Clean Code

Chapter 2 - Meaningful names

Use Intention-Revealing names

- 측정하는 것을 분명히 드러내는 변수를 사용해라

  ex)

  - int elapsedTimeInDays;
  - int daysSinceCreation;
  - int daysSinceModification;
  - int fileAgeInDays;

  - x, list1 같은 변수를 지양하라
- (비교문의 경우) 숫자를 변수로 나타내어 목적을 드러내라
  - ex) 4 → endFlag
- 어떤 정보를 포함하는지, 어디에 사용되는지 나타내기

Avoid disinformation

- 잘못 해석된 요인을 남기면 안된다.
- 예약어, 범용적으로 알려진 단어 사용을 지양하라. 의도한 의미와 다르면 더더욱
- 구분 어려운 변수 사용을 피하라

ex)

  - XYZControllerForEfficientHandlingOfStrings
  - XYZControllerForEfficientStorageOfStrings

- 스펠링을 정확히
- O & 0 // ㅣ & I & 1 같은 사용을 피하라

Make Meaningful Distinctions

- 프로그래머들은 비슷한 기능의 구분을 위해 misspelling이나 숫자 첨가의 유혹을 받는다. (a1, a2, …)
- a, an, the, s, info, Data등을 붙여 구분하려 하지 마라. 다른 변수를 만들어라

Use Pronounceable Names

- 사람의 특성상, 발음 가능한(말이 되는) 단어이어야 기억을 잘 하고, 직관적으로 의미를 파악한다.
- genymdhms → generationTimestamp

Use Searchable Names

- 의미를 잘 담아내기
- local variable은 주로 단어를, 넓은 scope variable에 단어 조합을

> *“The length of a name should correspond to the size of its scope”*

Avoid Encodings (변수에 사족 붙이지 않기)

1. **Hungarian Notation 피하기**
   - 변수 타입을 적는 행위

     ex)

    - g_ ⇒ 글로벌
    - i_ ⇒ int
    - ch_ ⇒ char
  - 데이터 타입이 바뀌는 경우가 있다.
  - 코드 읽기가 어렵다.
2. **Member Prefix 피하기**
  - 멤버임을 드러내는 접두어 (m_) 사용하지 않기
  - class scope는 충분히 작고, 사용할 필요가 적다.
3. **Interface - Implementaion distinction encoding**
  - Interface와 내부 구현이 겹치는 경우 prefix를 사용해야 할 수 있다.
  - 이때, Interface에 붙이기보다 Implementation에 붙이는 게 좋다.
  - I_ 쓰지 말기 ← 흔하다.

    ex) ShapeFactory to…
    - ShapeFactoryImp
    - CShapeFactory

Avoid Mental Mapping

Class Names

Method Names

Don’t Be Cute

Pick One Word per Concept

Don’t Pun

Use Solution Domain Names

Use Probleom Domain Names

Add Meaningful Context