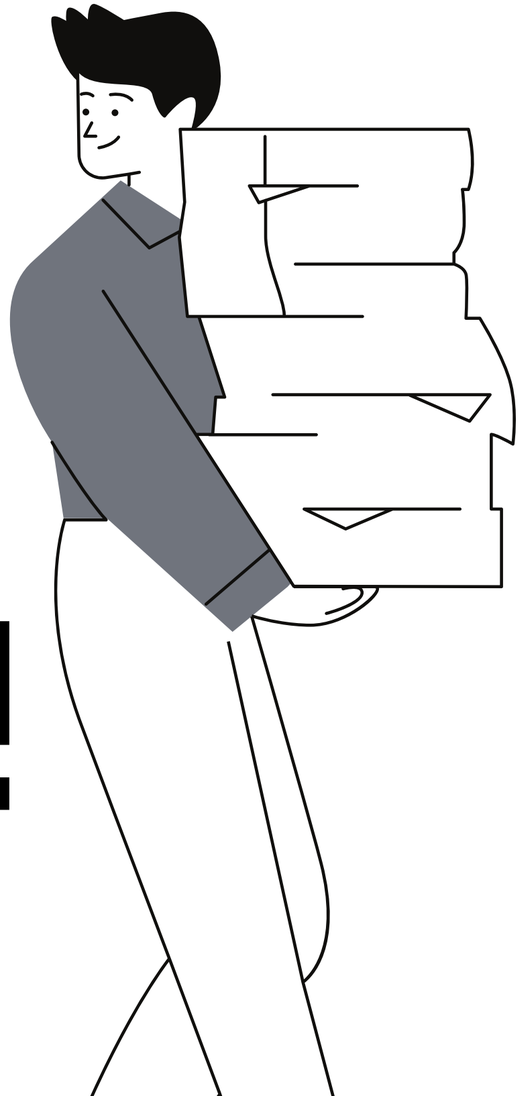
KonCC

2023년 7월 6일

함수 & 주석

클린코드 Chapter 3&4 - 김동민



오늘의 주제

3 함수

4 주석

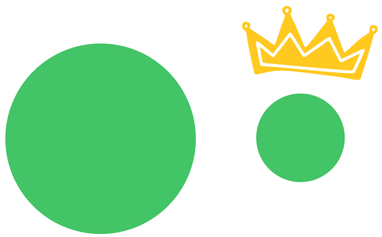
함수

함수

어떤 프로그램이든 가장 기본적인 단위는 함수이다.
읽기 쉽고 이해하기 쉬운 함수는 어떻게 작성해야하는가?

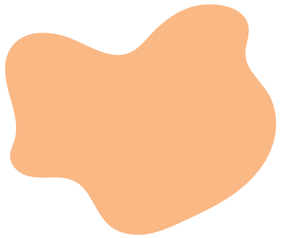
1

짝게 만들어라!



2

한 가지만 해라!



3

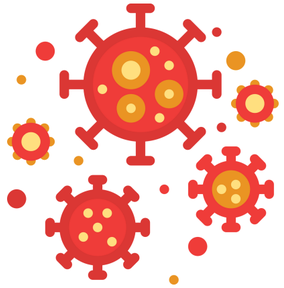
서술적인 이름을 사용하라!

4

함수 인수

5

부수 효과를 일으키지 마라!



작게 만들어라!

함수는 짧아야 한다.

