

CODE



SPITZ

83

OBJECT

1

2

3

4

5

6

# 합성과 의존성

# Template method

```
abstract class DiscountPolicy {  
    private Set<DiscountCondition> conditions = new HashSet<>();  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    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);  
}
```

# Template method

```
abstract class DiscountPolicy {  
    private Set<DiscountCondition> conditions = new HashSet<>();  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    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);  
}
```

# Template method

```
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);  
    }  
}
```

# Strategy

```
public class DiscountPolicy {
    private final Set<DiscountCondition> conditions = new HashSet<>();
    private final Calculator calculator;
    public DiscountPolicy(Calculator calculator){this.calculator = calculator;}
    public void addCondition(DiscountCondition condition){conditions.add(condition);}
    public Money calculateFee(Screening screening, int count, Money fee){
        for(DiscountCondition condition:conditions){
            if(condition.isSatisfiedBy(screening, count)) return calculator.calculateFee(fee);
        }
        return fee;
    }
}
```

# Strategy

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final Calculator calculator;  
    public DiscountPolicy(Calculator calculator){this.calculator = calculator;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)) return calculator.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

# Strategy

```
public class AmountCalculator implements Calculator {  
    private final Money amount;  
    public AmountCalculator(Money amount){  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```



# Strategy

```
public class AmountCalculator implements Calculator{  
    private final Money amount;  
    public AmountCalculator(Money amount){  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```

```
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);  
    }  
}
```

# Strategy

```
public class AmountCalculator implements Calculator{  
    private final Money amount;  
    public AmountCalculator(Money amount){  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```

```
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);  
    }  
}
```

# Strategy

```
public class AmountCalculator implements Calculator{  
    private final Money amount;  
    public AmountCalculator(Money amount){  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```

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

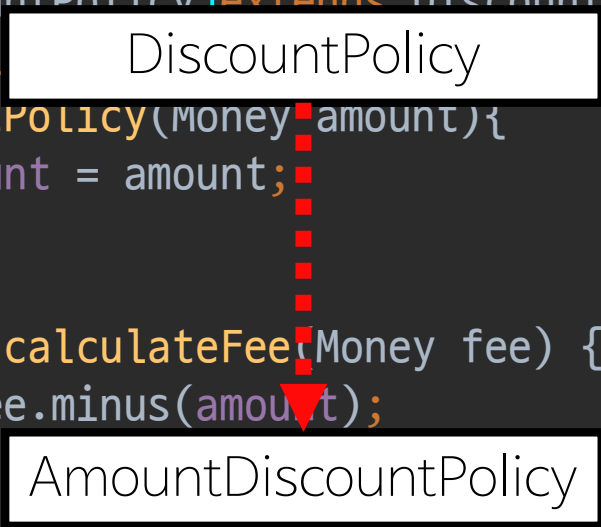
```
graph BT; ADP[AmountDiscountPolicy] --> DP[DiscountPolicy]
```

The diagram illustrates the relationship between the two classes. A white box labeled "DiscountPolicy" is positioned above the "AmountPolicy" class definition. A white box labeled "AmountDiscountPolicy" is positioned below the "AmountPolicy" class definition. A white arrow points from the "AmountDiscountPolicy" box up to the "DiscountPolicy" box, indicating that "AmountDiscountPolicy" inherits from "DiscountPolicy".

# Strategy

```
public class AmountCalculator implements Calculator{  
    private final Money amount;  
    public AmountCalculator(Money amount){  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```

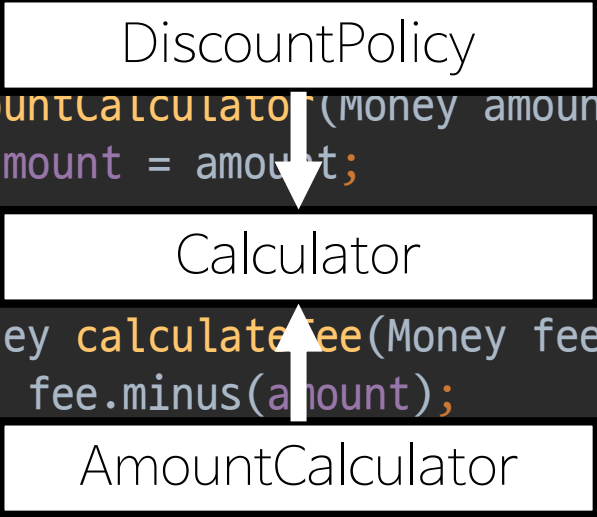
```
public class AmountPolicy extends DiscountPolicy{  
    private final  
    public AmountPolicy(Money amount){  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```



