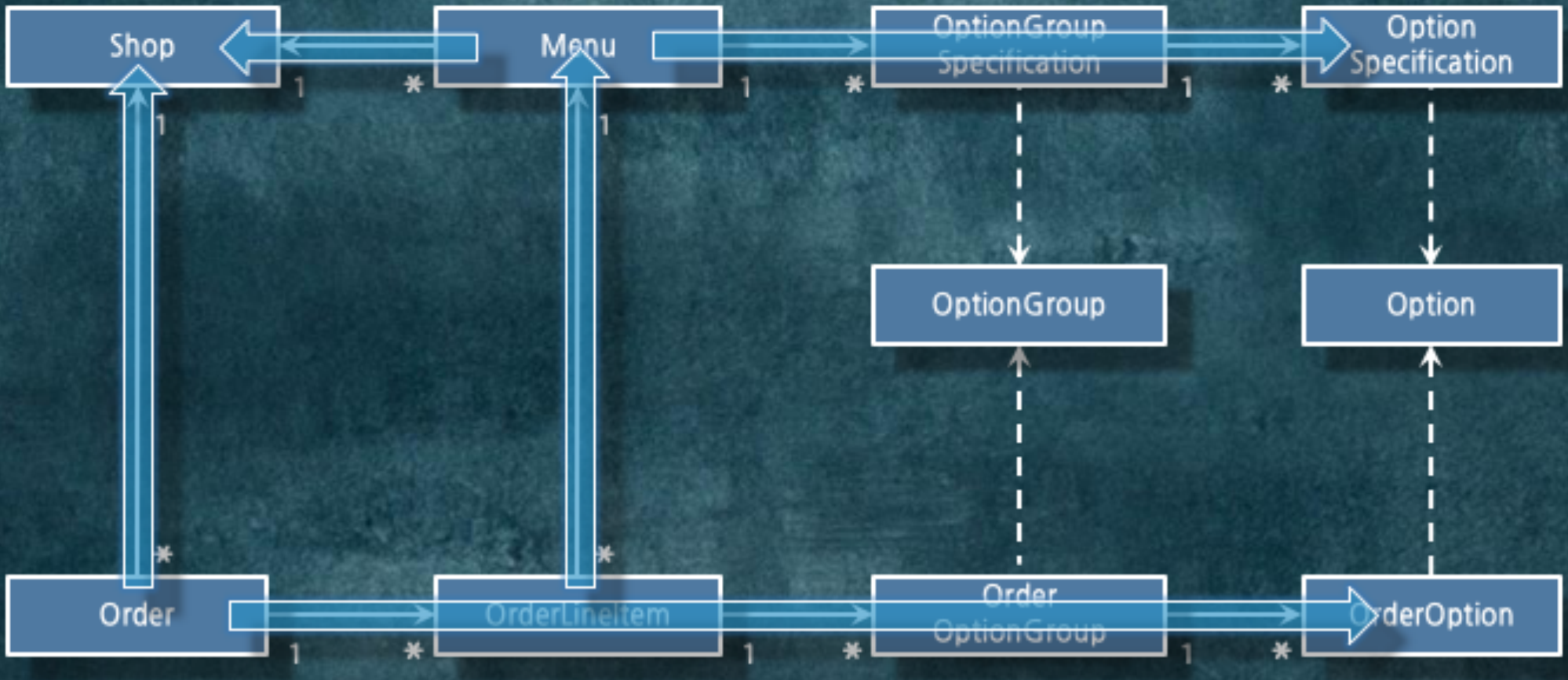
Order를 통해 원하는
OrderLineItem을
찾을 수있다.

객체 참조로 구현한 연관관계의 문제점

협력을 위해 필요하지만
두 객체 사이의 **결합도가 높아짐**


```
class Order {
  private List<OrderLineItem> orderLineItems;

  public void place() {
    validate();
    ordered();
  }

  private void validate() {
    for (OrderLineItem orderLineItem : orderLineItems) {
        orderLineItem.validate();
    }
  }
}
```

성능 문제 - 어디까지 조회할 것인가?



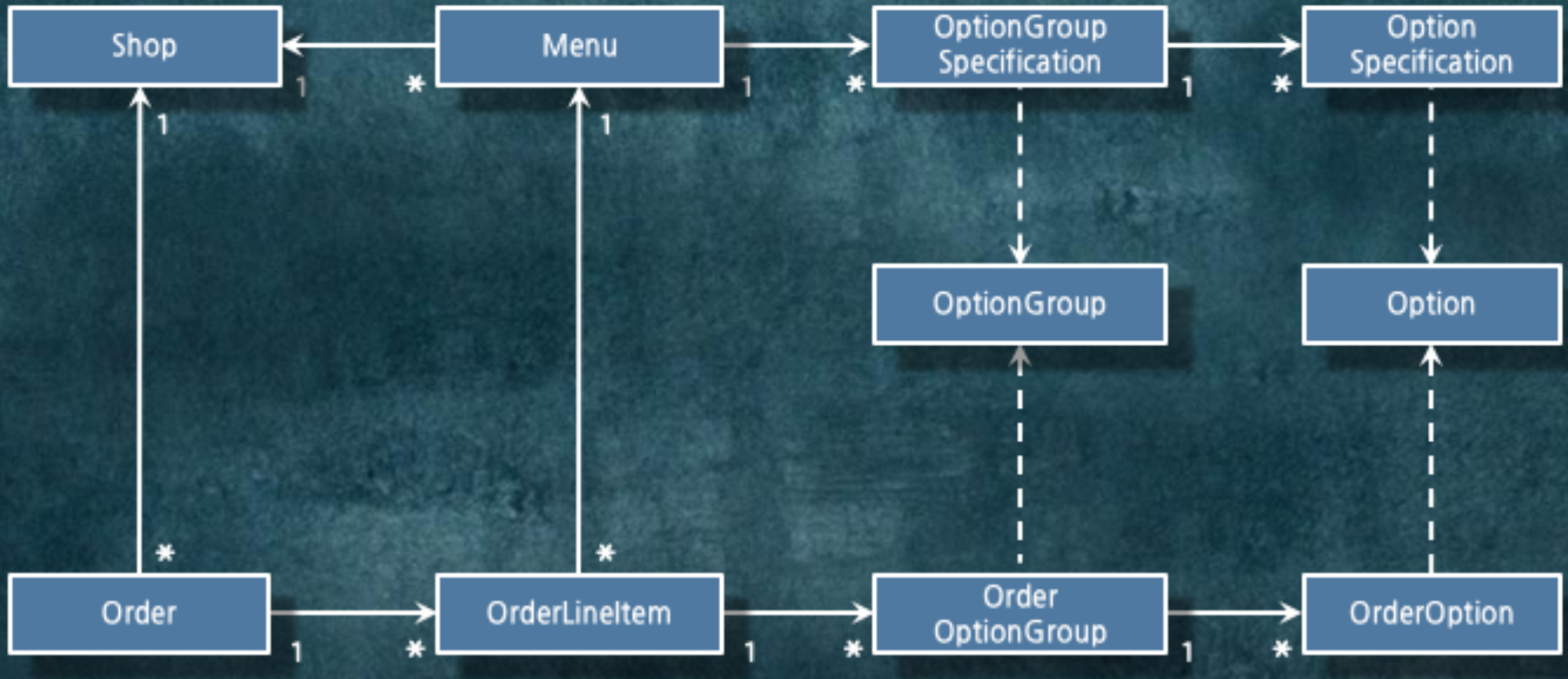
성능 문제 - 객체 그룹의 조회 경계가 모호


```
Hibernate:
    select
        order0_.ORDER_ID as ORDER_ID1_9_0_,
        order0_.STATUS as STATUS2_9_0_,
        order0_.ORDERED_TIME as ORDERED_3_9_0_,
        order0_.SHOP_ID as SHOP_ID5_9_0_,
        order0_.USER_ID as USER_ID4_9_0_,
        shop1_.SHOP_ID as SHOP_ID1_10_1_,
        shop1_.COMMISSION as COMMISSI2_10_1_,
        shop1_.COMMISSION_RATE as COMMISSI3_10_1_,
        shop1_.MIN_ORDER_AMOUNT as MIN_ORDE4_10_1_,
        shop1_.NAME as NAME5_10_1_,
        shop1_.OPEN as OPEN6_10_1_
    from
        ORDERS order0_
    left outer join
        SHOPS shop1_
            on order0_.SHOP_ID=shop1_.SHOP_ID
    where
        order0_.ORDER_ID=?
Hibernate:
    select
        orderlinei0_.ORDER_ID as ORDER_ID5_6_0_,
        orderlinei0_.ORDER_LINE_ITEM_ID as ORDER_LI1_6_0_,
        orderlinei0_.ORDER_LINE_ITEM_ID as ORDER_LI1_6_1_,
        orderlinei0_.FOOD_COUNT as FOOD_COU2_6_1_,
        orderlinei0_.MENU_ID as MENU_ID4_6_1_,
        orderlinei0_.FOOD_NAME as FOOD_NAM3_6_1_,
        menu1_.MENU_ID as MENU_ID1_3_2_,
        menu1_.FOOD_DESCRIPTION as FOOD_DES2_3_2_,
        menu1_.FOOD_NAME as FOOD_NAM3_3_2_,
        shop2_.SHOP_ID as SHOP_ID1_10_3_,
        shop2_.COMMISSION as COMMISSI2_10_3_,
        shop2_.COMMISSION_RATE as COMMISSI3_10_3_,
        shop2_.MIN_ORDER_AMOUNT as MIN_ORDE4_10_3_,
        shop2_.NAME as NAME5_10_3_,
        shop2_.OPEN as OPEN6_10_3_
    from
        ORDER_LINE_ITEMS orderlinei0_
    left outer join
        MENUS menu1_
            on orderlinei0_.MENU_ID=menu1_.MENU_ID
    left outer join
        SHOPS shop2_
            on menu1_.MENU_ID=shop2_.SHOP_ID
    where
        orderlinei0_.ORDER_ID=?
Hibernate:
    select
        optiongrou0_.MENU_ID as MENU_ID5_4_0_,
        optiongrou0_.OPTION_GROUP_SPEC_ID as OPTION_G1_4_0_,
        optiongrou0_.OPTION_GROUP_SPEC_ID as OPTION_G1_4_1_,
        optiongrou0_.BASIC as BASIC2_4_1_,
        optiongrou0_.EXCLUSIVE as EXCLUSIV3_4_1_,
        optiongrou0_.NAME as NAME4_4_1_
    from
        OPTION_GROUP_SPECS optiongrou0_
    where
        optiongrou0_.MENU_ID=?
```

```
Hibernate:
    select
        optionspec0_.OPTION_GROUP_SPEC_ID as OPTION_G4_5_0_,
        optionspec0_.OPTION_SPEC_ID as OPTION_S1_5_0_,
        optionspec0_.OPTION_SPEC_ID as OPTION_S1_5_1_,
        optionspec0_.NAME as NAME2_5_1_,
        optionspec0_.PRICE as PRICE3_5_1_
    from
        OPTION_SPECS optionspec0_
    where
        optionspec0_.OPTION_GROUP_SPEC_ID=?
Hibernate:
    select
        optionspec0_.OPTION_GROUP_SPEC_ID as OPTION_G4_5_0_,
        optionspec0_.OPTION_SPEC_ID as OPTION_S1_5_0_,
        optionspec0_.OPTION_SPEC_ID as OPTION_S1_5_1_,
        optionspec0_.NAME as NAME2_5_1_,
        optionspec0_.PRICE as PRICE3_5_1_
    from
        OPTION_SPECS optionspec0_
    where
        optionspec0_.OPTION_GROUP_SPEC_ID=?
Hibernate:
    select
        optionspec0_.OPTION_GROUP_SPEC_ID as OPTION_G4_5_0_,
        optionspec0_.OPTION_SPEC_ID as OPTION_S1_5_0_,
        optionspec0_.OPTION_SPEC_ID as OPTION_S1_5_1_,
        optionspec0_.NAME as NAME2_5_1_,
        optionspec0_.PRICE as PRICE3_5_1_
    from
        OPTION_SPECS optionspec0_
    where
        optionspec0_.OPTION_GROUP_SPEC_ID=?
Hibernate:
    select
        groups0_.ORDER_LINE_ITEM_ID as ORDER_LI3_7_0_,
        groups0_.ORDER_OPTION_GROUP_ID as ORDER_OP1_7_0_,
        groups0_.ORDER_OPTION_GROUP_ID as ORDER_OP1_7_1_,
        groups0_.NAME as NAME2_7_1_
    from
        ORDER_OPTION_GROUPS groups0_
    where
        groups0_.ORDER_LINE_ITEM_ID=?
Hibernate:
    select
        orderoptio0_.ORDER_OPTION_GROUP_ID as ORDER_OP1_8_0_,
        orderoptio0_.NAME as NAME2_8_0_,
        orderoptio0_.PRICE as PRICE3_8_0_
    from
        ORDER_OPTIONS orderoptio0_
    where
        orderoptio0_.ORDER_OPTION_GROUP_ID=?
```

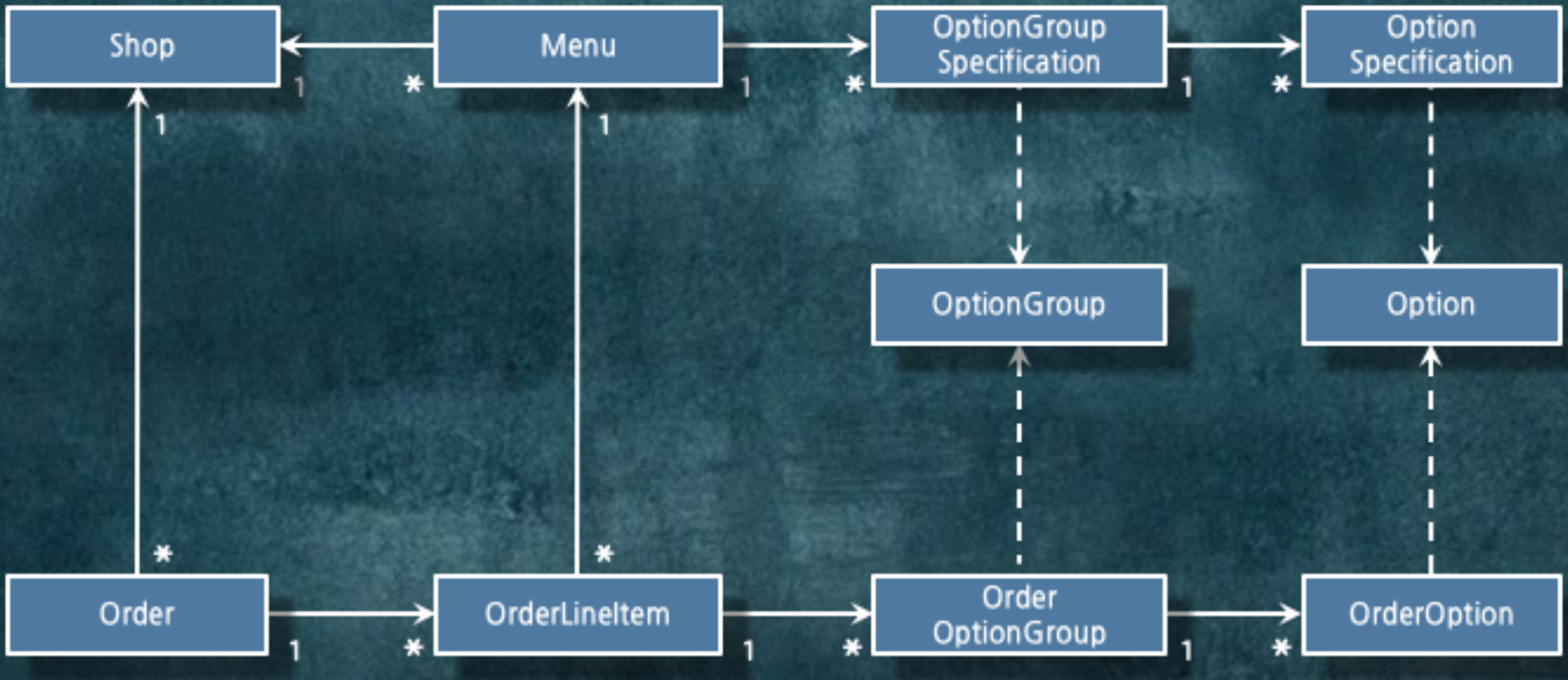
수정 시 도메인 규칙을 함께 적용할 경계는?

Order의 상태를 변경할 때 연관된 도메인 규칙을
함께 적용해야하는 객체의 범위는?



(다른말로) 트랜잭션 경계는 어디까지인가?

어떤 테이블에서 어떤 테이블까지
하나의 단위로 잠금Lock을 설정할 것인가?



결제 완료

```
public class OrderService {
  @Transactional
  public void payOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.payed();

    Delivery delivery = Delivery.started(order);
    deliveryRepository.save(delivery);
  }
}
```

결제 완료

```
public class OrderService {
  @Transactional
  public void payOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.payed();

    Delivery delivery = Delivery.started(order);
    deliveryRepository.save(delivery);
  }
}
```

```
public class Order {
  public enum OrderStatus {
    ORDERED, PAYED, DELIVERED
  }

  @Enumerated(EnumType.STRING)
  @Column(name="STATUS")
  private OrderStatus orderStatus;

  public void payed() {
    this.orderStatus = PAYED;
  }
}
```

결제 완료

```
public class OrderService {
  @Transactional
  public void payOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.payed();

    Delivery delivery = Delivery.started(order);
    deliveryRepository.save(delivery);
  }
}
```

```
public class Order {
  public enum OrderStatus {
    ORDERED, PAYED, DELIVERED
  }

  @Enumerated(EnumType.STRING)
  @Column(name="STATUS")
  private OrderStatus orderStatus;

  public void payed() {
    this.orderStatus = PAYED;
  }
}
```

```
@Entity
@Table(name="DELIVERIES")
public class Delivery {
    enum DeliveryStatus { DELIVERING, DELIVERED }

    @OneToOne
    @JoinColumn(name="ORDER_ID")
    private Order order;

    @Enumerated(EnumType.STRING)
    @Column(name="STATUS")
    private DeliveryStatus deliveryStatus;

    public static Delivery started(Order order) {
        return new Delivery(order, DELIVERING);
    }
}
```

배달 완료

```
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.delivered();

    Delivery delivery = deliveryRepository.findById(orderId) ...
    delivery.complete();
  }
}
```

배달 완료

```
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.delivered();

    Delivery delivery = deliveryRepository.findById(orderId) ...
    delivery.complete();
  }
}
```

```
public class Order {
  public void delivered() {
    this.orderStatus = DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

배달 완료

```
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.delivered();

    Delivery delivery = deliveryRepository.findById(orderId) ...
    delivery.complete();
  }
}
```

```
public class Order {
  public void delivered() {
    this.orderStatus = DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

```
public class Shop {
  private Ratio commissionRate;
  private Money commission = Money.ZERO;

  public void billCommissionFee(Money price) {
    commission = commission.plus(commissionRate.of(price));
  }
}
```

배달 완료

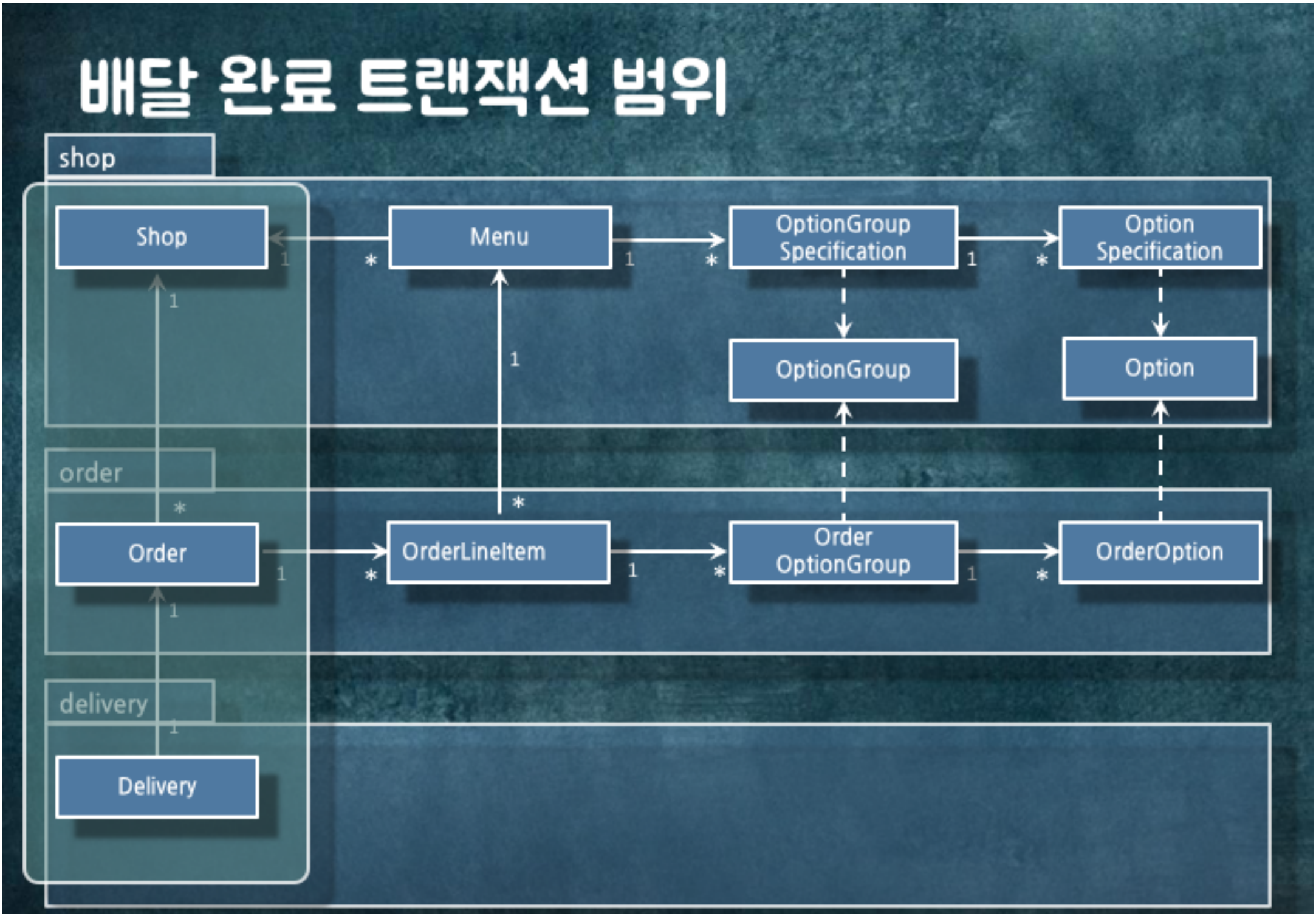
```java
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.delivered();

    Delivery delivery = deliveryRepository.findById(orderId) ...
    delivery.complete();
  }
}
```

```java
public class Order {
  public void delivered() {
    this.orderStatus = DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

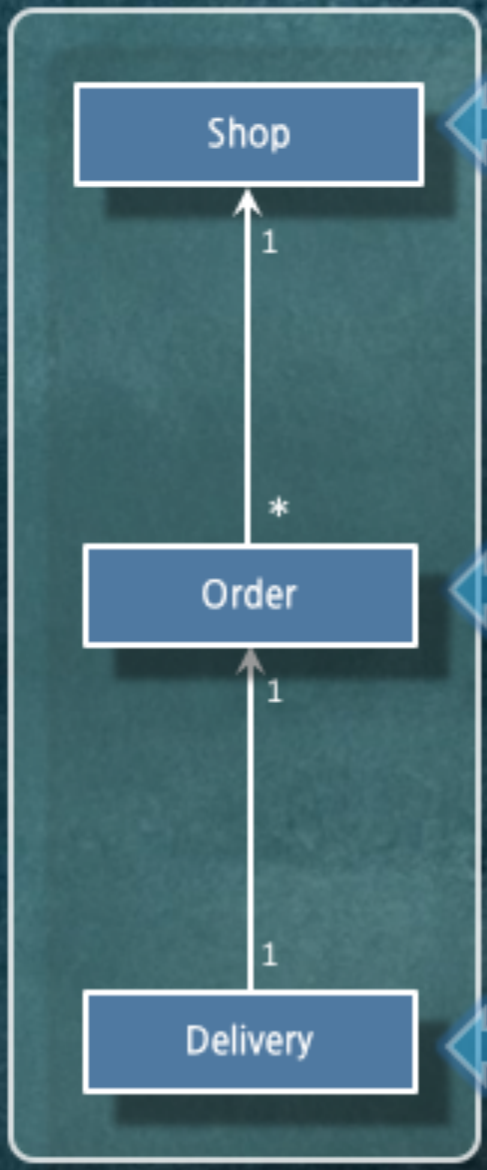
```java
public class Shop {
  private Ratio commissionRate;
  private Money commission = Money.ZERO;

  public void billCommissionFee(Money price) {
    commission = commission.plus(commissionRate.of(price));
  }
}
```

```java
public class Delivery {
  public void complete() {
    this.deliveryStatus = DELIVERED;
    this.order.completed();
  }
}
```

배달 완료 트랜잭션 범위


shop
Shop
Menu
OptionGroup Specification
Option Specification
OptionGroup
Option
order
Order
OrderLineItem
Order OptionGroup
OrderOption
delivery
Delivery
1
*

WARNING! 변경의 빈도가 다르다!



가게 상태 변경
계약 변경
(admin)

주문 상태 변경

배달 상태 변경

트랜잭션 경합으로 인한 성능 저하

가게 상태 변경
계약 변경
(admin)

주문 상태 변경

배달 상태 변경


객체참조가
꼭 필요할까?

객체 참조의 문제점

모든 객체가 연결돼 있기 때문에

```java
@Entity
@Table(name="SHOPS")
public class Shop {
}
```

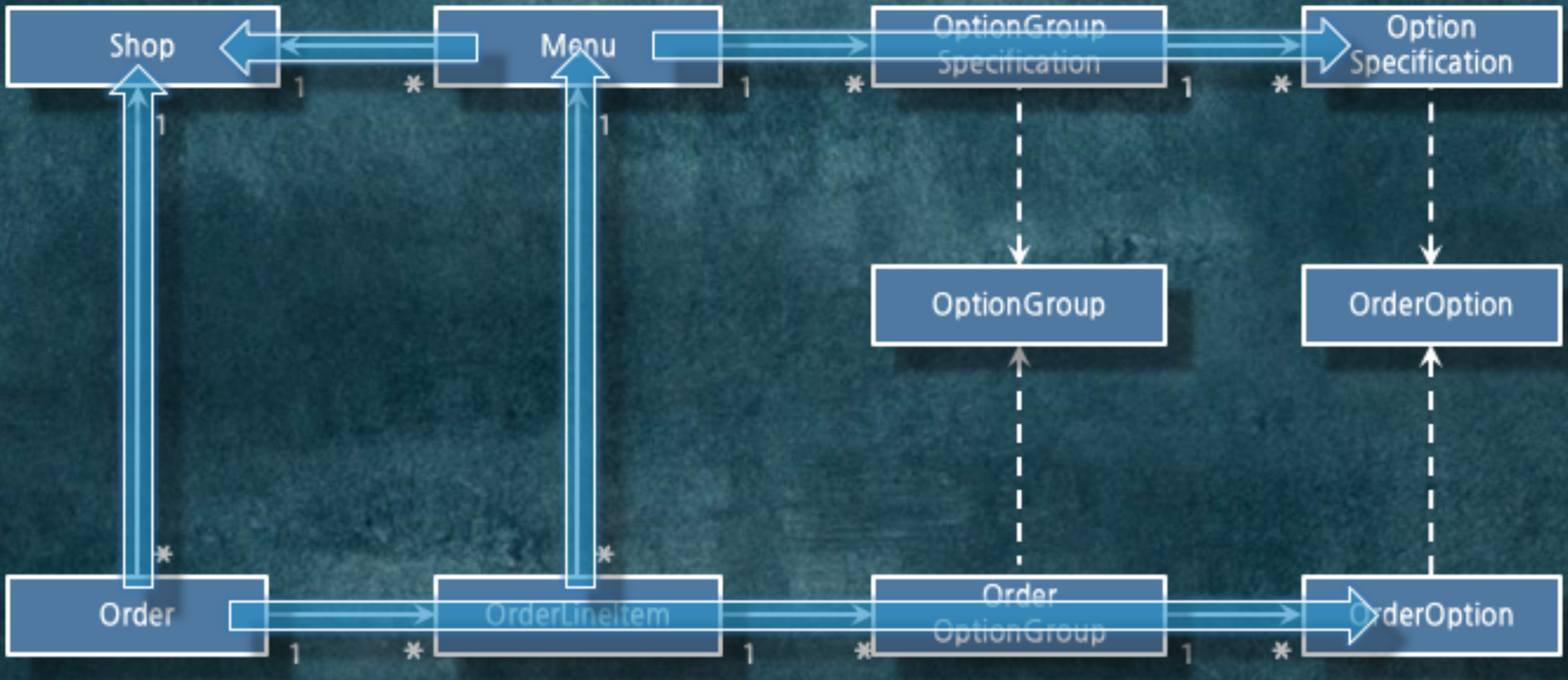
```java
@Entity
@Table(name="ORDERS")
public class Order {
    @ManyToOne
    @JoinColumn(name="SHOP_ID")
    private Shop shop;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name="ORDER_ID")
    private List<OrderLineItem> orderLineItems = new ArrayList<>();
}
```

