Refactoring 04 [뒤엉킨 변경]

코딩으로 학습하는 리팩토링

백기선

2023.02.09

Smell #7 뒤엉킨 변경 (Divergent Change)

- 소프트웨어는 응집도는 높아야 하고 결합도는 낮아야한다.
- 즉, 관련된 것이 같이 모여있는 (응집도) 가 높게 , 서로의 의존이 (결합도) 가 낮게!
- 모듈화가 잘 되어있어야 한다.
- 한모듈에서 하나의 책임만 지게하는 것이 좋다.

리팩토링 - 단계 쪼개기

- split Phase

- 서로 다른 일을 하는 코드를 각기 다른 모듈로 분리한다. 즉, 한가지 일만 하는 메소드로 만들어준다.
- 큰 흐름을 (프로세스)를 가지고 있는 메소드가 있다면 나눠주는것
- 예) 컴파일러 : 텍스트 읽어오기 , 실행 가능한 형태로 변경 이런 식으로 두개로 나눠주는것.
- 중간 데이터를 만들어서 단계를 구분하고 매개변수를 줄일 수 있다.

리팩토링 - 함수 옮기기

- Move Function

- 모듈화가 잘 되어있을수록 최소한의 지식만으로 프로그램을 변경할 수 있다.
- 함수가 사용하는 데이터가 다른 문맥을 더 파악할때, 함수가 다른 클라이언트에서도 필요로하는 경우. 함수를 옮기는걸 고려해 볼 만 하다.

리팩토링 - 클래스 추출하기

- Extract Class

- 클래스가 다루는 책임(Responsibility)이 많아 질 수록 클래스가 점점 커진다.
- 클래스를 쪼개는 기준
- 데이터나 메소드 중 일부가 매우 밀접한 관련이 있는 경우
- 일부 데이터가 대부분 같이 바뀌는 경우
- 데이터 또는 메소드 중 일부를 삭제한다면 어떻게 될 것인가?
- 하위 클래스를 만들어 책임을 분산 시킬 수 도 있다.

Smell #8 산탄총 수술 (Shotgun Surgery)

어떤 한 변경 사항이 생겼을 때 여러 곳을 손봐야 하는 상황이 있다면 모아야 한다!

클래스의 메소드나 필드가 불필요한 결합도가 높거나 여러곳에 흩어져 있는 경우다.

리팩토링 - 필드 옮기기

- Move Field

- 좋은 데이터 구조를 가지고 있다면, 해당 데이터에 기반한 어떤 행위를 코드로(메소드나 함수) 옮기는 것도 간편하고 단순해 진다.
- 어떤 데이터가 항상 같이 옮겨지거나 , 어떤 레코드(객체)를 변경할 때 다른 레코드를 변경해야 한다거나, 여러 레코드에 동일한 필드를 수정해야 하는 경우

리팩토링 - 함수 인라인

- Inline Function

- 함수 추출하기의 반대에 해당하는 리팩토링
- 잘못 추출 했을 때 원 상태로 되돌릴때 사용
- 단순히 메소드 호출을 감싸는 우회형 매소드라면 인라인으로 없앨 수 있다.

리팩토링 - 클래스 인라인

- Inline Class

- 클래스를 한 곳으로 모으는 것
- 리팩토링을 하면서 책임들을 한 곳으로 다시 모으기 위해 진행
- 두개의 클래스를 여러 클래스로 나눌때 한 곳을 모은후에 분배하는 경우도 있다.

Smell #9 기능 편애(Feature Envy)

한 기능에 욕심을 내다보면 필요 이상의 기능이나 데이터를 가지고 있는 경우가 생기가 된다.

예) 다른 객체의 getter를 여러개 사용하는 메소드

- 만약 여러 모듈을 참조하고 있다면 가장 많이 참조되고 있는 곳으로 기능을 옮기거나 함수를 여러개로 쪼개서 각 모듈로 분산 시킬 수도 있다.