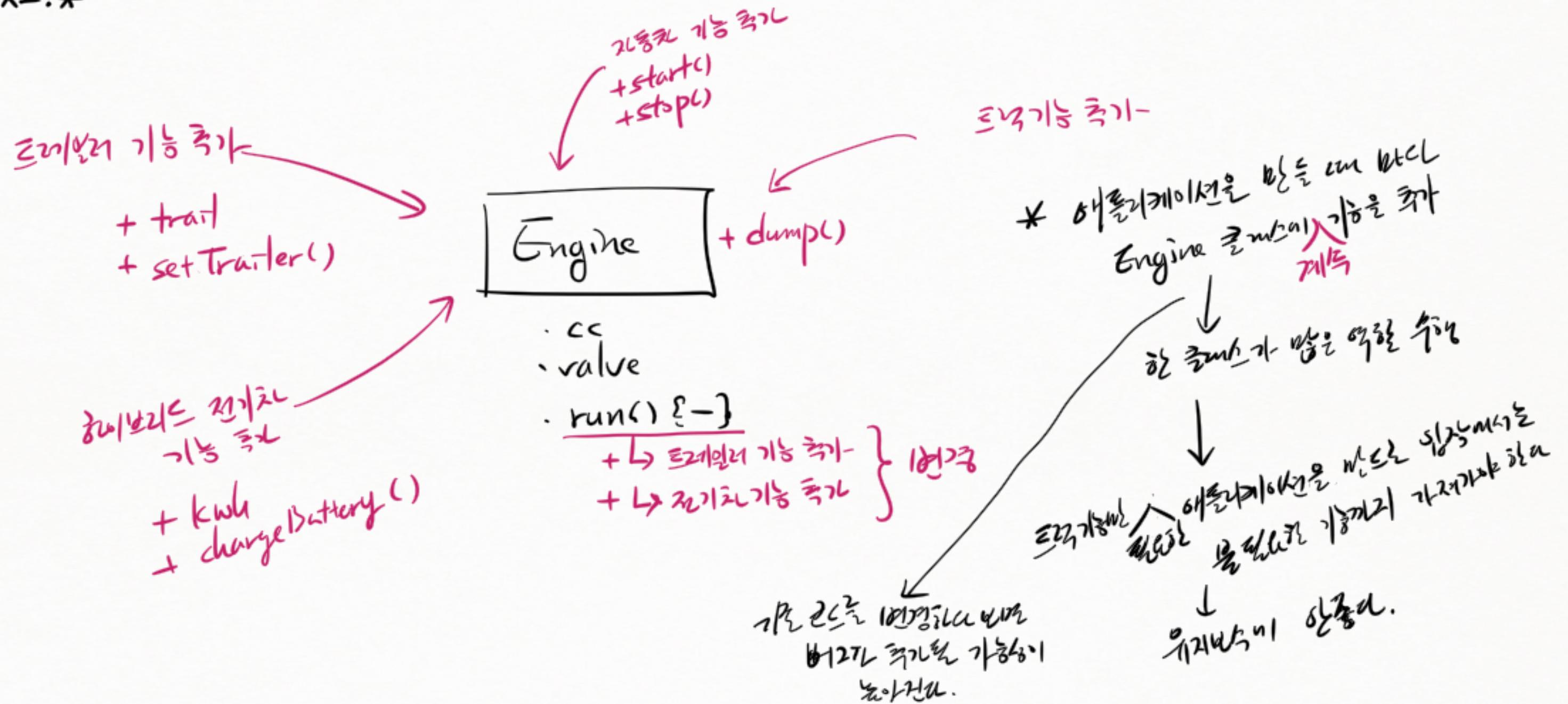


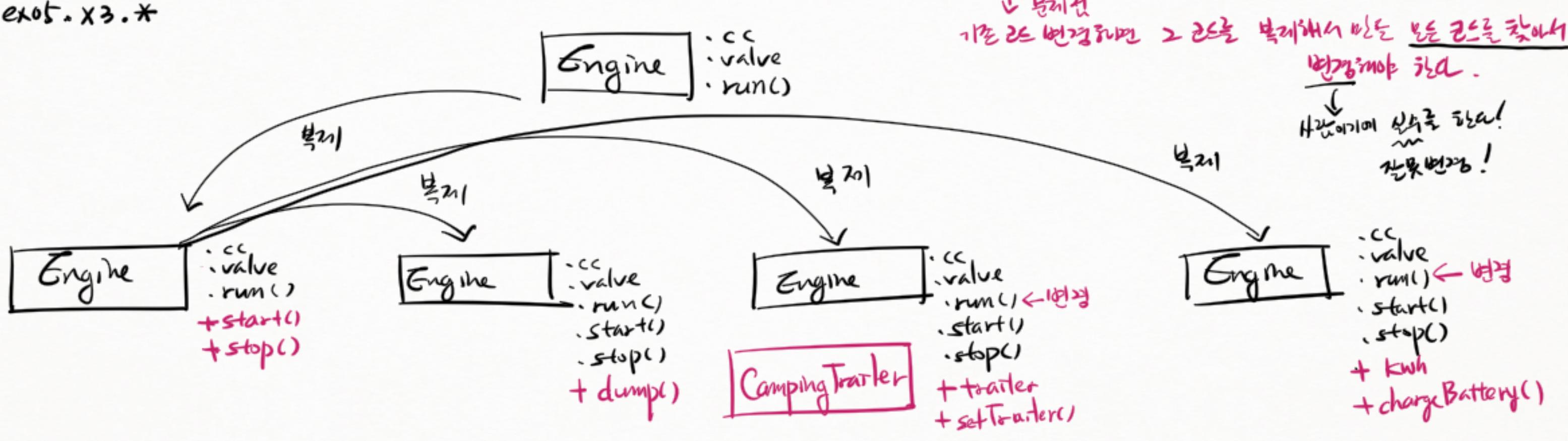
* 기능 확장 방법 1 - 기존 코드에 새 기능 추가

oop.ex5.x2.*



* 기능 확장 방법 2 - 복제한 코드에 새 기능 추가

oop.ex05.*



① 자동차 만들기
app1

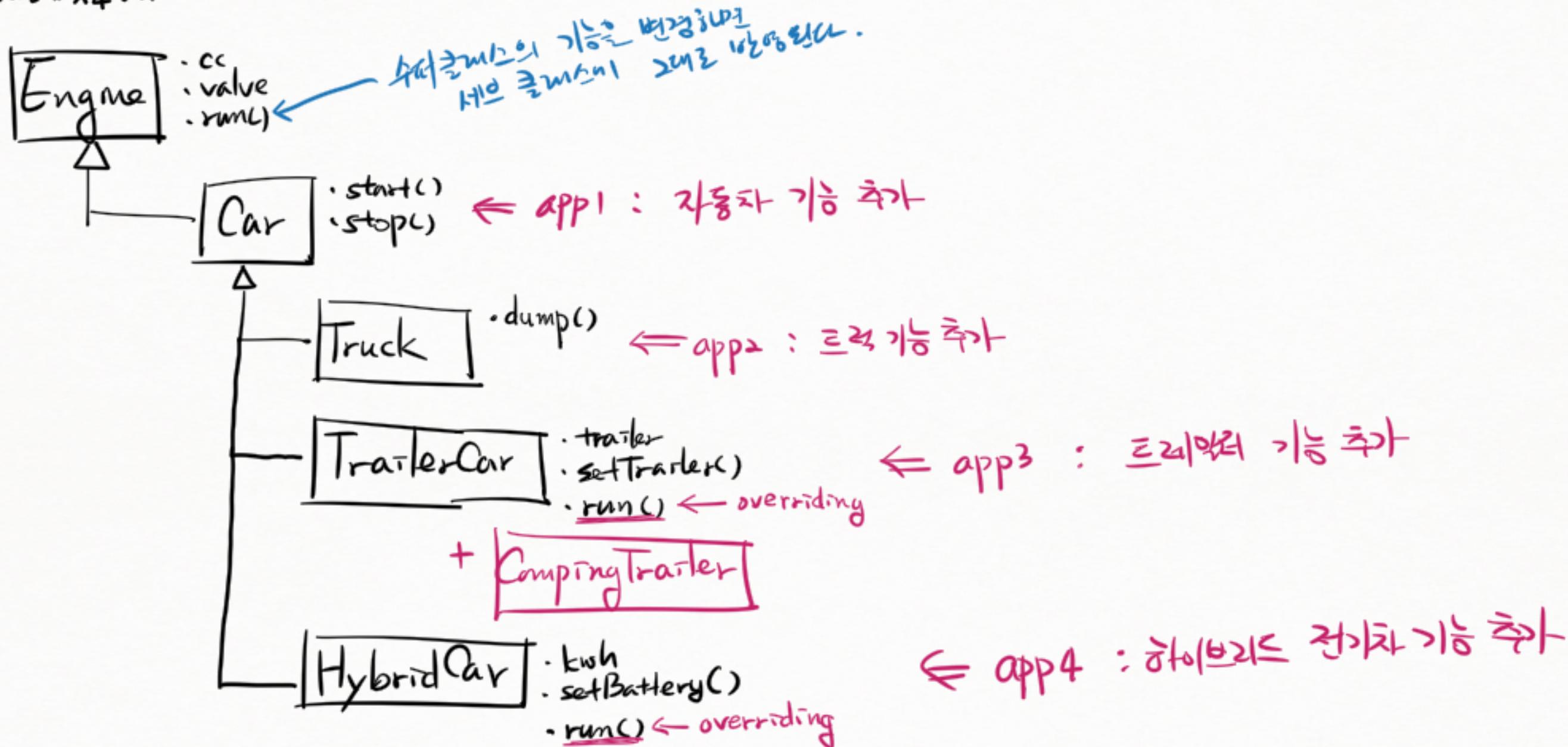
② 트레일러 만들기
app2

③ 캠핑카 만들기
app3

④ 차량으로 전기차 만들기
app4

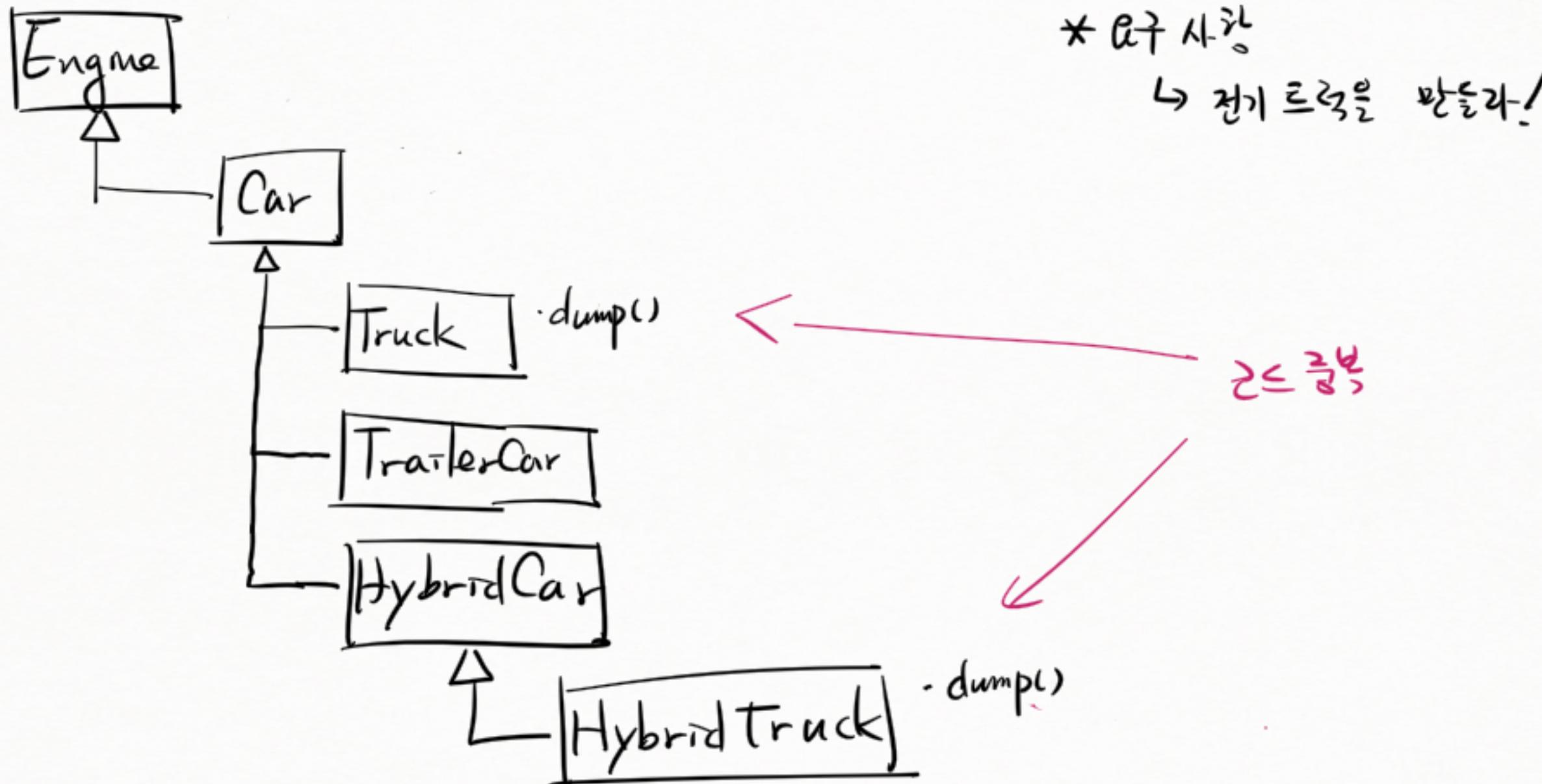
* 기능 확장 방법 3 - 상속을 이용한 확장.

