Clean Code Seminar #2

우 여 명

우리 모두 리얼리스트가 되자.
그러나 가슴 속에는 불가능한 꿈을 가지자.

체 게바라

Index

- #4. 주석

주석은 나쁜코드를 보완하지 못한다.

코드로 의도를 표현하라

좋은 주석

나쁜 주석

#4. 주석

나쁜 코드에 주석을 달지 마라. 새로 짜라.

– 브라이언 W. 커니핸, P.J. 플라우거

#4. 주석

주석은 필요악!

- 프로그래밍 언어 자체가 표현력이 풍부하다면,
- 우리에게 프로그래밍 언어를 치밀하게 사용해 의도를 표현할 능력이 있다면,
- 주석은 (거의) 필요하지 않다.
- 즉, 우리는 코드로 표현하지 못했기 때문에, 그 실패를 만회하기 위해서 주석을 사용한다.

#4. 주석

주석은 거짓말쟁이

- 주석은 시간이 지날 수록 코드와 멀어진다.
  그리고 결국에는 완전히 엉뚱한 소리를 한다.

- 왜냐하면, 우리는 주석까지 유지보수 할 수 없다.

```
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;
//Excample : "Tue, 02 Apr 2003 22:18:49 GMT
```

- HTTP_DATE_REGEXP에 대한 주석인데…

#4. 주석

진실은 한곳에만 존재한다.

바로 코드다.

#4. 주석

코드로 의도를 표현하라

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

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

#4. 주석

(그나마) 좋은 주석

용인 가능한, 그나마 글자 값을 하는 주석 소개

/ * 그러나 정말 좋은 주석은
주석을 달지 않을 방법을 찾아낸 주석이다 */

#4. 주석 - (그나마) 좋은 주석

법적인 주석

- 저적권 정보, 소유관 정보,
  표준 라이센스나 외부 라이센스 문서 참조 정보

- 코드의 변경과 무관한 주석

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

#4. 주석 - (그나마) 좋은 주석

정보를 제공하는 주석

[CASE 1]

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

protected abstract Responder responderBeingTested();
```

[CASE 2]

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

- 문자열에서 시각과 날짜를 변환하는 클래스 또는 메서드를 따로 만든다.

#4. 주석 - (그나마) 좋은 주석

의도를 설명하는 주석

[CASE 1]

```
public int compareTo(Object o) {
  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; // 오른쪽 유형이므로 정렬 순위가 더 높다.
}
```

- A.compareTo(B) < 0 이면 A 순위가 높다.
- 다른 어떤 객체(WikiPagePath) 보다 자기 객체에 높은 우선순위를 주기로 함.

#4. 주석 - (그나마) 좋은 주석

의도를 설명하는 주석

[CASE 2]

```
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'''");
  AtomicBoolean failFlag = new AtomicBoolean();
  failFlag.set(false);

  // 스레드를 대량으로 생성하는 방법으로 어떻게든 경쟁 조건을 만들려고 시도한다
  for (int i = 0; i < 25000; i++) {
    WidgetBilderThread widgetBilderThread =
            new WidgetBuilderThread(widgetBuilder, text, parent, failFlag);
    Thread thread = new Thread(widgetBilderThread);
    thread.start();
  }
  assertEquals(false, failFlag.get());
}
```

- 아! 스레드로 동시성 테스트를 하려는 구나!

#4. 주석 - (그나마) 좋은 주석

의미를 명료하게 밝히는 주석

[CASE 1]

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

  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
}
```

- 헤깔리는 compareTo 메서드에 대한 주석
- 단점 : 주석이 올바른지 검증하기 쉽지 않다. 즉, 주석을 믿을 수 있을까?

#4. 주석 - (그나마) 좋은 주석

결과를 경고하는 주석

[CASE 1]

```
// 여유 시간이 충분하지 않다면 실행하지 마십시오
public void _testWithReallyBigFile() {
  writeLinesToFile(1000000000);
  response.setBody(testFile);
  response.readyToSend(this);
  String responseString = output.toString();
  assertSubString("Content-Length: 1000000000", responseString);
  assertTrue(bytesSent > 1000000000);
}
```

- 더 좋은 방법 : @Ignore 어노테이션 사용

#4. 주석 - (그나마) 좋은 주석

결과를 경고하는 주석

[CASE 2]

```
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;
}
```

- javadocs에 SimpleDateFormat의 Thread safe한지 설명이 있다.
  http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html

- 여기서 주석은 합리적인편이다. 왜냐하면 Thread safe 정보는 놓치기 쉽다.

