


Clean Code

-naming-

1. 의도를 분명히 밝혀라

의도를 분명히 밝혀라

- 이 변수/함수가 **존재**하는 **이유**가 무엇인가?
- 이 변수/함수가 **수행** 하는 **기능**이 무엇인가?
- 이 변수/함수를 **사용**하는 **방법**은 어떻게 되는가?
- 이 변수/함수에 달린 **주석의 의미**를 반영할 수 있는가?
- 이 클래스에서 제공하는 **메소드 목록**에는 무엇이 있는가?



```
const d; // day  
  
vs  
  
const daySinceCreation;
```

- ➔ 변수명에 따로 주석이 필요하다면 의도를 분명히 드러내지 못한 것이다.
- ➔ 한줄 짜리 주석이라면 되도록 변수명에 포함해서 그 개념을 명확히 드러내는 것이 좋다.

의도를 분명히 밝혀라

✴ Q. 이 메소드가 하는 일은?

```
//의도가 분명하지 않은 코드의 예
public List<int[]> getThem(){
    List<int[]> list1 = new ArrayList<int[]>();
    for(int[] x : theList)
        if(x[0] == 4)
            list.add(x);
    return list1;
}
```

- list에서 무엇이 들었는가?
- list의 0번 인덱스 값인지 왜 확인하는가?
(왜 중요한가?)
- 매직넘버(0, 4)는 무슨 의미인가?
- 추가한 리스트는 어떻게 사용되는가?

➡ 문제는 코드의 맥락이 아니라 함축성에 있다.

➡ 코드의 맥락이 코드 자체에 명시적으로 드러나지 않는다.

의도를 분명히 밝혀라

✴ A. 지뢰찾기 게임판을 만드는 코드였다.



```
const getFlaggedCells = gameBoard => {  
  gameBoard.map((currentValue, index) => {  
    index === STATUS_VALUE && currentValue === FLAGGED ?  
      gameBoad.add(currentValue) : console.log('');  
  });  
}
```

theList	gameBoard
게임판 각 칸	단순 배열
배열 인덱스 0	칸 상태
값 4	깃발이 꽂힌 상태

✴ A-2. 한 걸음 더 나아가



```
//개선된 코드  
public List<Cell> getFlaggedCells() {  
  List<Cell> flaggedCells = new  
  ArrayList<Cell>();  
  for(Cell cell : gameBoard)  
    if(cell.isFlagged())  
      flaggedCells.add(cell);  
  return flaggedCells;  
}
```

✓ 검색하기 쉬운 코드를 적용하기 위해서 의도가 있는 변수 이름으로 변수가 사용되는 범위 크기에 맞게 이름의 길이를 늘렸다.

✓ 의미있는 맥락 추가: 쉽고 명확한 이름을 부여하기 위해 필요하다면 클래스를 생성하여(타입을 지정하여) 명사나 명사구 형태로 주석없이 목적과 존재이유를 설명하도록 한다.

✓ (예시에서는 Cell 클래스를 만들어 isFlagged라는 명시적 함수를 사용하여 FLAGGED라는 상수를 감쌌음)

2. 그릇된 정보를 피하라

그릇된 정보를 피하라

코드에 의미를 흐리는 그릇된 단서를 남기지 말라.

1. **축약어**, 변수에 **두가지 이상의 뜻**이 있는 경우
hp: hypotenuse(직각삼각형의 빗변), unix변종
 2. 서로 **흡사한 이름**
서로 다른 모듈에서 사용되는 XYZControllerHandlingOfStrings, XYZControllerStorageOfStrings
 3. 유사한 개념은 **유사한(일관된) 표기법** 사용
 4. 코드를 읽는 사람이 변수를 다른 이름으로 변환해서 읽지 않도록 한다.
- ➡ 유사한 개념에서 일관된 표기법을 사용하는 것은 정보 제공을 위함이다. 코드는 추리 소설이 아니다. 명시적으로 정보를 드러내어 독자로 하여금 바로 인지할 수 있게 돕자.
 - ➡ 코드를 읽는 사람이 변수를 해석해야 하는 상황은 코드 작성자가 문제 영역이나 해법 영역에서 사용하지 않는 이름을 선택했기 때문이다.
 - ➡ 기술 개념에는 기술이름이 가장 적합한 선택이기 때문에 해법 영역(전산 용어, 알고리즘 이름, 패턴 이름, 수학 용어 등)에서 가져온 이름을 사용해야 한다.
 - ➡ 코드의 맥락이 코드 자체에 명시적으로 드러나지 않는다면 명료한 코드가 아니다.

3. 의미 있게 구분하라

의미 있게 구분하라

✳️ 단지 컴파일러를 통과하기 위한 네이밍을 붙이지 말라.

```
//example 1 : List of consecutive numbers
const copyChars = ((a1/*->source*/, a2/*->destination*/) => {
    source.map(currentValue => {
        ...
    })
})

//example 2 : meaningless noise word
let productInfo; //productInfoData and so on...
let getTheActiveAccountsInfo = () => {} //How different getActiveAccount() is?
```

✓ 의미 없는 noise word나 과도한 정보 제공은 주의를 흐뜨러트린다.

✓ Handler, Manager, Processor, Data, Info, a, an, the와 같은 의미없는 단어나 중복된 표현, 애매한 표현, 축약어, 동사는 피한다.
(모든 지역 변수는 a를 사용하고 함수는 the를 사용하는 룰이 있는 등 팀에서 합의한 규칙이 없는 경우)

✓ 변수 이름에 쓸데없는 접두어나 타입 인코딩은 뻔다.
ex) Java 멤버변수 앞 m - Circle mCircle;

의미 있게 구분하라

✳ 한개념에 한 단어만 사용하라.

지금 까지 구현한 add 함수는 모두
기존 값 두개를 더해서
새로운 값을 할당하는 함수인데
새로 작성하는 함수는 값하나를 추가하는 함수네...
이 함수의 접두어에
add를 붙여도 괜찮을까?



→ 기존 add 함수와 맥락이 다르므로(독자적) insert나 append가 적당

✓ 이름이 달라진다는 것은 의미(개념)도 달라진다는 뜻이다.

나쁜예) productInfo, productData - 개념을 구분하지 않은 채 이름을 다르게 쓰지 않도록 한다.

✓ 독자적이고 일관성있는 어휘를 사용해야 한다.

(똑같은 기능의 메서드가 컴포넌트마다 fetch, retrieve, get 혹은 controller, manager, driver란 변수명으로 섞여 있는 경우 혼란을 야기하게 된다.)

✓ 변수 이름에 쓸데없는 접두어나 타입 인코딩은 뺀다.

ex) Java 멤버변수앞 m - Circle mCircle;

4. 쉬운 이름을 사용하라

쉬운 이름을 사용하라

발음하기 쉬운 이름을 사용하라.

➡ 회의 혹은 팀간에 지적 대화가 가능한 이름으로 작성하라
genymdhms...? ➡ generationTimeStamp

검색하기 쉬운 이름을 사용하라.

➡ 매직넘버(상수사용)을 기피해야 하는 이유
둘중에 검색이 쉬운 이름은? 7 vs MAX_CLASSES_PRE_STUDENT

➡ 검색이 쉽고, 찾은 후에 의미를 분석하지 않아도 되는 이름으로 작성하라
이런 관점에서 자바 개발자들은 긴이름을 짧은 이름보다 선호하지만 의미가 분명하다면 일반적으로 짧은 이름이 더 좋다.

➡ 변수 이름의 길이는 변수의 사용 범위와 비례하게 한다.
(변수를 여러 곳에서 사용한다면 검색하기 쉬워야 하므로)