oop.ex05.x4.*



* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한계

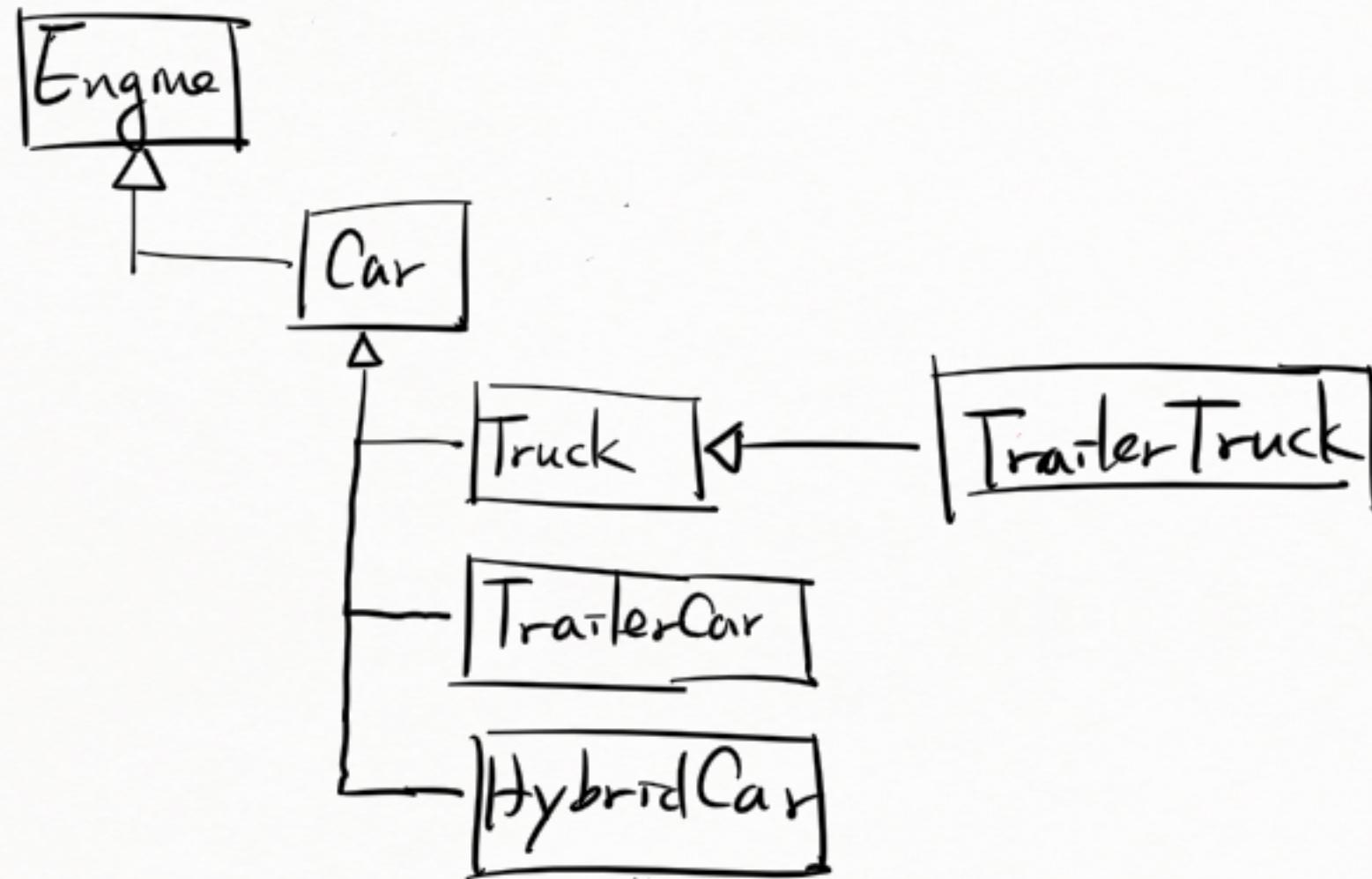
oop.ex05.x4.*



* 기능 확장
↳ 전기 트럭을 만들라!

* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한가지

oop. ex05. x4.*



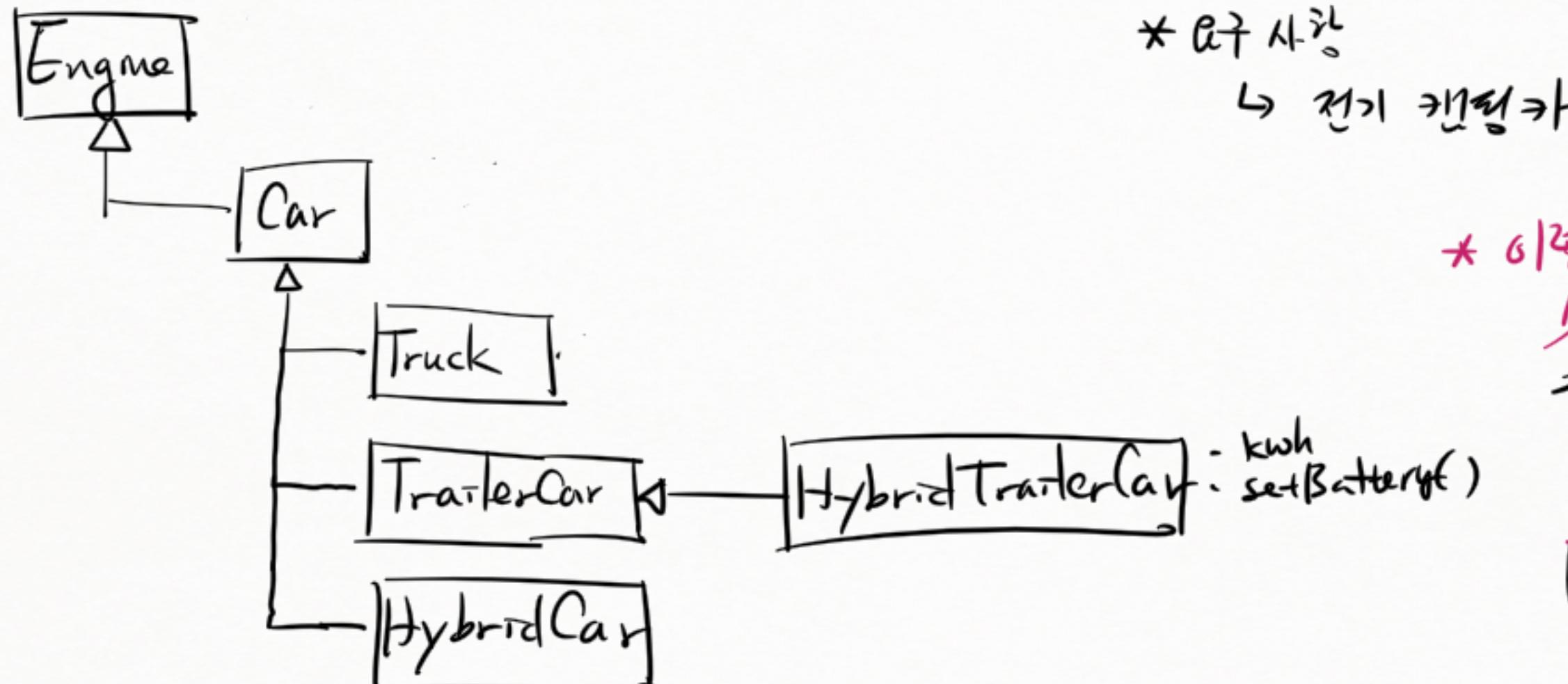
* 구조 사용

↳ 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 대상은 가능이 조합된
개체를 만들기 어렵기 때문.)