Norminette

II.2 서식

- 들여쓰기는 네 칸 크기의 탭으로 이루어져야 합니다. 일반적인 공백 네 칸이 아니라 진짜 탭을 말합니다.
- 각 함수는 함수 자체의 중괄호를 제외하고 최대 25줄이어야 합니다.
- 각 줄은 주석을 포함해 최대 80자의 열 너비를 가집니다. 주의: 탭 들여쓰기는 한 열로 계산하지 않으며, 탭이 해당되는 공백의 수 만큼으로 계산됩니다.
- 각 함수는 줄 바꿈으로 구분해야 합니다. 모든 주석과 전처리기 명령은 함수 바로 위에 있을 수 있습니다. 줄 바꿈은 이전 함수 다음에 와야 합니다.
- 한 줄에 한 명령만이 존재할 수 있습니다.
- 빈 줄은 공백이나 탭 들여쓰기 없이 비어 있어야 합니다.
- 줄은 공백이나 탭 들여쓰기로 끝날 수 없습니다.
- 두 개의 연속된 공백이 있을 수 없습니다.
- 모든 중괄호나 제어 구조 뒤는 줄바꿈으로 시작돼야 합니다.
- 줄의 끝이 아니라면 모든 콤마와 세미콜론 뒤에는 공백 문자가 따라와야 합니다.
- 모든 연산자나 피연산자는 하나의 공백으로 구분해야 합니다.

II.3 함수

- 한 함수에는 최대 4개의 명명된 매개변수를 가질 수 있습니다.
- 인자를 받지 않는 함수는 "void"라는 단어를 인자로 명시적으로 프로토타입 돼야 합니다.
- 함수 프로토타입 안의 매개 변수는 명명되어야만 합니다.
- 각 함수는 빈 줄로 다음 함수와 구분되어야 합니다.
- 각 함수에서 5개를 초과하여 변수를 선언할 수 없습니다.
- 함수의 리턴은 괄호 사이에 있어야 합니다.
- 각 함수의 리턴 자료형과 함수 이름 사이에는 한 번의 탭 들여쓰기가 있어야 합니다.

```
int my_func(int arg1, char arg2, char *arg3)
{
    return (my_val);
}

int func2(void)
{
    return ;
}
```

작게 만들어라!

함수는 짧아야 한다.

블록과 들여쓰기

indent 레벨은 두 단계를 넘어선 안된다.

fsku.c — Xcode(으)로 열기

```
            while(fileSize > 0) {
                unsigned int *dPtr = (unsigned int *)(dataBlocks + inodeTable[inum*4+3] * 512 + cnt * 4);
                for(int i = 0; i < 60; i++) {
                    if (dataBitmap[i] == 0) {
                        inodeTable[inum*4+1]++;
                        dataBitmap[i] = 1;
                        *dPtr = i;
                        break;
                    }
                }
                for(int i = 0; i < 512; i++) {
                    dataBlocks[*dPtr * 512 + i] = fileName[0];
                    fileSize--;
                    if (fileSize == 0) break;
                }
                cnt++;
            }
        }
        else {
            unsigned int iPtr = inodeTable[inum*4+3];
            unsigned int *dPtrs = (unsigned int *)(dataBlocks + iPtr * 512);

            int temp = -1;
            while(oldSize > 512) {
                oldSize -= 512;
                temp++;
            }
            unsigned int dPtrTemp = dPtrs[temp];
            for(int i = oldSize; i < 512; i++) {
                dataBlocks[dPtrTemp  * 512 + i] = fileName[0];
                fileSize--;
                if (fileSize == 0) break;
            }
            if (fileSize > 0) {
                int cnt = temp + 1;
                while(fileSize > 0) {
                    for(int i = 0; i < 60; i++) {
                        if (dataBitmap[i] == 0) {
                            inodeTable[inum*4+1]++;
                            dataBitmap[i] = 1;
                            dPtrs[cnt] = i;
                            break;
                        }
                    }
                    for(int i = 0; i < 512; i++) {
                        dataBlocks[dPtrs[cnt] * 512 + i] = fileName[0];
                        fileSize--;
                        if (fileSize == 0) break;
                    }
                    cnt++;
                }
            }
        }
    }
}

void myRead() {
    unsigned char inum = findInRoot();
```

한 가지만 해라!

함수는 한 가지를 해야 한다. 그 한가지를 잘 해야 한다. 그 한가지만을 해야 한다.

함수 당 추상화 수준은 하나로!

A하려면 B, C를 해야한다.

B하려면 D해야한다.

