Clean code

Ch 4. Comment

> *챕터를 읽고..*
> Comments를 추가하는 것보다 code를 improve 해야 한다는 말에 동감이 되고, 인상깊음.
> 그러나 Comments를 아예 쓰지 않는 것은 너무 이상적인 코드에게만 적용 가능한 것이지 않을까.

Code에서 적절한 위치에 작성된 comment는 그 무엇보다 유용하다. 하지만 잘못 작성된 comment는 code를 어지럽히고 해롭다. 프로그래밍 언어에 따라서, 혹은 개발자의 프로그래밍 언어를 사용하는 방식의 표현에 지대한 영향을 주는 언어에 대한 이해도에 따라서는 주석이 적게 필요하거나, 아예 필요 없을 수도 있다.

'Mess' 한 상태의 code를 보완하기 위해 comment를 사용하는 것은 분명히 좋지 않다. 그럴 때는 code를 clean-up하는 것의 우선순위가 당연히 옳다. 또한, comment 가 아닌 code에서 개발의도를 잘 설명하도록 해보자.

ex) 다음 중, 더 알아보기 쉬운 것은 후자이다.

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) &&
(employee.age > 65))
```

```
if (employee.isEligibleForFullBenefits())
```

Good Comments

- Legal Comments
- Informative Comments
  : basic information의 제공
- Explanation of Intent

- Clarification
  : obscure한 arguments, return value 등을 읽기 더 쉬운 형태로 작성하는 것. 틀린 해석을 적는 경우 risky하다.
- Warning of Consequences
  : ex) test case 를 제외한 이유를 굉장히 오래걸리기 때문인 것으로 설명 → `@Ignore`
- TODO
  : reminder
- Amplification
  : Amplify some parts of the code
- Javadocs in Public APIs

Bad Comments

- Mumbling
  : 의미가 불분명한 comment. author 자신만 알아볼 수 있도록 적힌 경우가 있음.
- Redundant Comments
- Misleading Comments
- Mandated Comments
  : ex) 모든 function 에 javadoc을 다는 경우
- Journal Comments
  : 과거에는 유용했으나, VCS가 있는 지금은 전혀 불필요.
- Noise Comments
  : obvious한 code의 설명 등
- Can be expressed by a function or variable
- Position Markers
- Comments after Closing Braces
- : ex) `while{ ...several lines... } //while`, 만일 braces 간격이 너무 멀어 필요하다 생각했다면 brace 의 부피를 먼저 줄여 볼 것.
- Adding who wrote the code
  : VCS만으로 충분!

- Commented-out code
- HTML Comments
- Explaining the code far away
- TMI
- Inobvious Comments
- Function Headers
- Javadocs in Nonpublic Code

Ch7. Error Handling

> *챕터를 읽고..*
> 내가 Exception 처리를 하는 데에 미숙하다는 것을 우선 앎.
> 우선은 책의 내용을 따라가려 할 것 같음.

Code의 작성과 떼낼 수 없는 것이 Error Handling이다.

> Error handling is important, but it *obscures logic*, it's wrong.

- Use Exceptions Rather Than Return Codes
- Write Your `try-catch-finally` Statement First
  : Exception이 발생될 수 있는 코드에서!
- Use Unchecked Exceptions
  : Checked exception - throw와 catch 사이의 모든 method에 exception이 필요. encapsulate 깨짐.
- Provide Context with Exception
- Define Exception Classes in Terms of a Caller’s Needs
  : *how they are caught* 를 고려하는 것이 가장 중요.
- Define the Normal Flow
  : SPECIAL CASE PATTERN

- Don’t Return Null & Don’t Pass Null
  : `NullPointerExceptions`, `InvalidArgumentException`

Ch8. Boundaries

> *챕터를 읽고..*
> 라이브러리의 사용 시, class로 감싸는 것이 필요하다는 것을 앎.

개발에 있어서 모든 코드를 직접 생산하지 않고, 라이브러리를 작성하거나 타 팀의 코드에 의존되는 개발을 하는 경우가 있다. 이때, own code 와 foreign code 의 경계를 어떻게 구분짓고, own boundary 를 깔끔하게 구분지을까.

- Using Third-Party Code
  : 해당 code를 직접 이용하지 않고, 사용할 class 를 선언하여 그 내부에서 이용하자.
  Third-Party Code에서 제공하는 인터페이스에서 일부 기능을 제한하기 용이함.
- Exploring and Learning Boundaries
  : Third-Party Code 의 이해를 위한 *learning test*.
  : Third-Party Code 의 version upgrade 로 사용하는 부분의 지원이 달라졌을 때, *boundary test* 가 있으면 migration이 용이.

Ch9. Unit Tests

> *챕터를 읽고..*
> *지속적으로 개발, 업데이트가 이루어질 코드에 대하여는 production code에 대한 테스트도 유지보수가 용이하도록 관리하는 것이 최우선.*

Agile & TDD

- The Three Laws of TDD
  : 이를 따르면 production code보다 앞서서 test code를 작성하게 되고, 관리하기 힘들 정도로 엄청난 양의 test를 쓰게 될 수도 있다.

  1. You may not write production code until you have written a failing unit test.

2. You may not write more of a unit test than is sufficient to fail, and not compiling is failing.
3. You may not write more production code than is sufficient to pass the cur- rently failing test.

- Keeping Tests Clean
  : Production code 가 지속적으로 개발됨에 따라 test code 도 수정되어야 함.
  : Test 가 dirty 함 → production 관리가 어려움 → 실패
- Clean Tests = **Readability**
- One Assert per Test
- F.I.R.S.T
  : Fast, Independent, Repeatable, Self-Validating, Timely

Ch10. Classes

> *챕터를 읽고..*
> *Class 를 어떻게 관리해야 하는지의 문제도 코드의 유지보수를 용이하게 하는 방향으로 서술됨.*

Higher levels of code organization

- Class Organization
  - Encapsulation
- Classes Should Be Small!
  - The Single Responsibility Principle - class or module should have ‘a’ reason to change
  - Cohesion
  - Maintaining Cohesion Results in Many Small Classes
- Organizing for Change
  - Isolating from Change