↳ 유지보수는 어렵기 때문.

* 기능학적 특성 4 : 디자인에서 기초 기능

기능적

Sedan

Truck

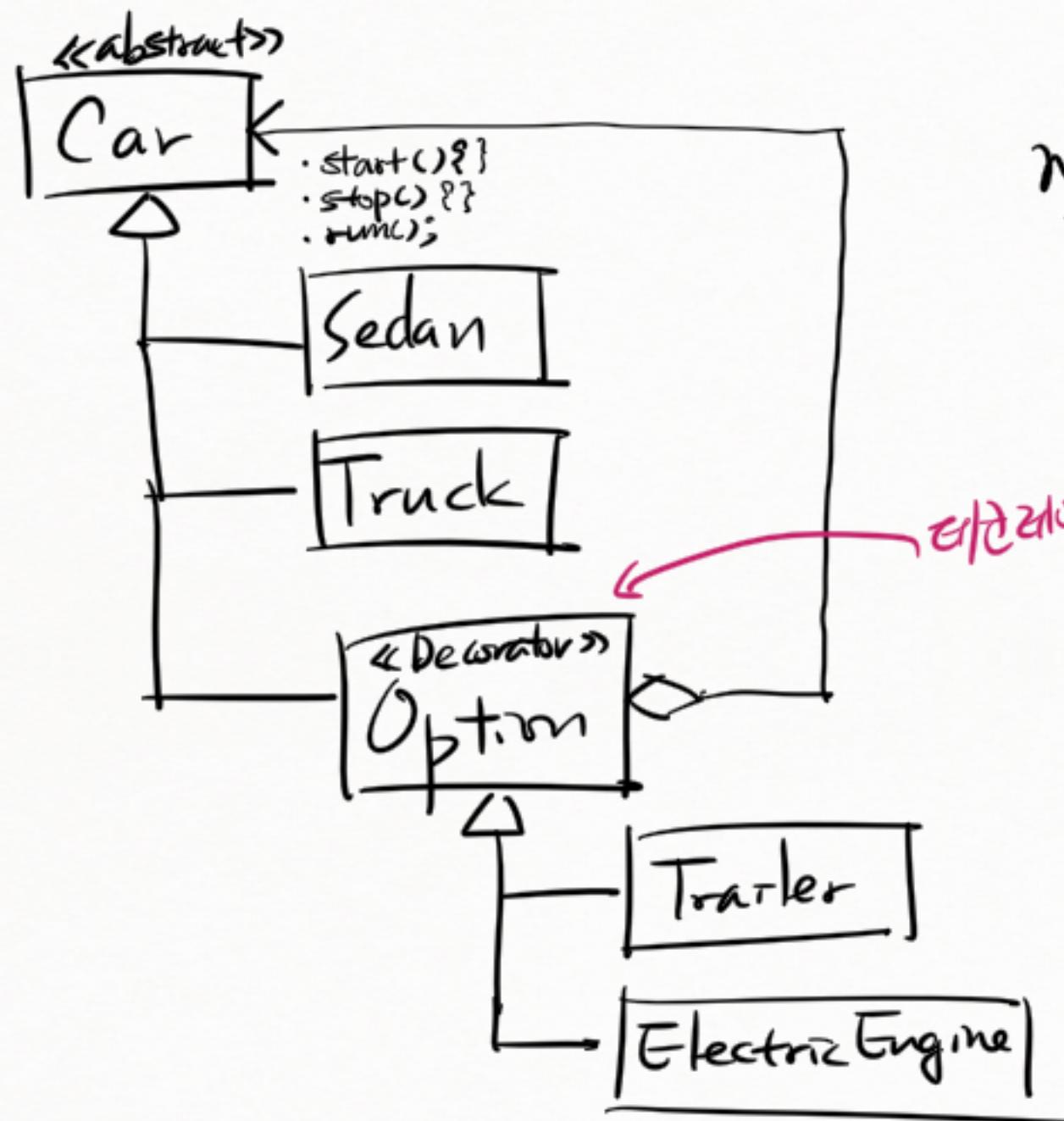
선택적

Gas Engine

Electric Engine

Trailer

* 테러리아 러닝 기법



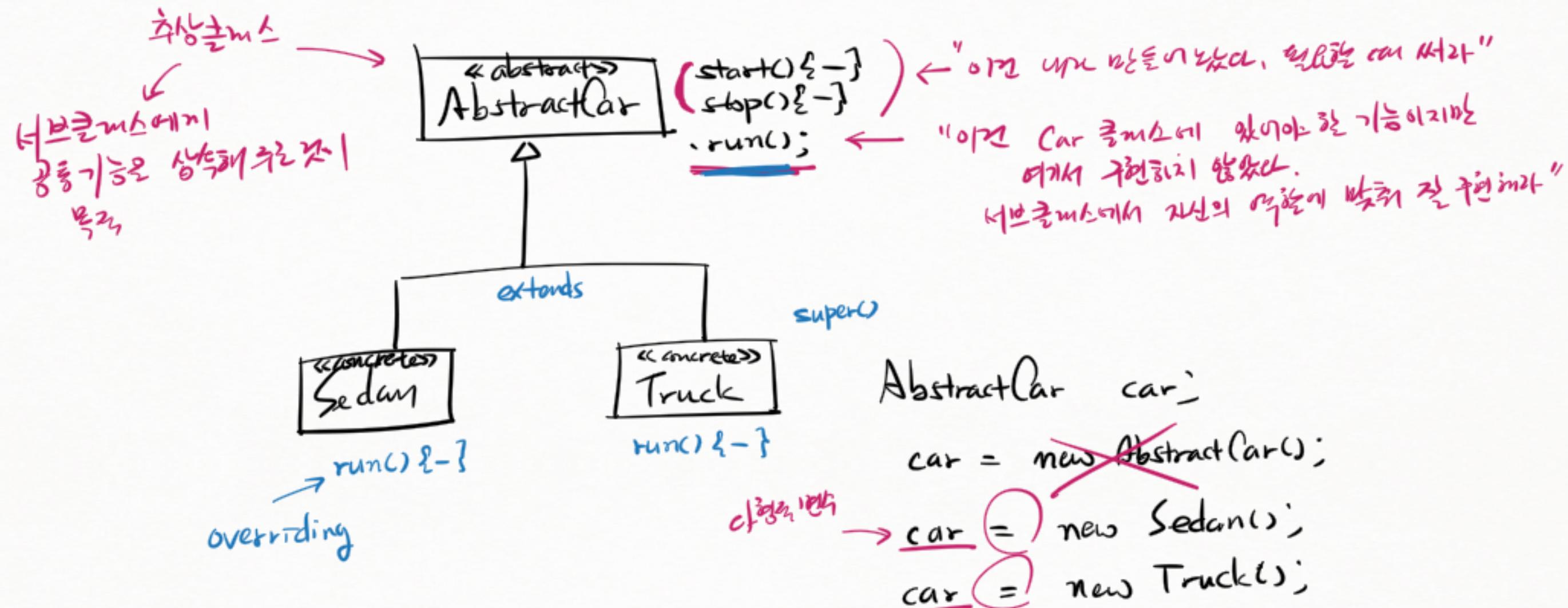
new Optim(new Sedan(new Optim(new ElectricEngine)))

new Optim(new Truck())

