클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

들어가면서

"**소프트웨어에서 이름은 어디나 쓰인다**. 우리는 변수에도 이름을 붙이고, 함수에도 이름을 붙이고,... 파일에도 이름을 붙인다. 여기저기 도처에서 이름을 사용한다. 이렇듯 많이 사용하므로 **이름을 잘 지으면 여러모로 편하다**."

의도를 분명히 밝혀라

"여기서는 **의도가 분명한 이름이 정말로 중요**하다는 사실을 거듭 강조한다. 좋을 이름을 지으려면 시간이 걸리지만 **좋은 이름으로 절약하는 시간이 훨씬 더 많다**."

"변수나 함수 그리고 클래스 이름은 다음과 같은 굵직한 질문에 모두 답해야한다. **존재 이유는? 수행 기능은? 사용 방법은?** 따로 주석이 필요하다면 의도를 분명히 드러내지 못했다는 말이다."

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

```
public List<int[]> getFlaggedCelles(){
  List<int[]> flaggedCells = new ArrayList<int[]>();
  for(int[] cell: gameBoard)
    if(cell[STATUS_VALUE] == FLAGGED)
      flaggedCells.add(cell);
  return flaggedCells;
}
```

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

그릇된 정보를 피하라

“프로그래머는 코드에 그릇된 단서를 남겨서는 안 된다. 그릇된 단서는 코드 의미를 흐린다. 나름대로 널리 쓰이는 의미가 있는 단어를 다른 의미로 사용해도 안 된다.”

“서로 흡사한 이름을 사용하지 않도록 주의한다.”

“유사한 개념은 유사한 표기법을 사용한다. 이것도 정보다. 일관성이 떨어지는 표기법은 그릇된 정보다.”

의미 있게 구분하라

“컴파일러나 인터프리터만 통과하려는 생각으로 코드를 구현하는 프로그래머는 스스로 문제를 일으킨다.”

“컴파일러를 통과할지라도 **연속된 숫자를 덧붙이거나 불용어(noise word)를 추가하는 방식은 적절하지 못하다**. 이름이 달라야 한다면 의미도 달라져야 한다. 연속적인 숫자를 덧붙인 이름(a1, a2, …, aN)은 의도적인 이름과 정반대다. 이런 이름은 그릇된 정보를 제공하는 이름도 아니며, 아무런 정보를 제공하지 못하는 이름일 뿐이다. 저자 의도가 전혀 드러나지 않는다.”

```
getActiveAccount();
getActiveAccounts();
getActiveAccountInfo();
```

이 프로젝트에 참여한 프로그래머는
어느 함수를 호출할지 어떻게 알까..?

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

발음하기 쉬운 이름을 사용하라

"사람들은 단어에 능숙하다. 우리 두뇌에서 상당 부분은 단어라는 개념만 전적으로 처리한다. 그리고 정의상으로 단어는 발음이 가능하다. 말을 처리하려고 발달한 두뇌를 활용하지 않는다면 안타까운 손해다."

"발음하기 어려운 이름은 토론하기도 어렵다."

검색하기 쉬운 이름을 사용하라

"문자 하나를 사용하는 이름과 상수는 텍스트 코드에서 쉽게 눈에 띄지 않는다는 문제점이 있다."

"MAX_CLASSES_PER_STUDENT" 는 grep으로 찾기가 쉽지만, 숫자 7은 은근히 까다롭다. 7이 들어가는 파일 이름이나 수식이 모두 검색되기 때문이다."

```
int s = 0;
for(int j=0; j<34; j++){
  s += (t[j]*4)/5;
}
```

