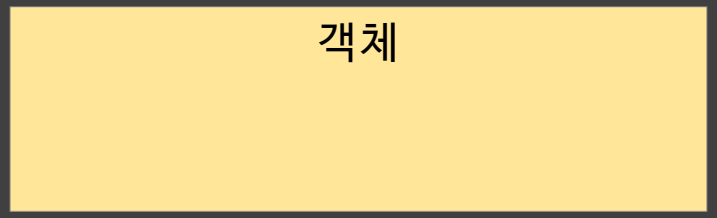
CODE  SPITZ

OBJECT

1 2 3 4 5 6

객체망과 객체간 통신

객체간 통신



객체간 통신

객체

내부상태는 **캡슐화**

외부소통은 메소드(**정보은닉**)

객체간 통신

객체
내부상태는 **캡슐화**
외부소통은 메소드(**정보은닉**)

객체
내부상태는 **캡슐화**
외부소통은 메소드(**정보은닉**)

객체간 통신

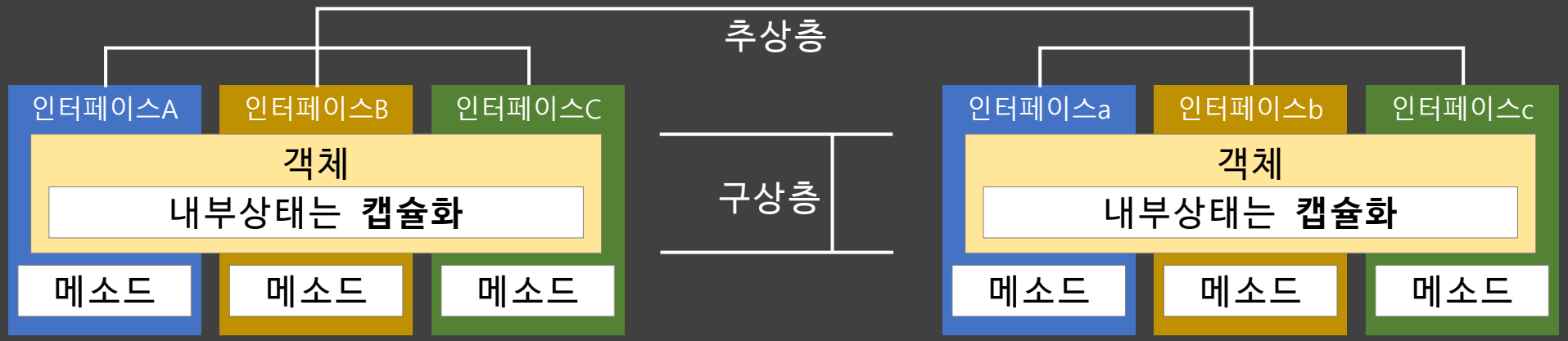
객체간 통신

객체간 통신

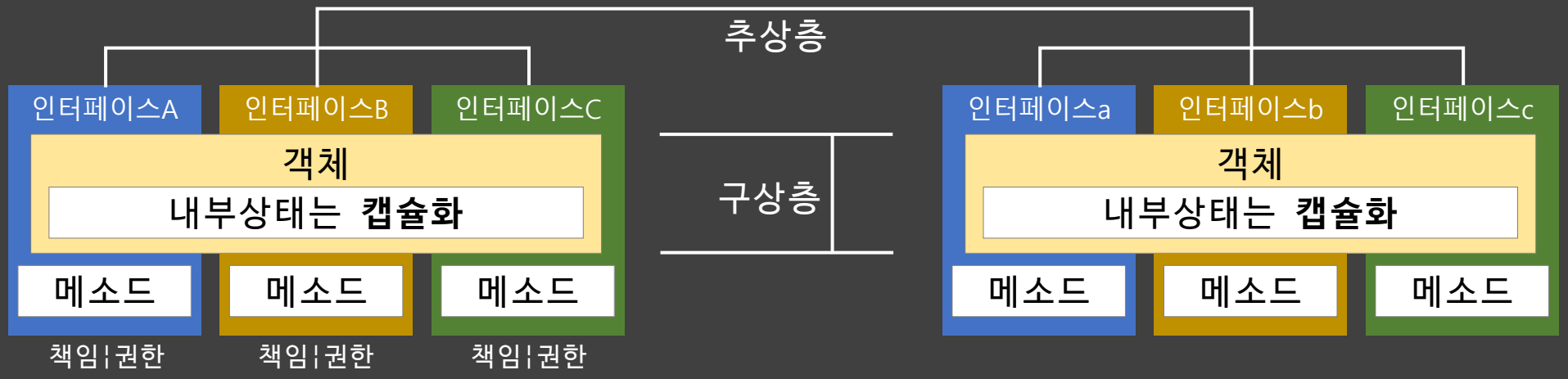

인터페이스A
인터페이스B
인터페이스C
객체
내부상태는 캡슐화
메소드
메소드
메소드


인터페이스a
인터페이스b
인터페이스c
객체
내부상태는 캡슐화
메소드
메소드
메소드

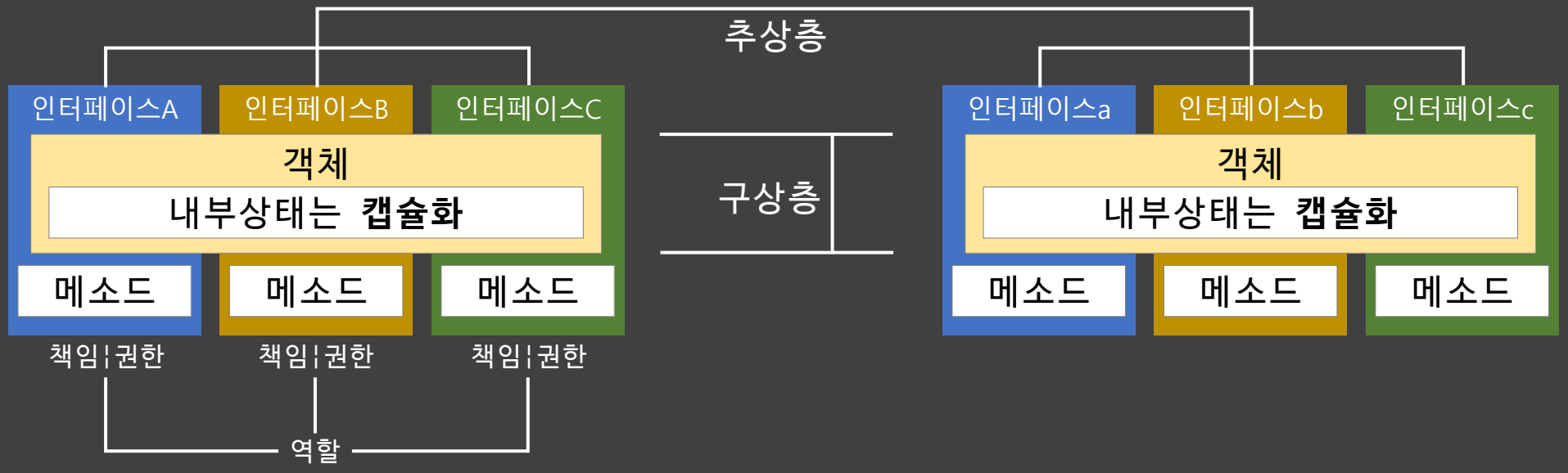
객체간 통신


추상층
인터페이스A
인터페이스B
인터페이스C
객체
내부상태는 캡슐화
메소드
메소드
메소드
구상층
인터페이스a
인터페이스b
인터페이스c
객체
내부상태는 캡슐화
메소드
메소드
메소드

