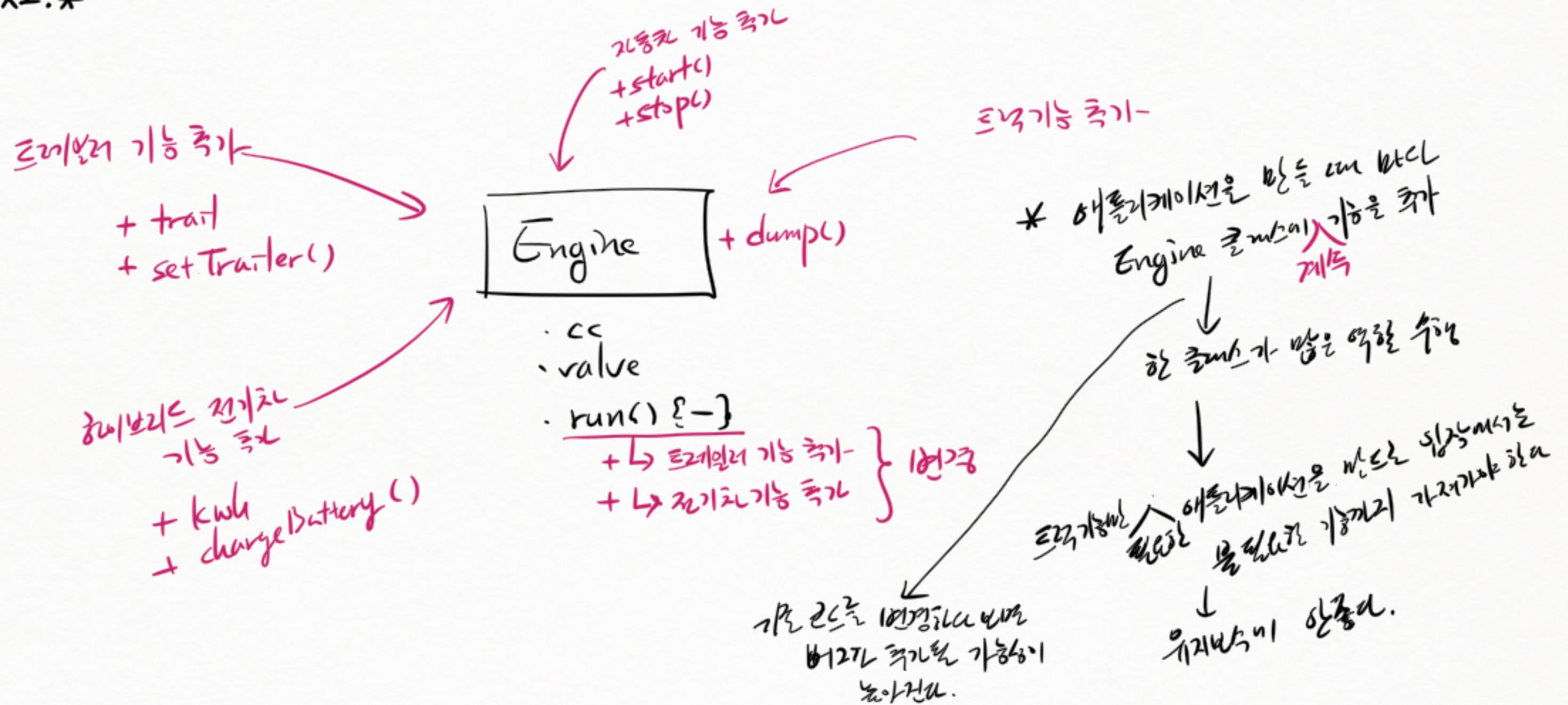


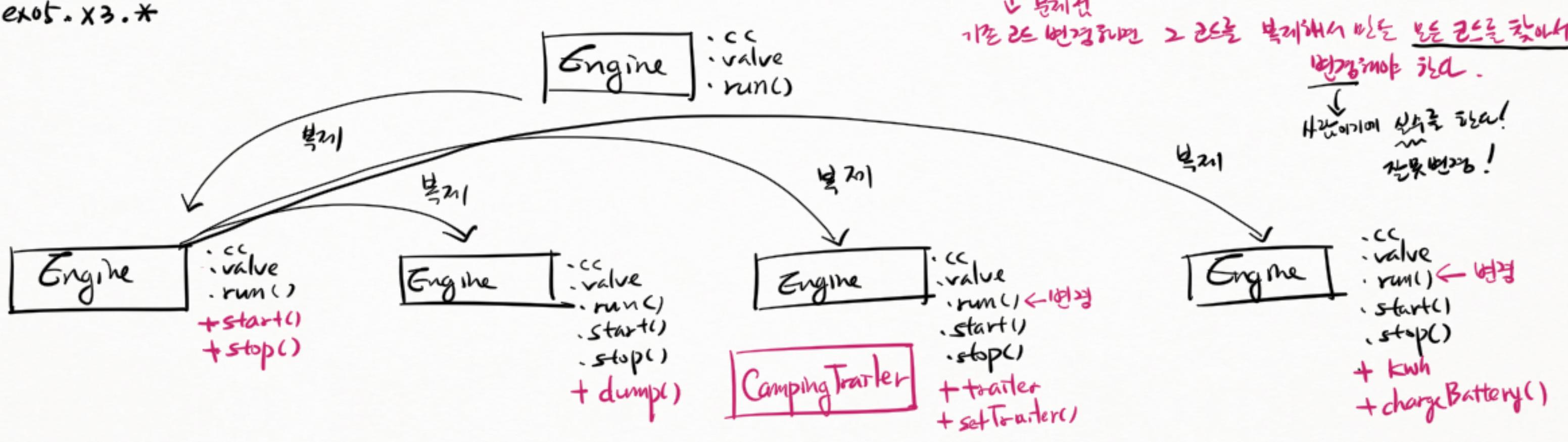
\* 가능학습 방법 | - 가능학습 방법 | 가능학습 방법 |

00P.0X05.X2.\*



## \* 기능 확장 방법 2 - 복제한 코드에 새 기능 추가

oop.ex05.\*



① 자동차 만들기  
app1

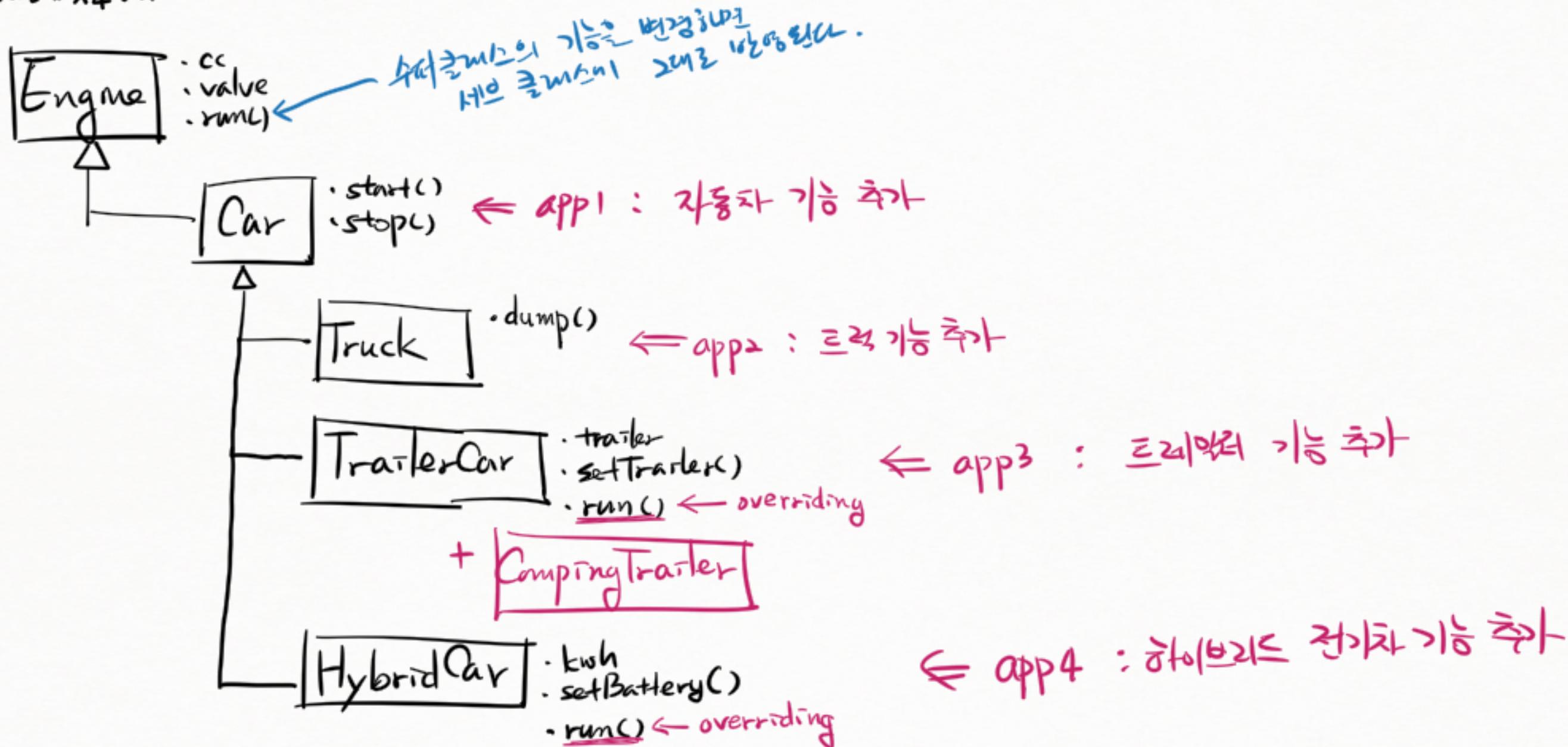
② 트레일러 만들기  
app2

③ 캠핑카 만들기  
app3

④ 헤드라이트 전기차 만들기  
app4

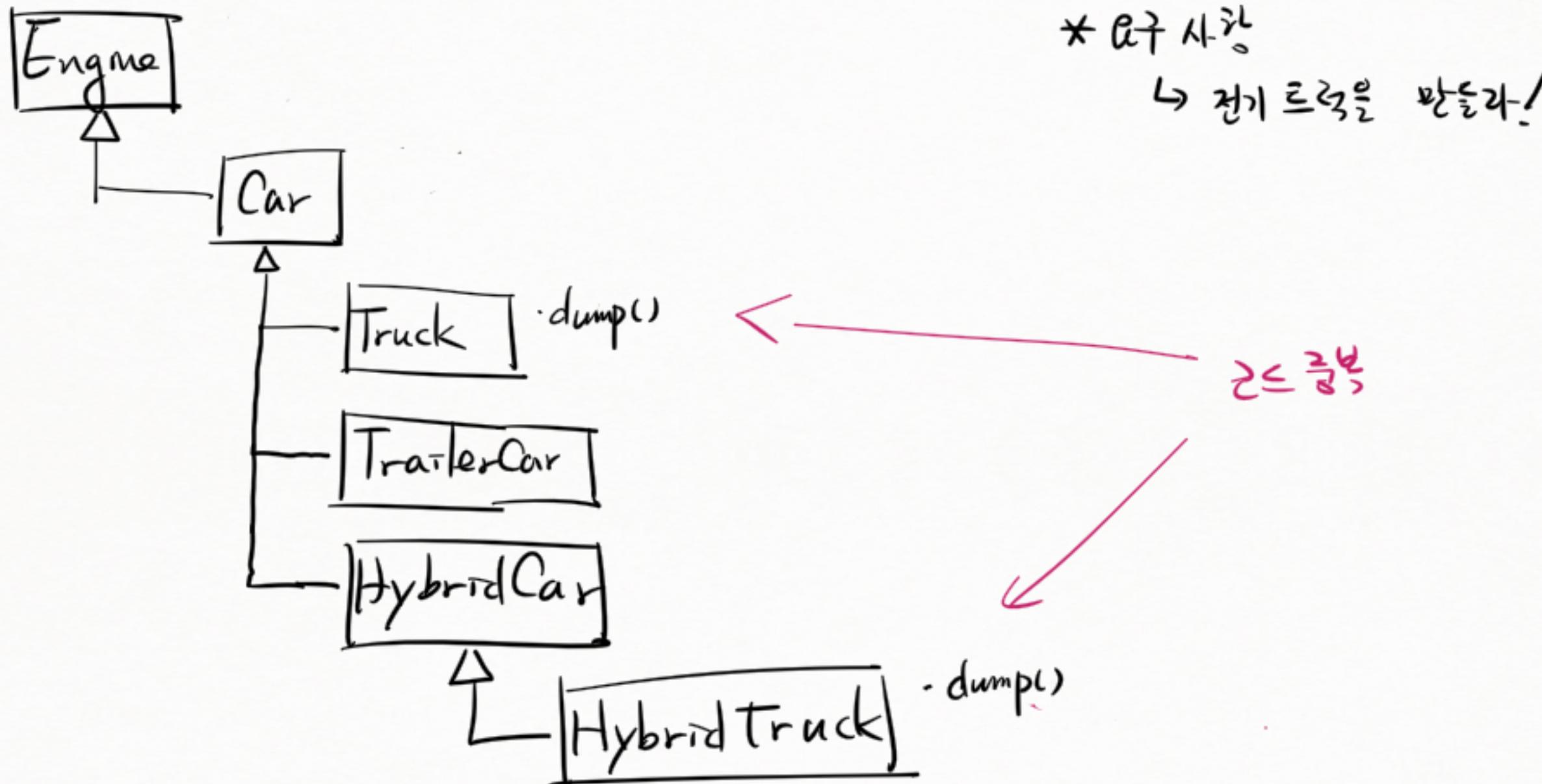
\* 기능 확장 방법 3 - 상속을 이용한 확장.

oop.ex05.x4.\*



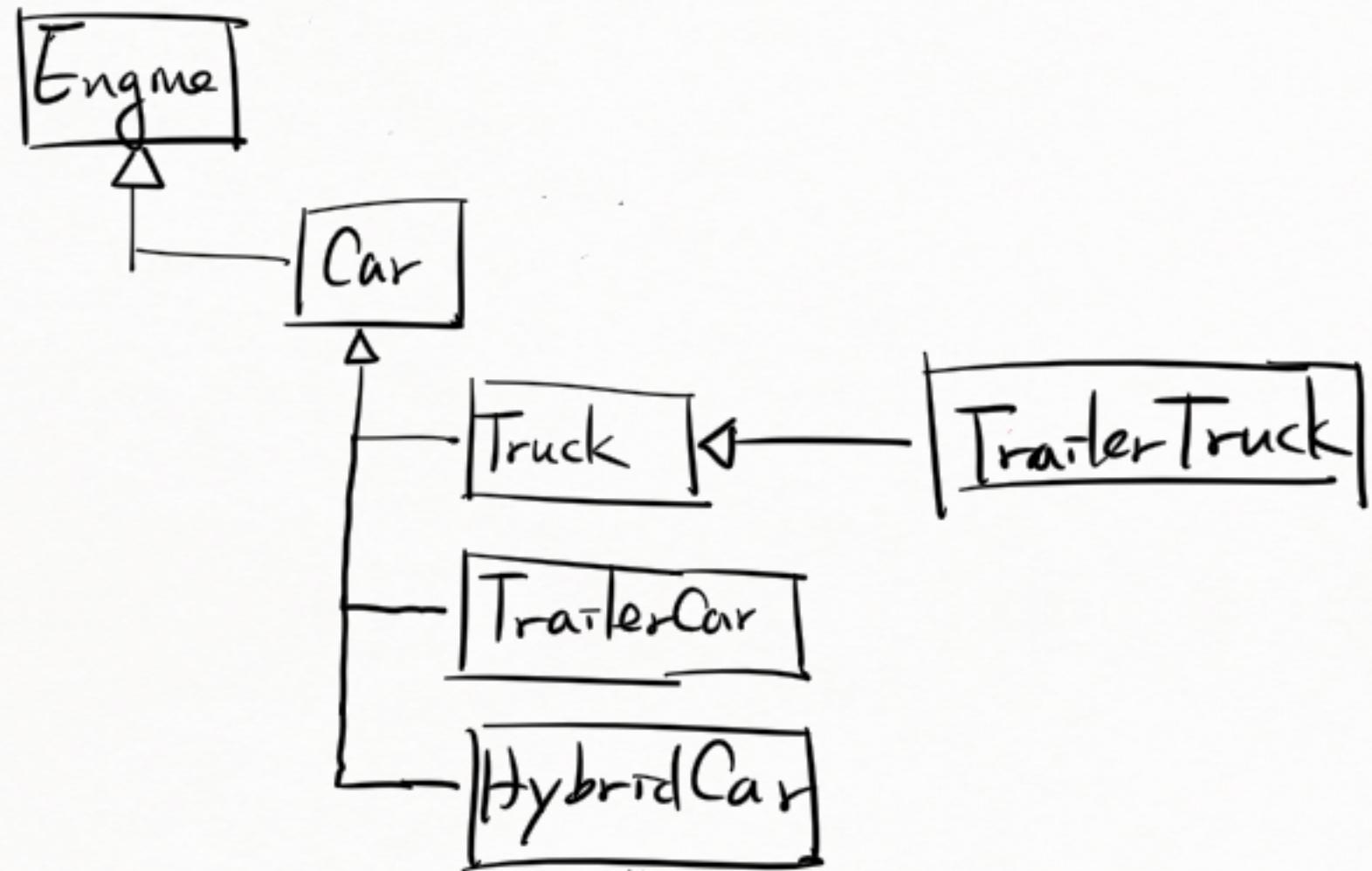
\* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한계

oop.ex05.x4.\*



\* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한가지

oop. ex05. x4.\*



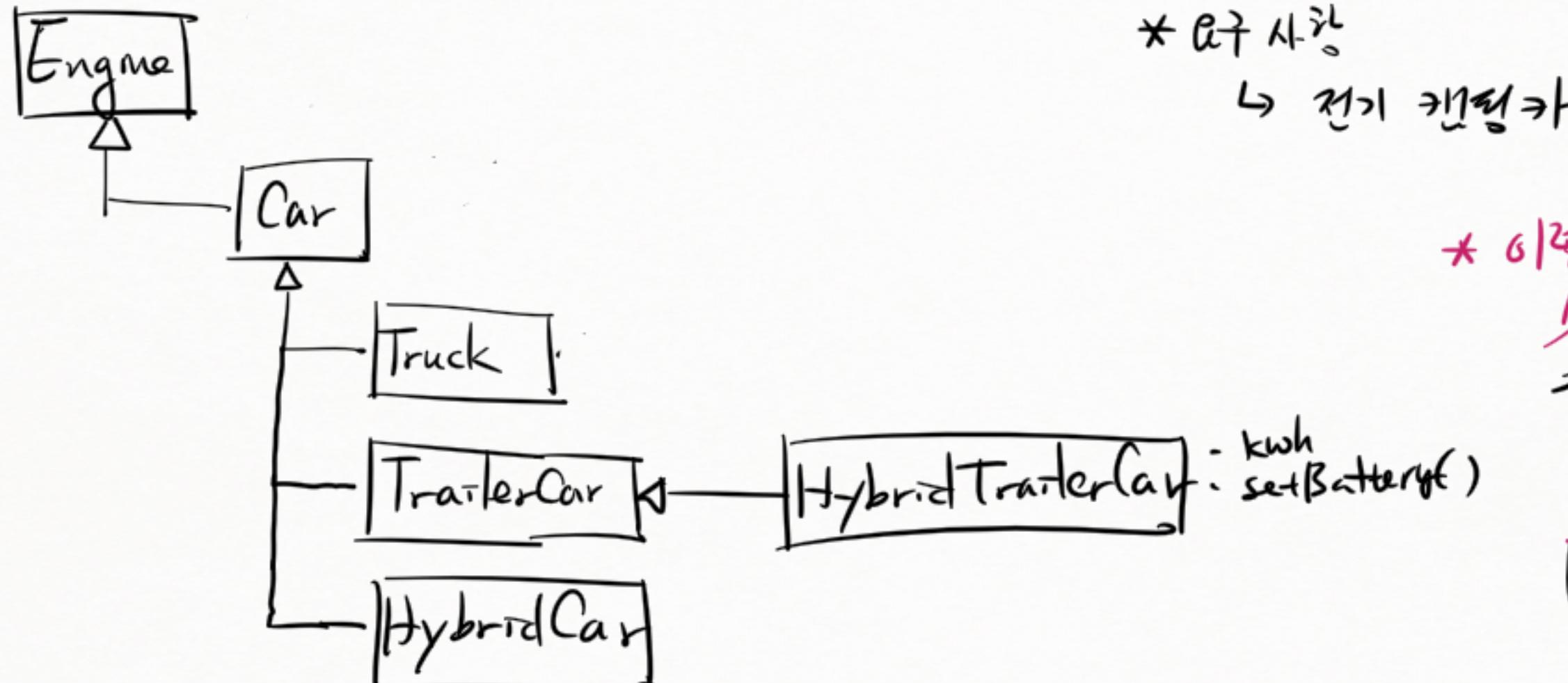
\* 추상화

↳ 트레일러를 뒤에 두 수 있는 트럭

· trailer  
· setTrailer()

\* 기능 학습 18주 3 - 상속을 통한 기능 학습의 한계

oop. ex05. x4.\*



\* 예시 사용

↳ 전기 캐리어카

\* 이렇게 기능 조합을 하려면  
수행은 네트워크로 구성된다

(상속으로 대상간 기능이 조합된  
개체를 만들기 어렵다.)

↳ 유지보수는 어렵지만...

