CODE  SPITZ

83

OBJECT

1 2 3 4 5 6

Type

Types

Role : 형을 통해 역할을 묘사함

Responsibility : 형을 통해 로직을 표현함

Message : 형을 통해 메세지를 공유함

Protocol : 객체 간 계약을 형을 통해 공유함

Types

Role : 형을 통해 역할을 묘사함

Responsibility : 형을 통해 로직을 표현함

Message : 형을 통해 메세지를 공유함

Protocol : 객체 간 계약을 형을 통해 공유함

Types

Role : 형을 통해 역할을 묘사함

Responsibility : 형을 통해 로직을 표현함

Message : 형을 통해 메세지를 공유함

Protocol : 객체 간 계약을 형을 통해 공유함

Types

Role : 형을 통해 역할을 묘사함

Responsibility : 형을 통해 로직을 표현함

Message : 형을 통해 메세지를 공유함

Protocol : 객체 간 계약을 형을 통해 공유함

static, enum, class

supported types

static : 단 한 개의 인스턴스가 존재(동시성 문제를 해결해야 함)

enum : 제한된 수의 인스턴스가 존재(제네릭에 사용불가 없음)

class : 무제한의 인스턴스가 존재

supported types

static : 단 한 개의 인스턴스가 존재(동시성 문제를 해결해야 함)

enum : 제한된 수의 인스턴스가 존재(제네릭에 사용불가 없음)

class : 무제한의 인스턴스가 존재

supported types

static : 단 한 개의 인스턴스가 존재(동시성 문제를 해결해야 함)

enum : 제한된 수의 인스턴스가 존재(제네릭에 사용불가 없음)

class : 무제한의 인스턴스가 존재

Condition

Condition

1. 조건 분기는 결코 제거할 수 없다.

2. 조건 분기에 대한 전략은 두 가지 뿐이다.
   - 내부에서 응집성있게 모아두는 방식
     장점 : 모든 경우의 수를 한 곳에서 파악할 수 있다.
     단점 : 분기가 늘어날 때마다 코드가 변경된다.

   - 외부에 분기를 위임하고 경우의 수 만큼 처리기를 만드는 방식
     장점 : 분기가 늘어날 때마다 처리기만 추가하면 된다.
     단점 : 모든 경우의 수를 파악할 수 없다.

Condition

1. 조건 분기는 결코 제거할 수 없다.

2. 조건 분기에 대한 전략은 두 가지 뿐이다.
   - 내부에서 응집성있게 모아두는 방식
     장점 : 모든 경우의 수를 한 곳에서 파악할 수 있다.
     단점 : 분기가 늘어날 때마다 코드가 변경된다.

   - 외부에 분기를 위임하고 경우의 수 만큼 처리기를 만드는 방식
     장점 : 분기가 늘어날 때마다 처리기만 추가하면 된다.
     단점 : 모든 경우의 수를 파악할 수 없다.

Condition

1. 조건 분기는 결코 제거할 수 없다.

2. 조건 분기에 대한 전략은 두 가지 뿐이다.
   - 내부에서 응집성있게 모아두는 방식
     장점 : 모든 경우의 수를 한 곳에서 파악할 수 있다.
     단점 : 분기가 늘어날 때마다 코드가 변경된다.

   - 외부에 분기를 위임하고 경우의 수 만큼 처리기를 만드는 방식
     장점 : 분기가 늘어날 때마다 처리기만 추가하면 된다.
     단점 : 모든 경우의 수를 파악할 수 없다.

Condition

1. 조건 분기는 결코 제거할 수 없다.

2. 조건 분기에 대한 전략은 두 가지 뿐이다.
   - 내부에서 응집성있게 모아두는 방식
     장점 : 모든 경우의 수를 한 곳에서 파악할 수 있다.
     단점 : 분기가 늘어날 때마다 코드가 변경된다.

   - 외부에 분기를 위임하고 경우의 수 만큼 처리기를 만드는 방식
     장점 : 분기가 늘어날 때마다 처리기만 추가하면 된다.
     단점 : 모든 경우의 수를 파악할 수 없다.

Cohesion

```
void processCondition(String condition){
    if(condition.equals("a")){
        a();
    }else if(condition.equals("b")){
        b();
    }else if(condition.equals("c")){
        c();
    }else if(condition.equals("d")){
        d();
    }else if(condition.equals("e")){
        e();
    }
}
```

Injection

```
void main(){
    String v = "c";
    Runnable run = null;
    if(v.equals("a")){
        run = new Runnable(){public void run(){System.out.println("a");}};
    }else if(v.equals("b")){
        run = new Runnable(){public void run(){System.out.println("b");}};
    }else if(v.equals("c")){
        run = new Runnable(){public void run(){System.out.println("c");}};
    }
    processCondition(run);
}
void processCondition(Runnable condition){
    condition.run();
}
```

Responsibility Driven

value = responsibility

시스템의 존재 가치는 사용자에게 제공되는 기능

사용자가 사용할 기능 = 시스템의 책임

시스템 차원의 책임을 더 작은 단위의 책임으로 분할

해당 책임을 추상화하여 역할을 정의함.

