

쉽게 익히는 클린코드

이번주 컨텐츠는?

- ~냄새) 추측성 일반화, 임시필드, 메세지 체인
- *리팩토링) 죽은 코드 제거, 특이 케이스 추가, 위임 숨기기

~냄새. 추측성 일반화

Speculative Generality

추후 이런 저런 기능이 생길 것을 예상하여, 필요로 할만한 기능을 만들었지만 결국에는 쓰이지 않는 코드가 발생한 경우

XP의 YAGNI(You aren't gonna need it) 원칙

[사용 가능한 리펙토링 기술]

계층 합치기(*Collapse Hierarchy*)

함수 인라인(*Inline Function*)

클래스 인라인(*Inline Class*)

함수 선언 변경하기(*Change Function Declaration*)

죽은 코드 제거하기(*Remove Dead Code*)

*리팩토링. 죽은 코드 제거하기



사용하지 않는 코드가 애플리케이션 성능이나 기능에 영향을 끼치지는 않는다.

하지만, 해당 소프트웨어가 어떻게 동작하는지 이해하려는 사람들에게는 꽤 고통을 줄 수 있다.

실제로 나중에 필요해질 코드라 하더라도 쓰이지 않는 코드라면(주석으로 남기는게 아닌) 제거해야 한다.

- 필요한 경우 형상관리 시스템으로 복원할 수 있다.

```
if(false) {  
    doSomethingThatUsedToMatter();  
}
```



~냄새. 임시 필드

Temporary Field

클래스에 있는 어떤 필드가 특정한 경우에만 값을 갖는 경우

어떤 객체의 필드가 "특정한 경우에만" 값을 가진다는 것을 이해하는 것은 일반적으로 예상하지 못하기 때문에 이해하기 어렵다.

[사용 가능한 리펙토링 기술]

클래스 추출하기(*Extract Class*)

함수 옮기기(*Move Function*)

특이 케이스 추가하기(*Introduce Special Case*)



*리팩토링. 특이 케이스 추가하기

어떤 필드의 특정한 값에 따라 동일하게 동작하는 코드가 반복적으로 나타난다면, 해당 필드를 감싸는 "특별한 케이스"를 만들어 해당 조건을 표현할 수 있다.

이러한 매커니즘을 "특이 케이스 패턴"이라고 부르고, "Null Object 패턴"이러한 패턴의 특수한 형태라고 볼 수 있다.

```
if(aCustomer == null)
    customerName =
        "occupant";
else
    customerName =
        customer.getName();
```

```
class UnknownCustomer
    extends Customer {
    String getName() {
        return "occupant";
    }
}
customerName =
    customer.getName();
```

~냄새. 메시지 체인

Message Chains

레퍼런스를 따라 계속해서 메소드 호출이 이어지는 코드

- 예) `this.member.getCredit().getLevel().getDescription()`

유지보수를 위해 해당 코드의 클라이언트가 코드 체인을 모두 이해해야 한다.

체인 중 일부가 변경된다면 클라이언트의 코드도 변경해야 한다.

[사용 가능한 리펙토링 기술]

위임 숨기기(*Hide Delegate*)

함수 추출하기(*Extract Function*) > 함수 옮기기(*Move Function*)

*리팩토링. 위임 숨기기

캡슐화란 어떤 모듈이 시스템의 다른 모듈을 최소한으로 알아야 한다는 것이다. 그래야 어떤 모듈을 변경할 때, 최소한의 모듈만 그 변경에 영향을 받을 것이고, 그래야 무언가를 변경하기 쉽다.

처음 객체 지향에서 캡슐화를 배울 때 필드를 메소드로 숨기는 것이라 배우지만, 메소드 호출도 숨길 수 있다.

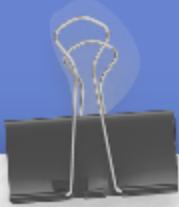
- 예) `person.department().manager() -> person.getManager()`
- 예제에서 이전 코드는 `department`를 통해 `Manager`에 접근 할 수 있지만, `getManager()`를 통해 위임을 숨긴다면 `person`의 `getManager()`만 알면 되며, 향후 `getManager()`의 내부 구현이 바뀌더라도 기존 호출 코드는 그대로 사용이 가능하다.

*리팩토링. 위임 숨기기

```
Person manager = aPerson.getDepartment().getManager();  
class Person {  
    getDepartment() { ... }  
}  
class Department {  
    getManager() { ... }  
}
```



```
Person manager = aPerson.getManager();  
class Person {  
    Person getManager() {  
        return getDepartment().getManager();  
    }  
    getDepartment() { ... }  
}  
class Department {  
    getManager() { ... }  
}
```



클린코드 끌팁을 놓치고 있다면?

클린코드 팁을
구독하세요:)

구독하기?!