주석

synopsis

> 나쁜 코드에 주석을 달지 마라. 새로 짜라.
> - 브라이언 W. 커니핸, P.J. 플라우거

주석을 한다는 것은

- 코드로 **의도를 표현하지 못한 것에 대한** 실패를 만회하기 위해 사용
  - 주석 없이 자신을 표현할 방법이 없는가?
  - (개인적 생각) 주석에 의존한다? 결합도가 높아진다? 👊🖐

  - (개인적 생각) A니까 B다는 아니지만, 의존성이 높아진다는 건 어딜가든 불편함
- 이럴 바에는 그냥 코드를 다시 짜는 게 낫다!

코드는 변화하고 진화한다.

그러면서 주석도 변해야 하는데, 현실적으로 그를 따라가지 못하는 경우가 많다.

```
MockRequest request;
private final String HTTP_DATE_REGEXP =
  "[SMTWF][a-z]{2}\\, \\s[0-9]{2}\\s[JFMASOND][a-z]{2}\\s"
+ "[0-9]{4}\\s[0-9]{2}\\:[0-9]{2}\\:[0-9]{2}\\sGMT";
private Response response;
private FitNesseContext context;
private FileResponder responder;
private Locale saveLocale;
// Example: "Tue, 02 Apr 2003 22:18:49 GMT"
```

`HTTP_DATE_REGEXP`와 주석 사이에 다른 인스턴스 변수 추가 가능성 높음

→ 주석을 유지할 필요가 없어짐

- 부정확한 주석은 아예 없는 주석보다 훨씬 더 나쁘다
- 진실은 코드에 있다. 왜? 실행되니까.
  - 주석이 꼭 필요하다고 해도 줄이려는 노력이 필요
  - 단, 스프링의 `어노테이션(Annotation, @)`은 예외

주석은 나쁜 코드를 보완하지 못한다

- 코드에 주석을 추가하는 일반적인 이유는 코드 품질이 나쁘기 때문이다?
    - 코드의 품질이 나빠서 주석을 달기보다는, 코드를 정리하는 게 Better

코드로 의도를 표현하라!

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

> 개인적으로 위의 코드를 보면 주석 한 번 읽고, 밑의 코드를 읽게 되는 것 같습니다.
> 하지만, 아래 코드는 메서드명 만으로도 의도를 파악할 수 있는 것 같습니다.

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

책에서는 위의 코드를 예시로 들면서 **주석으로 달려는 설명을 함수로 만들어도 충분히 표현이 가능**하다고 함

좋은 주석

법적인 주석

- (개인적 생각) 코드는 변화해야하는 주체

- 주석은 내버려둬도 코드가 실행되는 것 자체에는 문제 없음
  - 이러한 **자주 변하지 않았을 때 도움이 되는 특징**을 잘 활용할 수 있는 것이 **저작권 정보와 소유권 정보** 등이 있음
  - 물론, 무조건 법적인 정보일 필요 X
  - 표준 라이선스나 외부 문서를 참조하도록 유도할 수 있음

```
// Copyright (C) 2003, 2004, 2005 by Object Mentor, Inc. All rights reserved.
// GNU General Public License 버전 2 이상을 따르는 조건으로 배포한다.
```

- 위 소스는 **모든** 소스 파일 첫머리에 추가된 상태
  - 이는 IDE에서 최소화(축소)해서 코드만 표시 가능 (EX: IntelliJ IDEA)

정보를 제공하는 주석

```
// 테스트 중인 Responder 인스턴스를 반환한다.
protected abstract Responder responderInstance();
```

- `interface`나 `abstract class`의 경우에는 메서드 안의 내용물이 없음
  - 추상 메서드가 무엇을 반환할지를 알려주면, **상속**이나 **구현**할 때 도움이 될 수 있음
  - 여담으로, 함수 이름을 responderBeingTested로 바꾸면 주석이 필요 없어짐

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

정규 표현식일 경우에는 (개인적 생각) 코드로 한번에 알아보기 힘들 수 있음

이는 주석으로

> 시각, 날짜는 kk:mm:ss EEE, MMM dd, yyyy 형식이다

라고 표현해도 무방

하지만, 이는 위 예시와 마찬가지로

`SimpleDataFormat.format` 등의 시각과 날짜를 반환하는 **메서드를 활용**하면 주석을 제거하여 리팩토링이 가능

의도를 설명하는 주석

두 객체를 비교할 때, 다른 어떤 객체보다 자기 객체에 높은 우선순위를 주기로 결정하는 소스

```
public int compareTo(Object o) {
  if(o instanceof WikiPagePath) {
    WikiPagePath p = (WikiPagePath) o;

    String compressedName = Stringutil.join(names, "");
    String compressedArgumentName = StringUtil.join(p.names, "");
    String compressedName.compareTo(compressedArgumentName);
  }
  return 1;  // 옳은 유형이므로 정렬 순위가 더 높다.
}
```