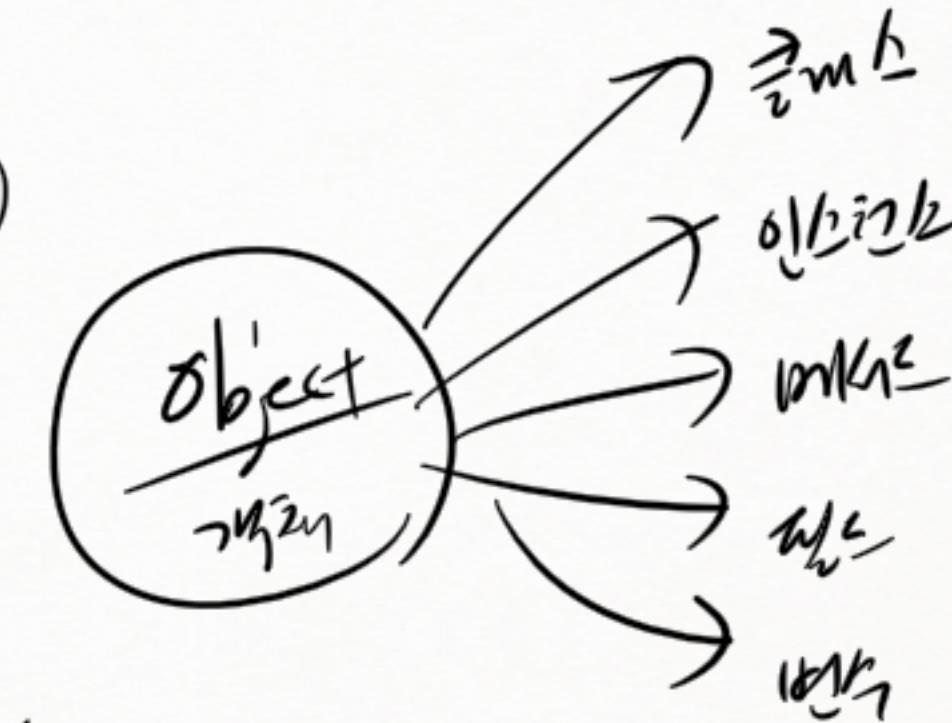
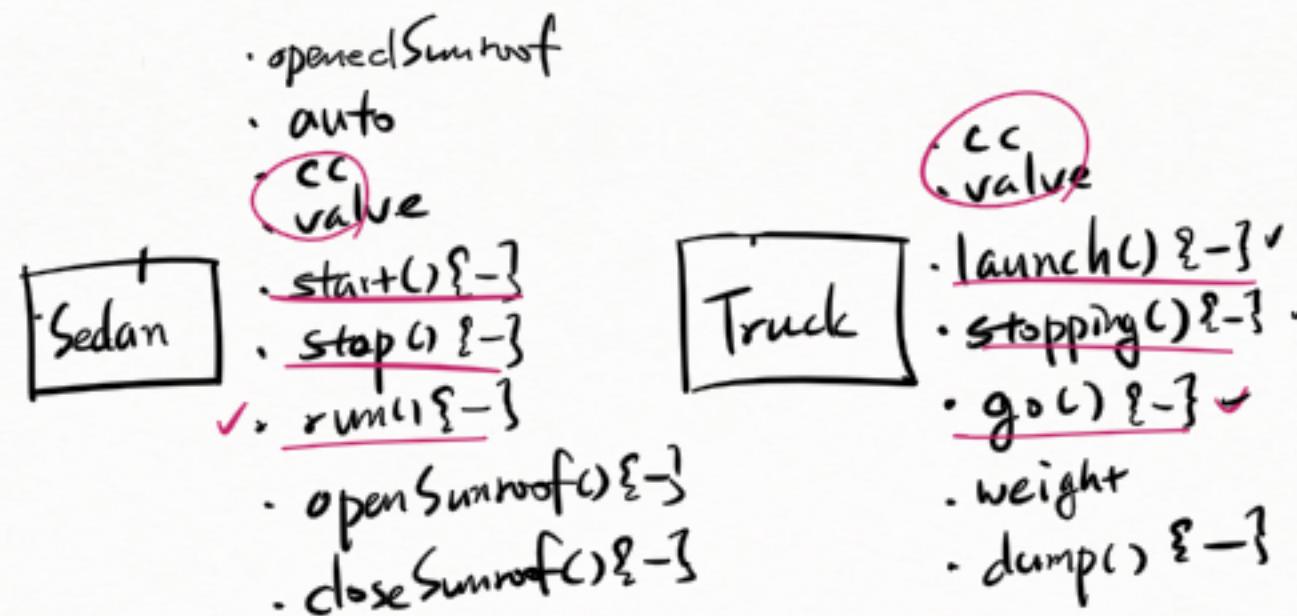
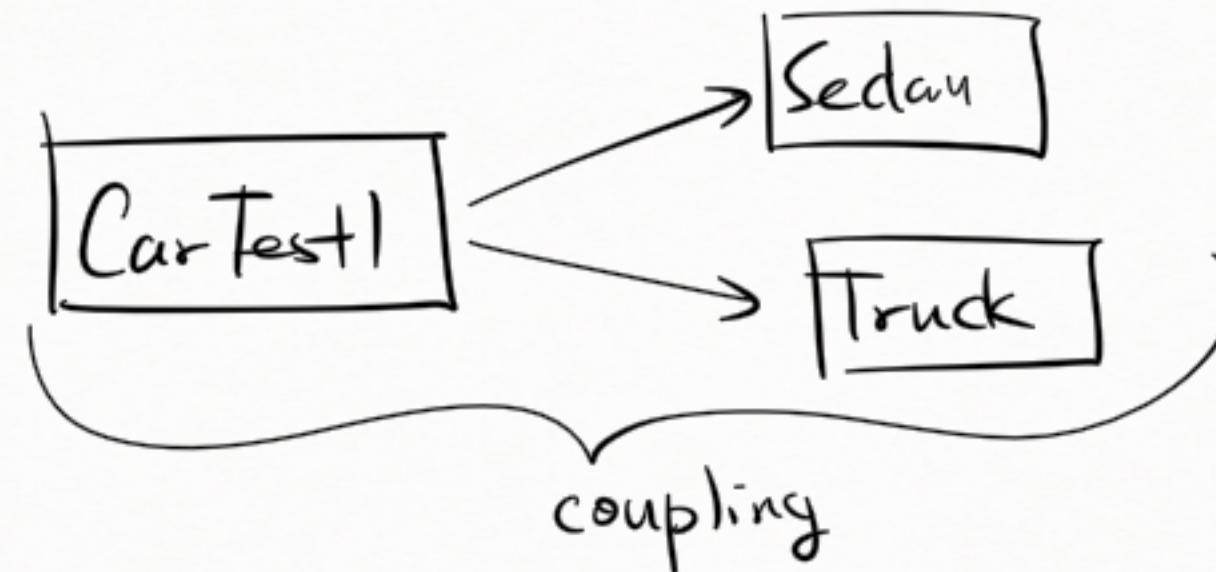
기존
클래스
추가
보다.

디코레이터: 주제에
별도로 선택 가능

* 추상클래스와 구현

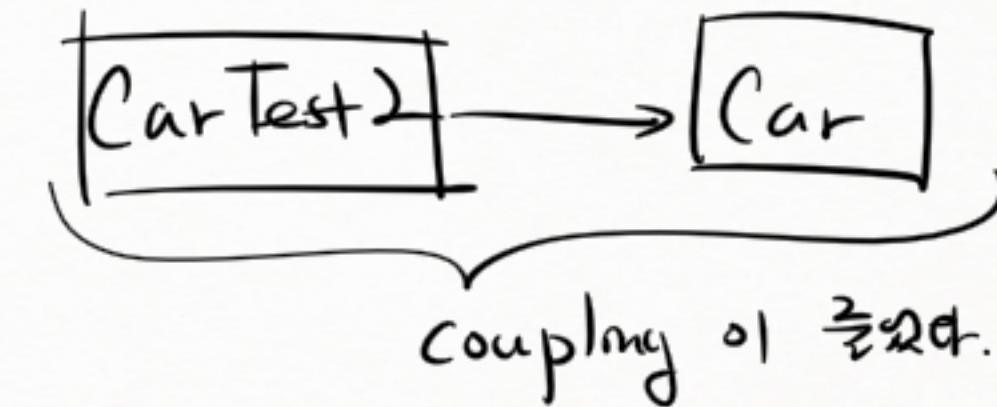
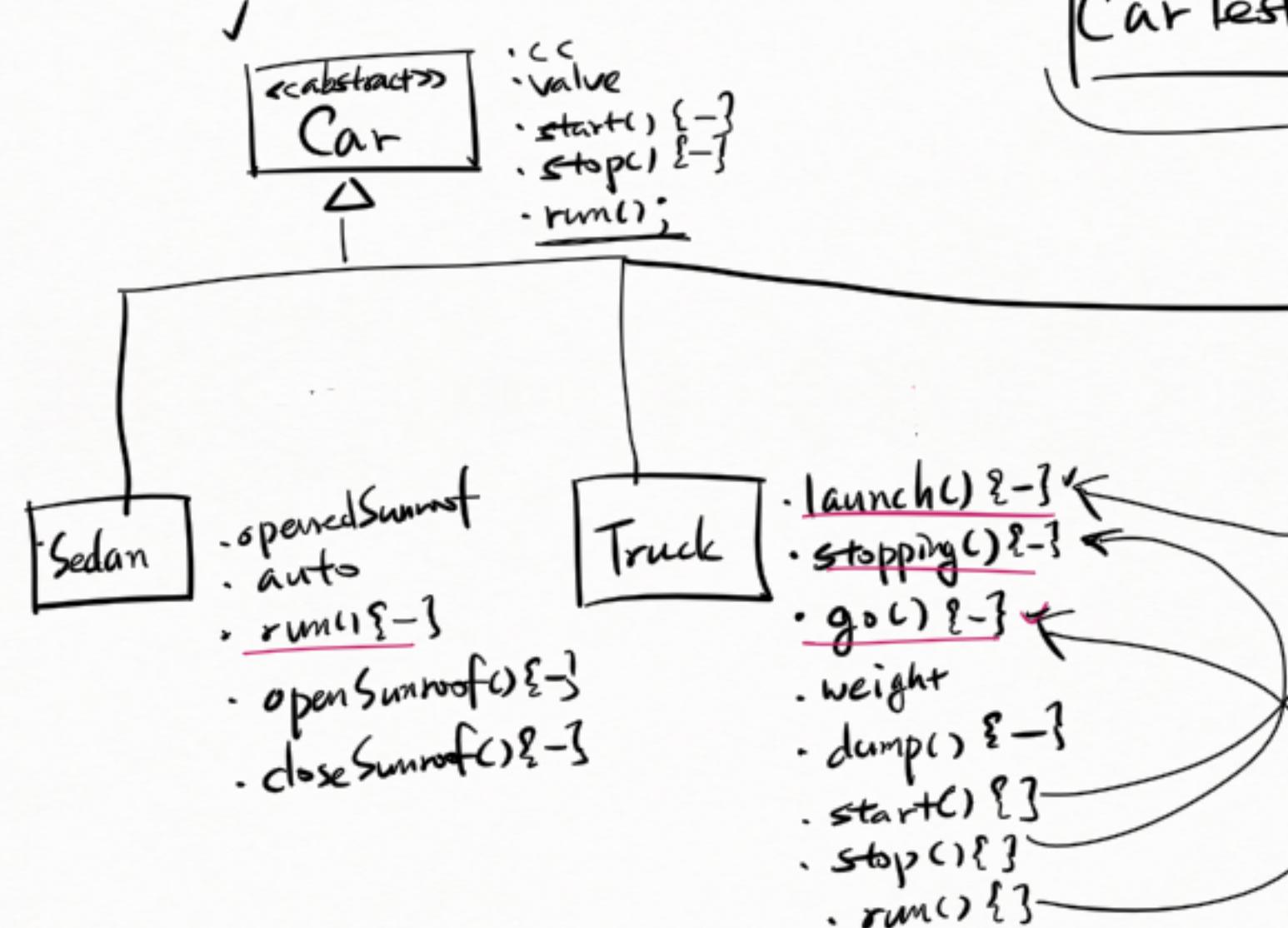