\* 기능학적 특성 4 : 디자인에서 기초 기능

기능적

Sedan

Truck

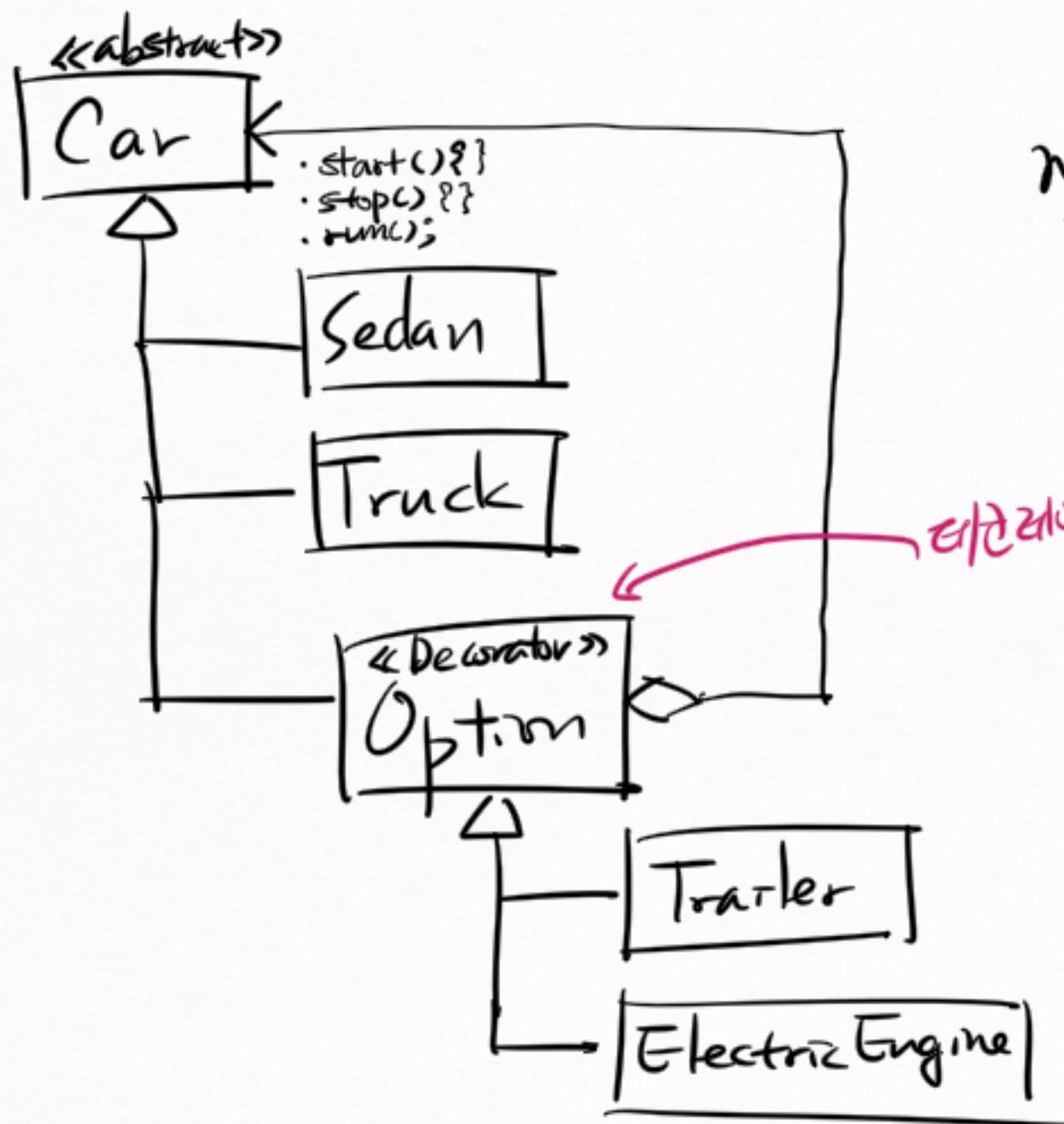
선택적

Gas Engine

Electric Engine

Trailer

## \* 테러리아 키미 기획



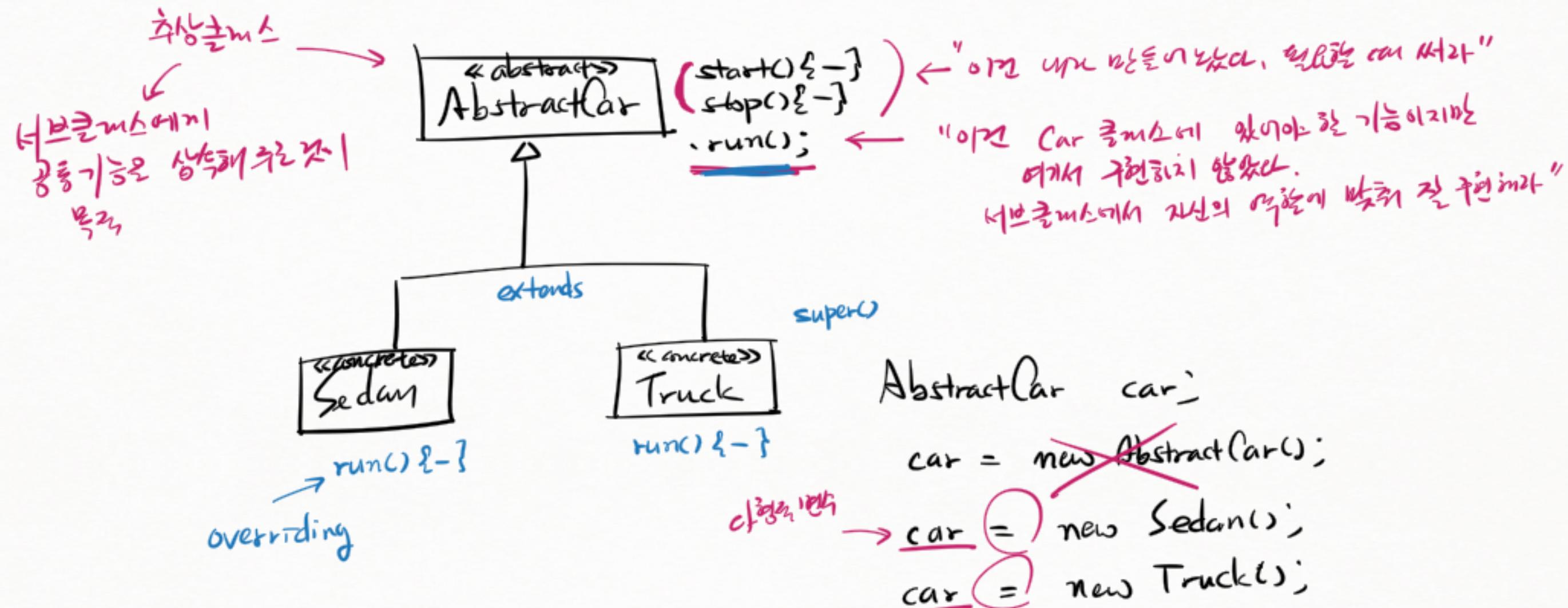
new 브라우저(new 브라우저(new 브라우저(주)))

단점 : 주제에 맞지 않아 선택 가능

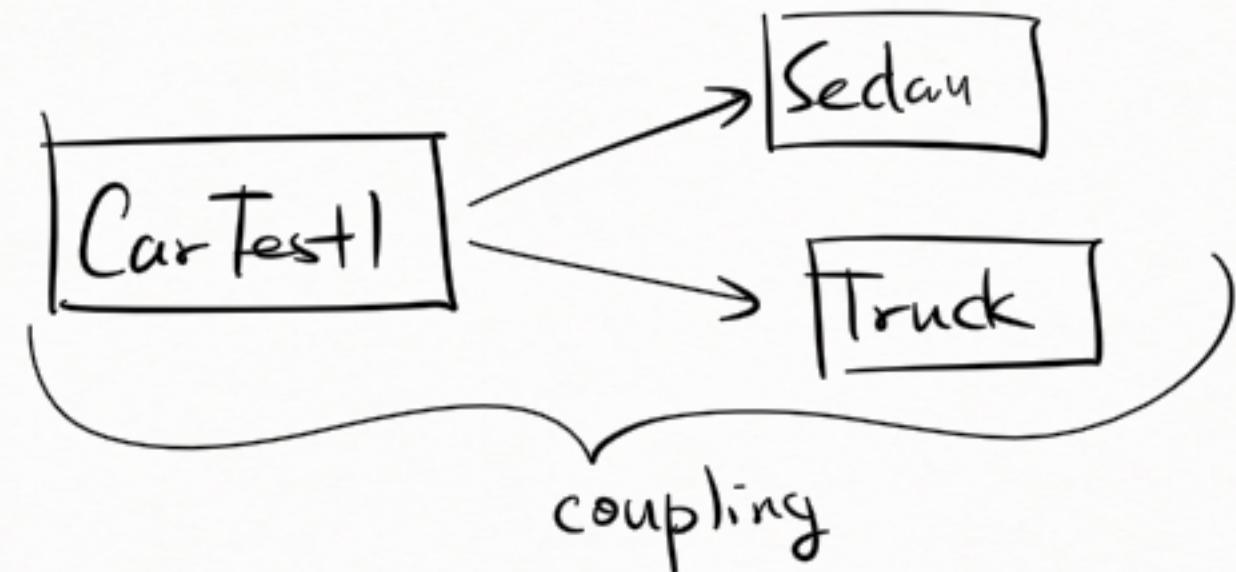
Ans 8m<sup>3</sup> (2km)

기 32  
Korean 02  
한국 07  
04d.

## \* 추상클래스와 구현



\* 실전 프로그래밍 1 단계



coupling

- openedSunroof
- auto
- **cc value**
- start() { - }
- stop() { - }
- ✓ · run() { - }
- openSunroof() { - }
- closeSunroof() { - }

**Sedan**

**Truck**

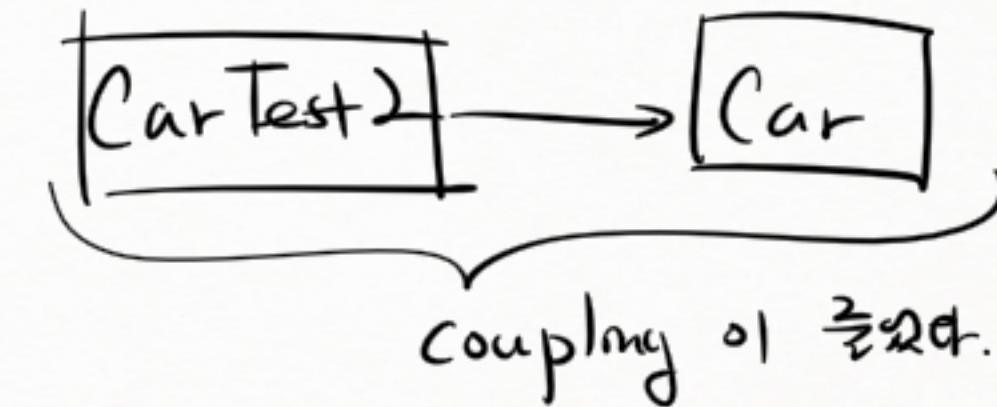
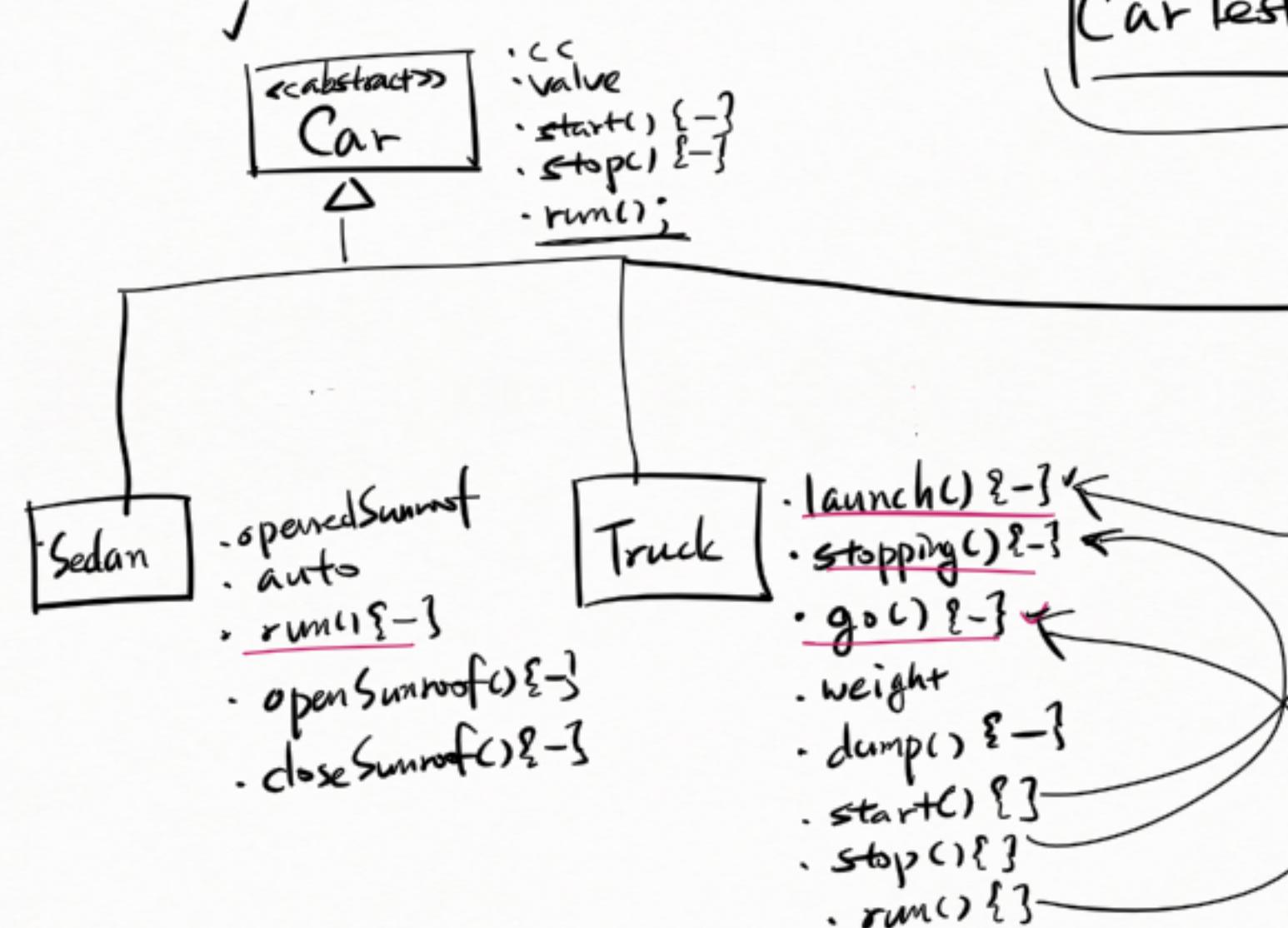
**Object**

- **cc value**
- launch() { - } ✓
- stopping() { - } -
- go() { - } ✓
- weight
- dump() { - }

