기술 학습/클린 코드

로버트 C. 마틴의 클린코드 정리 _ 4장 (주석)

벼리구름 2023. 7. 4. 17:44 수정 예약 공개 삭제

주석

나쁜 코드에 주석을 달지 마라. 새로 짜라.

주석은 필요악이다. 우리는 코드로 의도를 표현하지 못해서, 그러니깐 **실패**를 만회하기 위해 주석을 사용하는 것이다. 주석은 언제나 실패를 의미한다 이는 **코드로 자신을 표현할 방법을 찾지 못해 할 수 없이 주석을 사용**한다. 따라서 주석은 반겨 맞을 것이 아니다.

주석을 엄격하게 관리해야 한다고도 한다. 그 의견도 맞지만, 코드를 깔끔하게 정리하고 표현력을 강화하는 방향으로, 그래서 애초에 주석이 필요없는 에너지를 쏟아야 한다.

부정확한 주석은 아예 없는 주석보다 훨씬 더 나쁘다. 부정확한 주석은 독자를 현혹하고 오도한다.

진실은 한곳에만 존재한다. 그것이 바로 코드다. 코드만이 자기가 하는일을 진실되게 말한다. 코드만이 정확한 정보를 제공하는 유일한 출처다. 따라서 주석을 가능한 줄이도록 꾸준히 노력해야 한다.

주석은 나쁜 코드를 보완하지 못한다.

코드에 주석을 추가하느 일반적인 이유는 코드 품질이 나쁘기 때문이다. 코드를 짜고보니 짜임새가 엉망이고 알아먹기 어려우니깐 주석을 달게 되는것 지만 그게 **아니다! 코드를 정리해야 한다.**
자신이 저지른 난장판을 주석으로 설명하려 애쓰는 대신에 그 난장판을 깨끗이 치우는데 시간을 보내라 !

코드로 의도를 표현하라

코드만으로 의도를 설명하기 어려운 경우가 존재한다. 개발자들은 이를 코드가 훌륭한 수단이 아니기 때문으로 해석하며 주석을 단다. 이는 잘못된 생 래 예제를 비교해보자

```
// 직원에게 복지 혜택을 받을 자격이 있는지 검사한다.
if ((employee.flags && HOURLY_FLAG) && (employee.age > 65))
```

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


구름 개발일기장

좋은 주석

필요하거나 유익한 주석도 있다. 이를 소개한다. 하지만 명심하라 *정말로 좋은 주석은, 주석을 달지 않을 방법을 찾아낸 주석*이다.
(좋은 주석이란게 주석을 작성하는게 좋다라기 보다는 피치못하게 사용하는 경우라고 봐야 더 적합한것 같다.)

법적인(Legal) 주석

회사가 정립한 구현 표준에 맞춰 법적인 이유로 특정 주석을 넣으라고 명시하는 경우.
예를 들어, 각 소스 파일 첫머리에 주석으로 들어가는 저작권 정보와 소유권 정보는 필요하며 타당하다.

정보를 제공하는 주석

기본적인 정보를 주석으로 제공하면 편리한 경우.
예를 들어, 추상 메서드가 반활할 값을 설명한다.

첫번째 예제

```
// 테스트 중인 Responder 인스턴스를 반환하다.
protected abstract responderInstacne: Responder
```

위와 같은 주석이 설령 유용할지라도, 가능하다면, 아래와 같이 함수 이름에 정보를 담는편이 더 좋다.

```
protected abstract responderBeingTested: Responder
```

두번째 예제

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

위에 제시한 주석은 코드에서 사용한 정규표현식이 시각과 날짜를 뜻함을 설명한다. 하지만 이것도 이왕이면 시각과 날짜를 변화하는 클래스를 만들어 옮겨주면 더 좋고 깔끔하겠다.

의도를 설명하는 주석

구현을 이해하게 도와주는 선을 넘어 결정에 깔린 의도까지 설명하는 경우.