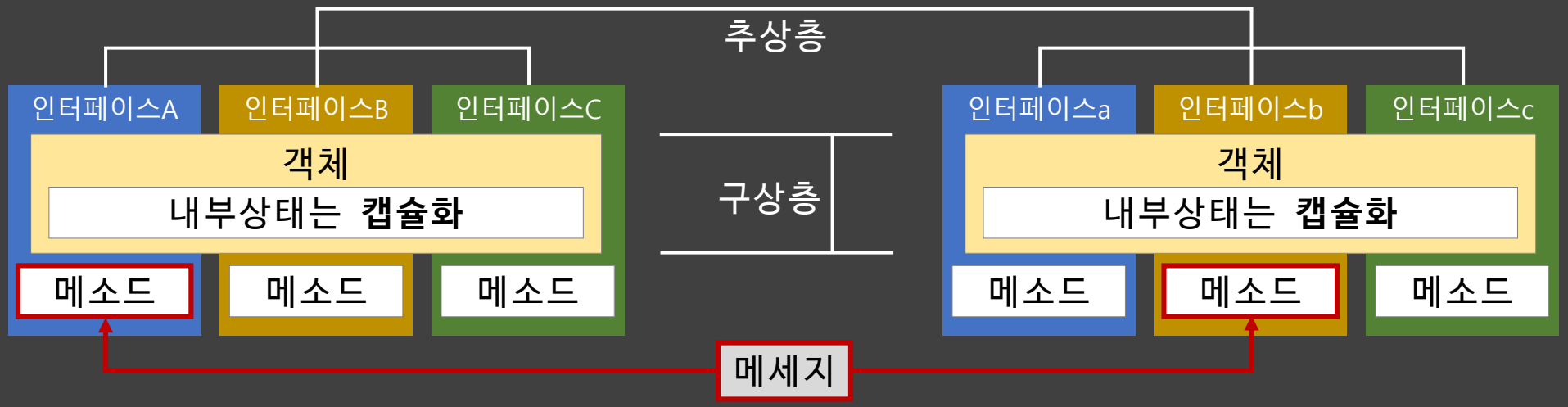
객체간 통신


추상층
인터페이스A
인터페이스B
인터페이스C
객체
내부상태는 캡슐화
메소드
메소드
메소드
책임¦권한
책임¦권한
책임¦권한
구상층
인터페이스a
인터페이스b
인터페이스c
객체
내부상태는 캡슐화
메소드
메소드
메소드

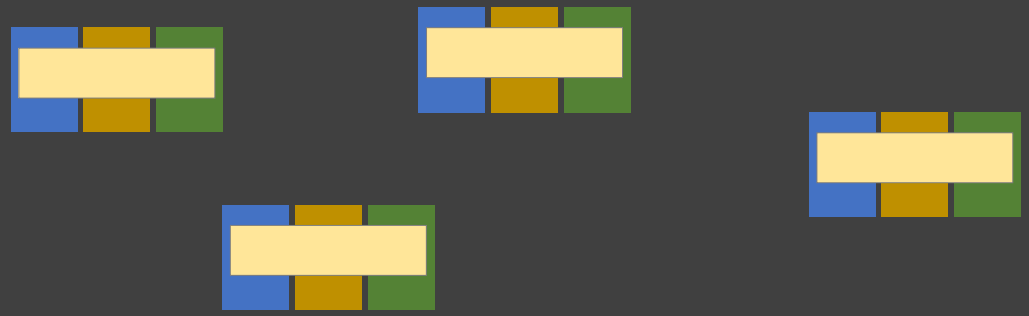
객체간 통신


추상층
인터페이스A
인터페이스B
인터페이스C
객체
내부상태는 캡슐화
메소드
메소드
메소드
책임¦권한
책임¦권한
책임¦권한
구상층
인터페이스a
인터페이스b
인터페이스c
객체
내부상태는 캡슐화
메소드
메소드
메소드
역할

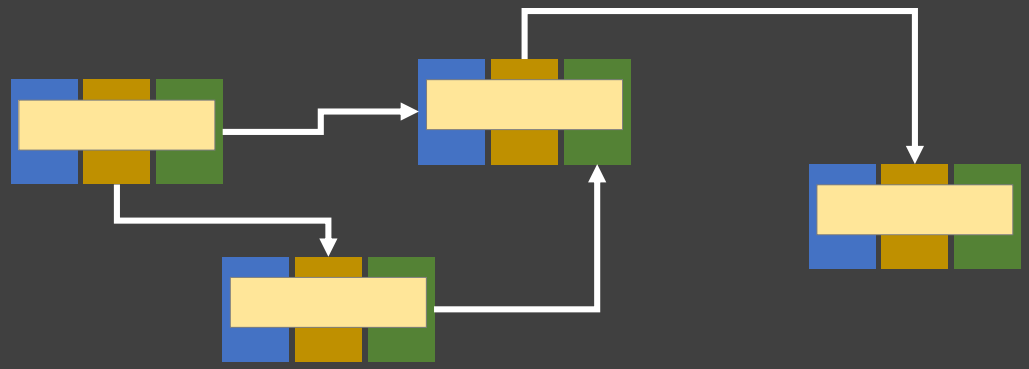
객체간 통신


추상층
인터페이스A
인터페이스B
인터페이스C
객체
내부상태는 캡슐화
메소드
메소드
메소드
구상층
인터페이스a
인터페이스b
인터페이스c
객체
내부상태는 캡슐화
메소드
메소드
메소드
메세지