```java
@Entity
@Table(name = "MENUS")
public class Menu {
    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name="MENU_ID")
    private List<OptionGroupSpecification> optionGroupSpecs = new ArrayList<>();
}
```

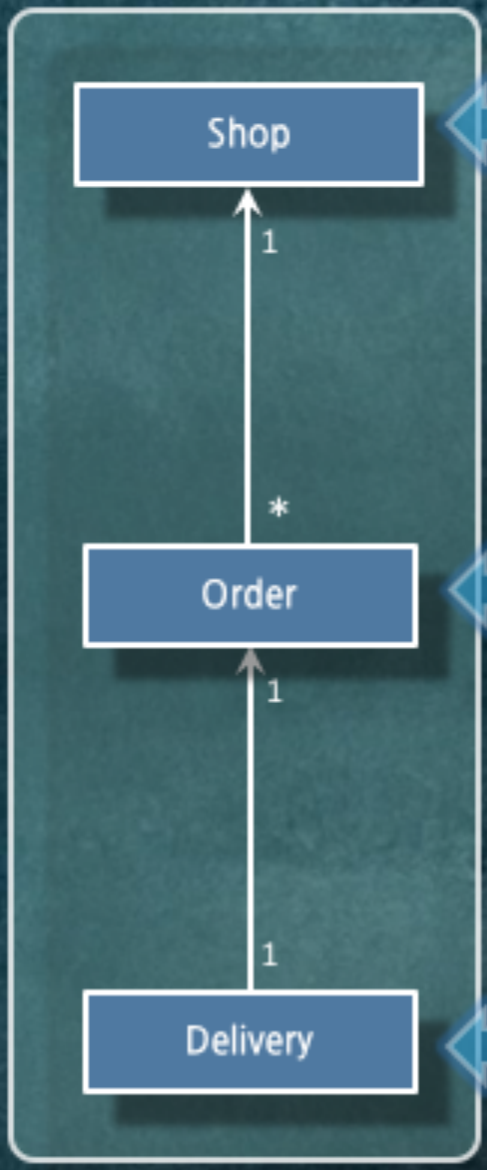
```java
@Entity
@Table(name="OPTION_GROUP_SPECS")
public class OptionGroupSpecification {
    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name="OPTION_GROUP_SPEC_ID")
    private List<OptionSpecification> optionSpecs = new ArrayList<>();
}
```

```java
@Entity
@Table(name="OPTION_SPECS")
public class OptionSpecification {
}
```

어떤 객체라도 접근 가능


Shop
Menu
OptionGroup Specification
Option Specification
OptionGroup
OrderOption
Order
OrderLineItem
Order OptionGroup
OrderOption
1
*

어떤 객체라도 함께 수정 가능



가게 상태 변경
계약 변경
(admin)

주문 상태 변경

배달 상태 변경


객체 참조는 결합도가 가장 높은 의존성

객체 참조 끊기

연관관계와 탐색가능성

Order에서 Shop으로
탐색가능

객체 참조를 통한 탐색(강한 결합도)

Order에서 Shop으로
탐색가능


```java
@Entity
@Table(name="ORDERS")
public class Order {
  @ManyToOne
  @JoinColumn(name="SHOP_ID")
  private Shop shop;
}
```

```java
@Entity
@Table(name="SHOPS")
public class Shop {
  @Id
  @GeneratedValue(strategy =
                  GenerationType.IDENTITY)
  @Column(name="SHOP_ID")
  private Long id;
}
```

Repository를 통한 탐색(약한 결합도)

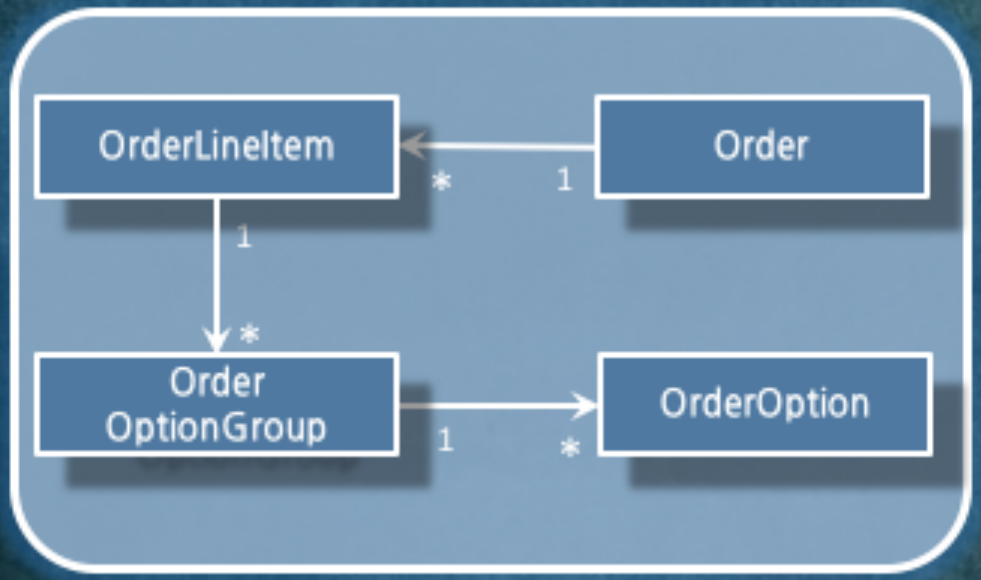
Order에서 Shop으로
탐색가능


```java
@Entity
@Table(name="ORDERS")
public class Order {
  @Column(name="SHOP_ID")
  private Long shopId;
}
```

```java
Shop shop = shopRepository.findById(order.getShopId())
```

```java
@Entity
@Table(name="SHOPS")
public class Shop {
  @Id
  @GeneratedValue(strategy =
              GenerationType.IDENTITY)
  @Column(name="SHOP_ID")
  private Long id;
}
```

어떤 객체들을 묶고 어떤 객체들을 분리할 것인가?



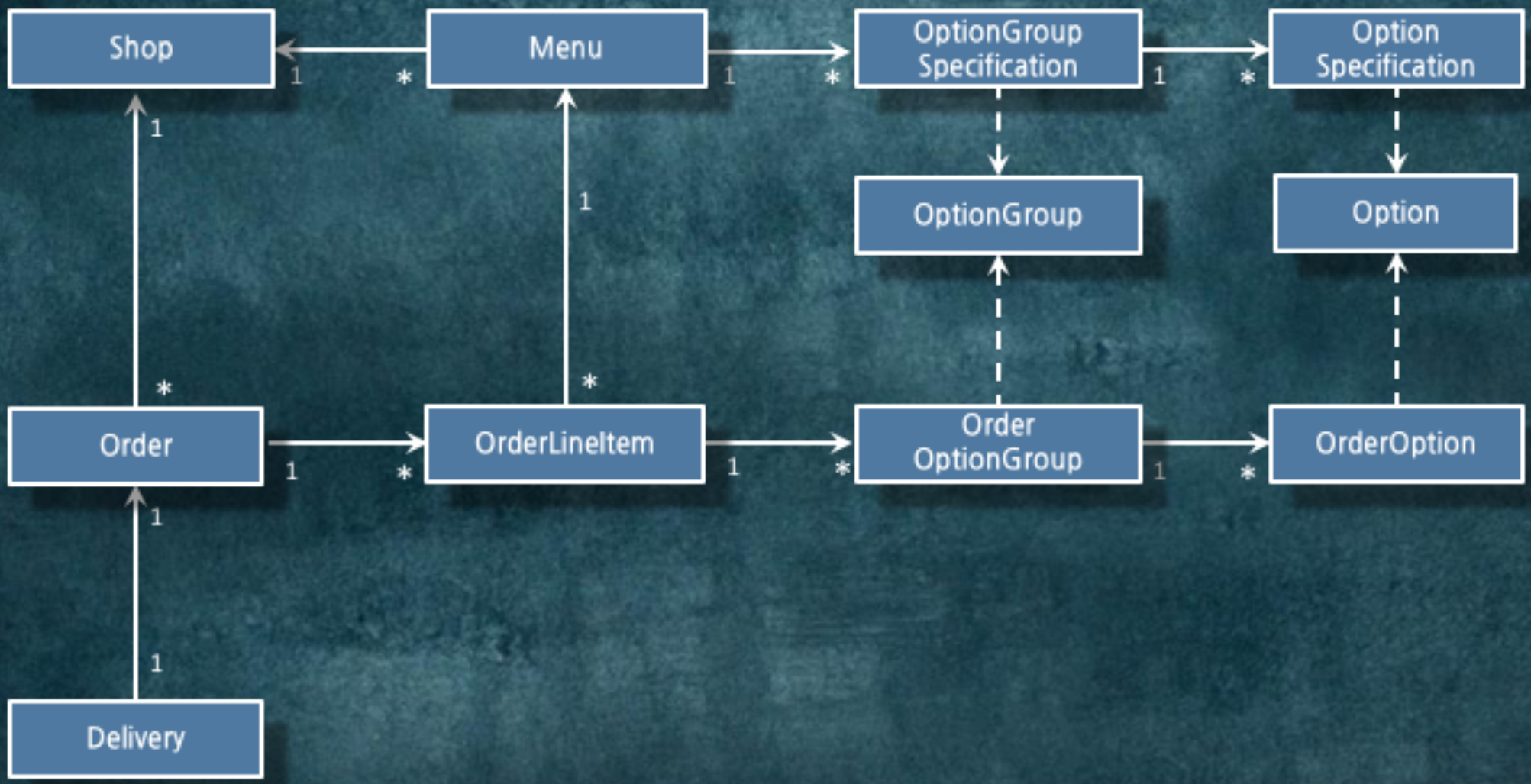
간단한 규칙

함께 생성되고 함께 삭제되는
객체들을 함께 묶어라

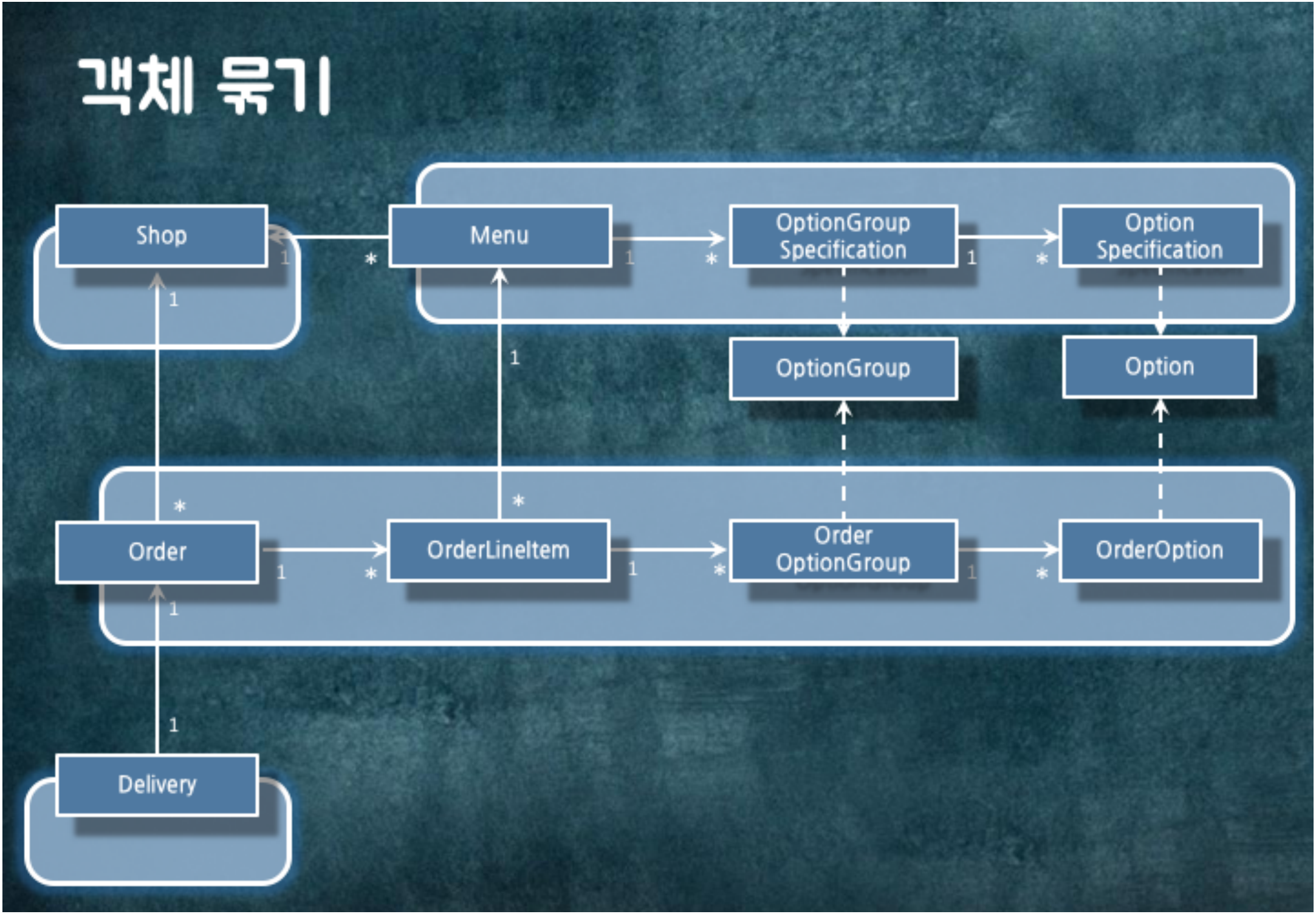
도메인 제약사항을 공유하는
객체들을 함께 묶어라

가능하면 분리하라

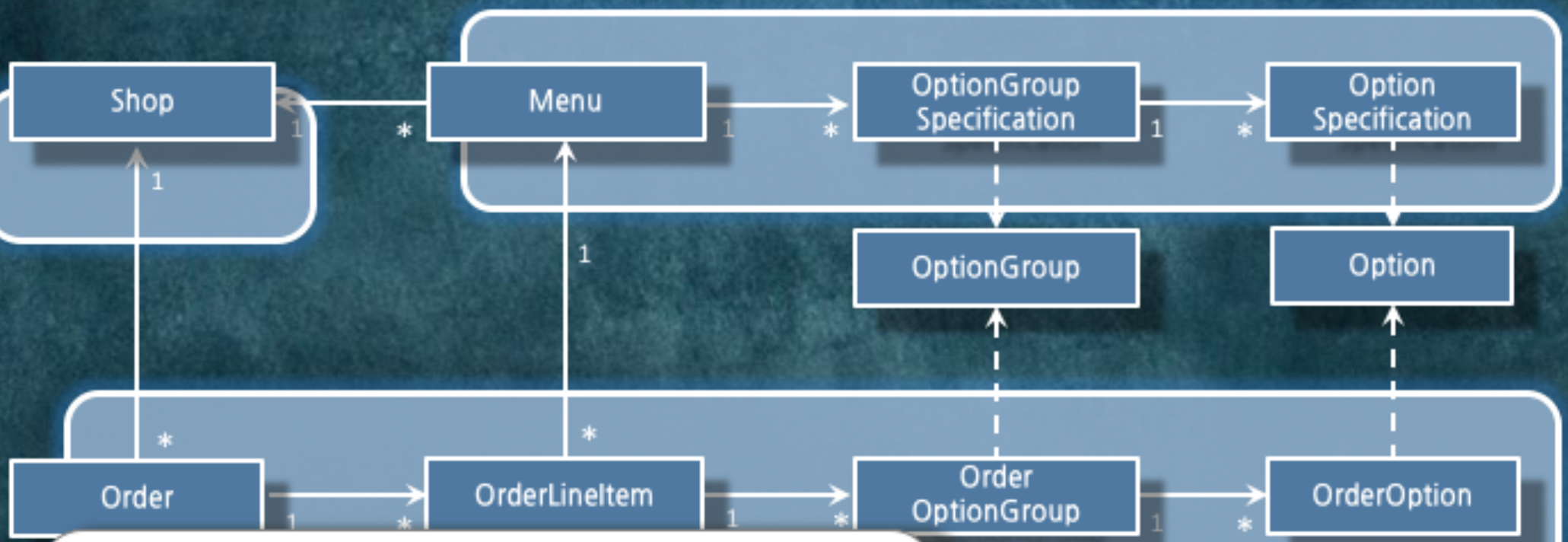
객체 묶기


Shop
Menu
OptionGroup Specification
Option Specification
OptionGroup
Option
Order
OrderLineItem
Order OptionGroup
OrderOption
Delivery
1
*

객체 묶기


Shop
Menu
OptionGroup Specification
Option Specification
OptionGroup
Option
Order
OrderLineItem
Order OptionGroup
OrderOption
Delivery
1
*

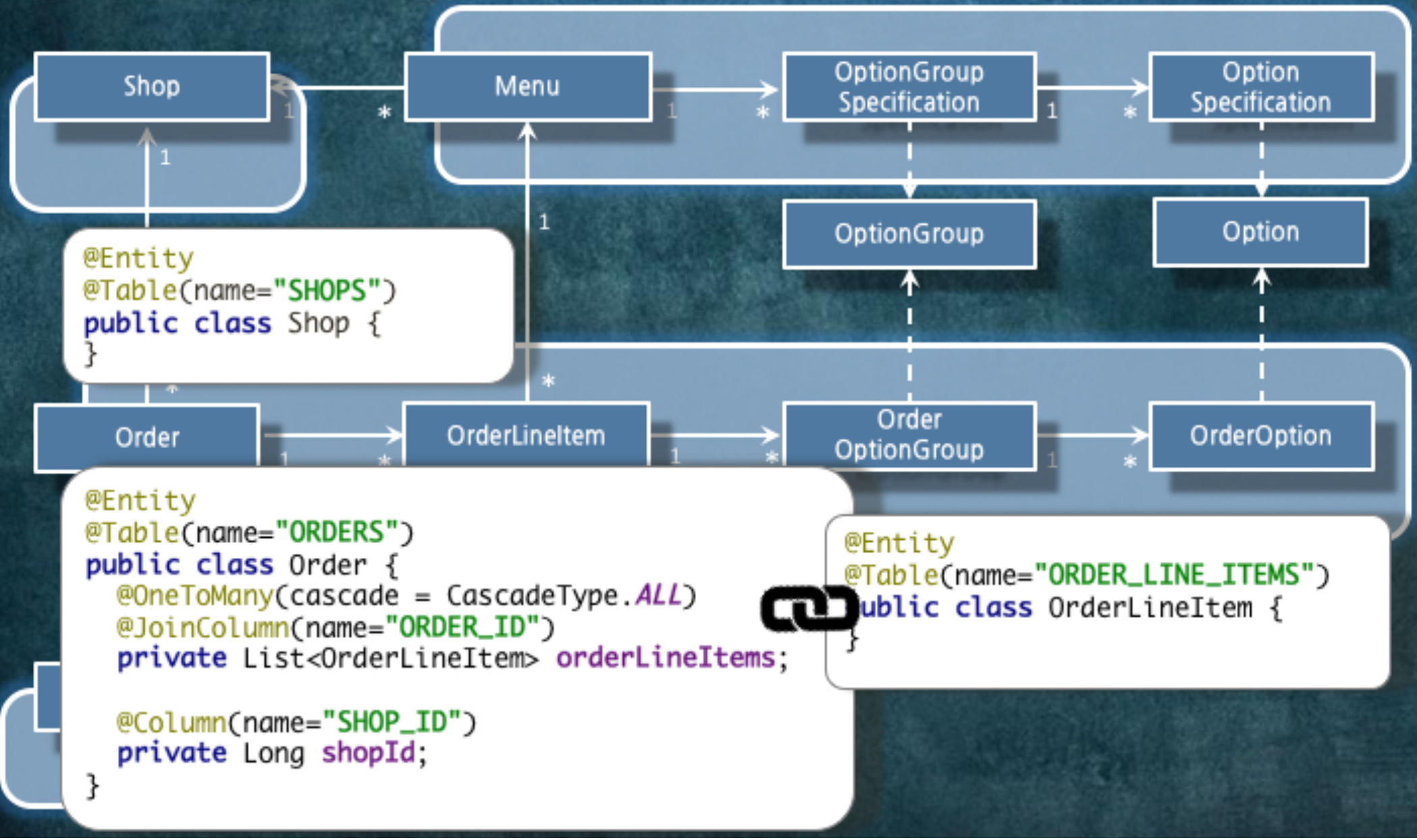
경계 안의 객체는 참조를 이용해 접근




```java
@Entity
@Table(name="ORDERS")
public class Order {
  @OneToMany(cascade = CascadeType.ALL)
  @JoinColumn(name="ORDER_ID")
  private List<OrderLineItem> orderLineItems;
}
```

```java
@Entity
@Table(name="ORDER_LINE_ITEM")
public class OrderLineItem {
}
```

경계 밖의 객체는 ID를 이용해 접근

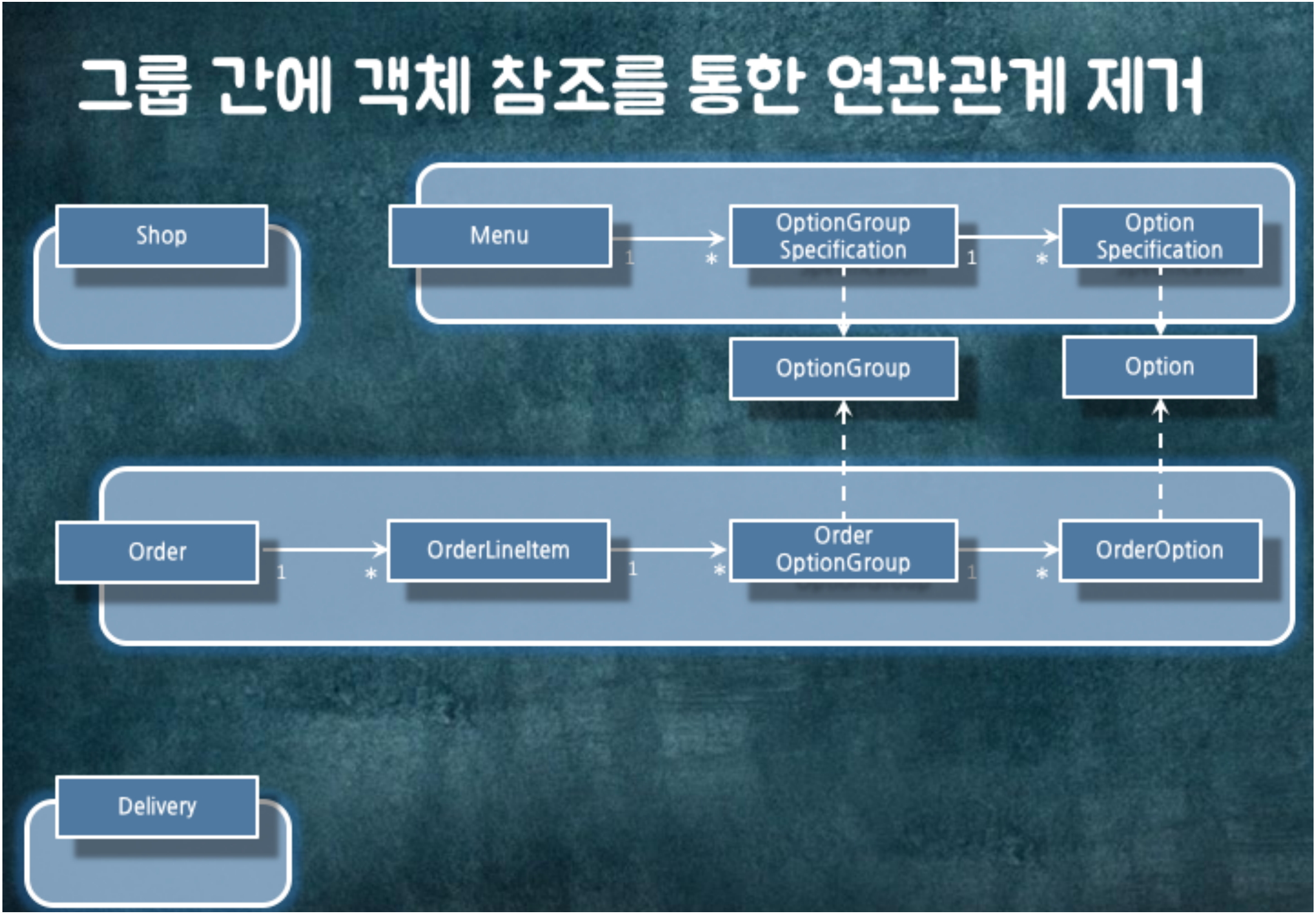



```java
@Entity
@Table(name="SHOPS")
public class Shop {
}
```

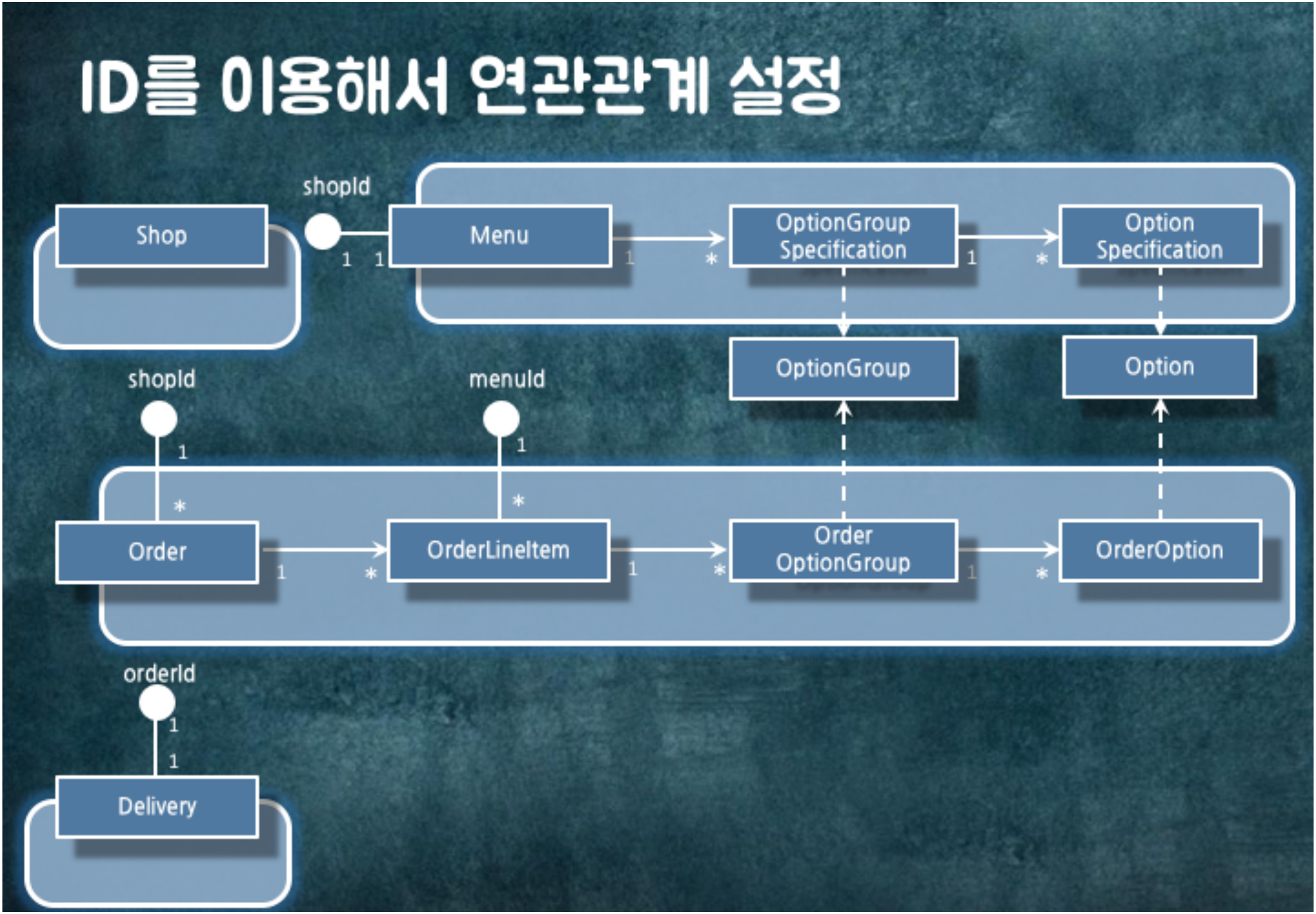
```java
@Entity
@Table(name="ORDERS")
public class Order {
  @OneToMany(cascade = CascadeType.ALL)
  @JoinColumn(name="ORDER_ID")
  private List<OrderLineItem> orderLineItems;