첫번째 예제
주석으로 흥미로운 결정을 기록한 예제이다. 두 객체를 비교할 때 저자는 다른 어떤 객체보다 자기 객체에 높은 우선순위를 주도록 했다.

```
fun compareTo(o: Any): Int {
    if (o is WikiPagePath) {
        val p: WikiPagePath = o as WikiPagePath
        val compressedName: String = StringUtil.join(names, "")
        val compressedArgumentName: String = StringUtil.join(p.names, "")
        return compressedName.compareTo(compressedArgumentName)
    }
```

더 주석이 존재하는 이유를 잘 타나낸 경우이다.

```
fun testConcurrentAddWidgets() {
    val widgetBuilder: WidgetBuilder = WidgetBuilder(BoldWidget::class)
    val text: String = "'''bold text'''"
    val parent: ParentWidget = BoldWidget(MockWidgetRoot(), "'''bold text'''")
    val failFlag = AtomicBoolean()
    failFlag.set(false)

    // 스레드를 대량 생성하는 방법으로 어떻게든 경쟁 조건을 만들려고 시도한다.
    for (i in 0 until 25000) {
        val widgetBuilderThread: WidgetBuilderThread = WidgetBuilderThread(widgetBuilder, text, pa
        val thread: Thread = Thread(widgetBuilderThread)
        thread.start()
    }
    assert(false == failFlag.get())
}
```

의미를 명료하게 밝히는 주석

모호한 인수나 반환값은 그 의미를 읽기 좋게 표현하면 이해하기 쉬워지는 경우.

물론 인수나 반환값 자체를 명확하게 하는게 좋다. 하지만 표준 라이브러리나 변경하지 못하는 코드에 속한다면 아래 예제처럼 의미를 명료하게 밝히

유용하다.

```
fun `test`() {
    val a = "a"
    val ab = "a.b"
    val b = "b"
    val aa = "a.a"
    val bb = "b.b"
    val ba = "b.a"

    a.compareTo(a) shouldBe 0       // a == a
    a.compareTo(b) shouldNotBe 0    // a != b
    ab.compareTo(ab) shouldBe 0     // ab == ab
    a.compareTo(b) shouldNotBe -1   // a < b
    aa.compareTo(ab) shouldBe -1    // aa < ab
}
```

물론 위 예제에서도 그릇된 주석을 달아놓을 위험은 **상당히!** 높다. 왜냐하면 주석이 올바른지 검증하기가 쉽지 않기 때문이다. 의미를 명료히 밝히는 ?

요하면서도 동시에 주석이 위험한 이유이다.

결과를 경고하는 주석

다른 프로그래머에게 결과를 경고할 목적으로 사용하는 경우.

첫번째 예제

```
    // given
    for (i in 0..Long.MAX_VALUE) {
        for (j in 0..Long.MAX_VALUE) {
            for (k in 0..Long.MAX_VALUE) {
                // 오래걸리는 로직
                if (i == k) i shouldBe k
            }
        }
    }
    // when
    // then
}
```

물론 지금은 아래처럼 처리할 수 있다.

```
@Test
@Disabled("너무 오래걸리는 테스트...")
fun `too long time test`() {
    // given
    // when
    // then
}
```

두번째 예제

```
class StandardHttpDate {
    companion object {
        fun makeStandardHttpDateFormat(): SimpleDateFormat {
            // SimpleDateFormat은 스레드에 안전하지 못하다.
            // 따라서 각 인스턴스를 독립적으로 생성해야 한다.
            val df: SimpleDateFormat = SimpleDateFormat("EEE, dd MMM yyyy HH::mm:ss z")
            df.timeZone = TimeZone.getTimeZone("Asia/Seoul")
            return df
        }
    }
}
```

이 경우 주석이 아주 합리적이다. 프로그램 효율을 높이기 위해 정적 초기화 함수를 사용하려던 프로그래머가 주석때문에 실수를 방지할 수 있다.

TODO 주석

'앞으로 할 일'을 //TODO 주석으로 남겨두는 경우.

