Clean Code Chapter 2 Meaningful Name

1. Use Intention-Revealing Names(의도를 분명하게 밝혀라)

   변수나 함수 그리고 클래스 이름은 다음과 같은 질문에 모두 답해야 함.

   변수(혹은 함수나 클래스)의 존재의 이유

   변수(혹은 함수나 클래스)의 수행 기능

   변수(혹은 함수나 클래스)의 사용 방법

2. Avoid Disinformation(잘못된 정보를 피해라)

   잘못된 단서는 코드 의미를 흐림.

   널리 쓰이는 의미가 있는 단어를 다른 의미로 사용하면 안됨.

   흡사한 이름을 사용하지 않도록 주의

   유사한 개념은 유사한 표기법을 사용(일관성이 떨어지는 표기법은 그릇된 정보)

3. Make Meaningful Distinctions(의미 있게 구분하라)

   이름이 달라야 한다면 의미도 달라져야 한다.

   Noise word(불용어)는 중복임.

   읽는 사람이 차이를 알도록 이름을 지어야 함.

4. Use Pronounceable Names(발음하기 쉬운 이름을 사용하라)

   발음하기 어려운 이름은 토론하기도 어려움.

5. Use Searchable Names(검색하기 쉬운 이름을 사용하라)

   이름 길이는 범위 크기에 비례해야 한다.

   변수나 상수를 코드 여러 곳에서 사용한다면 검색하기 쉬운 이름이 바람직함.

6. Avoid Encoding(인코딩을 피하라)

   인코딩에 유형이나 범위 정보까지 넣으면 이름을 해독하기 어려움.

   인코딩한 이름은 발음하기 어려우며 오타가 생기기도 쉬움.

   A. Hungarian Notation

      타입을 인코딩할 필요가 없으므로 헝거리식 표기법이나 기타 인코딩 방식이 방해가 됨

B. Member Prefixes

멤버 변수에 접두어를 붙일 필요도 없음.

클래스와 함수는 접두어가 필요 없을 정도로 작아야 마땅함

C. Interfaces and Implementations

인코딩이 필요한 경우가 때로는 있음.

인터페이스 클래스 이름과 구현 클래스 이름 중 하나를 인코딩해야 한다면 구현 클래스 이름을 하는 것이 좋음.

7. Avoid Mental Mapping(자신의 기억력을 의존 하지 마라)

코드를 읽으면서 변수 이름을 자신이 아는 이름으로 변환해야 한다면 그 변수 이름은 바람직하지 못함.

명료함이 최고이다.

8. Class Name

클래스 이름과 객체 이름은 명사나 명사구가 적합

Manager, Processor, Data, Info 등과 같은 단어는 피하고 동사는 사용하지 않음.

9. Method Name

메서드 이름은 동사나 동사구가 적합

접근자, 변경자, 조건자는 get, set, is를 붙임

생성자를 중복 정의할 경우 정적 팩토리 메서듯를 사용함

메서드는 인수를 설명하는 이름을 사용한다.

생성자 사용을 제한하고자 하면 해당 생성자를 private으로 선언

10. Don’t Be Cute(기발한 이름은 피해라)

재미있는 이름보다는 명료한 이름을 선택

의도를 분명하고 솔직하게 표현

11. Pick One Word per Concept(한 개념에 한 단어를 사용하라)

추상적인 개념 하나에 단어 하나를 선택해 이를 고수

메서드 이름은 독자적이고 일관적이어야 한다.

12. Don’t Pun(말장난을 하지 마라)

    한 단어를 두가지 목적으로 사용하지 마라.

    다른 개념에 같은 단어를 사용한다면 그것은 말장난에 불과

13. Use Solution Domain Names(해법 영역에서 가져온 이름을 사용해라)

    모든 이름을 문제 영역에서 가져오는 것은 현명하지 못함.