- `compareTo()`만 봐서는 **재정의했을 때 어떤 기능을 할지 모를 수 있고**
- **1을 반환했을 때가 어떤 상황인지** 한번에 파악이 되지 않을 수 있음

이는 반환값 옆에 주석을 작성하여 설명할 수 있음

```
public void testConcurrentAddWidgets() throws Exception {
  WidgetBuilder widgetBuilder =
    new WidgetBuilder(new Class[]{BoldWidget.class});
  String text = "'''bold text'''";
  ParentWidget parent =
    new BoldWidget(new MockWidgetRoot(), "'''bold text'''");
  AutomicBoolean failFlag = new AutomicBoolean();
  failFlag.set(false);

  // 스레드를 대량 생성하는 방법으로 어떻게든 경쟁 조건을 만들려 시도한다.
  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());
}
```

- 주석이 없었다면, 경쟁 시도를 일부러 만들려는 개발자의 의도를 파악하는데 시간이 많이 소모될 수 있음

의미를 명료하게 밝히는 주석

- 모호한 인수나 반환값은 **그 자체를 명확하게 만드는 게** Best
- 표준 라이브러리 등 변경하지 못하는 코드에 속하는 인수나 반환값을 사용할 때는 주석을 사용하여 의미를 명확하게 하는 것이 유용

```
public void testCompareTo() throws Exception {
  WikiPagePath a = PathParser.parse("PageA");
  WikiPagePath ab = PathParser.parse("PageA.PageB");
  WikiPagePath b = PathParser.parse("PageB");
  WikiPagePath aa = PathParser.parse("PageA.PageA");
  WikiPagePath bb = PathParser.parse("PageB.PageB");
  WikiPagePath ba = PathParser.parse("PageB.PageA");

  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
}
```

- 하지만, 이 부분에서도 책에서는 **주석이 올바른지 검증하기 쉽지 않기 때문**에
  - 의미를 명료하게 밝히는 주석이 필요하면서
  - 동시에 주석이 위험하다

  고 말함

결과를 경고하는 주석

- 특정 테스트를 꺼야 하는 이유를 설명하는 주석

```
// 여유 시간이 충분하지 않다면 실행하지 마십시오.
public void _testWithReallyBigFile() {
  writeLinesToFile(10000000);

  response.setBody(testFile);
  response.readyToSend(this);
  String responseString = output.toString();
  assertSubString("Content-Length: 1000000000", responseString);
  assertTrue(bytesSent > 1000000000);
}
```

- 주석으로 다른 프로그래머에게 이 테스트를 할 시 시간이 많이 걸린다는 걸 알려줌
- 테스트 코드의 프레임워크로 `JUnit5`를 많이 쓰므로 주석을 쓰지 않고 `@Ignore`등 다른 어노테이션을 쓰면 주석을 제거하여 더 클린한 코드로 바꿀 수 있음
  - 예) `@Ignore("실행이 너무 오래 걸린다.")`
  - ~~그런데 어노테이션도 주석 아닌가~~

```
public static SimpleDateFormat makeStandardHttpDateFormat() {
  // SimpleDateFormat은 스레드에 안전하지 못하다.
  // 따라서 각 인스턴스를 독립적으로 생성해야 한다.
  SimpleDateFormat df = new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss z");
  df.setTimeZone(TimeZone.getTimeZone("GMT"));
  return df;
}
```

- SimpleDateFormat의 특성상 정적 초기화를 사용하려고 하는데, 주석을 사용하여 혹여 위의 코드를 리팩토링 한답시고 다른 프로그래머가 고치게 되는 우를 범하는 것을 방지
  - 동시성 환경에서 SimpleDateFormat을 쓸 때 왜 주의해야하나요?!

동시성 환경에서 SimpleDateFormat 사용시 주의가 필요하다.