아래 함수는 함수를 구현하지 않은 이유와 미래 모습을 주석으로 설명하는 코드이다.

```
// TODO: MdM 현재 필요하지 않다.
// 체크아웃 모델을 도입하면 함수가 필요 없다.
protected makeVersion(): VersionInfo {
    return null
}
```


구름 개발일기장

중요성을 강조하는 주석

어떤 함수가 꼭 필요하다고 강조하는 경우.
하지만 이 경우는 함수로 빼면 충분히 주석없이 처리할 수 있겠다.

나쁜 주석

대다수 주석은 **나쁜 주석**이다.
대다수 주석은 **허술한 코드를 지탱**하거나, **엉성한 코드를 변명**하거나, **미숙한 결정을 합리화**하는 프로그래머가 주절거리는 독백에 불과하다.

주절거리는 주석

특별한 이유 없이 의무감으로 마지못해 다는 경우.
적어도 주석을 달기로 결정을 했다면 충분한 시간을 들여 최고의 주석을 달기위해 노력해야한다.

예제 코드

```
fun loadProperties() {
    try {
        val propertiesPath = "$propertiesLocation/$PROPERTIES_FILE"
        val propertiesStream = FileInputStream(propertiesPath)
        loadProperties.load(propertiesStream)
    } catch (e: IOException) {
        // 속성 파일이 없다면 기본값을 모두 메모리로 읽어 들였다는 의미이다.
    }
}
```

catch 블록 내 주석의 의미가 무엇일까?
IOException이 발생하면 속성파일이 없다라는 뜻이다. 주석 왈 "*그렇다면 이미 모든 기본값을 메모리로 읽여 들였다*"라고 말하고 있다.

여기서 궁금점이 생긴다. 예외가 발생했을 때 누가 모든 기본값을 읽어 들였다는건가?

- loadProperties.load()를 호출하기 전에 읽어 들이는건가?
- loadProperties.load()가 예외를 잡아 기본값을 읽어 들인 후 예외를 던지는 건가?
- loadProperties.load()가 파일을 읽어 들이기 전에 모든 기본값을 읽어 들이는가?
- 추후 나중에 주석에 기본값을 읽어 들이는 코드를 구현하려고 했는가?
- 그냥 비워놓으면 어색해서 써놓은 주저리에 불과한가?

위 답은 다른 코드를 뒤져봐야 알 수 있는 부분이다. 이해가 안되어서 다른 모듈까지 찾아봐야 한다면 해당 주석은 독자와 소통에 실패한 주석이다.

같은 이야기를 반복하는 주석

구름 개발일기장

첫번째 예제

```
// this.closed가 true일 때 반환되는 유틸리티 메서드이다.
// 타임아웃에 도달하면 예외를 던진다.
@Synchronized fun waitForClose(timeoutMills: Long) {
    if (!closed) {
        Thread.sleep(timeoutMills)
        if (!closed) {
            throw Exception("MockResponseSender could not be closed")
        }
    }
}
```

주석을 달아놓은 목적이 뭘까?

- 주석이 코드보다 더 많은 정보를 제공하지도 못한다.
- 코드를 정당화하는 주석이 아니다.
- 의도나 근거를 설명하는 주석이 아니다.
- 코드보다 읽기 쉽지도 않다.
- 실제 코드와 비교해 부정확하다.

이러한 주석으로 인해 코드를 제대로 보지 않고 주석만을 보고 이해한 후 넘어가게 만든다. 이런 주석은 중고차 판매원이 엔진 후드는 열어볼 필요가
객에게 아양떠는 행위와 비슷하다.

두번째 예제

```
open class ContainerBase : Container, LifeCycle, PipeLine, MBeanRegistration, Serializable {
    /**
     * 이 컴포넌트의 프로세서 지연값
     */
    protected val backgroundProcessorDelay = -1
    /**
     * 이 컴포넌트를 지원하기 위한 생명주기 이벤트
     */
    protected val lifeCycle = LifeCycleSupport(this)
    /**
     * 이 컴포넌트를 위한 컨테이너 이벤트 Listener
     */
    protected val listener = arrayListOf<String>()
    /**
     * 이 컴포넌트와 관련된 Loader 구현
     */
    protected val loader: String = ""
}
```

쓸모없고 중복된 javadocs가 매우 많다. 이런 주석은 코드만 지저분하고 정신없게 만든다.

