Clean Code Summary 4,7

Dongwon Kim

Comments(1)

- Less comment/more clear code is better
- Comment는 code와 함께 동시에 관리되기 힘들다.
- Orphan commen가 발생 -> inaccurate comment is worse than no comment!
- 지저분한 코드를 comment로 매울 생각 하지 말자.
- 최대한 code로 표현하자.

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

Or this?

```
if (employee.isEligiblePorFullBenefits())
```

Comments(2)

- Then what is good comments?
  - Legal comments – copyright/authorship etc..
  - Explanation of Intent – 왜 이 코드가 있는지..
  - 읽기 쉬운 간단한 형태로 표현할 때. Ex) compare(a,b) // a > b
    - Risk 존재.. Function name을 clear하게 짓는 것이 better.
  - 결과에 대한 경고.. –require(), assert()를 쓰면 되지 않을까?
    - 아닐 수도 있다. 예를 들자면, lazy val로 resul가 생성된다던지..
  - To-Do comment - IDE에 있는 기능을 사용하자.
  - 강조하고 싶을 때.

Comments(3)

- Bad comments!
  - 얼버무리지 마세요 – 다른 모듈을 찾아 볼 필요 없게 쓰라. 똑바로 쓰라.
  - Redundant comment – code가 clear할 경우 불필요.
  - 전혀 필요가 없는데 의무감에 javadoc을 생성하는 행위 – potential for lies and misleading.
  - 일기장을 써놓는 경우. – VCS before VCS, 차라리 git을 써라
  - Function/variabl에 more clear name을 주어 단순화 시킬 수 있는 경우에는 comment가 필요 없다.

```
// does the module from the global list <mod> depend on the
// subsystem we are part of?
if (smodule.getDependSubsystems().contains(subSysMod.getSubSystem()))
```

This could be rephrased without the comment as

```
ArrayList moduleDependees = smodule.getDependSubsystems();
String ourSubSystem = subSysMod.getSubSystem();
if (moduleDependees.contains(ourSubSystem))
```

Comments(4) – bad comment이어서.

- Marker – comment 아래부터의 function들을 설명하는 comment
  ex) // operator ////////////////////////// -> use very carefully.
- 중괄호 닫을 때 이를 comment로 설명하려는 행위 -> 차라리 function으로 extraction해라.
- comment로 code 지우는 행위 – git이 있잖아.. 이런거 하지말기.
- System wide comment – 최대한 가까운 코드만을 설명하도록 해라. 나중에 일일이 찾아서 고칠 수 없으니 거짓말을 하게 될 확률이 높아진다.

Error handling(1)

- Use Exception rather that return code
  - Caller가 모든 call 마다 error 를 확인할 필요가 없다.
  - Code 가 훨씬 간단하게 쓰여진다.
- Try-catch-finally 구조를 쓰면서 시작해라.
- Checked exception 을 사용하지 말라
  - Benefit 보다 dependency cost 가 더 크다.
- Stack trace 만으로 exception 판별하기는 어렵다.
  - Error message를 잘 쓸 것
- Exception class를 잘 설계할 것.
  - 여러 개를 정의하는 것이 아니라, 같은 handling 이 필요한 것은 같은 class로. 또한 try-catch를 내부적으로 하는 wrapper를 만들어 dependency를 줄일 수 있다.

```
public class LocalPort {
  private ACMEPort innerPort;

  public LocalPort(int portNumber) {
    innerPort = new ACMEPort(portNumber);
  }

  public void open() {
    try {
      innerPort.open();
    } catch (DeviceResponseException e) {
      throw new PortDeviceFailure(e);
    } catch (ATM1212UnlockedException e) {
      throw new PortDeviceFailure(e);
    } catch (GMXError e) {
```

Error handling(2)

- Special case에 대해 exception을 항상 사용하지 말자
  - Special Case Pattern – special case에 대해 class를 만들자.
- Null 을 return 하지 말자.
  - 이 경우에는 special case pattern 을 이용하거나, exception 던지기.
- Null 을 pass하지 말자.
  - Exception handler 를 정의하여 사용할 가치도 없는 경우.
  - 왜냐하면.. Null 이 들어온다면 function이 제대로 작동 못하고, 딱히 recover할 방법이 없다.
  - 아예 null을 pass 받는 것을 금지하는 게 옳다. – use assert!