제작  
이동  
제작  
제작  
제작  
제작

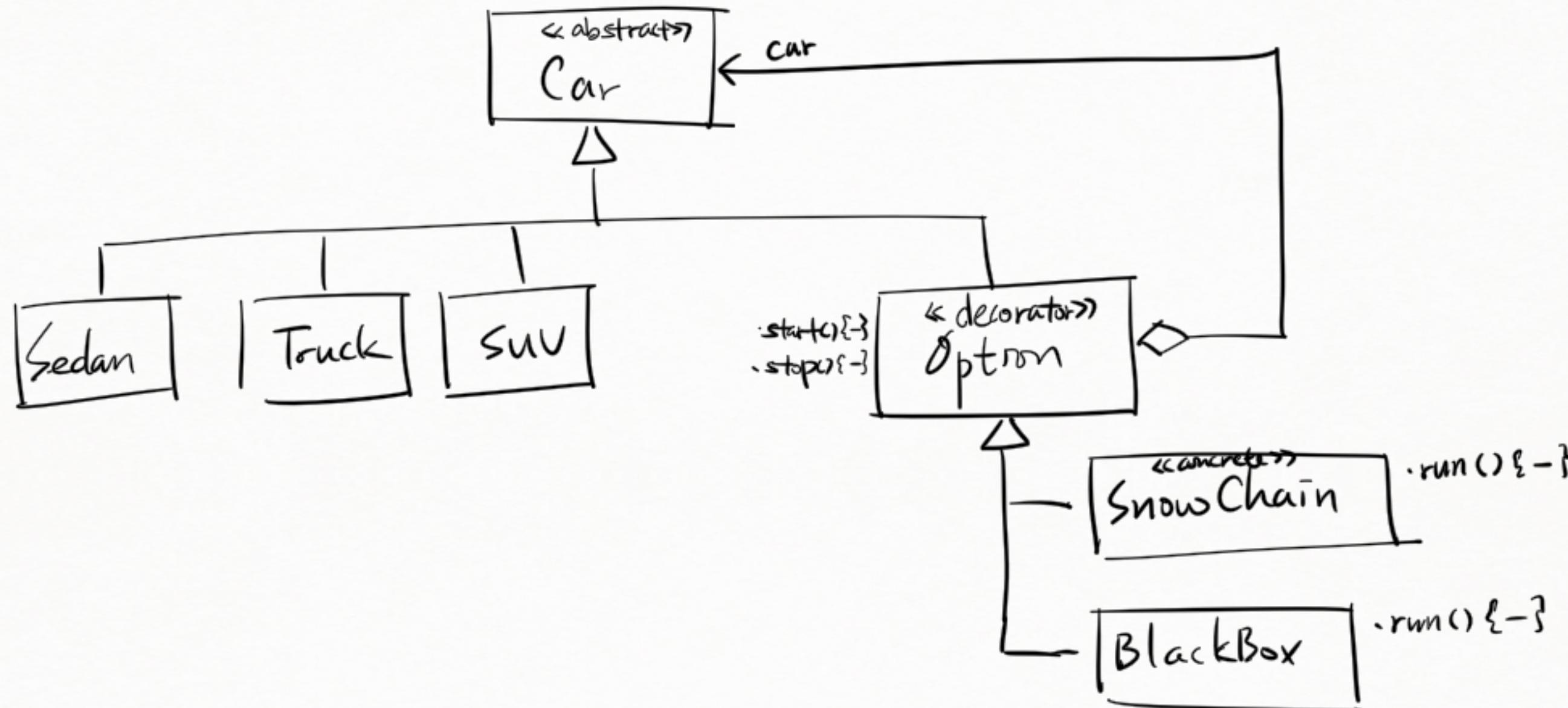
\* 상 하 한 한 한 한 한 - 품종, 품질, 성능

$\hookrightarrow$ : generalization

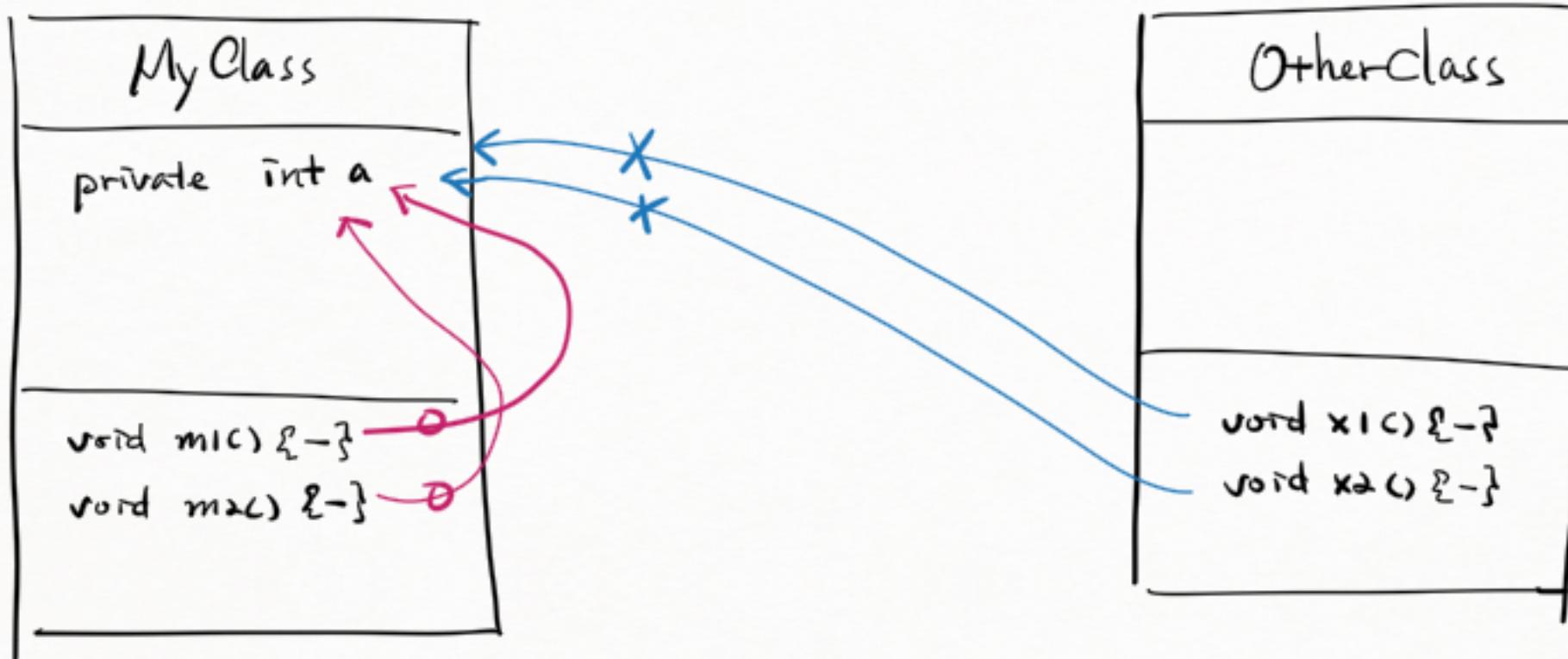


- enableLwd: boolean  
- activateLwd(boolean)  
- run() {-}

\* 차선을 3개로 나누면 차량이 1개로 통합

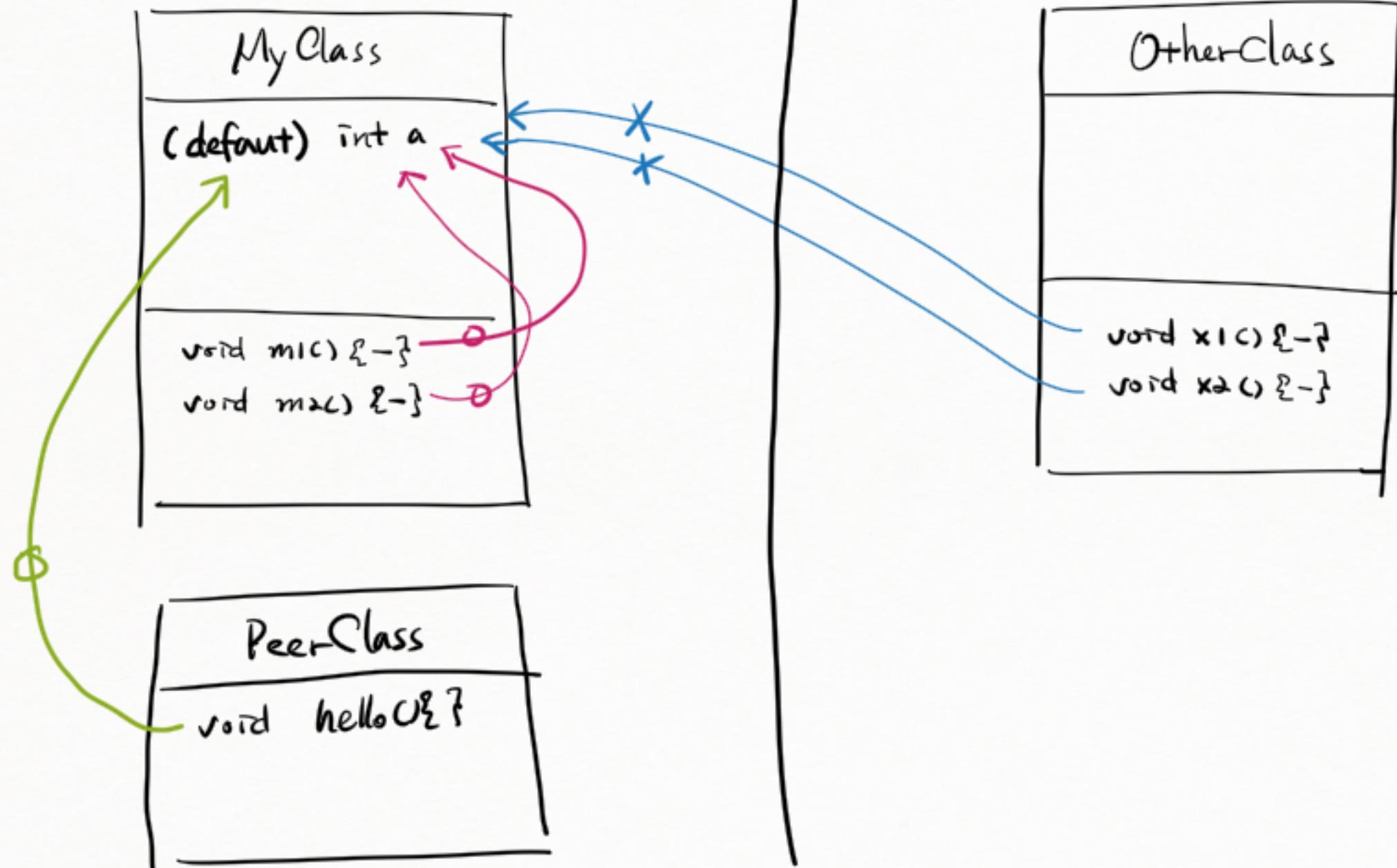


\* private - 같은 클래스의 멤버만 접근 가능

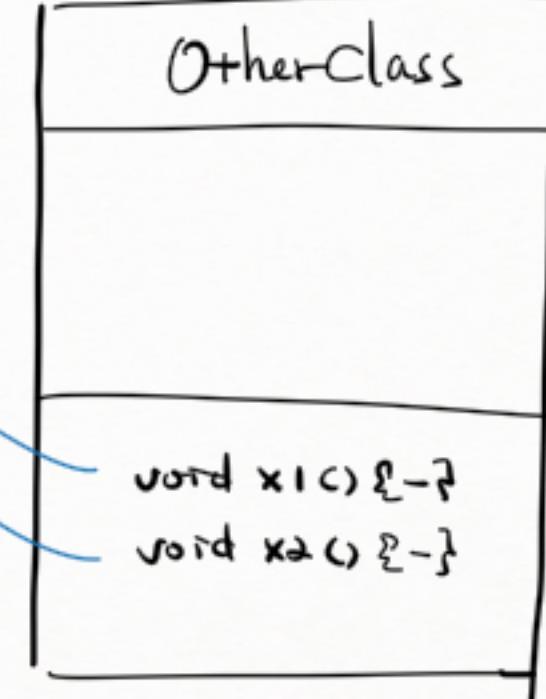


\* (default) - 같은 클래스의 멤버 접근 가능  
+ 같은 패키지 속 클래스의 멤버 접근 가능

com.eomcs.test1



com.eomcs.test2



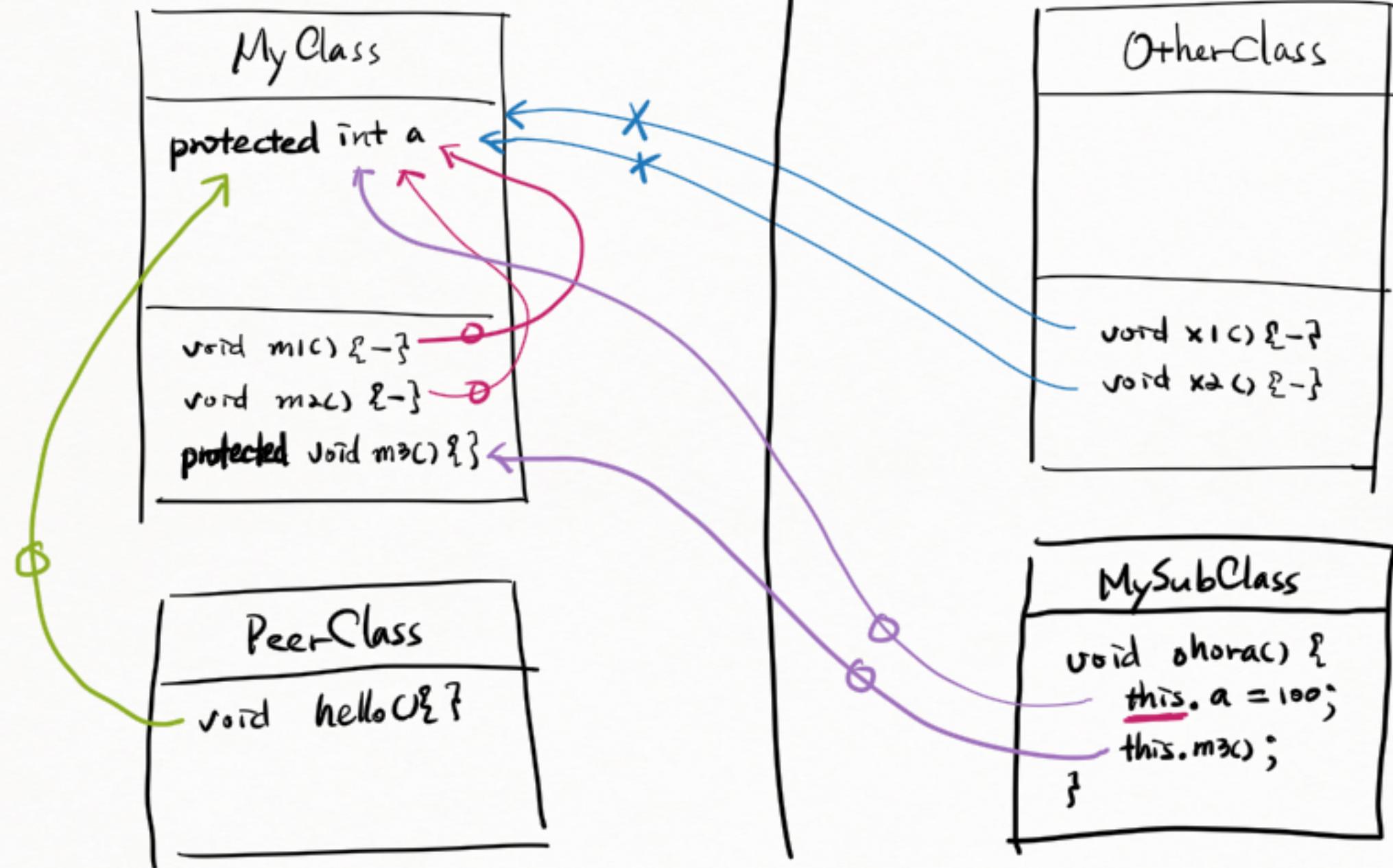
\* protected - 같은 패키지 내의 멤버 접근 가능

+ 같은 패키지 속 다른 클래스의 멤버 접근 가능 + 다른 클래스 접근 가능

com.eomcs.test1

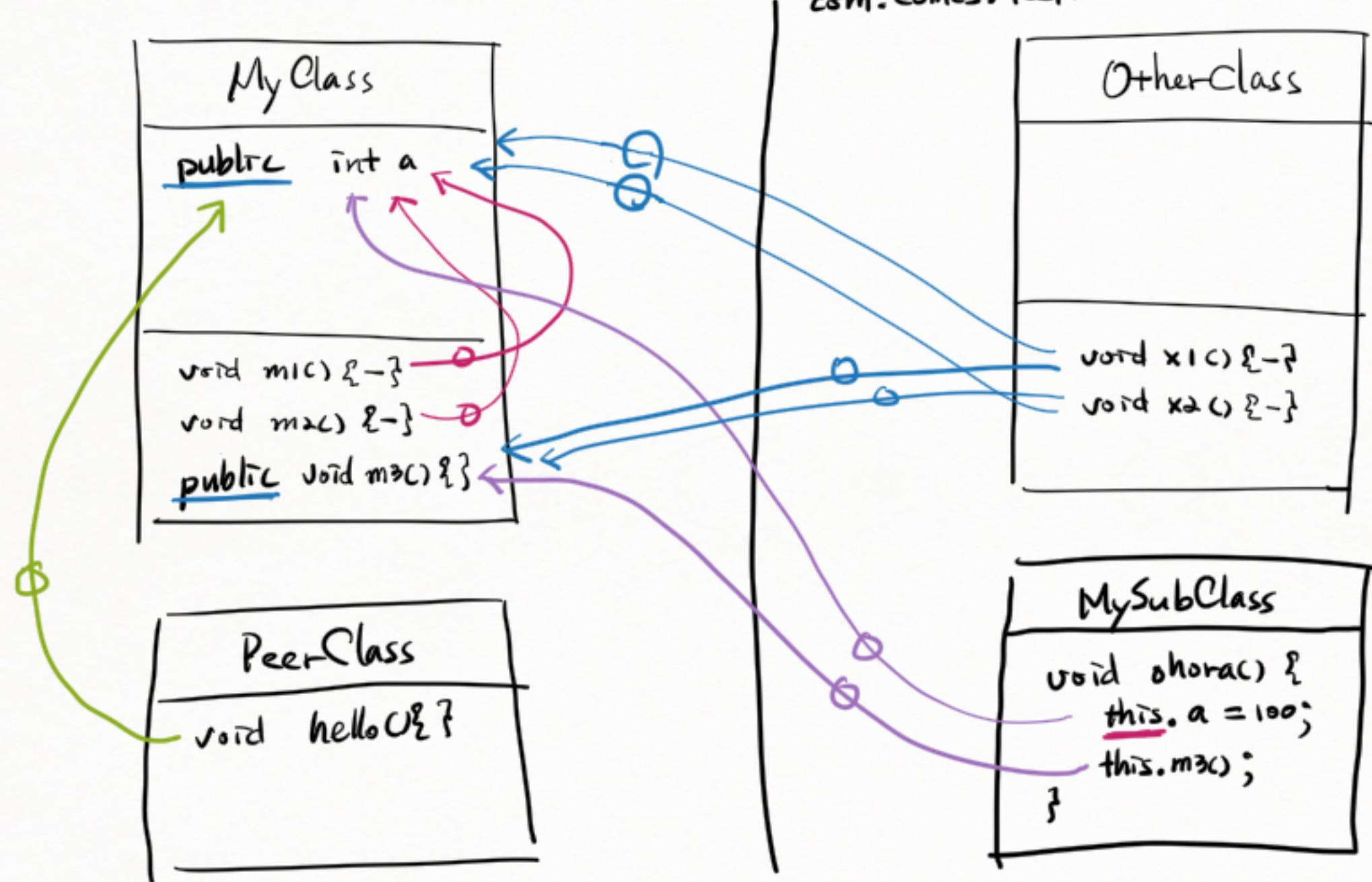
com.eomcs.test2

!!  
자신이 상속 받은 멤버와 더불어  
자신이 상속 받은 멤버와 더불어

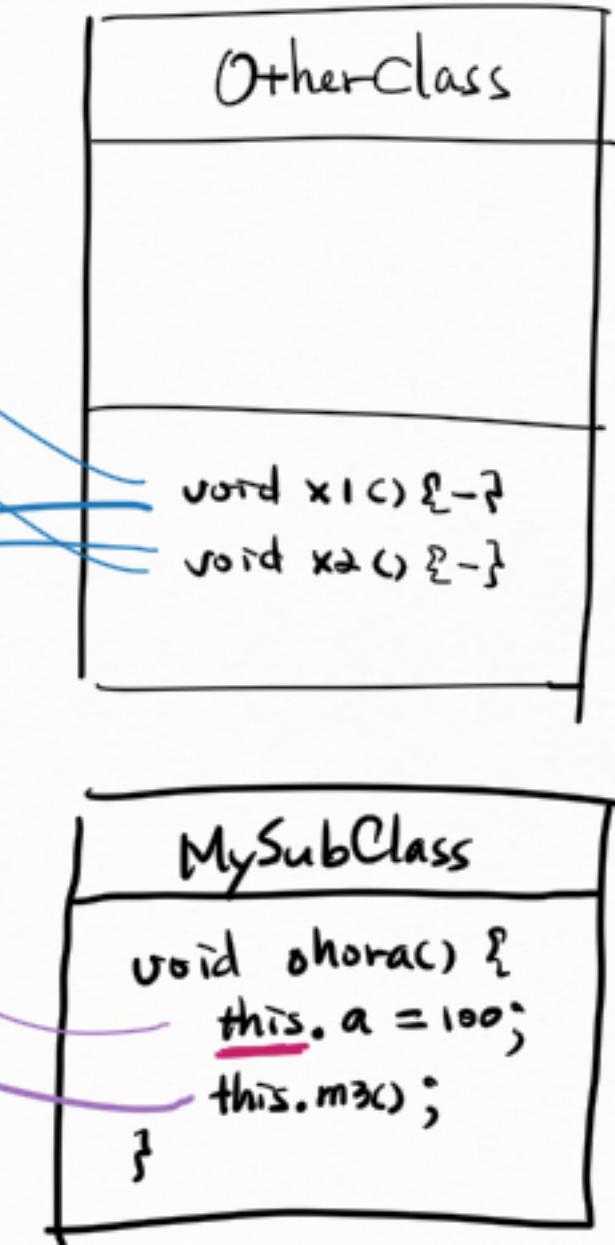


\* public - 모든 멤버 접근 가능

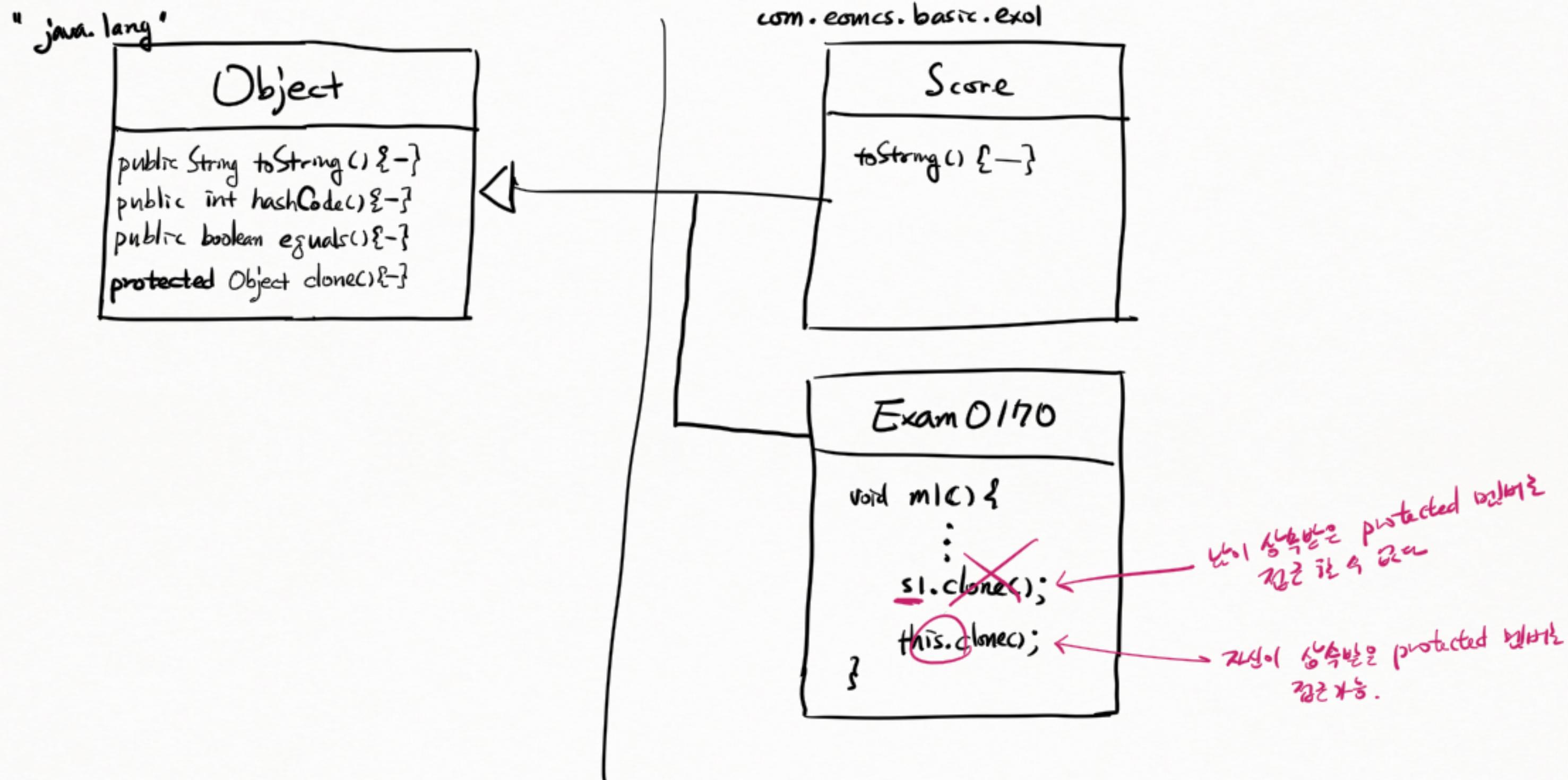
com.eomcs.test1



com.eomcs.test2

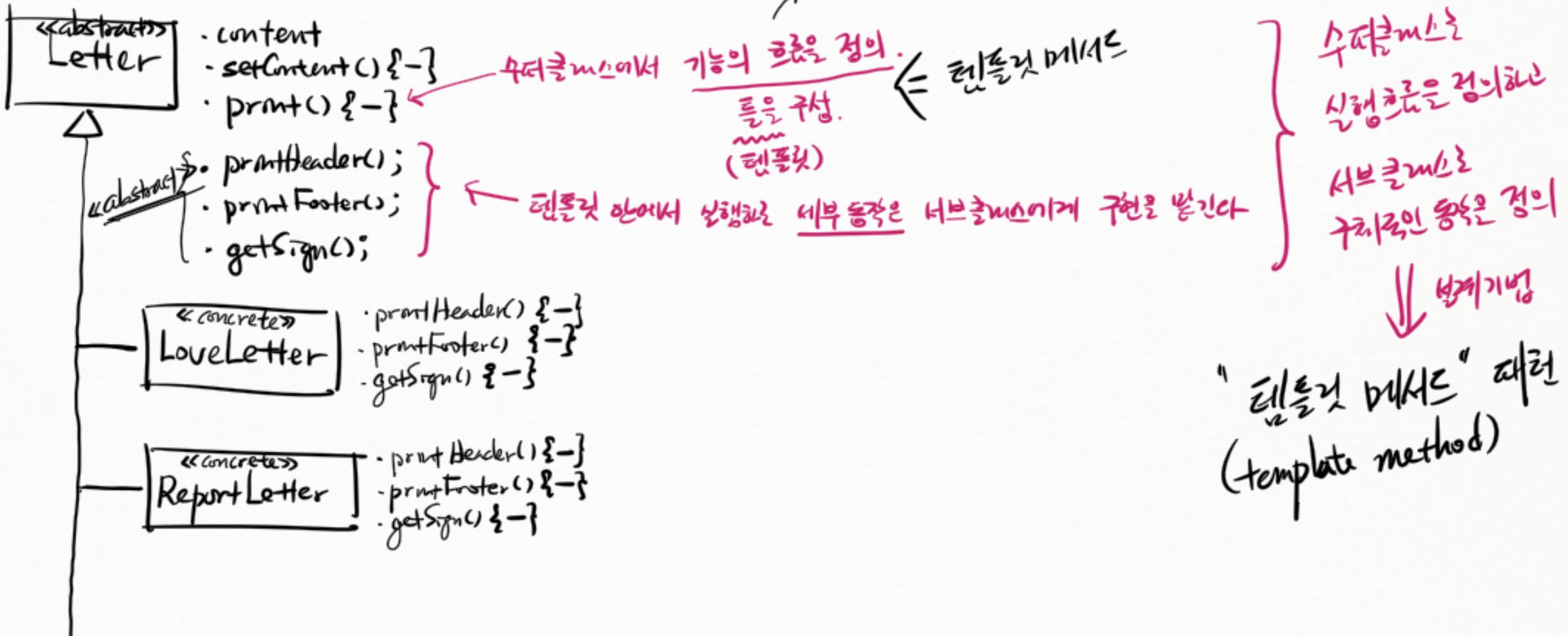


## \* clone 데일



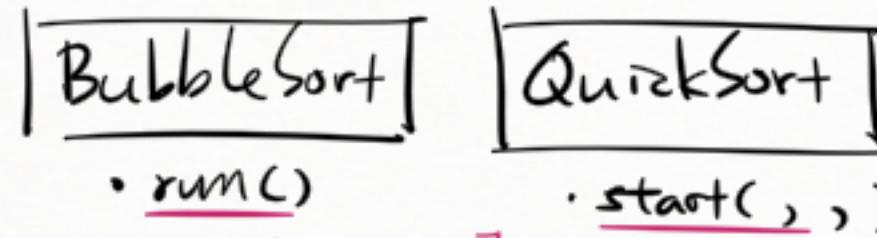
## \* 추상클래스

the skeleton of an algorithm



## \* 추상 클래스와 추상 메서드의 활용

① 각각 단일의 클래스 사용



• run()

