CODE  SPITZ

84

OBJECT 2

1 2 3 4 5 6

계약(contract)

계약(contract)

계약(contract)

전달받은 메세지의 규격(precondition)

계약(contract)

전달받은 메세지의 규격(precondition)

전달할 메세지의 규격(postcondition)

계약(contract)

전달받은 메세지의 규격(precondition)

전달할 메세지의 규격(postcondition)

객체 자신의 규격(class invariant)

계약(contract)

전달받은 메세지의 규격(precondition)

전달할 메세지의 규격(postcondition)

객체 자신의 규격(class invariant)

위임된 책임의 컨텍스트

Invariant

불변식(invariant)

불변식(invariant)

메세지와 무관한 객체의 상태

불변식(invariant)

메세지와 무관한 객체의 상태

일반적으로 필드값의 상태 점검

불변식(invariant)

메세지와 무관한 객체의 상태

일반적으로 필드값의 상태 점검

DI에게 위임하거나 초기화 할당으로 처리

```java
public class Plan{
    private Calculator calc;
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Plan{
    private Calculator calc;
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Plan{
    private Calculator calc;
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        calls.add(call);
    }
    public final void setCalculator(Ca
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls,
    }
}
```

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        return result;
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

precondition

사전조건(precondition)

일반적으로 validation

메시지로 받는 값을 스스로 검증함

검증이 확정된 형으로 갈음할 수 있음

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

null문제

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

1. 언어나 컴파일러의 기능

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(@NonNull Call call){
        calls.add(call);
    }
    public final void setCalculator(@NonNull Calculator calc){
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

2. if로 무시

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call != null) calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc != null) this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

3. 예외로 처리

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

추가적인 상태 검사

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

추가적인 상태 검사

```
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        if(calc.isEmpty()) throw new IllegalArgumentException("calc is empty");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

postcondition

사후조건(postcondition)

일반적으로 결과값 검증이라 함

보내줄 값이 올바름을 검증함

검증이 확정된 형으로 갈음할 수 있음

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        if(calc.isEmpty()) throw new IllegalArgumentException("calc is empty");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){...}
    public final void setCalculator(Calculator calc){...}
    public final Money calculateFee(){
        Money result = calc.calcCallFee(calls, Money.ZERO);
        if(calls.size() > 0 && (
            result.equals(Money.ZERO) ||
            result.isLessThan(Money.ZERO)
        )) throw new RuntimeException("calculate error");
        return result;
    }
}
```

계약별 책임할당

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        if(calc.isEmpty()) throw new IllegalArgumentException("calc is empty");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void
        if(calc == nu
        if(calc.isEmp
        this.calc = ca
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

According to inferred contract, method 'addCall' throws exception when parameter == null

```
16    plan.addCall(null);
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        if(calc.isEmpty()) throw new IllegalArgumentException("calc is empty");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Plan{
    private Calculator calc = new Ca
    private Set<Call> calls = new Ha
    public final void addCall(Call c
        if(call == null) throw new I
        calls.add(call);
    }
    public final void setCalculator(
        if(calc == null) throw new I
        if(calc.isEmpty()) throw new
        this.calc = calc;
    }
    public final Money calculateFee(
        return calc.calcCallFee(call
    }
}
```

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        return result;
    }
    public final boolean isEmpty(){
        return calcs.size() > 0;
    }
}
```

```
public class Plan{
    private Calculato
    private Set<Call>
    public final voic
        if(call == nu
        calls.add(cal
    }
    public final voic
        if(calc == nu
        if(calc.isEmp
        this.calc = c
    }
    public final Mone
        return calc.c
    }
}
```

```
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        return result;
    }
    public final void check(){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        calc.check();
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

사후계약조건 책임할당

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){...}
    public final void setCalculator(Calculator calc){...}
    public final Money calculateFee(){
        Money result = calc.calcCallFee(calls, Money.ZERO);
        if(calls.size() > 0 && (
            result.equals(Money.ZERO) ||
            result.isLessThan(Money.ZERO)
        )) throw new RuntimeException("calculate error");
        return result;
    }
}
```

```java
public class Plan{
    private Calculator
    private Set<Call> c
    public final void a
    public final void s
    public final Money
        Money result =
        if(calls.size()
            result.equa
            result.isLe
        )) throw new Ru
        return result;
    }
}
```

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(calls.size() > 0 && (
            result.equals(Money.ZERO) ||
            result.isLessThan(Money.ZERO)
        )) throw new RuntimeException("calculate error");
        return result;
    }
    public final void check(){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        calc.check();
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calc.calcCallFee(calls, Money.ZERO);
    }
}
```

협력을 통한 책임분할

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(calls.size() > 0 && (
            result.equals(Money.ZERO) ||
            result.isLessThan(Money.ZERO)
        )) throw new RuntimeException("calculate error");
        return result;
    }
    public final void check(){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        calc.check();
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

