2. 리팩터링 원칙 (챕터2)

Status	→Next

1. 리팩토링이란 무엇인가?

결과 / 행동 변경 없이 코드의 구조를 재조정하는 것

소프트웨어 기능을 보존하면서 설계, 구조 및 구현을 개선하는 것

리팩토링의 목표

- 복잡성 감소
- 가독성 향상
- 유지 보수성 개선
- 확장성 증가

→더 단순하고, 깔끔하고 표현력이 뛰어난 코드, 내부 아키텍처/객체 모델을 만듦

금지

- 기능 변경/추가
- 버그 수정
- 성능 개선
- 버전 업데이트

리팩토링 정의

리팩토링: [명사] 소프트웨어의 겉보기 동작은 그대로 유지한 채, 코드를 이해하고 수정하기 수비도록 내부 구조를 변경하는 기법

리팩토링(하다): [동사] 소프트웨어의 겉보기 동작은 그대로 유지한 채, 여러가지 리팩토링 기법을 적용해서 소프트웨어를 재구성하다

한마디로, `코드를 깨끗하게 만드는 작업`

2. 리팩토링이 왜 필요할까?

왜 코드를 개선하는 걸까?

- 개발 초기 단계부터 완벽한 코드/시스템 설계의 어려움
- 프로그램의 요구사항은 꾸준히 변경됨 (기능 추가/변경)
- 더럽고 복잡한 코드는 이해하기 어려움
- 예상하지 못한 에러가 발생하기 쉬움
- 복잡한 코드의 유지보수는 어려움 (수정 시간이 오래걸림)

리팩토링을 하는 이유

- 소프트웨어 설계가 좋아짐
  - 모든 코드가 언제나 고유한 일을 수행함을 보장함
  - 이해해야할 코드의 양이 작고, 실수 없이 수정할 수 있음
- 소프트웨어를 이해하기 쉬워짐
  - 코드의 목적이 잘 드러나게, 의도를 더 명확하게 표현
  - 코드가 잘 읽힘, 가독성이 좋아짐
- 버그를 쉽게 찾을 수 있음
  - 코드가 하는 일을 깊이 파악할 수 있음
- 프로그래밍 속도를 높일 수 있음
  - 내부 설계가 잘 된 소프트웨어는 새로운 기능을 추가할 지점을 빠르게 찾음
  - 작은 일부의 코드만 이해하여 빠르게 수정이 가능
  - 디버깅이 쉬움
  - 새로운 기능을 손쉽게 추가

3. 리팩토링 어떻게 해야하나?

리팩토링 과정

더러운 코드 존재한다면 우선 기존 프로그램의 기능과 동작을 유지할 수 있도록 Test Code(함수, 특정한 기능, UI, 성능, API 스펙)를 작성한다. 그 후 코드의 나쁜 냄새에 따라 여러 리팩토링 기법을 점진적으로 적용해나가면 된다.

중요한 점은 리팩토링 과정에서 버그를 발견하더라도 중간에 수정하지 않고 리팩토링을 끝낸 후 버그를 수정해야한다.

4. 언제 리팩토링 해야 하나?

수시로!

1. 프로젝트 시작단계

- 좋은 디자인 패턴으로 코드를 깔끔하게 작성
  - 기능 구현을 위한 코드 작성 → 테스트 코드 작성
  - 테스트 코드를 작성하기 위해 리팩토링을 해나가야 하는 경우도 있음
  - 3의 법칙 : 비슷한 일을 세 번째하게 되면 리팩토링한다.
  - 기능을 만들고, 테스트 코드를 작성한 후에 코드 리뷰를 받으며 코드를 이해하기 쉽게 만들도록 노력한다. (좋은 문서화)
  - 기능을 추가해야 한다면 기존의 코드에 테스트 코드가 있기 때문에 리팩토링을 통해 재사용성과 모듈성을 높일 수 있다.

2. 프로젝트 유지보수 단계

- 버그 수정

    - 버그를 검증할 수 있는 테스트 코드 작성 →코드를 이해하기 쉽게, 변경하기 쉽게 변경 (리팩토링) → 버그 수정
- 기능 추가, 디펜던시 마이그레이션
    - 기존의 기능들에 대한 테스트 확인 → 코드를 이해하기 쉽게, 변경하기 쉽게 변경 (리팩토링) → 기능 추가

3. 오래된 프로젝트

- 버그 수정 및 기능추가시에만
    - 수정이 필요한 모듈/코드 한정적으로 테스트 추가 → 리팩토링 →코드 수정 또는 기능 추가
    - 때로는 새로운 코드를 작성하는 것이 빠름

5. 리팩토링의 중요한 포인트

`무결점 클린코드, 완벽한 설계는 존재하지 않는다`를 인정!

야그니

You Aint't Gonna Need It

처음부터 코드를 깨끗하고 변경이 쉽고, 유지보수가 용이하도록 작성하는 것도 중요하지만 당장 필요하지 않는 것에 사로잡혀서 필요하지 않는 기능에 집착해서 코드를 복잡하게 만들거나 사용하지 않는 기능을 만들거나 지나치게 미래지향적으로 소프트웨어를 설계하지 않도록 주의해야 한다.