역할에 따라 협력이 정의됨.

value = responsibility

시스템의 존재 가치는 사용자에게 제공되는 기능

사용자가 사용할 기능 = 시스템의 책임

시스템 차원의 책임을 더 작은 단위의 책임으로 분할

해당 책임을 추상화하여 역할을 정의함.

역할에 따라 협력이 정의됨.

value = responsibility

시스템의 존재 가치는 사용자에게 제공되는 기능

사용자가 사용할 기능 = 시스템의 책임

시스템 차원의 책임을 더 작은 단위의 책임으로 분할

해당 책임을 추상화하여 역할을 정의함.

역할에 따라 협력이 정의됨.

value = responsibility

시스템의 존재 가치는 사용자에게 제공되는 기능

사용자가 사용할 기능 = 시스템의 책임

시스템 차원의 책임을 더 작은 단위의 책임으로 분할

해당 책임을 추상화하여 역할을 정의함.

역할에 따라 협력이 정의됨.

value = responsibility

시스템의 존재 가치는 사용자에게 제공되는 기능

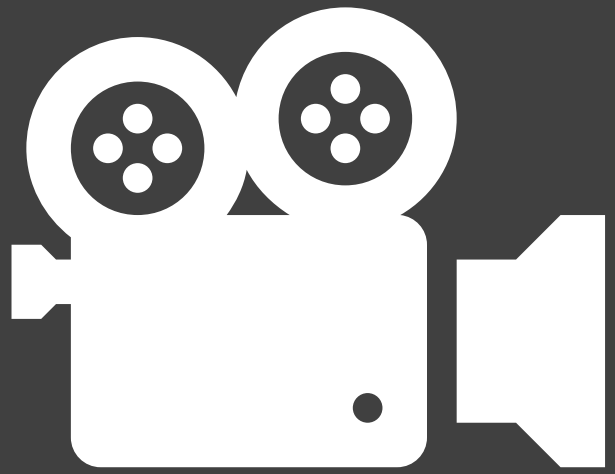
사용자가 사용할 기능 = 시스템의 책임

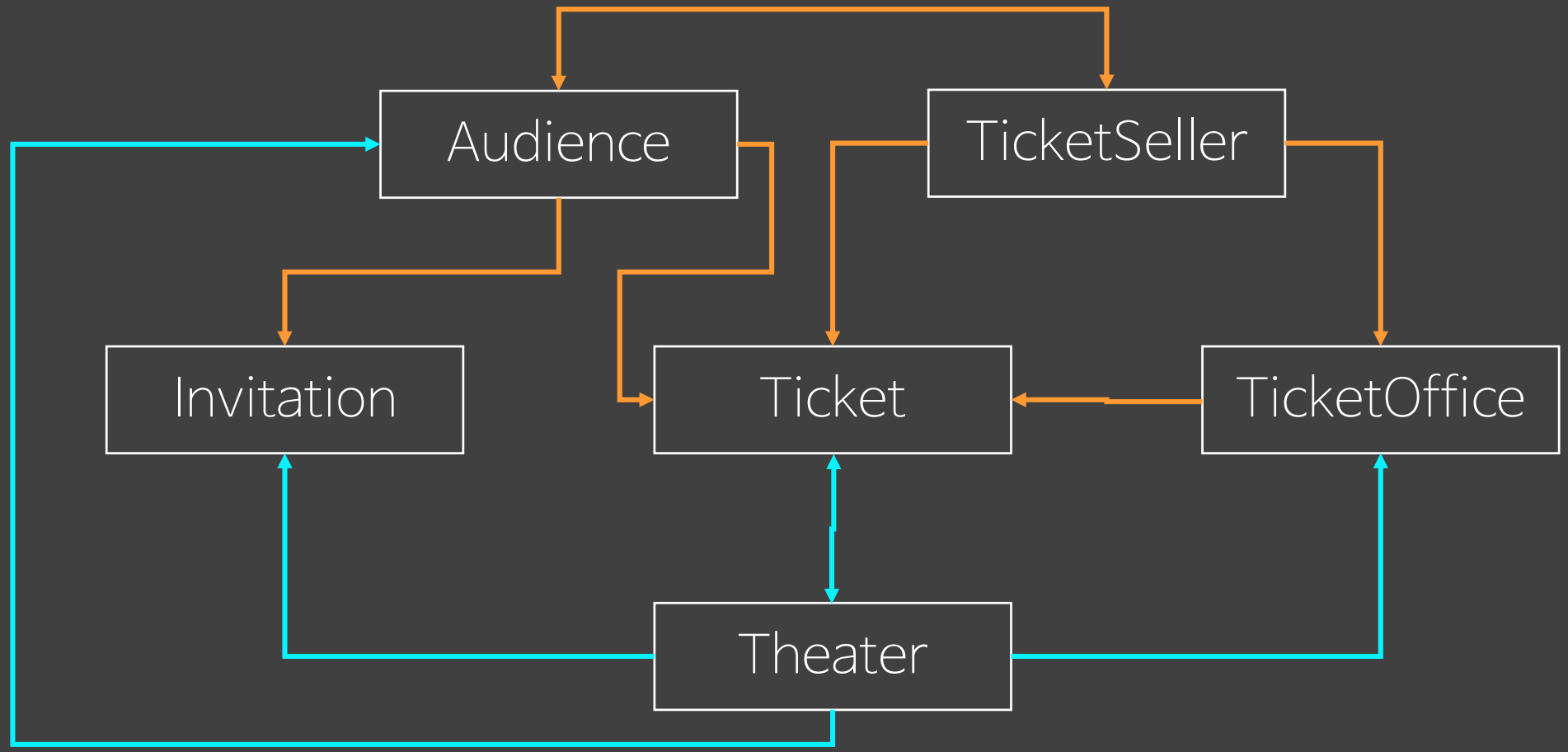
시스템 차원의 책임을 더 작은 단위의 책임으로 분할

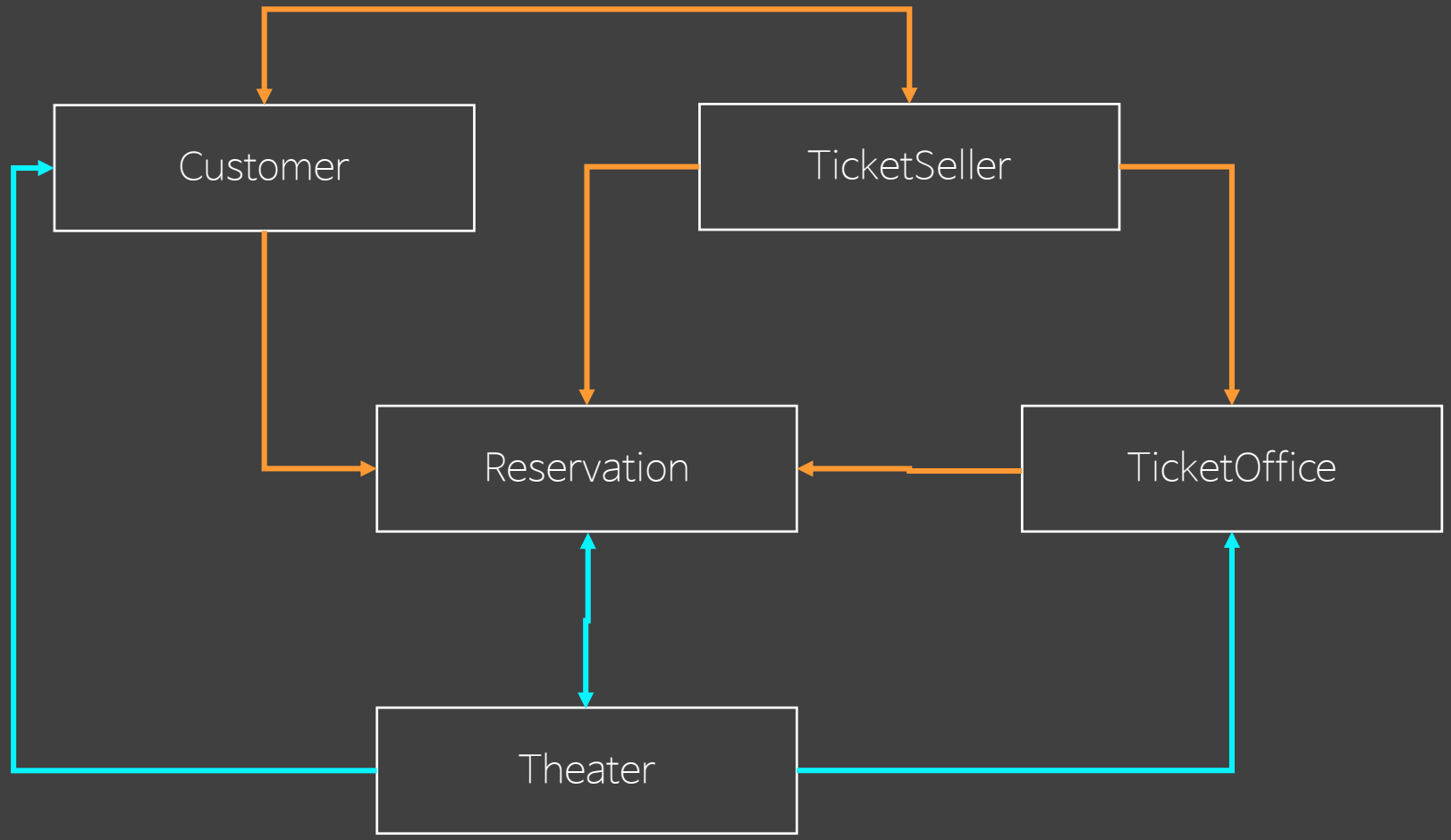
해당 책임을 추상화하여 역할을 정의함.