The diagram illustrates the relationship between the two classes. A solid red arrow points from the `AmountPolicy` class to the `DiscountPolicy` class, indicating that `AmountPolicy` inherits from `DiscountPolicy`. A dashed red arrow points from the `calculateFee` method in `AmountPolicy` to the `calculateFee` method in `DiscountPolicy`, indicating that `AmountPolicy` overrides the `calculateFee` method from `DiscountPolicy`.


# Strategy

```
public class AmountCalculator implements Calculator {  
    private final DiscountPolicy  
    public AmountCalculator(Money amount) {  
        this.amount = amount;  
    }  
    @Override  
    public Money calculateFee(Money fee) {  
        return fee.minus(amount);  
    }  
}
```



The diagram illustrates the Strategy design pattern for the AmountCalculator class. It shows three components: DiscountPolicy, Calculator, and AmountCalculator. A solid white arrow points from the DiscountPolicy box to the Calculator box, indicating that DiscountPolicy implements the Calculator interface. Another solid white arrow points from the AmountCalculator box to the Calculator box, indicating that AmountCalculator implements the Calculator interface. The code snippet shows AmountCalculator implementing the Calculator interface by using the DiscountPolicy class.

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



The diagram illustrates the Strategy design pattern for the AmountPolicy class. It shows two components: DiscountPolicy and AmountPolicy. A dashed red arrow points from the AmountPolicy box to the DiscountPolicy box, indicating that AmountPolicy extends the DiscountPolicy class. The code snippet shows AmountPolicy extending the DiscountPolicy class and overriding the calculateFee method.