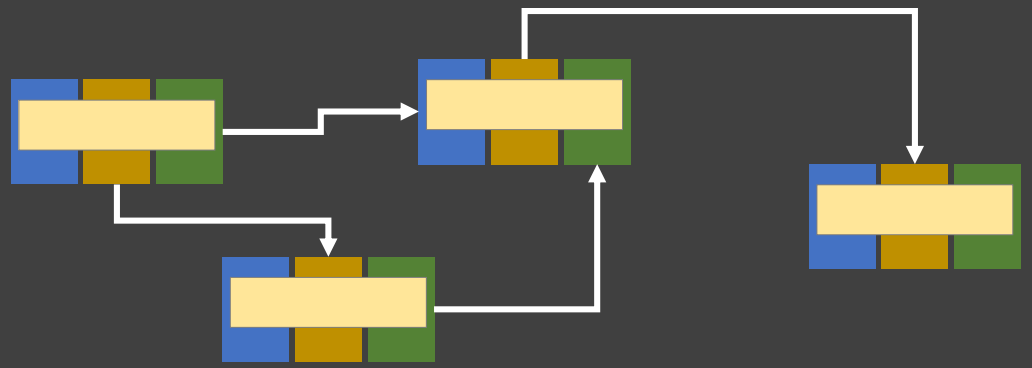
통신망의 구성

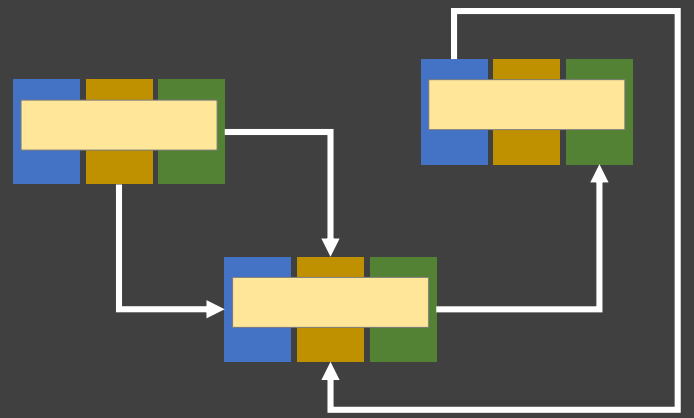


통신망의 구성



통신망의 구성

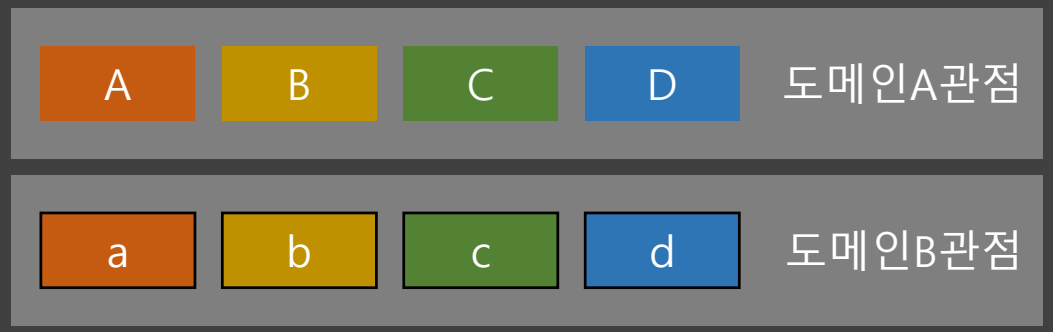




객체설계 난점

인터페이스의 그룹화

인터페이스의 그룹화



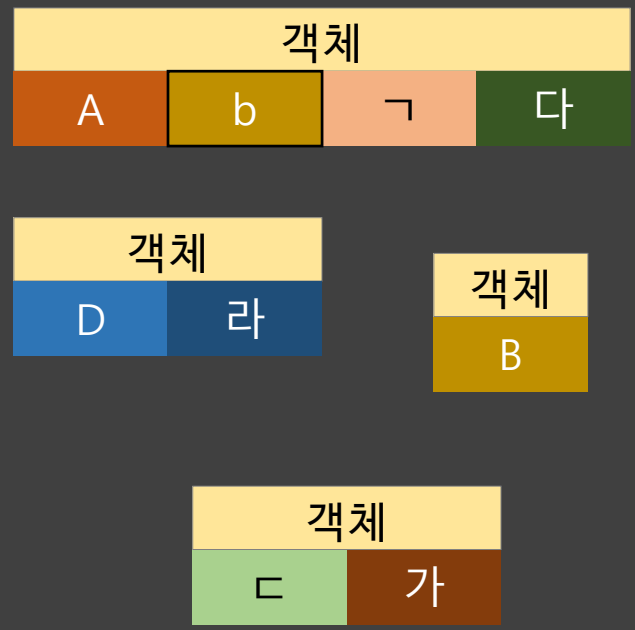
인터페이스의 그룹화

인터페이스의 그룹화

여러 관점을 수용하는 객체

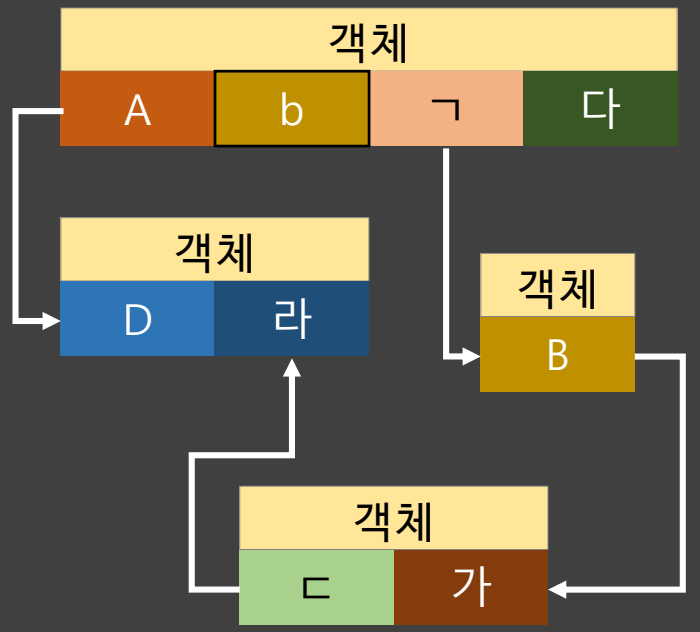
인터페이스의 그룹화

여러 관점을 수용하는 객체



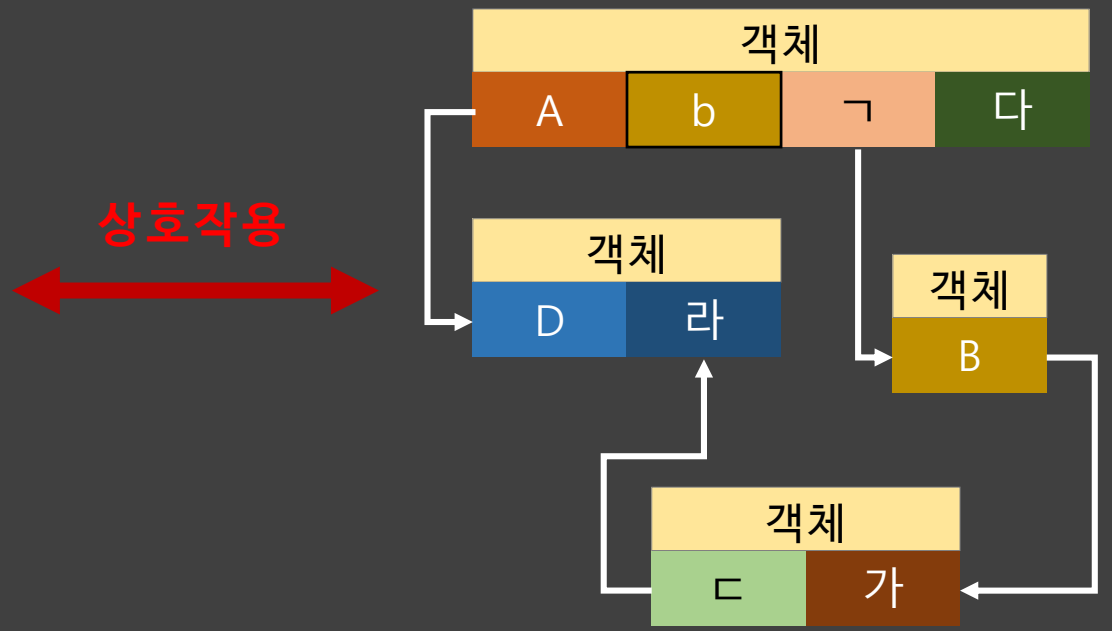
인터페이스의 그룹화

여러 관점을 수용하는 객체망



인터페이스의 그룹화

여러 관점을 수용하는 객체망



알려진 기본 설계요령

SOLID원칙

SRP Single Responsibility 단일책임 - 117

SOLID원칙

SRP Single Responsibility 단일책임 - 117

산탄총 수술
shotgun surgery

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

추상층의 정의가 너무 구체적이면 구상층의 구현에서 모순이 발생함.

추상층 - 생물
숨을 쉰다, 다리로 이동한다.

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

추상층의 정의가 너무 구체적이면 구상층의 구현에서 모순이 발생함.

추상층 - 생물
숨을 쉰다, 다리로 이동한다.

구상층

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

추상층의 정의가 너무 구체적이면 구상층의 구현에서 모순이 발생함.

추상층 - 생물
숨을 쉰다, 다리로 이동한다.

구상층
사람 ok!
타조 ok!

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

추상층의 정의가 너무 구체적이면 구상층의 구현에서 모순이 발생함.

추상층 - 생물
숨을 쉰다, 다리로 이동한다.

구상층
사람 ok!
타조 ok!
아메바 no!
독수리 no!
고래 no!

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

추상층의 정의가 너무 구체적이면 구상층의 구현에서 모순이 발생함.

추상층 - 생물
숨을 쉰다, ~~다리로 이동한다.~~

구상층
사람 ok!
타조 ok!
아메바 no!
독수리 no!
고래 no!

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전

추상층의 정의가 너무 구체적이면 구상층의 구현에서 모순이 발생함.

추상층 - 생물(숨을 쉰다), 다리이동(다리 로이동한다)

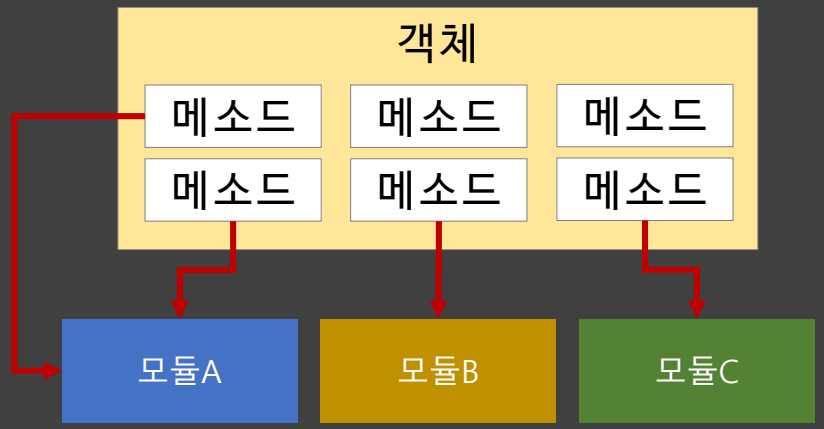
구상층
사람:생물,다리이동 ok!
타조:생물,다리이동 ok!
아메바:생물 ok!
독수리:생물 ok!
고래:생물 ok!