런타임에 의한 계약조건

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
    public final void check(){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
    }
}
```

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
    public final void check(){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        calc.check();
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
}
```

```java
public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        if(next == null) throw new IllegalArgumentException("next is null");
        calcs.add(next);
        return this;
    }
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
}
```

계약의 전파

```java
public class PricePerTime implements Calc{
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        this.price = price;
        this.second = second;
    }
    @Override
    public Money calc(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
             price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime implements Calc{
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        this.price = price;
        this.second = second;
    }
    @Override
    public Money calc(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
             price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime implements Calc{
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        this.price = price;
        this.second = second;
    }
    @Override
    public Money calc(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
             price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime implements Calc{
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price
        this.price = price;
        this.second = second;
    }
    @Override
    public Money calc(Set<Call> cal
        for(Call call:calls) result
             price.times((call.getD
        );
        return result;
    }
}
```

```java
public class Calculator {
    ...
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new Ill...
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
}
```

```java
public class PricePerTime implements Calc{
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price
        this.price = price;
        this.second = second;
    }
    @Override
    public Money calc(Set<Call> cal
        for(Call call:calls) result
            price.times((call.getD
        );
        return result;
    }
}
```

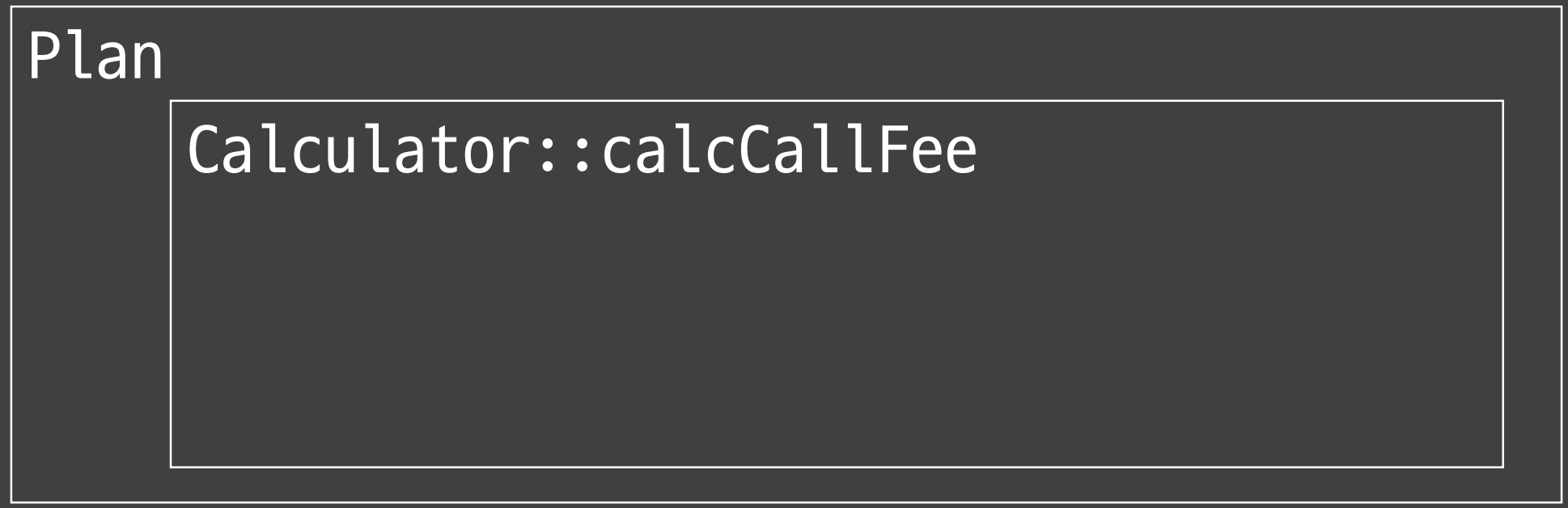
```java
public class Calculator {
    ...
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new Ill...
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
}
```

```java
public class PricePerTime implements Calc{
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price
        this.price = price;
        this.second = second;
    }
    @Over
    publi
        f
        )
        return result;
    }
}
```

```java
public class Calculator {
    ...
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new Ill...
    }
}
```

