1) before

```
class 클라이언트 {

    public void run() {
        세금계산 tax = new 세금계산();
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }

    public void doSomething2(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }
}

class 세금계산 {
    public int 계산1() {
            ... 긴 계산 절차 ...
        }

        public int 계산2() [
            ... 긴 계산 절차 ...
        }
}
```

2) 클라이언트에 구현

```
class 클라이언트 {
        권한관리 authority = new 권한관리();

    public void run() {
        세금계산 tax = new 세금계산();
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산 tax) {
        if (authority.권한조회())
            tax.계산1();
        else
            throw new Exception("권한이 없습니다");

        if (authority.권한조회())
            tax.계산2();
        else
            throw new Exception("권한이 없습니다");
    }

    public void doSomething2(세금계산 tax) {
        if (authority.권한조회())
            tax.계산1();
        else
            throw new Exception("권한이 없습니다");

        if (authority.권한조회())
            tax.계산2();
        else
            throw new Exception("권한이 없습니다");
    }
}

class 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}

class 권한관리 {
    public boolean 권한조회() {
       ...조회...
    }
}
```

3) 서비스 클래스에 구현

```
class 클라이언트 {

    public void run() {
        세금계산 tax = new 세금계산();
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }

    public void doSomething2(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }
}

class 세금계산 {
    권한관리 authority = new 권한관리();

    public int 계산1() {
        if (!authority.권한조회())
            throw new Exception("권한이 없습니다");
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        if (!authority.권한조회())
            throw new Exception("권한이 없습니다");
        ... 긴 계산 절차 ...
    }
}

class 권한관리 {
    public boolean 권한조회() {
       ...조회...
    }
}
```

4) 권한 정책 객체 분리

```
class 클라이언트 {

    public void run() {
        세금계산_권한통제 tax = new 세금계산_권한통제();
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산_권한통제 tax) {
        tax.계산1();
        tax.계산2();
    }

    public void doSomething2(세금계산_권한통제 tax) {
        tax.계산1();
        tax.계산2();
    }
}

class 세금계산_권한통제 {
    세금계산 tax = new 세금계산();
    권한관리 authority = new 권한관리();

    public int 계산1() {
        if (!authority.권한조회())
            throw new Exception("권한이 없습니다");
        return tax.계산1();
    }

    public int 계산2() [
        if (!authority.권한조회())
            throw new Exception("권한이 없습니다");
        return tax.계산2();
    }
}

class 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}

class 권한관리 {
    public boolean 권한조회() {
       ...조회...
    }
}
```

5) 다형성 구현 #1

클라이언트의 세금계산 메소드 호출을 다형성 호출로 개선

```
class 클라이언트 {

    public void run() {
        세금계산 tax = new 세금계산_권한통제();
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }

    public void doSomething2(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }
}

interface 세금계산 {
    int 계산1();
    int 계산2();
}

class 세금계산_권한통제 implements 세금계산 {
    세금계산A tax = new 세금계산A();
    권한관리 authority = new 권한관리();

    public int 계산1() {
        if (!authority.권한조회())
            throw new Exception("권한이 없습니다");
        return tax.계산1();
    }

    public int 계산2() [
        if (!authority.권한조회())
            throw new Exception("권한이 없습니다");
        return tax.계산2();
    }
}

class 세금계산A  implements 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}

class 세금계산B  implements 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }
```

```
    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}
```

6) 다형성 구현 #2

세금계산_권한통제 클래스의 세금계산 메소드 호출을 다형성 호출로 개선

```
class 클라이언트 {

    public void run() {
        세금계산  tax = new 세금계산_권한통제A();
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }

    public void doSomething2(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }
}

interface 세금계산 {
    int 계산1();
    int 계산2();
}

class 세금계산_권한통제A implements 세금계산 {
    세금계산 tax = new 세금계산A();

    public int 계산1() {
        ... 권한정책A...
        return tax.계산1();
    }

    public int 계산2() [
        ... 권한정책B...
        return tax.계산2();
    }
}

class 세금계산_권한통제B implements 세금계산 {
    세금계산 tax = new 세금계산A();

    public int 계산1() {
        ... 권한정책A...
        return tax.계산1();
    }

    public int 계산2() [
        ... 권한정책B...
        return tax.계산2();
    }
}

class 세금계산A  implements 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }
```

```
    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}

class 세금계산B  implements 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}
```

7) Proxy 패턴

```
class 클라이언트 {
    세금계산  tax

    public 클라이언트(세금계산  tax) {
        this.tax = tax;
    }

    public void run() {
        doSomething1(tax);
        doSomething2(tax);
    }

    public void doSomething1(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }

    public void doSomething2(세금계산 tax) {
        tax.계산1();
        tax.계산2();
    }
}

interface 세금계산 {
    int 계산1();
    int 계산2();
}

class 세금계산A  implements 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}

class 세금계산B  implements 세금계산 {
    public int 계산1() {
        ... 긴 계산 절차 ...
    }

    public int 계산2() [
        ... 긴 계산 절차 ...
    }
}

class 세금계산_권한통제A implements 세금계산 {
    세금계산 tax;

    public 세금계산_권한통제A(세금계산 tax) {
        this.tax = tax;
    }

    public int 계산1() {
        ... 권한조회A...
```

```
        return tax.계산1();
    }

    public int 계산2() [
        ... 권한조회A...
        return tax.계산2();
    }
}

class 세금계산_권한통제B implements 세금계산 {
    세금계산 tax;

    public 세금계산_권한통제B(세금계산 tax) {
        this.tax = tax;
    }

    public int 계산1() {
        ... 권한조회B...
        return tax.계산1();
    }

    public int 계산2() [
        ... 권한조회B...
        return tax.계산2();
    }
}


///// 객체 조립 사례
클라이언트 client1 = new 클라이언트(new 세금계산A());
client1.run();

클라이언트 client2 = new 클라이언트(new 세금계산_권한통제A(new 세금계산B()));
client2.run();

클라이언트 client3 = new 클라이언트(new 세금계산_권한통제B(new 세금계산B()));
client3.run();
```

클라이언트 코드는 세금계산 인터페이스만 참조한다.
   아래 계층의 클래스를 참조하지 않는다.
   권한통제 클래스나 세금계산 클래스를 참조하지 않는다.

권한통제 클래스는 세금계산 인터페이스만 참조한다.
   아래 계층의 클래스를 참조하지 않는다.
   세금계산 클래스를 참조하지 않는다.