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리

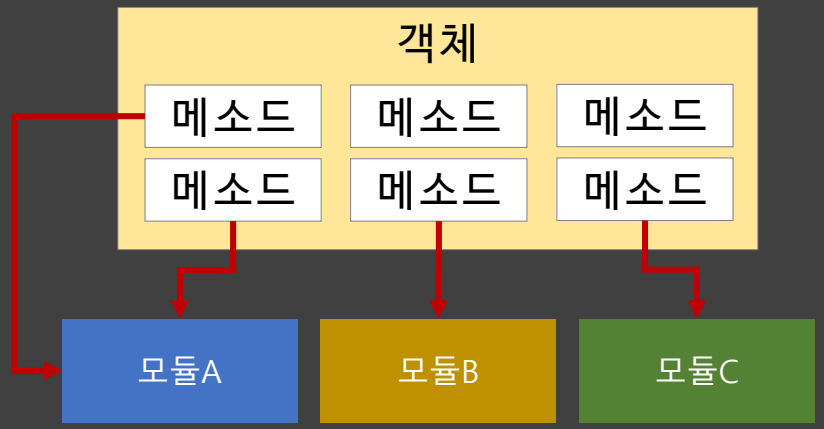
SOLID원칙

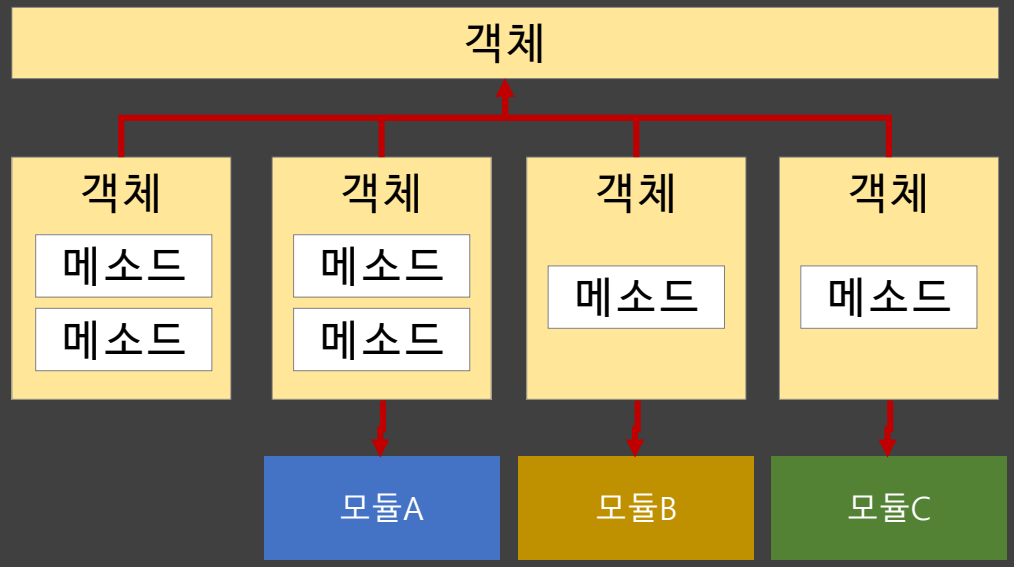
SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리



SOLID원칙

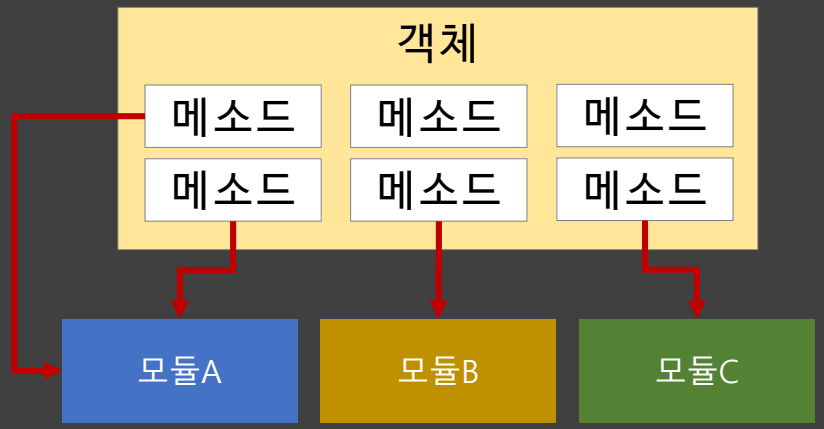
SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리

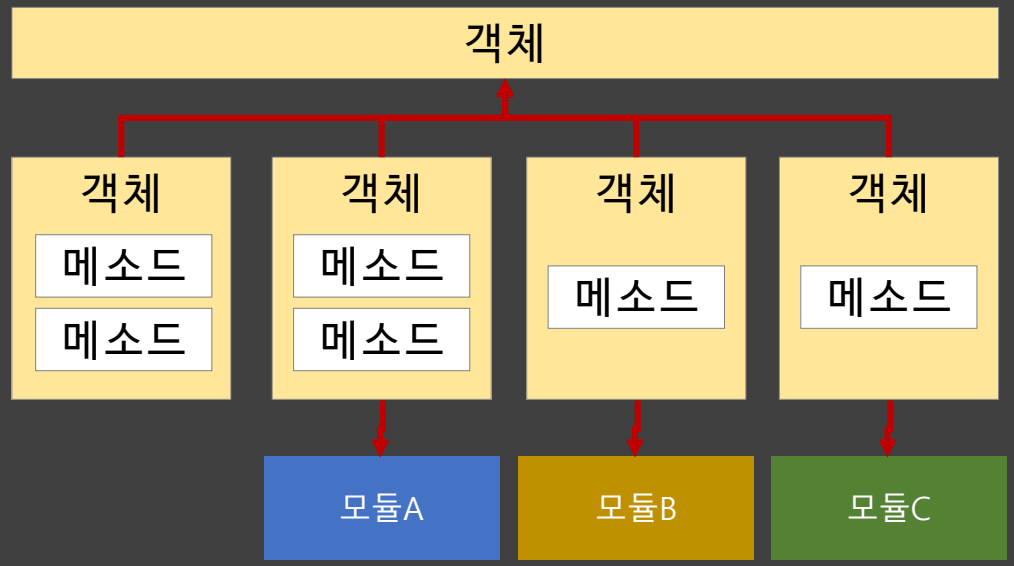




SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리





SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

고차원의 모듈은 저차원의 모듈에 의존하면 안된다.
이 두 모듈 모두 추상화된 것에 의존해야 한다.

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

고차원의 모듈은 저차원의 모듈에 의존하면 안된다.
이 두 모듈 모두 추상화된 것에 의존해야 한다.

추상화 된 것은 구체적인 것에 의존하면 안 된다.
구체적인 것이 추상화된 것에 의존해야 한다.

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

DI Dependency Injection 의존성주입
(IoC Inversion of Control 제어역전)

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

DI Dependency Injection 의존성주입
(IoC Inversion of Control 제어역전)

DRY Don't Repeat Yourself 중복방지

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

DI Dependency Injection 의존성주입
(IoC Inversion of Control 제어역전)

DRY Don't Repeat Yourself 중복방지

Hollywood Principle 의존성 부패방지

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

DI Dependency Injection 의존성주입
(IoC Inversion of Control 제어역전)

DRY Don't Repeat Yourself 중복방지

Hollywood Principle 의존성 부패방지

Law of demeter 최소 지식

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

DI Dependency Injection 의존성주입
(IoC Inversion of Control 제어역전)

DRY Don't Repeat Yourself 중복방지

Hollywood Principle 의존성 부패방지

Law of demeter 최소 지식

classA.methodA의 최대지식한계
- classA의 필드 객체
- methodA가 생성한 객체
- methodA의 인자로 넘어온 객체

SOLID원칙

SRP Single Responsibility 단일책임 - 117
OCP Open Closed 개방폐쇄
LSP Liskov Substitusion 업캐스팅 안전
ISP Interface Segregation 인터페이스분리
DIP Dependency Inversion 다운캐스팅금지

DI Dependency Injection 의존성주입
(IoC Inversion of Control 제어역전)