  @Column(name="SHOP_ID")
  private Long shopId;
}
```

```java
@Entity
@Table(name="ORDER_LINE_ITEMS")
public class OrderLineItem {
}
```

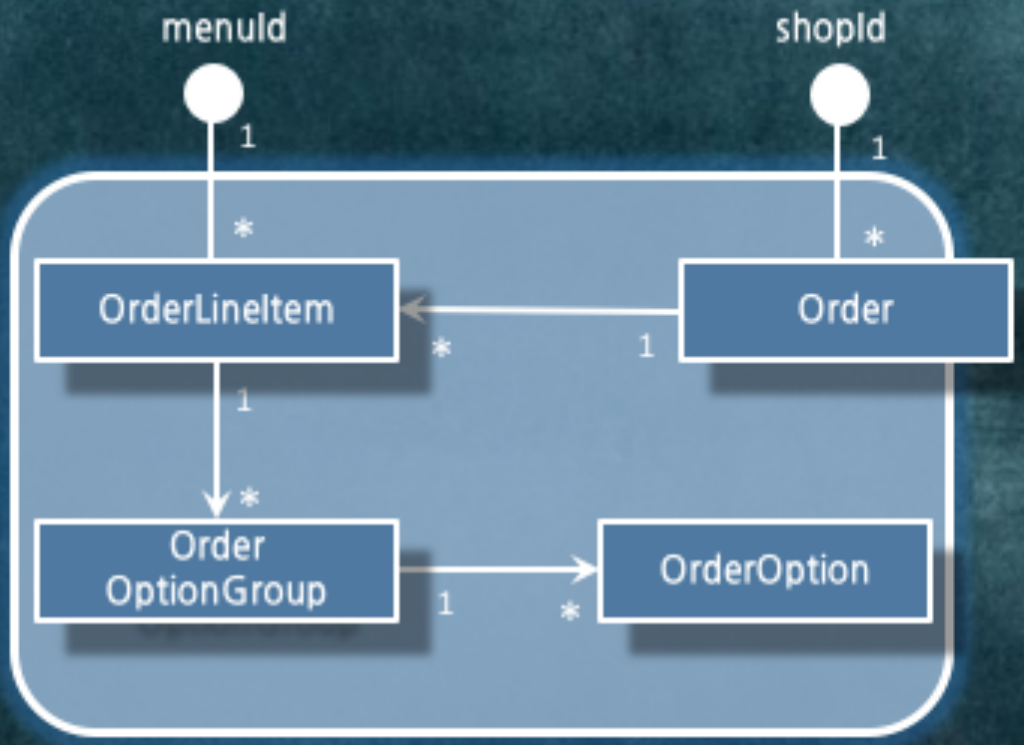
그룹 간에 객체 참조를 통한 연관관계 제거



ID를 이용해서 연관관계 설정

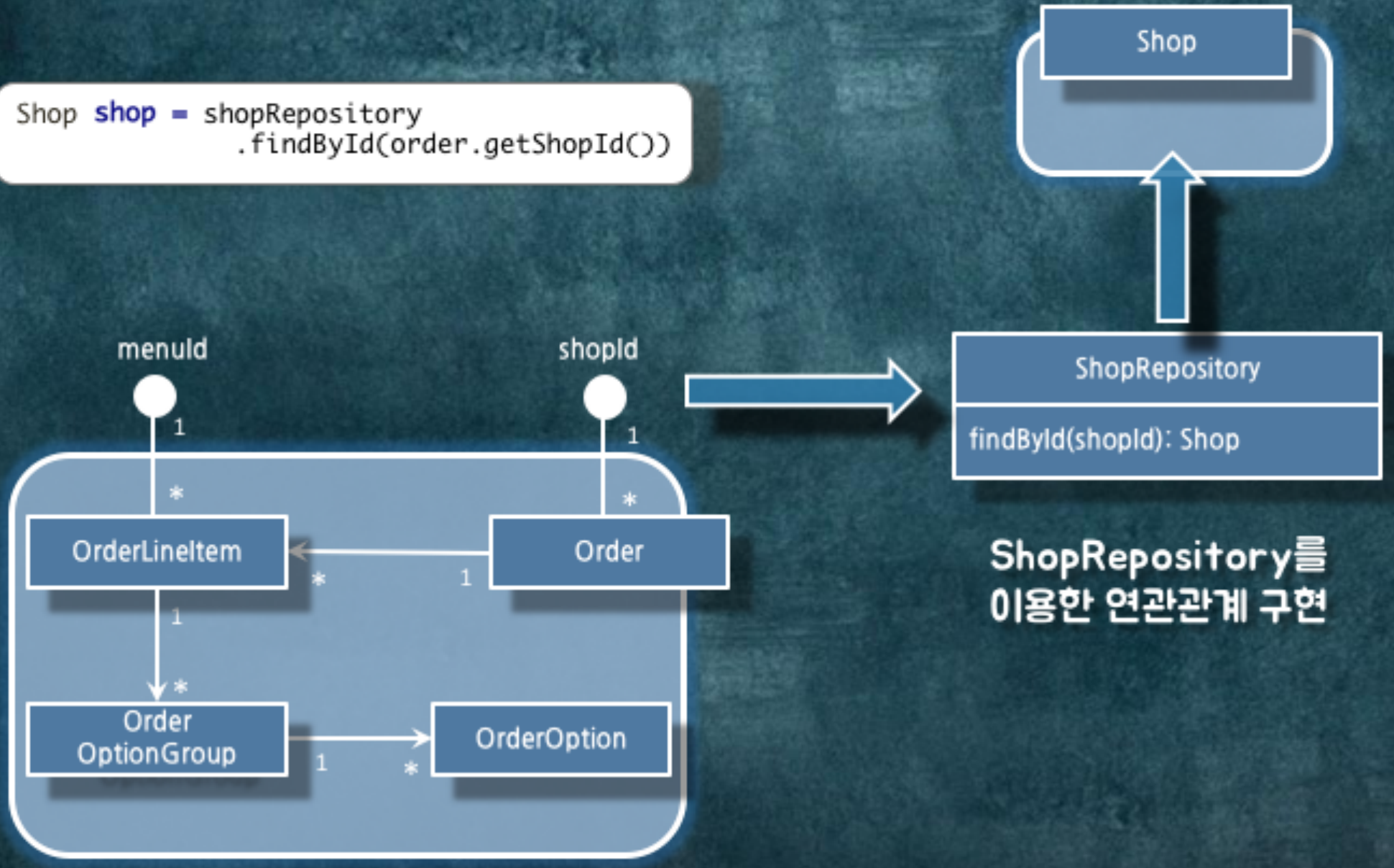


Order의 Shop을 탐색하고 싶다면



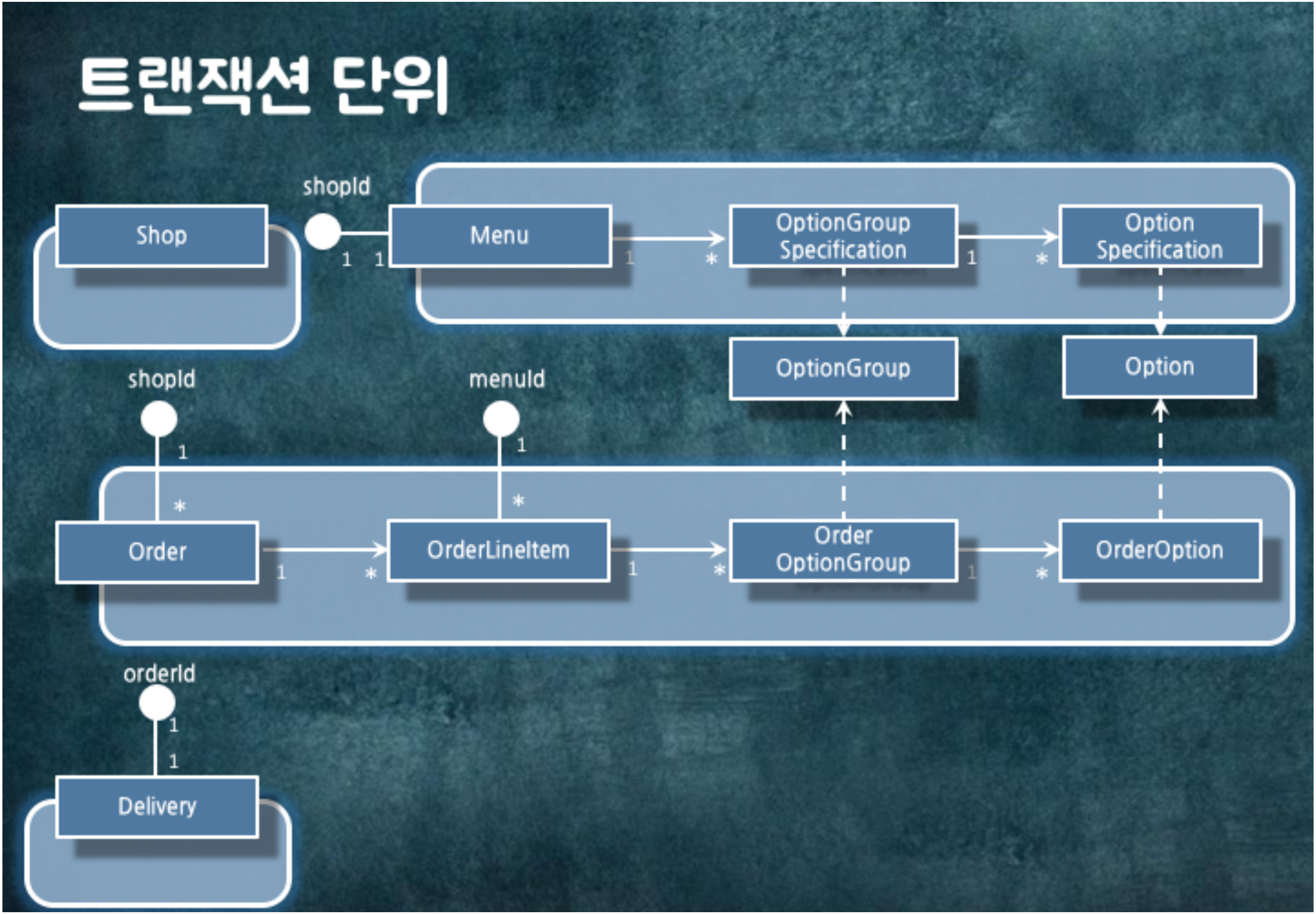
ShopRepository를 통해 탐색

```
Shop shop = shopRepository
              .findById(order.getShopId())
```

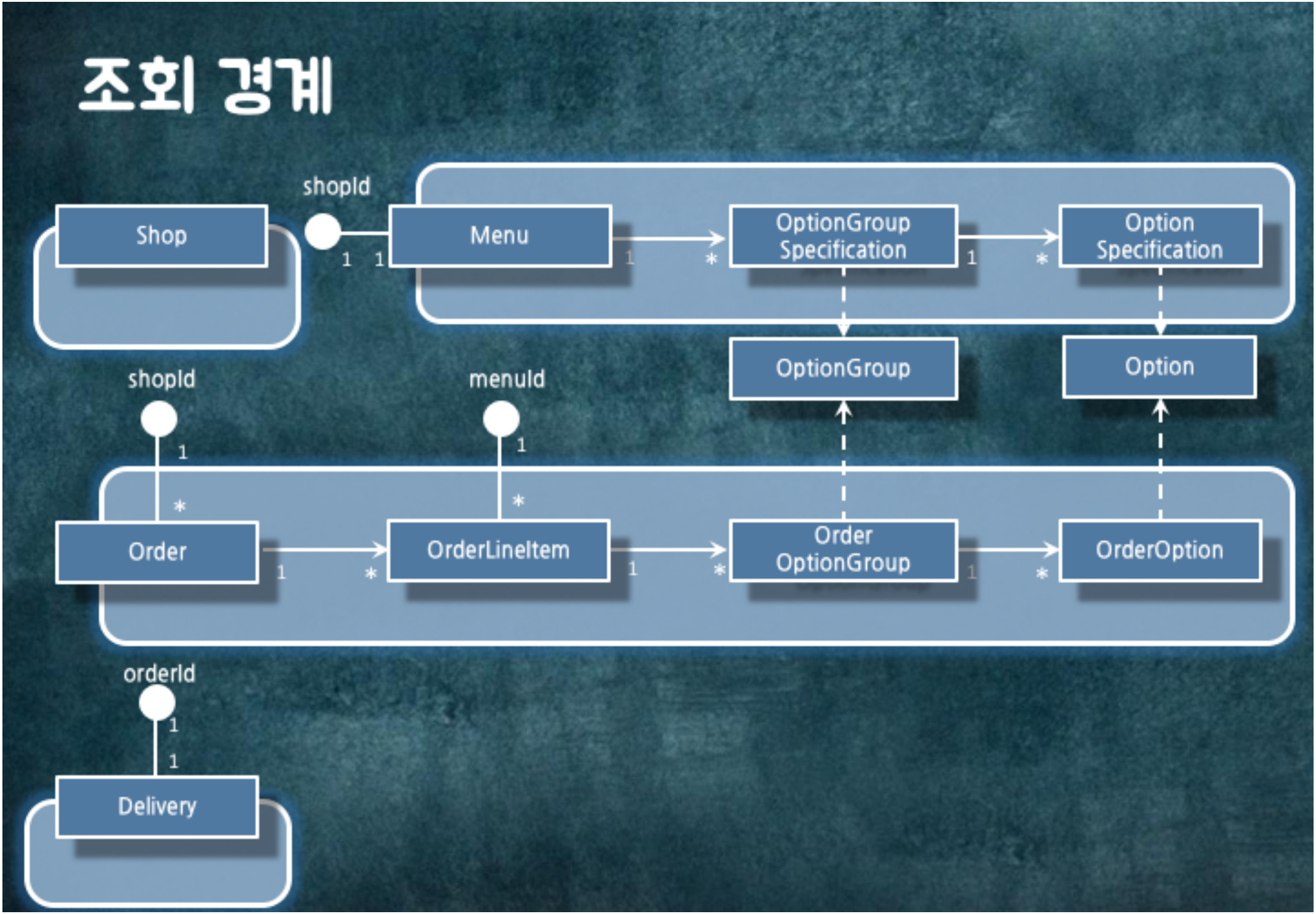


ShopRepository를
이용한 연관관계 구현

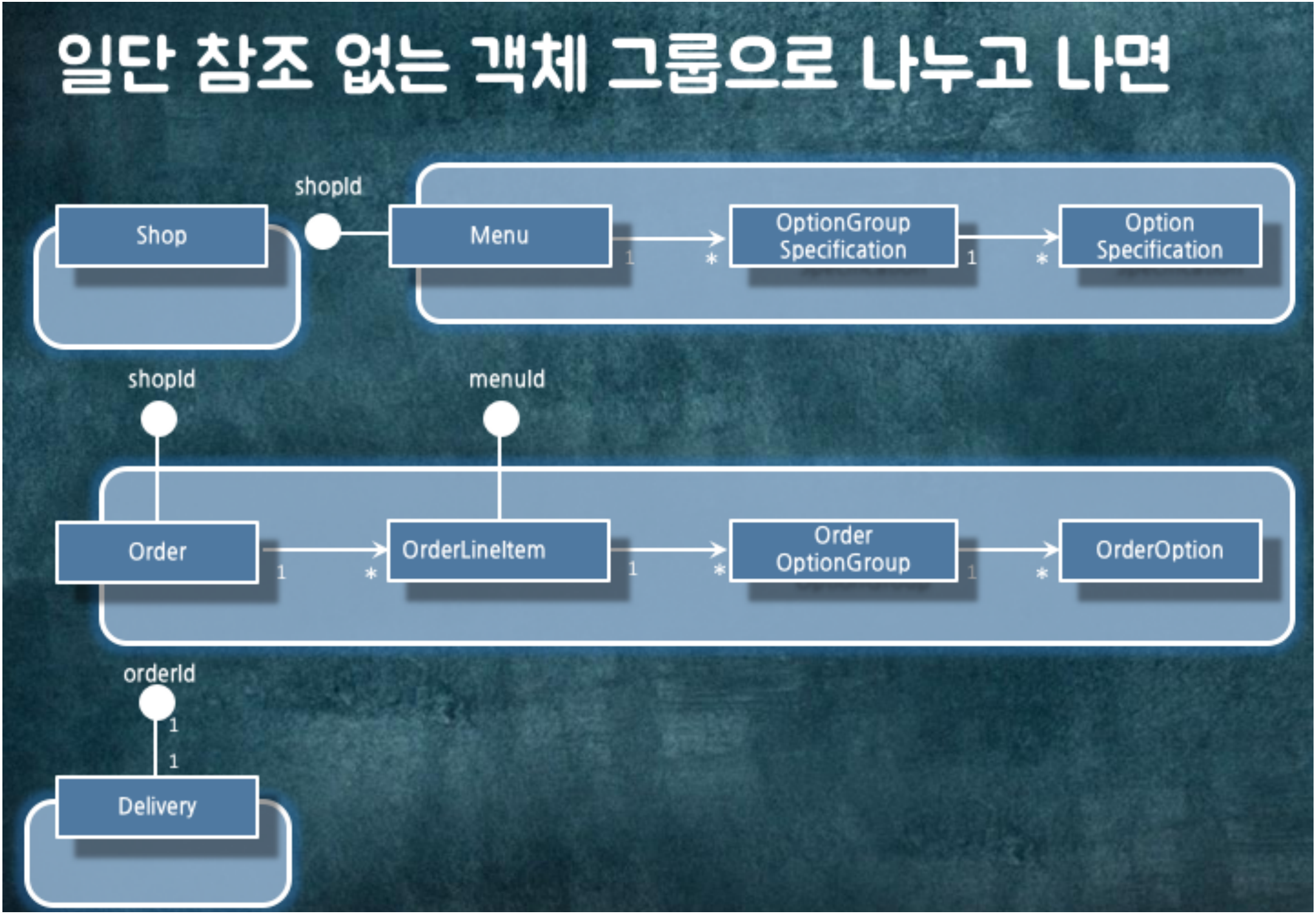
트랜잭션 단위


shopId
Shop
Menu
OptionGroup Specification
Option Specification
OptionGroup
Option
shopId
menuId
Order
OrderLineItem
Order OptionGroup
OrderOption
orderId
Delivery

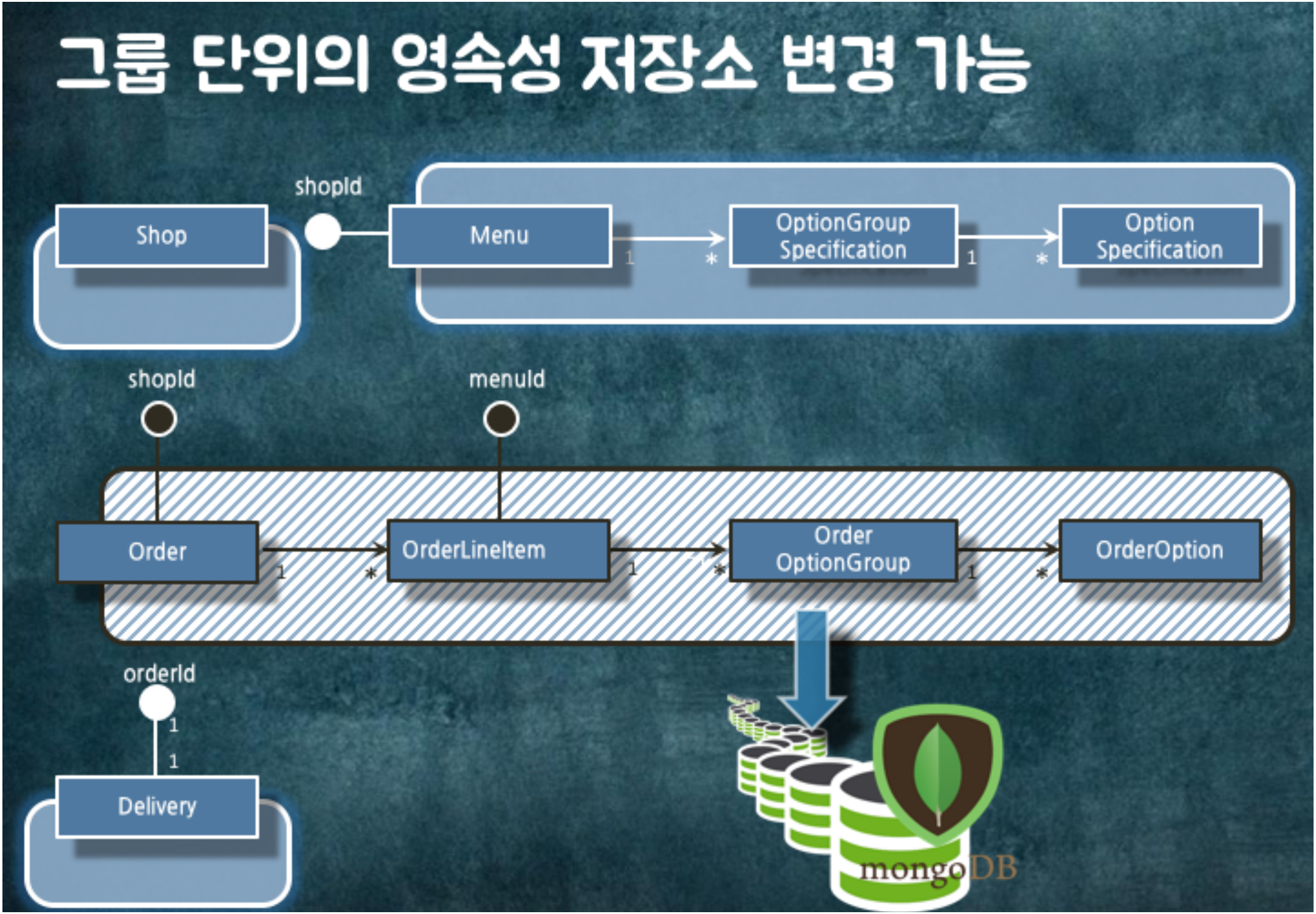
조회 경계


shopId
Shop
1
1
Menu
1
*
OptionGroup Specification
1
*
Option Specification
OptionGroup
Option
shopId
menuId
1
1
*
*
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
orderId
1
1
Delivery

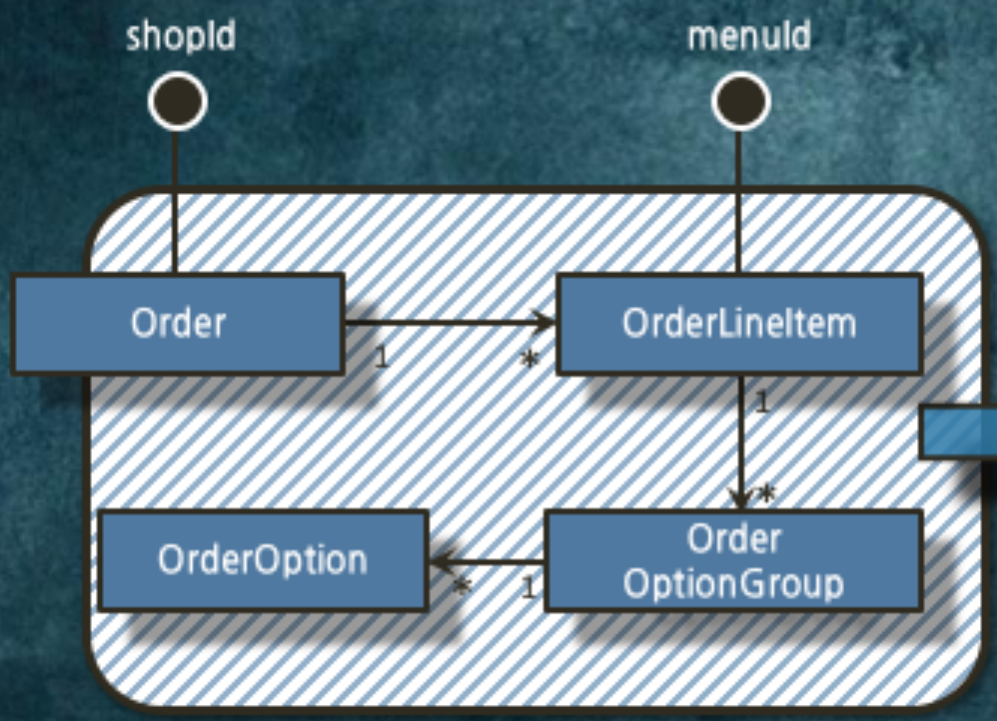
일단 참조 없는 객체 그룹으로 나누고 나면



그룹 단위의 영속성 저장소 변경 가능



그룹은 트랜잭션/조회/비즈니스 제약의 단위



하나의 단위로 저장

db.getCollection('orders').find({})

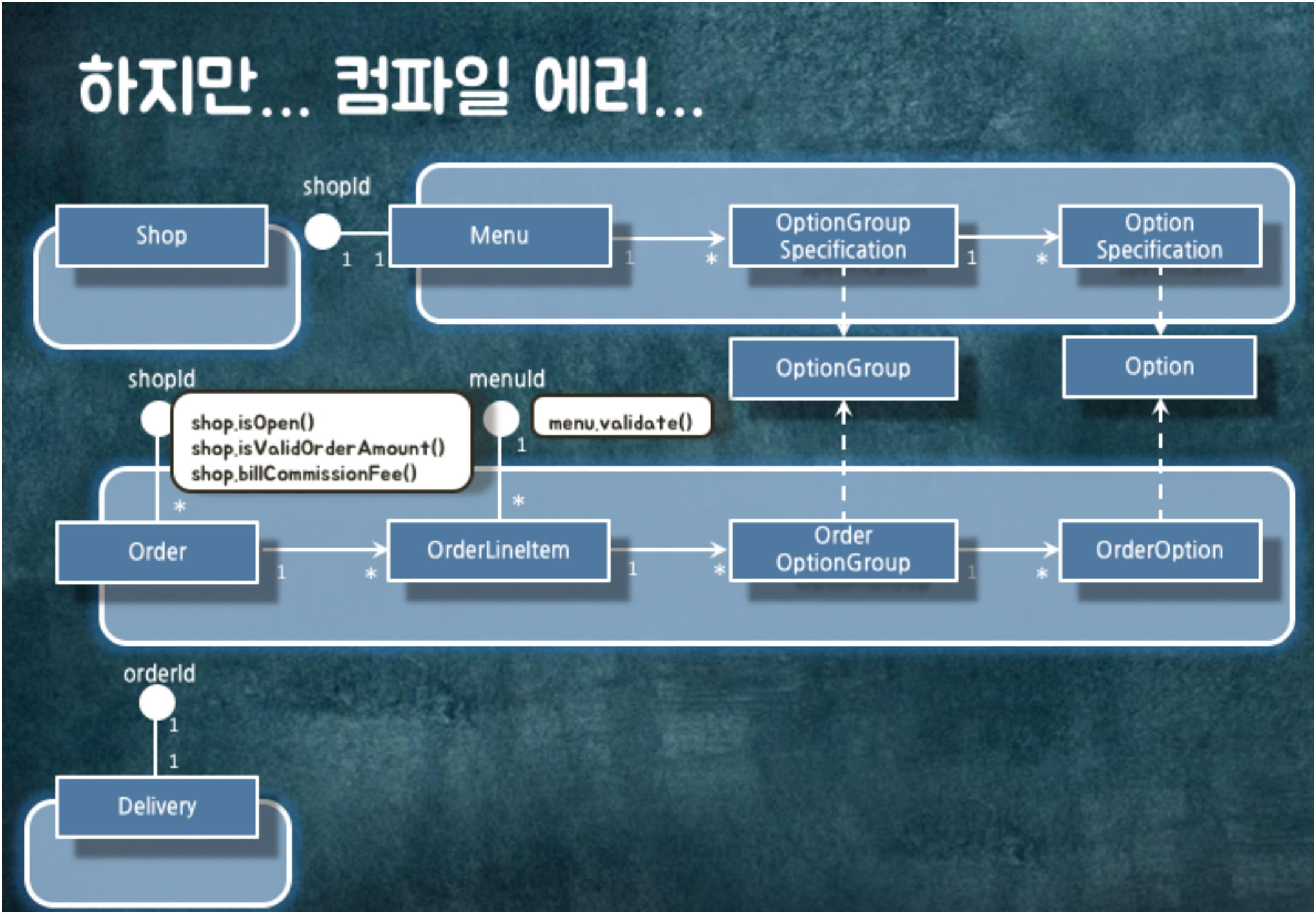
food localhost:27017 food

```
db.getCollection('orders').find({})
```

orders 0.002 sec. 0 50

```
/* 1 */
{
    "_id" : NumberLong(1),
    "userId" : NumberLong(1),
    "shopId" : NumberLong(1),
    "orderLineItems" : [
        {
            "menuId" : NumberLong(1),
            "name" : "삼겹살 1인세트",
            "count" : 2,
            "groups" : [
                {
                    "name" : "기본",
                    "orderOptions" : [
                        {
                            "name" : "소(250g)",
                            "price" : {
                                "amount" : "12000"
                            }
                        }
                    ]
                }
            ]
        }
    ],
    "orderedTime" : ISODate("2019-06-16T14:35:20.161Z"),
    "orderStatus" : "DELIVERED"
}
```

하지만... 컴파일 에러...


shopId
Shop
1
1
Menu
1
*
OptionGroup Specification
1
*
Option Specification
OptionGroup
Option
shopId
shop.isOpen()
shop.isValidOrderAmount()
shop.billCommissionFee()
menuId
menu.validate()
1
*
*
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
orderId
1
1
Delivery

1st 컴파일 에러!!

```java
@Entity
@Table(name="ORDERS")
public class Order {
  private Shop shop;

  @Column(name="SHOP_ID")
  private Long shopId;

  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format(
                "최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }
}
```

1st 컴파일 에러!!