오해할 여지가 있는 주석

```
// this.closed가 true일 때 반환되는 유틸리티 메서드이다.
```

주석 내용과 다르게 코드에서는 *this.closed* 가 *true*로 변하는 순간에 (true가 될 때) 메서드가 반환되지 않는다. *this.closed* 가 *true* 여야 반환된다. t
니라면 timeoutMills를 전부 기다렸다가 그래도 true가 아니면 예외를 던진다.

코드보다 읽기 어려운 주석에 담긴 '살짝 잘못된 정보'로 인해 this.cloesd가 true로 변하는 순간 바로 반환되리라고 생각한 프로그래머가 이 함수를 호
고, 코드가 굼벵이처럼 느려진 이유를 찾느라 골머리를 앓게 된다.

의무적으로 다는 주석

모든 함수에 javadocs를 달거나 모든 변수에 주석을 달아야 한다는 규칙으로 다는 주석.
어리석기 그지없는 규칙이다. 이런 주석은 코드를 복잡하게 만들고, 거짓말을 퍼뜨리고, 혼동과 무실서를 초래한다.

예제 코드

```
/**
 * @param title CD 제목
 * @param author CD 저자
 * @param tracks CD 트랙 수
 * @param durationInMinutes CD 길이 (단위: 분)
 */
fun addCD(
    title: String,
    author: String,
    tracks: Int,
    durationInMinutes: Int,
) {
    val cd = CD()
    cd.title = title
    cd.author = author
    cd.tracks = tracks
    cd.durationInMinutes = durationInMinutes
    cdList.add(cd)
}
```

위와 같은 **javadocs 주석은 아무런 가치가 없다**. 오히려 코드만 헷갈리게 만들며, 거짓말할 가능성을 높이고, 잘못된 정보를 제공할 여지만 만든다.

이력을 기록하는 주석

모듈을 편집할 때마다 모듈 첫머리에 작성하는 주석.

예제 코드

```
/**
 * 변경 이력 (11-Oct-2001부터)
 * --------------------------
 * 11-Oct-2001 : 클래스를 다시 정리하고 새로운 패키지인 com.jeffry.date로 옮겼다. (DG)
 * 27-Oct-2002 : getDescription() 메서드를 추가했으며
 *               NotableDate class를 제거했다. (DG)
```

구름 개발일기장

과거에는 소스 코드 형상 관리 시스템이 없었으므로 변경 이력을 기록하고 관리하는 관례가 바람직했다. 이제는 혼란만 가중할 뿐이므로 완전히 제거
좋다.

있으나 마나 한 주석

너무 당연한 사실을 언급하며 새로운 정보를 제공하지 못하는 주석.

첫번째 예제

```
/**
 * 기본 생성자
 */
constructor(num: Int, name: String) {
    AnnualDateRule(num, name)
}

/**
 * 월 중 일자
 */
private val dayOfMonth: Int = 0

/**
 * 월 중 일자를 반환한다.
 * @return 월 중 일자
 */
fun getDayOfMonth(): Int {
    return dayOfMonth
}
```

저렇게 당연한 주석은 지나친 참견이라 개발자가 주석을 무시하는 습관에 빠지게 된다. 코드를 읽으며 자동으로 주석을 건너뛴다. 결국 **코드를 변경하
은 거짓말을 하게 된다.**