DRY Don't Repeat Yourself 중복방지

Hollywood Principle 의존성 부패방지

Law of demeter 최소 지식

classA.methodA의 최대지식한계
- classA의 필드 객체
- methodA가 생성한 객체
- methodA의 인자로 넘어온 객체

열차전복
train wreck

의존성부패방지와 최소지식의 모순

두 원칙을 지키면 해당 객체와 메세지를 주고 받는 것은 가능하나
내부를 들여다볼 방법이 없음

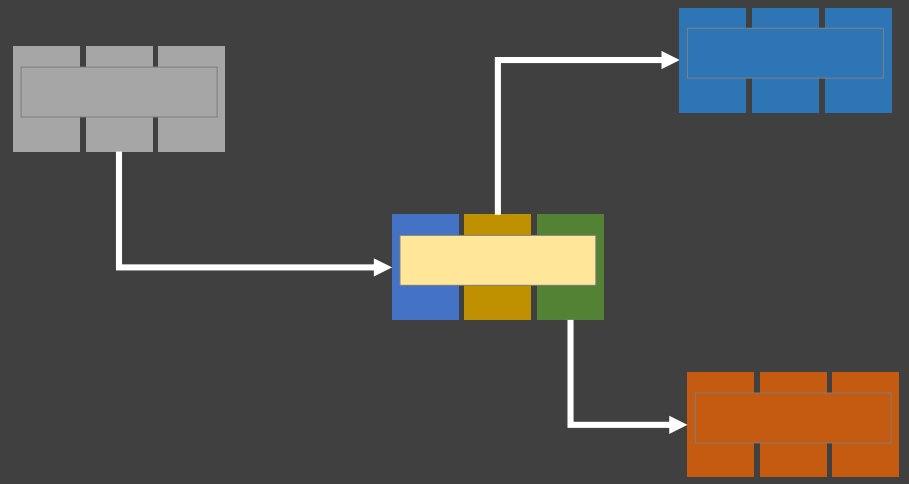
두 원칙을 지키면 해당 객체와 메세지를 주고 받는 것은 가능하나
내부를 들여다볼 방법이 없음

두 원칙을 지키면 해당 객체와 메세지를 주고 받는 것은 가능하나
내부를 들여다볼 방법이 없음

객체통신망 - 객체는 메세지를 주고 받는다.

두 원칙을 지키면 해당 객체와 메세지를 주고 받는 것은 가능하나
내부를 들여다볼 방법이 없음

객체통신망 - 객체는 메세지를 주고 받는다.

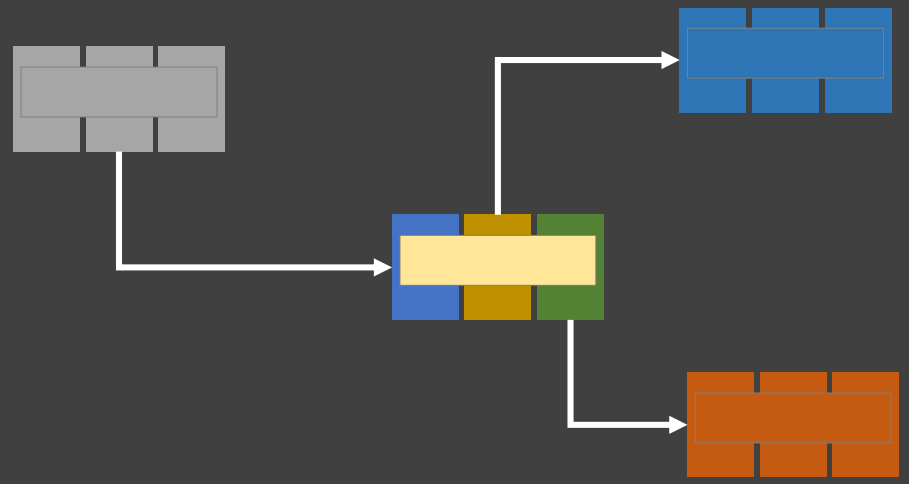


두 원칙을 지키면 해당 객체와 메세지를 주고 받는 것은 가능하나 내부를 들여다볼 방법이 없음

객체통신망 - 객체는 메세지를 주고 받는다.

결국 객체가 제대로 작동하는가를 테스트하려면...

1. 객체통신망에서 테스트할 객체에게 메세지를 보낸 뒤
2. 그 객체가 이웃 객체에게 메세지를 잘 보냈는지 확인
3. 3번을 위해 통신한 이웃 객체를 조사하면 된다.



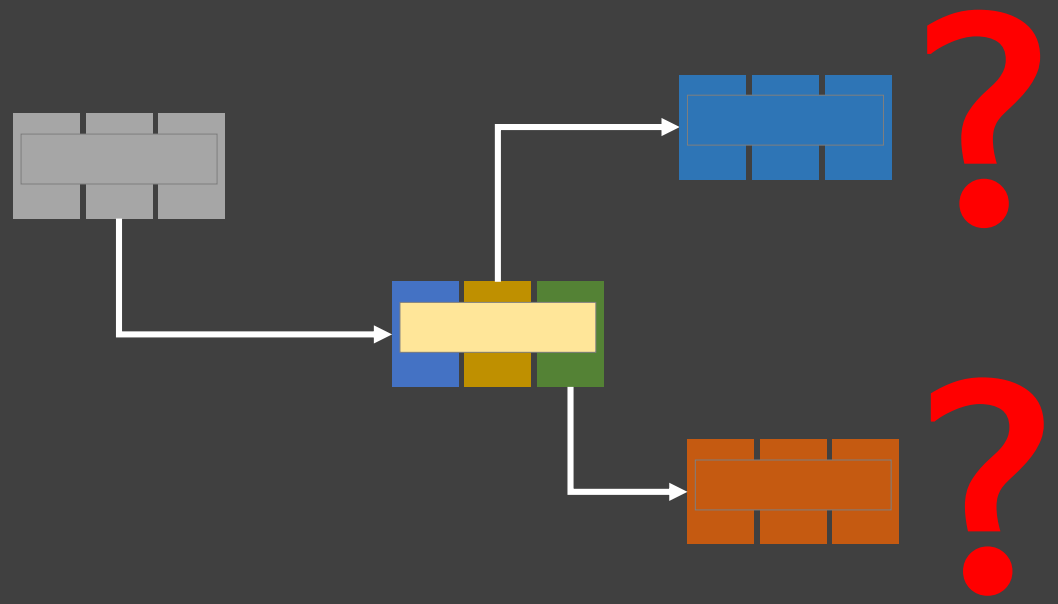
두 원칙을 지키면 해당 객체와 메세지를 주고 받는 것은 가능하나 내부를 들여다볼 방법이 없음

객체통신망 - 객체는 메세지를 주고 받는다.

결국 객체가 제대로 작동하는가를 테스트하려면...

1. 객체통신망에서 테스트할 객체에게 메세지를 보낸 뒤
2. 그 객체가 이웃 객체에게 메세지를 잘 보냈는지 확인
3. 3번을 위해 통신한 이웃 객체를 조사하면 된다.

이웃한 객체의 내부를 들여다볼 방법도 없는데?



Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

context

mockery = 모조객체 = context

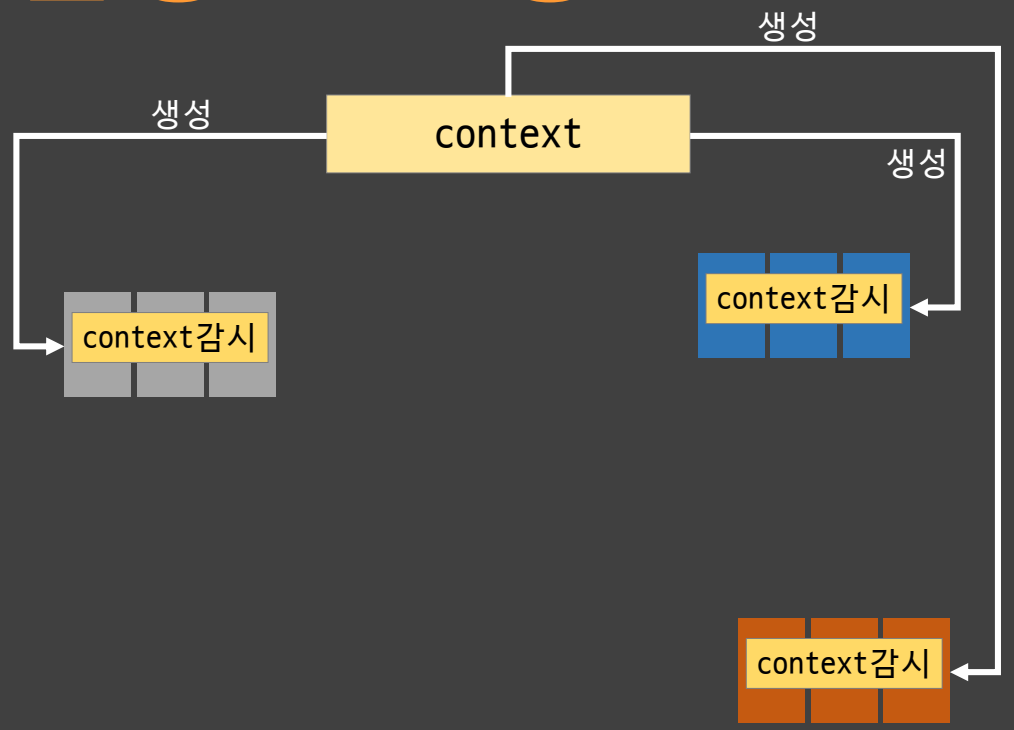
Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

mockery = 모조객체 = context

1. 필요한 목객체를 생성하고
2. 테스트할 객체를 둘러싼 객체망을 구성한 뒤(DI)
3. 트리거가 되는 메세지를 일으켜
4. 각 목객체의 상태를 확인한다.



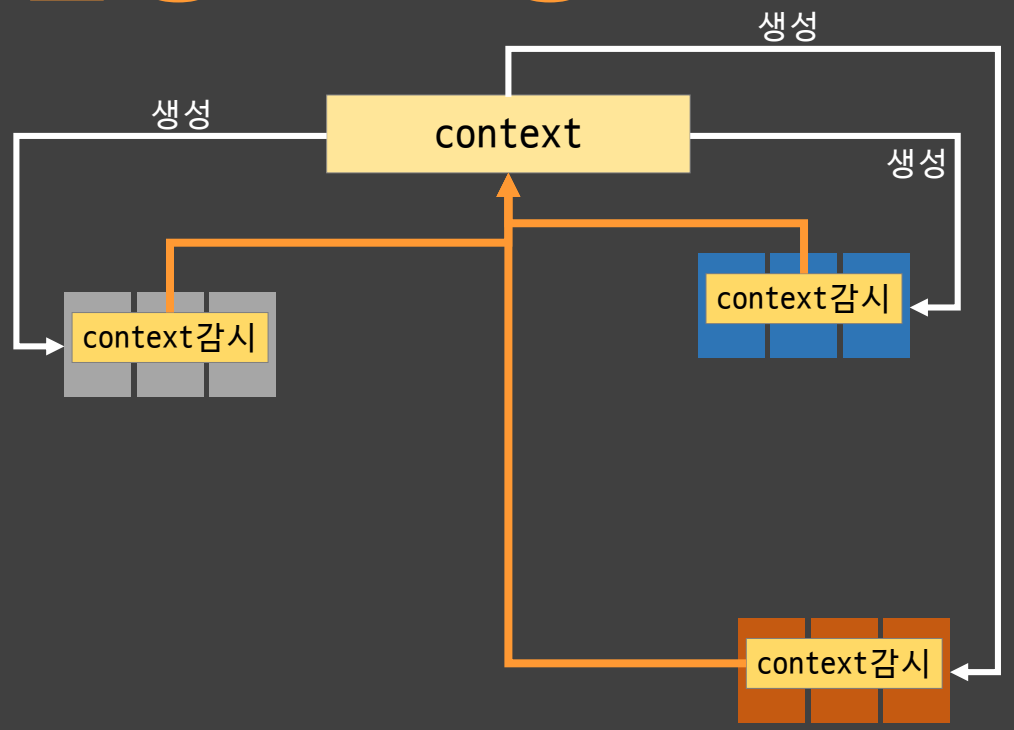
Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

mockery = 모조객체 = context

1. 필요한 목객체를 생성하고
2. 테스트할 객체를 둘러싼 객체망을 구성한 뒤(DI)
3. 트리거가 되는 메세지를 일으켜
4. 각 목객체의 상태를 확인한다.



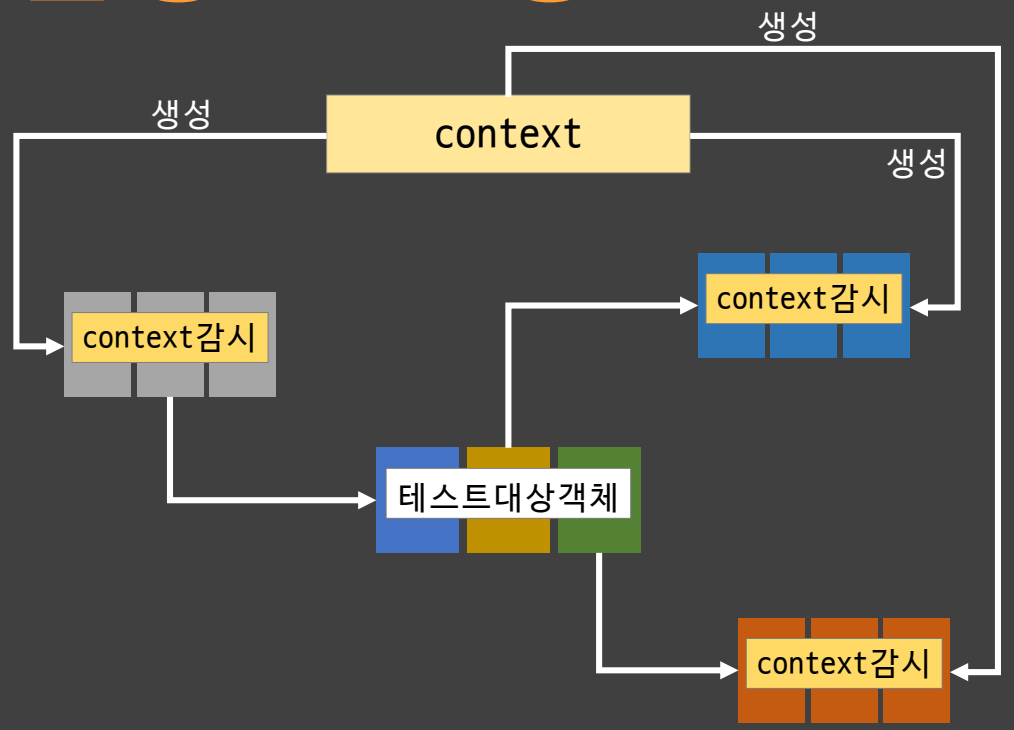
Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

mockery = 모조객체 = context

1. 필요한 목객체를 생성하고
2. 테스트할 객체를 둘러싼 객체망을 구성한 뒤(DI)
3. 트리거가 되는 메세지를 일으켜
4. 각 목객체의 상태를 확인한다.