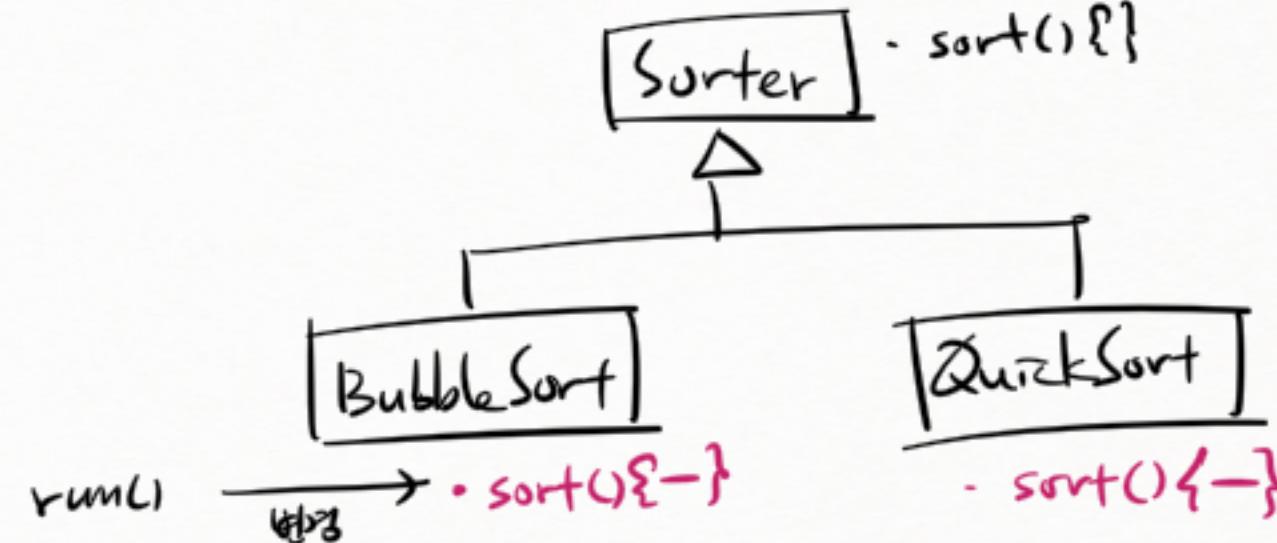
$\uparrow$   
두 개의 정렬 클래스를 같은 부류가 아니기 때문에  
한 개의 print() 메서드를 사용할 수 없다.  
모든 내용은 각자.

• display(BubbleSort, int[]) { }

• display(QuickSort, int[]) { }

개선

② 같은 단일의 클래스로 보기



run()  $\xrightarrow{\text{매개변수}} \cdot \text{sort()}\{-\}$

$\cdot \text{sort()}\{-\}$

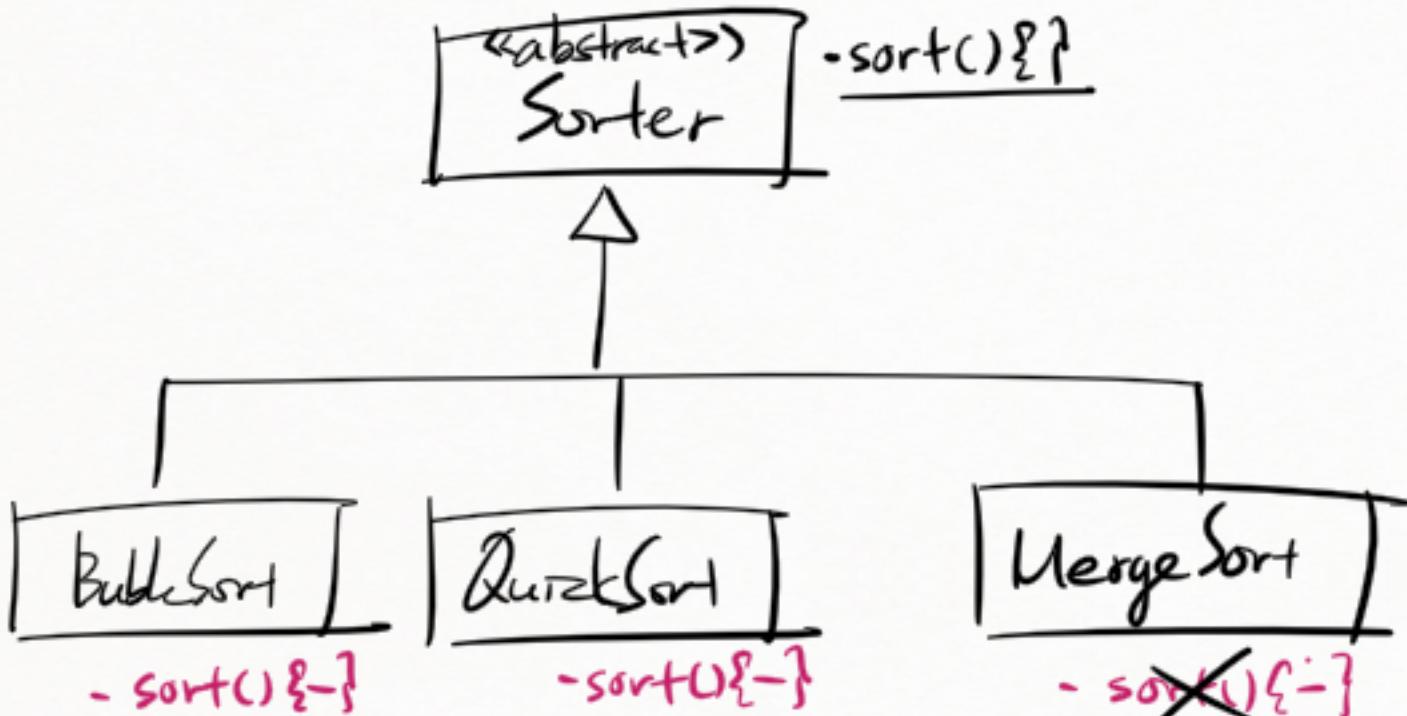
같은 단일화된  
print()를 두 클래스를 위한 대신

• display(Surter, int[]) { }

s.sort();  $\leftarrow$    
  $\cdot$  sort() 메서드  
  $\cdot$  같은 이름의 메서드 호출  
  $\cdot$  같은 이름의 메서드 가능

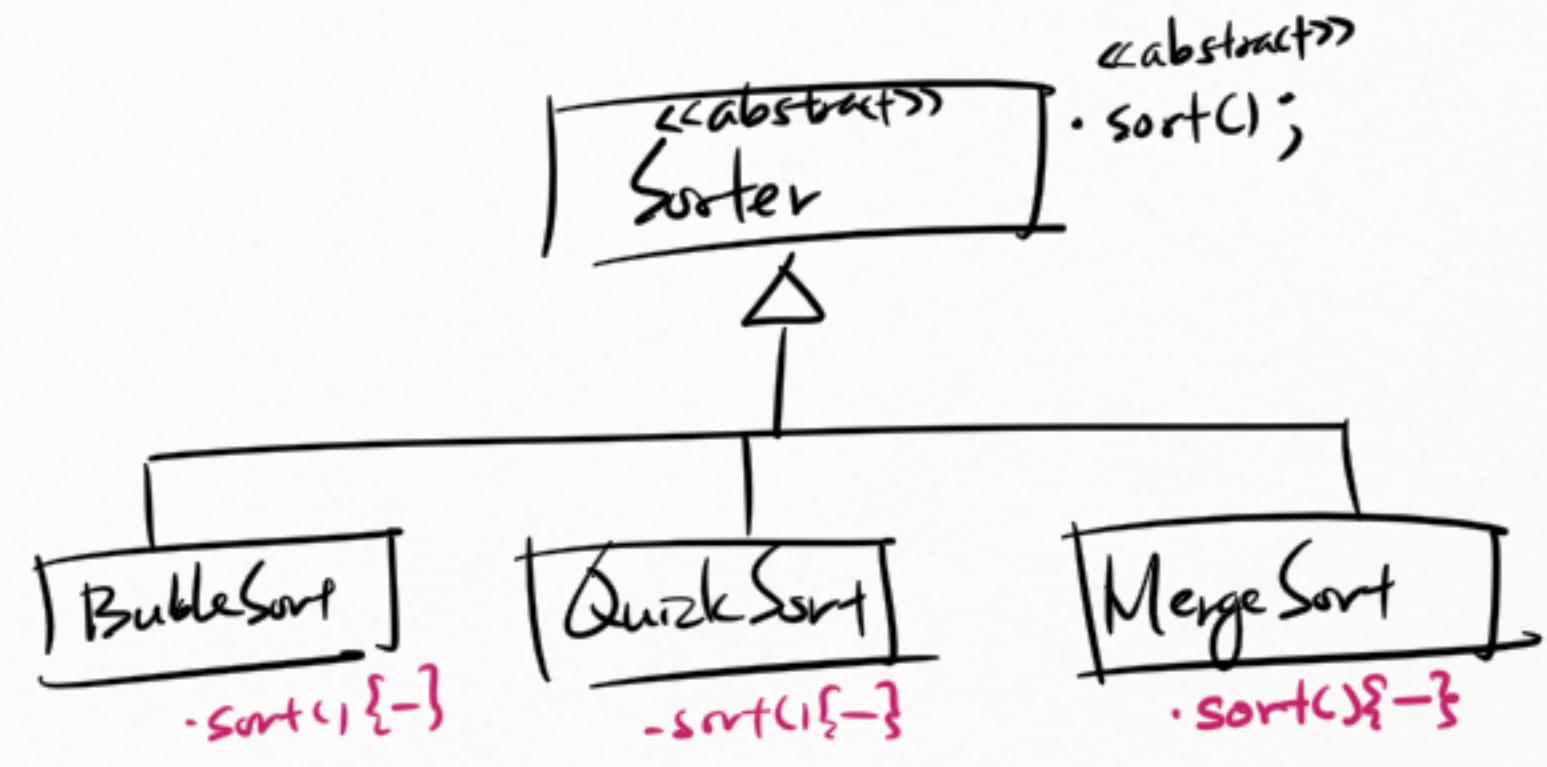
\* 추상클래스와 추상메서드의 활용

③ 추상클래스로 추상클래스로 나누기



```
-point(Sorter, int[])
    s.sort()
```

④ 추상메서드로 분리하는 것

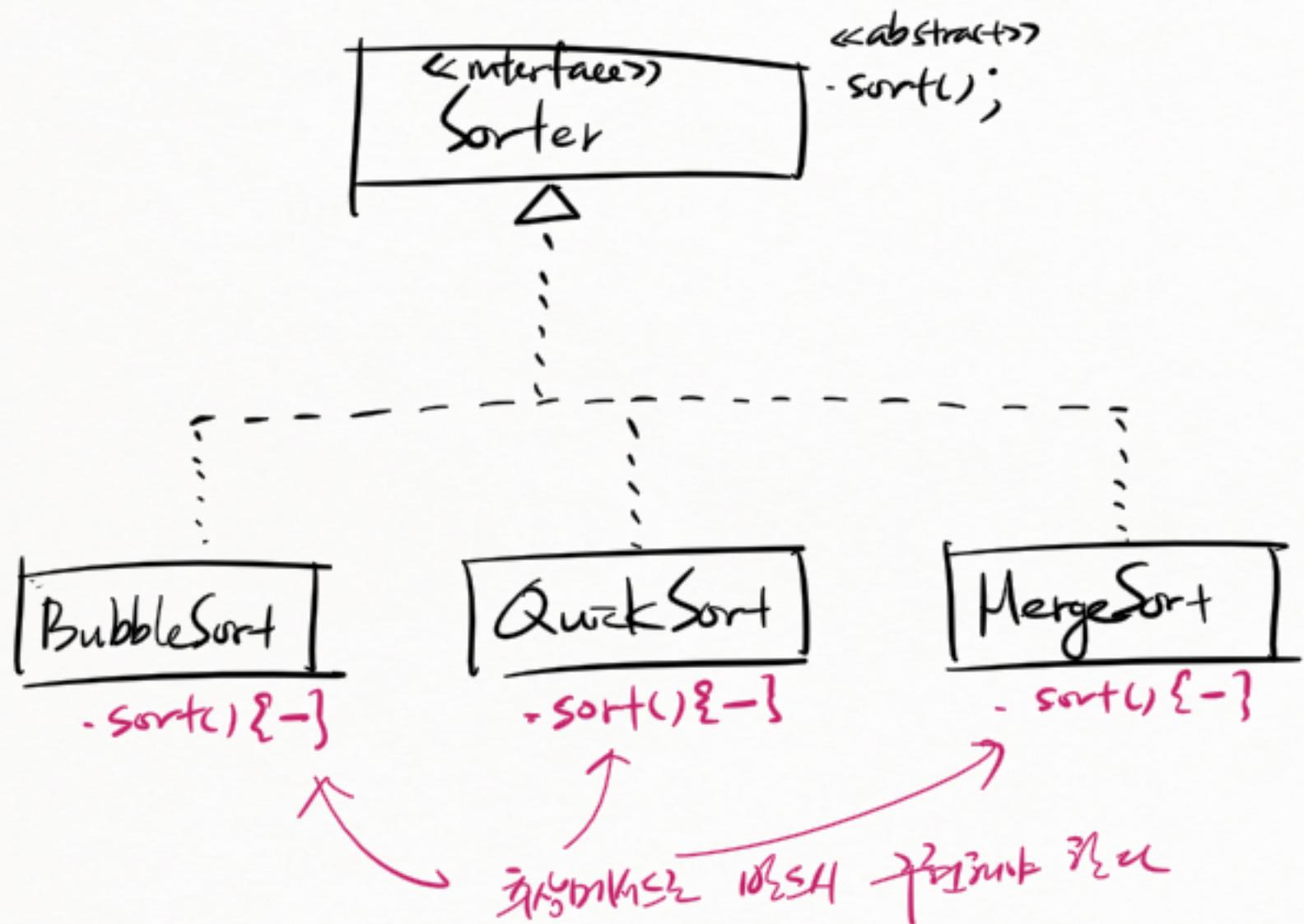


이 클래스를 sort()로  
구현해야 할 경우 가능하다  
Abstract의 sort()는  
아직 정의되어 있지 않음  
구현해야 할 수가 아니라!

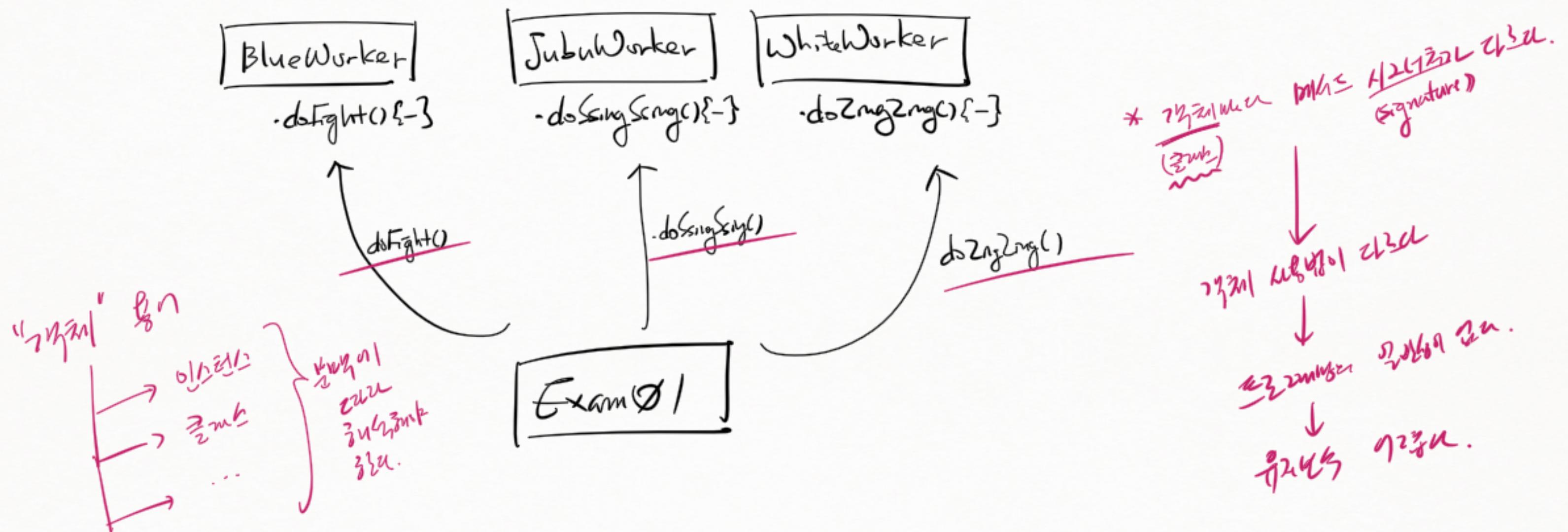
원래 추상 메서드는 implements interface!