```java
@Entity
@Table(name="ORDERS")
public class Order {
  private Shop shop;

  @Column(name="SHOP_ID")
  private Long shopId;

  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (
      th

    }

    for
      or
    }
  }
}
```

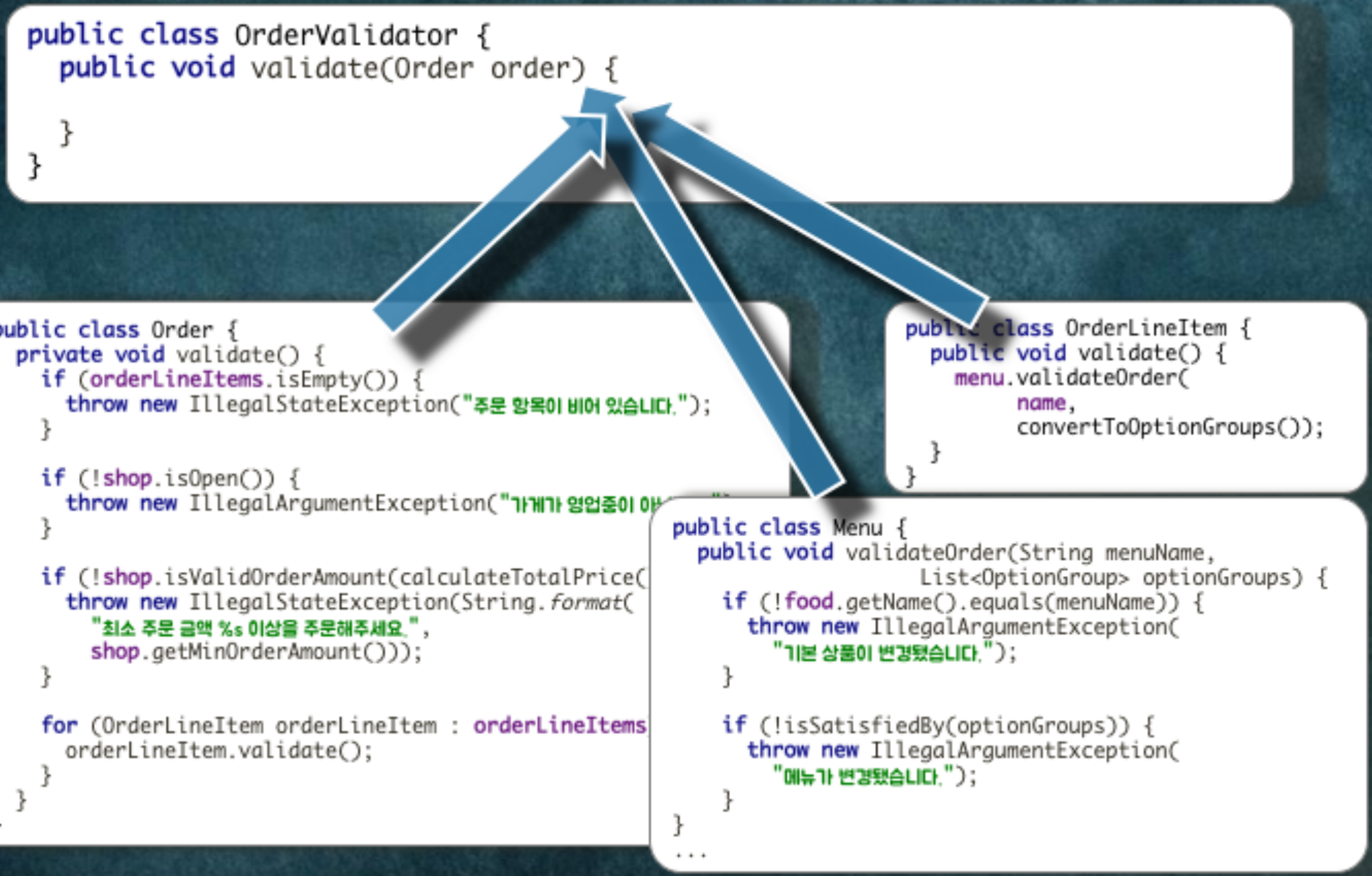
```java
@Entity
@Table(name="ORDER_LINE_ITEMS")
public class OrderLineItem {
  private Menu menu;

  @Column(name="MENU_ID")
  private Long menuId;

  public void validate() {
    menu.validateOrder(name, convertToOptionGroups());
  }
}
```


객체를 직접 참조하는 로직을
다른 객체로 옮기자!

새 객체 OrderValidator를 준비하고

```
public class OrderValidator {
  public void validate(Order order) {

  }
}
```

Validation Logic을 이동



```
public class OrderValidator {
  public void validate(Order order) {

  }
}
```

```
public class Order {
  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice(
      throw new IllegalStateException(String.format(
        "최소 주문 금액 %s 이상을 주문해주세요.",
        shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems
      orderLineItem.validate();
    }
  }
}
```

```
public class OrderLineItem {
  public void validate() {
    menu.validateOrder(
        name,
        convertToOptionGroups());
  }
}
```

```
public class Menu {
  public void validateOrder(String menuName,
                            List<OptionGroup> optionGroups) {
    if (!food.getName().equals(menuName)) {
      throw new IllegalArgumentException(
        "기본 상품이 변경됐습니다.");
    }

    if (!isSatisfiedBy(optionGroups)) {
      throw new IllegalArgumentException(
        "메뉴가 변경됐습니다.");
    }
}
...
```

Validation Logic 모으기

```
@Component
public class OrderValidator {
  public void validate(Order order) { validate(order,getShop(order),getMenus(order)); }

  private void validate(Order order, Shop shop, Map<Long, Menu> menus) {
    if (!shop.isOpen()) throw new IllegalArgumentException("가게가 영업중이 아닙니다.");

    if (!shop.isValidOrderAmount(order.calculateTotalPrice()))
        throw new IllegalStateException(String.format("최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));

    if (order.getOrderLineItems().isEmpty()) throw new IllegalStateException("주문 항목이 비어 있습니다.");

    for (OrderLineItem item : order.getOrderLineItems()) {
      validateOrderLineItem(item, menus.get(item.getMenuId()));
    }
  }

  private void validateOrderLineItem(OrderLineItem item, Menu menu) {
    if (!menu.getFood().getName().equals(item.getName()))
      throw new IllegalArgumentException("기본 상품이 변경됐습니다.");

    for(OrderOptionGroup group : item.getGroups()) {
      validateOrderOptionGroup(group, menu);
    }
  }

  private void validateOrderOptionGroup(OrderOptionGroup group, Menu menu) {
    for(OptionGroupSpecification spec : menu.getOptionGroupSpecs()) {
      if (spec.isSatisfiedBy(group.convertToOptionGroup())) {
        return;
      }
    }

    throw new IllegalArgumentException("메뉴가 변경됐습니다.");
  }
  ...
}
```

OrderValidator를 이용한 구현

```java
@Service
public class OrderService {
  private OrderMapper orderMapper;
  private OrderValidator orderValidator;
  private OrderRepository orderRepository;

  @Transactional
  public void placeOrder(Cart cart) {
    Order order = orderMapper.mapFrom(cart);

    order.place(orderValidator);

    orderRepository.save(order);
  }
}
```

```java
public class Order {
  public void place(OrderValidator orderValidator) {
    orderValidator.validate(this);
    ordered();
  }
}
```

객체지향은 여러 객체를 오가며 로직 파악

```
public class Order {
  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format(
        "최소 주문 금액 %s 이상을 주문해주세요.",
        shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }
}
```

```
public class OrderLineItem {
  public void validate() {
    menu.validateOrder(name, convertToOptionGroups());
  }
}
```

```
public class Menu {
  public void validateOrder(String menuName,
                            List<OptionGroup> optionGroups) {
    if (!food.getName().equals(menuName)) {
      throw new IllegalArgumentException(
        "기본 상품이 변경됐습니다.");
    }

    if (!isSatisfiedBy(optionGroups)) {
      throw new IllegalArgumentException(
        "메뉴가 변경됐습니다.");
    }
}
...
```

전체 Validation Logic을 한 눈에

```
@Component
public class OrderValidator {
  public void validate(Order order) { validate(order,getShop(order),getMenus(order)); }

  private void validate(Order order, Shop shop, Map<Long, Menu> menus) {
    if (!shop.isOpen()) throw new IllegalArgumentException("가게가 영업중이 아닙니다.");

    if (!shop.isValidOrderAmount(order.calculateTotalPrice()))
        throw new IllegalStateException(String.format("최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));

    if (order.getOrderLineItems().isEmpty()) throw new IllegalStateException("주문 항목이 비어 있습니다.");

    for (OrderLineItem item : order.getOrderLineItems()) {
      validateOrderLineItem(item, menus.get(item.getMenuId()));
    }
  }

  private void validateOrderLineItem(OrderLineItem item, Menu menu) {
    if (!menu.getFood().getName().equals(item.getName()))
      throw new IllegalArgumentException("기본 상품이 변경됐습니다.");

    for(OrderOptionGroup group : item.getGroups()) {
      validateOrderOptionGroup(group, menu);
    }
  }

  private void validateOrderOptionGroup(OrderOptionGroup group, Menu menu) {
    for(OptionGroupSpecification spec : menu.getOptionGroupSpecs()) {
      if (spec.isSatisfiedBy(group.convertToOptionGroup())) {
        return;
      }
    }

    throw new IllegalArgumentException("메뉴가 변경됐습니다.");
  }
  ...
```

낮은 응집도의 객체가

```
public class Order {
  public void place() {
    validate();
    ordered();
  }

  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format("최소 주문 금액 %s 이상을 주문해주세요.",
                                                    shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }

  private void ordered() {
    this.orderStatus = OrderStatus.ORDERED;
  }

  public void payed() {
    this.orderStatus = OrderStatus.PAYED;
  }

  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }

  public Money calculateTotalPrice() {
    return Money.sum(orderLineItems, OrderLineItem::calculatePrice);
  }
```

validation

주문 처리

높은 응집도의 객체로 단일-책임 원칙

```java
public class Order {
  public void place(OrderValidator orderValidator) {
    orderValidator.validate(this);
    ordered();
  }

  private void ordered() {
    this.orderStatus = OrderStatus.ORDERED;
  }

  public void payed() {
    this.orderStatus = OrderStatus.PAYED;
    registerEvent(new OrderPayedEvent(this));
  }

  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    registerEvent(new OrderDeliveredEvent(this));
  }

  public Money calculateTotalPrice() {
    return Money.sum(orderLineItems, OrderLineItem::calculatePrice);
  }
}
```

주문 처리

때로는 절차지향이 객체지향보다 좋다

```
@Component
public class OrderValidator {
  public void validate(Order order) {
    validate(order,getShop(order),getMenus(order));
  }

  private void validate(Order order, Shop shop, Map<Long, Menu> menus) {
    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(order.calculateTotalPrice())) {
        throw new IllegalStateException(String.format("최소 주문 금액 %s 이상을 주문해주세요.", shop.getMinOrderAmount()));
    }

    if (order.getOrderLineItems().isEmpty()) {
      throw new IllegalStateException("주문 항목이 없어 있습니다.");
    }

    for (OrderLineItem item : order.getOrderLineItems()) {
      validateOrderLineItem(item, menus.get(item.getMenuId()));
    }
  }

  private void validateOrderLineItem(OrderLineItem item, Menu menu) {
    if (!menu.getFood().getName().equals(item.getName())) {
      throw new IllegalArgumentException("다른 상품이 반영됐습니다.");
    }

    for(OrderOptionGroup group : item.getGroups()) {
      validateOrderOptionGroup(group, menu);
    }
  }

  private void validateOrderOptionGroup(OrderOptionGroup group, Menu menu) {
    for(OptionGroupSpecification spec : menu.getOptionGroupSpecs()) {
      if (spec.isSatisfiedBy(group.convertToOptionGroup())) {
        return;
      }
    }

    throw new IllegalArgumentException("메뉴가 변경됐습니다.");
  }

  private Shop getShop(Order order) {
    return shopRepository.findById(order.getShopId()).orElseThrow(IllegalArgumentException::new);
  }

  private Map<Long, Menu> getMenus(Order order) {
    return menuRepository.findAllById(order.getMenuIds()).stream().collect(toMap(Menu::getId, identity()));
  }
}
```

```
public class Order {
  private void validate() {
    if (orderLineItems.isEmpty()) {
      throw new IllegalStateException("주문 항목이 비어 있습니다.");
    }

    if (!shop.isOpen()) {
      throw new IllegalArgumentException("가게가 영업중이 아닙니다.");
    }

    if (!shop.isValidOrderAmount(calculateTotalPrice())) {
      throw new IllegalStateException(String.format(
        "최소 주문 금액 %s 이상을 주문해주세요.",
        shop.getMinOrderAmount()));
    }

    for (OrderLineItem orderLineItem : orderLineItems) {
      orderLineItem.validate();
    }
  }
}
```

```
public class OrderLineItem {
  public void validate() {
    menu.validateOrder(name, convertToOptionGroups());
  }
}
```

```
public class Menu {
  public void validateOrder(String menuName, List<OptionGroup> optionGroups) {
    if (!food.getName().equals(menuName)) {
      throw new IllegalArgumentException("다른 상품이 반영됐습니다.");
    }

    if (!isSatisfiedBy(optionGroups)) {
      throw new IllegalArgumentException("메뉴가 변경됐습니다.");
    }
  }
  ...
```

2nd 컴파일 에러 - 배달 완료

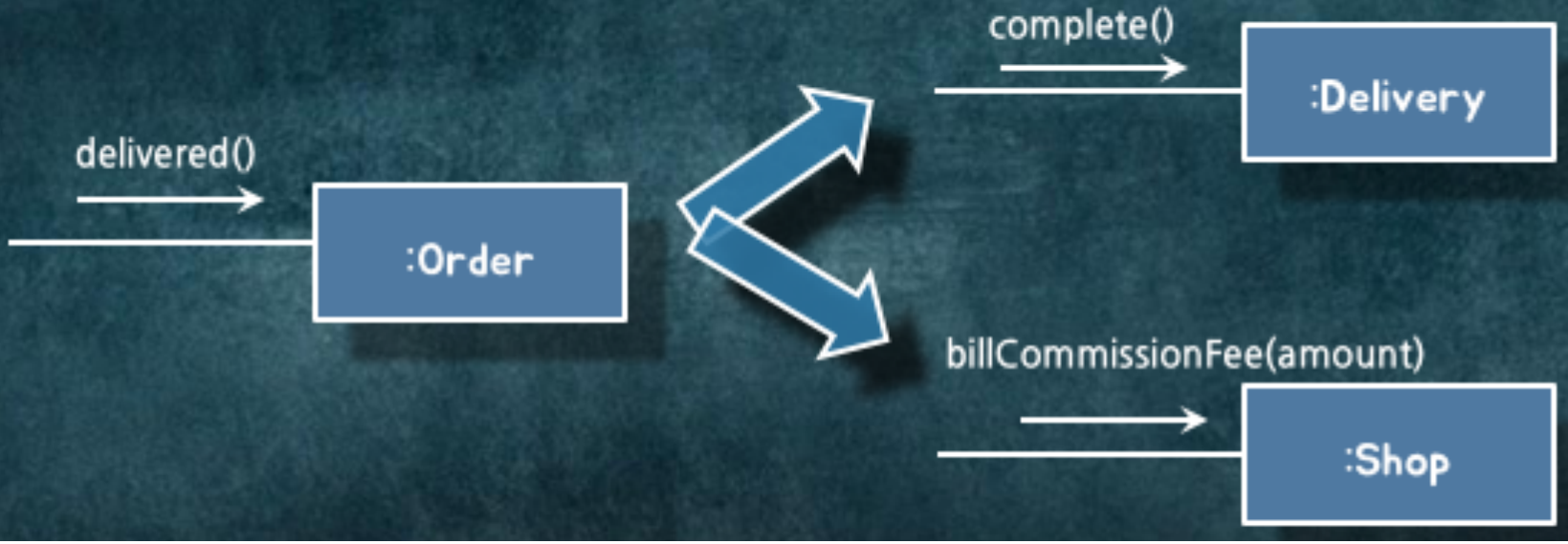
```java
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    order.delivered();

    Delivery delivery = deliveryRepository.findById(orderId) ...
    delivery.complete();
  }
}
```

```java
@Entity
@Table(name="ORDERS")
public class Order {
  private Shop shop;

  @Column(name="SHOP_ID")
  private Long shopId;

  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

본질: 도메인 로직의 순차적 실행

```java
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    order.delivered();
    delivery.complete();
  }
}
```

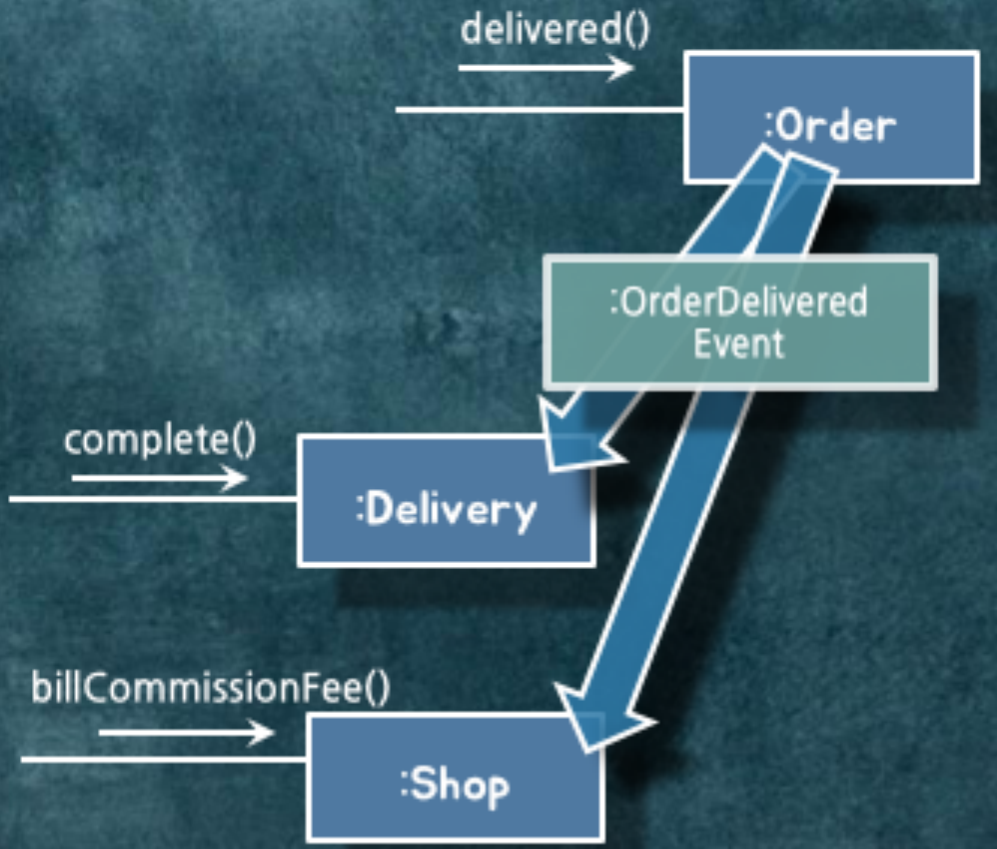
```java
public class Order {
  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```



두 가지 해결방법

절차지향 로직OrderValidator와 동일

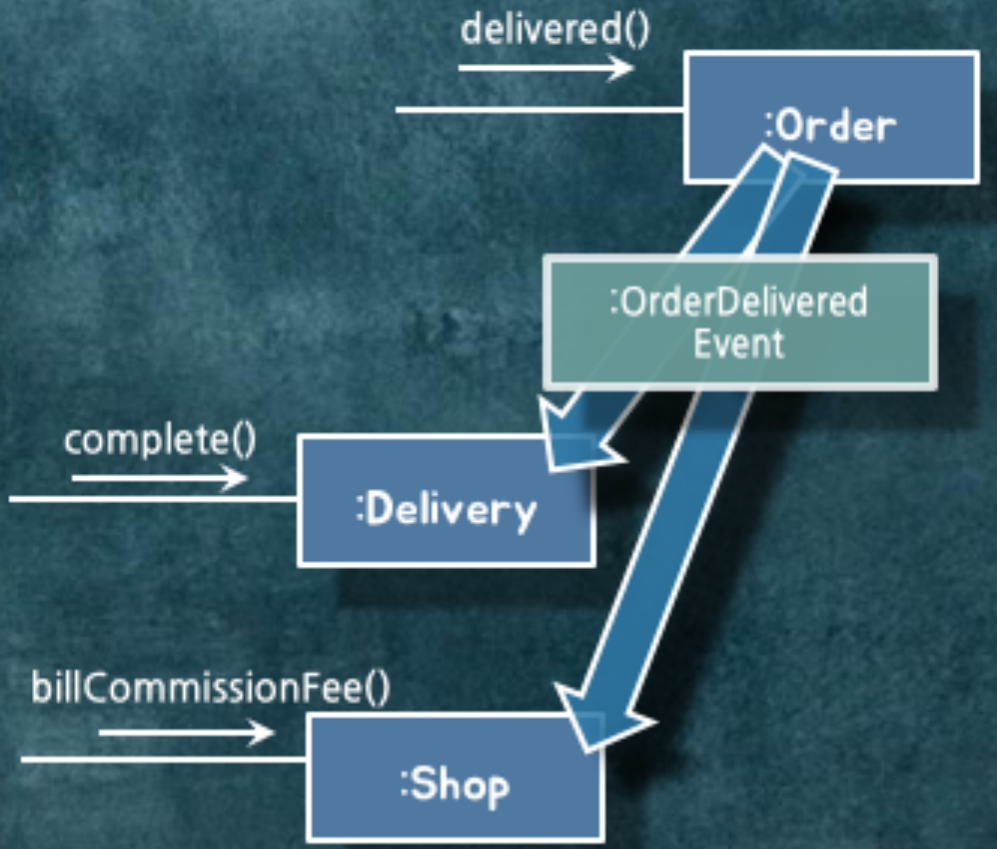
도메인 이벤트Domain Event 퍼블리싱



첫번째 방법 - 절차지향

절차지향 로직(OrderValidator 방식)

도메인 이벤트Domain Event 퍼블리싱

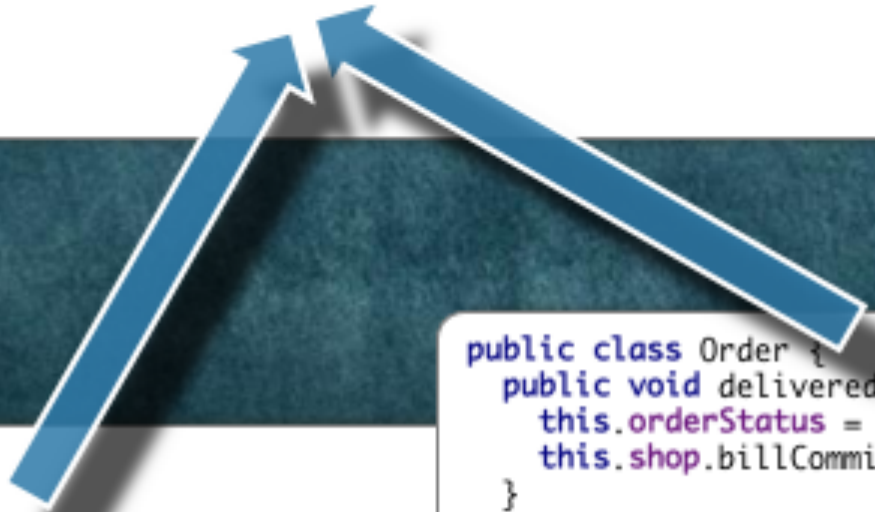


OrderDeliveredService를 추가하고

```
public class OrderDeliveredService {
  @Transactional
  public void deliverOrder(Long orderId) {

  }
}
```

배달 완료 로직 이동

```java
public class OrderDeliveredService {
  @Transactional
  public void deliverOrder(Long orderId) {

  }
}
```



```java
public class Order {
  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

```java
public class OrderService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId)
                    .orElseThrow(IllegalArgumentException::new);
    order.delivered();

    Delivery delivery = deliveryRepository.findById(orderId)
                    .orElseThrow(IllegalArgumentException::new);
    delivery.complete();
  }
}
```

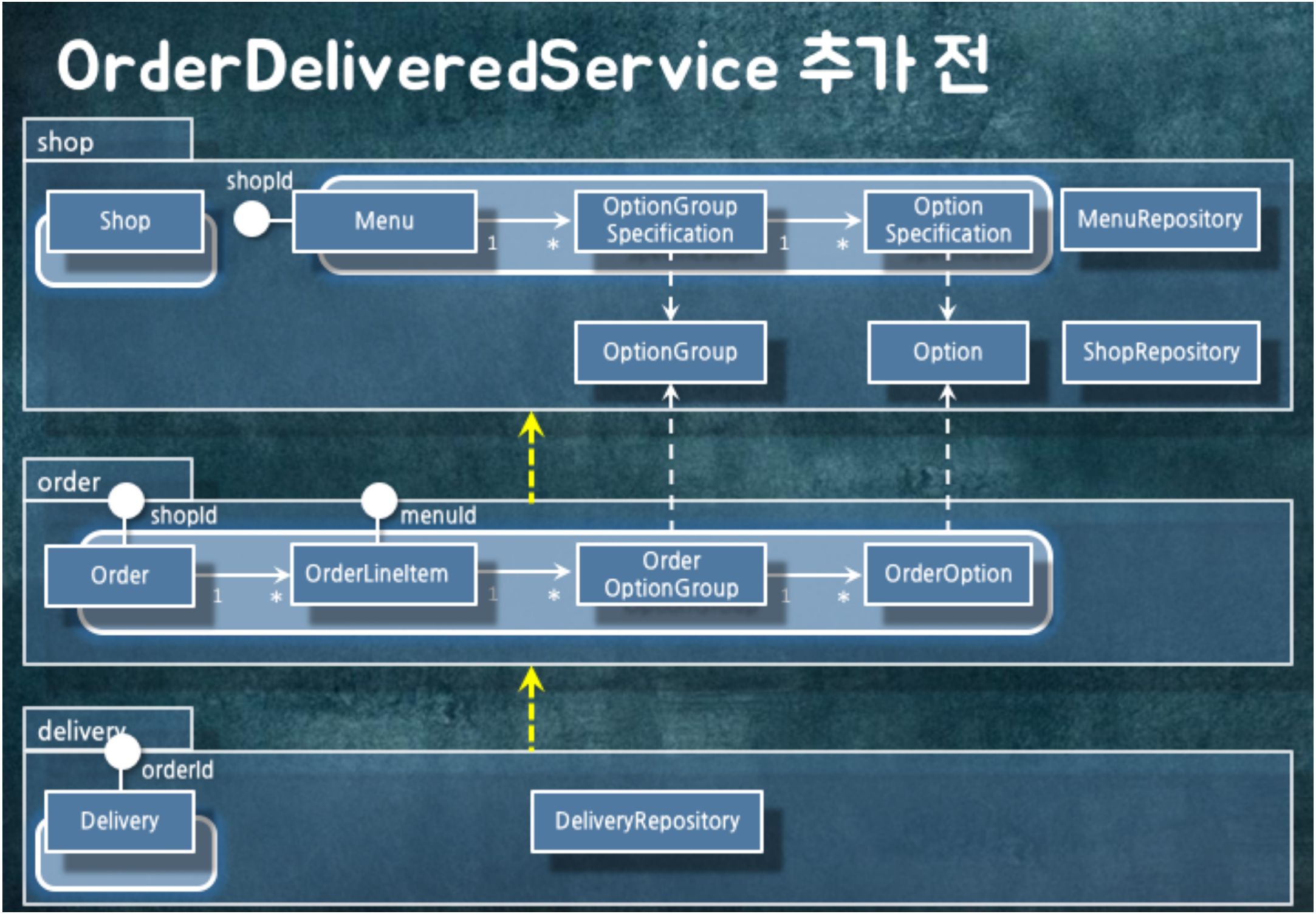
절차지향적인 OrderDeliveredService

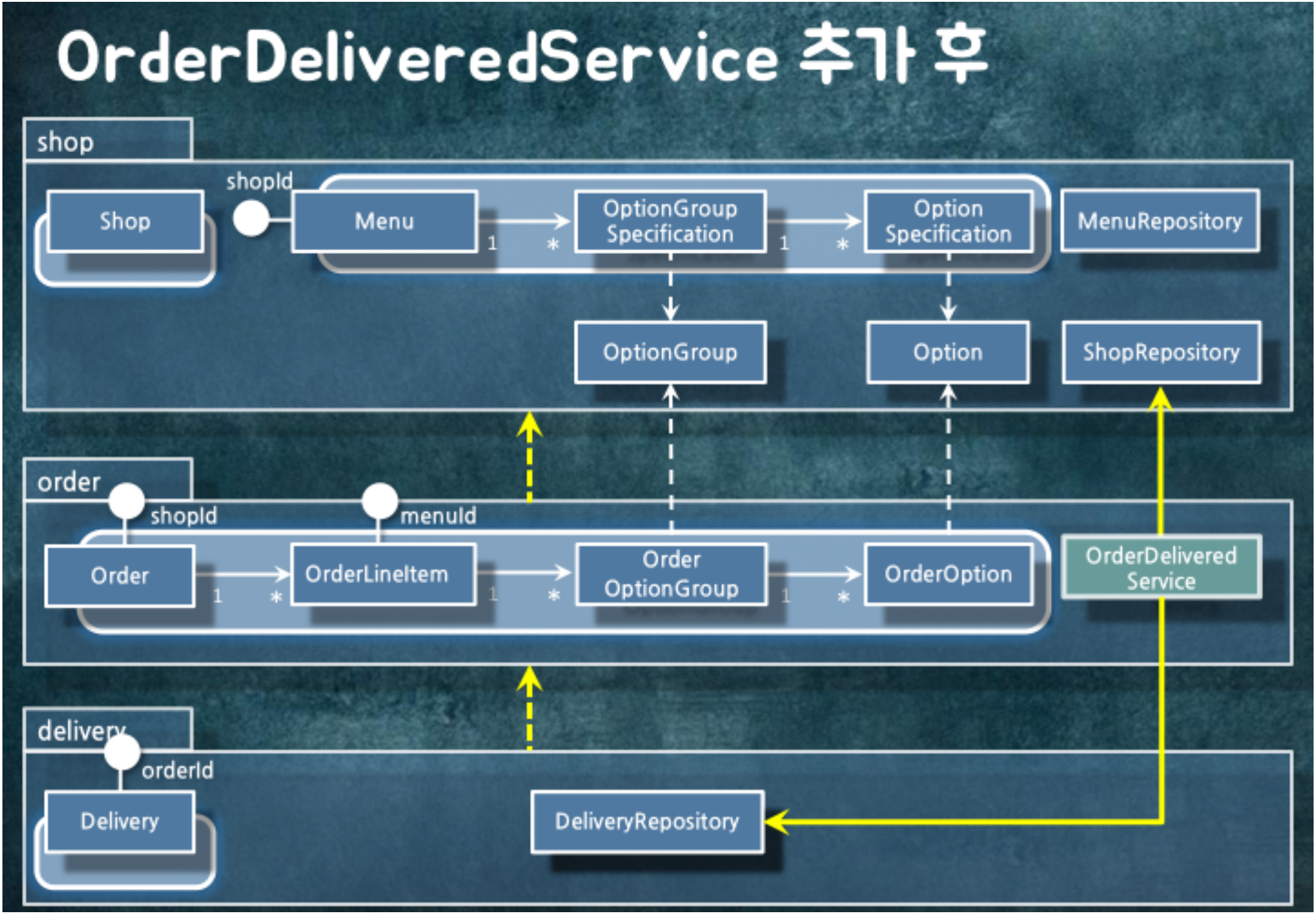
```
public class OrderDeliveredService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    Shop shop = shopRepository.findById(ordet.getShopId()) ...
    Delivery delivery = deliveryRepository.findById(orderId) ...

    order.delivered();
    shop.billCommissionFee(order.calculateTotalPrice());
    delivery.complete();
  }
}
```