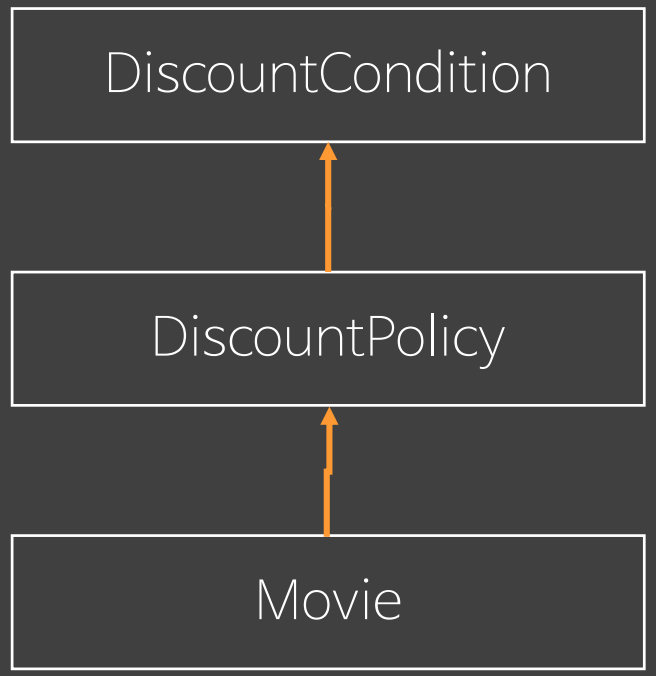
역할에 따라 협력이 정의됨.

Theater with Reservation




Audience
TicketSeller
Invitation
Ticket
TicketOffice
Theater


Customer
TicketSeller
Reservation
TicketOffice
Theater


DiscountCondition
DiscountPolicy
Movie

```
Theater theater = new Theater(Money.of(100.0));
Movie movie = new Movie<AmountDiscount>(
        "spiderman",
        Duration.ofMinutes(120L),
        Money.of(5000.0),
        new SequenceAmountDiscount(Money.of(1000.0), 1)
);
theater.addMovie(movie);
for(int day = 7; day < 32; day++){
    for(int hour = 10, seq = 1; hour < 24; hour += 3, seq++){
        theater.addScreening(
                movie,
                new Screening(
                        seq,
                        LocalDateTime.of(2019, 7, day, hour, 00, 00),
                        100
                )
        );
    }
}
```

```
Theater theater = new Theater(Money.of(100.0));
Movie movie = new Movie<AmountDiscount>(
        "spiderman",
        Duration.ofMinutes(120L),
        Money.of(5000.0),
        new SequenceAmountDiscount(Money.of(1000.0), 1)
);
theater.addMovie(movie);
for(int day = 7; day < 32; day++){
    for(int hour = 10, seq = 1; hour < 24; hour += 3, seq++){
        theater.addScreening(
                movie,
                new Screening(
                        seq,
                        LocalDateTime.of(2019, 7, day, hour, 00, 00),
                        100
                )
        );
    }
}
```

```
Theater theater = new Theater(Money.of(100.0));
Movie movie = new Movie<AmountDiscount>(
        "spiderman",
        Duration.ofMinutes(120L),
        Money.of(5000.0),
        new SequenceAmountDiscount(Money.of(1000.0), 1)
);
theater.addMovie(movie);
for(int day = 7; day < 32; day++){
    for(int hour = 10, seq = 1; hour < 24; hour += 3, seq++){
        theater.addScreening(
                movie,
                new Screening(
                        seq,
                        LocalDateTime.of(2019, 7, day, hour, 00, 00),
                        100
                )
        );
    }
}
```

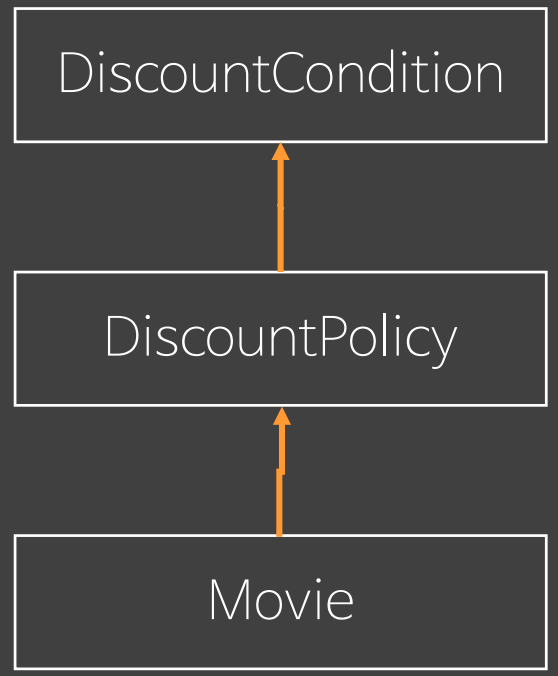
```
TicketOffice ticketOffice = new TicketOffice(Money.of(0.0));
theater.contractTicketOffice(ticketOffice, 10.0);
TicketSeller seller = new TicketSeller();
seller.setTicketOffice(ticketOffice);

for(Screening screening:theater.getScreening(movie)){
    customer.reverse(seller, theater, movie, screening, 2);
    boolean isOk = theater.enter(customer, 2);
    System.out.println(isOk);
    break;
}
```

```
TicketOffice ticketOffice = new TicketOffice(Money.of(0.0));
theater.contractTicketOffice(ticketOffice, 10.0);
TicketSeller seller = new TicketSeller();
seller.setTicketOffice(ticketOffice);

for(Screening screening:theater.getScreening(movie)){
    customer.reverse(seller, theater, movie, screening, 2);
    boolean isOk = theater.enter(customer, 2);
    System.out.println(isOk);
    break;
}
```

```
TicketOffice ticketOffice = new TicketOffice(Money.of(0.0));
theater.contractTicketOffice(ticketOffice, 10.0);
TicketSeller seller = new TicketSeller();
seller.setTicketOffice(ticketOffice);

Customer customer = new Customer(Money.of(20000.0));
for(Screening screening:theater.getScreening(movie)){
    customer.reverse(seller, theater, movie, screening, 2);
    boolean isOk = theater.enter(customer, 2);
    System.out.println(isOk);
    break;
}
```