\* 추상클래스 만든 인터페이스를 끌 때

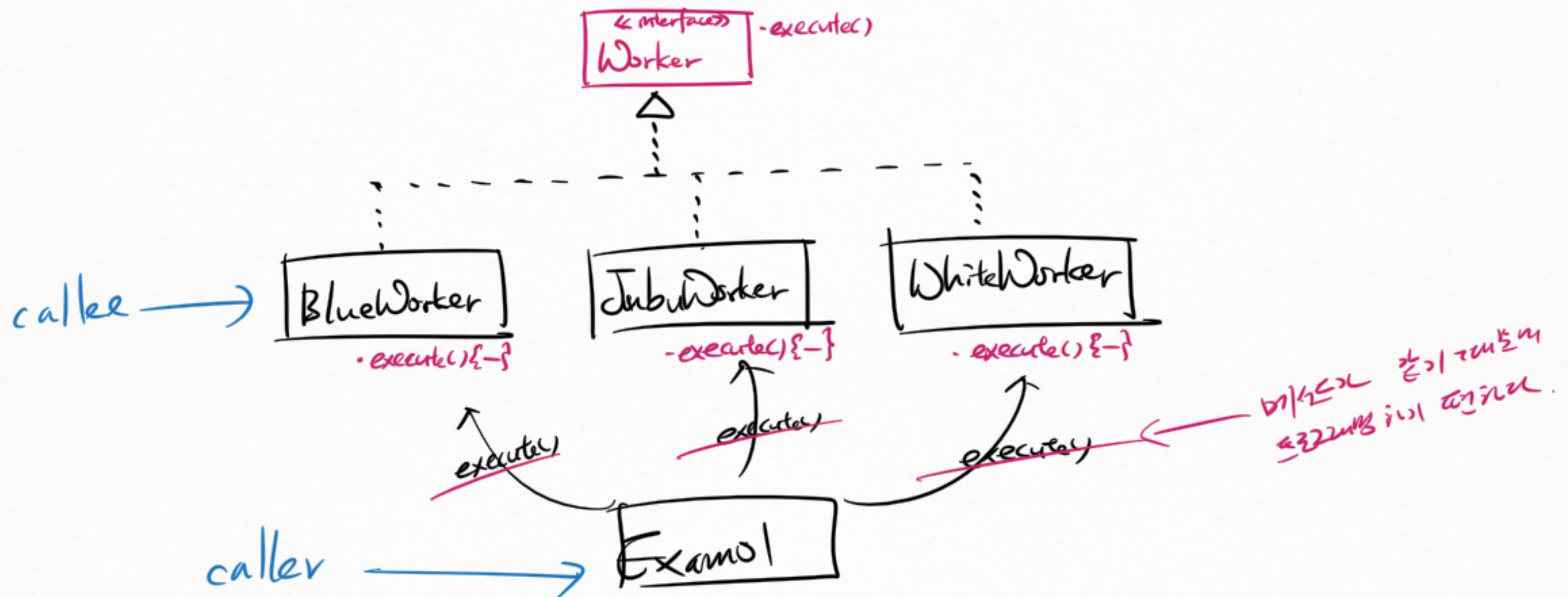
⑤ 추상클래스 만든 인터페이스로 만들기



\* 인터페이스 사용 전 : - oop. ex09. ai. before

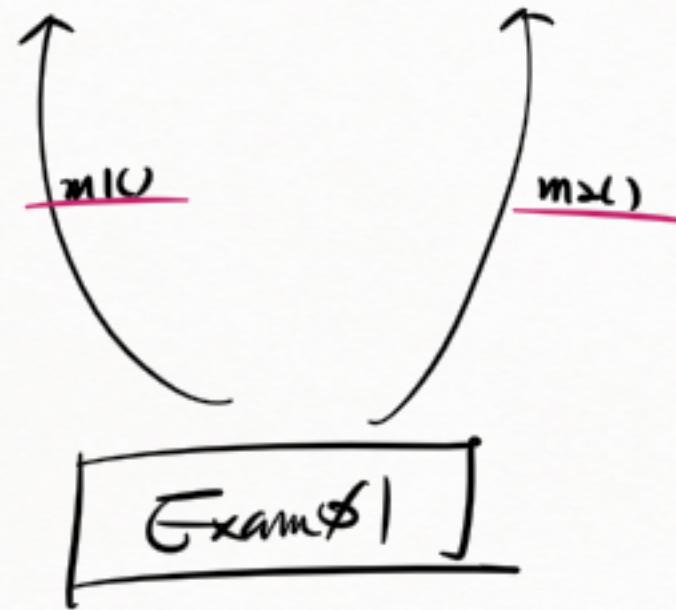
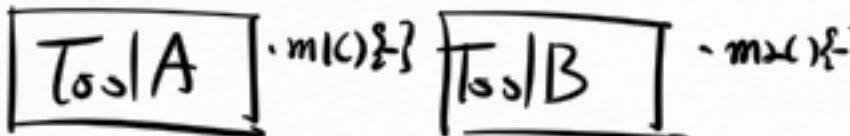


\* 인터페이스 Abstraction : - oop. ex9. al. after



## \* 인터페이스 사용 전 / 후

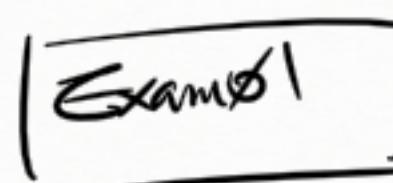
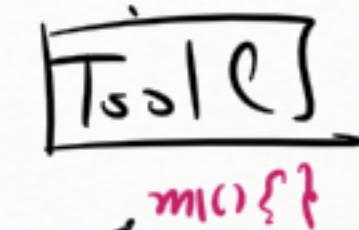
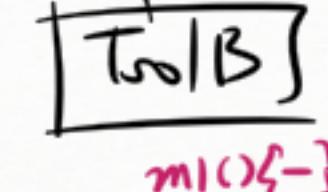
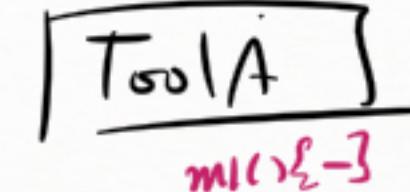
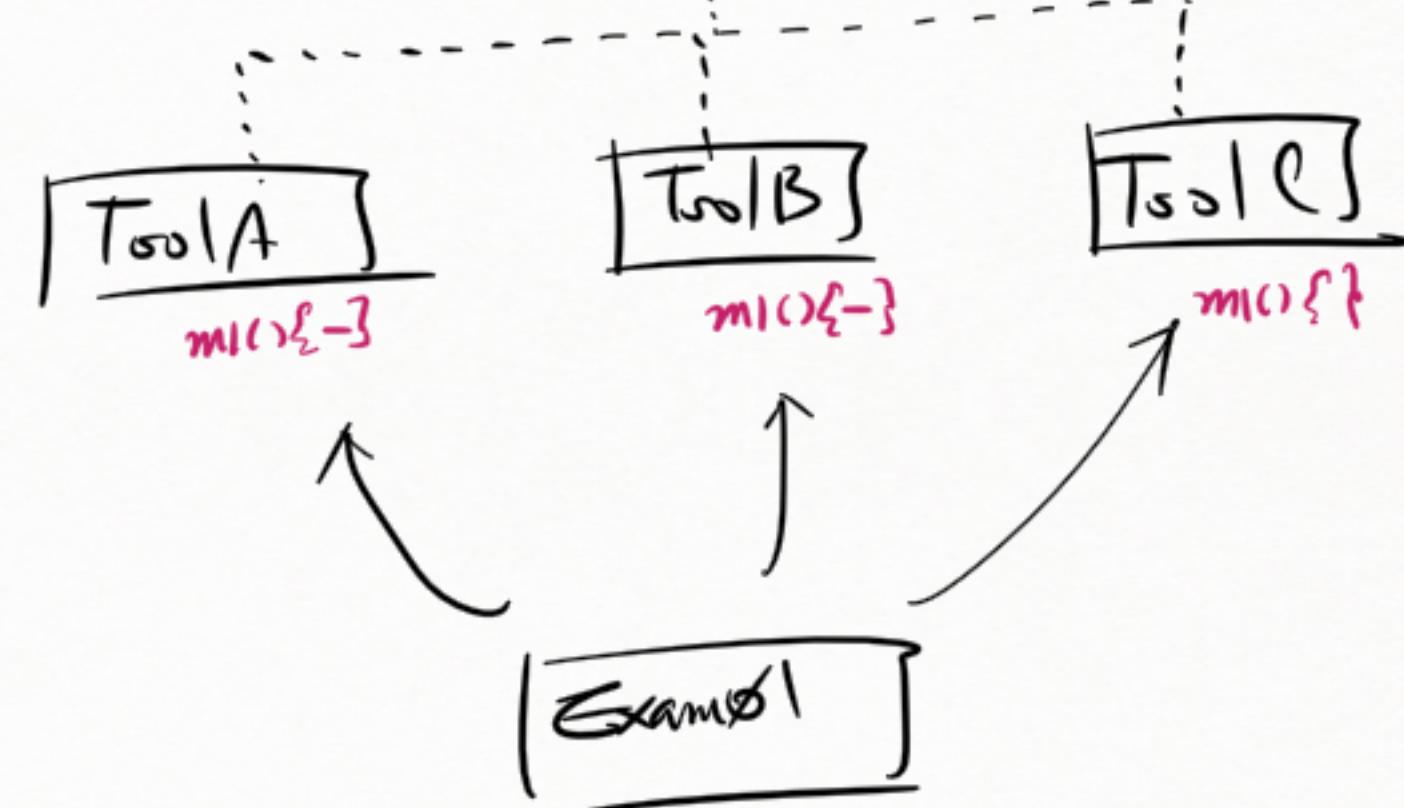
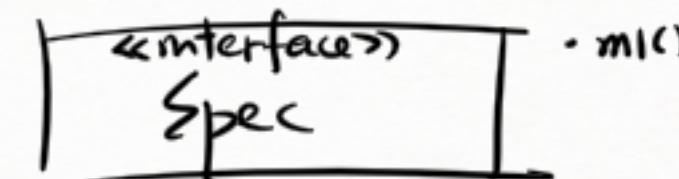
① 사용전



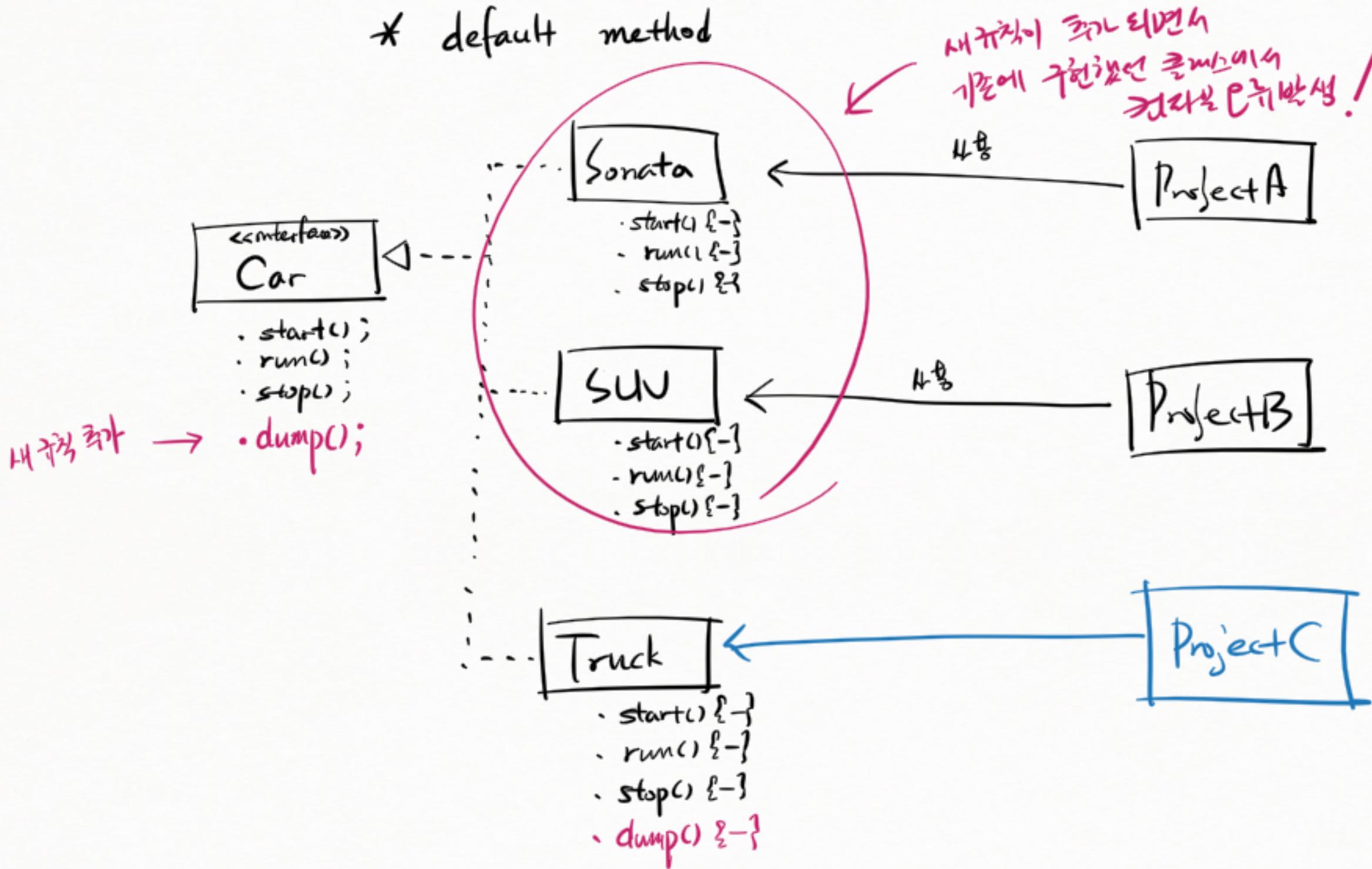
단점  
m1(), m2() 모두가 필요

단점  
유저에게 복잡

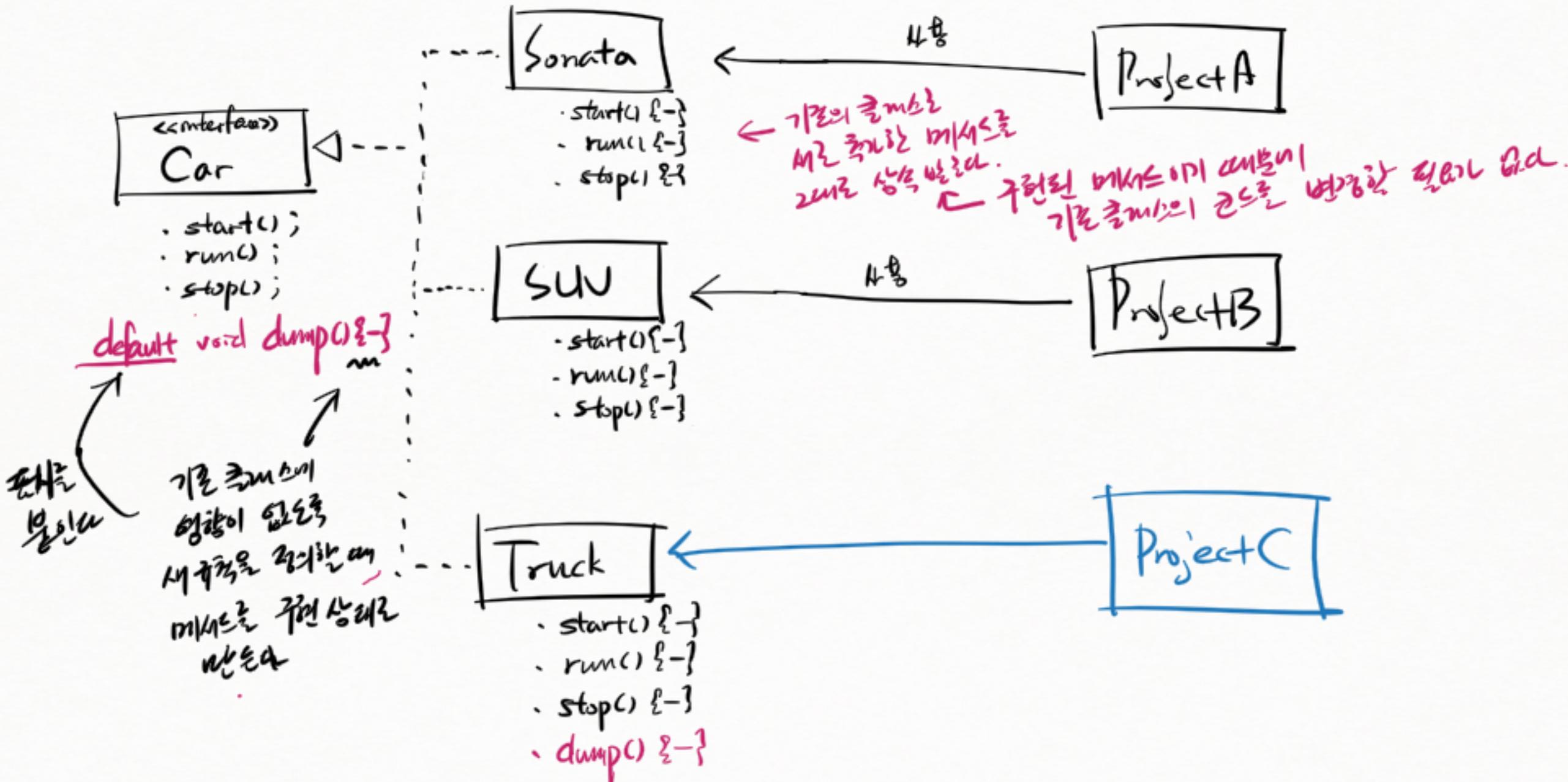
개선 → ② 사용 후



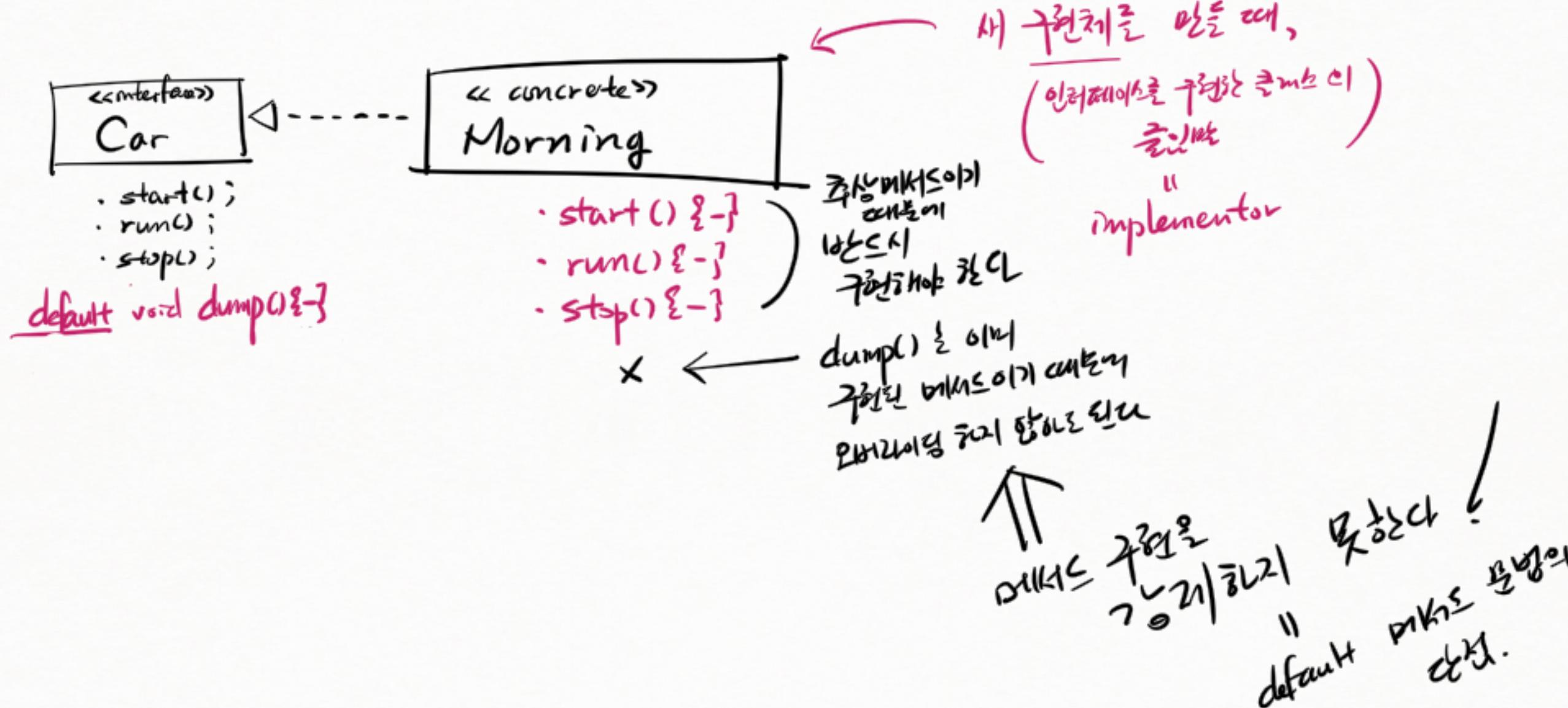
## \* default method



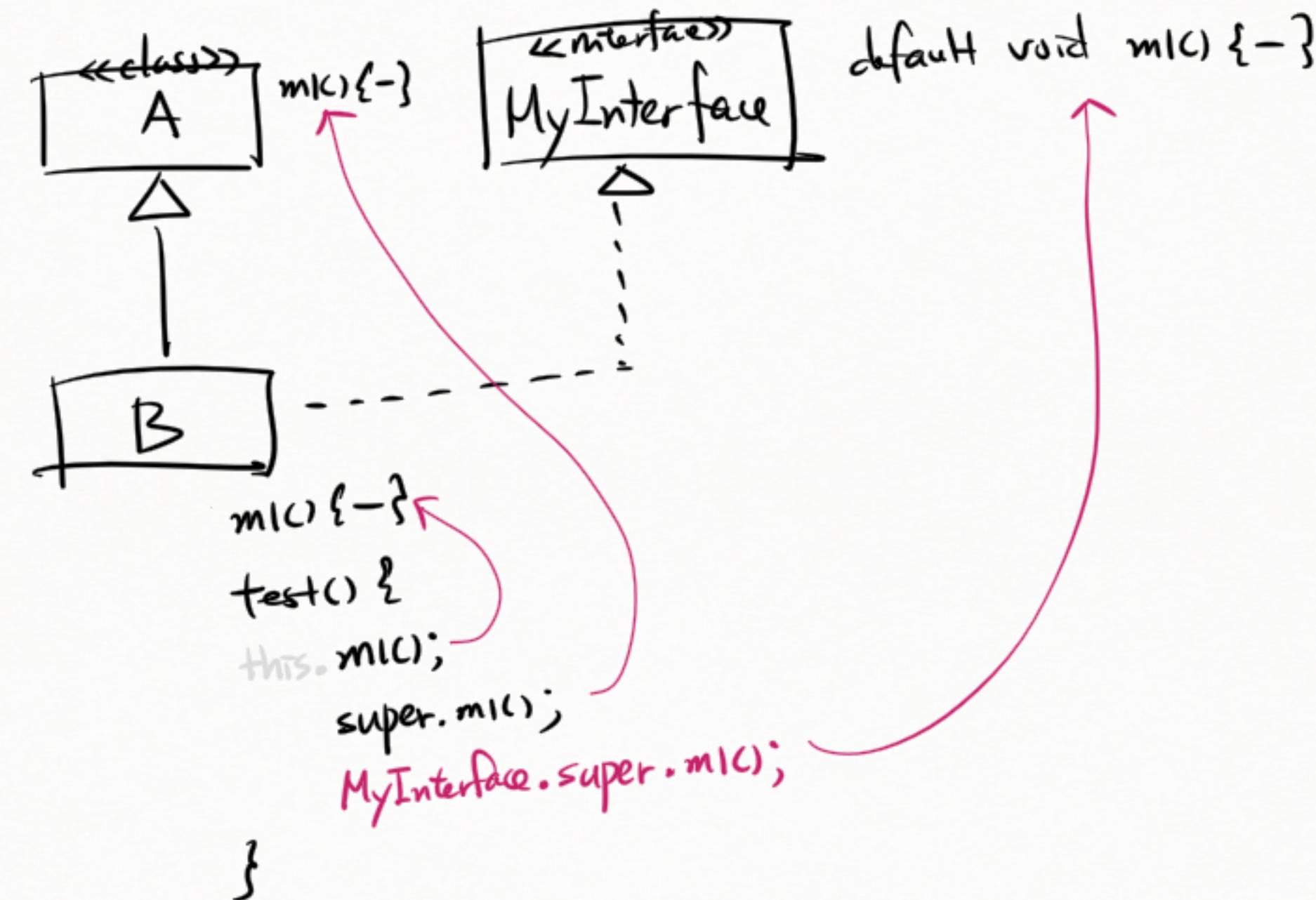
\* default method ← 기존의 구현 클래스의 상황을 유지하면서  
새 규칙을 추가하고 싶을 때,