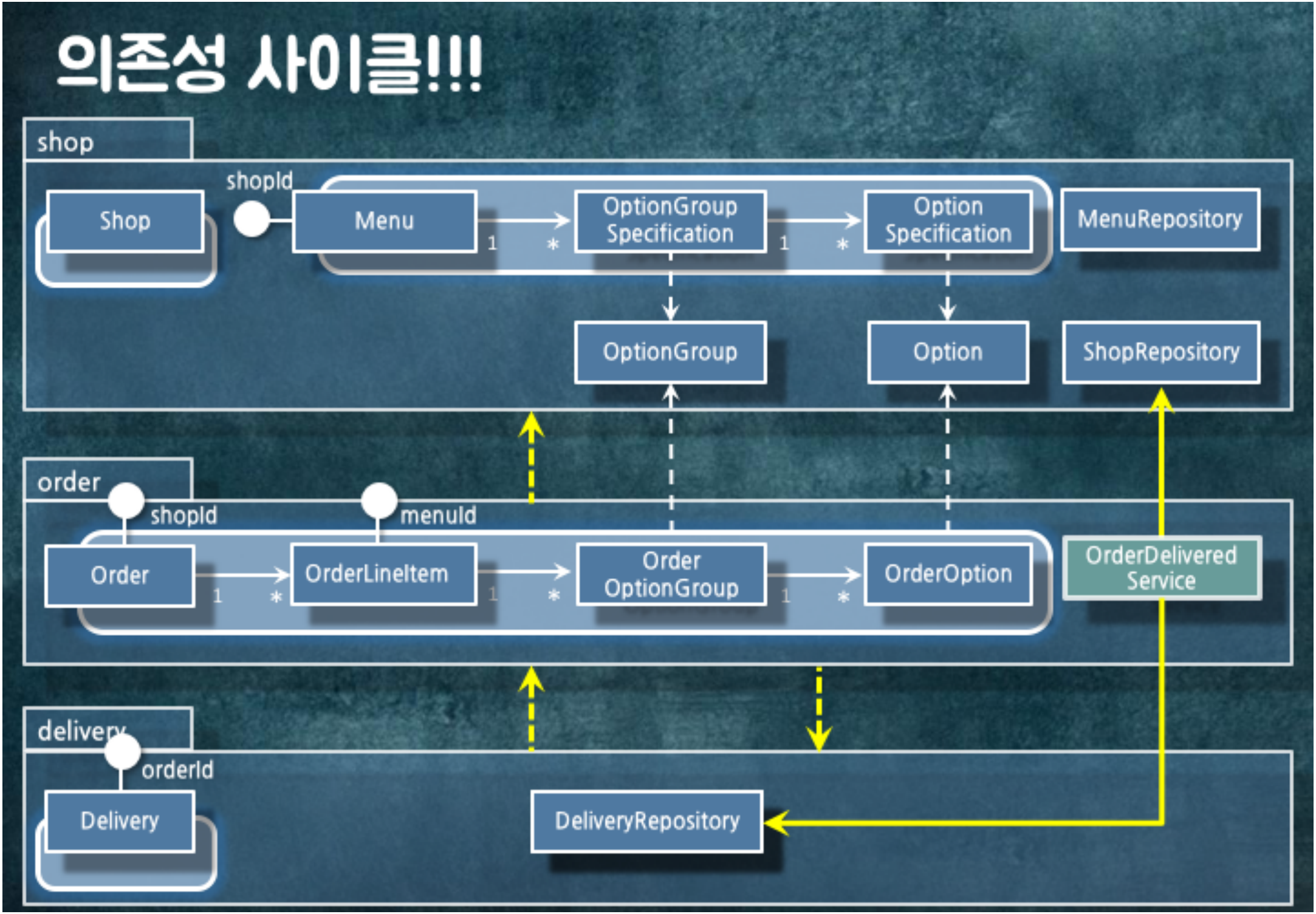
OrderService 의존성 주입

```java
public class OrderService {
  private OrderDeliveredService orderDeliveredService;

  @Transactional
  public void deliverOrder(Long orderId) {
    orderDeliveredService.deliverOrder(orderId);
  }
}
```

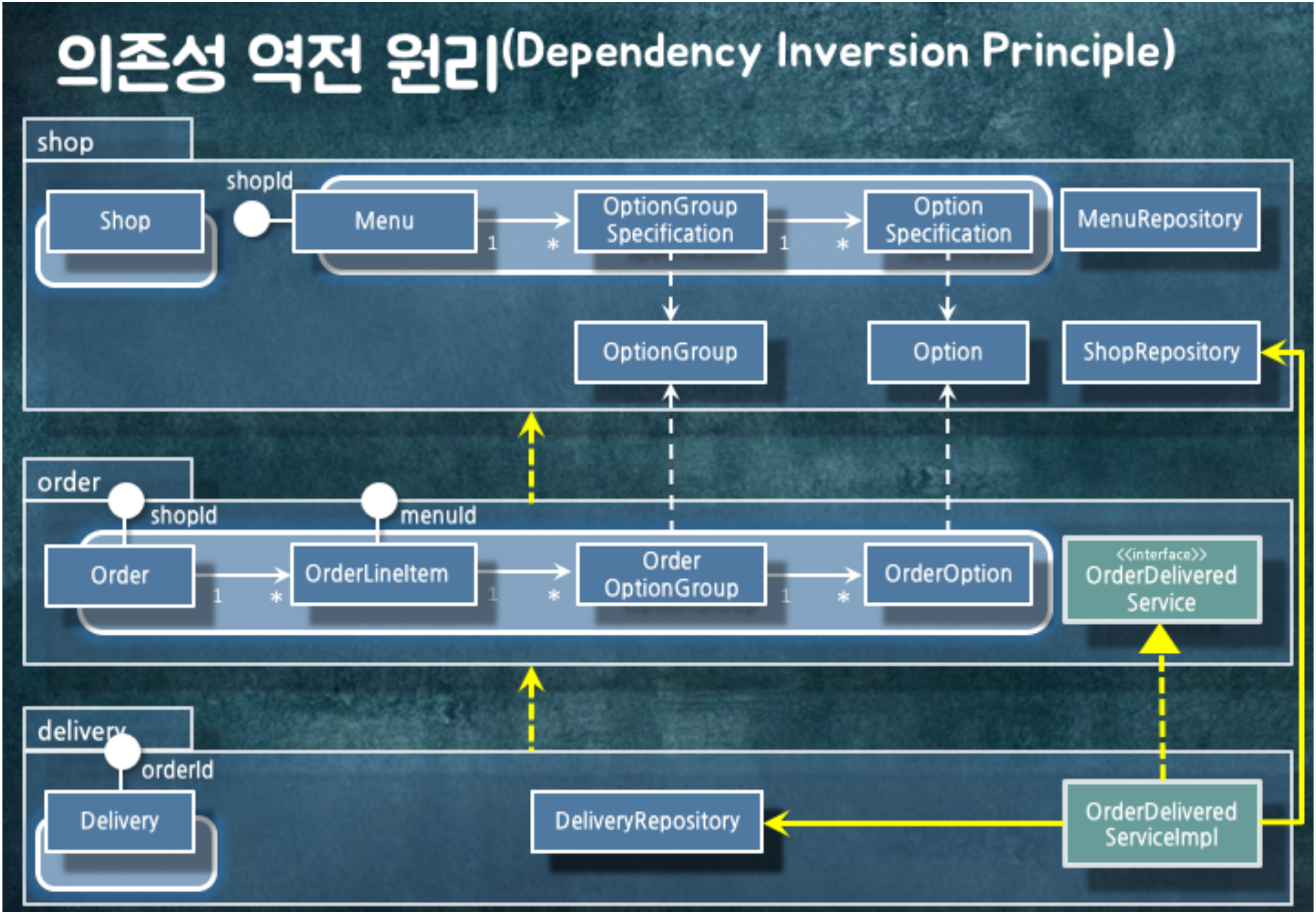
```java
@Component
public class OrderDeliveredService {
  @Transactional
  public void deliverOrder(Long orderId) {
    Order order = orderRepository.findById(orderId) ...
    Shop shop = shopRepository.findById(order.getShopId()) ...
    Delivery delivery = deliveryRepository.findById(orderId) ...

    order.delivered();
    shop.billCommissionFee(order.calculateTotalPrice());
    delivery.complete();
  }
}
```


OrderDeliveredService 추가 전
shop
shopId
Shop
Menu
1
*
OptionGroup Specification
1
*
Option Specification
MenuRepository
OptionGroup
Option
ShopRepository
order
shopId
menuId
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
delivery
orderId
Delivery
DeliveryRepository

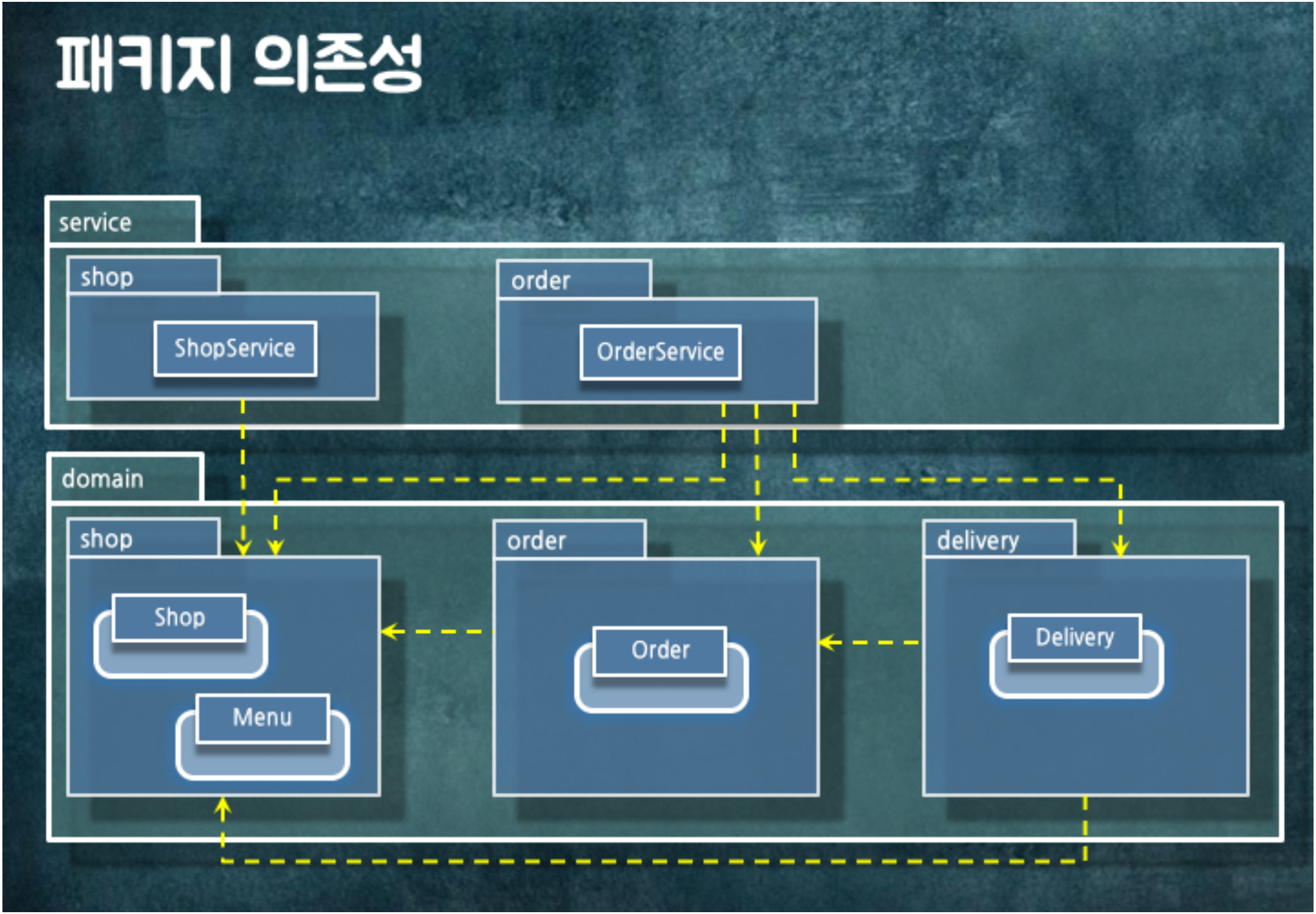

OrderDeliveredService 추가 후
shop
shopId
Shop
Menu
1
*
OptionGroup Specification
1
*
Option Specification
MenuRepository
OptionGroup
Option
ShopRepository
order
shopId
menuId
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
OrderDelivered Service
delivery
orderId
Delivery
DeliveryRepository


의존성 사이클!!!
shop
shopId
Shop
Menu
1
*
OptionGroup Specification
1
*
Option Specification
MenuRepository
OptionGroup
Option
ShopRepository
order
shopId
menuId
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
OrderDelivered Service
delivery
orderId
Delivery
DeliveryRepository


인터페이스를 이용해서
의존성을 역전시키자


의존성 역전 원리(Dependency Inversion Principle)
shop
shopId
Shop
Menu
1
*
OptionGroup Specification
1
*
Option Specification
MenuRepository
OptionGroup
Option
ShopRepository
order
shopId
menuId
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
<<interface>>
OrderDelivered Service
delivery
orderId
Delivery
DeliveryRepository
OrderDelivered ServiceImpl

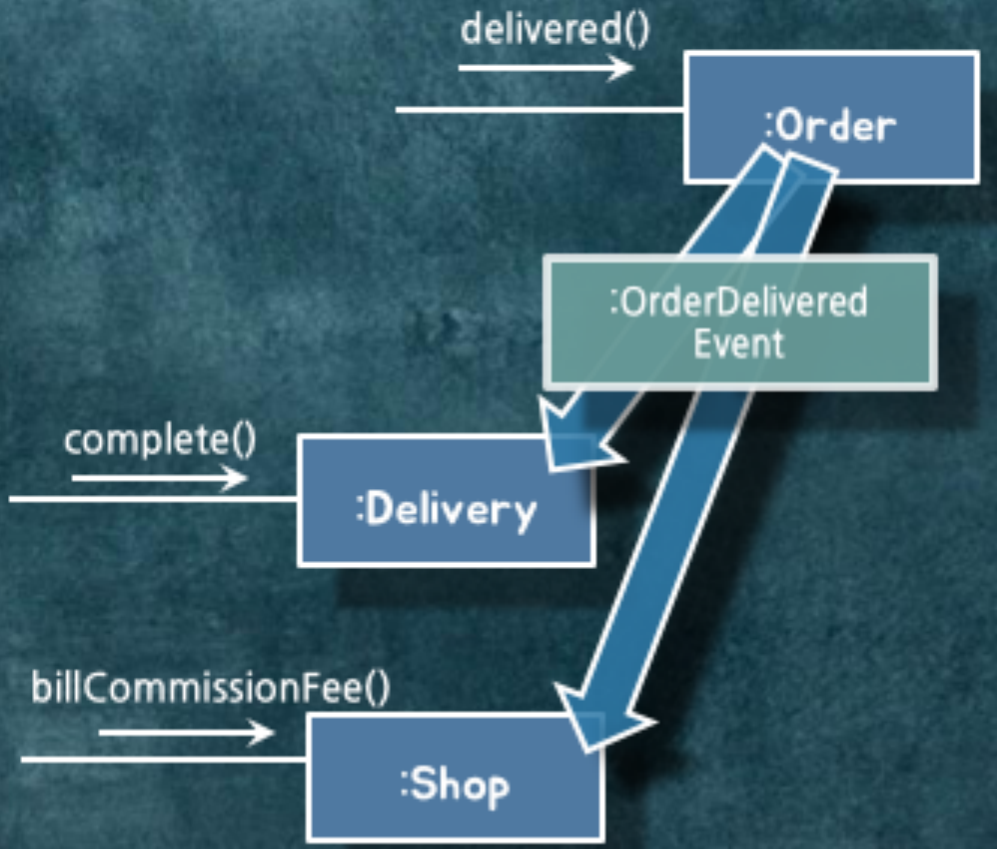
패키지 의존성


service
shop
ShopService
order
OrderService
domain
shop
Shop
Menu
order
Order
delivery
Delivery

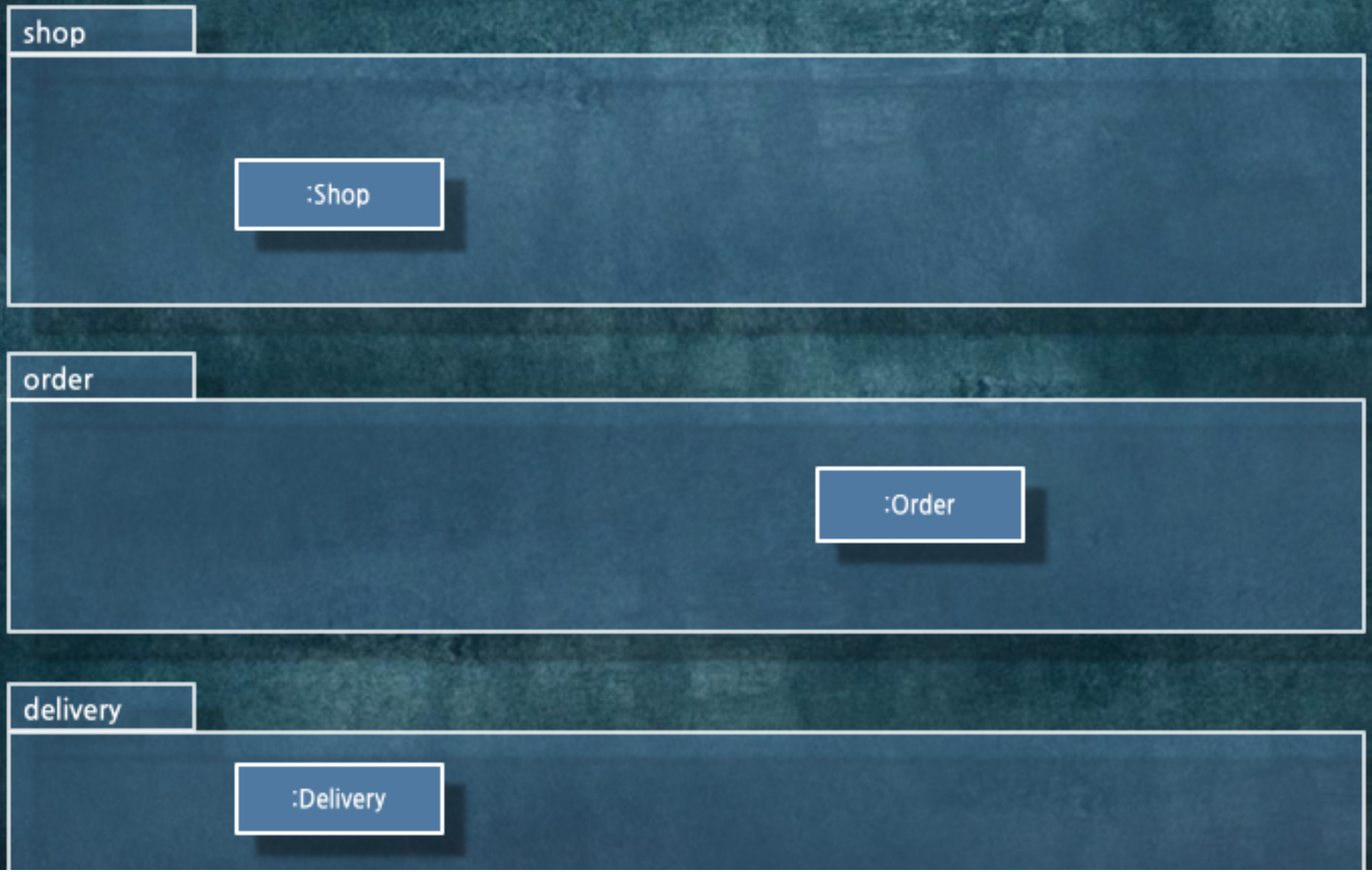
두번째 방법

절차지향 로직(OrderValidator 방식)

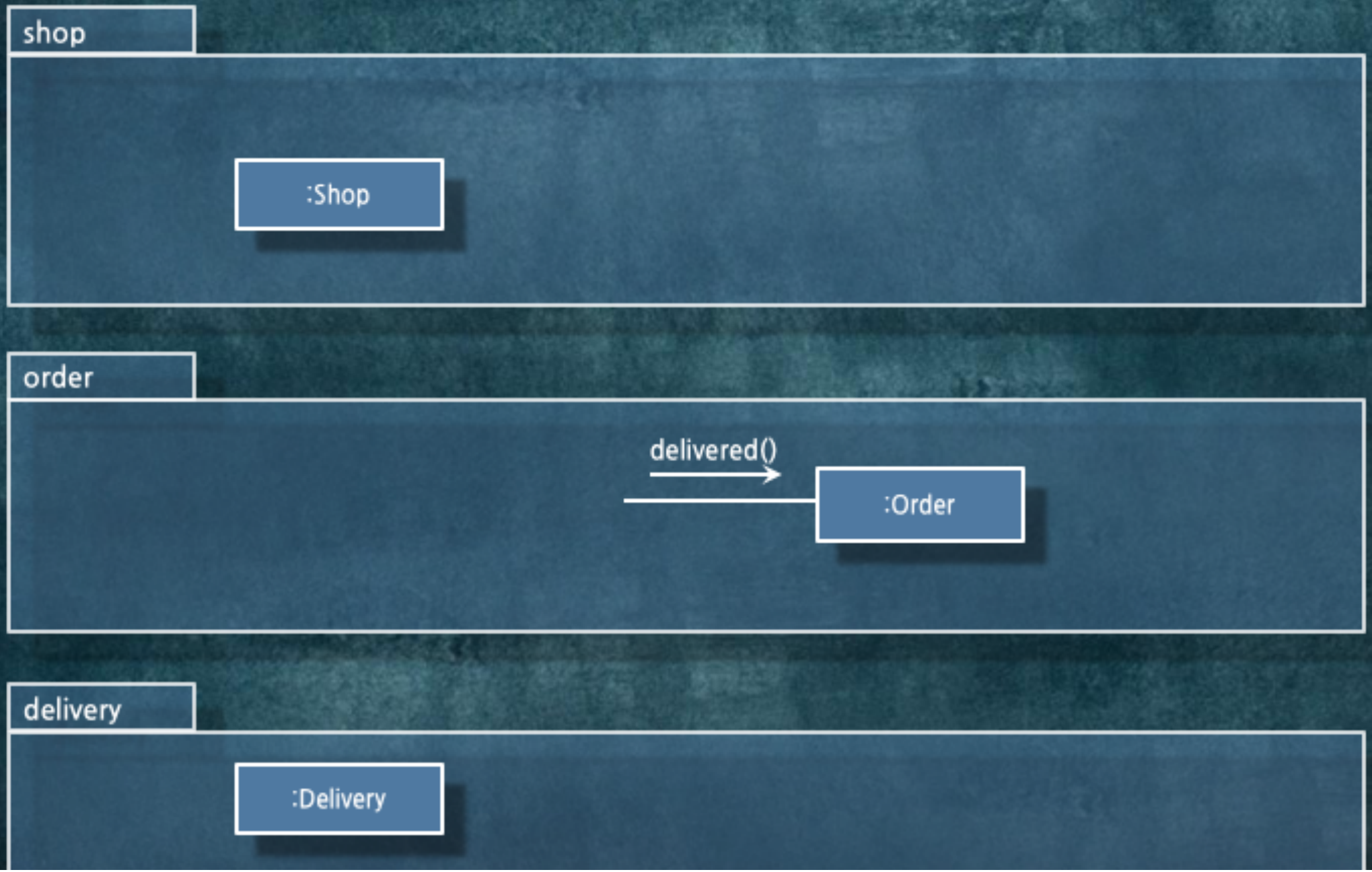
도메인 이벤트Domain Event 퍼블리싱



Domain Event를 이용한 의존성 제거

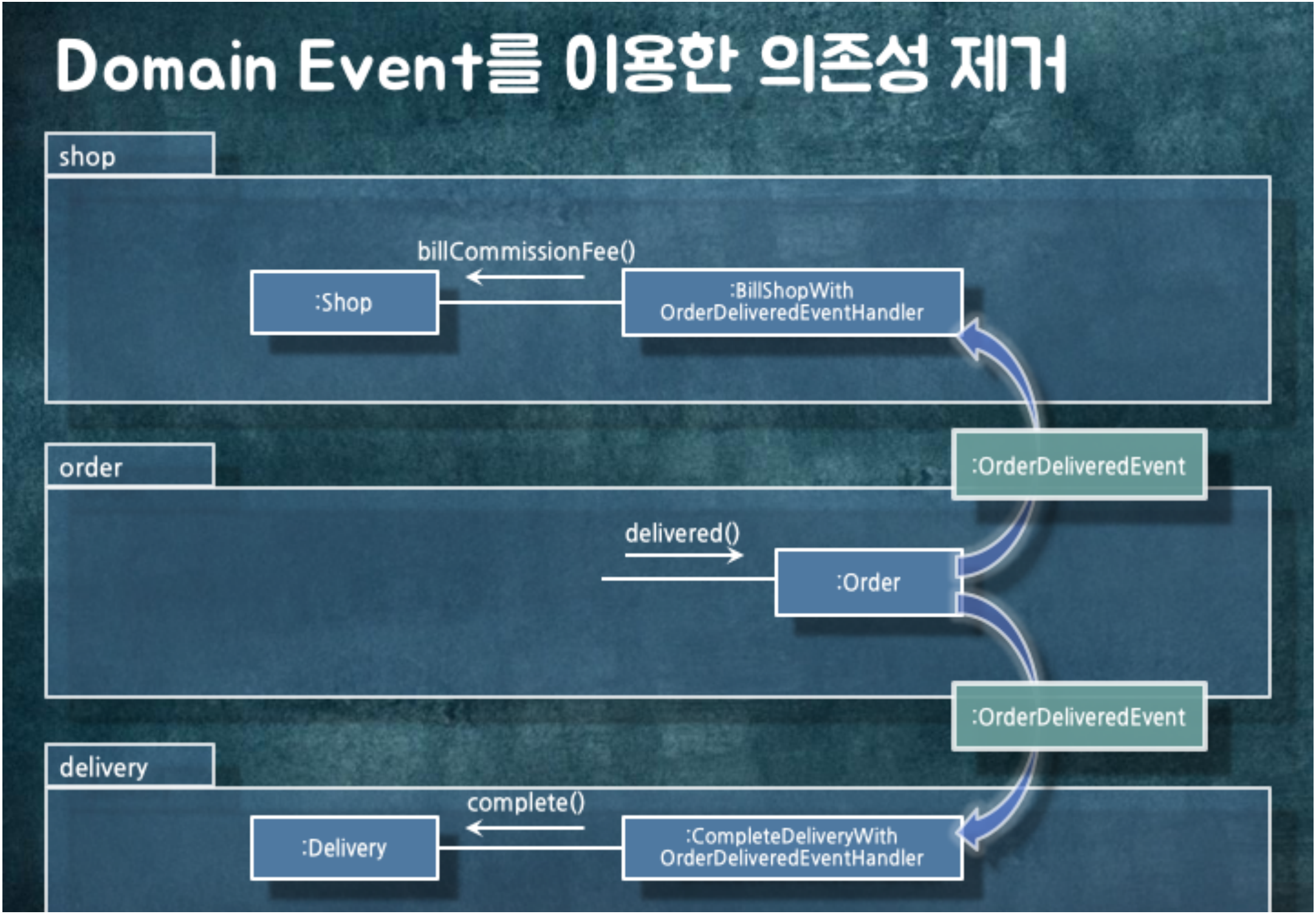


Domain Event를 이용한 의존성 제거



Domain Event를 이용한 의존성 제거

Domain Event를 이용한 의존성 제거



Order가 Shop을 직접 호출하던 로직을

```
public class Order {
  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

Order가 Domain Event 발행하도록 수정

