Clean Code

목차

Chapter 2 : Meaningful Names

우리는 변수, 함수, 클래스, 패캐지 등 코드를 작성하는 과정에서 여러 이름들을 작성하게 된다. 지금부터 좋은 이름을 작성하기 위한 규칙들을 다루고자 한다.

- Use Intention Revealing Name
  - 이름을 붙일 때 해당 이름이 존재하는 이유와 어떤 기능이 존재하며, 어떻게 사용하면 되는지 의도를 잘 표현해야 한다.
  - Code의 implicity를 줄이기 위해, 함축적인 표현을 지양해야 한다.
- Avoid Disinformation
  - 코드를 작성할 때 잘못 해석될 수 있는 가능성을 남겨두면 안된다.
  - 프로그래머가 자주 사용하는 단어(list, vector)를 다른 의미로 사용하면 안된다.
  - 또한, 유사한 이름을 사용하고자 한다면, 기능이 유사한 경우에만 사용이 가능하다.
- Make Meaningful Distinctions
  - 단순히 compiler, interpreter의 문제를 해결하기 위해 이름을 아주 단순하며 의미를 알 수 없게 변경시키는 것은 지양해야 한다.
  - 특히, a1, a2와 같이 숫자만 붙여 구분짓는 것, 혹은 info랑 data와 같이 이름만으로 기능을 구분하기 어렵게 되는 상황을 피해야 한다.
- Use Pronounceable Names
  - 우리는 프로그램을 작성할 때 협업이 필수적이기 때문에 발음 가능한 형태로 변수를 작성해야 한다.
- Use Searchable Names

    - 코드를 작성할 때 종종 검색 기능을 사용하게 되는데, 이때 검색이 용이한 형태로 이름을 작성해야 한다.
    - 특히, 하나의 문자로 이뤄진 이름은 사용을 피해야 한다. 만약 i, k와 같이 이름을 작성하고 싶다면 해당 기능이 매우 좁은 범위 내에서만 사용해야 함을 유의해야 한다.
- Avoid Encodings
    - 과거엔 타입이나 영역같은 정보를 이름에 encoding해 사용하곤 했다. 대표적인 예가 바로 Hungarian Notation이다.
    - 그러나, compiler의 기능이 발전하며 타입을 강제하게 되면서 다양한 타입이 존재하게 되었고, 이름에 굳이 타입 정보를 기입하는 것은 가독성만 떨어트리는 요소가 되었다.
    - 마찬가지로 클래스의 멤버들에 대해 m_을 붙이던 것도 이제 더 이상 필요하지 않다.
- Avoid Mental Mapping
    - 코드를 읽는 과정에서 독자가 변수 이름을 머리 속에서 변환하는 과정이 일어나서는 안된다.
    - i, j, k등은 우리가 관습적으로 loop을 순회할 때 사용하는 이름이기에 괜찮지만, 그 외의 경우에 단일 문자 이름을 사용하는 것은 안된다.
- Class Name
    - 클래스나 객체의 이름은 반드시 명사/명사구를 사용해야 한다.
- Method Name
    - 메서드는 반드시 동사/동사구를 사용해야 한다.
- Pick One Word per Concept
    - 프로그램을 작성할 때 하나의 추상적인 개념은 하나의 단어로만 표현해야 한다.
    - Fetch, Retrieve, Get은 모두 가져오다는 뜻을 가지고 있기에, 하나의 단어만을 골라 같은 기능엔 해당 이름을 사용해야 한다.
- Don’t Pun
    - 그러나, 같은 개념이라고 다른 맥락으로 사용되는 기능을 같은 이름으로 표현해서는 안된다.
    - 어떤 변수에 값을 더하는 Add와, 리스트의 끝에 변수를 추가하는 Append는 비슷해보여도 전혀 다른 맥락을 가지고 있기 때문에 분리시키는게 맞다.
- Use Solution Domain Names

    - 코드의 독자도 프로그래머이기 때문에 Computer Science의 용어를 사용하면 더 쉽게 이해할 수 있다.
    - 기술적 개념을 잘 적용시키는 것이 바람직하다.
- Use Problem Domain Names
    - 적절한 프로그래머 용어가 없다면, 문제에서 용어를 가져오고, 문제 용어가 더 적합하다고 판단되는 경우에도 문제 영역에서 이름을 가져오면 된다.
- Add Meaningful Context
    - 코드의 독자가 이름 자체로 많은 의미를 가지기 위해 이름을 class, function, namespace에 넣어둘 필요가 있다. 만약, 이것이 불가능하다면 prefix를 이용해 표현할 수 있다.
- Don’t Add Gratuitous Context
    - 그러나, 무의미한 의미를 부여하고자 prefix를 사용하고, class의 이름에 정보를 추가할 필요는 없다.
    - 모든 객체가 같은 클래스를 사용할 때는 prefix가 불필요하고, class가 너무 특정한 대상을 지칭하게 만들어서는 안된다.

Chapter 3 : Functions

과거엔 프로그래밍이 루틴, 서브루틴으로 / 프로그램, 서브프로그램, 함수로 표현하였으나, 현재는 프로그래밍을 오직 함수만을 이용해 표현할 수 있게 되었다. 따라서, 함수를 작성할 때에도 다음의 규칙들을 지켜야 명료한 코드라고 할 수 있다.

