# Clean Code 요약문

20160169 최수민

## CH2 - Meaningful Names

1. 핵심
    a. 의도가 잘 드러나는 이름을 지어라.
2. 새롭게 배운 것
    a. 이름에 자료형(e.g. ~List)가 들어가는 경우 잘못된 이해를 불러올 수 있다.
    b. Info, Data는 noise word로, 별다른 의미가 없다.
    c. 발음 가능한 이름으로 정하는 게 좋다. 이 부분은 생각해 본 적이 없는데, 확실히 아는 단어나 발음이 쉬운 이름으로 정해야 변수의 의미가 잘 와닿는 것 같다.
    d. 함수 이름에는 동사를 포함해야 한다.
3. 중요하다고 느낀 것
    a. 이름 짓는 데 시간을 많이 쓰더라도, 불필요한 단어를 없애고 정확한 의도를 담은 이름으로 정할 수 있도록 해야 한다.

## CH3 - Functions

1. 핵심
    a. 함수는 작게 나누어서, 되도록 한 함수가 하나의 기능을 하도록 해라
    b. 함수의 인자는 적을수록 좋다.
2. 새롭게 배운 것
    a. 함수에는 하나의 추상화 레벨만 존재하도록 해라.
    b. 함수의 이름이 한 가지 기능을 나타내도록 분리하면 좋다. 간단하게는 함수의 이름에 and가 들어가지 않도록 해보면 좋을 것 같다.
        i. error handling 역시 하나의 기능이다. try-catch 구문을 바깥으로 빼 별도의 함수를 만들자.
    c. 함수의 인자가 많을수록 테스트하는 데도 어렵다. 인자가 많다면 연관 있는 인자들을 하나의 class로 wrapping하는 것도 좋다.
    d. 되도록이면 출력용 인자는 피하는 것이 좋다.
3. 중요하다고 느낀 것
    a. 함수가 짧을수록 각 함수가 하는 일이 투명하게 드러나게 된다.
    b. 개발하면서 함수 작성을 피할 수 없는 만큼, 함수를 작성할 때 더 간결하고 가독성 좋은 코드를 작성하기 위해 많이 노력해야 할 것 같다.

## CH4 - Comments

1. 핵심
    a. 주석이 필요 없는 설명적인 네이밍, 코딩을 해라
    b. 꼭 주석을 써야겠다면 그만큼 시간 투자를 해서 명확하게 작성해라
2. 새롭게 배운 것
    a. '좋은 주석'도 존재하긴 한다. 하지만 이러한 주석을 반드시 달아야 하는지 다시 한 번 고민하고, 만약 주석이 꼭 필요하다고 판단되면 누가 봐도 이해할 수 있게 명확히 쓸 수 있도록 시간과 노력을 투자해 작성해야 한다.
        i. copyright 등을 명시한 주석이나 TODO comments
        ii. 프로그래머의 의도를 알리기 위한 주석
        iii. 결과에 대해 경고하는 주석
        iv. Public API에 대한 Javadocs 주석
    b. Javadocs처럼 코드 안에서 documentation을 하게 해주는 라이브러리들의 경우, Nonpublic code에 대해 문서화를 하지 않도록 주의해야 한다.
    c. variable를 잘 활용하거나 함수를 짧게 쓰는 습관만 들여도 가독성이 크게 좋아지고 불필요한 주석을 줄일 수 있다는 것을 느꼈다.
3. 중요하다고 느낀 것
    a. 주석은 되도록 안 쓰는 것이 좋다. 특히 코드가 변하는 것에 주석이 맞춰 따라가기는 정말 정말 어려우니 주석을 쓰는 것보다 코드를 깔끔하고 expressive하게 써서 주석이 아예 필요 없도록 하는 것이 좋다.
    b. 주석의 안 좋은 예에서 version history를 쓴다거나 누가 쓴 코드인지 적어두는 등의 문제들은 git을 사용하면 해결되는 문제였다. 그만큼 git에 익숙해지고 앞으로도 잘 활용하는 게 필요할 것 같다.
    c. 필요 없는 주석은 지워야 한다. 예전에 사용했지만 지금은 사용하지 않는 코드를 혹시 몰라 남겨둘 필요가 없다. git이 저장해줄 것이다.

## CH7 - Error Handling

1. 핵심
    a. Error code를 리턴하는 방식을 사용하지 말고 적절한 Exception을 발생시켜라
    b. try-catch-finally 구문을 기본으로 사용해라
2. 새롭게 배운 것
    a. try-catch-finally 구문으로부터 시작해 TDD(Test-driven development)를 하는 방법이 굉장히 새로운 내용이었다. 먼저 특정 exception이 발생하는 것을 테스트하는 unit test를 작성하고, 그에 대한 로직을 작성할 때도 try-catch-finally 구문부터 시작한다면 exception 처리를 빼먹을 일도 없을 것이다.

b. Exception 처리도 simple wrapper를 만들어 처리해주면 코드가 훨씬 깔끔하고 나중에 유지보수도 쉬워진다. Wrapper를 만들어 처리하면 좋은 대표적인 경우들은 다음과 같다.
    i. third-party library를 사용할 때. 외부 코드에 대한 dependency를 줄일 수 있다는 측면에서도 이점이 있다.
    ii. 예외가 있어 Exception을 통해 분기해 처리하는 경우
    iii. Null을 리턴하는 경우. 특히 null의 경우 null을 그냥 리턴해서도 안되지만 null을 패스하는 것은 더더욱 좋지 않으며, callee 함수에서도 필요할 경우 assert문 등을 활용해 적절히 막아야 한다.

3. 중요하다고 느낀 것
    a. Exception을 발생시킬 때도 해당 exception이 발생한 context를 충분히 제공해야 한다. 즉 에러 메시지도 충분한 정보를 포함하도록 작성해야 한다.

## CH8 - Boundaries

1. 핵심
    a. Third-party library를 사용할 땐 Wrapper class를 만들어 사용해라
    b. Learning tests를 해라
2. 새롭게 배운 것
    a. Learning tests는 외부 모듈의 사용을 익힐 때 관련된 테스트를 작성하면서 익히는 것을 말한다. 이를 통해 내가 사용할 부분에 대해 익히는 것 뿐만 아니라, 외부 모듈에 대한 테스트도 함께 작성할 수 있으므로 외부 모듈에 버전 업그레이드 등의 변화가 있을 때에도 해당 테스트를 활용해 내가 작성한 코드에 문제가 없을지 확인할 수 있다는 장점이 있다. 외부 모듈을 사용할 때 해당 모듈에 대한 세세한 테스트는 당연히 어렵지만, 적어도 사용할 부분에 대해선 테스트를 만들어 관리하는 게 필요하다.
3. 중요하다고 느낀 것
    a. Third-party library를 사용할 땐 거의 필수적으로 wrapper class를 만드는 것이 필요한 것 같다. 이는 외부 모듈에 대한 의존도를 줄여 모듈의 변화에 대처하는 것이나 다른 모듈로 변경할 때 큰 이점을 가져다준다. 또한 모듈에서 필요 없는 기능에 대해 숨겨두고 접근할 수 없도록 할 수 있다는 장점도 있다.