

# [2기][item51] 메서드 시그니처를 신중히 설계해라


1. 메서드 이름을 신중히 짓자
2. 편의 메서드를 너무 많이 만들지 말자
3. 매개변수 목록은 짧게 유지하자
4. 매개변수의 타입으로는 클래스보다는 인터페이스가 더 낫다.
5. boolean vs 원소 두개 짜리 열거 타입

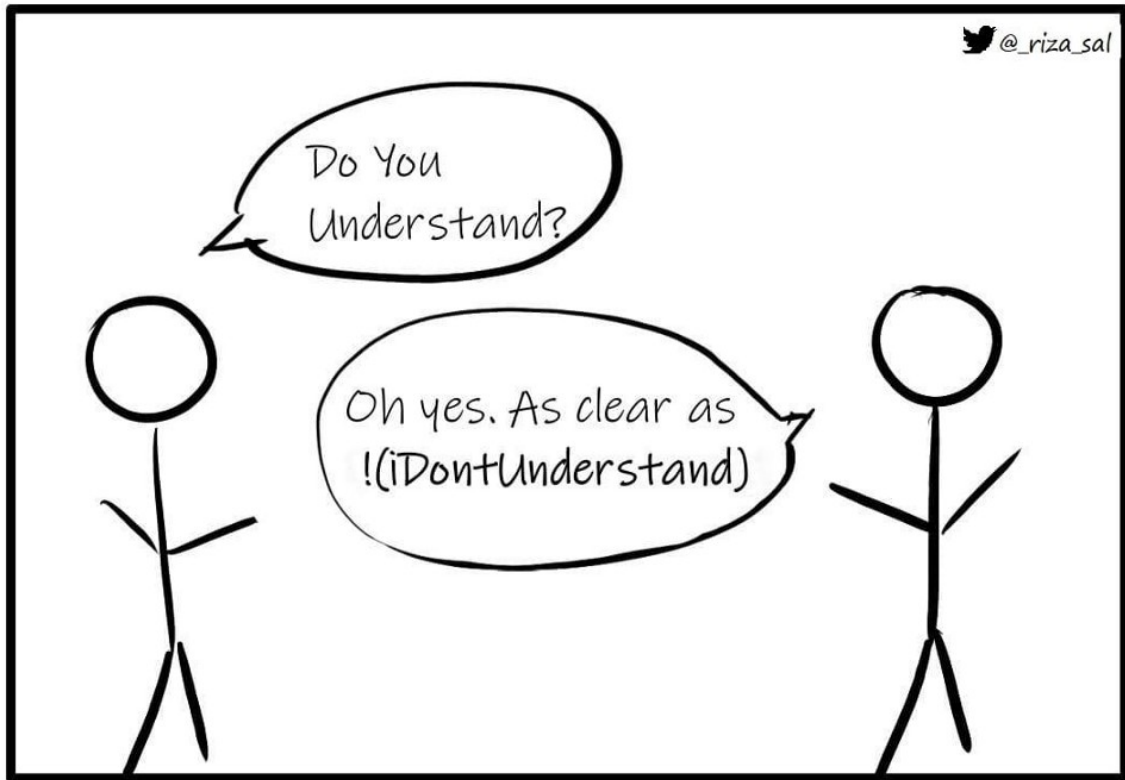
## 1. 메서드 이름을 신중히 짓자

- 표준 명명 규칙을 따라야 한다.

### 9. Naming Conventions

Your search did not match any results. We suggest you try the following to help find what you're looking for: Check the spelling of your keyword search. Use synonyms for the keyword you typed, for example, try "application" instead of "software." Try one of the popular searches shown below.

 <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>



<https://levelup.gitconnected.com/knot-of-nots-avoiding-negative-names-for-boolean-methods-641896a94a42>

- 이해할 수 있어야 한다.
- 같은 패키지에 속한 다른 이름들과 일관성이 있어야 한다.
- 개발자 커뮤니티에서 널리 받아들여지는 이름이어야 한다.
- 긴 이름은 피한다.
- 애매하면 자바 라이브러리 API 가이드를 참고하라.

<https://docs.oracle.com/javase/8/docs/api/>

## 2. 편의 메서드를 너무 많이 만들지 말자

- 모든 메서드는 각각 자신(만)의 소임을 다해야 한다(다하기만 하면 된다).

- 메서드가 너무 많으면 익히고, 사용하고, 문서화하고, 테스트하고, 유지보수하기 어렵다.
- 아주 자주 쓰일 경우에만 별도의 약칭 메서드를 두고 애매하다면 만들지 말라

### 3. 매개변수 목록은 짧게 유지하자

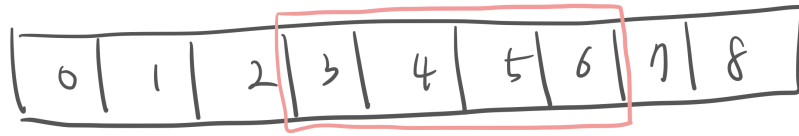
- 4개 이하여야 좋다
- 같은 타입의 매개변수가 여러 개 연달아 나오는 경우 특히 해롭다.
  - 사용자가 매개변수 순서를 기억하기 어렵다
  - 실수로 순서를 바꿔 입력해도 에러 없이 그대로 컴파일 된다.

```
public Person(int 학년, int 반) {
    this.학년 = 학년;
    this.반 = 반;
}
```

### ⚡ 과하게 긴 매개변수 목록을 짧게 줄여주는 기술

1) 여러 메서드로 쪼갬다.

