Refactoring 02 [긴 함수, 긴 매개변수 목록]

코딩으로 학습하는 리팩토링

백기선

2022.11.18

Smell #3 긴 함수 (Long Function)

- 짧은 함수는 많은 문맥 전환을 필요로 한다 vs 함수가 길 수록 더 이해할 수 없다.
- 최근에는 서브루틴 호출로 오버헤드가 발생했지만 요즘에는 콜 스택오버헤드까지 고민할 필요는 없다.

주석 → 짧은 코드 → 코드 이름으로 변경해보자.

대부분은 **'함수 추출하기 (Extract Function)'** 으로 해결가능하다**.**

만약 함수를 추출하는데 매개변수가 많아지면 리팩토링 기술을 사용해 사용가능.

리팩토링 - 임시 변수를 질의 함수로 바꾸기

- Replace Temp with Query

임시 변수를 함수로 추출해서 코드에서 동일한 연산을 줄일 수 있다.

임시 변수가 기존의 변수에서 추출이 가능하다면 메소드(쿼리 메소드)로 변경하여 사용하면 매개변수를 줄일 수 있다.

리팩토링 - 매개변수 객체 만들기

- Introduce Parameter Object

같은 매개변수들이 여러 메소드에 걸쳐 나타나나다면 그 매개변수들을 묶은 자료구조를 만들 수있다.

매개변수가 데이터간의 관계르 보다 명시적으로 나타낼 수있고

매개변수의 수를 줄일 수도 있고

도메인을 이해하는데 중요한 역할을 하는 클래스로 발전할 수도 있다.

리팩토링 - 객체 통째로 넘기기

- Preserve Whole Object

파라미터를 넘길때 한 객체에서 추출할 수 있는 요소들이면 매개변수 목록을 줄 일수있다.

다만 의존성을 고려해야한다. 만약 이것을 적용할때 매소드의 위치가 적절하지 않을 수도 있음을 인식해야 한다.

리팩토링 - 함수를 명령으로 바꾸기

- Replace Function with Command

함수를 독립적인 객체인 Command로 만들어 사용할 수 있다.

커맨드 패턴을 적용하면 undo 기능을 만들거나 복잡한 기능 구현하는 메소드를 구현, 상속이나 템플릿 사용등 여러 장점을 얻을 수있다.

리팩토링 - 조건문 분해하기

- Decompose Conditional

조건에 따라 달라지면 긴 함수가 만들어지니 이것을 리팩토링

조건 과 액션 모두 의도를 표현해야 한다.

리팩토링 - 반복문 쪼개기

- Split Loop

하나의 반복문에서 여러 다른 작업을 하게된다면?

수정할 때마다 여러 작업을 모두 고려하며 코딩해야 한다.

반복문을 쪼개면 쉽게 이해하고 수정할 수 있다.

성능 문제를 야기할 수 있지만 리팩토링은 성능 최적화와 별개 작업.

리팩토링 - 조건문을 다형성으로 바꾸기

- Replace Conditional with Polymorphism

조건문을 다형성을 사용해서 교체하는 것. 스위치나 이프 문을 조건이 다를때마다 각각 행동하게 하고 싶을 때 다형성을 사용한 클래스를 만들거나 바꾸게 도록 작성하는 것.

다른 여러 클래스로 분해하여 사용가능

Smell #4 긴 매개변수 목록 (Long Parameter List)

어떤 함수에 매개변수가 많을수록 함수의 역할을 이해하기 어려워진다.

- 매개변수에서 다른 매개변수를 뽑아낼 수 있다면 매개변수 질의 함수로 바꾸기 사용
- 동일 객체에서 가져올 수있다면 객체를 통째로 넘기는것을 고려가능
- 매개변수들이 같은 셋으로 넘어간다면 매개변수 객체 만들기를 사용.
- 매개변수가 플래그로 사용이 된다면 플래그 인수 제거하기 사용
- 일부 함수가 일부 매개변수를 공통으로 사용한다면 여러 함수를 클래스로 묶을 수 있다.

리팩토링 - 매개변수를 질의 함수로 바꾸기

- Replace Parameter with Query

함수의 매개변수의 목록은 함수의 다양성을 대변하고 이해하는 중요한 요소가 된다.

→ 이해하는 요소가 짧고 명확 할 수록 좋다.

한 매개변수가 다른 매개변수를 통해 알아낼 수 있다면 '중복 매개변수' 라고 생각 할 수있다.

매개변수를 전달하는 것은 함수를 호출하는 쪽에서 호출하는 쪽의 책임을 줄이고 함수 내부에서 책임지도록 노력하는것.

다만, 새로운 의존성이 생기거나 하는 상황을 염두해 둘 필요가 있다.

함수의 선언을 변경하거나 변수를 질의 함수로 바꾸는 등으로 리팩토링을 진행.

리팩토링 - 플래그 인수 제거하기

- Remove Flag Argument

플래그는 보통 함수 매개변수로 전달해서, 함수 내부의 로직을 분기하는데 사용되는데

플래그를 사용하게 되면 플래그에 따른 차이를 외부에서 확인이 어렵다.

플래그가 많으면 → 너무 많은 역할이 들어있다.

플래그가 단 하나라면 → 외부에서 파악하기 힘들어 진다.

그러므로 조건문을 분해하여 사용하기로 사용하는 것이 좋다.

리팩토링 - 여러 함수를 클래스로 묶기

- Combine Functions into Class

관련있는 매개변수들이 여러 함수에서 자주 사용된다면

해당 메소드를 모아서 클래스를 만들수 있다. (매개변수 목록들을 필드로 만들어 사용)

Smell #5 전역 데이터 **(Global Data)**

전역 데이터는 아무곳에서나 변경될 수 있는 문제가 있다.

상수라면 문제 발생 여지가 줄어들지만, 상수가 아니라면 어디에서 변경 되었는지 어느 부분에서 문제가 생겼는지 찾기 힘들어진다.

- 변수를 캡슐화 하는 리팩토링을 사용해 접근을 제어하거나 어디서 사용하는지 파악하기 쉽게 만들 수 있다.

리팩토링 - 변수 캡슐화하기

- Encapsulate Variable

변수를 변경하기 보다는 메소드를 변경하는 것이 쉽게 된다. 하지만 변수는 모두 한번에 변경해야 하기 때문에 문제가 생긴다.

그러므로 변수에 접근하는 것을 메소드로 감싸주어 사용하는 곳에 대한 메소드를 통해 변수에 접근하도록 하는 것이 좋다.