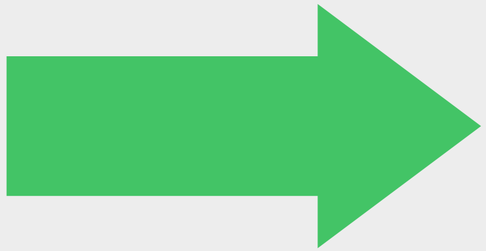
D하려면 F, G해야한다.

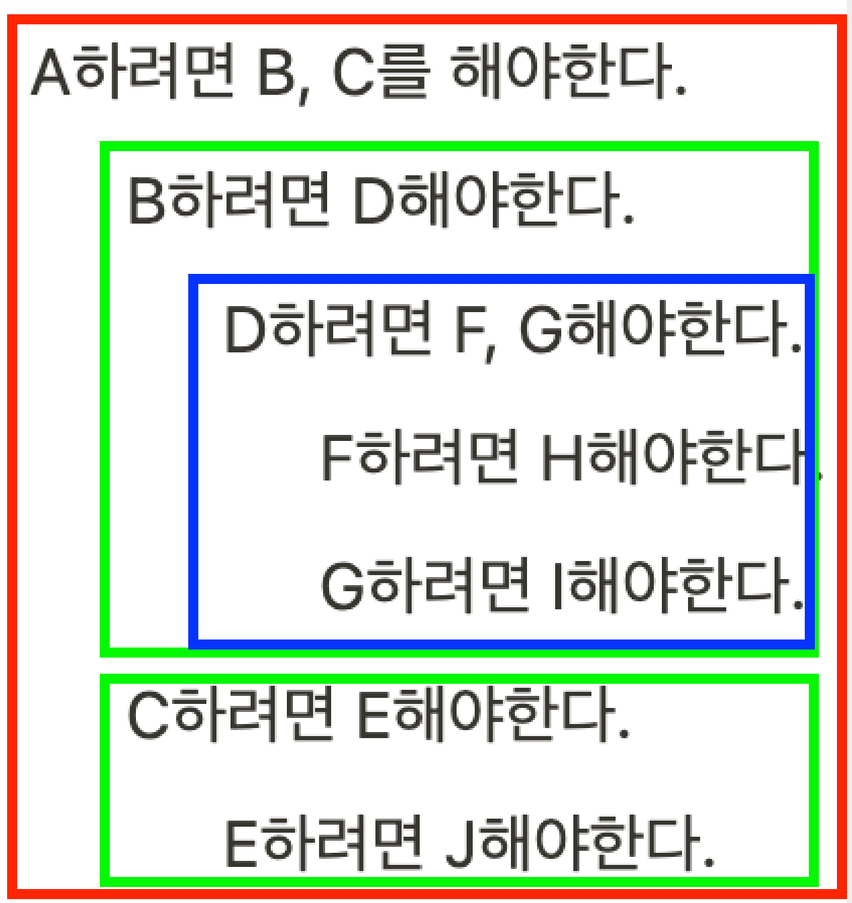
F하려면 H해야한다.

G하려면 I해야한다.

C하려면 E해야한다.

E하려면 J해야한다.





Switch 문

switch 문을 저차원 클래스에 숨기고 절대로 반복하지 않는다.

나쁜 코드

```
public Money calculatePay(Employee e) throws InvalidEmployeeType {
        switch (e.type) {
                case COMMISSIONED:
                        return calculateCommissionedPay(e);
                case HOURLY:
                        return calculateHourlyPay(e);
                case SALARIED:
                        return calculateSalariedPay(e);
                default:
                        throw new InvalidEmployeeType(e.type);
        }
}
```

Switch 문

switch 문을 저차원 클래스에 숨기고 절대로 반복하지 않는다.