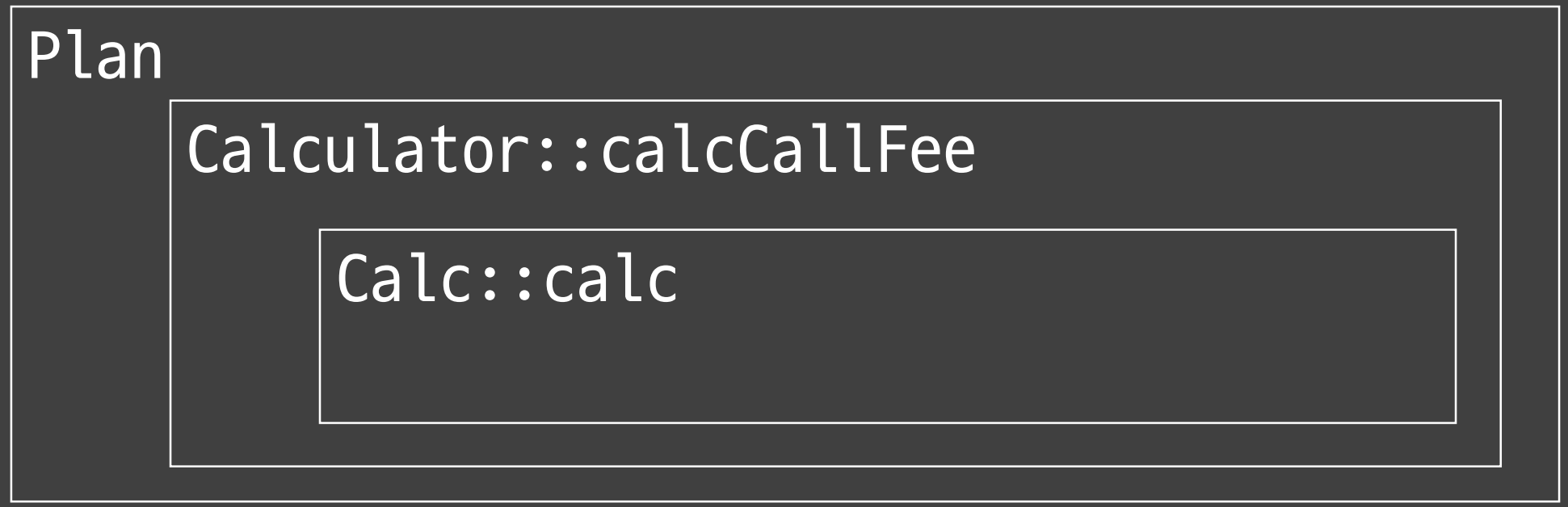
```java
public class Plan{
    ...
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

가시성을 통한 계약보증


Plan
Calculator::calcCallFee

```java
package plan;

public class Plan{
    private Calculator calc = new Calculator();
    private Set<Call> calls = new HashSet<>();
    public final void addCall(Call call){
        if(call == null) throw new IllegalArgumentException("call is null");
        calls.add(call);
    }
    public final void setCalculator(Calculator calc){
        if(calc == null) throw new IllegalArgumentException("calc is null");
        this.calc = calc;
    }
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
package plan;

public class Calculator {
    private Set<Calc> calcs = new HashSet<>();
    public final Calculator setNext(Calc next){
        if(next == null) throw new IllegalArgumentException("next is null");
        calcs.add(next);
        return this;
    }
    final Money calcCallFee(Set<Call> calls, Money result){          internal
        if(calcs.size() > 0) throw new IllegalArgumentException("calc is empty");
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
}
```


Plan
Calculator::calcCallFee
Calc::calc

```java
package plan;

interface Calc {
    Money calc(Set<Call> calls, Money result);
}
```

```java
package plan;

public class Tex implements Calc{
    private final double ratio;
    public Tex(double ratio){
        this.ratio = ratio;
    }
    @Override
    public Money calc(Set<Call> calls, Money result) {
        return result.plus(result.times(ratio));
    }
}
```

```java
package plan;

abstract class Calc {
    abstract Money calc(Set<Call> calls, Money result);
}
```

```java
package plan;

public class Tex implements Calc{
    private final double ratio;
    public Tex(double ratio){
        this.ratio = ratio;
    }
    @Override
    public Money calc(Set<Call> calls, Money result) {
        return result.plus(result.times(ratio));
    }
}
```

```java
package plan;