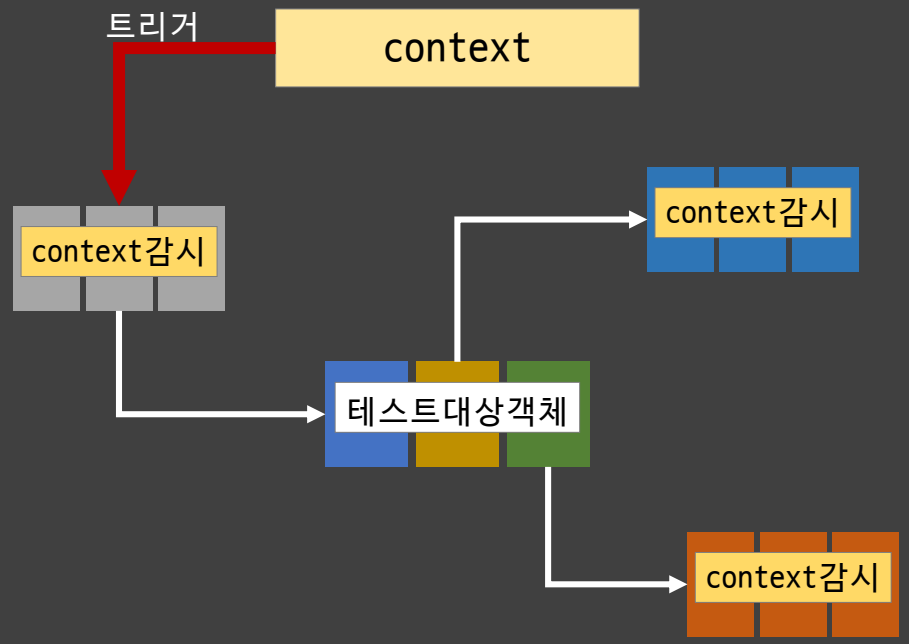
Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

mockery = 모조객체 = context

1. 필요한 목객체를 생성하고
2. 테스트할 객체를 둘러싼 객체망을 구성한 뒤(DI)
3. 트리거가 되는 메세지를 일으켜
4. 각 목객체의 상태를 확인한다.



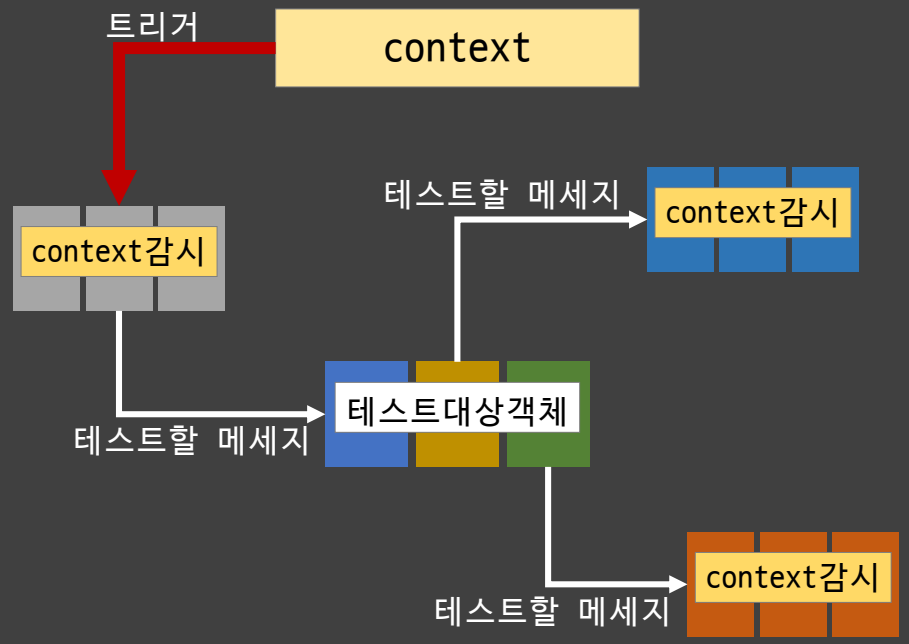
Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

mockery = 모조객체 = context

1. 필요한 목객체를 생성하고
2. 테스트할 객체를 둘러싼 객체망을 구성한 뒤(DI)
3. 트리거가 되는 메세지를 일으켜
4. 각 목객체의 상태를 확인한다.



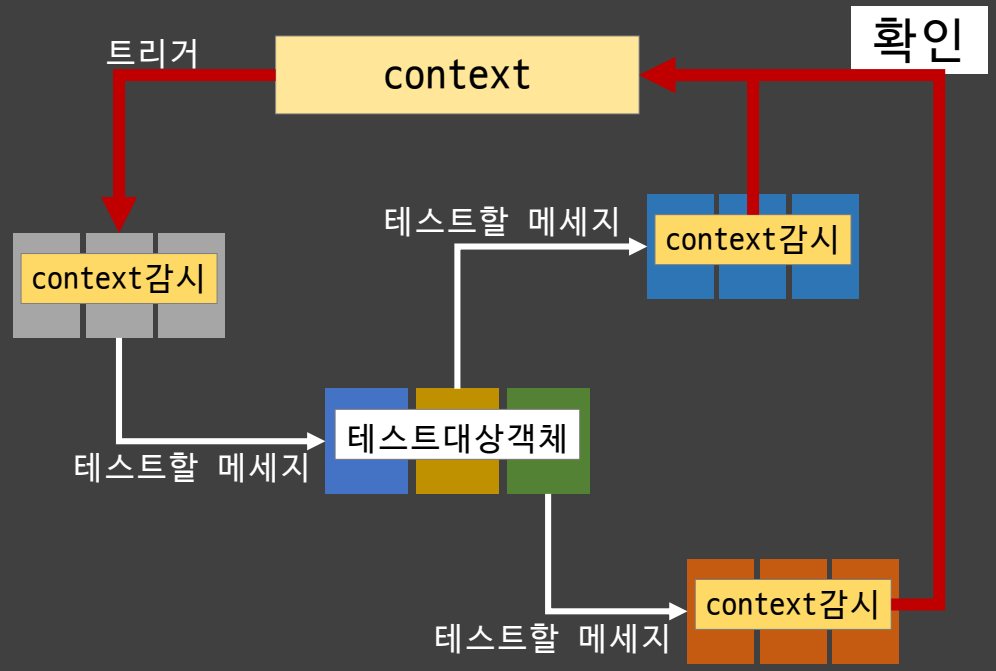
Mock객체를 활용한 검증

mockery(모조객체) 와 mock(목객체)

테스트관리객체 테스트용 모의객체

mockery = 모조객체 = context

1. 필요한 목객체를 생성하고
2. 테스트할 객체를 둘러싼 객체망을 구성한 뒤(DI)
3. 트리거가 되는 메세지를 일으켜
4. 각 목객체의 상태를 확인한다.



GRASP

Information Expert

정보 담당자	Information Expert
소유권한	Creator
컨트롤러	Controller
낮은 연결	Low Coupling
높은 응집도	High Cohesion
간접 참조	Indirection
다형성	Polymorphism
순수 조립	Pure Fabrication
변화 보호	Protected Variations

Information Expert

해당 정보를 갖고 있는 객체에게 책임을 할당하라.
객체의 본질과 데이터 은닉을 지킬 수 있는 패턴

Information Expert

해당 정보를 갖고 있는 객체에게 책임을 할당하라.
객체의 본질과 데이터 은닉을 지킬 수 있는 패턴

Creator

객체 시스템의 이질적인 부분인 생성 시에도 정보전문가 패턴을 따르자.
어떤 객체가 대상을 포함하거나, 이용하거나, 부분으로 삼거나, 잘 알고 있다면 그 대상을 생성하게 시키자.

Controller

미디에이터 패턴의 설계판 확장으로 서브시스템으로 묶을 수 있다면 컨트롤러를 도입하자

Controller

미디에이터 패턴의 설계판 확장으로 서브시스템으로 묶을 수 있다면 컨트롤러를 도입하자

Low Coupling & High Cohesion

결합도를 낮추고 응집도를 높이는 패턴은 다른 양상으로 나타남. 결합도를 낮추려면 아는 객체 수를 줄여야 함. 하지만 더 중요한 것은 단방향 의존성임. 이에 비해 응집도를 높이려면 객체를 도출할 때부터 변화율을 고려해야 함.

Protected Variations

추상적인 수준에서 책임을 정의하여 다양한 구상가능성으로부터 사용할 모듈을 보호하라.

Protected Variations

추상적인 수준에서 책임을 정의하여 다양한 구상가능성으로부터 사용할 모듈을 보호하라.

Polymorphism

전략패턴처럼 분기가 예상되는 책임이라면 다형성을 이용하라.

Pure Fabrication

공통된 기능이나 순수 기능적인 객체는 따로 모아서 작성한다.