# Template Method & Strategy

# Template Method & Strategy

템플릿 : 런타임에 타입선택(세트)  
추상메소드로 의존성 역전

# Template Method & Strategy

템플릿 : 런타임에 타입선택(세트)  
추상메소드로 의존성 역전

전략 : 런타임에 합성(조립)  
추가 인터페이스로 의존성 분산



# Template Method & Strategy

템플릿 : 런타임에 타입선택(세트)  
추상메소드로 의존성 역전 → 조합  
폭발

전략 : 런타임에 합성(조립)  
추가 인터페이스로 의존성 분산

# Template Method & Strategy

템플릿 : 런타임에 타입선택(세트)  
추상메소드로 의존성 역전 → 조합  
폭발

전략 : 런타임에 합성(조립)  
추가 인터페이스로 의존성 분산 → 의존성  
폭발

# Template Method & Strategy

템플릿 : 런타임에 타입선택(세트)  
추상메소드로 의존성 역전



전략 : 런타임에 합성(조립)  
추가 인터페이스로 의존성 분산



# 생성사용패턴과 팩토리

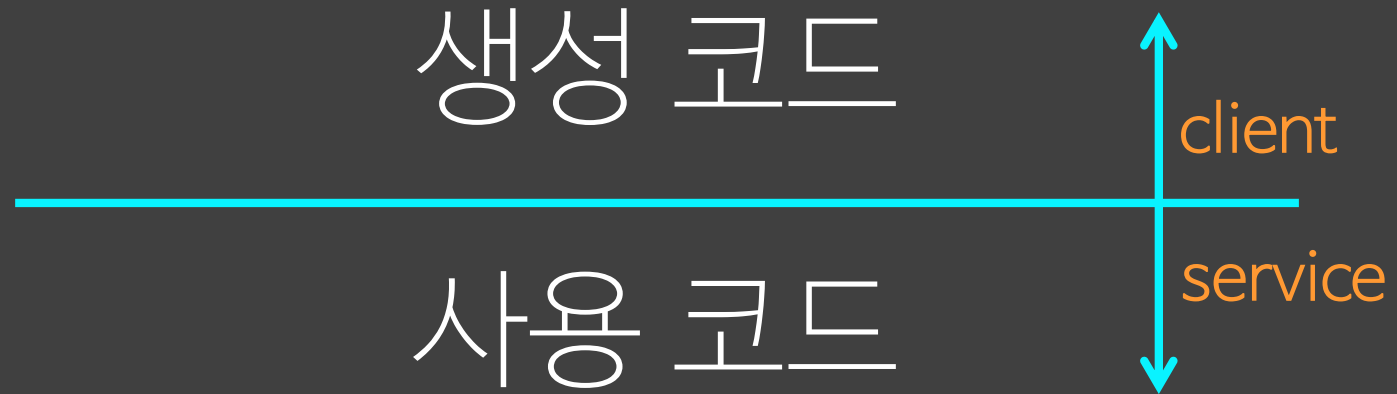
# 생성사용패턴

생성 코드

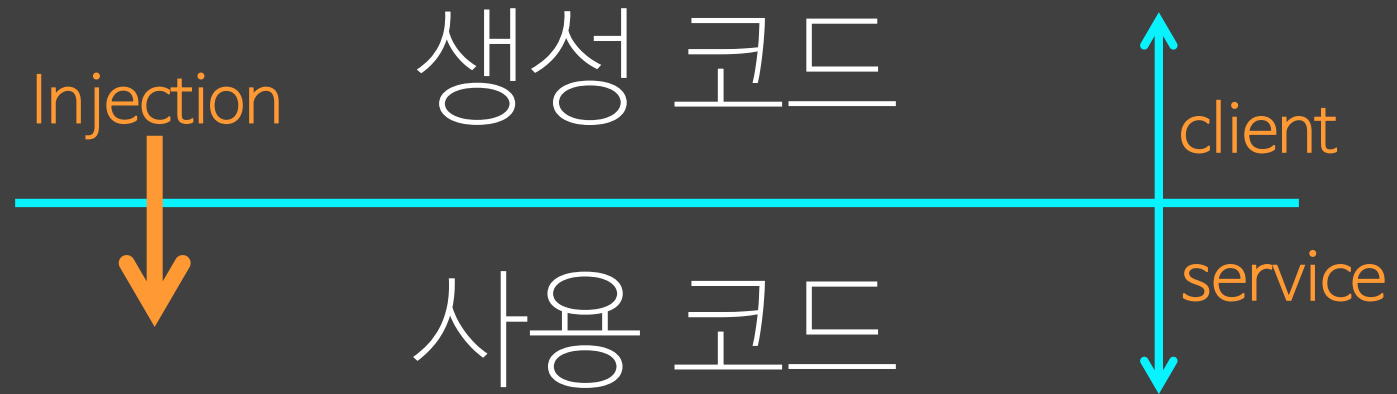
---

사용 코드

# 생성사용패턴



# 생성사용패턴



# Injection

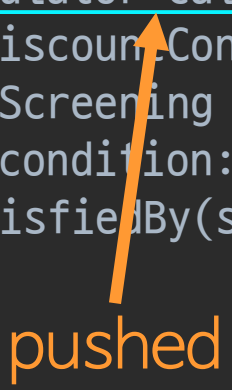


# Injection

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final Calculator calculator;  
    public DiscountPolicy(Calculator calculator){this.calculator = calculator;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)) return calculator.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

# Injection

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final Calculator calculator;  
    public DiscountPolicy(Calculator calculator){this.calculator = calculator;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)) return calculator.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```



The diagram illustrates the concept of dependency injection. An orange arrow points from the word "pushed" to the `Calculator calculator` parameter in the `DiscountPolicy` constructor. The parameter is also enclosed in a light blue rectangular box.