abstract class Calc {
    abstract Money calc(Set<Call> calls, Money result);
}
```

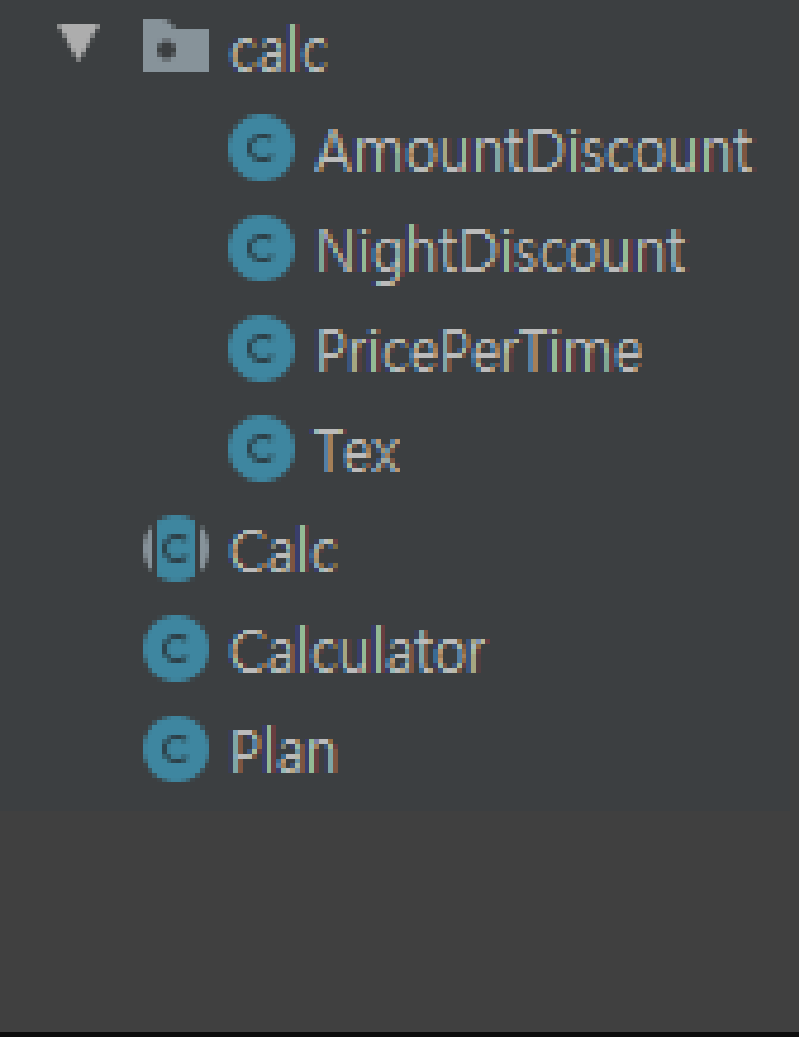
```java
package plan;

public class Tex extends Calc{
    private final double ratio;
    public Tex(double ratio){
        this.ratio = ratio;
    }
    @Override
    final Money calc(Set<Call> calls, Money result) {
        return result.plus(result.times(ratio));
    }
}
```

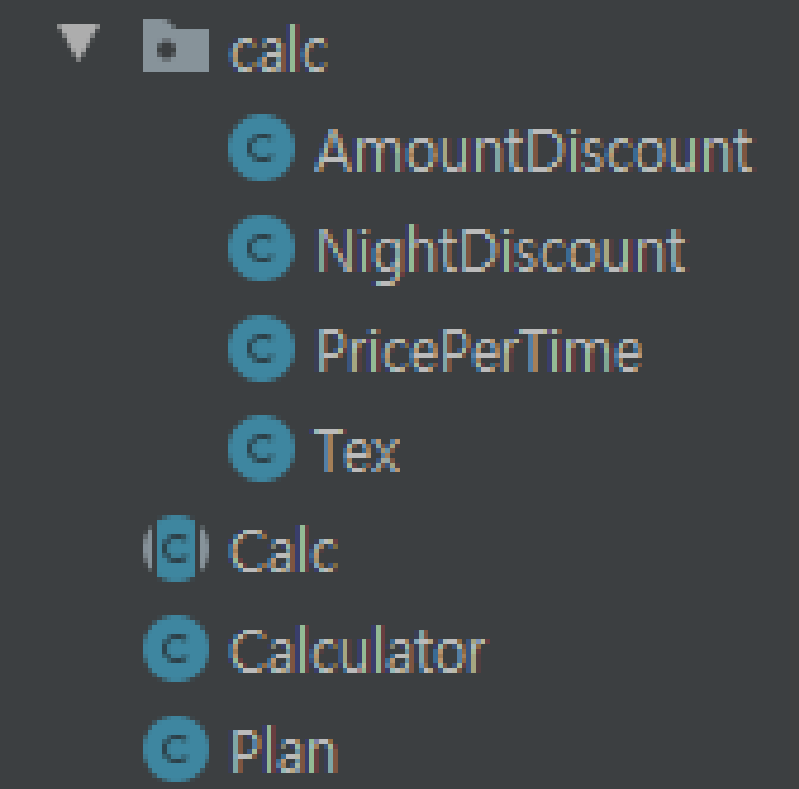
```java
package plan;

abstract class Calc {
    Money calc(Set<Call> calls, Money result){return calculate(calls, result);}
    abstract protected Money calculate(Set<Call> calls, Money result);
}
```

