Clean Code
[3장 함수]

~ 함수 인수

Main Reader : 위승빈

이슈

수경

[Switch 문] - 추상팩토리에서 파생 클래스, 다형성 등을 사용하면 해결?

예제를 유심히 볼 수 있던 이슈

진홍

[함수] - 3-2 예제가 3-1 코드와 정말 같은 동작을 하는지?

책을 맹신하면 안 된다는 교훈을 줬던 이슈

진홍

[함수] - '추상화 수준' 이라는 개념을 좀 더 명확하게 설명하자면 어떻게 설명해야 할까?

조사하면서 여러 정보를 접하고 생각을 정리할 수 있게 해준 이슈

함수

1. 함수는 어떻게 써야할까?
2. 작게 만들어라!
3. 한 가지만 해라!
4. 함수 당 추상화 수준은 하나로!
5. Switch 문
6. 서술적인 이름을 사용하라!
7. 함수 인수

함수는 어떻게 써야할까?

함수는 프로그램의 가장 기본적인 단위이다.

함수는 정보를 파악하기 쉬워야 한다.

함수는 읽기 쉽고 이해하기 쉬워야한다.

함수는 의도를 분명히 표현해야 한다.

함수는 처음 읽는 사람도 프로그램 내부를 직관적으로 파악할 수 있어야 한다.

작게 만들어라!

함수를 만드는 첫째 규칙은 '작게!'다. 함수를 만드는 둘째 규칙은 '더 작게!'다.

각 함수는 명백해야 하며, 이야기 하나를 표현할 수 있어야 한다.

모든 함수가 2~4줄 만큼 짧으면 좋다.

함수에서 들여쓰기 수준은 1단이나 2단을 넘어서면 안 된다.

작게 만들면 함수는 읽고 이해하기 쉬워진다.

한 가지만 해라!

함수는 한 가지를 해야 한다. 그 한 가지를 잘 해야 한다. 그 한 가지만을 해야 한다.

지정된 함수 이름 아래에서 추상화 수준이 하나인 단계만 수행한다면 그 함수는 한 가지 작업만 하는 것이다.

단순히 다른 표현이 아니라 의미 있는 이름으로 다른 함수를 추출할 수 있다면 그 함수는 여러 작업을 하는 셈이다.

한 가지 작업만 하는 함수는 자연스럽게 섹션으로 나누기 어렵다.

함수 당 추상화 수준은 하나로!

함수가 확실히 '한 가지' 작업만 하려면 함수 내 모든 문장의 추상화 수준이 동일해야 한다.

한 함수 내에 추상화 수준을 섞으면 코드를 읽는 사람이 헷갈린다.

각 함수는 일정한 추상화 수준을 유지하면서 한 가지 작업만 해야 한다.

내려가기 규칙 (함수 당 추상화 수준은 하나로!)

코드는 위에서 아래로 이야기처럼 읽혀야 좋다.

한 함수 다음에는 추상화 수준이 한 단계 낮은 함수가 온다.

즉, 위에서 아래로 프로그램을 읽으면 함수 추상화 수준이 한 번에 한 단계씩 낮아진다.

이를 내려가기 규칙이라고 부르기로 하자.

문단을 읽듯이 프로그램도 쭉 읽혀야 한다.

Switch 문

본질적으로 switch 문은 N가지를 처리한다.
switch문은 작게 만들기 어렵고, 한 가지 작업만 하는 switch문을 만들기 어렵다.

다형성을 이용하여 각 switch 문을 저차원 클래스에 숨기고 반복하지 않도록 한다.

저자는 다형적 객체를 생성하는 코드 안에서는 switch 문을 사용할 수 있다고 한다.
상속 관계로 switch 문을 숨긴 후에는 절대로 다른 코드에 노출하지 않아야 한다.

switch 문은 불가피하게 사용되는 상황도 더러 생긴다.

다형성을 이용해 Switch 문 숨기기 (Switch 문)

목록 3-4 : switch 문을 사용해 Employee 유형을 판별하여 다른 값을 계산한다.

목록 3-5 : 추상 팩토리 안에서 switch 문을 사용해 Employee 파생 클래스의 인스턴스를 생성한다.