#4. 주석 - (그나마) 좋은 주석

중요성을 강조하는 주석

[CASE 1]

```
String listItemcontent = match.group(3).trim();
// 여기서 trim은 굉장히 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()));
```

- 코드만으로는 중요성이 부각되지 않을 때!

#4. 주석 - (그나마) 좋은 주석

기타 괜찮은 주석

- (잘 계획되고 관리되는) TODO 주석
  - IDE에서 자동으로 생성되는 TODO주석 (X)

- 공개 API Javadocs

#4. 주석

나쁜 주석

대다수의 주석

허술한 코드를 지탱하고,

엉성한 코드를 변명하고,

미숙한 결정을 합리화하는 나쁜 주석

#4. 주석 - 나쁜 주석

주절거리는 주석

[CASE 1]

```
public void loadProperties() {
  try {
    String propertiesPath = propertiesLocation + "/" + PROPERTIES_FILE;
    FileInputStream propertiesStream = new FileInputStream(propertiesPath);
    loadedProperties.load(propertiesStream);
  } catch (IOException e) {
    // 속성 파일이 없다면 기본값을 모두 메모리로 읽어 들였다는 의미다.
  }
}
```

- 누가 언제 어떻게 읽어 들였다는 거지?
- 메모리에 없을 때만 가져오도록 수정.

#4. 주석 - 나쁜 주석

같은 이야기를 중복하는 주석

[CASE 1]

```
// this.closed가 true일 때 반환되는 유틸리티 메서드다.
// 타임아웃에 도달하면 예외를 던진다.
public synchronized void waitForClose(final long timeoutMillis)
throws Exception {
  if (!closed) {
    wait(timeoutMillis);
    if (!closed)
      throw new Exception("MockResponseSender could not be closed");
  }
}
```

- ! 때문에 헤깔린다.
- 코드만으로도 주석의 내용정도는 알 수 있다.

#4. 주석 - 나쁜 주석

같은 이야기를 중복하는 주석

[CASE 2]

```
/**
 * 이 컴포넌트의 프로세서 지연값
 */
protected int backgroundProcessorDelay = -1;

/**
 * 이 컴포넌트를 지원하기 위한 생명주기 이벤트
 */
protected LifecycleSupport lifecycle = new LifecycleSupport(this);
```

- 쓸모없고 중복된 javadocs

#4. 주석 - 나쁜 주석

오해할 여지가 있는 주석

[CASE 1]

```
// this.closed가 true일 때 반환되는 유틸리티 메서드다.
// 타임아웃에 도달하면 예외를 던진다.
public synchronized void waitForClose(final long timeoutMillis)
throws Exception {
  if (!closed) {
    wait(timeoutMillis);
    if (!closed)
      throw new Exception("MockResponseSender could not be closed");
  }
}
```

- 주석의 내용이 틀린건 아닌데 코드와 미묘하게 다르다.

- this.closed == true 이면 메서드가 반환된다.

- 단순히 타임아웃에 도달하면 예외를 던지는 것이 아니고,
  this.closed == false일때, 타임아웃을 기다렸다가 예외를 던진다.

#4. 주석 - 나쁜 주석

있으나 마나 한 주석

- 뻔한 주석은 지나친 참견이다. 그런데,

[CASE 1]

```
private void startSending() {
  try {
    doSending();
  } catch (SocketException e) {
    // 정상 누군가 요청을 멈췄다
  } catch (Exception e) {
    try {
      response.add(ErrorResponder.makeExceptionString(e));
      response.closeAll();
    } catch (Exception e1) {
      // 이게 뭐야
    }
  }
}
```

- 첫번째 캐치절의 주석은 적절해 보인다.(저자)

- add 혹은 closeAll 메서드에 throw Exception이 있어서,
  개발자는 짜증이 난 것 같다.

#4. 주석 - 나쁜 주석

있으나 마나 한 주석

- 있으나 마나 한 주석으로 분풀이 하지 말고 코드 구조를 개선해보자.