```java
package plan;

public class Tex extends Calc{
    private final double ratio;
    public Tex(double ratio){
        this.ratio = ratio;
    }
    @Override
    final Money calc(Set<Call> calls, Money result) {
        return result.plus(result.times(ratio));
    }
}
```

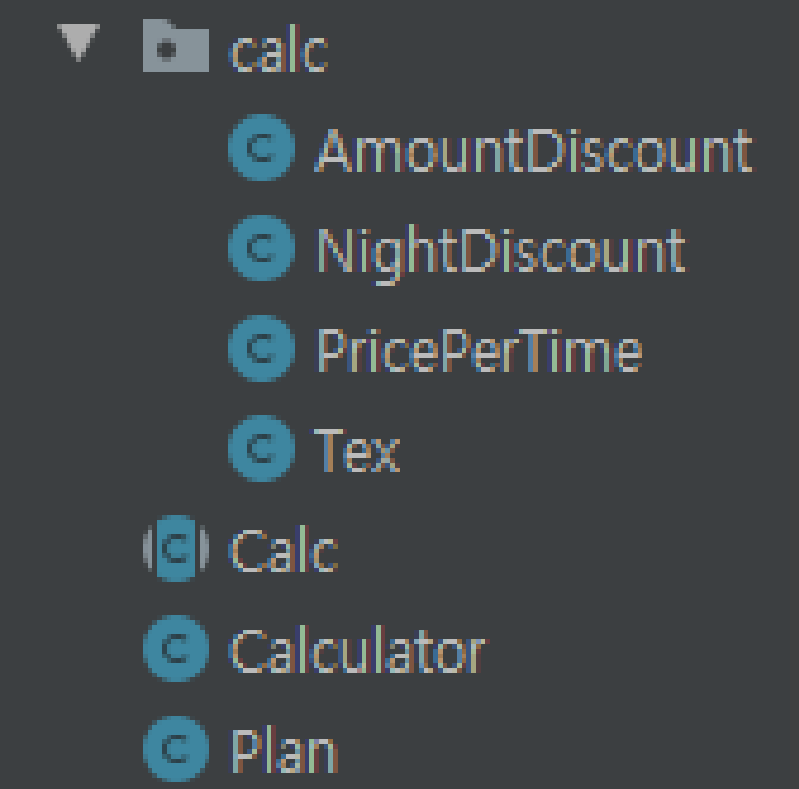
```java
package plan;

public abstract class Calc {
    Money calc(Set<Call> calls, Money result){return calculate(calls, result);}
    abstract protected Money calculate(Set<Call> calls, Money result);
}
```

```java
package plan.calc;

public class Tex extends Calc{
    private final double ratio;
    public Tex(double ratio){
        this.ratio = ratio;
    }
    @Override
    final protected Money calculate(Set<Call> calls, Money result) {
        return result.plus(result.times(ratio));
    }
}
```


calc
AmountDiscount
NightDiscount
PricePerTime
Tex
Calc
Calculator
Plan



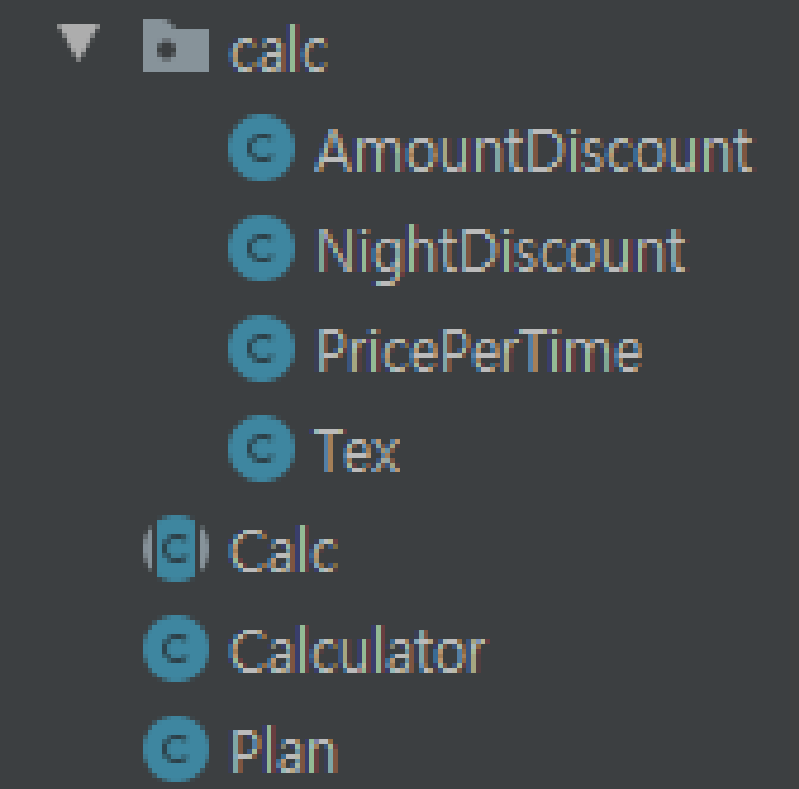