    기술 개념에는 기술 이름이 가장 적합한 선택

14. Use Problem Domain Names(문제 영역에서 가져온 이름을 사용해라)

    적절한 프로그래머 용어가 없으면 문제 영역에서 이름을 가져옴.

    문제 영역과 해법 영역을 구분할 줄 알아야 함.

    문제 영역 개념과 관련이 깊은 경우 문제 영역에서 이름을 가져와야 함.

15. Add Meaningful Context(의미 있는 맥락을 추가하라)

    클래스, 함수, 이름 공간에 맥락을 부여

16. Don’t Add Gratuitous Context(불필요한 맥락을 없애라)

    일반적으로 짧은 이름이 긴 이름보다 좋다.(단, 의미가 분명한 경우에 한해서)

    이름에 불필요한 맥락을 추가하지 않도록 주의 해야함.

Clean Code Chapter 3 Functions

1. Small(작게 만들어라)

   함수가 작을수록 좋음.

   A. 블록과 들여쓰기

      If문/else문/while문 등에 들어가는 블록은 한 줄이어야 함.

      중첩 구조가 생길 만큼 함수가 커져서는 안 된다는 뜻

2. Do One Thing

   함수는 한 가지를 해야 한다. 그 한가지를 잘해야하고 그 한 가지만을 해야 한다.

   함수를 만드는 이유는 큰 개념을 다음 추상화 수준에서 여러 단계로 나눠 수행하기 위해서임.

A. Sections within Functions

함수가 section으로 나뉜다면 여러 작업을 한다는 증거임.

3. One Level of Abstraction per Function(함수 당 추상화 수준은 하나로)

함수가 확실히 한 가지 작업만 하려면 함수 내 모든 문장의 추상화 수준이 동일해야함.

함 함수 내 추상화 수준을 섞으면 근본 개념인지 세부사항인지 구분하기 어려움.

근본 개념과 세부 사항을 뒤섞기 시작하면 세부사항을 함수에 점점 더 추가함.

A. Reading Code from Top to Bottom: The Stepdown Rule(내려가기 규칙)

코드는 위에서 아래로 이야기처럼 읽혀야 조음.

위에서 아래로 프로그램을 읽으면 추상화 수준이 한번에 한단계씩 낮아짐.

핵심: 짧으면서도 한가지만 하는 함수

4. Switch Statement

Switch문은 작게 만들기 어려움. 본질적으로 switch 문은 N가지를 처리함.

switch 문을 완전히 피할 방법은 없지만 저차원 클래스에 숨기고 절대로 반복하지 않은 방법이 있는데 이는 다형성 polymorphism을 이용하는 것임.

5. Use Descriptive Names

좋음 이름을 지어줘야 함.

함수가 작고 단순할수록 서술적인 이름을 고르기도 쉬움.

길고 서술적인 이름이 짧고 어려운 이름보다 좋음.

서술적인 이름을 사용하면 개발자 머릿속에서도 설계가 뚜렷해지므로 코드를 개선하기 쉬워짐.

이름을 붙일 때는 일관성이 있어햐 함.

모듈 내에서 함수 이름은 같은 문구, 명사, 동사를 사용

6. Function Arguments

이상적인 인수 개수는 0개(무항) 그 다음이 1개(단항), 다음은 2개(이항), 3개(삼항)은 가능한 피하는 편이 좋고 4개 이상(다항)은 특별한 이유가 필요함. 특별한 이유가 있어도 사용하면 안 됨.

최선은 입력 인수가 없는 경우이며 차선은 입력 인수가 1개 뿐인 경우임,

A. Common Monadic Forms(많이 쓰는 단항 형식)

함수에 인수 1개를 넘기는 이유: 인수에 질문을 던지는 경우 or 인수를 뭔가 변환해 결과를 반환하는 경우

다소 드물게 사용하지만 그래도 유용한 단항 함수 형식이 이벤트임

이벤트 함수는 입력 인수만 있고 출력 인수는 없음.