두번째 예제

```
fun startSending() {
    try {
        doSending()
    } catch (_: SocketException) {
        // 정상. 누군가 요청을 멈췄다.
    } catch (_: Exception) {
        try {
            response.add(ErrorResponder.makeExceptionString)
            response.closeAll()
        } catch (_: Exception) {
            // ?! 이게 뭐야!
        }
    }
}
```

```
fun startSending() {
    try {
        doSending()
    } catch (_: SocketException) {
        // 정상. 누군가 요청을 멈췄다.
    } catch (_: Exception) {
        addExceptionAndCloseResponse()
    }
}

private fun addExceptionAndCloseResponse() {
    try {
        response.add(ErrorResponder.makeExceptionString)
        response.closeAll()
    } catch (_: Exception) {

    }
}
```

있으나 마나한 주석을 달려는 유혹에서 벗어나자. 코드를 정리하자. 더 나으며 행복한 프로그래머가 되는 지름길이다.

무서운 잡음

javadocs로 문서를 제공해야 된다라는 잘못된 욕심으로 작성된 주석.

아래 javadocs에서 제공하는것이 뭘까?

정답: 없다.

단순히 문서를 제공해야 된다라는 욕심에 탄생한 잡음일 뿐이다.

```
/** The name. */
private var name: String = ""

/** The version. */
private var version: String = ""

/** The licenseName. */
private var licenseName: String = ""

/** The version. */
private var info: String = ""
```

심지어 info의 경우 볻북으로 인한 오류도 있다.

함수나 변수로 표현할 수 있다면 주석을 달지마라

```
// 전역 목록 <smodule>에 속하는 모듈이 우리가 속한 하위 시스템에 의존하는가?
if (smodule.getDependSystems().contains(subSysMod.getSubSystem()))

---

val moduleDependees: ArrayList<String> = smodule.getDependSystems()
```

위치를 표현하는 주석

소스파일에서 특정 위치를 표시하는 주석.

```
// Actions ////////////////////////////////
```

극히 드물지만 위와 같은 배너 아래에 특정 기능을 모아놓았을 때 유용한 경우도 있긴하다. 하지만 위 형태의 주석은 가독성만 해치므로 제거해야 마땅히 연속적인 "///..."와 같은 잡음은 제거하는게 좋다.

닫는 괄호에 주석

닫는 괄호에 달아놓는 주석.
중첩이 심하고 장황한 함수에 달면 의미가 있을 수 있다. 하지만 우리가 선호하는 작고 캡슐화된 함수에서는 잡음일 뿐이다.
닫는 괄호에 주석을 달아야 겠다라는 생각이 든다면 대신 함수를 줄이도록 시도하자.

```
fun main() {
    try {
        while() {
            code ...
            code ...
            code ...
        } // while
        println("hi")
        println("good")
        println("morning")
    } // try
    catch (e: Exception) {
        code ...
    } // catch
} // main
```

주석으로 처리한 코드

주석으로 처리한 코드만큼 밉살스러운 관행도 드물다. 코드에 주석처리하고 남기지 말자!

주석 처리한 코드는 다른 사람들이 지우기를 주저한다.

- 이유가 남겨서 놓았으리라.
- 중요하니깐 지우면 안된다.

이렇게 되면 질 나쁜 와인병 바닥에 앙금이 쌓이듯 쓸모 없는 코드가 쌓여가게 된다.
현재 우리는 우수한 소스 코드 형상 관리 시스템을 사용한다. 이제 주석 처리할 필요가 없다. 그냥 코드를 삭제해라. 잃어버릴 염려는 없다.

HTML 주석

```
/**
* <p/>
* ...
* </P>
*/
```

전역 정보

주석을 정 달아야 한다면 근처에 있는 코드에 대해서만 기술하라.
코드 일부에 주석을 달면서 시스템의 전반적인 정보를 담아서 기술하지 마라.

```
/**
* 적합성 테스트가 작동하는 포트: 기본값은 8082.
*/
fun setFitnessPort(fintnessePort: Int) {
    this.fitnessePort = fintnessePort
}
```

이와 같은 주석이 달린 함수에서 기본 포트값은 전혀 통제하지 못하고 있다. 즉, 주석에서 설명하는 포트는 시스템 어딘가에 있는 다른 함수를 설명하는
이다. 포트 기본값이 변하더라도 위 주석은 변하리 없다.