좋은 코드

```
public abstract class Employee {
        public abstract boolean isPayday();
        public abstract Money calculatePay();
        public abstract void deliverPay(Money pay);
}
------------------
public interface EmployeeFactory {
        public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType;
}
------------------
public class EmployeeFactoryImpl implements EmployeeFactory {
        public Employee makeEmployee(EmployeeRecord r) throws InvalidEmployeeType {
                switch (r.type) {
                        case COMMISSIONED:
                                return new CommissionedEmployee(r) ;
                        case HOURLY:
                                return new HourlyEmployee(r);
                        case SALARIED:
                                return new SalariedEmploye(r);
                        default:
                                throw new InvalidEmployeeType(r.type);
                }
        }
}
```

서술적인 이름을 사용하라!

이름을 정할 때 시간이 많이 들어도 괜찮다. 이때 들이는 시간이 아무리 길어도 읽기 어렵게 짓고 훗날 다시 해석하느라 쓰는 시간보다는 짧다.

함수가 하는 일을 충분히 설명하까!

C# Copy

```
[System.Obsolete("The recommended alternative is to use the Membership APIs,
such as Membership.CreateUser. For more information, see
http://go.microsoft.com/fwlink/?LinkId=252463.")]
public static string HashPasswordForStoringInConfigFile (string password,
string passwordFormat);
```

함수 인수

인수는 적을수록 좋다. 인수는 개념을 이해하기 어렵게 만든다. 코드를 읽는 사람이 인수의 의미를 해석해야한다.

많이 쓰는 단항 형식

1. 인수에게 질문을 던지는 경우 `boolean fileExists("MyFile")`
2. 인수를 뭔가로 변환해 결과를 반환하는 경우 `InputStream fileOpen("MyFile")`
3. 이벤트 `passwordAttemptFailedNtimes(int attempts)`

위 경우들이 아니라면 단항 함수는 가급적 피하자!

함수 인수

인수는 적을수록 좋다. 인수는 개념을 이해하기 어렵게 만든다. 코드를 읽는 사람이 인수의 의미를 해석해야한다.

플래그 인수

```
// bad
render(boolean isSuite)

// good
renderForSuite();
renderForSingleTest();
```

함수 인수

인수는 적을수록 좋다. 인수는 개념을 이해하기 어렵게 만든다. 코드를 읽는 사람이 인수의 의미를 해석해야한다.

이항 함수

인수가 2개인 함수는 인수가 1개인 함수보다 이해하기 어렵다. 인수의 순서의 이해, 인수를 무시해야 하는지 판단하는 데에 시간이 소요된다. ⇒ 결국 문제를 일으킨다.

```
writeFiled(ouputStream, name)
```

⇒

```
outputStream.writeField(name)
```

가능하면 단항 함수로 변경!

함수 인수

인수는 적을수록 좋다. 인수는 개념을 이해하기 어렵게 만든다. 코드를 읽는 사람이 인수의 의미를 해석해야한다.

삼항 함수

함수 인수

인수는 적을수록 좋다. 인수는 개념을 이해하기 어렵게 만든다. 코드를 읽는 사람이 인수의 의미를 해석해야한다.

인수 객체

```
Circle makeCircle(double x, double y, double radius);
```

⇒

```
Circle makeCircle(Point center, double radius);
```

일부를 개념을 표현하여 클래스 변수로 선언하여 묶어주자!

함수 인수

인수는 적을수록 좋다. 인수는 개념을 이해하기 어렵게 만든다. 코드를 읽는 사람이 인수의 의미를 해석해야한다.

동사와 키워드

`write(name)` : '이름'이 무엇이든 '쓴다'.

`writeField(name)` : '이름'이 '필드'라는 사실이 분명하게 드러남.