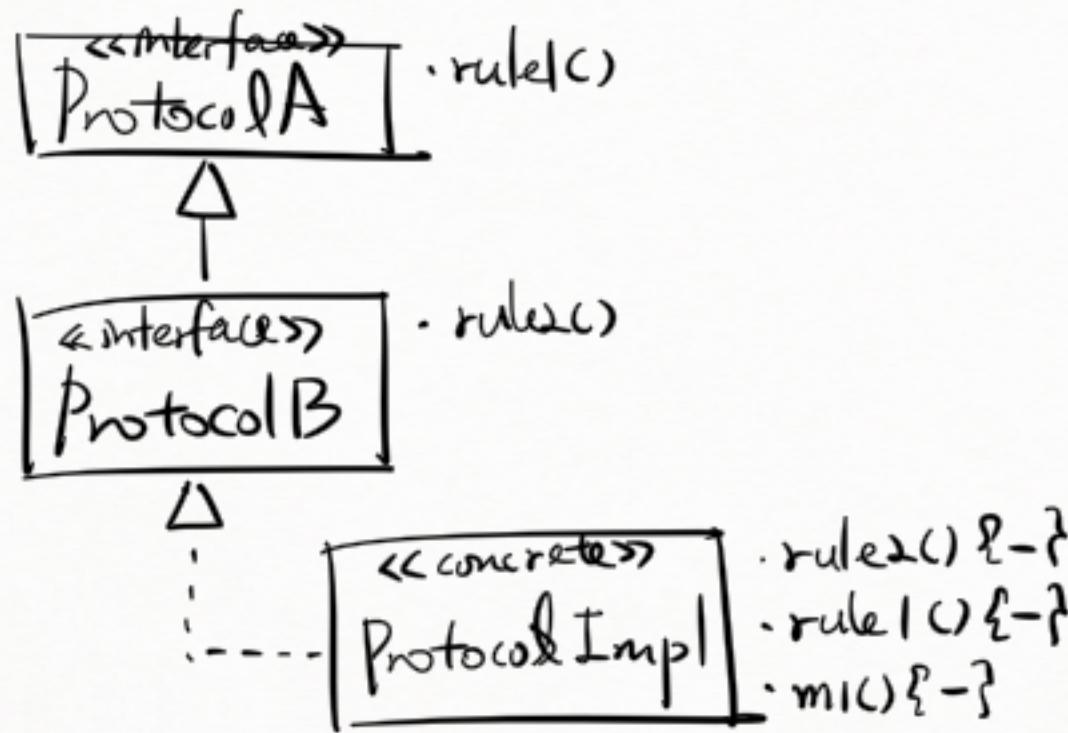
## \* default method o/a 문제점



\* super el. 인터페이스.super



\* 인터페이스 상속 - oop.ex09.c.\*

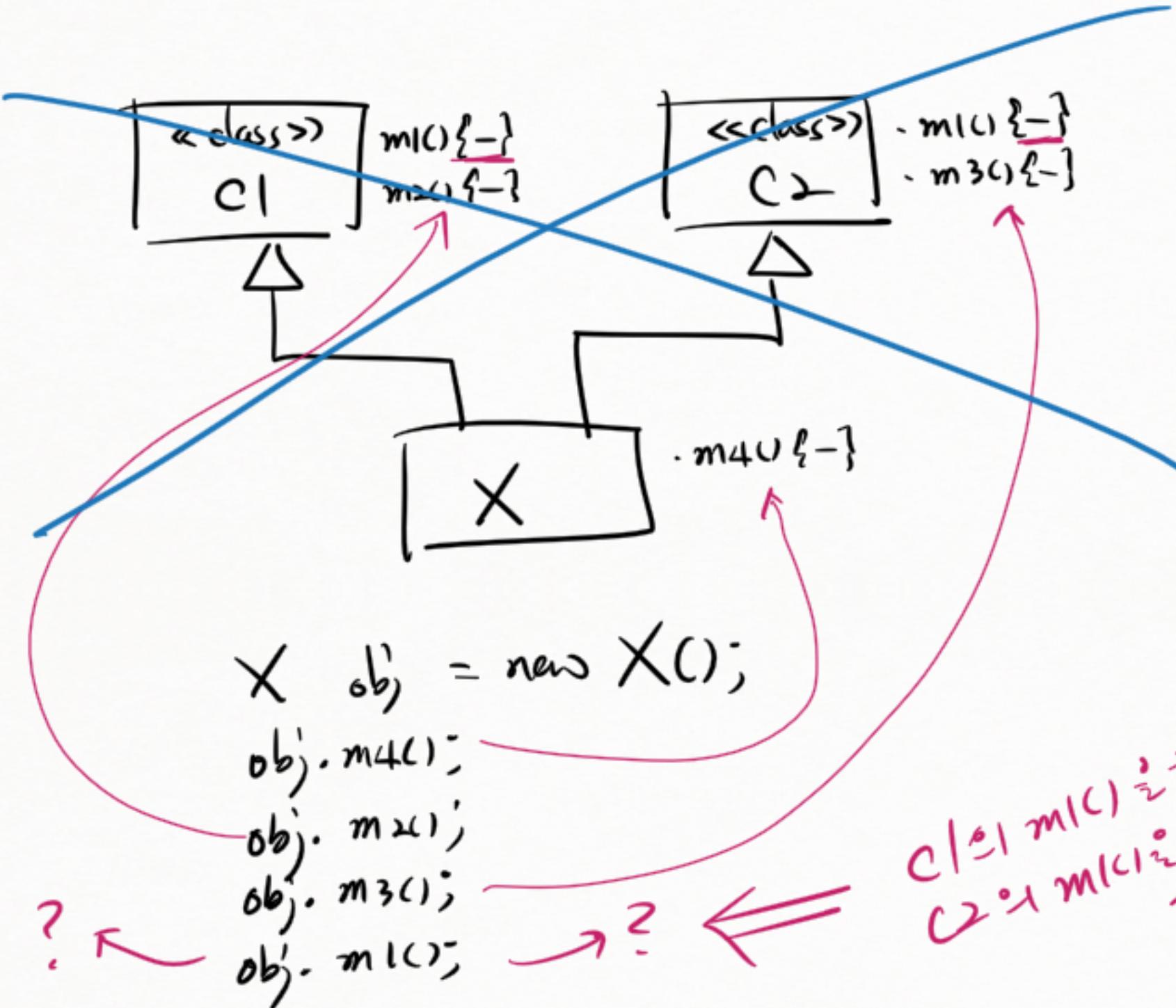


ProtocolImpl obj = new ProtocolImpl();  
obj.m1(); // ok  
obj.rule2(); // ok  
obj.rule1(); // ok

ProtocolB b = obj;  
**b.m1(); // error**  
**b.rule2(); // ok**  
**b.rule1(); // ok**

ProtocolA a = obj;  
m1(); // error  
rule2(); // error  
rule1(); // ok

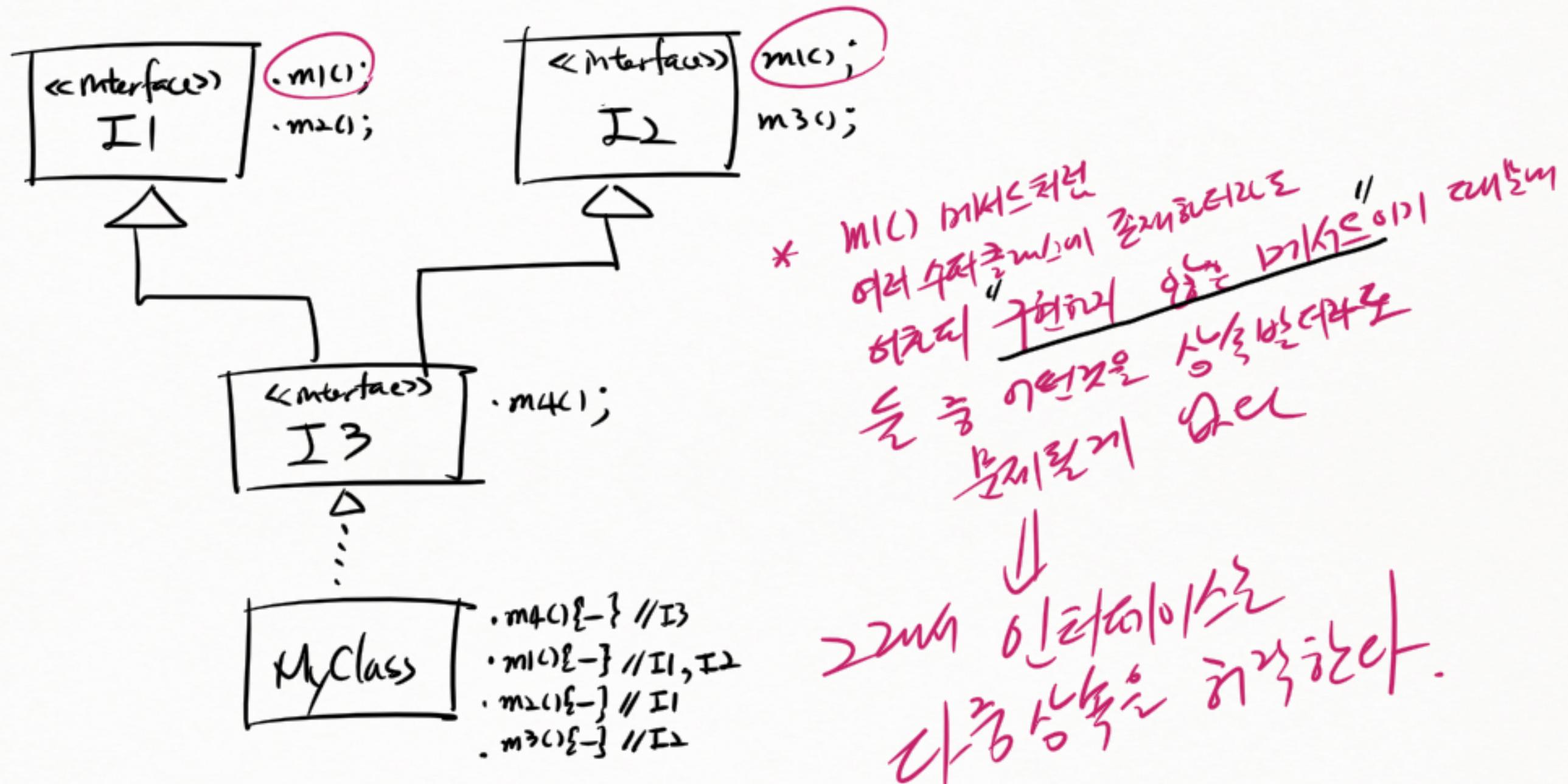
\* ~~클래스~~ 다음 상속과 인터페이스 다음 상속



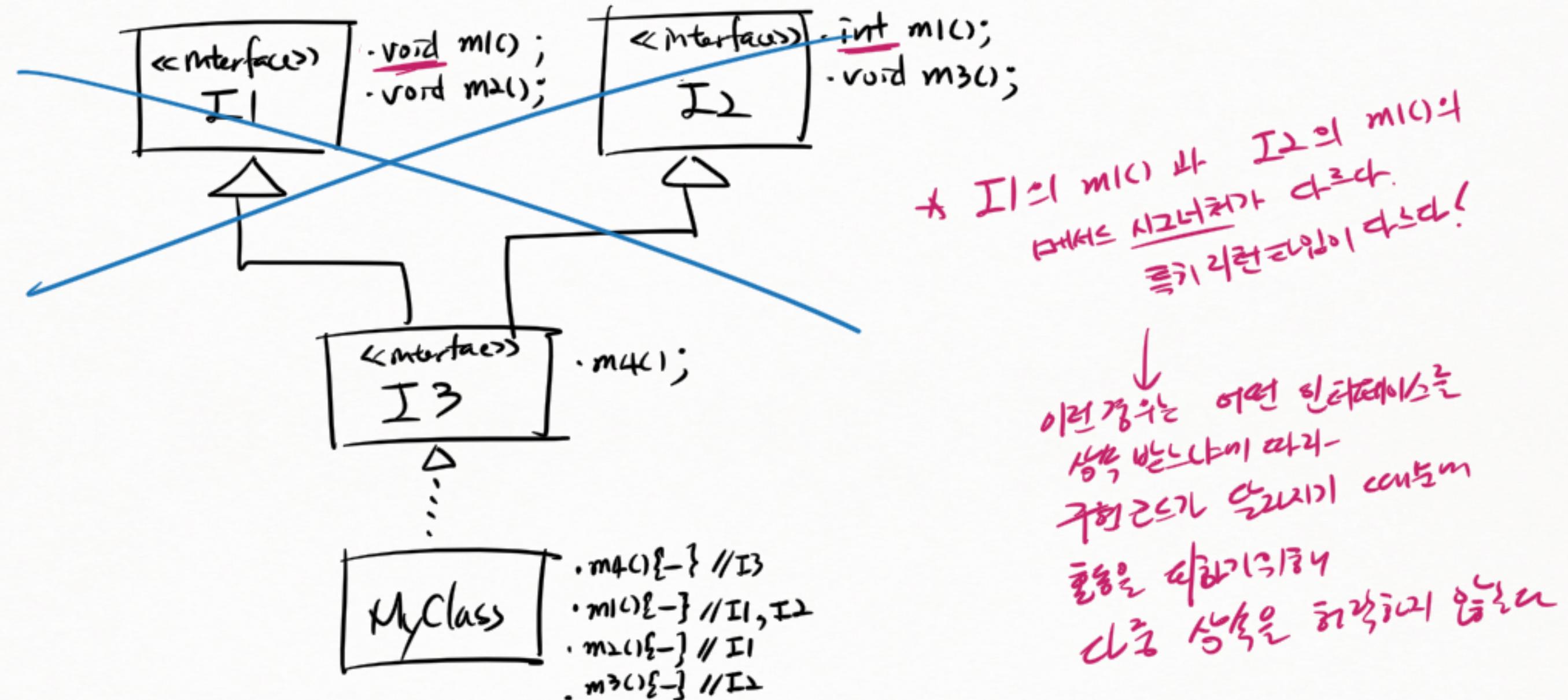
2차 상속  
제한된 상속  
제한된 상속

CH :  
m1()은 허용되는가?  
m1()은 허용되는가?

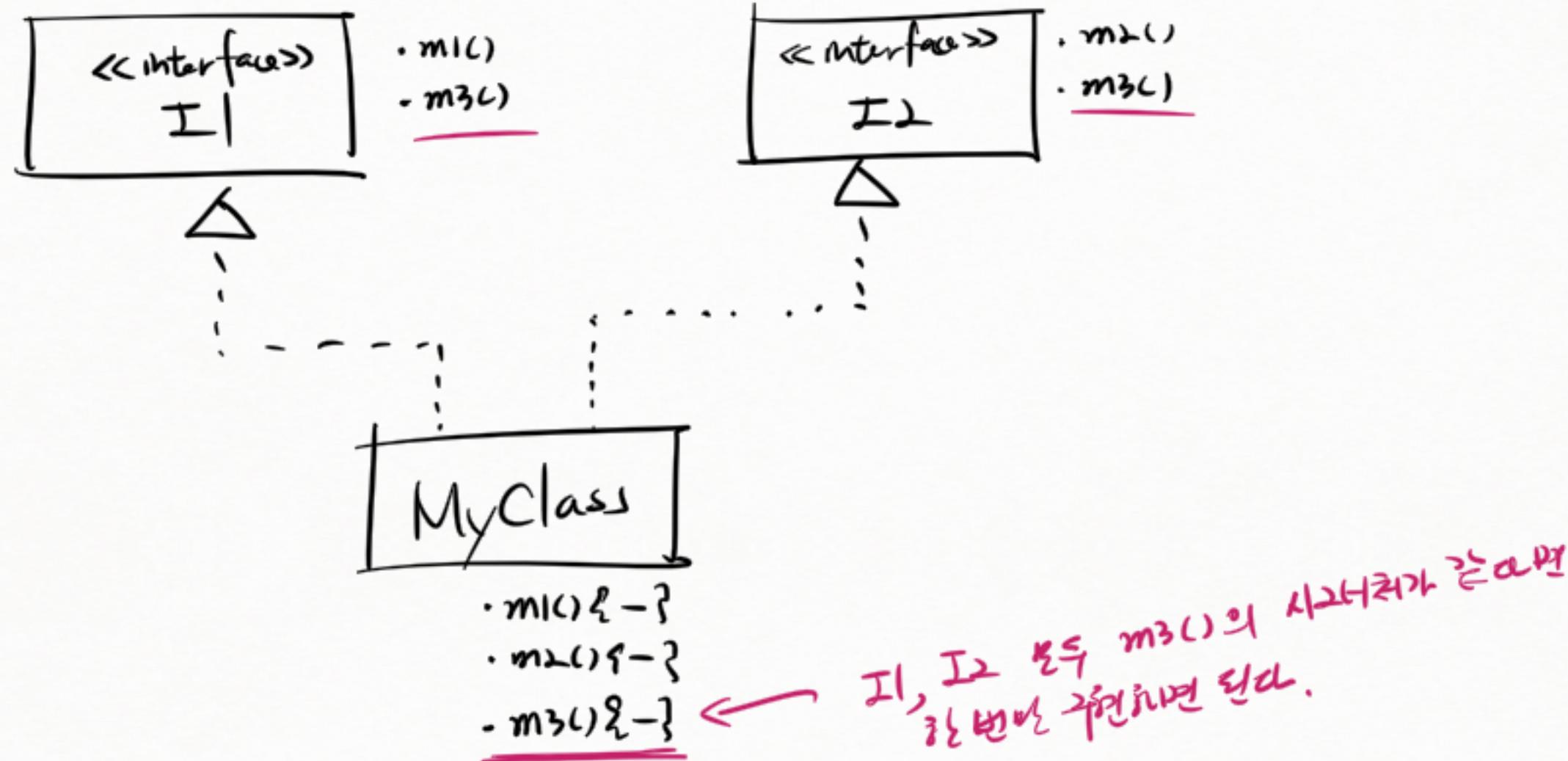
\*  $\frac{1}{2}$ m1 다중 상속과 인터페이스 다중상속



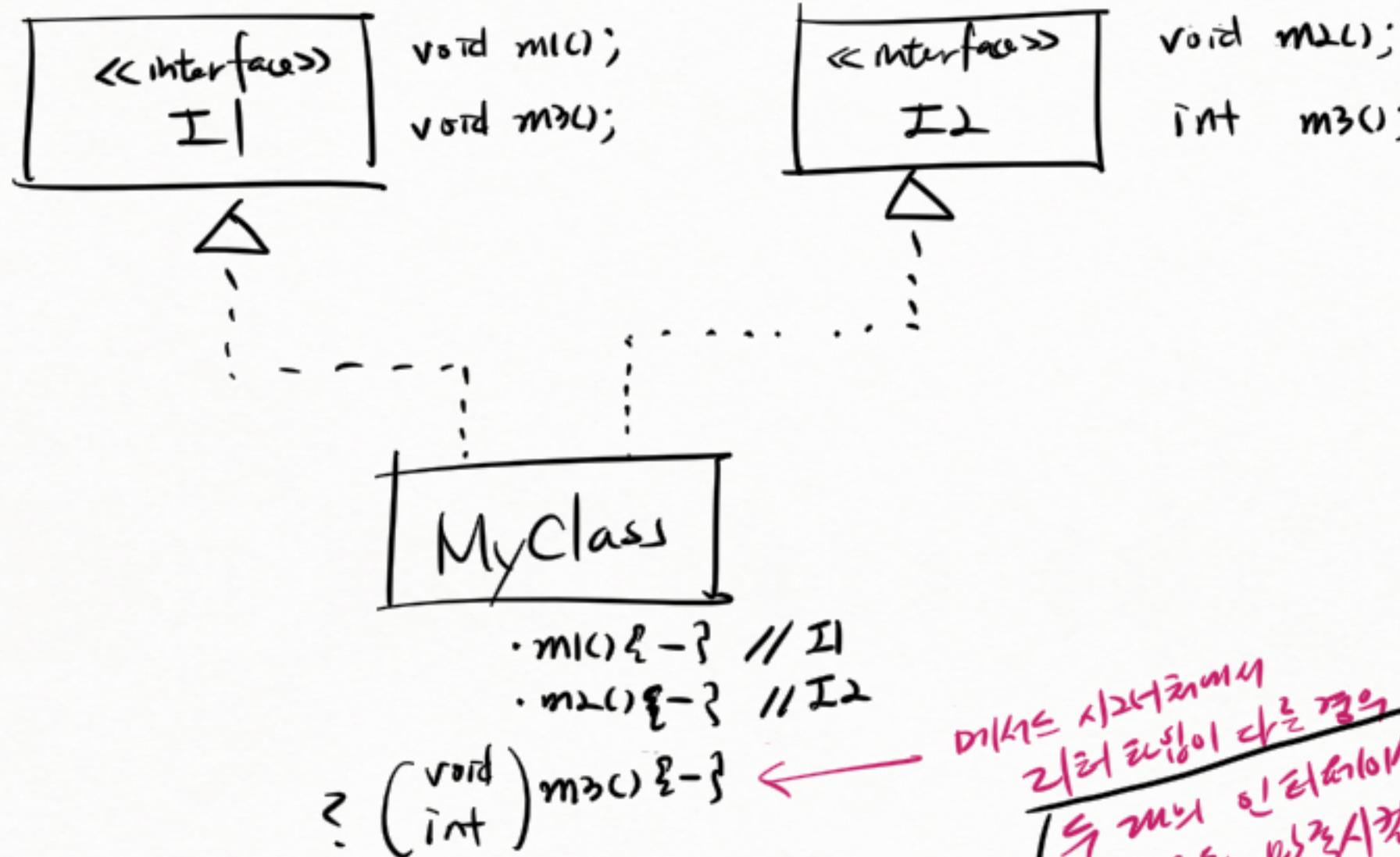
\* 인터페이스 다중 상속 불가!