- Small
    - 함수는 가능한 작게 만들어야 한다.
    - If/else/while 등을 사용하더라도 한 줄로 표현이 가능해야 하고, 중첩 구조가 생길 정도로 커지지 않는 것이 좋다.
- Do One Thing
    - 함수의 크기가 커지는 가장 주요 원인은 하나의 함수에서 여러 작업을 하려고 하기 때문이다.
    - 즉, 함수를 여러 단계의 step으로 나눌 때, 각 step이 추상화 수준이 하나인 구조로 구현되어야 한다.

    - 만약 함수가 쉽게 정의, 초기화 등의 부분들로 구분이 된다면, 해당 함수는 한 가지 이상을 하고 있는 것이다.
- One Level of Abstraction per Function
    - 함수 내부의 statement들은 모두 같은 추상화 수준을 가지고 있어야 한다.
    - 예를 들어, 어떤 객체의 정보를 가져오는 것과, 객체의 하위에 정보를 추가하는 것은 다른 추상화 수준이므로, 하나의 함수 내에 존재해서는 안된다.
- Reading Code from Top to Bottom : The Stepdown Rule

Chapter 4 : Comments

코드를 작성하는 과정에서, 우린 종종 Comment(주석)를 사용하는 경우가 있다. 주석의 적절한 활용법은 코드에서 우리의 “failure”를 표현하기 위한 보충방법이며, 절대로 긍정적이 상황을 위해 쓰여서는 안된다. 주석을 사용하게 되면 우리의 의도를 코드 상에 잘 담아낼 수 없고, 코드의 변화에 맞춰 주석을 잘 변화시키지 않기에 시간이 흐를수록 의미가 퇴색되어버리기 때문이다. 따라서, 주석을 작성하게 된다면 이를 잘 유지보수해 코드의 상황을 잘 타나낼 수 있어야 한다.

A) Good Comment?

좋은 주석은 우리가 코드에서 표현할 수 없는 정보들을 담아내는 경우에만 존재할 수 있다.

- Legal Comment : 법적인 연유로 copyright과 authorhip등을 나타내기 위해 사용하는 경우.
  하지만, 가능하다면 외부 문서나 라이센스 쪽에서 이를 기입하는 것이 더 좋음
- Informative Comment : 의도하였던 정규 표현식의 예시 등을 기입해 정보를 알려줌.

```
//format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher = Pattern.compile{
  "\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

- Explanation of Intent : 우리가 판단했던 결과에 따른 의도를 나타내기 위한 주석.

```
// This is our best attempt to get a race condition
// by creating large number of threads
function ~~
```

- Clarification : 표준 라이브러리와 같이 직접 코드를 변경시킬 수 없을 때 상황을 구별하기 위해 사용

```
assertTrue(~~) // a == a
assertTrue(~~) // a != b
assertTrue(~~) // a < b
반복되는 구조, 코드 이해도를 높이기 위해 주석 추가 가능
```

- Warning of Consequences : 프로그래머가 실수로 잘못된 입력을 주지 않도록 주의하는 경고문
  그러나, 최근엔 @Ignore라는 attribute를 이용해서 대체 가능하다.
- TODO Comment : 함수가 왜 그러한 구현 방식을 채택했는지, 혹은 함수가 앞으로 어떤 식으로 작성되어야 할 것인지 등에 대해 설명하기 위해 작성할 수 있다.
  최근 좋은 IDE들은 특별한 제스쳐나 특징들로 TODO comment를 따로 표시해두곤 한다.

```
// TODO : We expect this to go away when we do the checkout model
```

- Amplification : 어떤 부분의 중요성을 강조하기 위해 사용하곤 한다.
- public API 등을 이용해 작성하는 것도 좋다.

B) Bad Comment?

- 함수나 변수의 이름 등을 재 정의해서 표현할 수 있는 내용의 주석들은 모두 나쁜 예에 속한다.
- 또한, 오래된 주석 등으로 인해 잘못된 방향으로 유도하는 경우도 있다.
- Journal Comment : 업데이트 로그를 작성하는 것처럼 주석을 작성하는 방식이다. 최근엔 이런 로그 들이 불필요하므로 코드 이해도를 높이기 위해 제거하는 것이 좋다.
- Noise Comment : 코드를 읽으면 당연히 알 수 있는 정보(반환값)를 주석으로 나타낸 것.
- Position Marker : 특정 위치를 강조하기 위해 ////////////// 와 같은 형태로 표시해 두는 것.
- Closing Brace Comment : 코드가 길어지게 되면 닫는 괄호가 각각 무엇에 대응되는 것인지 구분이 잘 안가 주석으로 표시한 것.
  주석을 작성하기 보단 코드의 길이를 줄이는데 집중하는 것이 맞다.
- Commented-Out Code : 개발 과정에서 더 이상 안쓰이게 된 코드 부분 들을 컴파일 과정에서 제거하기 위해 주석처리 한 것. 다른 사람이 이 코드를 보거나 내가 오랜 시간이

지난 후에 봐도 삭제해도 되는 주석인지 판단이 잘 안된다.

- Nonlocal Information : 주석과 해당 주석이 나타내는 코드가 동 떨어져 있는 경우.

Chapter 7 : Error Handling