# Factory

```
public interface CalculatorFactory {  
    Calculator getCalculator();  
}
```

# Factory

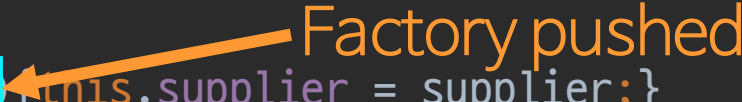
```
public interface CalculatorFactory {  
    Calculator getCalculator();  
}  
  
public class AmountCalculatorFactory implements CalculatorFactory {  
    private final Money money;  
    private AmountCalculator cache;  
    public AmountCalculatorFactory(Money money){this.money = money;}  
    @Override  
    synchronized public Calculator getCalculator(){  
        if(cache == null) cache = new AmountCalculator(money);  
        return cache;  
    }  
}
```

# Lazy Pull

```
public class DiscountPolicy {
    private final Set<DiscountCondition> conditions = new HashSet<>();
    private final CalculatorFactory supplier;
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}
    public void addCondition(DiscountCondition condition){conditions.add(condition);}
    public Money calculateFee(Screening screening, int count, Money fee){
        for(DiscountCondition condition:conditions){
            if(condition.isSatisfiedBy(screening, count)){
                return supplier.getCalculator().calculateFee(fee);
            }
        }
        return fee;
    }
}
```

# Lazy Pull

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.getCalculator().calculateFee(fee);  
            }  
        }  
        return fee;  
    }  
}
```



Factory pushed

# Lazy Pull

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.getCalculator().calculateFee(fee);  
            }  
        }  
        return fee;  
    }  
}
```

Factory pushed

Lazy pulled

# Lazy Pull

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.getCalculator().calculateFee(fee); 디미터법칙 위반  
            }  
        }  
        return fee;  
    }  
}
```

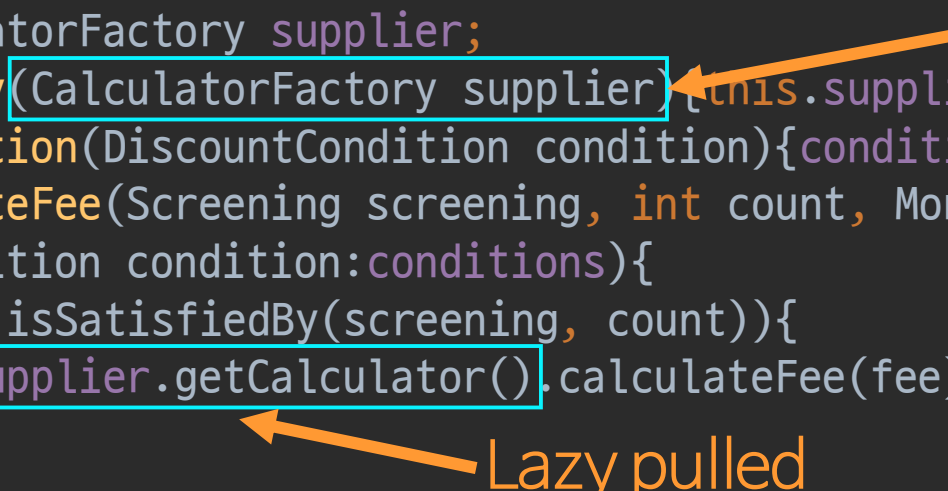
Factory pushed

Lazy pulled



# Lazy Pull

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.getCalculator().calculateFee(fee);  
            }  
        }  
        return fee;  
    }  
}
```