- 자칫 잘못하면 메서드가 너무 많아질 수 있지만 직교성을 높여 오히려 메서드 수를 줄여 주는 효과가 있다.
- 그렇다고 무한정 작게 나누는 게 능사는 아니고, API 사용자의 눈높이에 맞게 API가 다루는 개념의 추상화 수준에 맞게 조절해야 한다.
- 예를 들어 어떤 리스트의 서브 리스트에서 특정 원소의 인덱스를 찾는 메서드를 지원해주지는 않는다.



4는 몇번째 인덱스인가?

⇒ 1

예를 들어 `indexOfInSublist(int fromIndex, int toIndex, Object o);` 와 같은 메서드는 매개변수를 3개 받아야 한다.

대신 부분리스트를 반환하는 `List<E> sublist(int fromIndex, int toIndex);`

특정 원소의 인덱스를 반환하는 `int lastIndexOf(Object o);`

두 메서드를 조합해서 사용할 수 있으며 각 메서드에서 사용되는 매개변수 개수는 `indexOfInSublist(...)` 보다 적다. 이처럼 원자적으로 쪼개진 메서드를 사용하면 매개변수를 줄일 수 있다.



### 직교성이 높다?

직교성이 높다 = 공통점이 없는 기능들이 잘 분리되어 있다, 기능을 원자적으로 쪼개 제공한다

예를 들어 MSA 아키텍처는 직교성이 높고, 모놀리틱 아키텍처는 직교성이 낮다고 할 수 있다.



### 직교성을 높여 오히려 메서드 수를 줄여주는 효과가 있다?

기본 기능이 잘 갖춰진 메서드가 있다면 아무리 복잡한 기능도 조합해서 만들 수 있다.

기능을 원자적으로 쪼개다 보면 자연스럽게 중복이 줄고 결합성이 낮아져 코드를 수정하기 수월해진다.

2) 매개변수 여러개를 묶어주는 도우미 클래스를 만든다.

- 잇따른 매개변수 몇 개를 독립된 하나의 개념으로 묶을 수 있을 때 추천하는 기법
- 예를 들어 카드게임을 구현하는 경우 카드의 숫자와 무늬를 뜻하는 매개변수를 하나의 카드라는 클래스가 멤버 변수로 숫자와 무늬를 가지고 있도록 할 수 있다.
- API는 물론 클래스 내부 구현도 깔끔해질 것이다.

### 3) 객체 생성에 빌더 패턴을 메서드 호출에 응용하자

- 매개변수가 많지만 그 중 일부는 생략해도 괜찮을 때 도움이 된다.

```
fun method(val1, val2, val3, val4, val5, ..., valN)
```

```
class Values {
    val val1
    val val2
    val val3
    ...
    val valN
}

fun method(Value v)

method (Values.builder()
    .setVal1(...)
    .setVal2(...)
    ...
    .build()
)
```

## 4. 매개변수의 타입으로는 클래스보다는 인터페이스가 더 낫다.

- 특정 구현체만 사용하도록 제한하지 않는게 좋다.
- 매개변수를 HashMap 타입으로 받는 것보다 Map 인터페이스로 받도록 정의해두면, HashMap 뿐 아니라 TreeMap, ConcurrentHashMap, TreeMap의 부분 맵 등 어떠한 Map 이더라도 인수로 건넬 수 있다.
- 혹 입력 데이터가 다른 형태로 존재한다면 명시한 특정 구현체의 객체로 옮겨 담느라 비싼 복사 비용을 치러야 한다. (List 를 Set 으로 변경한다던지 ...)

## 5. boolean vs 원소 두개 짜리 열거 타입

- 메서드 이름 상 boolean 을 받아야 의미가 더 명확할 때는 예외다.
- 열거 타입이 읽고 쓰기 더 쉽고, 나중에 선택지를 추가할 수도 있다.
- 예를 들어 온도계 클래스의 생성자에 매개변수로 Fahrenheit 인지 아닌지를 넣는 것 보다 TemperatureScale 이라는 열거 타입을 만들어 넘기는게 더 편하다. 후자의 경우 또 다른 scale 이 생긴다면 추가하기도 쉽다.

```
Thermometer.newInstance(true)
Thermometer.newInstance(false)
```

```
Thermometer.newInstance(TemperatureScale.FAHRENHEIT)
Thermometer.newInstance(TemperatureScale.CELSIUS)
// Thermometer.newInstance(TemperatureScale.KELVIN)
```