chapter 4: Comments

introduction

- comment **LIE**s.
  - 코드는 계속 바뀌고 진화하는 반면, 주석은 그 흐름을 쫓아가기 힘들다.
  - 주석을 정확하게 유지보수하는 것은 힘든 일이다. 차라리 코드를 더 열심히 고쳐라.
  - 부정확한 주석은 없으니만 못하다.

comment and bad code

- 사실 주석을 쓰는 주된 이유는, 코드가 엉망이어서 어떻게든 설명을 주석으로 다는 것이다.
  - 그 시간에 코드를 정리하라!

comment and code

- 아래 코드와 주석을 보자

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

- 그냥 함수 이름에 써도 될 내용 아닌가?

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

  - 이렇게 작성해도 의도는 충분히 전달되었다.
  - 함수를 하나 만들고 이름을 잘 지으면 주석이 필요 없는 상황이 생긴다.

Good Comments

Legal Comments

맨 위에 copyright, release date 등을 적어놓는 것이다.

Informative Comments

- 주석이 없다면 어떤 정규식을 표현한건지 바로 알아보기 어렵다.

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

- 함수 반환값과 boolean 연산이 헷갈리므로 간단하게 주석을 추가했다.

```
assertTrue(a.compareTo(a) == 0);    // a == a
assertTrue(a.compareTo(b) != 0);    // a != b
assertTrue(ab.compareTo(ab) == 0);  // ab == ab
assertTrue(a.compareTo(b) == -1);   // a < b
assertTrue(aa.compareTo(ab) == -1); // aa < ab
assertTrue(ba.compareTo(bb) == -1); // ba < bb
assertTrue(b.compareTo(a) == 1);    // b > a
assertTrue(ab.compareTo(aa) == 1);  // ab > aa
assertTrue(bb.compareTo(ba) == 1);  // bb > ba
```

- 이런 경우가 아니라면 주석을 방대하게 쓰지 말고 함수 이름을 잘 짓자

Consider the following stretch of code:

```
// 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))
```

Explanation of Intent

- 만약 반환값이 0,1과 같다면 왜 1인지 설명해주는 식이다.

```
if(o instanceof WikiPagePath)
{
  WikiPagePath p = (WikiPagePath) o;
  String compressedName = StringUtil.join(names, "");
  String compressedArgumentName = StringUtil.join(p.names, "");
  return compressedName.compareTo(compressedArgumentName);
}
return 1; // we are greater because we are the right type.
```

- 또는 테스트 코드에서 넣은 값들에 설명을 붙이는 경우도 있겠다.

```
//This is our best attempt to get a race condition
//by creating large number of threads.
for (int i = 0; i < 25000; i++) {
  WidgetBuilderThread widgetBuilderThread =
    new WidgetBuilderThread(widgetBuilder, text, parent, failFlag);
  Thread thread = new Thread(widgetBuilderThread);
  thread.start();
}
assertEquals(false, failFlag.get());
```

TODO Comments

요즘 IDE는 TODO를 자동으로 인식해서 모아보기를 제공하기도 한다.
그래도 구현하는 대로 삭제해 주도록 하자

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
protected VersionInfo makeVersion() throws Exception
{
  return null;
}
```

Bad Comments

Mumbling(중얼대기)

그냥... 왠지 주석을 써야 할 것 같고... 해서 대충 쓰지 마라

```
  }
  catch(IOException e)
  {
    // No properties files means all defaults are loaded
  }
```

so what??

Redundant Comments

```
// Utility method that returns when this.closed is true.
// Throws an exception if the timeout is reached.
public synchronized void waitForClose(final long timeoutMilli
throws Exception
{
    if(!closed)
    {
        wait(timeoutMillis);
        if(!closed)
            throw new Exception("MockResponseSender could not
    }
}
```

- 코드보다 긴 주석, 그냥 코드의 실행 순서를 설명하는 주석은 필요 없다.
  - 코드를 읽으면 바로 알 수 있는데 주석이 왜 필요한가?
- 변수 선언 하나하나에 주석을 달아놓는것도 지양하자.
- 사실 위에 나와있는 예시의 주석도 말이 이상하다.
  - returns when this.closed is true가 아니라
    returns if this.closed is true 이다.
    - false일경우 타임아웃하고 예외를 던질 뿐이다.
      즉 when보다 if라고 써야 정확하다.

Journal Comments

맨 위에 코드 변경 이력을 git 메시지처럼 남겨 놓는 것이다.

git이 없던 시절에나 사용했지, 이제는 필요 없다.

Position Markers

필요 없다

```
////////ACTIONS///////
```

Closing Brace Comments

필요 없다

```
try{
    while{
        ...
    } //while
} //try
```

Commented-Out Code

지우기는 망설여지고 더 안쓰는 코드에 주석처리 해놓지 말기!

- 완벽히 테스트하고 그냥 지우자
- git이 생긴 이래로 필요 없어졌다

TMI

Inobvious connection

코드에 있는 숫자나 변수 설명을 주석으로 빼놓는 경우다.

그냥 define을 쓰거나 이름을 잘 짓는걸로 대체하자.

```
/*
 * start with an array that is big enough to hold all the pixels
 * (plus filter bytes), and an extra 200 bytes for header info
 */
this.pngBytes = new byte[((this.width + 1) * this.height * 3) + 200];
```

What is a filter byte? Does it relate to the +1? Or to the *3? Both? Is a pixel a byte? Why 200? The purpose of a comment is to explain code that does not explain itself. It is a pity when a comment needs its own explanation.