[CASE 1 - 리펙토링]

```
private void startSending() {
  try {
    doSending();
  } catch (SocketException e) {
    // 정상 누군가 요청을 멈췄다
  } catch (Exception e) {
    addExceptionAndCloseResponse(e);
  }
}

private void addExceptionAndCloseResponse(Exception e) {
  try {
    response.add(ErrorResponder.makeExceptionString(e));
    response.closeAll();
  } catch (Exception e) {
  }
}
```

#4. 주석 - 나쁜 주석

함수나 변수로 표현할 수 있다면 주석을 달지 마라

[CASE 1]

```
//전역 목록 <smodule>에 속하는 모듈이 우리가 속한 하위시스템에 의존하는가
if (smodule.getDependSubsystems().contains(subSysMod.getSubSystem())) {
}

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

- 혹시나 주석을 먼저 달고 그에 맞춰서 코드를 작성한다면,
  주석이 필요 없을 정도로 코드로 표현해야 한다.

- 위 방법은 프로그램의 가독성을 높이는 효과적인 방법중의 하나로,
  계산을 여러 단계로 나누고 중간 값으로 서술적인 변수 이름을 사용하는 방법이다.

#4. 주석 - 나쁜 주석

모호한 관계

[CASE 1]

```
/*
 * 모든 픽셀을 담을 만큼 충분한 배열로 시작한다(여기에 필터 바이트를 더한다).
 * 그리고 헤더 정보를 위해 200바이트를 더한다
 */
this.pngBytes = new byte[((this.width + 1) * this.height * 3) + 200];
```

- 필터바이트? +1 ? * 3 ?
- 200을 더하는 이유는?
- 주석 자체가 다시 설명을 요구한다.

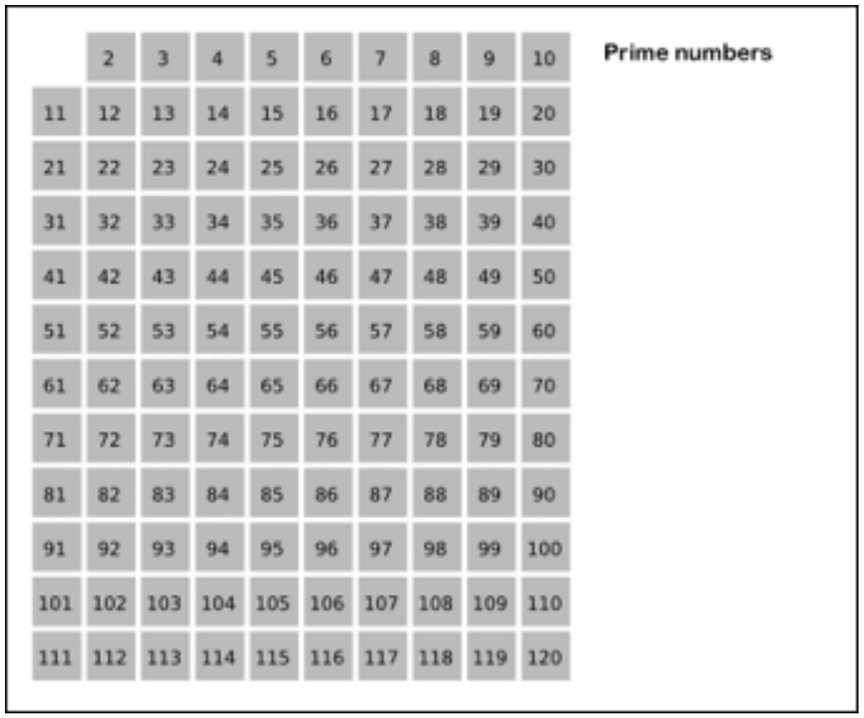
#4. 주석 - 나쁜 주석

기타

- 의무적으로 다는 주석 - 예) javadocs 파라미터
- 이력을 기록하는 주석 -> 형상 관리 툴로
- 위치를 표시하는 주석 / 닫는 괄호에 다는 주석
  -> 중첩이 심하고 장황하다는 의미이므로 리펙토링
- 공로를 돌리거나 저자를 표시하는 주석 / SR 주석
- 주석으로 처리한 코드
- Javadocs HTML 주석
- 함수 헤더
- 비공개 코드에서 javadocs

#4. 주석 - 연습

에라스토테네스의 체