```
public
Plan::calculateFee()
```

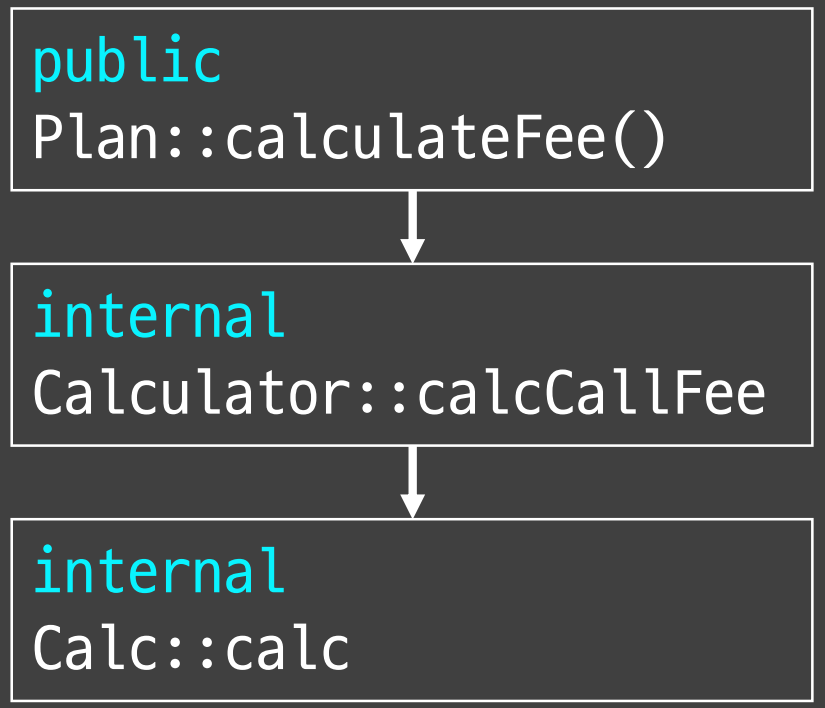


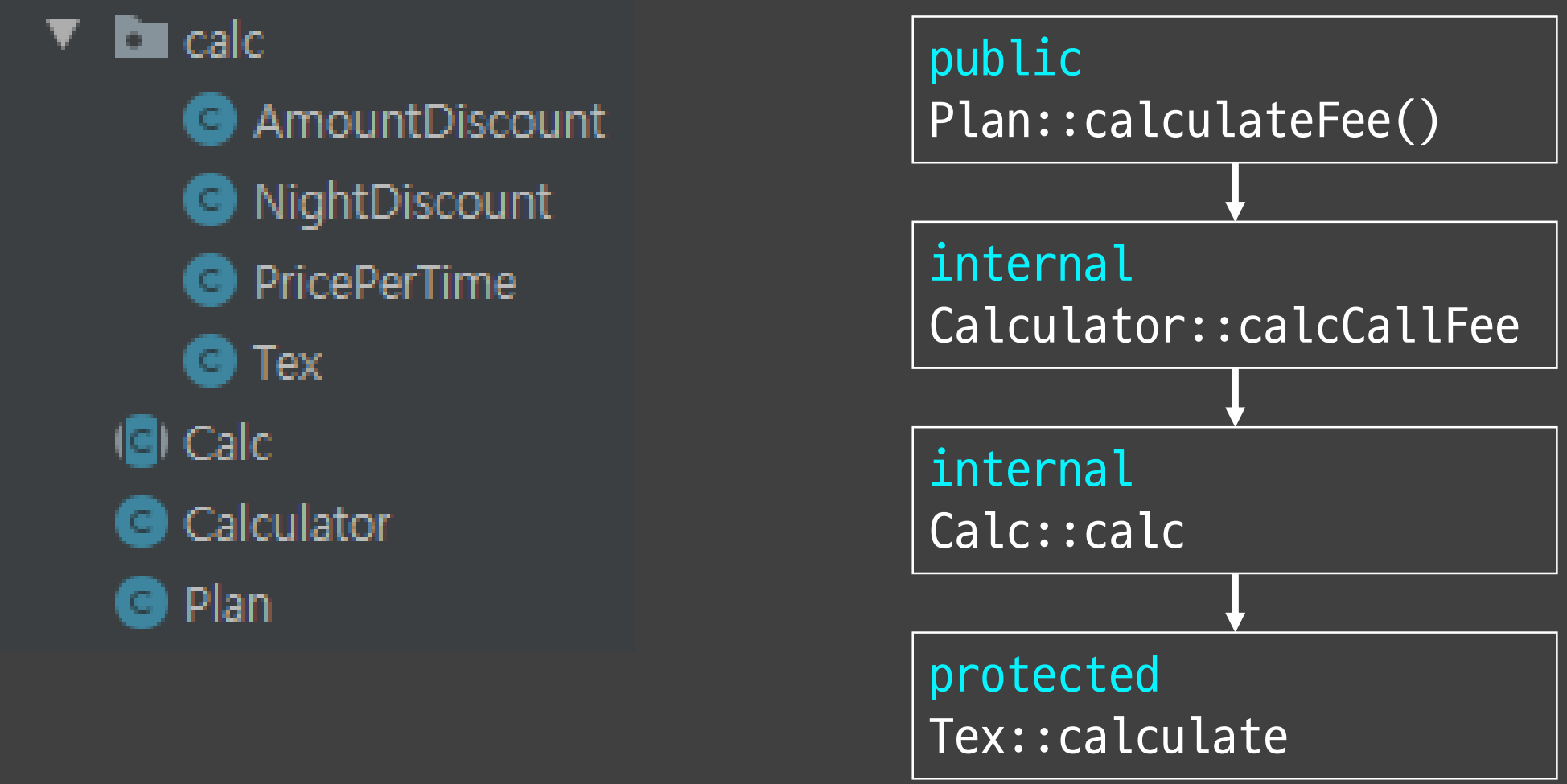