## \* 인터페이스의 e종 특성



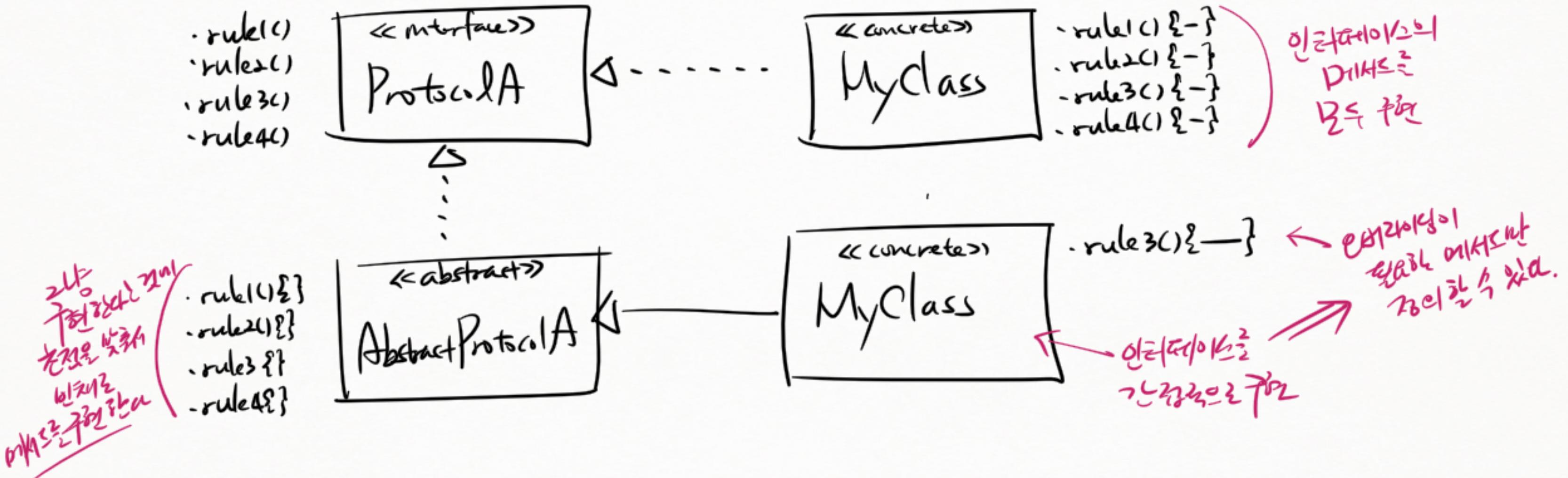
\* 인터페이스는 다형성을 위한!



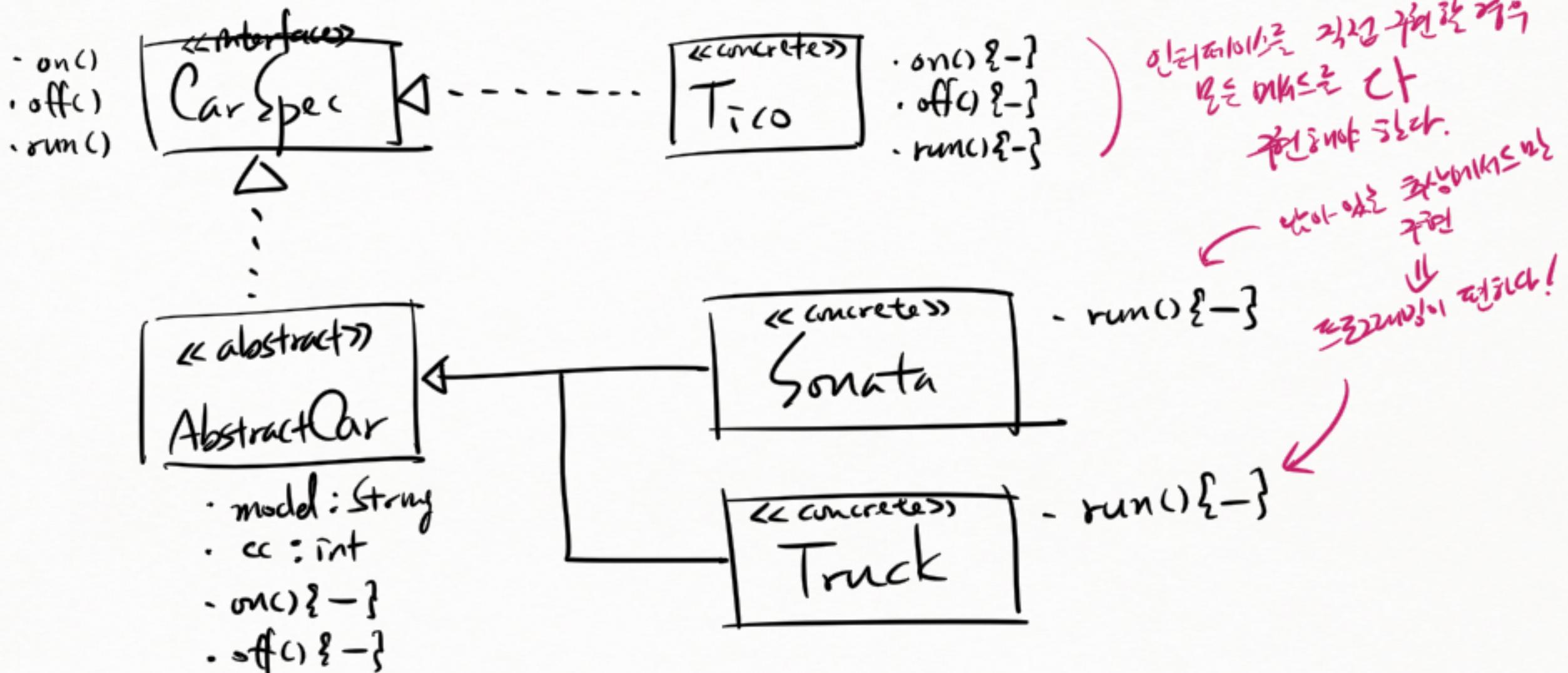
MyClass는 시그니처를  
갖고 있는 인터페이스를  
구현하는 경우에  
만약에 인터페이스가  
정의한 멤버를 정의하지  
않았을 때는 ?

MyClass는 시그니처를  
갖고 있는 인터페이스를  
구현하는 경우에  
만약에 인터페이스가  
정의한 멤버를 정의하지  
않았을 때는 ?

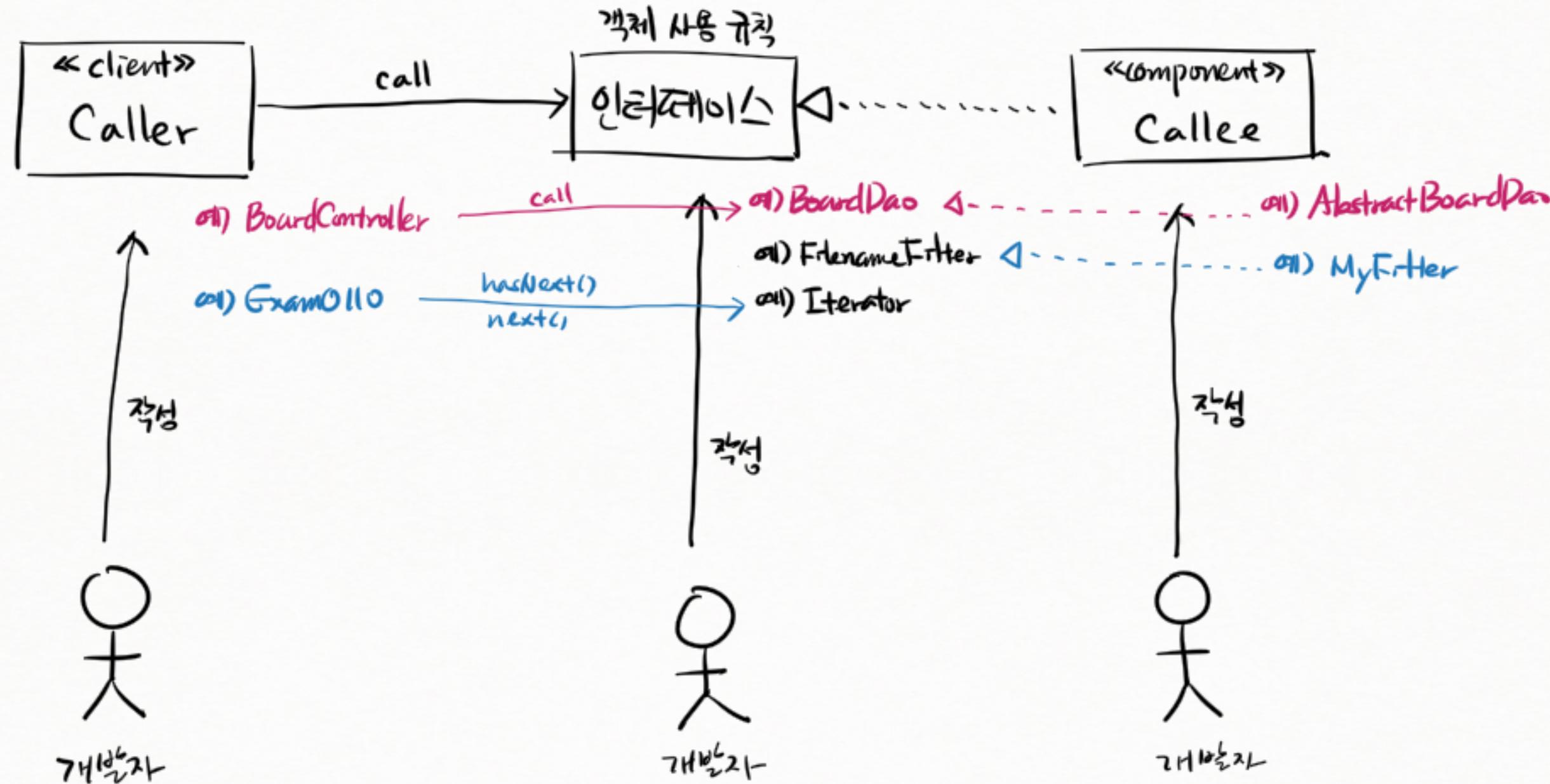
## \* 인터페이스와 추상 클래스의 혼란



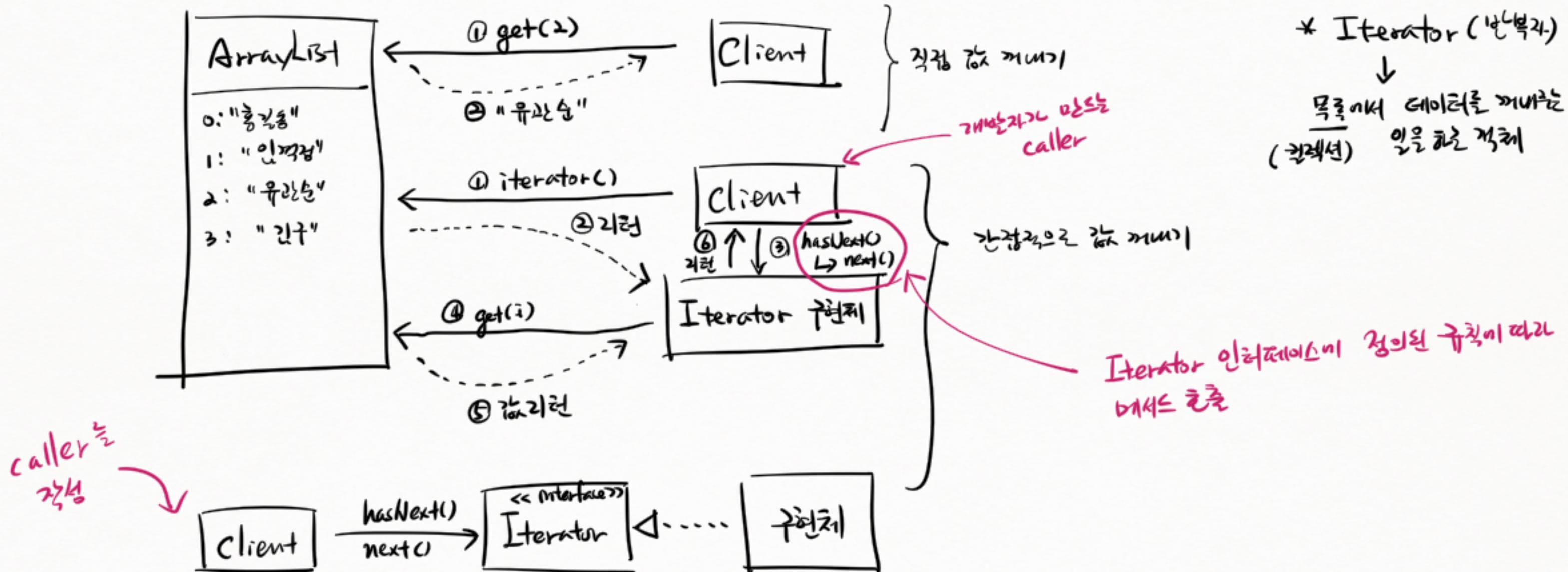
\* 인터페이스와 추상클래스 활용 예)



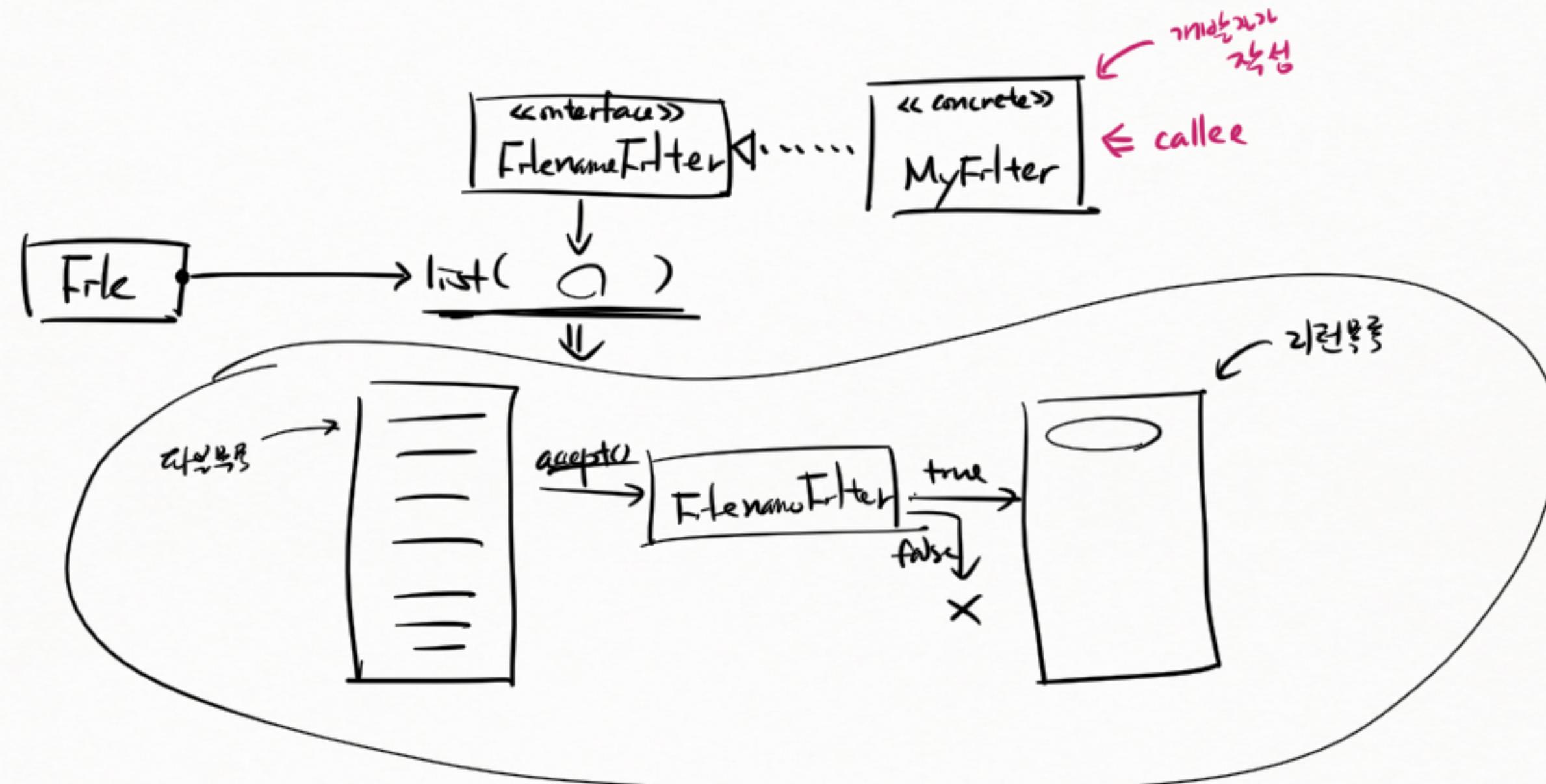
\* 인터페이스와 구현체



## \* Caller 개발 입장



\* callee 2 번째 괄호

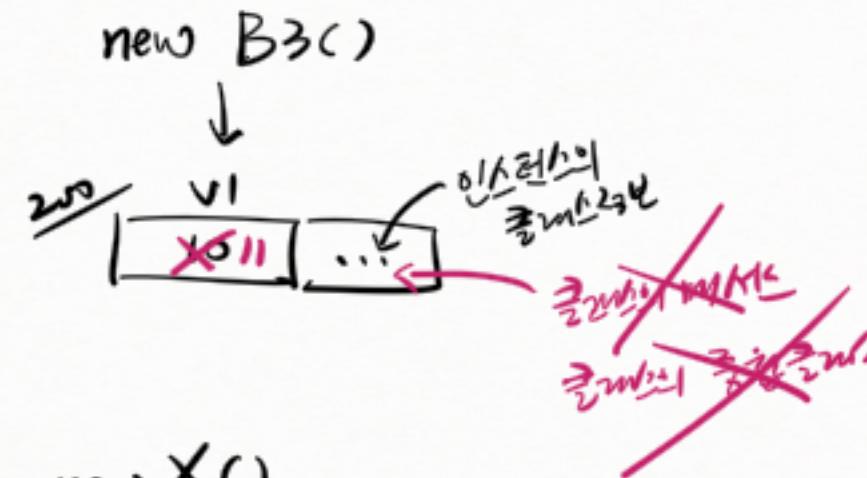


\* inner 클래스의this  
(com.romcs. oop. ex1. c. Exam0230)

B3 outer = new B3()

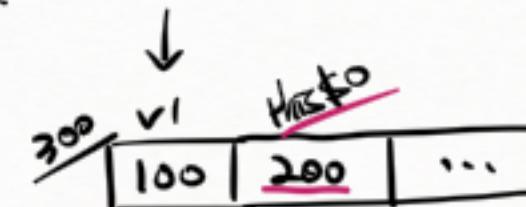
outer

200



B3.X x1 = outer.new X()

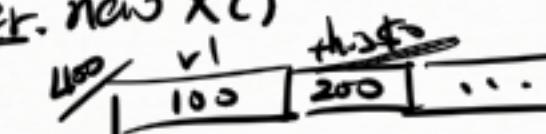
x1  
300



x1.test()

B3.X x2 = outer.new X()

400



Method Area

void test()  
this = 300  
v1 = 1000

출23: v1 → 1000

출23: this.v1 → 100  
300

출23: B3.this.v1 → 11  
↓  
this →  
100

B3 outer2 = new B3()

outer2  
500

v1  
x22  
...

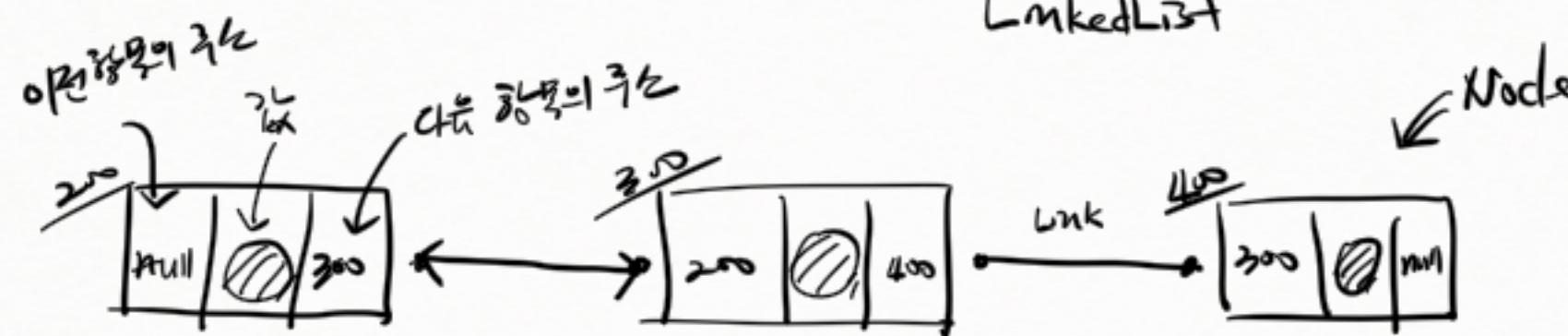
B3.X x3 = outer2.new X()

x3  
600

v1  
this  
...