```java
/**
 * 이 클래스는 사용자가 지정한 최대 값까지 소수를 생성한다.
 * 사용된 알고리즘은 에라스토테네스의 체다.
 * <p>
 * 에라스토테네스 : 기원전 276년에 리비아 키레네에서 출생,
 * 기원전 194년에 사망
 * 지구 둘레를 최초로 계산한 사람이자 달력에 윤년을 도입한 사람.
 * 알렉산드리아 도서관장을 역임.
 * <p>
 * 알고리즘은 상당히 단순하다.
 * 2에서 시작하는 정수 배열을 대상으로 2의 배수를 모두 제거한다.
 * 다음으로 남은 정수를 찾아 이 정수의 배수를 모두 지운다.
 * 최대 값의 제곱근이 될 때까지 이를 반복한다.
 *
 * @author Alphonse
 * @version 13 Feb 2002 atp
 */
public class GeneratePrimes {
  /**
   * @param maxValue 소수를 찾아낼 최대 값
   */
  public static int[] generatePrimes(int maxValue) {
    if (maxValue >= 2) // 유일하게 유효한 경우
    {
      // 선언
      int s = maxValue + 1; // 배열 크기
      boolean[] f = new boolean[s];
      int i;
      // 배열을 참으로 초기화
      for (i = 0; i < s; i++)
        f[i] = true;
      // 소수가 아닌 알려진 숫자를 제거
      f[0] = f[1] = false;
      // 체
      int j;
      for (i = 2; i < Math.sqrt(s) + 1; i++) {
        if (f[i]) // i가 남아있는 숫자라면 이 숫자의 배수를 구한다.
        {
          for (j = 2 * i; j < s; j += i)
            f[j] = false; // 배수는 소수가 아니다.
        }
      }
      // 소수 개수는
      int count = 0;
      for (i = 0; i < s; i++) {
        if (f[i])
          count++; // 카운트 증가
      }
      int[] primes = new int[count];
      // 소수 결과를 배열로 이동한다.
      for (i = 0, j = 0; i < s; i++) {
        if (f[i]) // 소수일 경우에
          primes[j++] = i;
      }
      return primes; // 소수를 반환한다.
    } else
      // maxValue < 2
      return new int[0]; // 입력이 잘못되면 비어 있는 배열을 반환한다.
  }
}
```

#4. 주석 - 연습

에라스토테네스의 체

```java
package util;

/**
 * 이 클래스는 사용자가 지정한 최대값까지 소수를 구한다.
 * 알고리즘은 에라스토테네스의 체다.
 * 2에서 시작하는 정수 배열을 대상으로 작업한다.
 * 처음으로 남아 있는 정수를 찾아 배수를 모두 제거한다.
 * 배열에 더 이상 배수가 없을 때까지 반복한다.
 */
public class PrimeGenerator {
  private static boolean[] crossedOut;
  private static int[] result;

  public static int[] generatePrimes(int maxValue) {
    if(maxValue < 2) {
      return new int[0];
    } else {
      uncrossIntegersUpTo(maxValue);
      crossOutMultiples();
      putUncrossedIntegersIntoResult();
      return result;
    }
  }

  private static void uncrossIntegersUpTo(int maxValue) {
    crossedOut = new boolean[maxValue+1];
    for (int i = 2; i< crossedOut.length; i++)
      crossedOut[i] = false;
  }


  private static void crossOutMultiples() {
    int limit = determineIterationLimit();
    for (int i = 2; i <= limit ; i++)
      if (notCrossed(i))
        crossOutMultiplesOf(i);
  }

  private static void crossOutMultiplesOf(int i) {
    for (int muliple = 2*i;
         muliple < crossedOut.length ;
         muliple += i)
      crossedOut[muliple] = true;
  }

  private static int determineIterationLimit() {
    // 배열에 있는 모든 배수는 배열 크기의 제곱근 보다 작은 소수의 인수.
    // 따라서 이 제곱근보다 더 큰 숫자의 배수는 제거할 필요가 없다.

    double iterationLimit = Math.sqrt(crossedOut.length)
    return (int) iterationLimit;
  }

  private static boolean notCrossed(int i) {
    return crossedOut[i] == false;
  }

  private static void putUncrossedIntegersIntoResult() {
    result = new int[numberOfUncrossedIntegers()];
    for(int j=0, i=2; i<crossedOut.length; i++) {
      if (notCrossed(i))
        result[j++] = i;
    }
  }

  private static int numberOfUncrossedIntegers() {
    int count = 0;
    for (int i=2; i<crossedOut.length; i++) {
      if(notCrossed(i)) {
        count++;
      }
    }
    return count;
  }
}
```

- cross out : (보통 틀린 단어에 위에) 줄을 긋다. 제외시키다.
- sqrt : square root, 제곱근 ex) 4의 제곱근 = 2
- multiple : (수학) 배수

#4. 주석

결론

- 주석은 필요악
- 표현할 수 있다면, 함수와 변수로 표현하자.
- 그럴 수 없을 경우에만 주석을 사용하고,
  관리를 잘 하자.