```
public class Order {
  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    this.shop.billCommissionFee(calculateTotalPrice());
  }
}
```

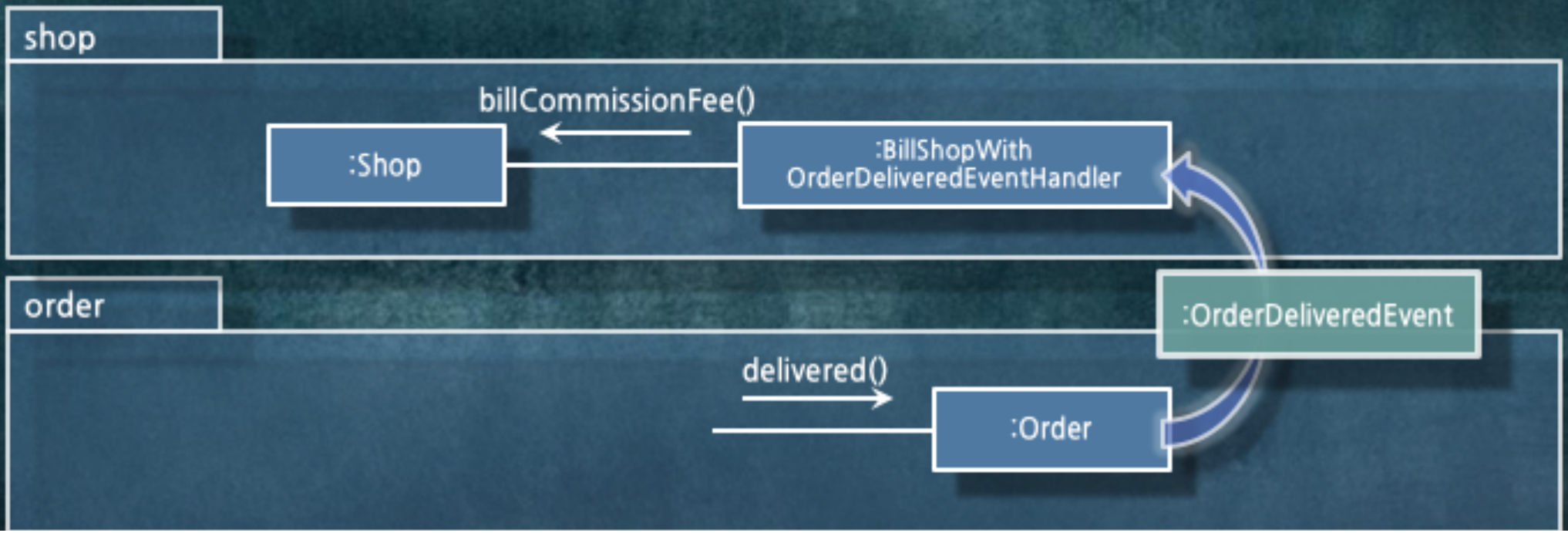

```
public class Order extends AbstractAggregateRoot<Order> {
  public void delivered() {
    this.orderStatus = OrderStatus.DELIVERED;
    registerEvent(new OrderDeliveredEvent(this));
  }
}
```

```
public class OrderDeliveredEvent {
  private Order order;

  public Long getOrderId() {...}
  public Long getShopId() { ...}
  public Money getTotalPrice() {...}
}
```

* Spring Data Aggregate Abstraction 이용

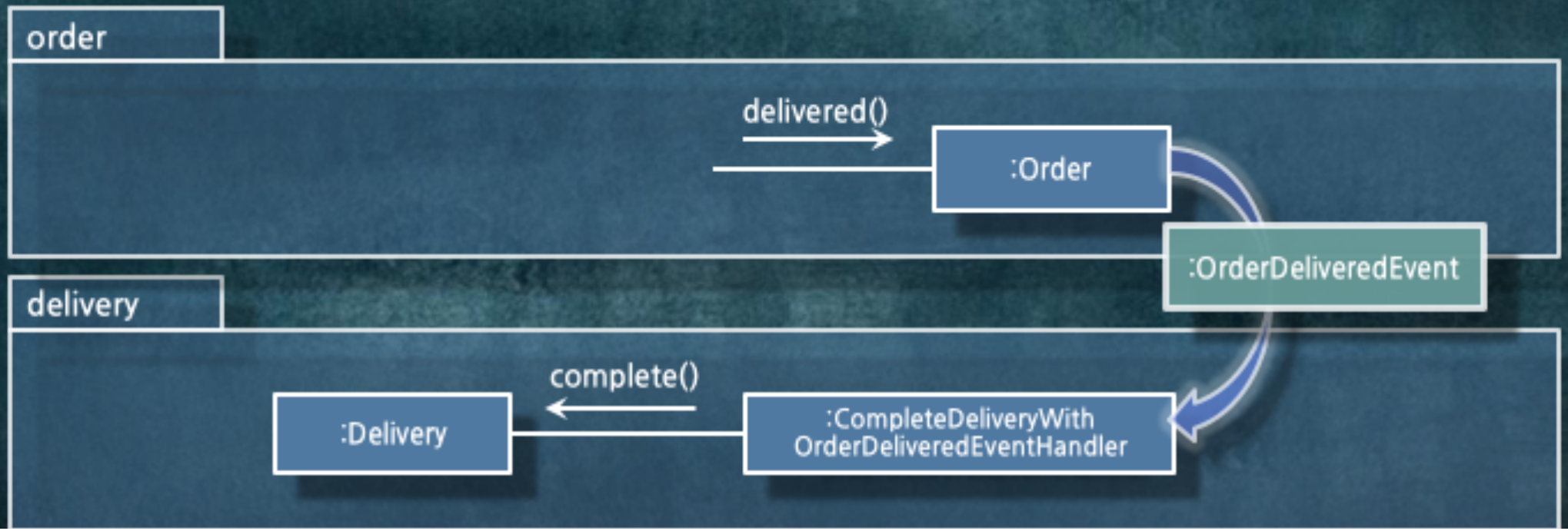
Shop 이벤트 핸들러

```java
@Component
public class BillShopWithOrderDeliveredEventHandler {
  @Async
  @EventListener
  @Transactional
  public void handle(OrderDeliveredEvent event) {
    Shop shop = shopRepository.findById(event.getShopId()) ...
    shop.billCommissionFee(event.getTotalPrice());
  }
}
```

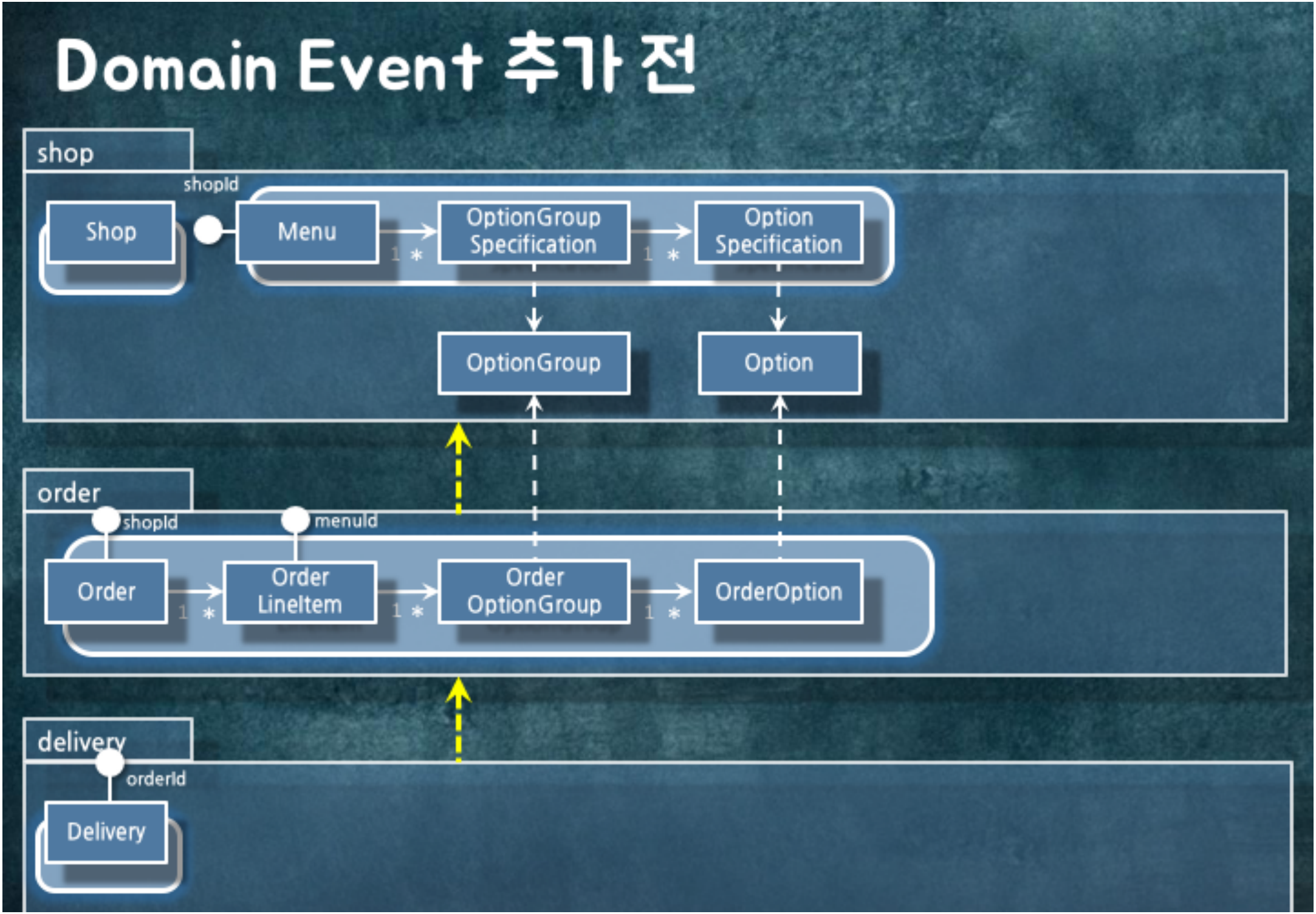


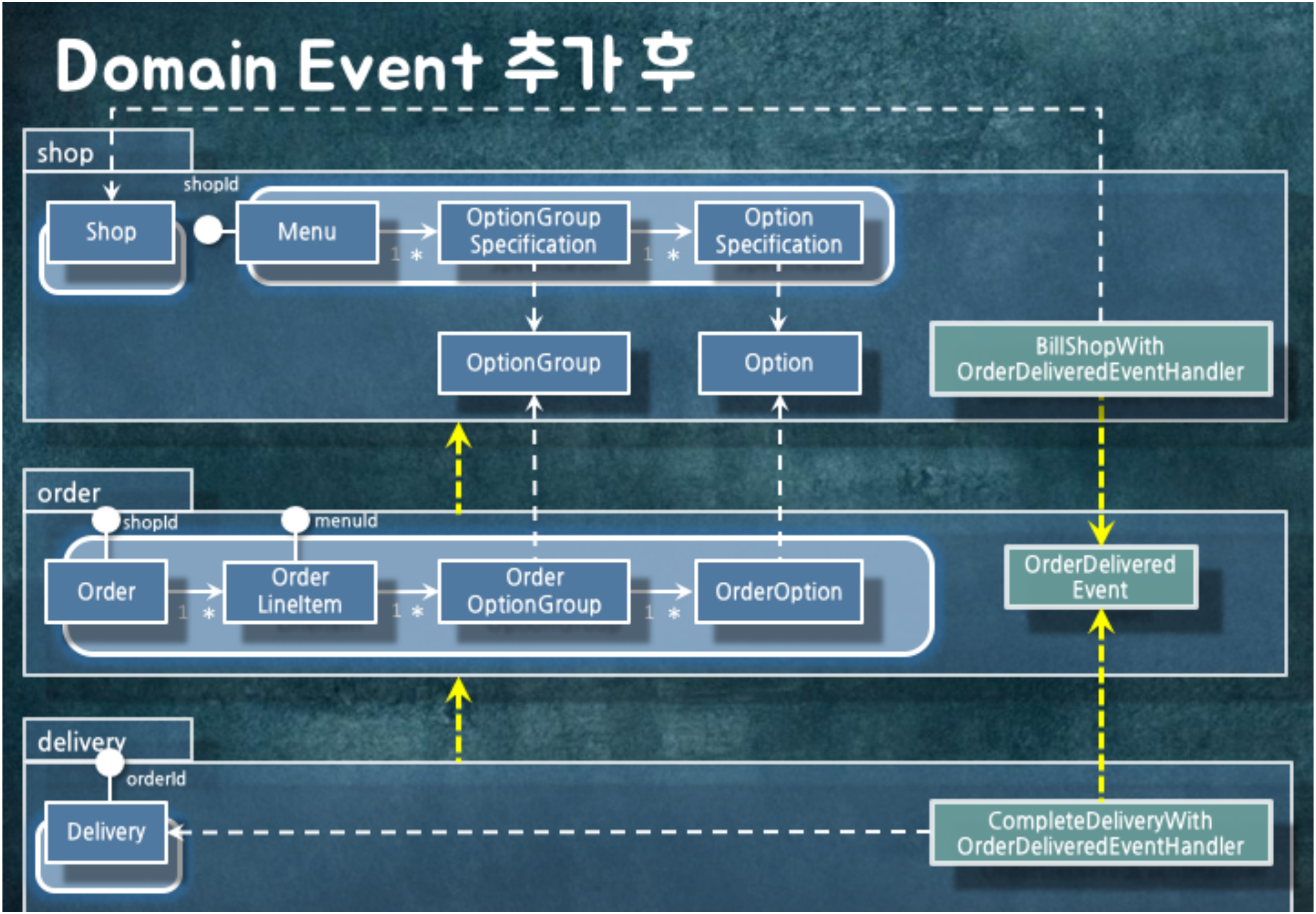
Delivery 이벤트 핸들러

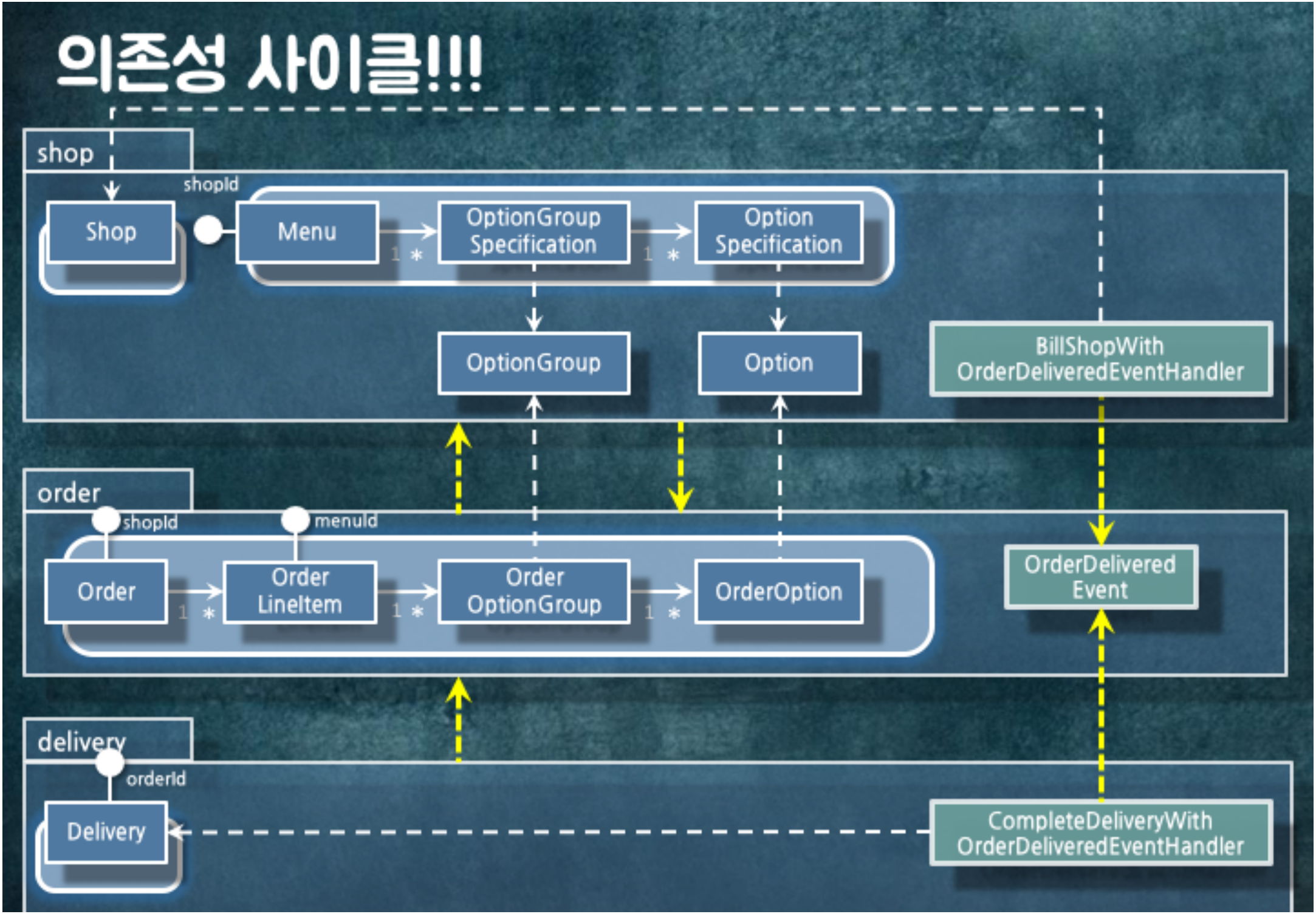
```
@Component
public class CompleteDeliveryWithOrderDeliveredEventHandler {
  @Async
  @EventListener
  @Transactional
  public void handle(OrderDeliveredEvent event) {
    Delivery delivery = deliveryRepository.findById(event.getOrderId()) ...
    delivery.complete();
  }
}
```

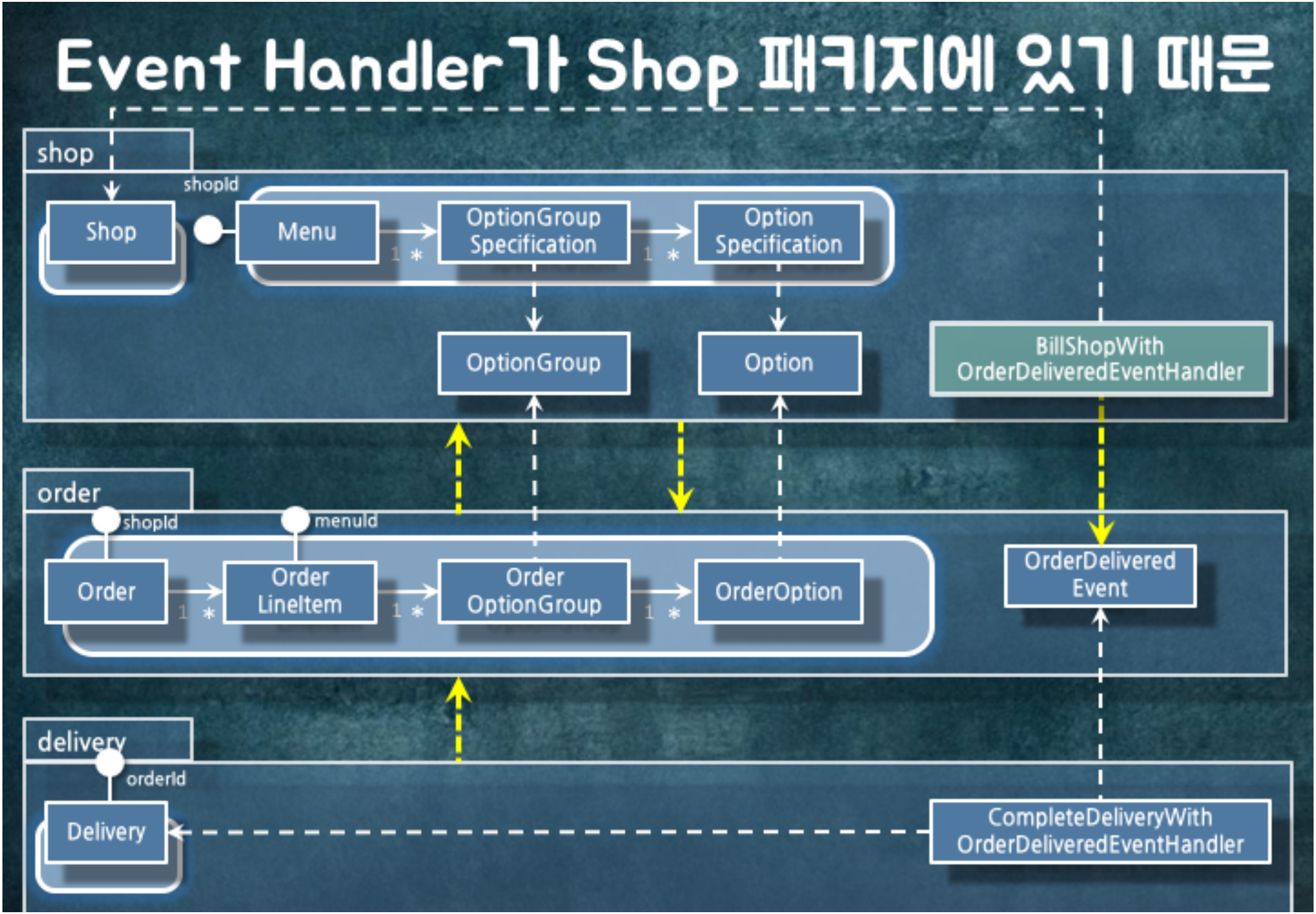


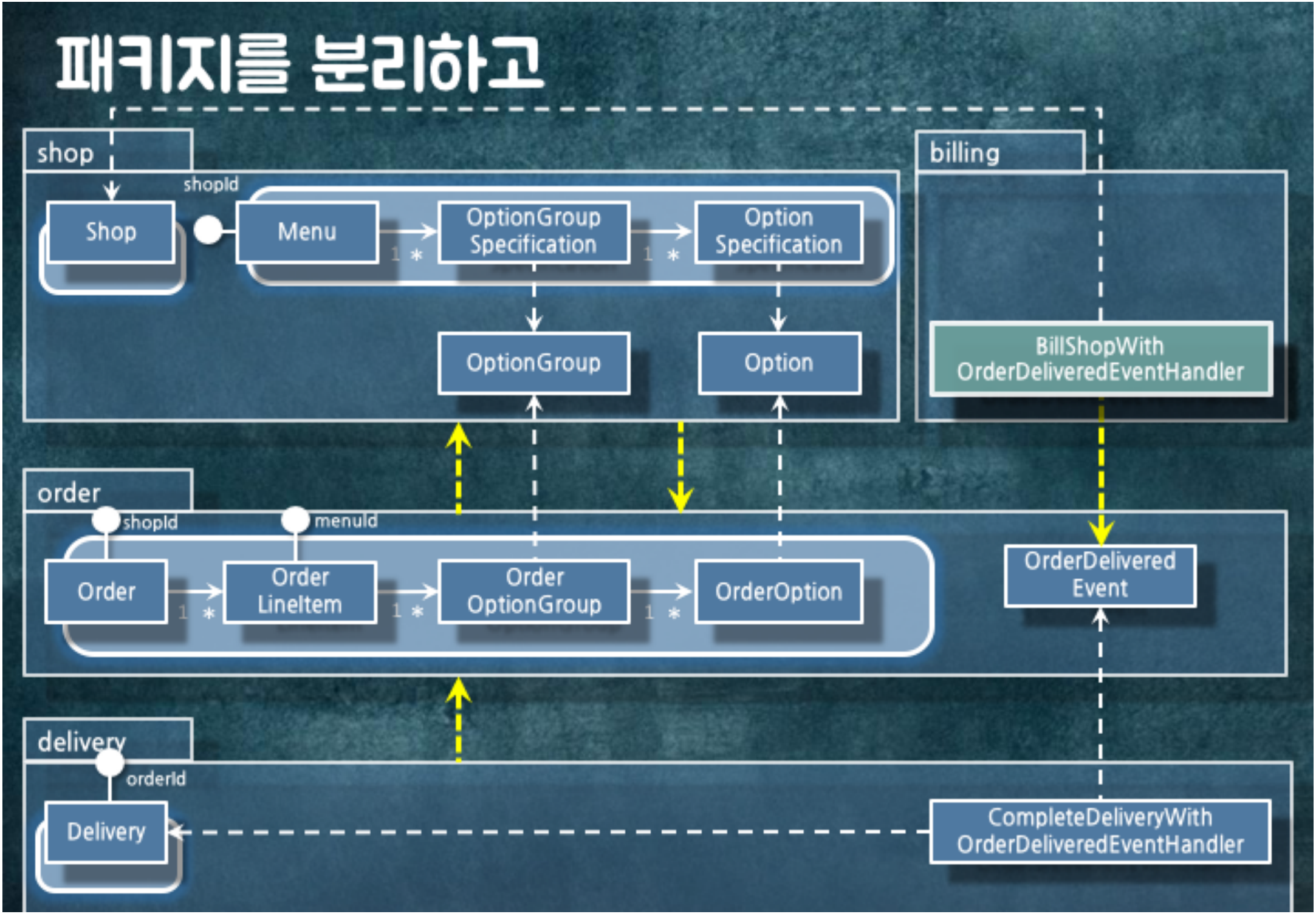
Domain Event 추가 전




Domain Event 추가 후
shop
shopId
Shop
Menu
1 *
OptionGroup Specification
1 *
Option Specification
OptionGroup
Option
BillShopWith OrderDeliveredEventHandler
order
shopId
menuId
Order
1 *
Order LineItem
1 *
Order OptionGroup
1 *
OrderOption
OrderDelivered Event
delivery
orderId
Delivery
CompleteDeliveryWith OrderDeliveredEventHandler


의존성 사이클!!!
shop
shopId
Shop
Menu
1 *
OptionGroup Specification
1 *
Option Specification
OptionGroup
Option
BillShopWith OrderDeliveredEventHandler
order
shopId
menuId
Order
1 *
Order LineItem
1 *
Order OptionGroup
1 *
OrderOption
OrderDelivered Event
delivery
orderId
Delivery
CompleteDeliveryWith OrderDeliveredEventHandler


Event Handler가 Shop 패키지에 있기 때문
shop
shopId
Shop
Menu
1 *
OptionGroup Specification
1 *
Option Specification
OptionGroup
Option
BillShopWith OrderDeliveredEventHandler
order
shopId
menuId
Order
1 *
Order LineItem
1 *
Order OptionGroup
1 *
OrderOption
OrderDelivered Event
delivery
orderId
Delivery
CompleteDeliveryWith OrderDeliveredEventHandler


패키지를 분리하고
shop
shopId
Shop
Menu
1 *
OptionGroup Specification
1 *
Option Specification
OptionGroup
Option
billing
BillShopWith OrderDeliveredEventHandler
order
shopId
menuId
Order
1 *
Order LineItem
1 *
Order OptionGroup
1 *
OrderOption
OrderDelivered Event
delivery
orderId
Delivery
CompleteDeliveryWith OrderDeliveredEventHandler

Event Handler가 의존하는 코드를 Shop에서 분리

```java
public class Shop {
  private Ratio commissionRate;

  private Money commission = Money.ZERO;

  public void billCommissionFee(Money price) {
    commission = commission.plus(commissionRate.of(price));
  }
}
```