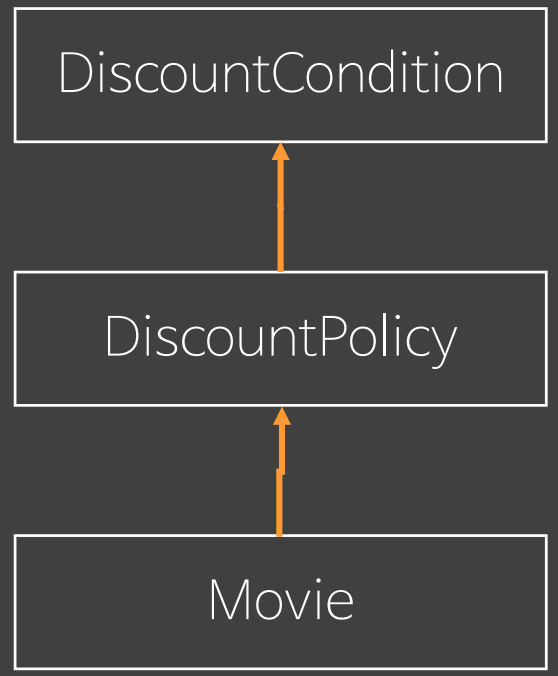
Discount Type

```
interface DiscountCondition{
    public boolean isSatisfiedBy(Screening screening, int audienceCount);
    public Money calculateFee(Money fee);
}
```



```
interface DiscountCondition{
    public boolean isSatisfiedBy(Screening screening, int audienceCount);
    public Money calculateFee(Money fee);
}
```

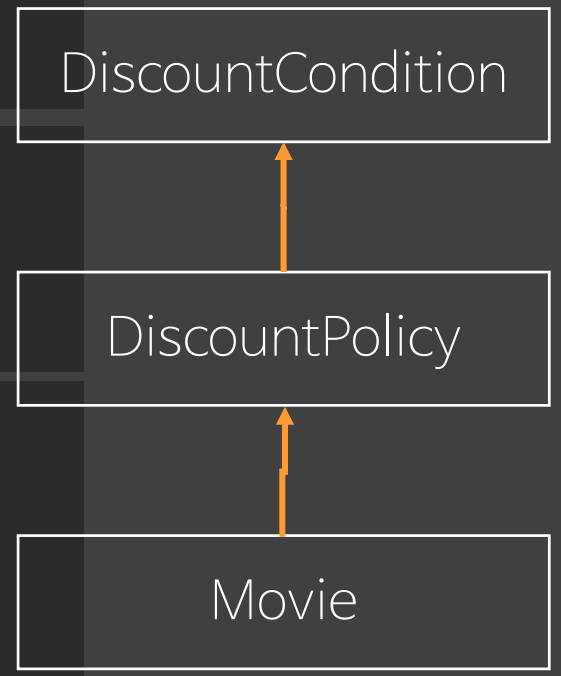
```
interface DiscountPolicy {
    interface AMOUNT extends DiscountPolicy{}
    interface PERCENT extends DiscountPolicy{}
    interface NONE extends DiscountPolicy{}
}
```



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

```java
interface DiscountPolicy {
    interface AMOUNT extends DiscountPolicy{}
    interface PERCENT extends DiscountPolicy{}
    interface NONE extends DiscountPolicy{}
}
```