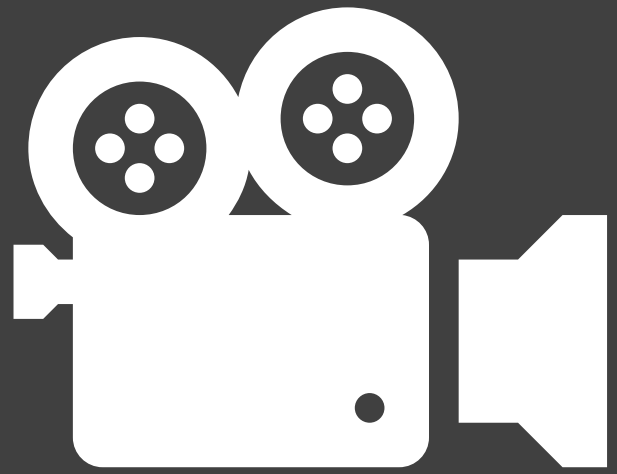
Pure Fabrication

공통된 기능이나 순수 기능적인 객체는 따로 모아서 작성한다.

Indirection

직접 참조관계를 피하고 중계 객체를 이용하면 개별 객체의 충격을 중계 객체에서 흡수할 수 있다.

Theater



```java
public class Movie {
    private final String title;
    private final Duration runningTime;
    private final Money fee;
    private final DiscountPolicy policy;
    public Movie(String title, Duration runningTime, Money fee, DiscountPolicy policy){
        this.title = title;
        this.runningTime = runningTime;
        this.fee = fee;
        this.policy = policy;
    }
    Money calculateFee(Screening screening, int count){
        return policy.calculateFee(screening, count, fee);
    }
}
```

```java
abstract class DiscountPolicy {
    private Set<DiscountCondition> conditions = new HashSet<>();
    public void addCondition(DiscountCondition condition){
        conditions.add(condition);
    }
    public void copyCondition(DiscountPolicy policy){
        policy.conditions.addAll(conditions);
    }
    public Money calculateFee(Screening screening, int count, Money fee){
        for(DiscountCondition condition:conditions){
            if(condition.isSatisfiedBy(screening, count)) return calculateFee(fee);
        }
        return fee;
    }
    protected abstract Money calculateFee(Money fee);
}
```

```java
public class AmountPolicy extends DiscountPolicy {
    private final Money amount;
    public AmountPolicy(Money amount){
        this.amount = amount;
    }
    @Override
    public Money calculateFee(Money fee) {
        return fee.minus(amount);
    }
}
```

```java
public class AmountPolicy extends DiscountPolicy {
    private final Money amount;
    public AmountPolicy(Money amount){
        this.amount = amount;
    }
    @Override
    public Money calculateFee(Money fee) {
        return fee.minus(amount);
    }
}
public class PercentPolicy extends DiscountPolicy {
    private final Double percent;
    public PercentPolicy(Double percent){
        this.percent = percent;
    }
    @Override
    public final Money calculateFee(Money fee) {
        return fee.minus(fee.multi(percent));
    }
}
```

```java
public class NosalePolicy extends DiscountPolicy {
    @Override
    public final Money calculateFee(Money fee) {
        return fee;
    }
}
```

```java
interface DiscountCondition{
    boolean isSatisfiedBy(Screening screening, int audienceCount);
}
```

```java
interface DiscountCondition{
    boolean isSatisfiedBy(Screening screening, int audienceCount);
}

public class PeriodCondition implements DiscountCondition {
    private final LocalDateTime whenScreened;
    public PeriodCondition(LocalDateTime whenScreened){
        this.whenScreened = whenScreened;
    }
    @Override
    public boolean isSatisfiedBy(Screening screening, int audienceCount) {
        return screening.whenScreened.equals(whenScreened);
    }
}
```

```java
interface DiscountCondition{
    boolean isSatisfiedBy(Screening screening, int audienceCount);
}

public class SequenceCondition implements DiscountCondition {
    private final int sequence;
    public SequenceCondition(int sequence){
        this.sequence = sequence;
    }
    @Override
    public boolean isSatisfiedBy(Screening screening, int audienceCount) {
        return screening.sequence == sequence;
    }
}
```

```java
public class TicketSeller {
    private TicketOffice ticketOffice;
    public void setTicketOffice(TicketOffice ticketOffice){
        this.ticketOffice = ticketOffice;
    }
    Reservation reserve(Customer customer, Theater theater, Movie movie, Screening screening, int count){
        Reservation reservation = Reservation.NONE;
        Money price = movie.calculateFee(screening, count);
        if(customer.hasAmount(price)){
            reservation = ticketOffice.reserve(theater, movie, screening, count);
            if(reservation != Reservation.NONE) customer.minusAmount(price);
        }
        return reservation;
    }
}
```

```java
public class TicketSeller {
    private TicketOffice ticketOffice;
    public void setTicketOffice(TicketOffice ticketOffice){
        this.ticketOffice = ticketOffice;
    }
    Reservation reserve(Customer customer, Theater theater, Movie movie, Screening screening, int count){
        Reservation reservation = Reservation.NONE;
        Money price = movie.calculateFee(screening, count);
        if(customer.hasAmount(price)){
            reservation = ticketOffice.reserve(theater, movie, screening, count);
            if(reservation != Reservation.NONE) customer.minusAmount(price);
        }
        return reservation;
    }
}
```

```java
public class TicketSeller {
    private TicketOffice ticketOffice;
    public void setTicketOffice(TicketOffice ticketOffice){
        this.ticketOffice = ticketOffice;
    }
    Reservation reserve(Customer customer, Theater theater, Movie movie, Screening screening, int count){
        Reservation reservation = Reservation.NONE;
        Money price = ticketOffice.calculateFee(movie, screening, count);
        if(customer.hasAmount(price)){
            reservation = ticketOffice.reserve(theater, movie, screening, count);
            if(reservation != Reservation.NONE) customer.minusAmount(price);
        }
        return reservation;
    }
}
```

```java
public class TicketOffice {
    private Money amount;
    private Map<Theater, Double> commissionRate = new HashMap<>();
    public TicketOffice(Money amount){this.amount = amount;}
    boolean contract(Theater theater, Double rate){}
    boolean cancel(Theater theater){}
    Money calculateFee(Movie movie, Screening screening, int count) {
        return movie.calculateFee(screening, count).multi((double)count);
    }
    Reservation reserve(Theater theater, Movie movie, Screening screening, int count){
        if(!commissionRate.containsKey(theater) || ...) return Reservation.NONE;
        Reservation reservation = theater.reserve(movie, screening, count);
        if(reservation != Reservation.NONE){
            Money sales = movie.calculateFee(screening, count).multi((double)count);
            Money commission = sales.multi(commissionRate.get(theater));
            amount = amount.plus(commission);
            theater.plusAmount(sales.minus(commission));
        }
        return reservation;
    }
}
```

```java
public class TicketOffice {
    private Money amount;
    private Map<Theater, Double> commissionRate = new HashMap<>();
    public TicketOffice(Money amount){this.amount = amount;}
    boolean contract(Theater theater, Double rate){}
    boolean cancel(Theater theater){}
    Money calculateFee(Movie movie, Screening screening, int count) {
        return movie.calculateFee(screening, count).multi((double)count);
    }
    Reservation reserve(Theater theater, Movie movie, Screening screening, int count){
        if(!commissionRate.containsKey(theater) || ...) return Reservation.NONE;
        Reservation reservation = theater.reserve(movie, screening, count);
        if(reservation != Reservation.NONE){
            Money sales = movie.calculateFee(screening, count).multi((double)count);
            Money commission = sales.multi(commissionRate.get(theater));
            amount = amount.plus(commission);
            theater.plusAmount(sales.minus(commission));
        }
        return reservation;
    }
}
```

```java
public class TicketOffice {
    private Money amount;
    private Map<Theater, Double> commissionRate = new HashMap<>();
    public TicketOffice(Money amount){this.amount = amount;}
    boolean contract(Theater theater, Double rate){}
    boolean cancel(Theater theater){}
    Money calculateFee(Movie movie, Screening screening, int count) {
        return movie.calculateFee(screening, count).multi((double)count);
    }
    Reservation reserve(Theater theater, Movie movie, Screening screening, int count){
        if(!commissionRate.containsKey(theater) || ...) return Reservation.NONE;
        Reservation reservation = theater.reserve(movie, screening, count);
        if(reservation != Reservation.NONE){
            Money sales = calculateFee(movie, screening, count);
            Money commission = sales.multi(commissionRate.get(theater));
            amount = amount.plus(commission);
            theater.plusAmount(sales.minus(commission));
        }
        return reservation;
    }
}
```