```
public
Plan::calculateFee()
```

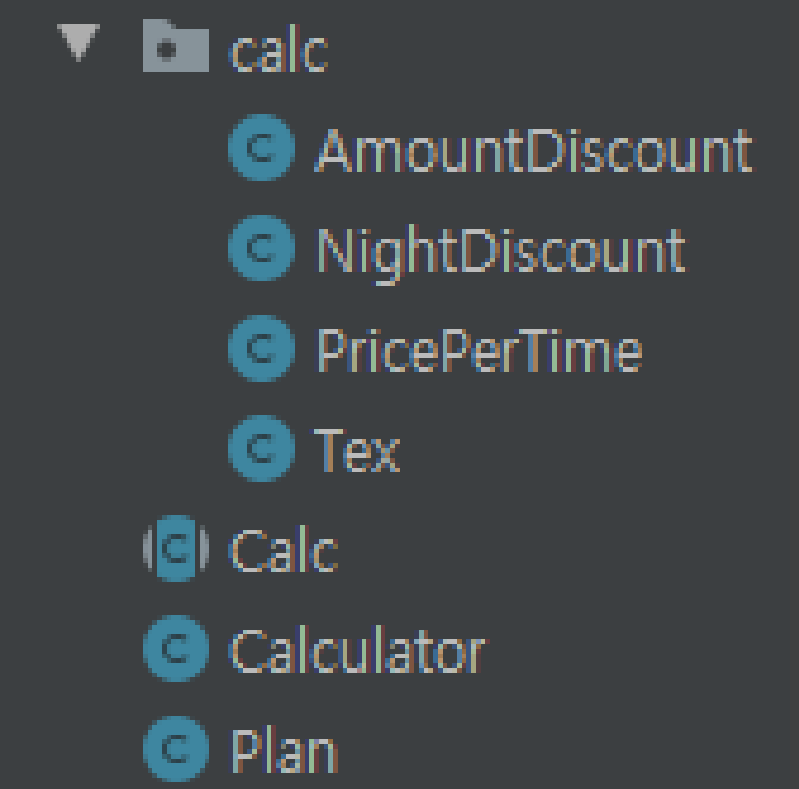
↓

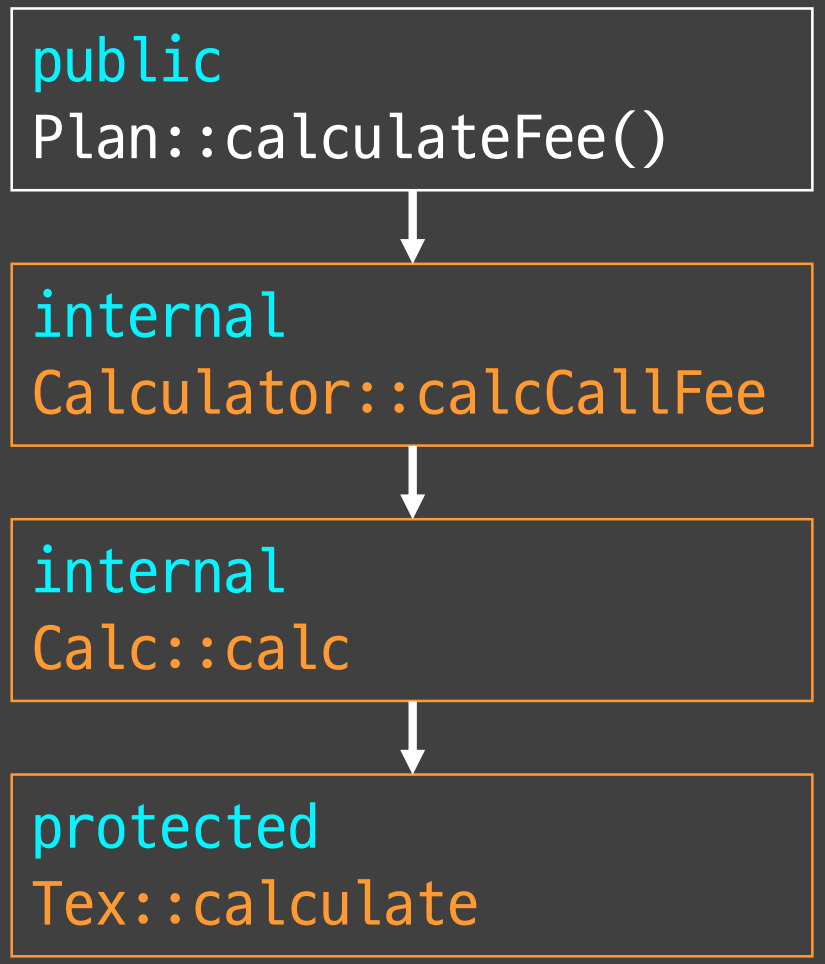
```
internal
Calculator::calcCallFee
```


calc
AmountDiscount
NightDiscount
PricePerTime
Tex
Calc
Calculator
Plan


public
Plan::calculateFee()
internal
Calculator::calcCallFee
internal
Calc::calc


calc
AmountDiscount
NightDiscount
PricePerTime
Tex
Calc
Calculator
Plan
public
Plan::calculateFee()
internal
Calculator::calcCallFee
internal
Calc::calc
protected
Tex::calculate


calc
AmountDiscount
NightDiscount
PricePerTime
Tex
Calc
Calculator
Plan


public
Plan::calculateFee()
internal
Calculator::calcCallFee
internal
Calc::calc
protected
Tex::calculate

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price
        this.price = price;
        this.second = second;
    }
    @Over
    final
        f
        )
        return result;
    }
}
```

```java
public class Calculator {
    ...
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new Ill...
        ...
    }
}
```

```java
public class Plan{
    ...
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price
        this.price = price;
        this.second = second;
    }
    @Over
    final
        f
        )
        return result;
    }
}
```

```java
public class Calculator {
    ...
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new Ill...
                                          );
    }
}
```

```java
public class Plan{
    ...
    public final Money calculateFee(){
        return calls.size() == 0 ? Money.ZERO : calc.calcCallFee(calls, Money.ZERO);
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price
        this.price = price;
        this.second = second;
    }
    @Override
    final protected Money calculate
        for(Call call:calls) result
             price.times((call.getD
        );
        return result;
    }
}
```

```java
public class Calculator {
    ...
    public final Money calcCallFee(Set<Call> calls, Money result){
        if(calcs.size() > 0) throw new Ill...
        for(Calc calc:calcs) result = calc.calc(calls, result);
        if(result.isLessThanOrEqual(Money.ZERO)) {