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



```java
public class SequenceAmountDiscount extends SequenceDiscount implements DiscountPolicy.AMOUNT{
    private final Money amount;
    public SequenceAmountDiscount(int sequence, Money amount) {
        super(sequence);
        this.amount = amount;
    }
    @Override
    public Money calculateFee(Money fee) {
        return fee.minus(amount);
    }
}
```

```java
public class SequencePercentDiscount extends SequenceDiscount implements DiscountPolicy.PERCENT{
    private final double pecent;
    public SequencePercentDiscount(int sequence, double pecent) {
        super(sequence);
        this.pecent = pecent;
    }
    @Override
    public Money calculateFee(Money fee) {
        return fee.minus(fee.multi(pecent));
    }
}
```

implement

```java
abstract public class AmountDiscount implements DiscountPolicy.AMOUNT, DiscountCondition{
    private final Money amount;
    AmountDiscount(Money amount) {
        this.amount = amount;
    }
    @Override
    public final Money calculateFee(Money fee) {
        return fee.minus(amount);
    }
}
```

```java
abstract public class AmountDiscount implements DiscountPolicy.AMOUNT, DiscountCondition{
    private final Money amount;
    AmountDiscount(Money amount) {
        this.amount = amount;
    }
    @Override
    public final Money calculateFee(Money fee) {
        return fee.minus(amount);
    }
}
abstract public class PercentDiscount implements DiscountPolicy.PERCENT, DiscountCondition{
    private final double pecent;
    PercentDiscount(double pecent) {
        this.pecent = pecent;
    }
    @Override
    public final Money calculateFee(Money fee) {
        return fee.minus(fee.multi(pecent));
    }
}
```

```java
public class SequenceAmountDiscount extends AmountDiscount{
    private final int sequence;
    public SequenceAmountDiscount(Money amount, int sequence){
        super(amount);
        this.sequence = sequence;
    }
    @Override
    public boolean isSatisfiedBy(Screening screening, int audienceCount) {
        return screening.sequence == sequence;
    }
}
```

```java
public class SequenceAmountDiscount extends AmountDiscount{
    private final int sequence;
    public SequenceAmountDiscount(Money amount, int sequence){
        super(amount);
        this.sequence = sequence;
    }
    @Override
    public boolean isSatisfiedBy(Screening screening, int audienceCount) {
        return screening.sequence == sequence;
    }
}
```

```java
public class SequencePercentDiscount extends PercentDiscount{
    private final int sequence;
    public SequencePercentDiscount(double pecent, int sequence){
        super(pecent);
        this.sequence = sequence;
    }
    @Override
    public boolean isSatisfiedBy(Screening screening, int audienceCount) {
        return screening.sequence == sequence;
    }
}
```

```java
public class Movie<T extends DiscountPolicy & DiscountCondition> {
    private final String title;
    private final Duration runningTime;
    private final Money fee;
    private final Set<T> discountConditions = new HashSet<>();
    public Movie(String title, Duration runningTime, Money fee, T...conditions){
        this.title = title;
        this.runningTime = runningTime;
        this.fee = fee;
        this.discountConditions.addAll(Arrays.asList(conditions));
    }
    Money calculateFee(Screening screening, int audienceCount){
        for(T condition:discountConditions){
            if(condition.isSatisfiedBy(screening, audienceCount)){
                return condition.calculateFee(fee).multi((double)audienceCount);
            }
        }
        return fee.multi((double)audienceCount);
    }
}
```

```java
public class Movie<T extends DiscountPolicy & DiscountCondition> {
    private final String title;
    private final Duration runningTime;
    private final Money fee;
    private final Set<T> discountConditions = new HashSet<>();
    public Movie(String title, Duration runningTime, Money fee, T...conditions){
        this.title = title;
        this.runningTime = runningTime;
        this.fee = fee;
        this.discountConditions.addAll(Arrays.asList(conditions));
    }
    Money calculateFee(Screening screening, int audienceCount){
        for(T condition:discountConditions){
            if(condition.isSatisfiedBy(screening, audienceCount)){
                return condition.calculateFee(fee).multi((double)audienceCount);
            }
        }
        return fee.multi((double)audienceCount);
    }
}
```

value object

```java
public class Money {
    public static Money of(Double amount){return new Money(amount);}
    private final Double amount;
    Money(Double amount){this.amount = amount;}
    public Money minus(Money amount){
        return new Money(this.amount > amount.amount ? this.amount - amount.amount : 0.0);
    }
    public Money multi(Double times) {
        return new Money(this.amount * times);
    }
    public Money plus(Money amount) {
        return new Money(this.amount + amount.amount);
    }
    public boolean greaterThen(Money amount) {
        return this.amount >= amount.amount;
    }
}
```

```java
public class Money {
    public static Money of(Double amount){return new Money(amount);}
    private final Double amount;
    Money(Double amount){this.amount = amount;}
    public Money minus(Money amount){
        return new Money(this.amount > amount.amount ? this.amount - amount.amount : 0.0);
    }
    public Money multi(Double times) {
        return new Money(this.amount * times);
    }
    public Money plus(Money amount) {
        return new Money(this.amount + amount.amount);
    }
    public boolean greaterThen(Money amount) {
        return this.amount >= amount.amount;
    }
}
```

```java
public class Money {
    public static Money of(Double amount){return new Money(amount);}
    private final Double amount;
    Money(Double amount){this.amount = amount;}
    public Money minus(Money amount){
        return new Money(this.amount > amount.amount ? this.amount - amount.amount : 0.0);
    }
    public Money multi(Double times) {
        return new Money(this.amount * times);
    }
    public Money plus(Money amount) {
        return new Money(this.amount + amount.amount);
    }
    public boolean greaterThen(Money amount) {
        return this.amount >= amount.amount;
    }
}
```