* 실전 프로그래밍 1 단계



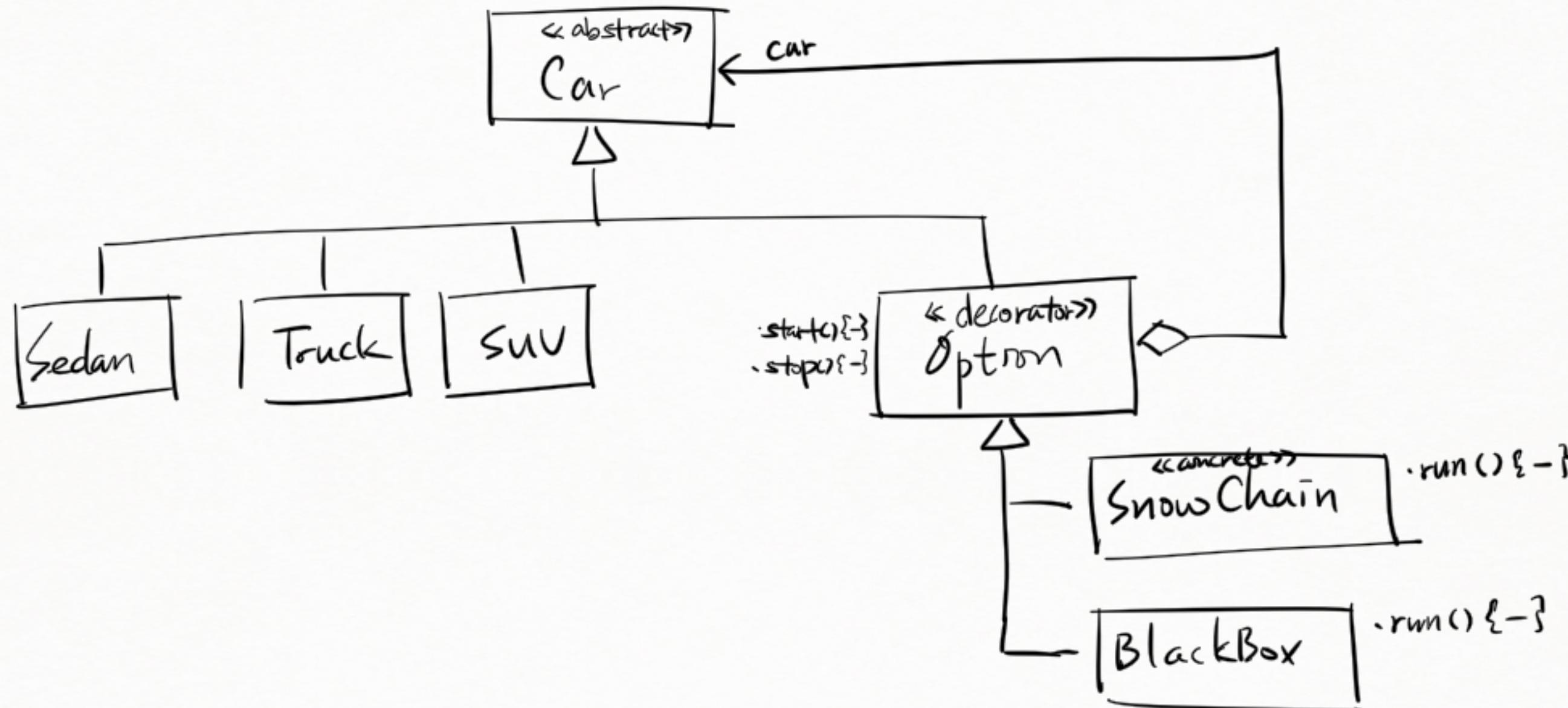
* 상 하 한 한 한 한 한 - 품종, 품질, 성능

\hookrightarrow : generalization

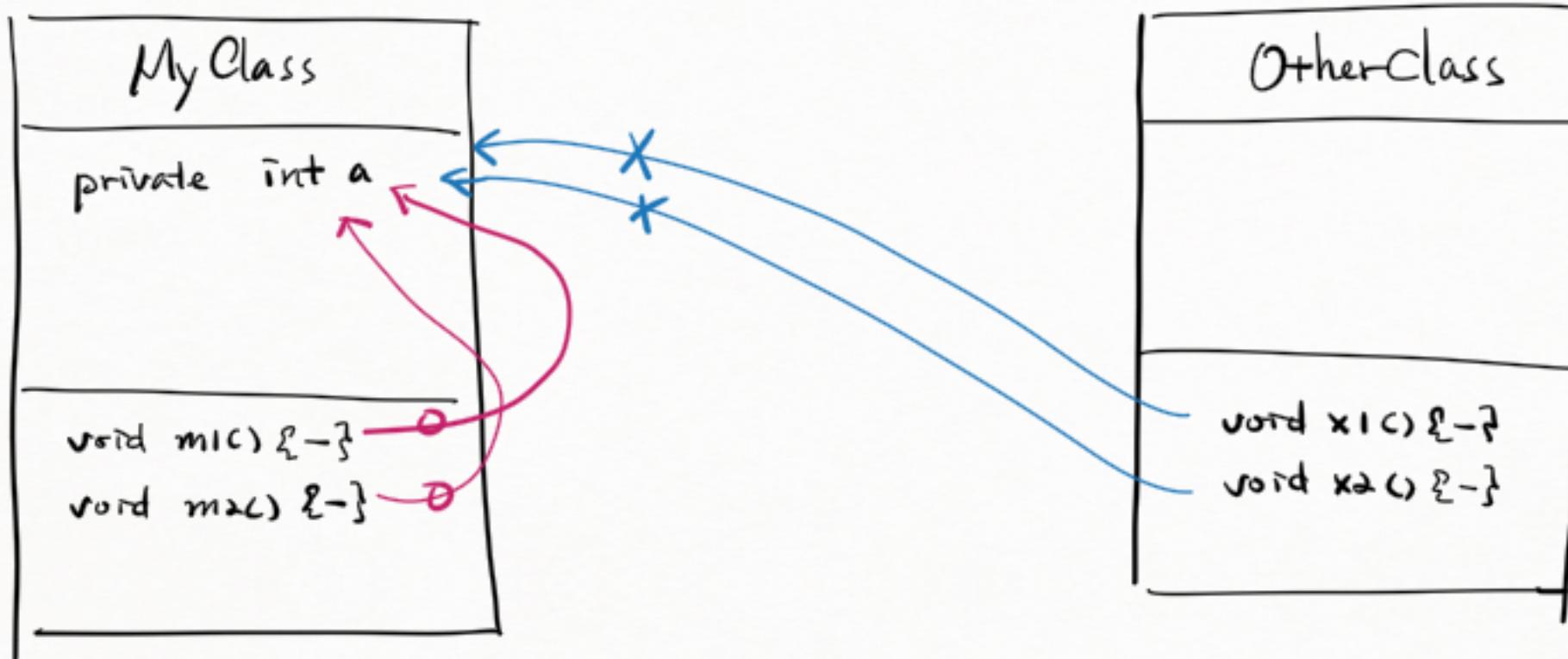


- enableLwd: boolean
- activateLwd(boolean)
- run() {-}

* 차선을 3개로 나누면 차량이 1개로 통합

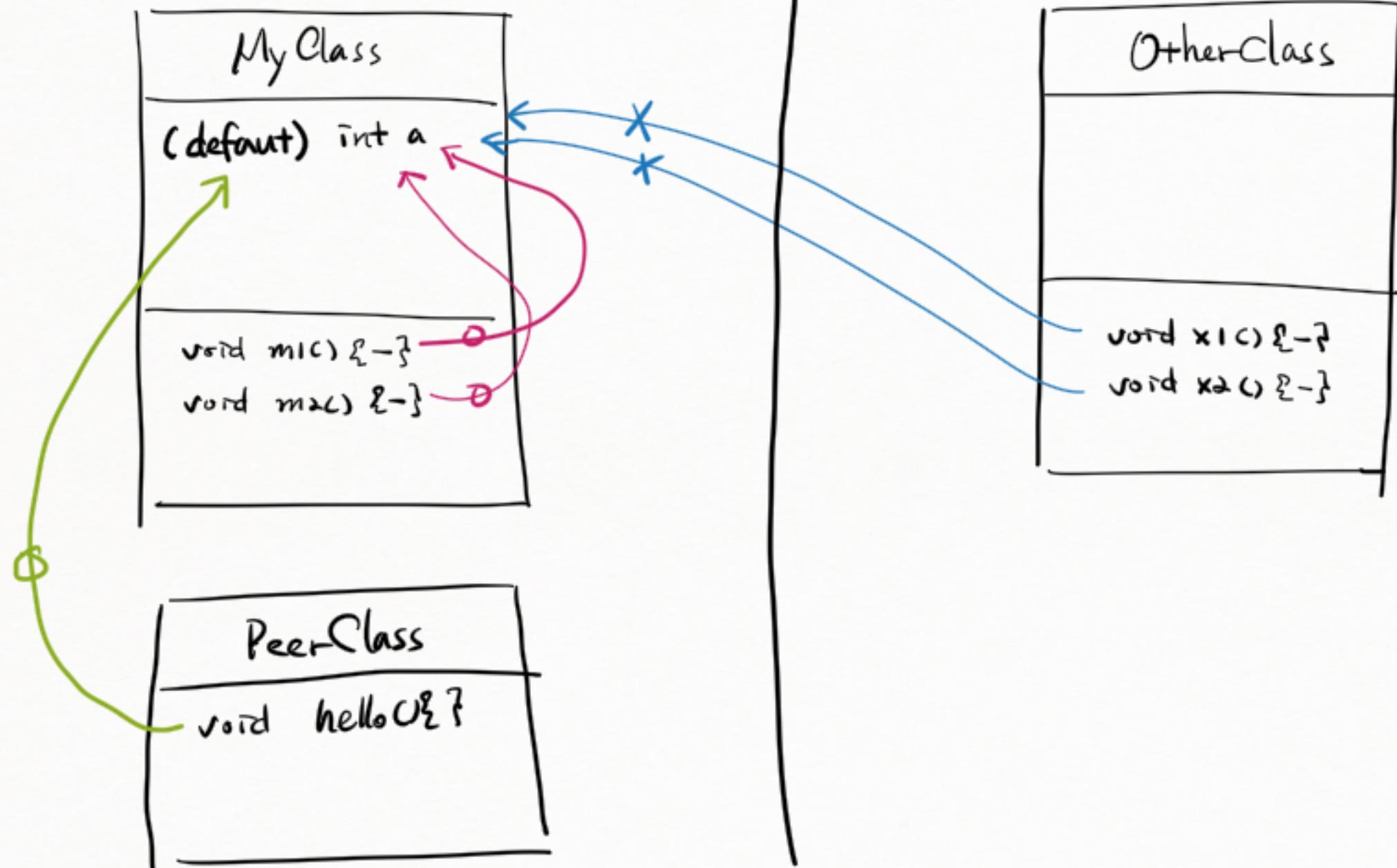


* private - 같은 클래스의 멤버만 접근 가능



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