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for(int j=0; j<NUMBER_OF_TASKS; j++){
  int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
  int realTaskWeeks = (realTaskDays / WORK_DAYS_PER_WEEK);
  sum += realTasksWeeks;
}
```

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

인코딩을 피하라

헝가리식 표기법
"...IDE는 코드를 컴파일하지 않고도 타입 오류를 감지할 정도로 발전했다. 따라서 이제는 헝가리식 표기법이나 기타 인코딩 방식이 오히려 방해가 될 뿐이다. 변수, 함수, 클래스 이름이나 타입을 바꾸기가 어려워지며, 읽기도 어려워진다. 독자를 오도할 가능성도 커진다."

멤버 변수 접두어
"...코드를 읽을수록 접두어는 관심 밖으로 밀려난다. 결국은 접두어는 옛날에 작성한 구닥다리 코드라는 징표가 되버린다."

인터페이스 클래스와 구현 클래스
"때로는 인코딩이 필요한 경우도 있다."

Shape Factory 를 구현한다고 가정하고, 인터페이스와 구현 클래스의 이름을 정하는 예시라면
인터페이스의 이름이 ShapeFactory 가 되고, 구현 클래스의 이름이 ShapeFactoryImp 가 되는게 좋다.
JAVA 의 SOLID 원칙중 D(DIP) 가 있는데, '의존은 항상 구체적인 것이 추상화된 것에 의존해야 한다.' 는 원칙.
즉, ShapeFactory 라는 추상화된 개념은 항상 고정되어 있고 구현 클래스는 언제든지 TriangleFactory, CircleFactory 등 바뀔 가능성이 있기 때문이다.
ex> List<String> names = new ArrayList<String>();
List<String> names = new LinkedList<String>();

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

자신의 기억력을 자랑하지 마라

“독자가 코드를 읽으면서 변수 이름을 자신이 아는 이름으로 변환해야 한다면 그 변수 이름은 바람직하지 못하다. 이는 일반적으로 **문제 영역이나 해법 영역에서 사용하지 않는 이름을 선택했기 때문에 생기는 문제**다.”

“r 이라는 변수가 호스트와 프로토콜을 제외한 소문자 URL 이라는 사실을 언제나 기억한다면 확실히 똑똑한 사람이다. 똑똑한 프로그래머와 전문가 프로그래머 사이에서 나타나는 차이점 하나만 들자면, 전문가 프로그래머는 **명료함이 최고**라는 사실을 이해한다. 전문가 프로그래머는 자신의 능력을 좋은 방향으로 사용해 **남들이 이해하는 코드를 내놓는다**.”

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

클래스 이름

“클래스 이름과 객체 이름은 **명사나 명사구**가 적합하다.”
ex> Customer, WikiPage, Account, AddressParser

메서드 이름

“클래스 이름과 객체 이름은 **동사나 동사구**가 적합하다.”
ex> postPayment, deletePage, save

“**생성자를 중복정의** 할 때는 **정적 팩토리 메서드를 사용**한다.”

Complex fulcrumPoint = new Complex(23.0);  Complex fulcrumPoint = Complex.FormRealNumber(23.0);

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

 # 기발한 이름은 피하라

"재미난 이름보다 **명료한 이름을 선택**하라."

 # 한 개념에 한 단어를 사용하라

"**추상적인 개념 하나에 단어 하나를 선택해 이를 고수**한다. 예를 들어, 똑같은 메서드를 클래스마다 fetch, retrieve, get 으로 제각각 부르면 혼란스럽다."

"**일관성 있는 어휘**는 코드를 사용할 프로그래머가 반갑게 여길 선물이다."

 # 말장난을 하지 마라

"**한 단어를 두 가지 목적으로 사용하지 마라**. 다른 개념에 같은 단어를 사용한다면 그것은 말장난에 불과하다."

"**프로그래머는 코드를 최대한 이해하기 쉽게** 짜야한다. 집중적인 탐구가 필요한 코드가 아니라 **대충 훑어봐도 이해할 코드 작성이 목표**다. 의미를 해독할 책임이 독자에게 있는 논문 모델이 아니라 **의도를 밝힐 책임이 저자에게 있는** 잡지 모델이 바람직하다."

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

해법 영역에서 가져온 이름을 사용하라

“**코드를 읽을 사람도 프로그래머**라는 사실을 명심한다.”

“모든 이름을 문제 영역에서 가져오는 정책은 현명하지 못하다. 같은 개념을 다른 이름으로 이해하던 동료들이 매번 고객에게 의미를 물어야하기 때문이다.”

문제 영역에서 가져온 이름을 사용하라

“적절한 ‘프로그래머 용어’가 없다면 문제 영역에서 이름을 가져온다. 그러면 코드를 보수하는 프로그래머가 분야 전문가에게 의미를 물어 파악할 수 있다.”

“우수한 프로그래머와 설계자라면 **해법 영역과 문제 영역을 구분할 줄 알아야**한다. 문제 영역 개념과 깊은 코드라면 문제 영역에서 이름을 가져와야 한다.”

클린코드 스터디 1주차. 2장 〈의미 있는 이름〉

의미 있는 맥락을 추가하라

“스스로 의미가 분명한 이름이 없지 않다. 하지만 대다수 이름은 그렇지 못하다. 그래서 클래스, 함수, 이름 공간에 넣어 맥락을 부여한다. 모든 방법이 실패하면 마지막 수단으로 접두어를 붙인다.”

객체지향언어에서는 클래스를 잘 활용하여 묶어내고 의미를 부여하는 방법이 제일 좋다.

불필요한 맥락을 없애라

“ ‘고급 휘발유 충선소(Gas Station Deluxe) 라는 애플리케이션을 짠다고 가정하자. 모든 클래스 이름을 GSD로 시작하겠다는 생각은 전혀 바람직하지 못하다.”

“일반적으로는 짧은 이름이 긴 이름보다 좋다. 단, 의미가 분명한 경우에 한해서다. 이름에 불필요한 맥락을 추가하지 않도록 주의한다.”