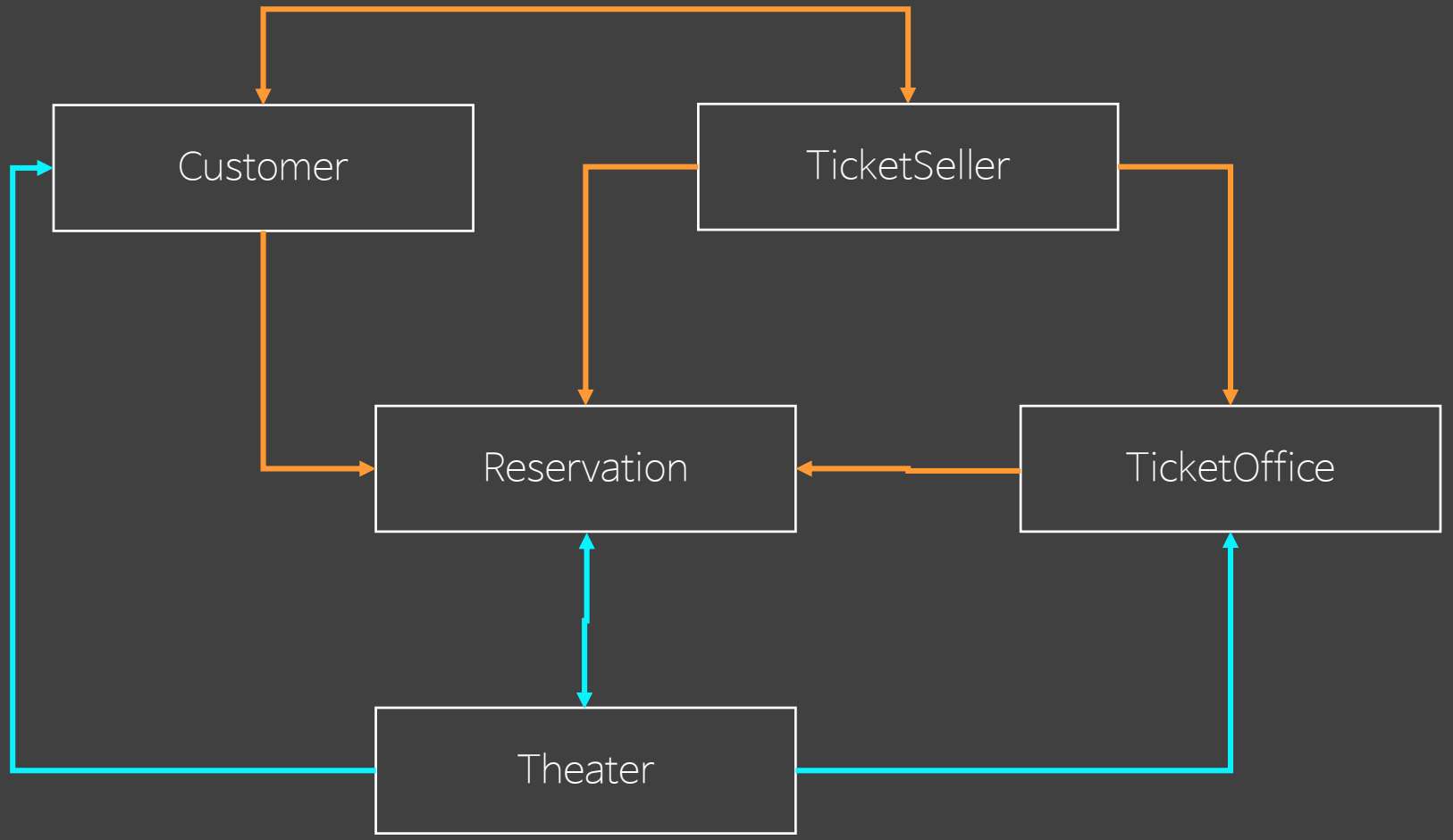
reservation & …

```java
public class Reservation {
    static final Reservation NONE = new Reservation(null, null, null, 0);

    final Theater theater;
    final Movie movie;
    final Screening screening;
    final int count;

    Reservation(Theater theater, Movie movie, Screening screening, int audienceCount){
        this.theater = theater;
        this.movie = movie;
        this.screening = screening;
        this.count = audienceCount;
    }
}
```

```java
public class Screening{
    private int seat;
    final int sequence;
    final LocalDateTime whenScreened;
    public Screening(int sequence, LocalDateTime when, int seat){
        this.sequence = sequence;
        this.whenScreened = when;
        this.seat = seat;
    }
    boolean hasSeat(int count){
        return seat >= count;
    }
    void reserveSeat(int count){
        if(hasSeat(count)) seat -= count;
        else throw new RuntimeException("no seat");
    }
}
```

```java
public class Screening{
    private int seat;
    final int sequence;
    final LocalDateTime whenScreened;
    public Screening(int sequence, LocalDateTime when, int seat){
        this.sequence = sequence;
        this.whenScreened = when;
        this.seat = seat;
    }
    boolean hasSeat(int count){
        return seat >= count;
    }
    void reserveSeat(int count){
        if(hasSeat(count)) seat -= count;
        else throw new RuntimeException("no seat");
    }
}
```