com.eomcs.test1



com.eomcs.test2

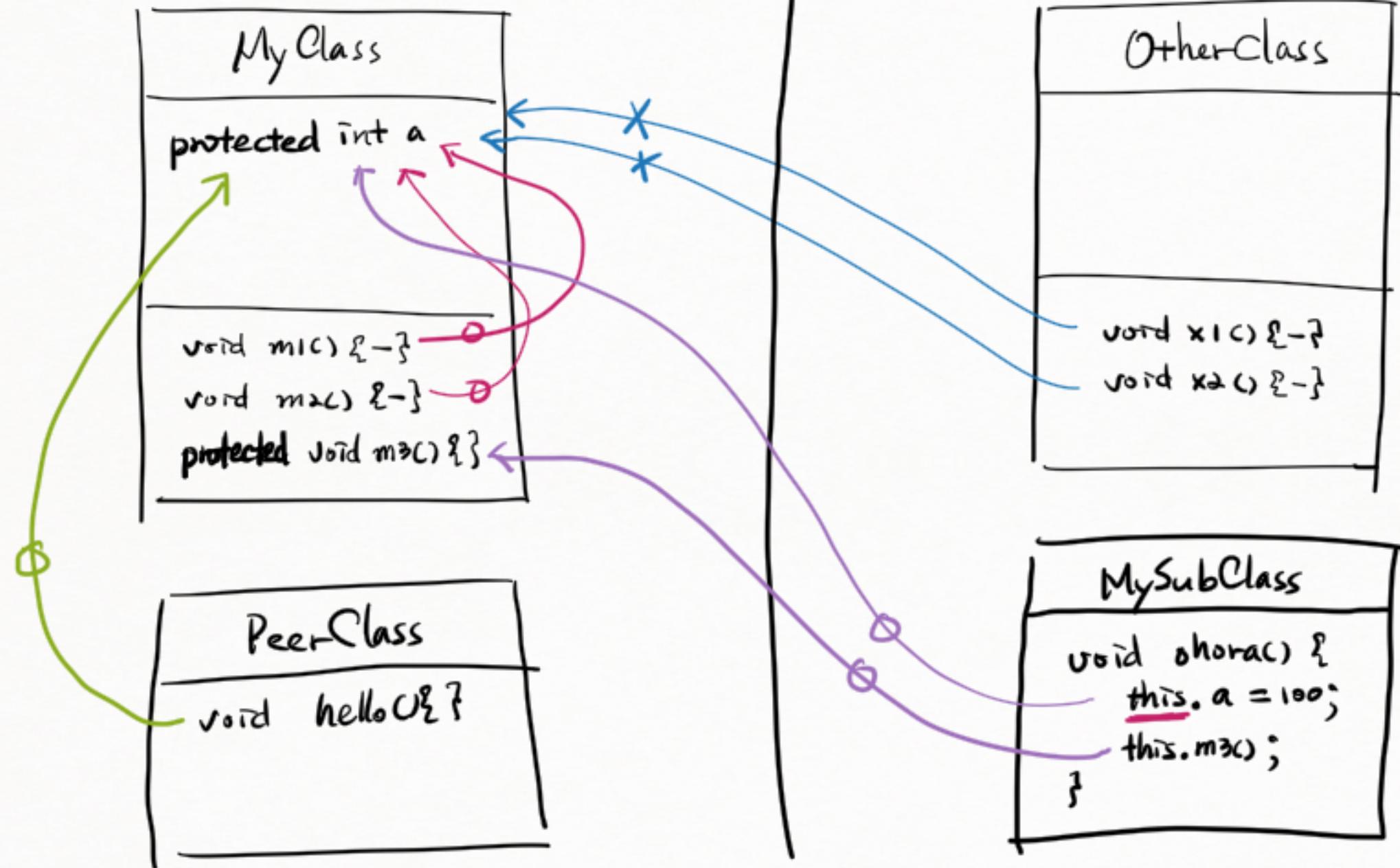
* protected - 같은 패키지 내의 멤버 접근 가능

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

com.eomcs.test1

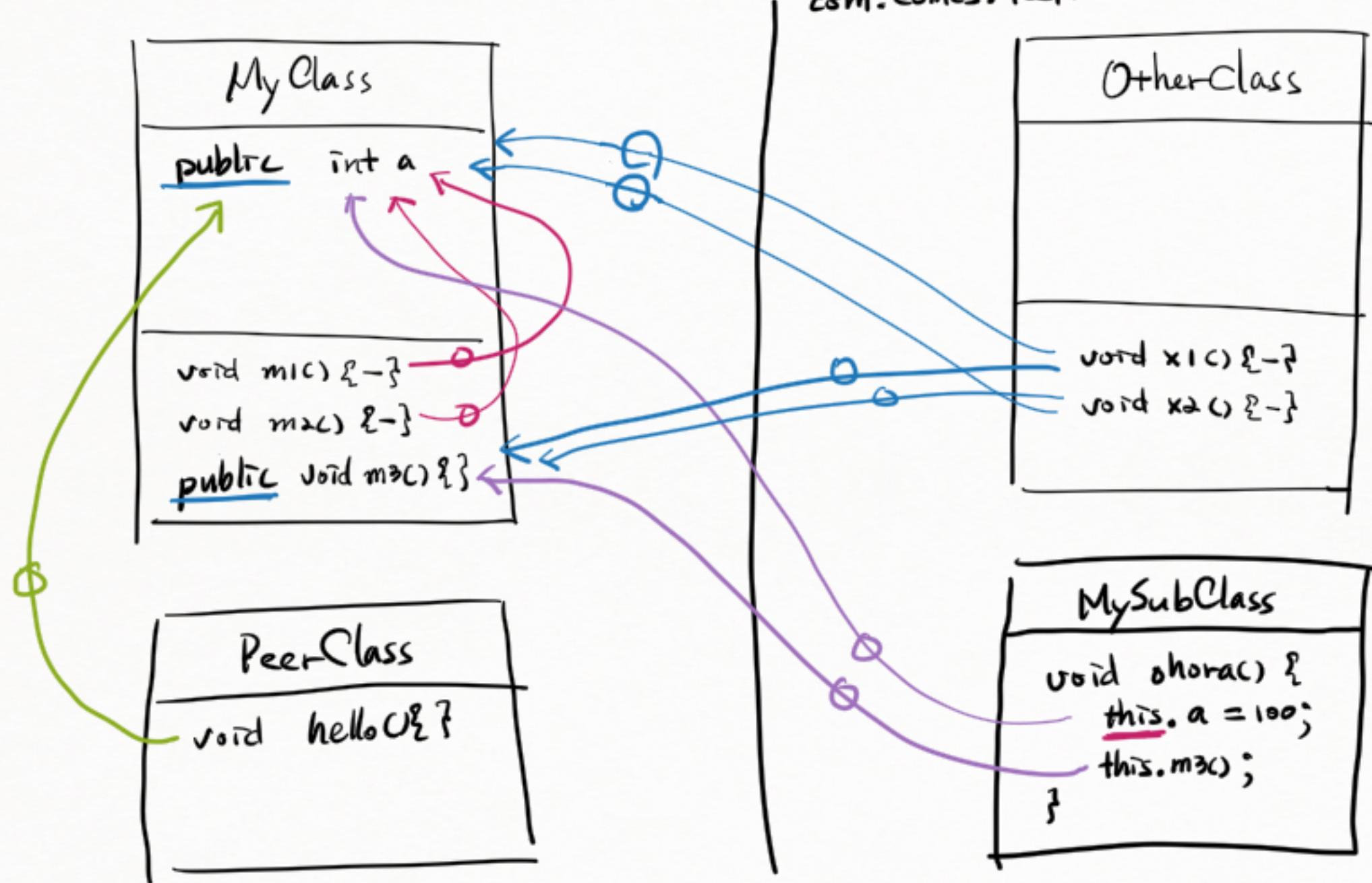
com.eomcs.test2

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

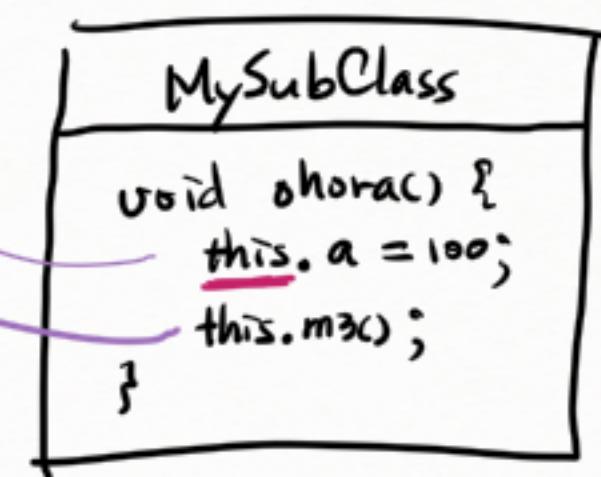
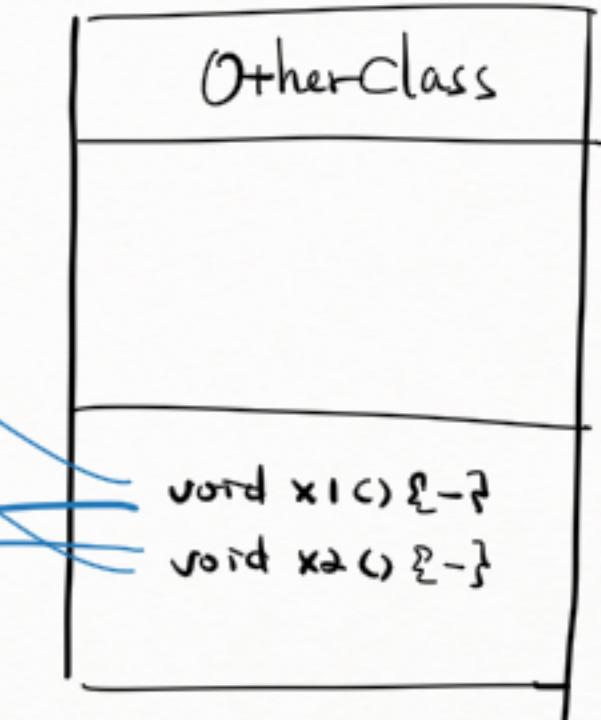