            throw new RuntimeException("calculate error");
        }
        return result;
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        this.price = price;
        this.second = second;
    }
    @Override
    final protected Money calculate(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
             price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        this.price = price;
        this.second = second;
    }
    @Override
    final protected Money calculate(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
             price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        this.price = price;
        this.second = second;
    }
    @Override
    final protected Money calculate(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
            price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        if(price == null || price.isLessThanOrEqual(Money.ZERO)){
            throw new IllegalArgumentException("invalid price");
        }
        if(second == null || second.compareTo(Duration.ZERO) <= 0){
            throw new IllegalArgumentException("invalid second");
        }
        this.price = price;
        this.second = second;
    }
    @Override
    final protected Money calculate(Set<Call> calls, Money result) {
        for(Call call:calls) result = result.plus(
            price.times((call.getDuration().getSeconds() / second.getSeconds()))
        );
        return result;
    }
}
```

```java
public class PricePerTime extends Calc {
    private final Money price;
    private final Duration second;
    public PricePerTime(Money price, Duration second){
        ...
    }
    @Override
    final protected Money calculate(Set<Call> calls, Money result) {
        Money sum = Money.ZERO;
        for(Call call:calls){
            Money r = price.times((call.getDuration().getSeconds() / second.getSeconds()));
            if(r.isLessThanOrEqual(Money.ZERO)){
                throw new RuntimeException("calculate error");
            }
            sum = sum.plus(r);
        }
        return result.plus(sum);
    }
}
```