theater


Customer
TicketSeller
Reservation
TicketOffice
Theater

```java
public class Theater {
    public static final Set<Screening> EMPTY = new HashSet<>();
    private final Set<TicketOffice> ticketOffices = new HashSet<>();
    private final Map<Movie, Set<Screening>> movies = new HashMap<>();
    private Money amount;

    public Theater(Money amount){
        this.amount = amount;
    }
```

```java
public class Theater {
    public static final Set<Screening> EMPTY = new HashSet<>();
    private final Set<TicketOffice> ticketOffices = new HashSet<>();
    private final Map<Movie, Set<Screening>> movies = new HashMap<>();
    private Money amount;

    public Theater(Money amount){
        this.amount = amount;
    }

    public boolean addMovie(Movie movie){
        if(movies.containsKey(movie)) return false;
        movies.put(movie, new HashSet<>());
        return true;
    }
    public boolean addScreening(Movie movie, Screening screening){
        if(!movies.containsKey(movie)) return false;
        return movies.get(movie).add(screening);
    }
```

```
public boolean contractTicketOffice(TicketOffice ticketOffice, Double rate){
    if(!ticketOffice.contract(this, rate)) return false;
    return ticketOffices.add(ticketOffice);
}
public boolean cancelTicketOffice(TicketOffice ticketOffice){
    if(!ticketOffices.contains(ticketOffice) || !ticketOffice.cancel(this)) return false;
    return ticketOffices.remove(ticketOffice);
}
void plusAmount(Money amount){
    this.amount = this.amount.plus(amount);
}
```

```
public boolean contractTicketOffice(TicketOffice ticketOffice, Double rate){
    if(!ticketOffice.contract(this, rate)) return false;
    return ticketOffices.add(ticketOffice);
}
public boolean cancelTicketOffice(TicketOffice ticketOffice){
    if(!ticketOffices.contains(ticketOffice) || !ticketOffice.cancel(this)) return false;
    return ticketOffices.remove(ticketOffice);
}
void plusAmount(Money amount){
    this.amount = this.amount.plus(amount);
}

public Set<Screening> getScreening(Movie movie){
    if(!movies.containsKey(movie) || movies.get(movie).size() == 0) return EMPTY;
    return movies.get(movie);
}
boolean isValidScreening(Movie movie, Screening screening){
    return movies.containsKey(movie) && movies.get(movie).contains(screening);
}
```

```
public boolean enter(Customer customer, int count){
    Reservation reservation = customer.reservation;
    return reservation != Reservation.NONE &&
        reservation.theater == this &&
        isValidScreening(reservation.movie, reservation.screening) &&
        reservation.count == count;
}
Reservation reserve(Movie movie, Screening screening, int count){
    if(!isValidScreening(movie, screening) || !screening.hasSeat(count)) return Reservation.NONE;
    screening.reserveSeat(count);
    return new Reservation(this, movie, screening, count);
}
```

```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){
        if(commissionRate.containsKey(theater)) return false;
        commissionRate.put(theater, rate);
        return true;
    }
    boolean cancel(Theater theater){
        if(!commissionRate.containsKey(theater)) return false;
        commissionRate.remove(theater);
        return true;
    }
```

```
Reservation reserve(Theater theater, Movie movie, Screening screening, int count){
    if(
      !commissionRate.containsKey(theater) ||
      !theater.isValidScreening(movie, screening) ||
      !screening.hasSeat(count)
    ) return Reservation.NONE;
    Reservation reservation = theater.reserve(movie, screening, count);
    if(reservation != Reservation.NONE) {
        Money sales = movie.calculateFee(screening, count);
        Money commission = sales.multi(commissionRate.get(theater));
        amount = amount.plus(commission);
        theater.plusAmount(sales.minus(commission));
    }
    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;
    }
}
```

```
public class Customer{
    Reservation reservation = Reservation.NONE;
    private Money amount;
    public Customer(Money amount){this.amount = amount;}
    public void reverse(TicketSeller seller,Theater theater,Movie movie,Screening screening,int count){
        reservation = seller.reserve(this, theater, movie, screening, count);
    }
    boolean hasAmount(Money amount){
        return this.amount.greaterThen(amount);
    }
    void minusAmount(Money amount){
        this.amount = this.amount.minus(amount);
    }

}
```

Practice

practice #1

본 예제에서는 Sequence를 통한 할인조건만 구현되어 있다. Period 및 한번에 예약하는 사람의 수가 일정수를 넘어가면 할인해주는 Count 조건에 따른 할인조건을 구현하라.

practice #2

현재의 예제는 영화와 상영이라는 컨텍스트로 역할로 예매를 진행한다. 상영은 본디 시간표일 뿐이므로 좌석수 등을 갖을 수 없다.
극장이 상영관을 소유하게 하고 상영이 상영관과 협력하여 예매시의 잔여좌석수를 관리하도록 개선하라.