* public - 모든 멤버 접근 가능

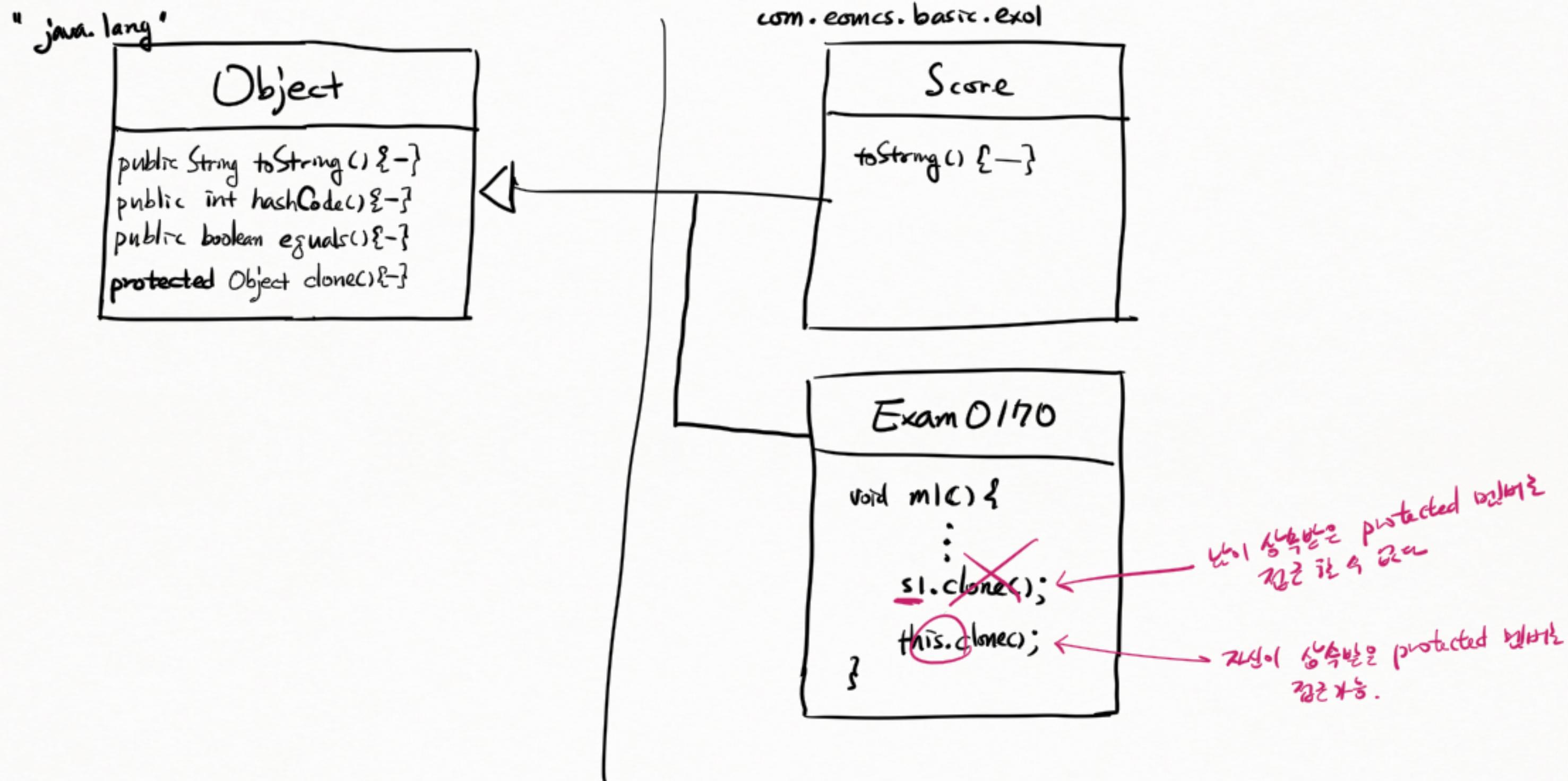
com.eomcs.test1



com.eomcs.test2

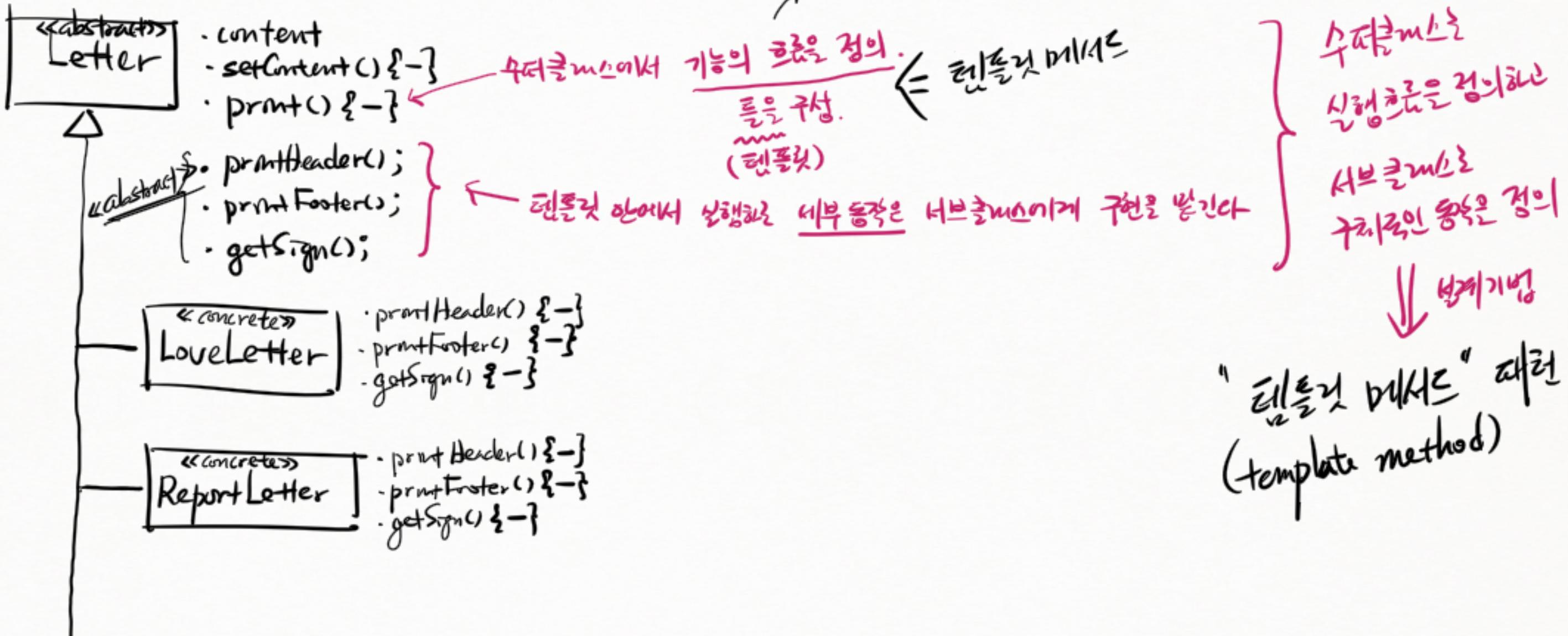


* clone 데일



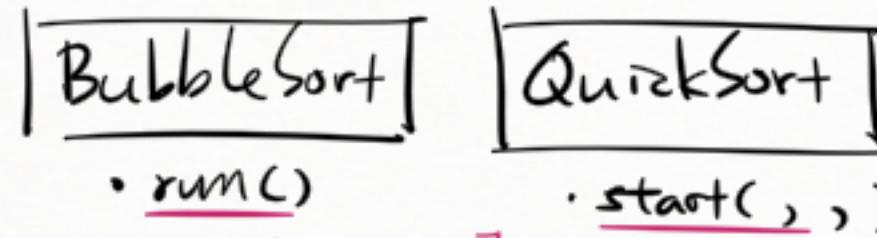
* 추상클래스

the skeleton of an algorithm



* 추상 클래스와 추상 메서드의 활용

① 각각 단일의 클래스 사용

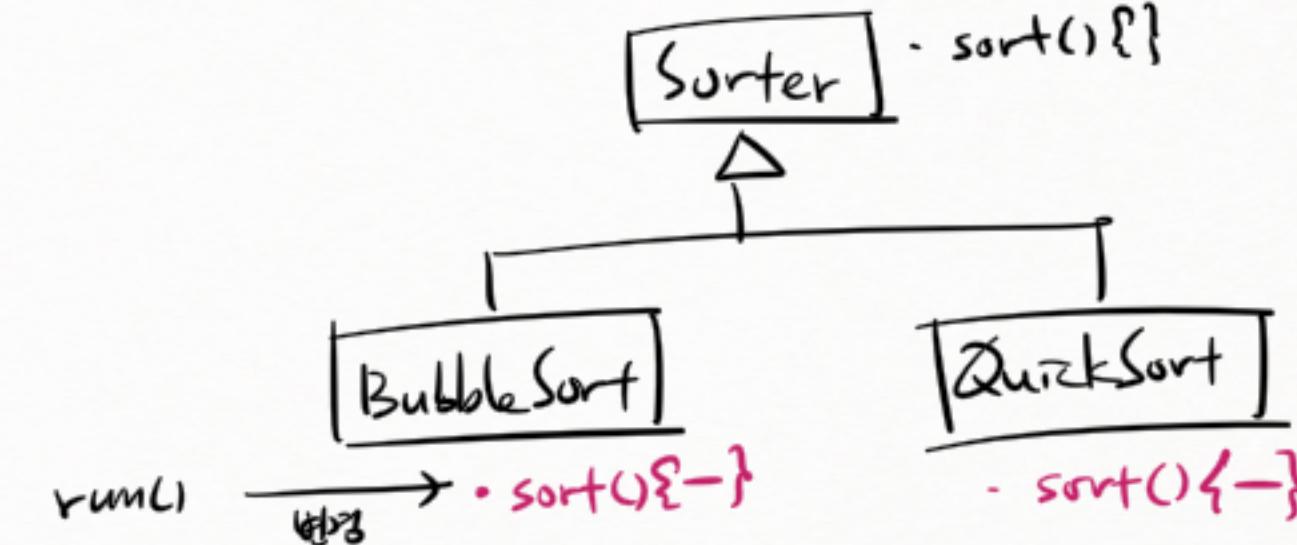


• run()
 ↗
 두 개의 정렬 클래스를 같은 부류가 아니기 때문에
 같은 메소드인 print() 메서드를 사용할 수 없다.
 다른 이름으로 사용.