Factory pushed

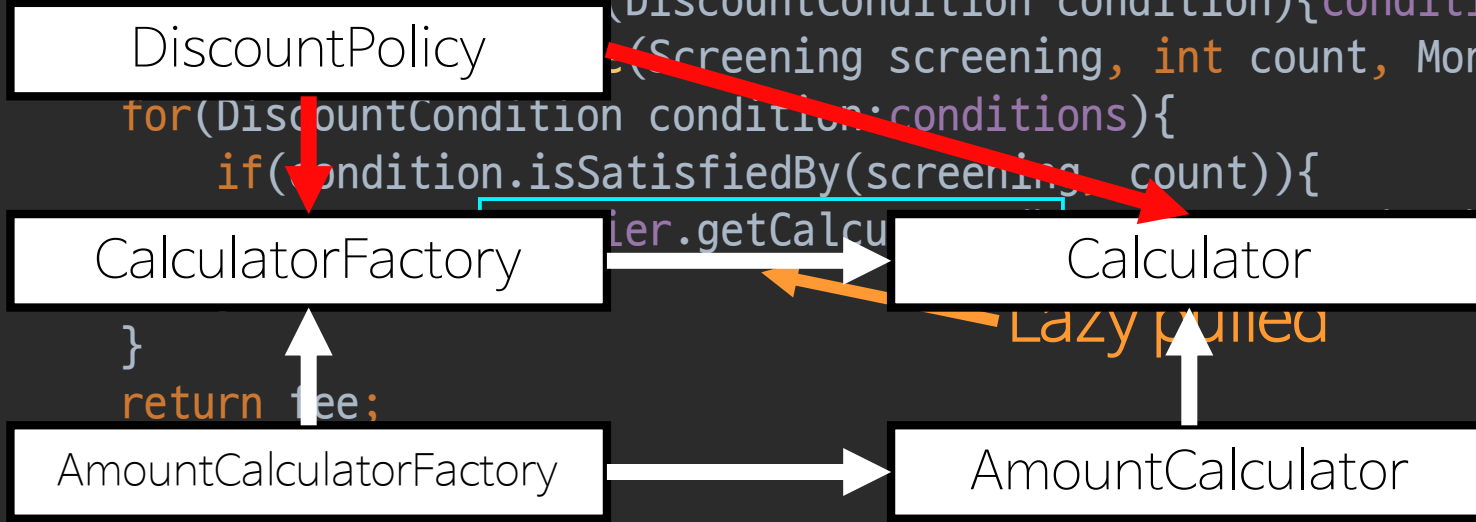
Lazy pulled

디미터법칙 위반

1. factory와 calculator를 알게
2. factory만 알게

# Lazy Pull

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculate(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.getCalculator(screening, count).calculate(fee);  
            }  
        }  
        return fee;  
    }  
}
```



디미터법칙 위반

1. factory와 calculator를 알게

2. factory만 알게

# Lazy Pull

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculate(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return fee;  
            }  
        }  
        return supplier.getCalculator().calculate(screening, count, fee);  
    }  
}
```

DiscountPolicy

CalculatorFactory

Calculator

AmountCalculatorFactory

AmountCalculator

Factory pushed

Lazy pulled

디미터법칙 위반

1. factory와 calculator를 알게

2. factory만 알게

# 위임된 팩토리

```
public interface CalculatorFactory {  
    Money calculateFee(Money fee);  
}
```

# 위임된 팩토리

```
public interface CalculatorFactory {  
    Money calculateFee(Money fee);  
}  
  
public class AmountCalculatorFactory implements CalculatorFactory {  
    private final Money money;  
    private AmountCalculator cache;  
    public AmountCalculatorFactory(Money money){this.money = money;}  
    synchronized private Calculator getCalculator(){  
        if(cache == null) cache = new AmountCalculator(money);  
        return cache;  
    }  
    @Override  
    public Money calculateFee(Money fee){return getCalculator().calculateFee(fee);}  
}
```

# 위임된 팩토리

```
public class DiscountPolicy {
    private final Set<DiscountCondition> conditions = new HashSet<>();
    private final CalculatorFactory supplier;
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}
    public void addCondition(DiscountCondition condition){conditions.add(condition);}
    public Money calculateFee(Screening screening, int count, Money fee){
        for(DiscountCondition condition:conditions){
            if(condition.isSatisfiedBy(screening, count)){
                return supplier.calculateFee(fee);
            }
        }
        return fee;
    }
}
```

# 위임된 팩토리

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.calculateFee(fee);  
            }  
        }  
        return fee;  
    }  
}
```

Calculator?

# 위임된 팩토리

```
public interface CalculatorFactory {  
    Money calculateFee(Money fee);  
}  
  
public class AmountCalculatorFactory implements CalculatorFactory {  
    private final Money money;  
    private AmountCalculator cache;  
    public AmountCalculatorFactory(Money money){this.money = money;}  
    synchronized private Calculator getCalculator(){  
        if(cache == null) cache = new AmountCalculator(money);  
        return cache;  
    }  
    @Override  
    public Money calculateFee(Money fee){return getCalculator().calculateFee(fee);}  
}
```



# 위임된 팩토리