프로그램은 함수 호출을 이벤트로 해석해 인력 인수로 시스템 상태를 바꿈.

B. Flag Argument

함수로 Bool 값을 넘기는 것은 별로임

이는 함수가 한꺼번에 여러 가지를 처리한다고 대놓고 공표하는 것임.

C. Dyadic Functions(이항 함수)

인수가 2개인 함수는 인수가 1개인 함수보다 이해하기 어려움

이항함수가 불가피한 경우도 존재 하지만 그만큼 위험이 따른다는 사실을 이해하고 가능하면 단항함수로 바꾸도록 애써야 함.

D. Triads(삼항 함수)

삼항 함수는 이해하기 엄청 어려움 따라서 만들 때는 신중히 고려해야 함.

E. Argument Objects

인수가 2~3개 필요한 경우 일부를 독자적인 클래스 변수로 선언할 가능성을 생각해 봐야 함.

F. Argument Lists

인수 개수가 가변적인 함수도 필요

가변 인수 전부를 동동하게 취급하면 List 형 인수 하나로 취급할 수 있음.

G. Verbs and Keyword

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

단항 함수는 함수와 인수가 동사/명사 쌍을 이뤄야 함

함수 이름에 키워드를 추가하면 인수 순서를 기억할 필요가 없어짐.

7. Have No Side Effects

Side Effect는 시간적인 결합이나 순서 종속성을 초래함.

A. Output Arguments

일반적으로 인수를 함수 입력으로 해석

하지만 출력 인수로 사용하는 경우도 존재

출력 인수가 불가피한 경우는 객체 지향 프로그래밍이 나오기 전에는 있었음.

하지만 객체 지향 언어에서는 출력 인수를 사용할 필요가 없음.

출력 인수로 사용하라고 설계한 변수가 this이므로

출력 인수는 일반적으로 피해야 함.

함수에서 상태를 변경해야 한다면 함수가 속한 객체 상태를 변경하는 방식을 택함.

8. Command Query Separation(명령과 조회를 분리하라)

함수는 뭔가를 수행하거나 뭔가에 답하거나 둘 중 하나만 해야 함.

둘 다 하면 안됨.

객체 상태를 변경하거나 아니면 객체 정보를 반환하거나 둘 중 하나

둘 다 하면 혼란을 초래

9. Prefer Exceptions to Returning Error Code(오류 코드보다 예외를 사용하라)

오류 코드 대신 예외를 사용하면 오류 처리 코드가 원래 코드에서 분리 됨.

A. Extract Try/Catch Blocks

Try/Cath 블록은 코드 구조에 혼란을 일으키며, 정상 동작과 오류 처리 동작을 뒤섞음. 따라서 Try/Catch 블록을 별도 함수로 뽑아 내는 편이 좋음.

B. Error Handling is One Thing

함수는 한 가지 작업만 해야 함.

오류 처리도 한 가지 작업에 속함

오류 처리하는 함수는 오류만 처리해야 마땅함.

C. The Error.java Dependency Magnet

오류 코드를 반환하다는 이야기는 클래스든 열거형 변수든 어디선가 오류 코드를 정의한다는 뜻임.

새 오류 코드를 추가하게 되면 오류 코드를 사용하는 모든 클래스를 전부 다시 컴파일 해야함.

따라서 오류 코드 대신 예외를 사용하면 재컴파일 재배치 없이 새 예외 클라스를 추가할 수 있음.

10. Don't Repeat Yourself

중복도 문제임

코드 길이가 늘어날 뿐 아니라 알고리즘이 변하면 중복 부분을 모두 손 봐야함.

11. Structured Programming

함수를 작게 만든다면 return break continue를 여러 차례 사용해도 괜찮음.

때로는 단일 입출구 규칙보다 의도를 표현하기 쉬워짐.

Goto 문은 큰 함수에서만 의미가 있으므로 작은 함수에서는 피해야함.