```java
public class Shop {
  private Ratio commissionRate;

  public Money calculateCommissionFee(Money price) {
    return commissionRate.of(price);
  }
}
```

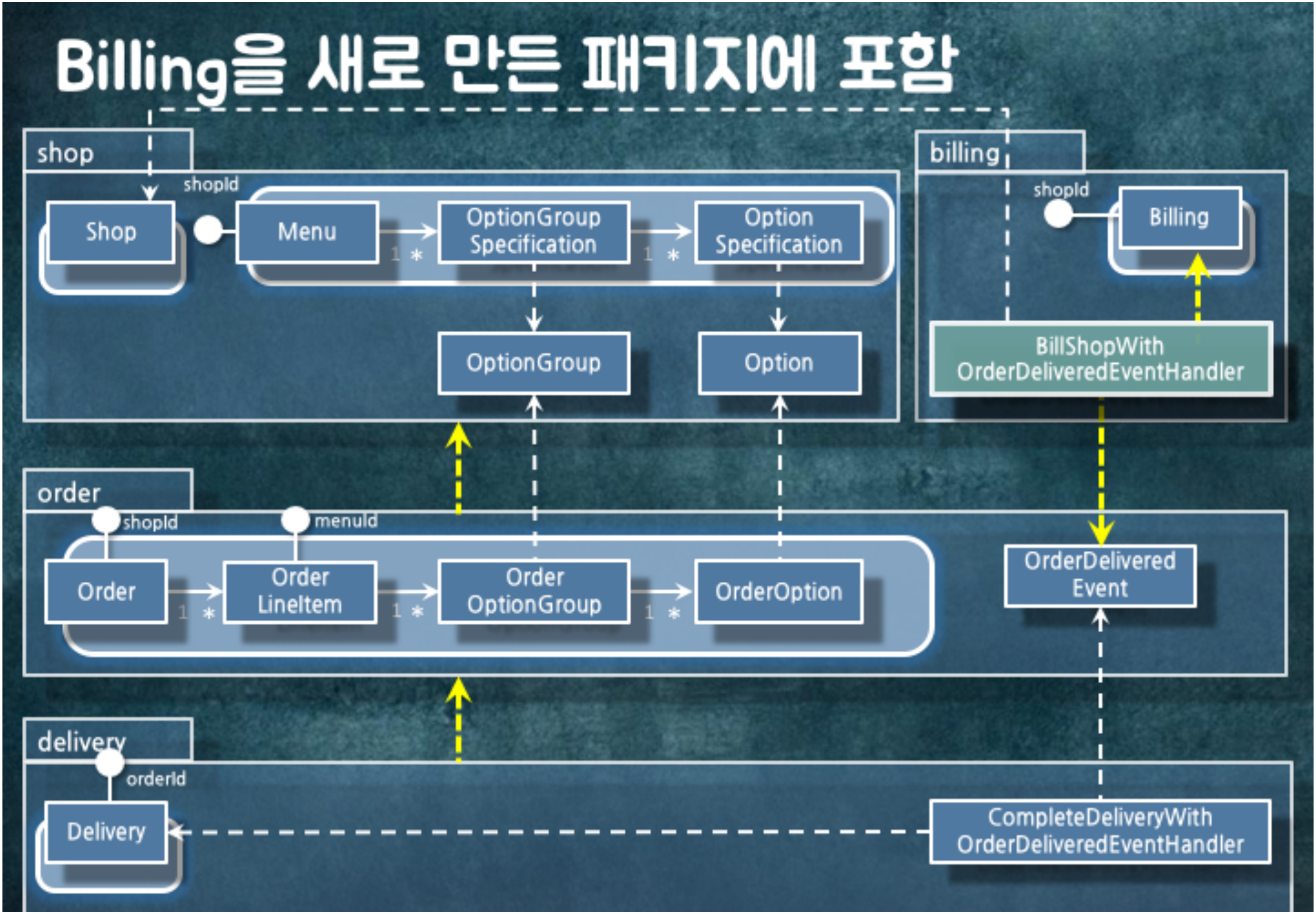
```java
public class Billing {
  private Long shopId;
  private Money commission = Money.ZERO;

  public void billCommissionFee(Money commission) {
    commission = commission.plus(commission);
  }
}
```

Event Handler에서 Shop과 Billing 사용

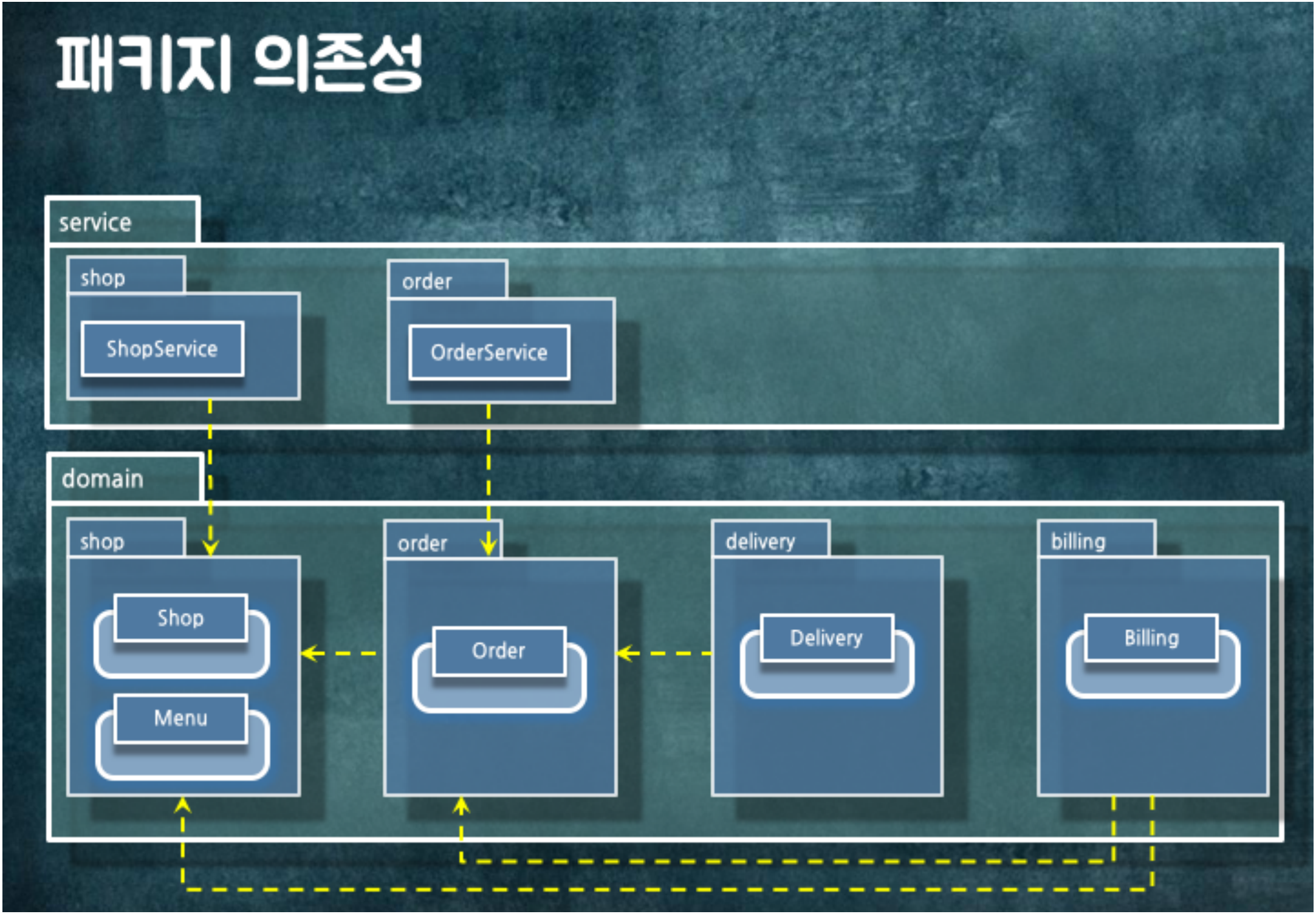
```java
@Component
public class BillShopWithOrderDeliveredEventHandler {
  @Async
  @EventListener
  @Transactional
  public void handle(OrderDeliveredEvent event) {
    Shop shop = shopRepository.findById(event.getShopId()) ...
    Billing billing = billingRepository.findByShopId(event.getShopId())
                                       .orElse(new Billing(event.getShopId()));

    billing.billCommissionFee(shop.calculateCommissionFee(event.getTotalPrice());
  }
}
```

```java
public class Shop {
  private Ratio commissionRate;

  public Money calculateCommissionFee(Money price) {
    return commissionRate.of(price);
  }
}
```

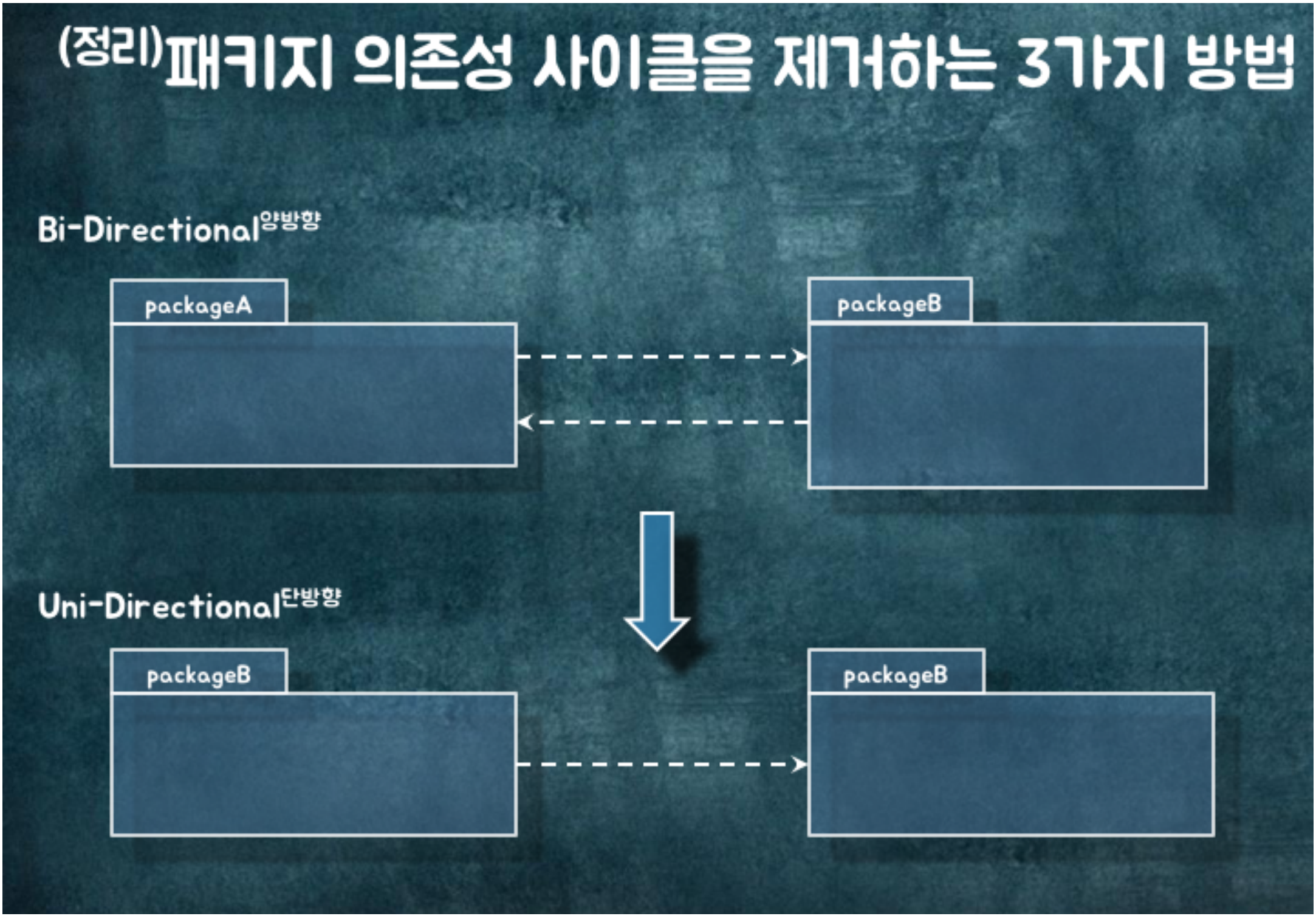
```java
public class Billing {
  private Long shopId;
  private Money commission = Money.ZERO;