`assertExpectedEqualsActual(expected, actual)`

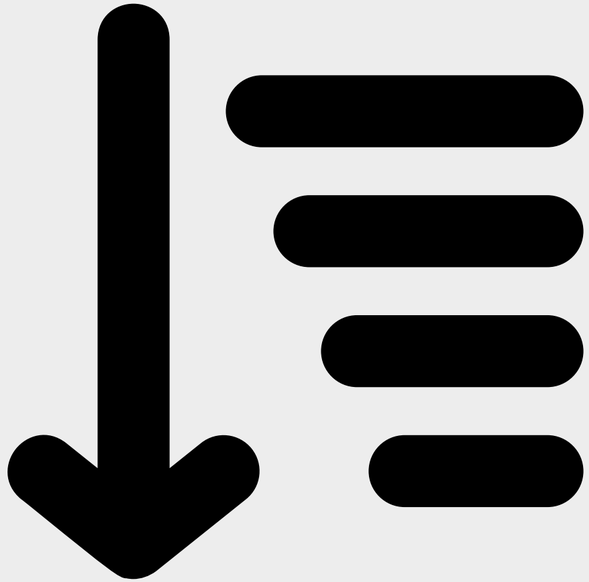
함수 이름에 인수 이름을 넣으면 인수 순서를 기억할 필요가 없어진다!

부수 효과를 일으키지 마라!

Side effect : 함수 내에서 함수 외부에 영향을 끼치는 것

부수 효과

시간적인 결합



순서 종속성

명령과 조회를 분리하라!

객체 상태를 변경하거나 객체 정보를 반환하거나 둘 중 하나

나쁜 코드

```
int nameSet(char **planeName, char *value);
```

```
if (nameSet(&plane1, "ABC123")) {
    sum++;
    manage();
}
```

명령과 조회를 분리하라!

객체 상태를 변경하거나 객체 정보를 반환하거나 둘 중 하나

좋은 코드

```
if (plane1) {
    nameSet(&plane1, "ABC123");
    sum++;
    manage();
}
```

오류 코드보다 예외를 사용하라!

try / catch

나쁜 코드

```
if (deletePage(page) == E_OK) {
  if (registry.deleteReference(page.name) == E_OK) {
    if (configKeys.deleteKey(page.name.makeKey()) == E_OK) {
      logger.log("page deleted");
    } else {
      logger.log("configKey not deleted");
    }
  } else {
    logger.log("deleteReference from registry failed");
  }
} else {
  logger.log("delete failed");
  return E_ERROR;
}
```

오류 코드보다 예외를 사용하라!

try / catch

좋은 코드

```
public void delete(Page page) {
        try {
                deletePageAndAllReferences(page);
        } catch (Exception e) {
                logError(e);
        }
}

private void deletePageAndAllReferences(Page page) throws Exception {
        deletePage(page);
        registry.deleteReference(page.name);
        configKeys.deleteKey(page.name.makeKey());
}

private void logError(Exception e) {
        logger.log(e.getMessage());
}
```

오류 코드보다 예외를 사용하라!

try / catch

의존성 자석

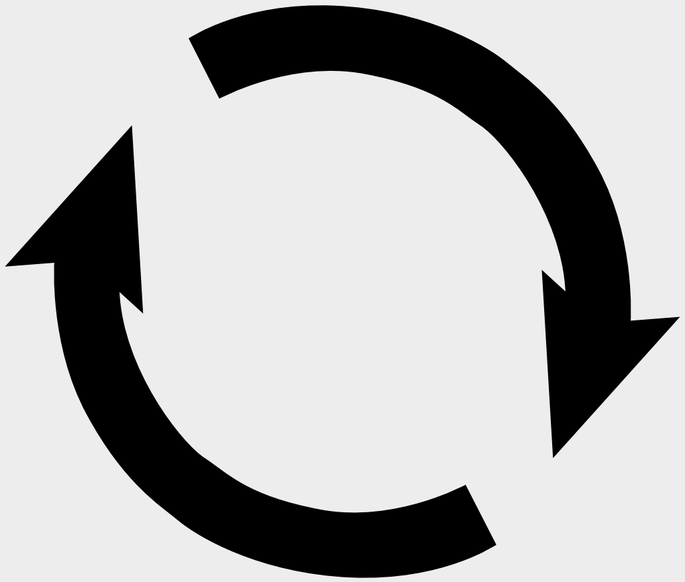
```
public enum Error {
        OK,
        INVALID,
        NO_SUCH,
        LOCKED,
        OUT_OF_RESOURCES,
        WAITING_FOR_EVENT;
}
```

새 오류코드 추가나 변경시 코스트 ↑

반복하지 마라! & 구조적 프로그래밍

중복은 독이다. & 모든 함수와 함수 내 모든 블록에 입구와 출구는 하나씩만 존재해야한다.