```
public class DiscountPolicy {
    private final Set<DiscountCondition> conditions = new HashSet<>();
    private final CalculatorFactory supplier;
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}
    public void addCondition(DiscountCondition condition){conditions.add(condition);}
    public Money calculateFee(Screening screening, int count, Money fee){
        for(DiscountCondition condition:conditions){
            if(condition.isSatisfiedBy(screening, count)){
                return supplier.calculateFee(fee);
            }
        }
        return fee;
    }
}
```

# 위임된 팩토리

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculate(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.calculate(screening, count, fee);  
            }  
        }  
        return fee;  
    }  
}
```

DiscountPolicy

CalculatorFactory

Calculator

AmountCalculatorFactory

AmountCalculator



# 위임된 팩토리

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final CalculatorFactory supplier;  
    public DiscountPolicy(CalculatorFactory supplier){this.supplier = supplier;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return supplier.calculateFee(fee);  
            }  
        }  
        return fee;  
    }  
}
```

```
graph TD  
    DP[DiscountPolicy] --> C[Calculator]  
    ACF[AmountCalculatorFactory] --> C  
    ACF --> AC[AmountCalculator]
```

추상 팩토리 메소드 패턴

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final Set<DiscountCondition> conditions = new HashSet<>();  
    private final Calculator factory;  
    public DiscountPolicy(Calculator factory){this.factory = factory;}  
    public void addCondition(DiscountCondition condition){conditions.add(condition);}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:conditions){  
            if(condition.isSatisfiedBy(screening, count)){  
                return factory.calculateFee(fee);  
            }  
        }  
        return fee;  
    }  
}
```

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition: factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

# DiscountCondition 위임

```
public interface PolicyFactory extends Calculator{
    Set<DiscountCondition> getConditions();
}

public class AmountCalculatorFactory implements PolicyFactory{
    private final Money money;
    private AmountCalculator cache;
    private final Set<DiscountCondition> conditions = new HashSet<>();
    public AmountCalculatorFactory(Money money){this.money = money;}
    synchronized private Calculator getCalculator(){
        if(cache == null) cache = new AmountCalculator(money);
        return cache;
    }
    public void addCondition(DiscountCondition condition){conditions.add(condition);}
    public void removeCondition(DiscountCondition condition){conditions.remove(condition);}
    @Override public Money calculateFee(Money fee){return getCalculator().calculateFee(fee);}
    @Override public Set<DiscountCondition> getConditions(){return conditions;}
}
```



# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

디미터법칙 위반

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

디미터법칙 위반                      디미터법칙 위반

factory만 알게

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

디미터법칙 위반                      디미터법칙 위반

factory만 알게              여러 객체에 대한 의존성 전체를 위임

# DiscountCondition 위임

```
public class DiscountPolicy {  
    private final PolicyFactory factory;  
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}  
    public Money calculateFee(Screening screening, int count, Money fee){  
        for(DiscountCondition condition:factory.getConditions()){  
            if(condition.isSatisfiedBy(screening, count)) return factory.calculateFee(fee);  
        }  
        return fee;  
    }  
}
```

# DiscountCondition 위임

```
public interface PolicyFactory extends Calculator{
    default Money calculateFee(Screening screening, int count, Money fee) {
        for(DiscountCondition condition:getConditions()){
            if(condition.isSatisfiedBy(screening, count)) return calculateFee(fee);
        }
        return fee;
    }
    Set<DiscountCondition> getConditions();
}
```

# DiscountCondition 위임

```
public interface PolicyFactory extends Calculator{
    default Money calculateFee(Screening screening, int count, Money fee) {
        for(DiscountCondition condition:getConditions()){
            if(condition.isSatisfiedBy(screening, count)) return calculateFee(fee);
        }
        return fee;
    }
    Set<DiscountCondition> getConditions();
}
```

```
public class DiscountPolicy {
    private final PolicyFactory factory;
    public DiscountPolicy(PolicyFactory factory){this.factory = factory;}
    public Money calculateFee(Screening screening, int count, Money fee){
        return factory.calculateFee(screening, count, fee);
    }
}
```