3-5 에서는 switch 문을 숨겼다. 다른 곳에서는 알 필요가 없어졌다.

급여를 어떻게 계산할지는 각 파생 클래스에서 구현하고, 함수는 인터페이스를 거쳐서 호출된다. 이제 다른 값을 계산하기 위해서 switch 문을 수정할 필요가 없어졌다.

다형적 객체를 생성하기 위해서만 switch 문이 쓰인다.

서술적인 이름을 사용하라!

“코드를 읽으면서 짐작했던 기능을 각 루틴이 그대로 수행한다면 깨끗한 코드라 불러도 되겠다.” – 워드

함수가 작고 단순할수록 서술적인 이름을 고르기도 쉬워진다.

서술적인 이름을 사용하면 코드를 개선하기 쉬워진다.

이름이 길어도 괜찮다. 이름을 정하느라 시간을 들여도 괜찮다.

함수 이름을 정할 때는 여러 단어가 쉽게 읽히는 명명법을 사용한다.

여러 단어를 사용해 함수 기능을 잘 표현하는 이름을 선택한다.

이름을 붙일 때는 일관성이 있어야 한다. 모듈 내에서 함수 이름은 같은 문구, 명사, 동사를 사용한다.

함수 인수

함수에서 이상적인 인수 개수는 0개다. 다음은 1개고, 다음은 2개다. 3개는 가능한 피하는 편이 좋다. 4개 이상은 특별한 이유가 필요하다. 특별한 이유가 있어도 사용하면 안 된다.

인수는 개념을 이해하기 어렵게 만든다. 인수 개수가 늘어날 수록 더 이해하기 어렵다.

출력 인수는 입력 인수보다 이해하기 어렵다.

최선은 입력 인수가 없는 경우다.

함수 인수

많이 쓰는 단항 형식

- **인수에 질문을 던지는 경우**
- **인수를 뭔가로 변환해 결과를 반환하는 경우**
- **입력 인수로 시스템 상태를 바꾸는 이벤트의 경우**

위의 경우가 아니라면 단항 함수는 가급적 피한다.

입력 인수를 변환하는 함수라면 변환 결과는 반환값으로 돌려준다.

플래그 인수

플래그 인수는 추하다.

함수가 한꺼번에 여러 가지를 처리한다고 대놓고 공표한다.

true인 경우, false인 경우인 함수를 따로 나누는 게 낫다.

함수 인수

이항 함수

인수 2개가 한 값을 표현하거나, 자연적인 순서인 경우 이항 함수가 적절하다.

불가피한 경우로 이항 함수를 사용해야 할 때도 있다. 하지만 **단항 함수로 바꾸도록 애써야 한다.**

삼항 함수

상함 함수를 만들 때는 신중히 고려하자.

여러 사람이 더러 알 수 있는 중요한 사항의 경우는 삼항 함수를 사용할 수도 있다.

함수 인수

인수 객체

인수가 여러개 필요하다면 일부를 독자적인 클래스 변수로 선언해보자.

변수를 묶어 넘기려면 이름을 붙여야 하므로 결국 눈속임을 위한 객체가 아니라, 개념을 표현하게 되는 것이다.

인수 목록

인수 개수가 가변적인 함수도 필요할 때가 있다.

가변 인수를 전부 동등하게 취급하면 List 형 인수 하나로 취급할 수 있다. 가변 인수들을 묶어 하나의 인수로 생각할 수 있는 것이다.

가변 인수를 취하는 함수는 단항, 이항, 삼항 함수로 취급될 수 있다. 하지만 이를 넘어서는 인수를 사용할 경우에는 문제가 있다.

함수 인수

동사와 키워드

함수의 의도나 인수의 순서와 의도를 제대로 표현하려면 좋은 함수 이름이 필수다.

단항 함수는 **함수와 인수가 동사/명사 쌍**을 이뤄야 한다.

함수 **이름에 키워드를 추가**하여 인수 순서를 기억할 필요 없도록 하자.

명확한 이름으로 최대한
작게 한 가지 일을 하는
각 함수들로 이야기를
표현해 봅시다!