중복



구조적 프로그래밍

return문은 하나!

break, continue 사용 X

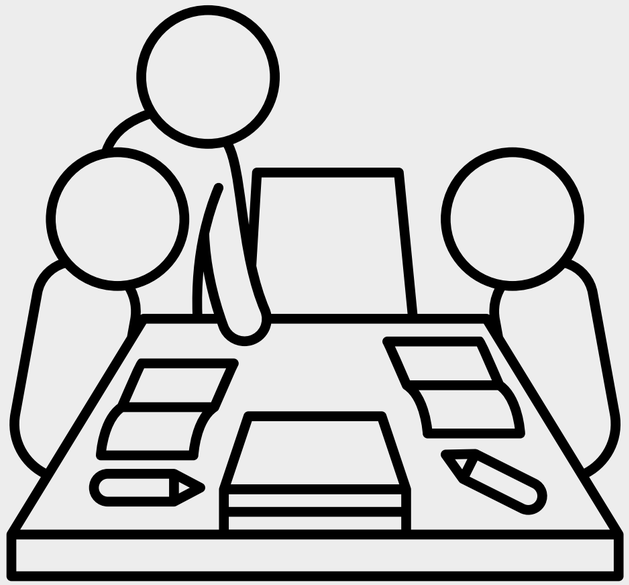
goto는 절대 사용 X

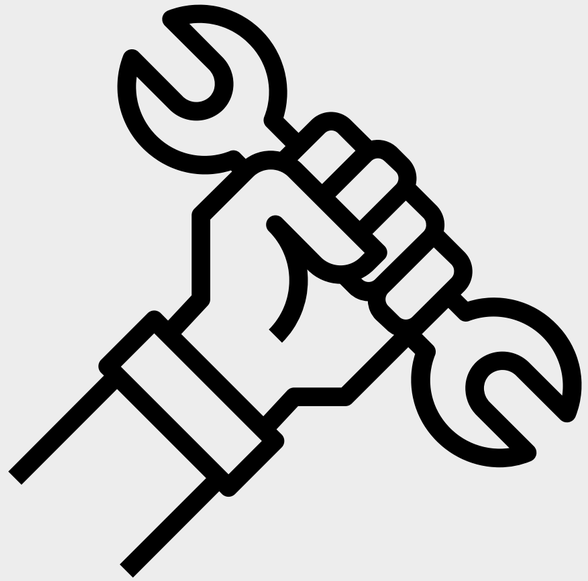
But, 함수의 크기가 작다면 사용 O

함수를 어떻게 짜죠?

처음부터 탁 짜내는 것이 가능한 사람은 없다.

하나씩 하나씩





결론

함수는 프로그램의 가장 기본적인 단위

시스템

시스템은 구현할 프로그램이 아니라 풀어갈 이야기이다.

주석

주석

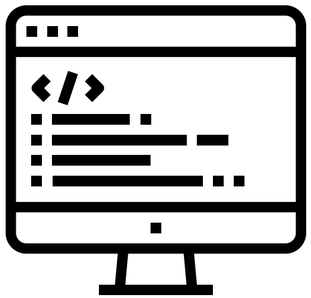
나쁜 코드에 주석을 달지 마라. 새로 짜라.
브라이언 W. 커니핸, P. J 플라우거

1

주석은 나쁜 코드를 보완하지 못한다

2

코드로 의도를 표현하라!