개요웹은 기본적으로 동시성을 기반으로 하지만 의외로 우리 개발자들은 동시성에 취약한 코드를 작성하곤 한다.책에도 종종 소개되고 본인도 경험해본, 심지어 최근 회사 레거시 프로젝트 코드에서도 발견한

https://css-org.github.io/2016/11/16/dangerous-simpledateformat/

[Java(자바)] SimpleDateFormat 사용법, 멀티쓰레드 환경에서의 예제

Java를 이용해서 프로그래밍을 하다보면 날짜 데이터를 다뤄야 할 때가 많다. 날짜를 쉽게 파싱하고 다루기 위해 SimpleDateFormat 클래스를 많이 사용한다. 이번 포스트에서는 SimpleDateFormat 클래스의 사용 예제와 멀티쓰레드에서

https://soft.plusblog.co.kr/18

TODO 주석

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

- 당장 구현은 어렵지만, 필요하다 여길 때 유용하게 쓸 수 있음
- 종류

```
// TODO 더 이상 필요 없는 기능을 삭제하라는 알림
// TODO 누군가에게 문제를 봐달라는 요청
// TODO 더 좋은 이름을 떠올려달라는 부탁
// TODO 앞으로 발생할 이벤트에 맞춰 코드를 고쳐달라는 주의
```

- 다만, 이러한 TODO 주석도 많으면 좋지 않으므로, 주기적으로 점검하여 없애면 좋음
- IDE를 활용하여 주석이 어디있는지 다 확인할 수 있음

중요성을 강조하는 주석

```
String listItemContent = match.group(3).trim();
// 여기서 trim은 정말 중요하다. trim 함수는 문자열에서 시작 공백을 제거한다.
// 문자열에 시작 공백이 있으면 다른 문자열로 인식되기 때문이다.
new ListItemWidget(this, listItemContent, this.level + 1);
return buildList(text.substring(match.end());
```

- trim()이라는 메서드를 붙여야 하는 상황이라면?
- EX) 아 이 기능을 구현할 때는 문자열에 공백이 안 오게끔 주의를 기울여야겠다!

공개 API에서 Javadocs(공식문서)

iterator

`Iterator<E> iterator()`

Returns an iterator over the elements in this list in proper sequence.

Specified by:

`iterator` in interface `Collection<E>`

Specified by:

`iterator` in interface `Iterable<E>`

Returns:

an iterator over the elements in this list in proper sequence

Iterator 에 대해서

자바에는 인터페이스로 정의된 Iterator가 있습니다. 이것은 무엇일까요.. 어떤 역할을 할까요..? Iterator는 반복자라고 해석이 되어있습니다. 메서드도 딸랑 3개가 끝입니다. hasNext(), next(), remove() hasNext()는 다음에 반

https://invincure.tistory.com/entry/Iterator-에-대해서

static <E> List<E>	of()	Returns an unmodifiable list containing zero elements.
static <E> List<E>	of(E e1)	Returns an unmodifiable list containing one element.
static <E> List<E>	of(E... elements)	Returns an unmodifiable list containing an arbitrary number of elements.
static <E> List<E>	of(E e1, E e2)	Returns an unmodifiable list containing two elements.
static <E> List<E>	of(E e1, E e2, E e3)	Returns an unmodifiable list containing three elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4)	Returns an unmodifiable list containing four elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4, E e5)	Returns an unmodifiable list containing five elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4, E e5, E e6)	Returns an unmodifiable list containing six elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)	Returns an unmodifiable list containing seven elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)	Returns an unmodifiable list containing eight elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)	Returns an unmodifiable list containing nine elements.
static <E> List<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)	Returns an unmodifiable list containing ten elements.

- 물론, 공식문서도 사람이 작성하는 것이기 때문에 잘못된 정보를 줄 수는 있다.
- (개인적 생각) 공식문서는 설명이 더 많아졌으면 좋겠다…

좋은 주석은 어떤 특징을 가지고 있을까?

- 간결하다
- 코드와 관련된 설명을 담는다
- 자유롭게 사용하되 지나치게 사용하지 않는다

> 코드베이스가 진화할수록 과거에 내린 결정의 맥락을 남겨두는 것이 도움이 됩니다.

> 코드는 결코 완벽하지 않기에 코드 자체만으로 문서화가 되는 경우는 거의 없습니다.

> 코드 주석은 코드가 무슨 일을 하는지에 대한 혼란과 모호성을 줄여 주고 코드 자체에 없는 유용한 맥락과 정보를 제공해 줍니다.

> 혼자 개발하는 프로젝트에서도 자신이 작성한 코드에 다시 적응하고 방향을 잡는 데 도움을 줍니다.

출처

Docs for Developers 기술 문서 작성 완벽 가이드 p. 55

Docs for Developers 기술 문서 작성 완벽 가이드

소프트웨어 개발자를 안내하는 테크니컬 라이팅을 위한 실무 노트 우아한형제들, 카카오, AWS, LINE Plus, 쿠팡, NHN, 데브시스터즈, 넷마블 등 국내 테크니컬 라이터 11인 인터뷰 특별 수록!

https://m.hanbit.co.kr/store/books/book_view.html?p_code=B8810771470