  public void billCommissionFee(Money commission) {
    commission = commission.plus(commission);
  }
}
```


Billing을 새로 만든 패키지에 포함
shop
shopId
Shop
Menu
1 *
OptionGroup Specification
1 *
Option Specification
OptionGroup
Option
billing
shopId
Billing
BillShopWith OrderDeliveredEventHandler
order
shopId
menuId
Order
1 *
Order LineItem
1 *
Order OptionGroup
1 *
OrderOption
OrderDelivered Event
delivery
orderId
Delivery
CompleteDeliveryWith OrderDeliveredEventHandler

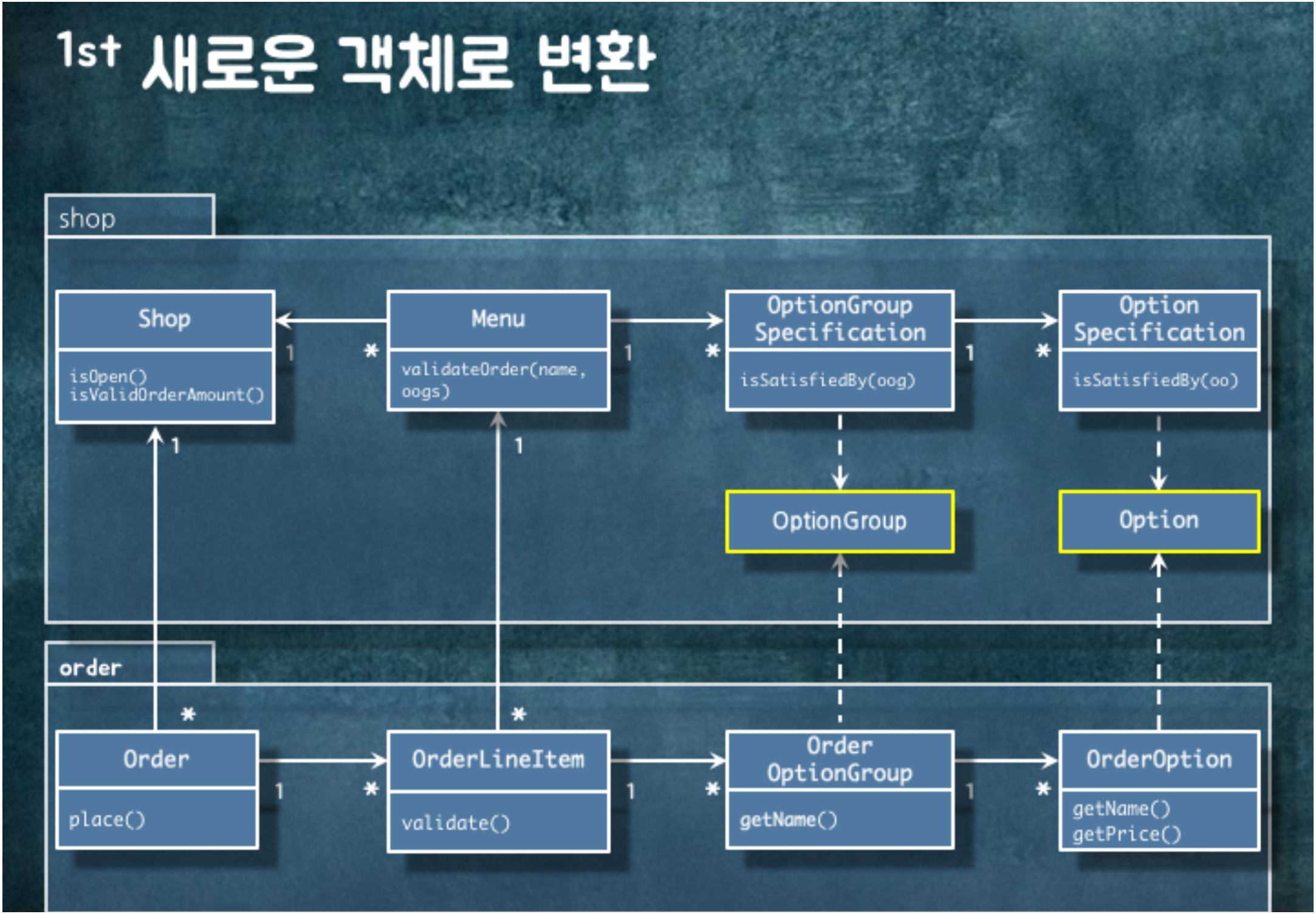
패키지 의존성

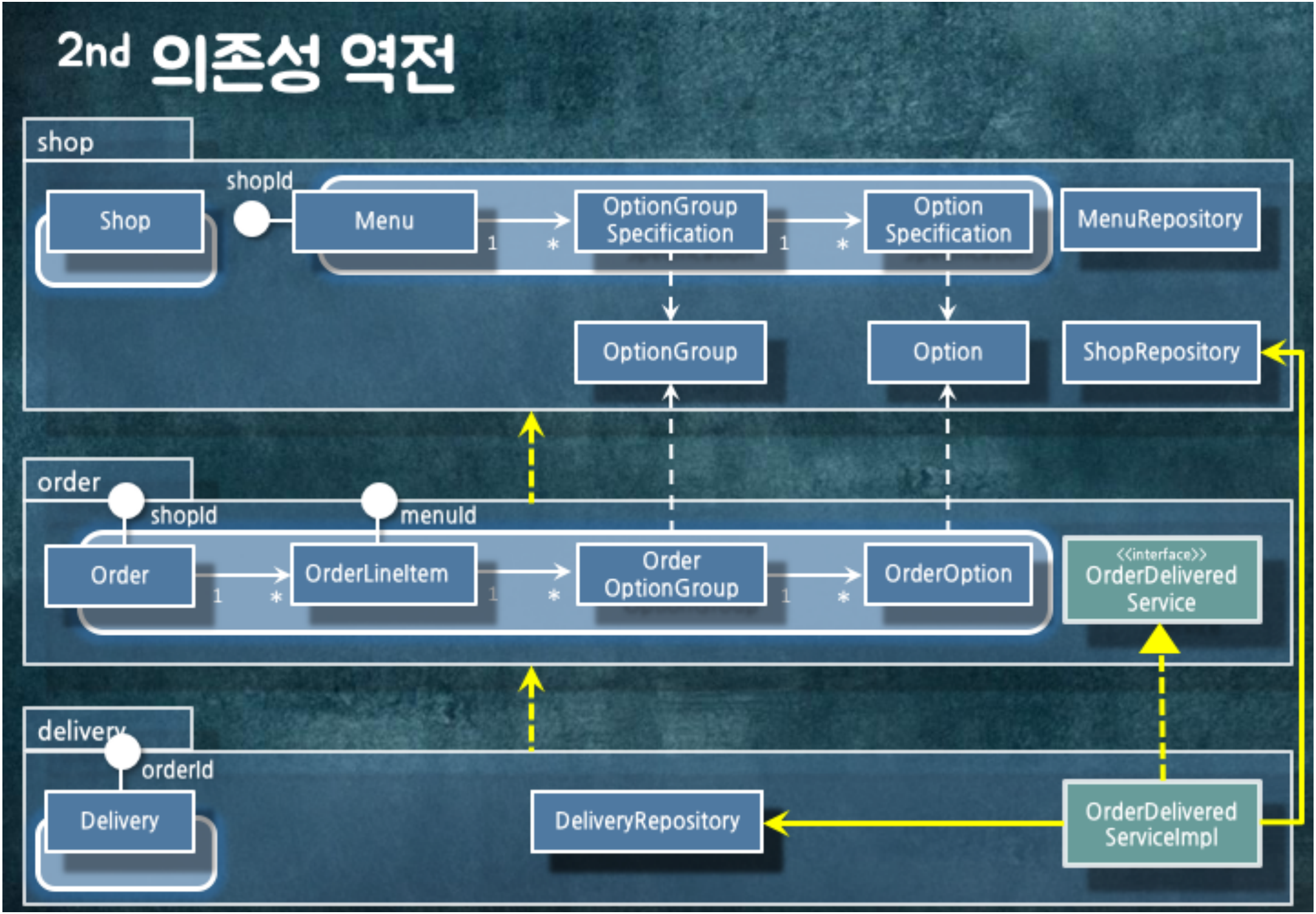

service
shop
ShopService
order
OrderService
domain
shop
Shop
Menu
order
Order
delivery
Delivery
billing
Billing

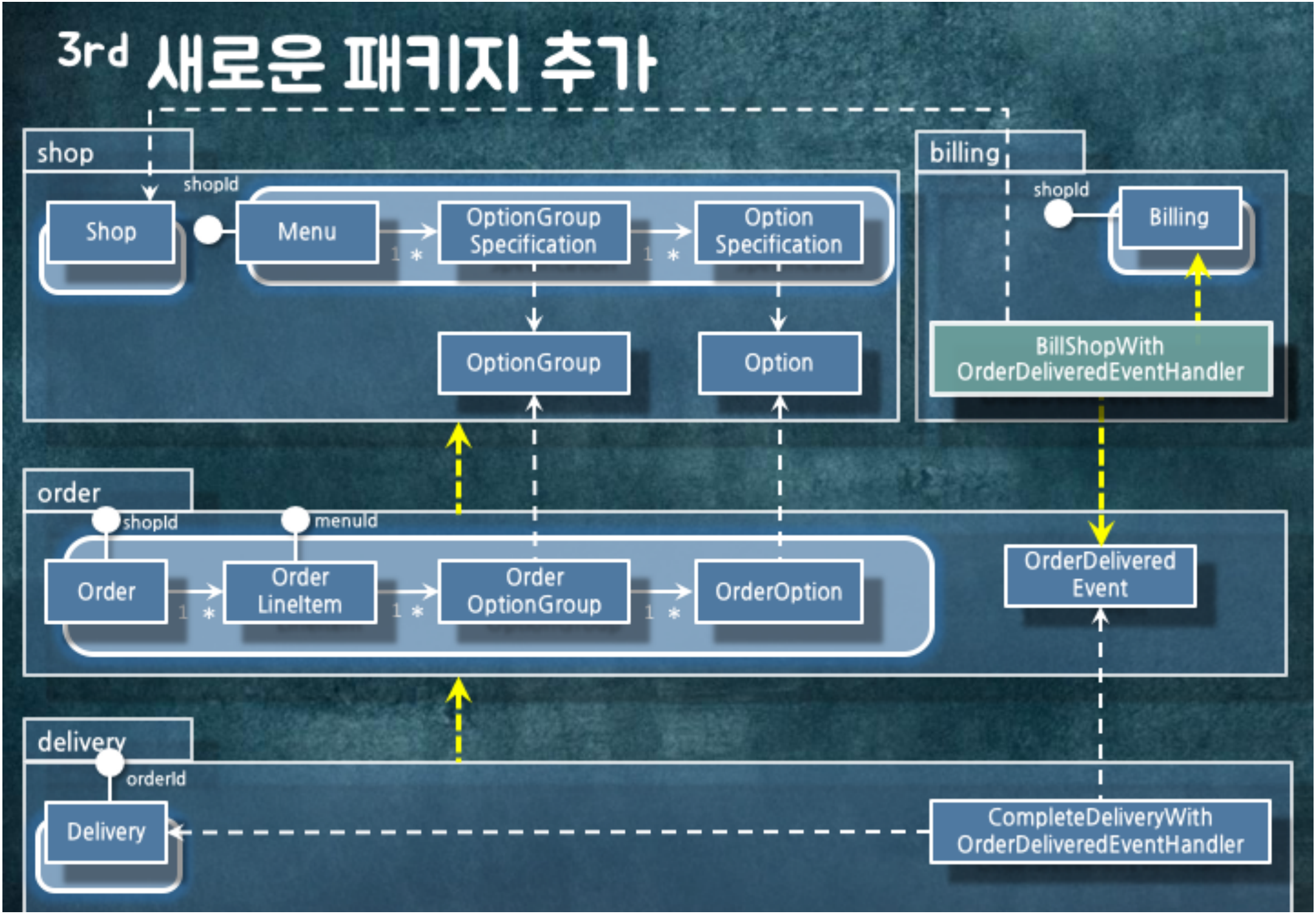
(정리)패키지 의존성 사이클을 제거하는 3가지 방법



1st 새로운 객체로 변환



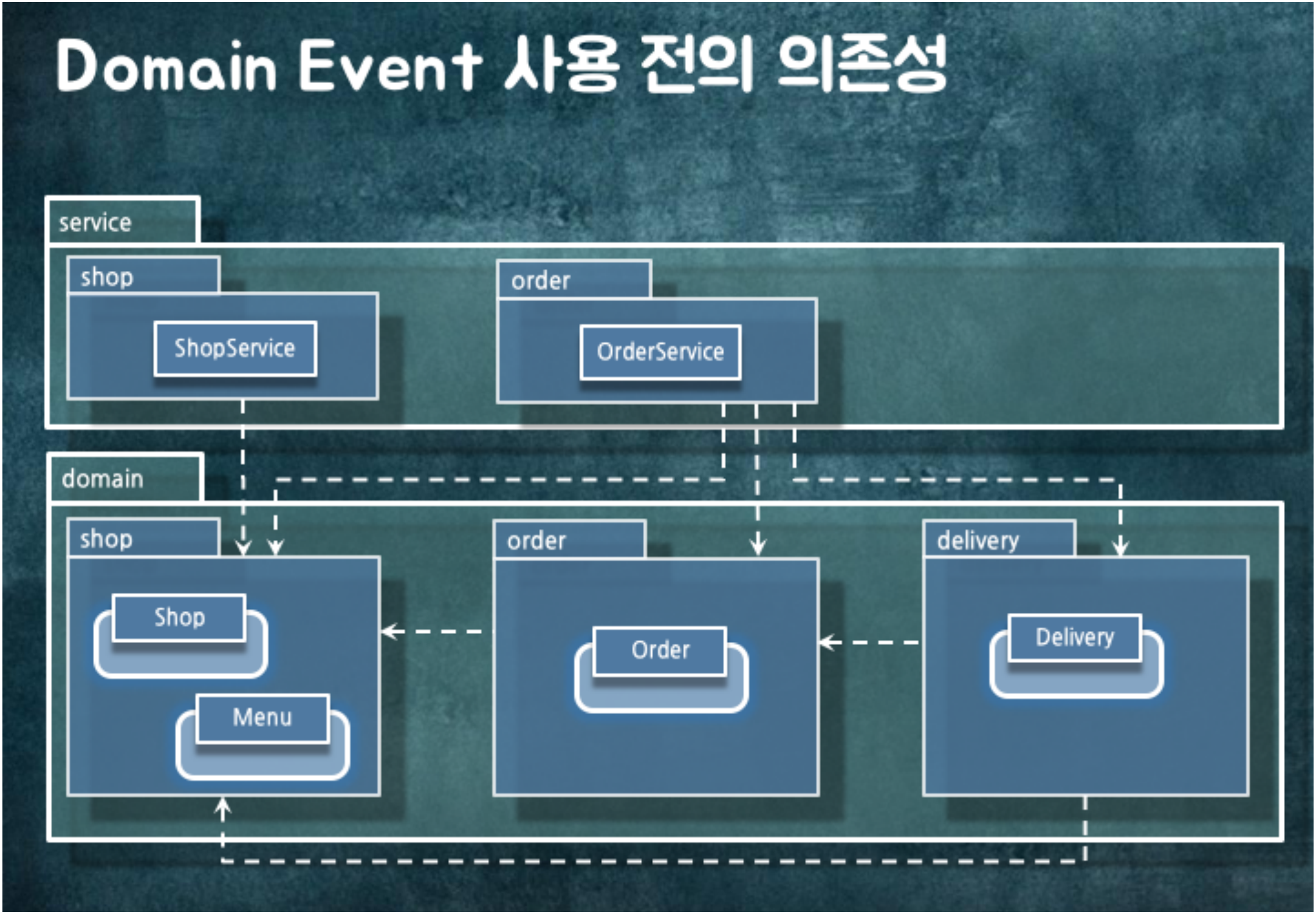

2nd 의존성 역전
shop
shopId
Shop
Menu
1
*
OptionGroup Specification
1
*
Option Specification
MenuRepository
OptionGroup
Option
ShopRepository
order
shopId
menuId
Order
1
*
OrderLineItem
1
*
Order OptionGroup
1
*
OrderOption
<<interface>>
OrderDelivered Service
delivery
orderId
Delivery
DeliveryRepository
OrderDelivered ServiceImpl

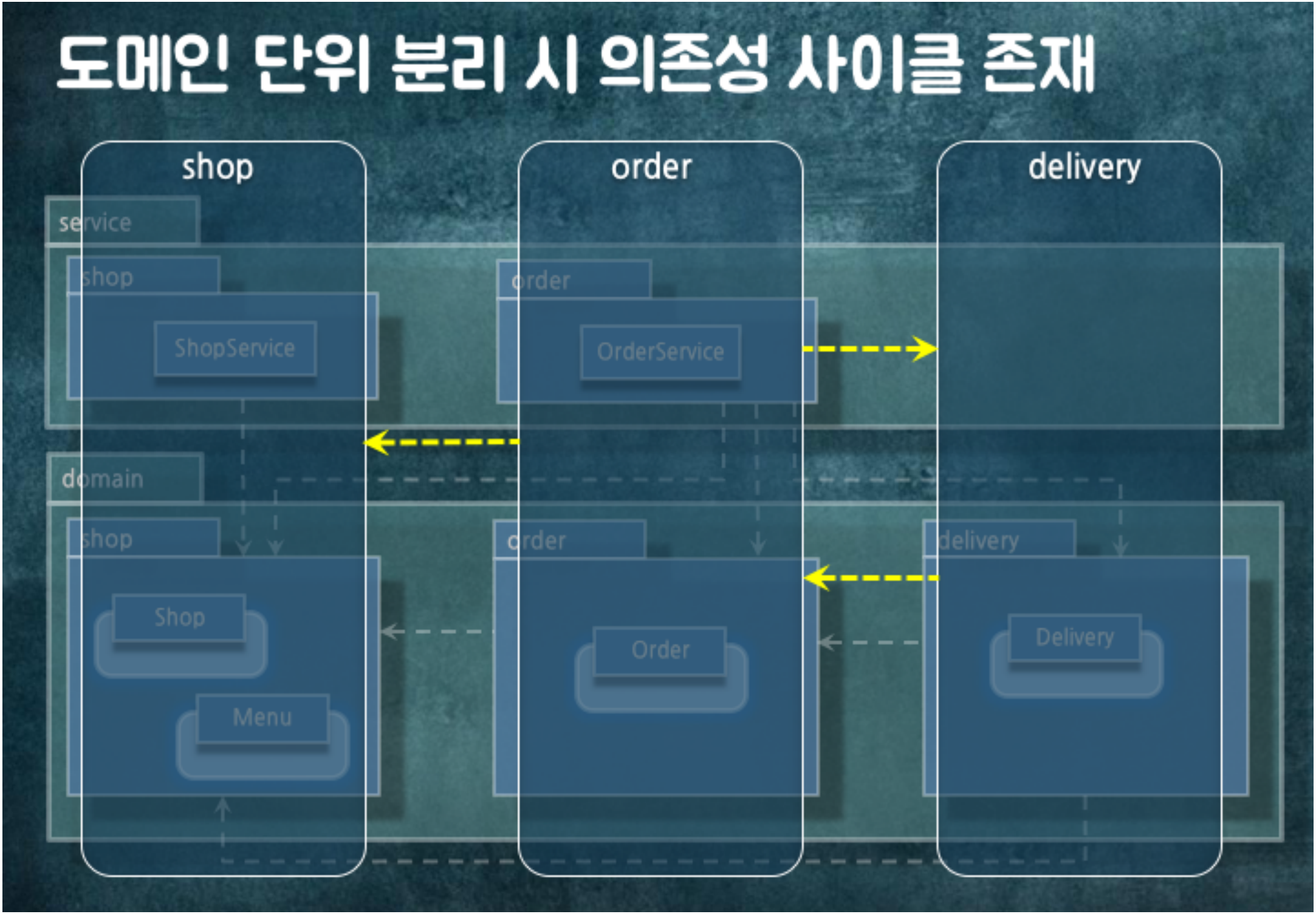

3rd 새로운 패키지 추가
shop
shopId
Shop
Menu
1 *
OptionGroup Specification
1 *
Option Specification
OptionGroup
Option
billing
shopId
Billing
BillShopWith OrderDeliveredEventHandler
order
shopId
menuId
Order
1 *
Order LineItem
1 *
Order OptionGroup
1 *
OrderOption
OrderDelivered Event
delivery
orderId
Delivery
CompleteDeliveryWith OrderDeliveredEventHandler

Part - 4

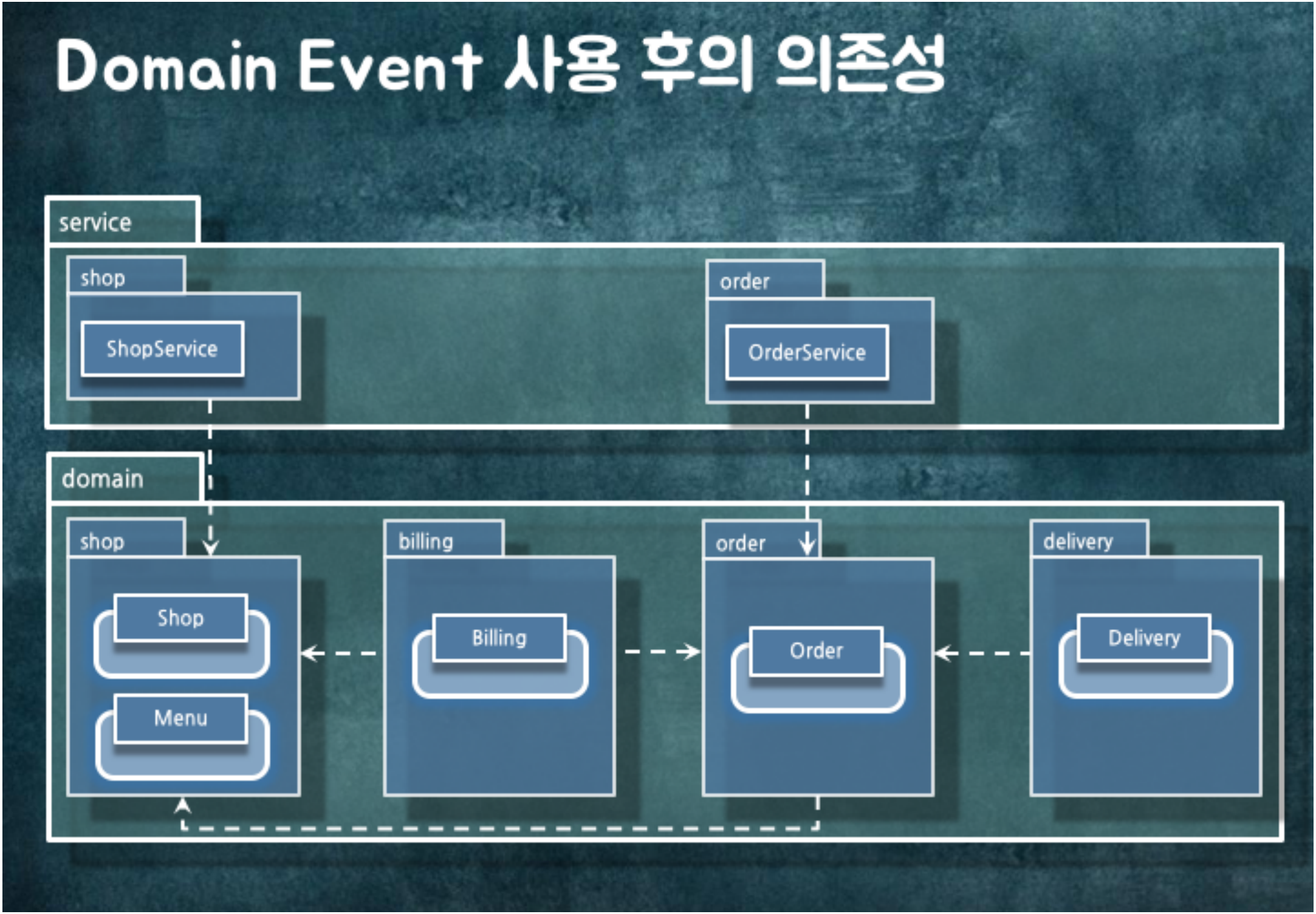
의존성과 시스템 분리

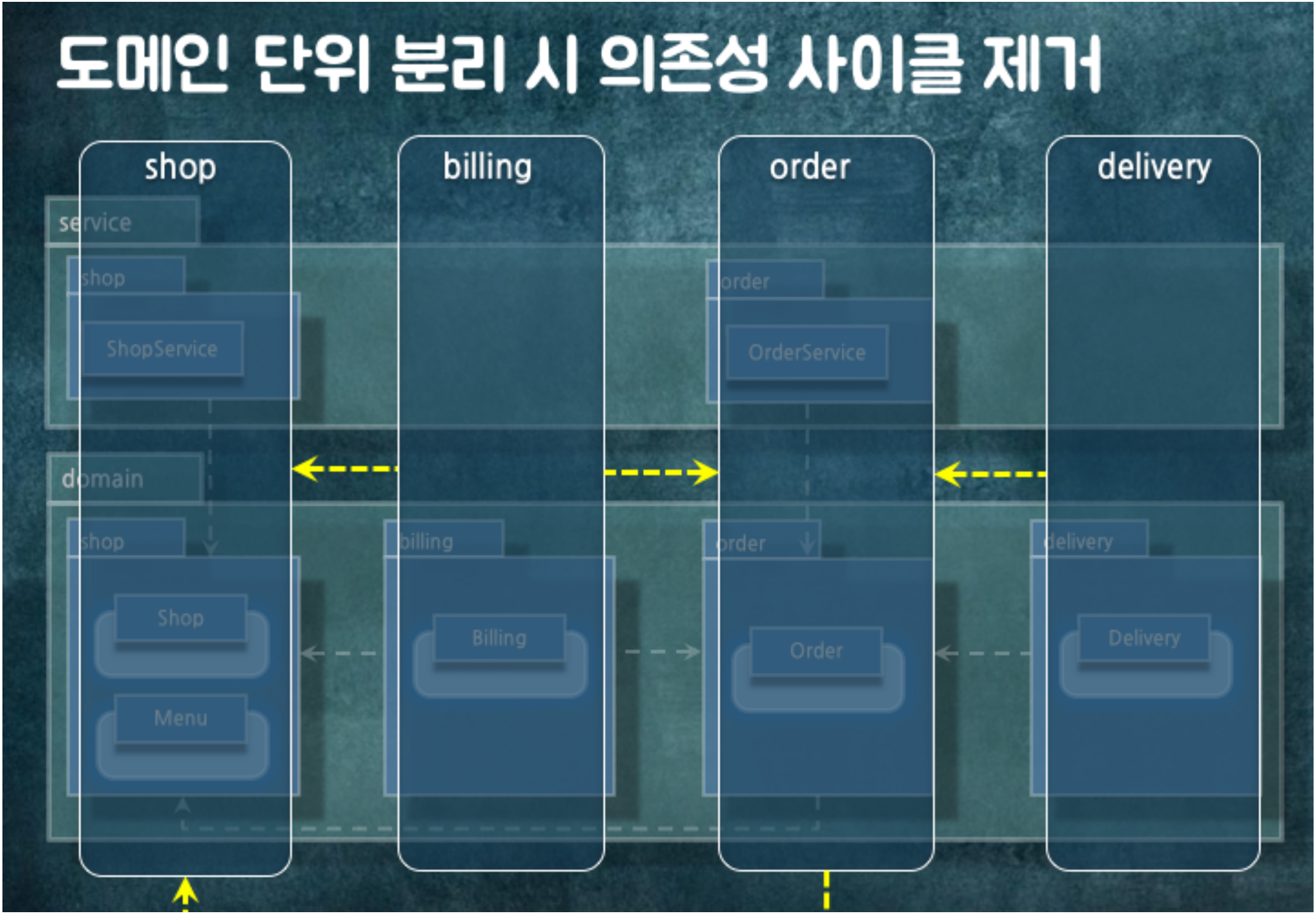
Domain Event 사용 전의 의존성



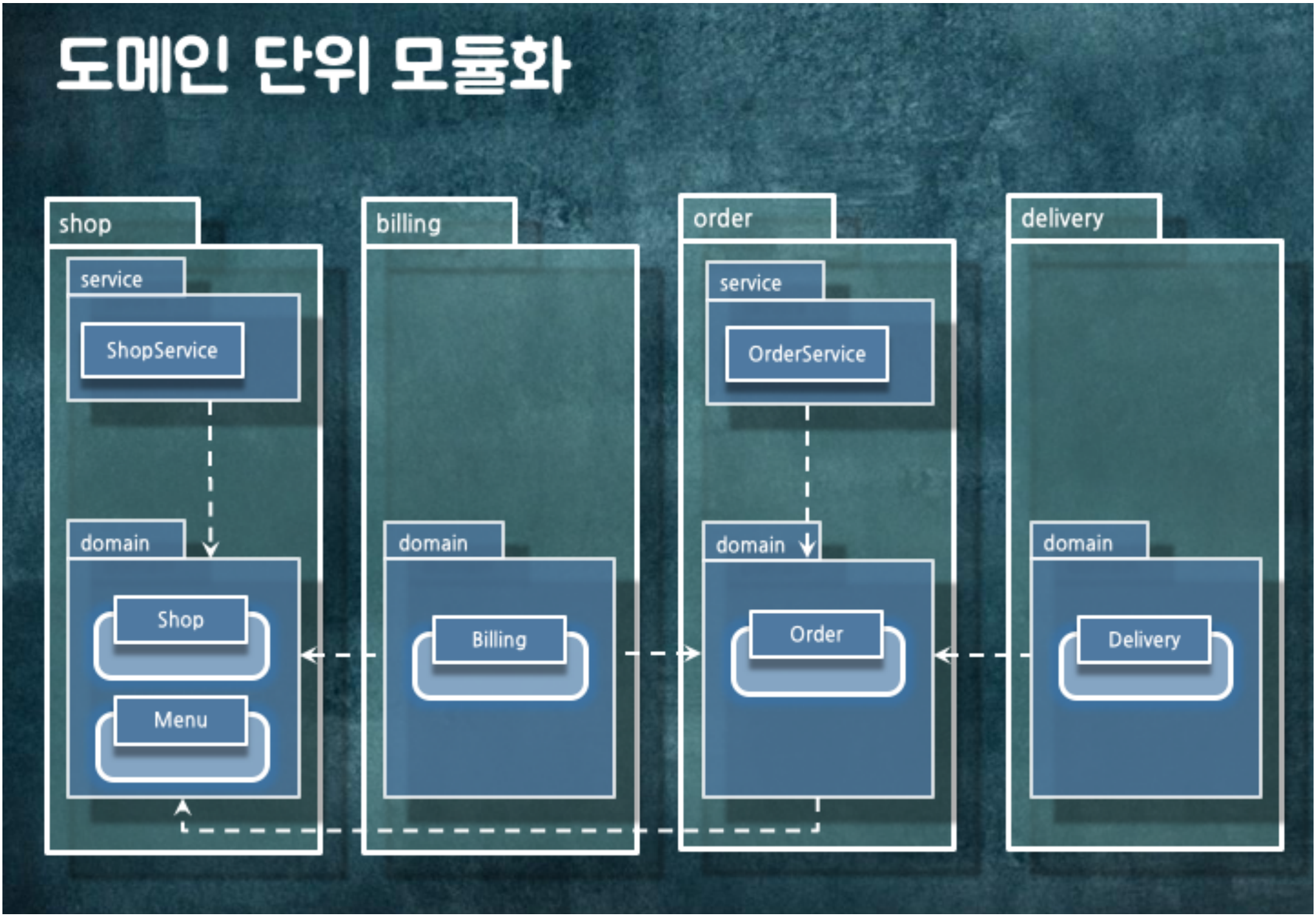

도메인 단위 분리 시 의존성 사이클 존재
shop
order
delivery
service
shop
ShopService
order
OrderService
domain
shop
Shop
Menu
order
Order
delivery
Delivery

Domain Event 사용 후의 의존성

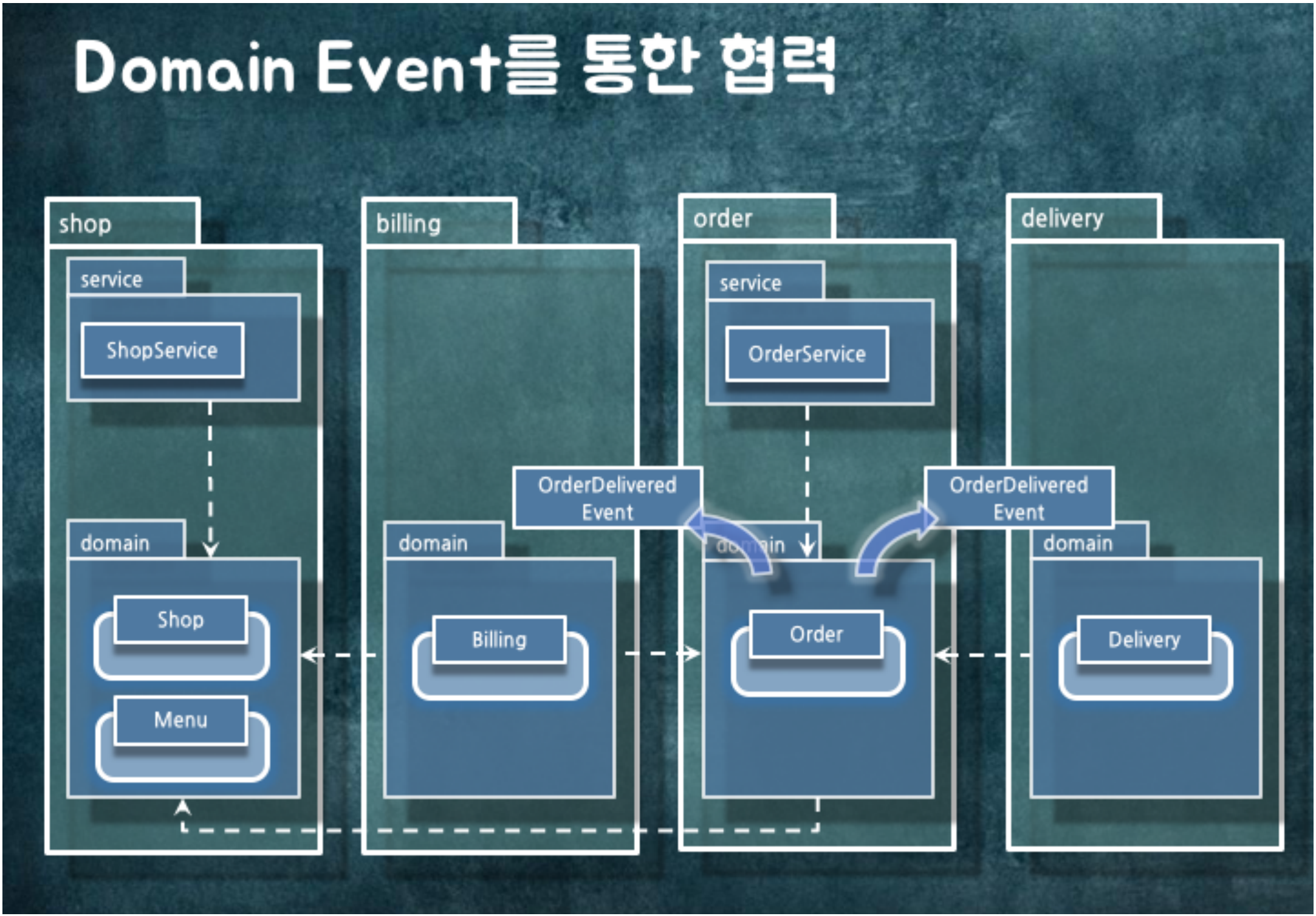



도메인 단위 분리 시 의존성 사이클 제거
shop
billing
order
delivery
service
shop
ShopService
order
OrderService
domain
shop
Shop
Menu
billing
Billing
order
Order
delivery
Delivery

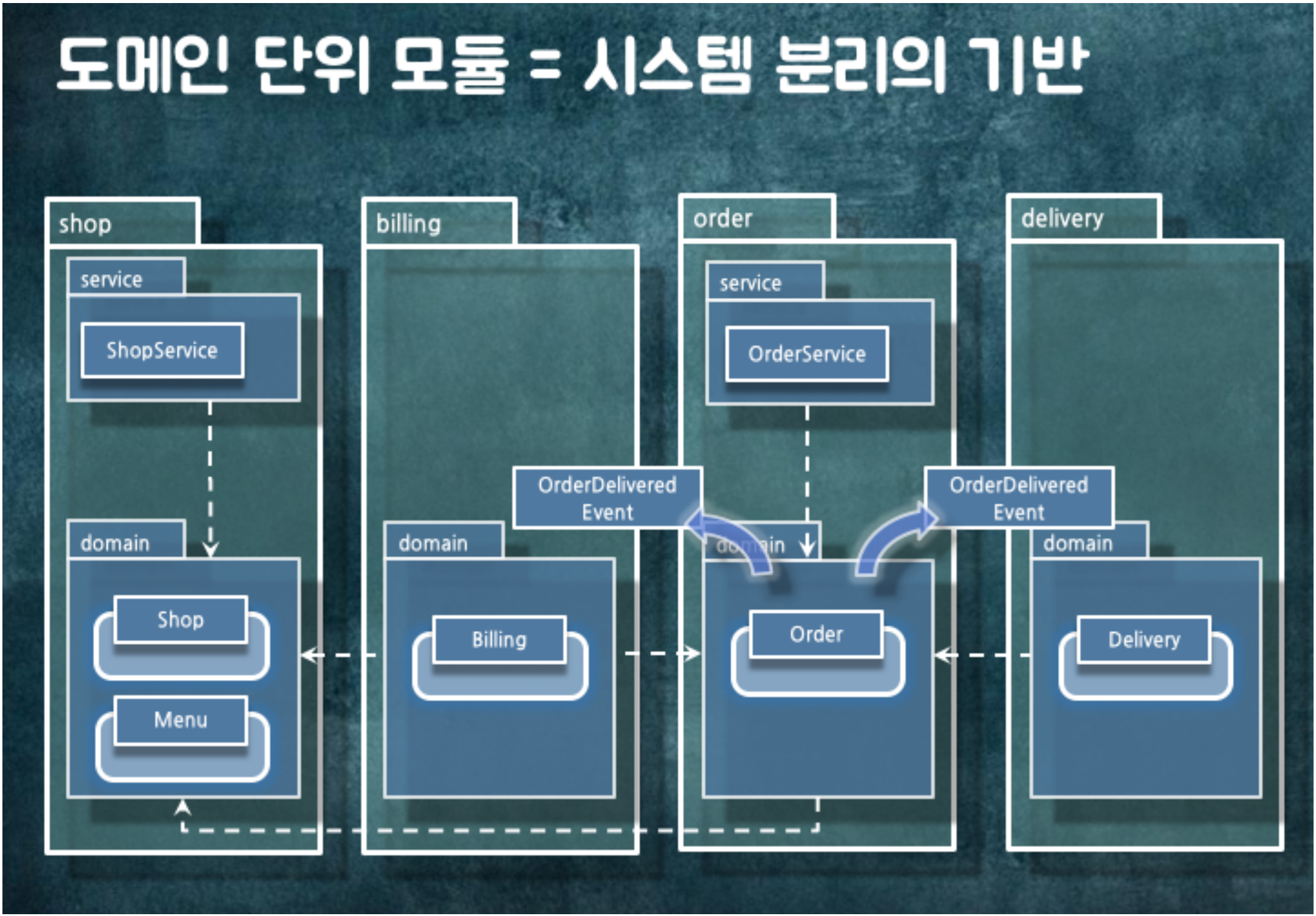
도메인 단위 모듈화


shop
service
ShopService
domain
Shop
Menu
billing
domain
Billing
order
service
OrderService
domain
Order
delivery
domain
Delivery

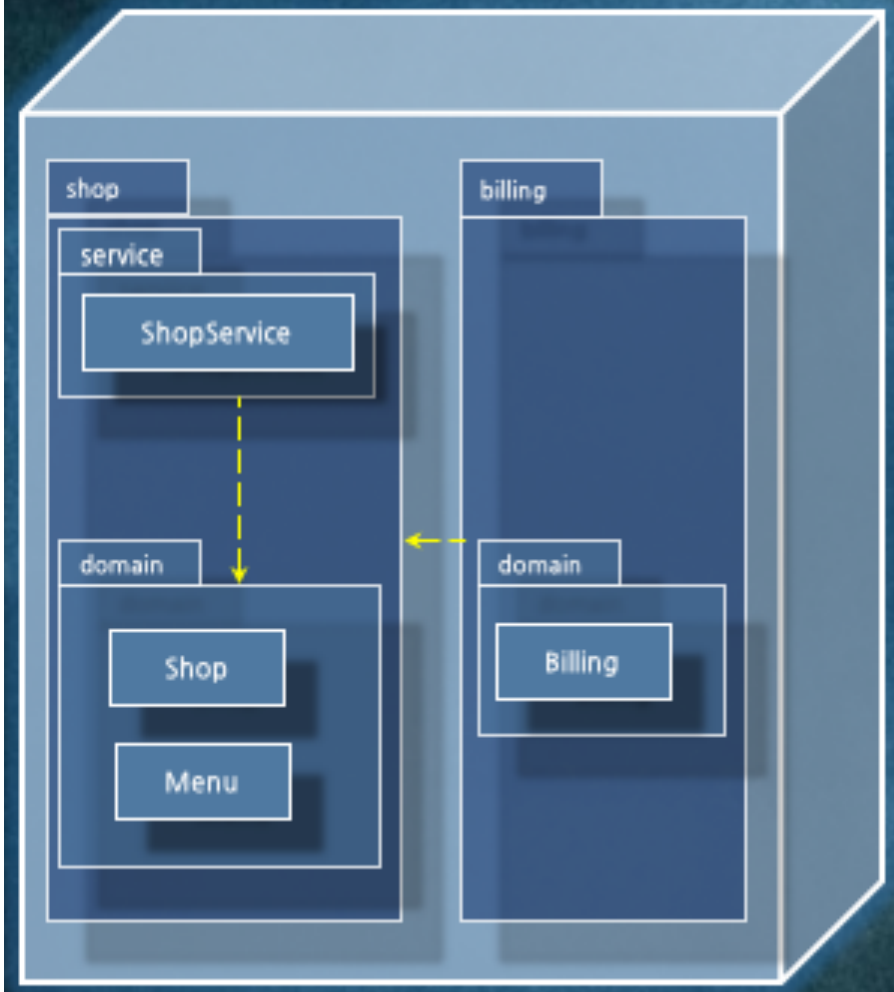
Domain Event를 통한 협력

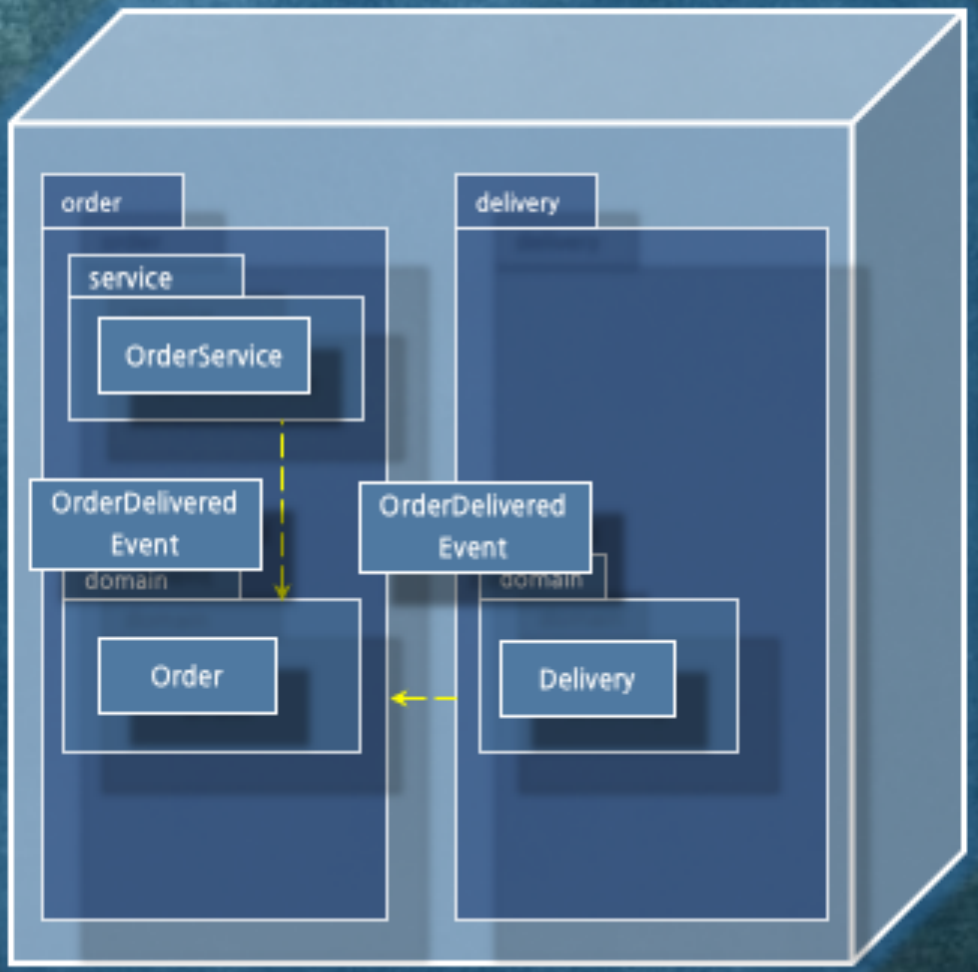


도메인 단위 모듈 = 시스템 분리의 기반

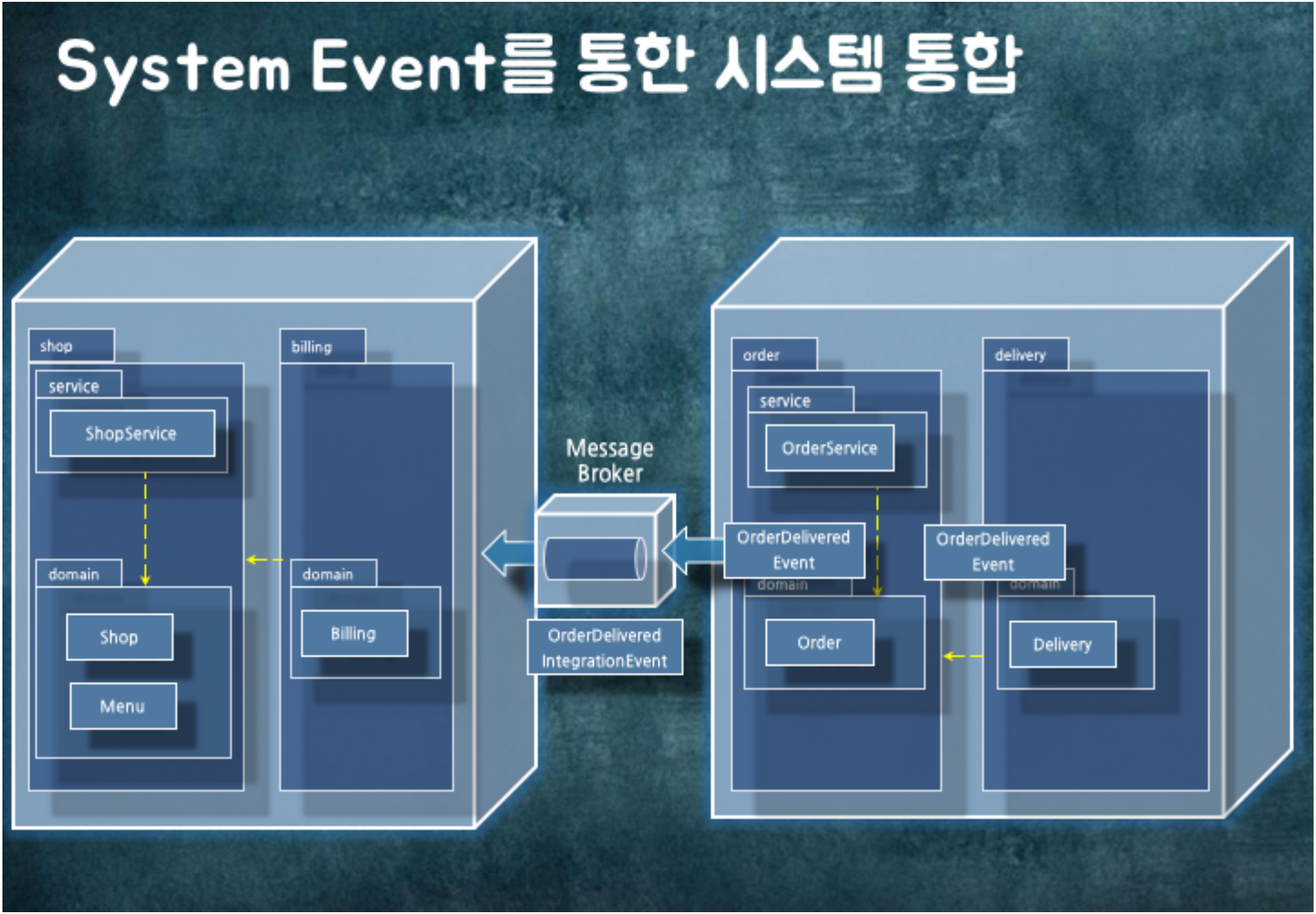


도메인 단위로 시스템 분리 가능





System Event를 통한 시스템 통합




의존성을 따라
시스템을 진화시켜라


끗