너무 많은 정보

```
/*
RFC 2045 - Multipurpose Internet Mail Extension
6.8.  Base64 Content-Transfer-Encoding
The encoding process represents 24-bit groups of input bits as output
strings of 4 encoded characters.  Proceeding from left to right, a
24-bit input group is formed by concatenating 3 8bit input groups.
These 24 bits are then treated as 4 concatenated 6-bit groups, each
of which is translated into a single digit in the base64 alphabet.
When encoding a bit stream via the base64 encoding, the bit stream
must be presumed to be ordered with the most-significant-bit first.
That is, the first bit in the stream will be the high-order bit in
the first 8bit byte, and the eighth bit will be the low-order bit in
the first 8bit byte, and so on.
*/
class Base64Coder {
    fun encoding() {
        ...
    }
    fun decoding() {
        ...
    }
}
```

위 처럼 주석에다가 흥미로운 역사, 관련 없는 정보를 장황하게 늘어놓지 마라. base64를 인코딩 / 디코딩 하는 함수에 저런 내용은 불필요하다.

모호한 관계

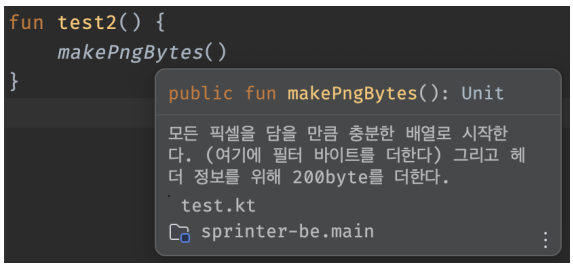
```
 * 그리고 헤더 정보를 위해 200byte를 더한다.
 */
fun makePngBytes{
    val pngBytes = ByteArray(((this.width) + 1) * this.height * 3 + 200)
}
```

위 주석에서 말하는 필터 바이트가 무엇인지 알 수 있는가? +1을 의미하는지, * 3을 의미하는지, 둘다 인지?
한 픽셀이 한 바이트인가? 200을 추가하는 이유는?

주석을 다는 이유는 코드만으로 설명이 부족해서이고, 주석에서 다시 설명을 요구하는 안타까운 형태이다.

함수헤더

함수 헤더 작성하지 마라.



짧은 함수는 긴 설명이 필요없다. 짧고 한 가지만 수행하며 이름을 잘 붙인 함수가 주석으로 헤더를 추가한 함수보다 훨씬 좋다.

비공개 코드에서 Javadocs

공개하지 않을 코드라면 Javadocs는 쓸모가 없다. 유용하지도 않고 Javadocs가 요구하는 형식으로 인해 코드만 보기 싫고 산만해진다.

'**기술 학습** > **클린 코드**' 카테고리의 다른 글

로버트 C. 마틴의 클린코드 정리 _ 5장 (형식 맞추기) (0)

로버트 C. 마틴의 클린코드 정리 _ 1~3장 (깨끗한 코… (0)

비밀글

Related Articles more

로버트 C. 마틴의 클린코드 정리 _ 5장 (형식 맞추기)
2023.07.11
기술 학습/클린 코드

로버트 C. 마틴의 클린코드 정리 _ 1~3장 (깨끗한 코드, 의미 있는 이름, ...
17:00:41
기술 학습/클린 코드

최근에 올라온 글
아파치 카프카 애플리케이션 프로…
아파치 카프카 애플리케이션 프로…
로버트 C. 마틴의 클린코드 정리 _ …
로버트 C. 마틴의 클린코드 정리 _ …

최근에 달린 댓글
우왓 감사합니당 😄
안녕하세요 :) java, 스프링 부트로 …
안녕하세요. 구름님 개발환경이 궁…
구름님 혹시 개발 툴 어떤거 쓰셨…

TAG more
마우스 동유럽 류블라냐
마우스 패드 레이저
크로아티아 부다페스트
키보드 손목 … mx master s…