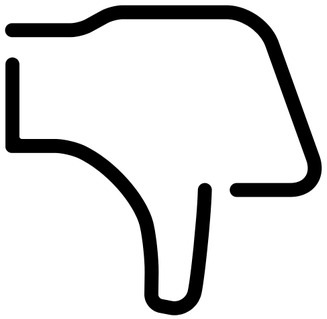


3

좋은 주석

4

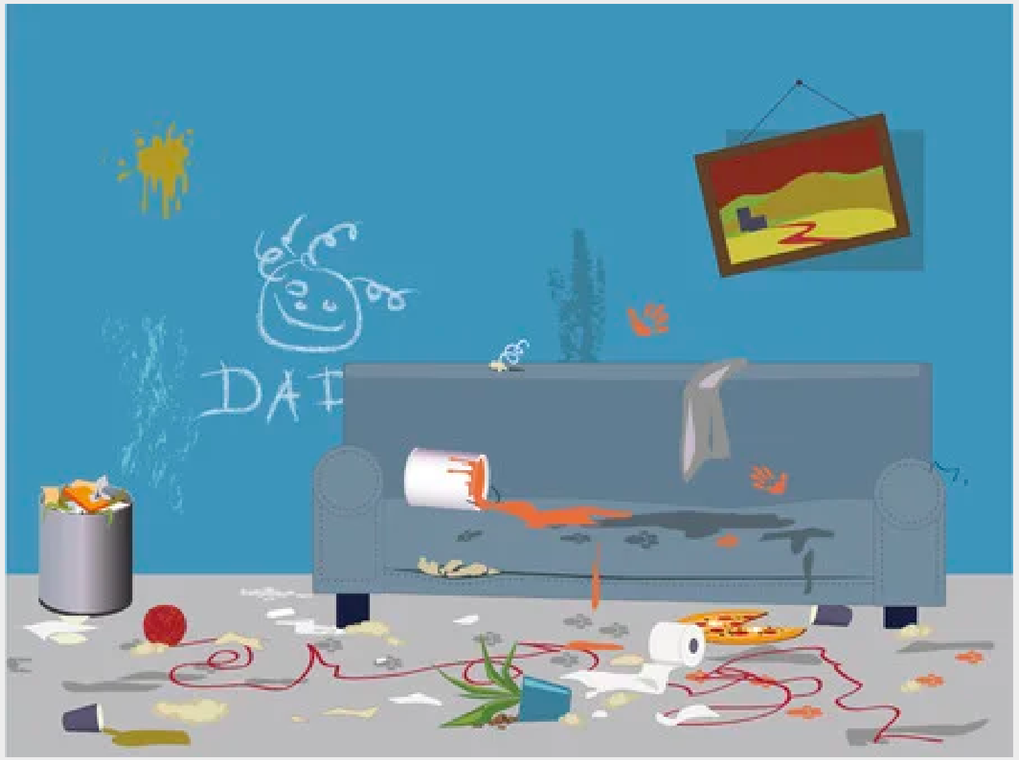
나쁜 주석



주석은 나쁜 코드를 보완하지 못한다

난장판을 주석으로 수습하려하지말고, 난장판을 수습하자!

코드 > 주석



코드로 의도를 표현하라!

코드만으로 대다수의 의도를 표현할 수 있다!

서술적인 코드

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

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

좋은 주석

Good Comment

- 법적인 주석
- 정보를 제공하는 주석
- 의도를 설명하는 주석
- 의미를 명료하게 밝히는 주석
- 결과를 경고하는 주석
- TODO 주석
- 중요성을 강조하는 주석
- 공개 API에서 Javadocs

나쁜 주석

Bad Comment

- 주절거리는 주석
- 같은 이야기를 중복하는 주석
- 오해할 여지가 있는 주석
- 의무적으로 다는 주석
- 이력을 기록하는 주석
- 있으나 마나 한 주석
- 무서운 잡음
- 함수나 변수로 표현할 수 있다면 주석을 달지 마라
- 위치를 표시하는 주석
- 닫는 괄호에 다는 주석
- 공로를 돌리거나 저자를 표시하는 주석
- 주석으로 처리한 코드
- HTML 주석
- 전역 정보
- 너무 많은 정보
- 모호한 관계
- 함수 헤더
- 비공개 코드에서 Javadocs

감사합니다!

참여해 주셔서
감사합니다.
좋은 하루 보내세요.