}

## \* Linked List



head 200

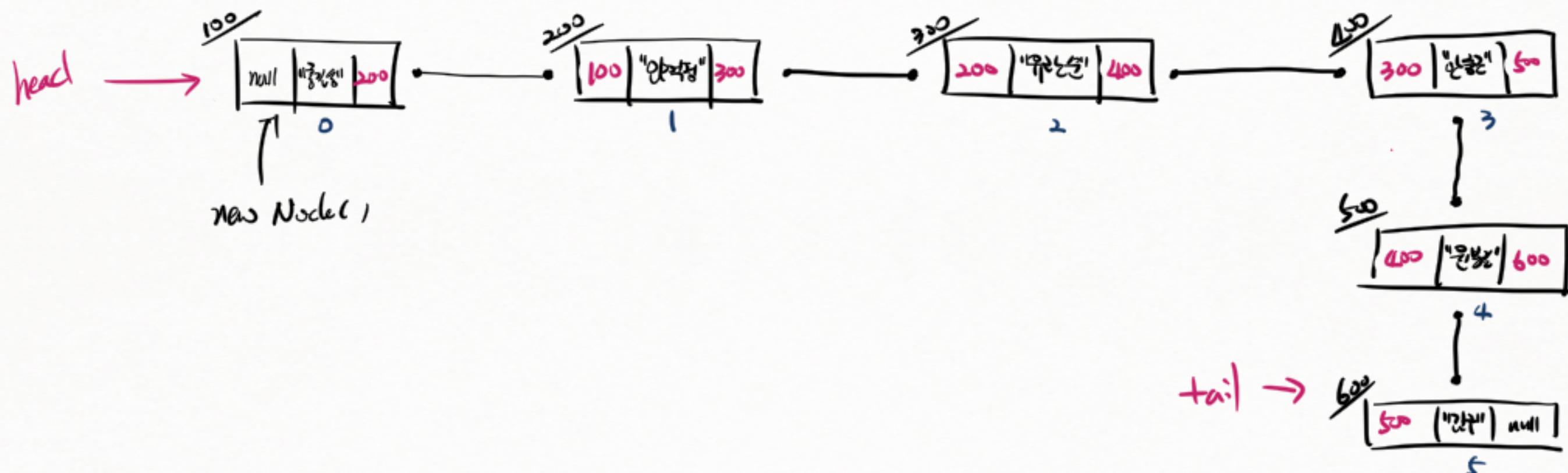
tail 400

## \* LinkedList - 鏡像串

head [100]

tail [600]

size [1]

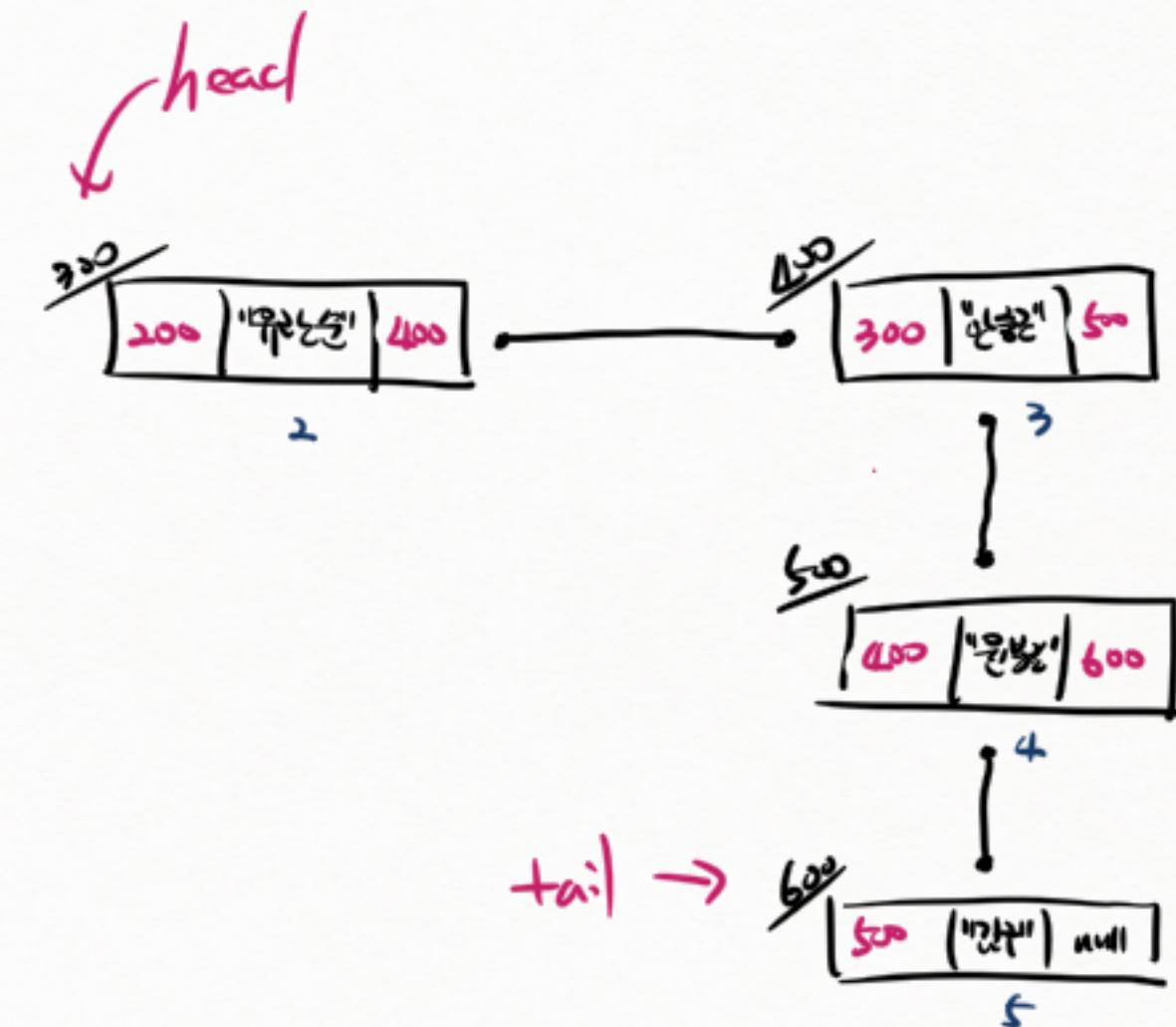
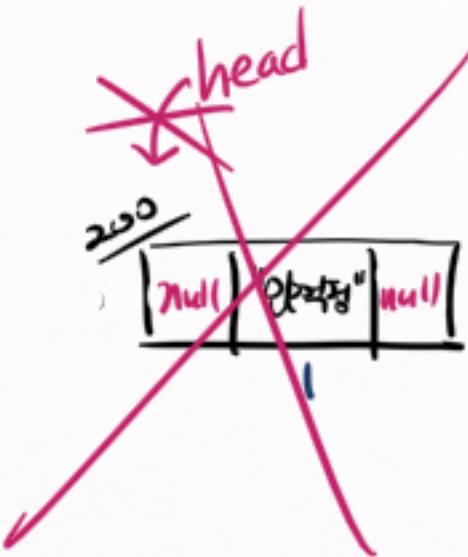
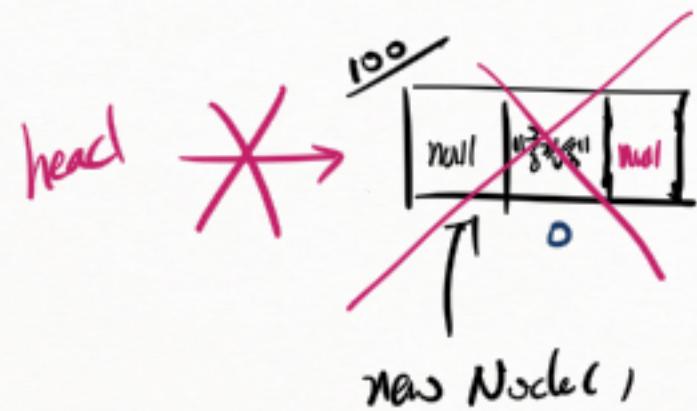


## \* LinkedList - 2번 삭제 (맨 끝 값)

head [300]

tail [600]

size [1]



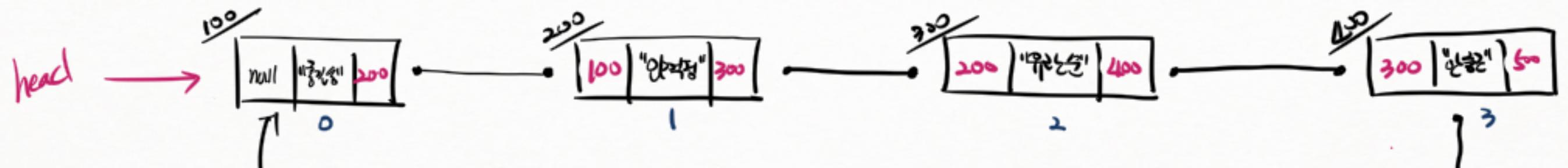
}  
 remove(0);  
 remove(0);

## \* LinkedList - ဇုန် လုပ် (မြင် လိပ်)

head [100]

tail [500]

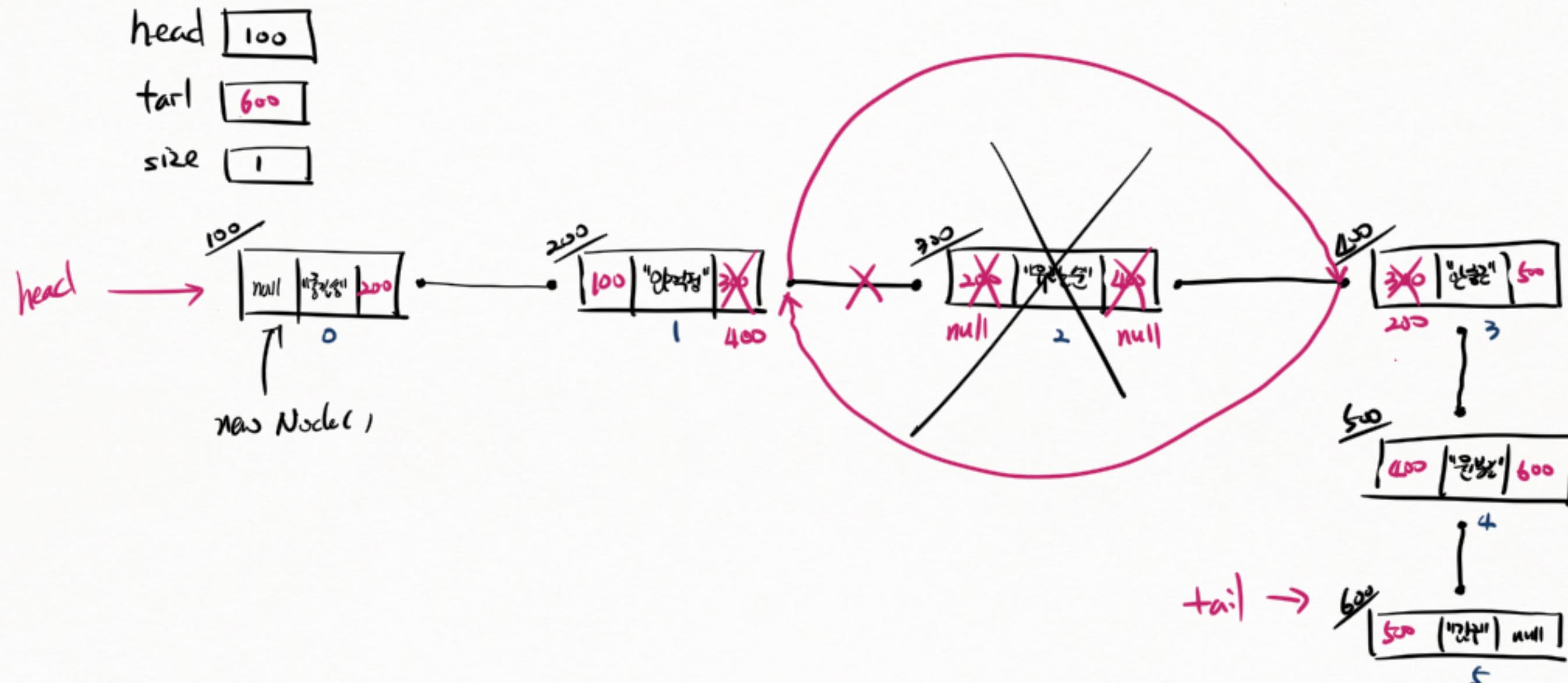
size [1]



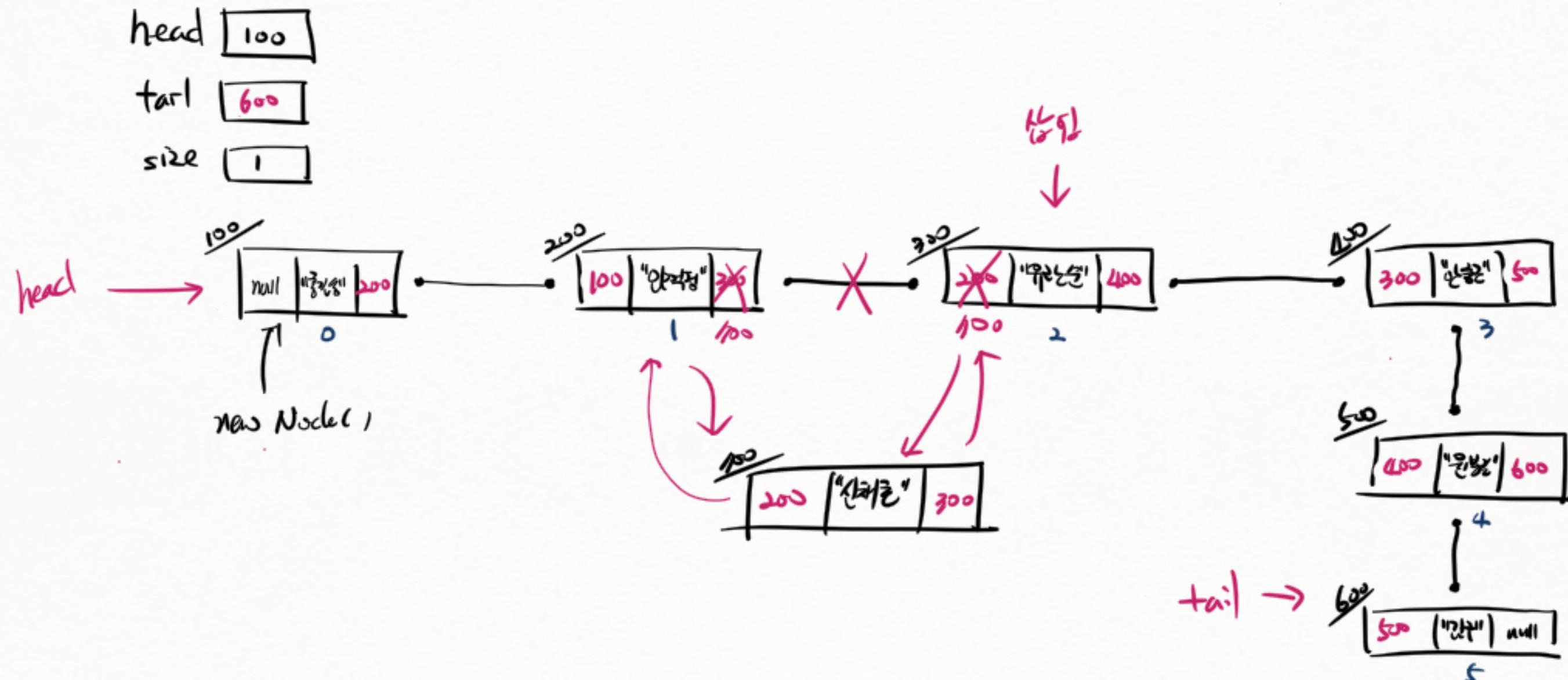
`new Node(1)`



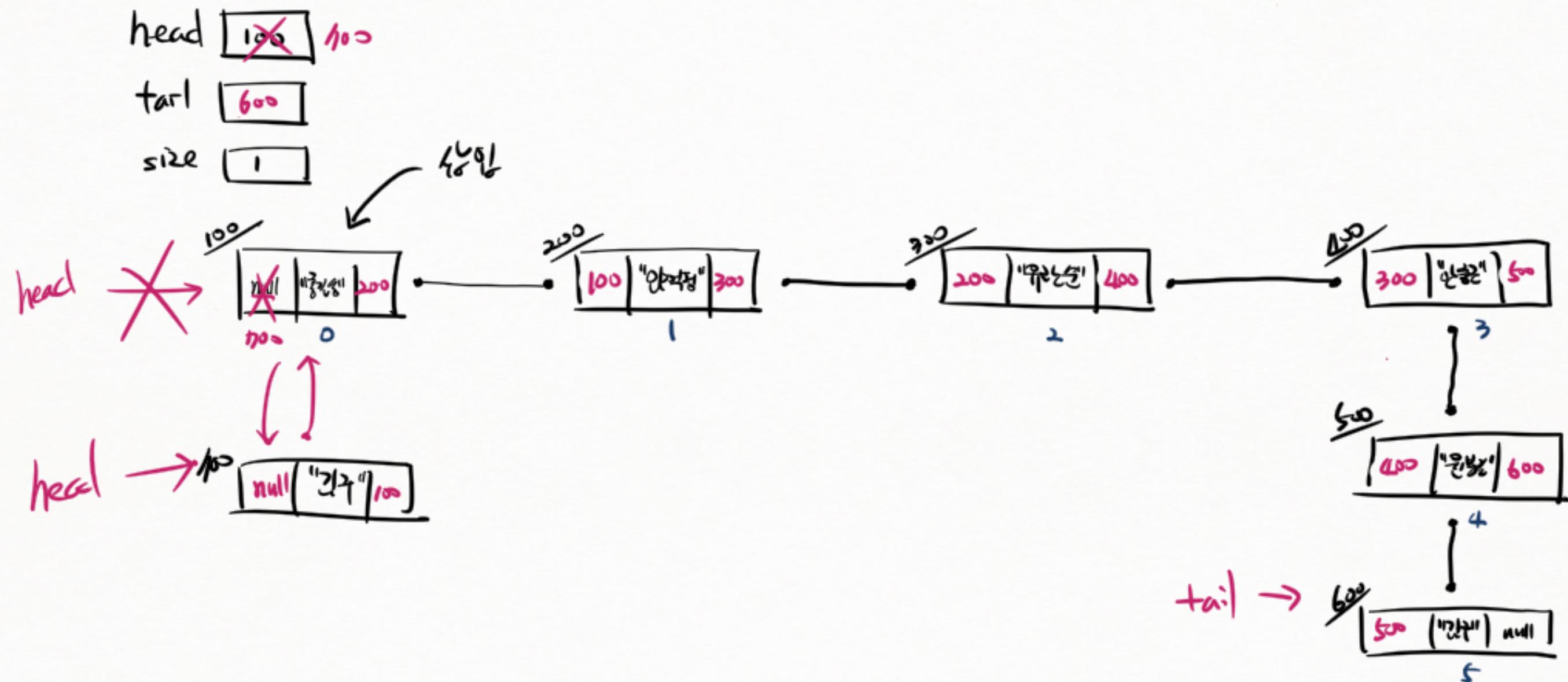
# \* LinkedList - یۆر اەلەدە ( ىچىلىق ئۆز )



## \* LinkedList - 2번 풀이!



## \* LinkedList - ဇုနစ် (မျဉ်)

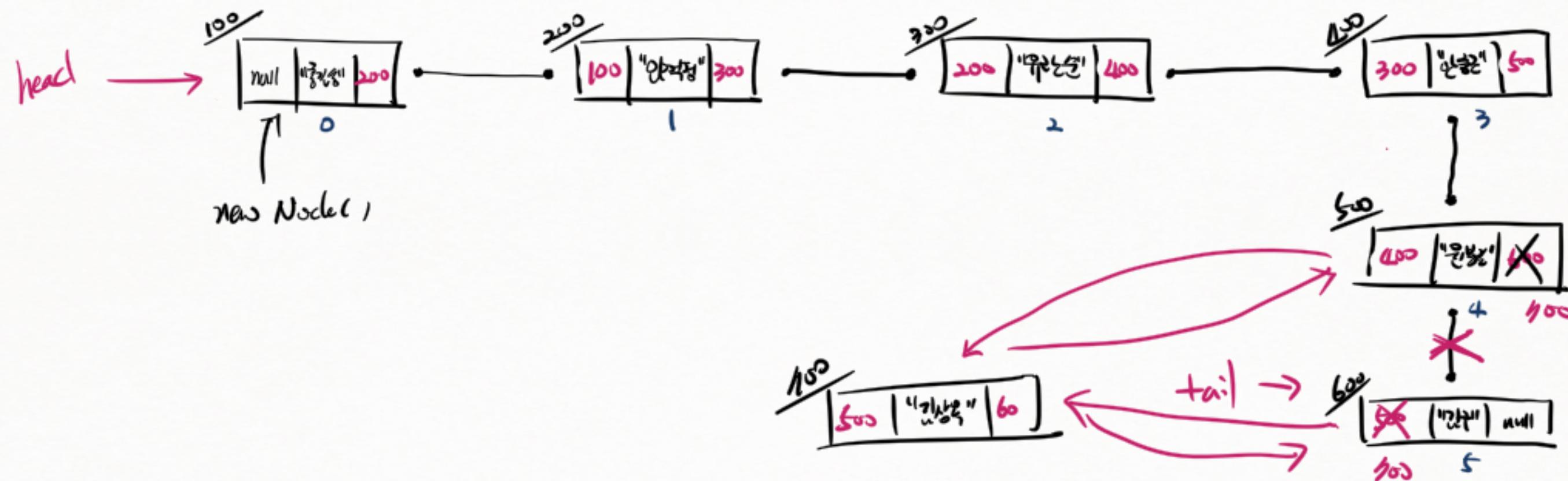


## \* LinkedList - ဇုန်လွှာ (DL ၏)

head [ 100 ]

tail [ 600 ]

size [ 1 ]



## \* LinkedList - 끝 추가