- display(BubbleSort, int[]){-}
- display(QuickSort, int[]){-}

개선

② 같은 단일의 클래스로 보기

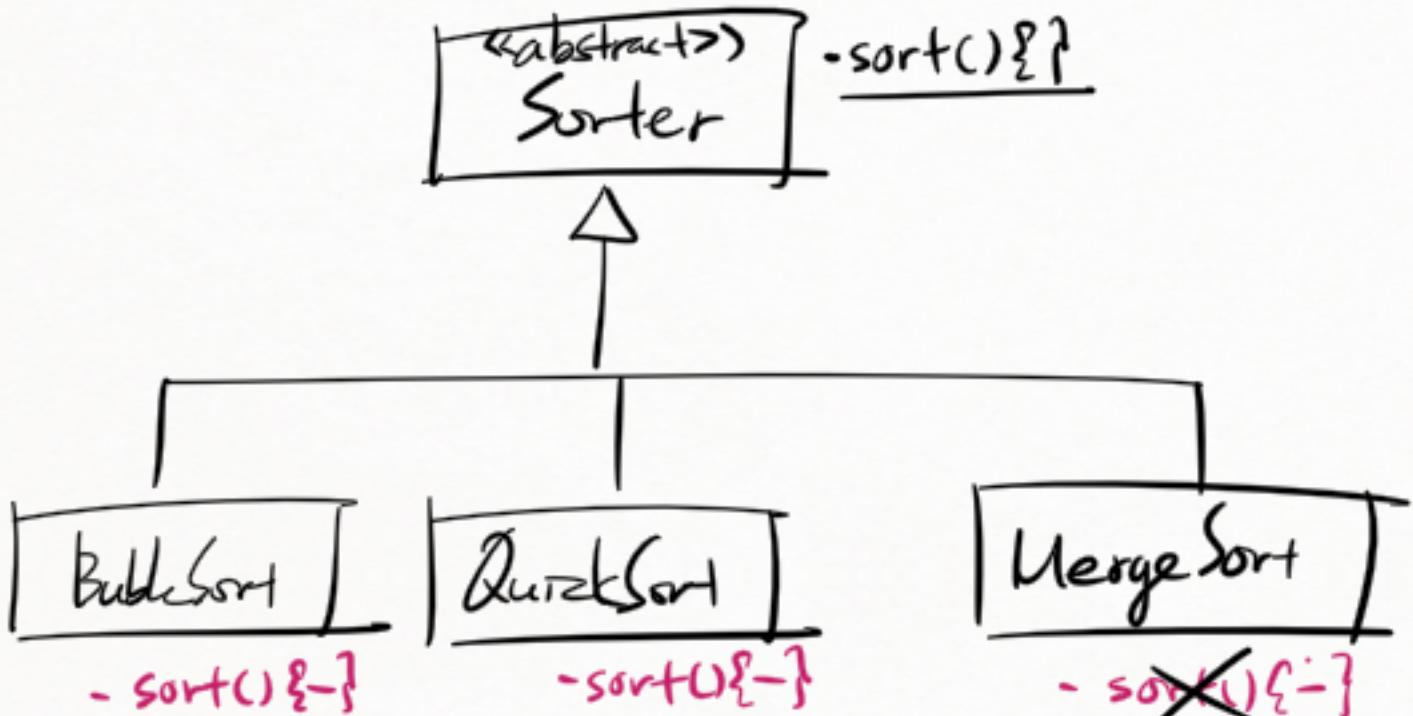


run()
 ↗
 같은 단일화된
 print()를 두 클래스를 위한 대체

• display(Surter, int[]){
 s.sort(); ←
 }
 ↗
 클래스에 상관없이
 같은 이름의 메서드 호출
 //
 일관성 확보 가능

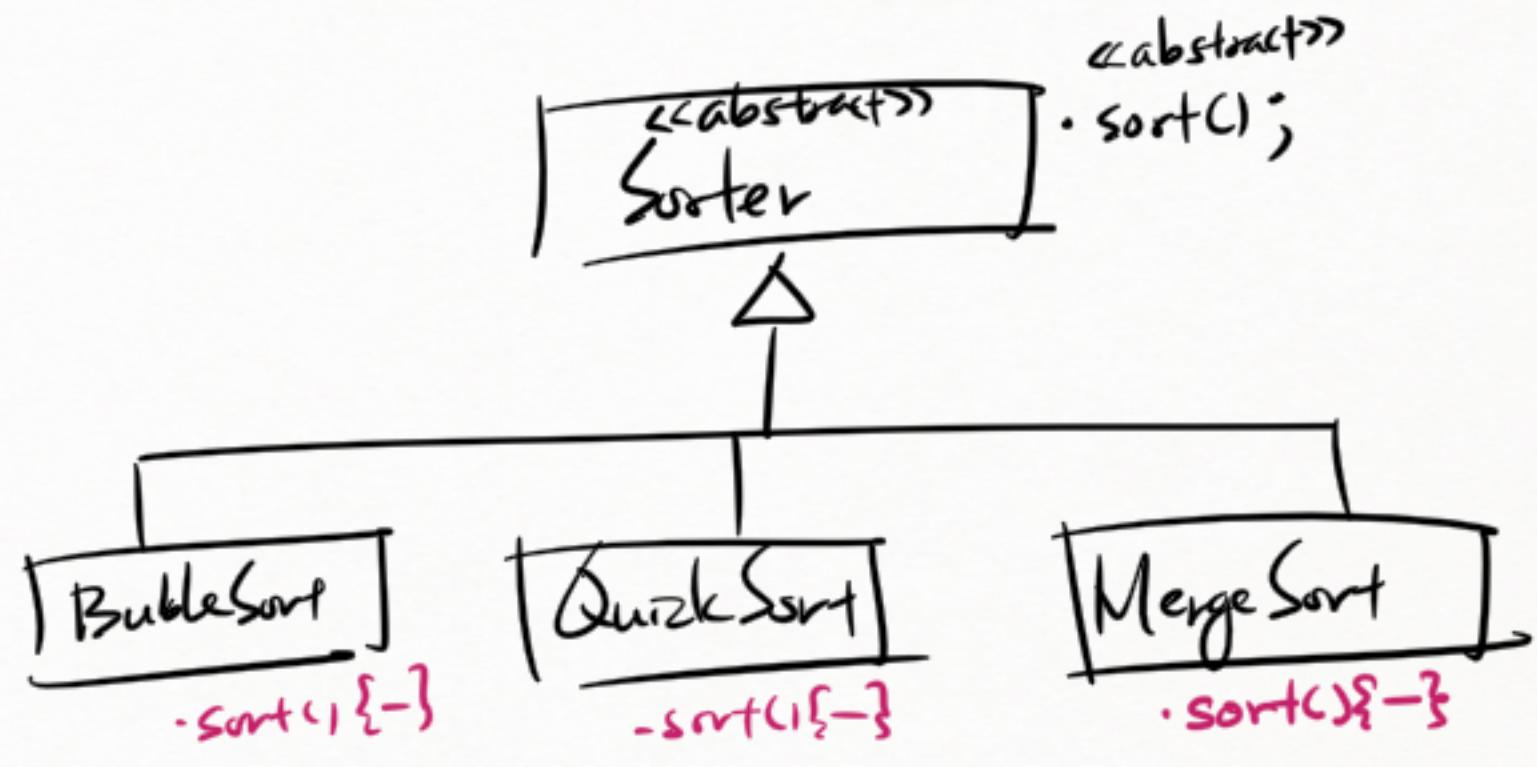
* 추상클래스와 추상메서드의 활용

③ 추상클래스로 추상클래스로 나누기



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

④ 추상메서드로 분리하는법

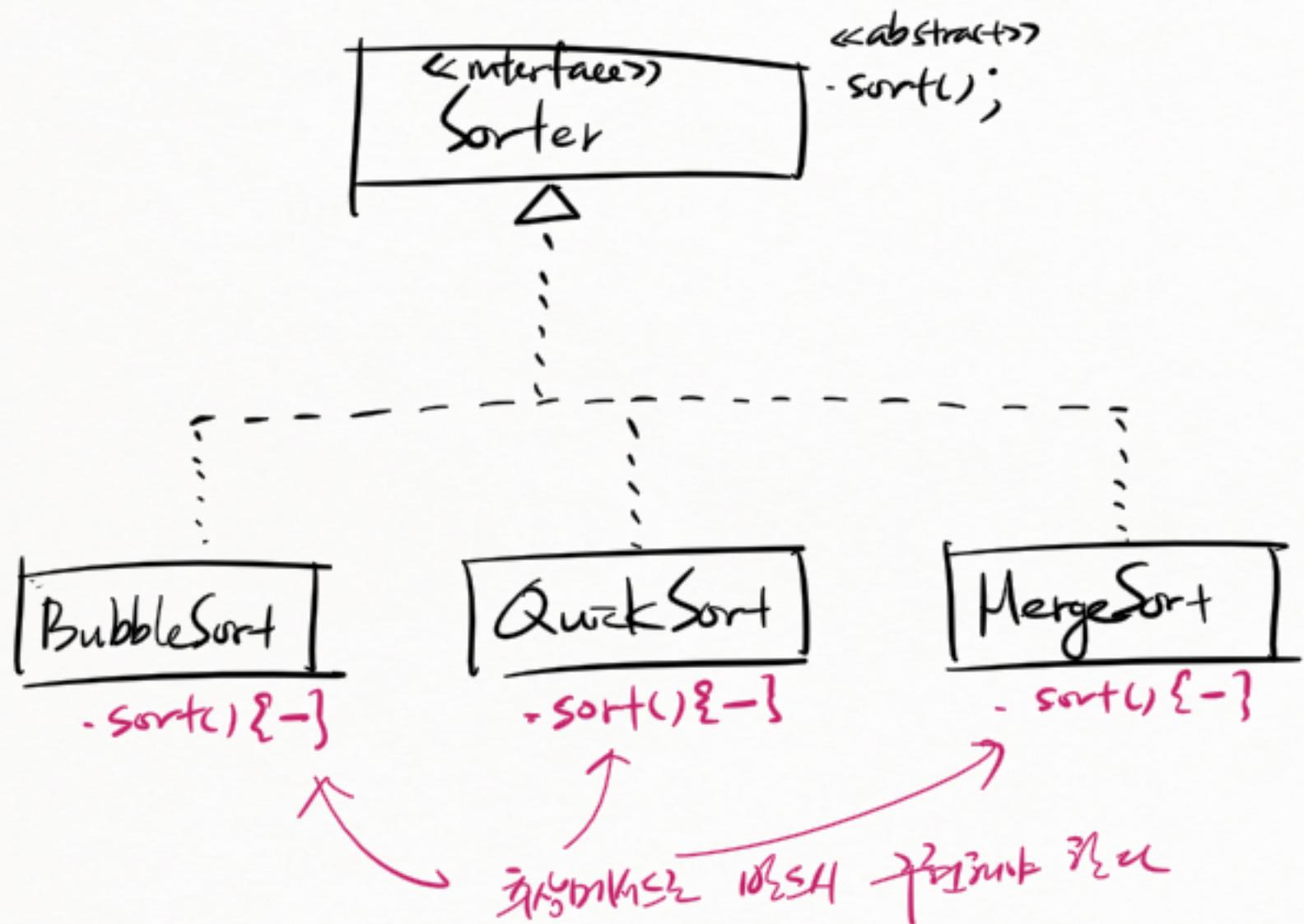


이 클래스를 sort()로
구현해야 할 경우 가능하다
Abstract의 sort()는
아직 정의되어 있지 않음
구현해야 할 수가 아니라!

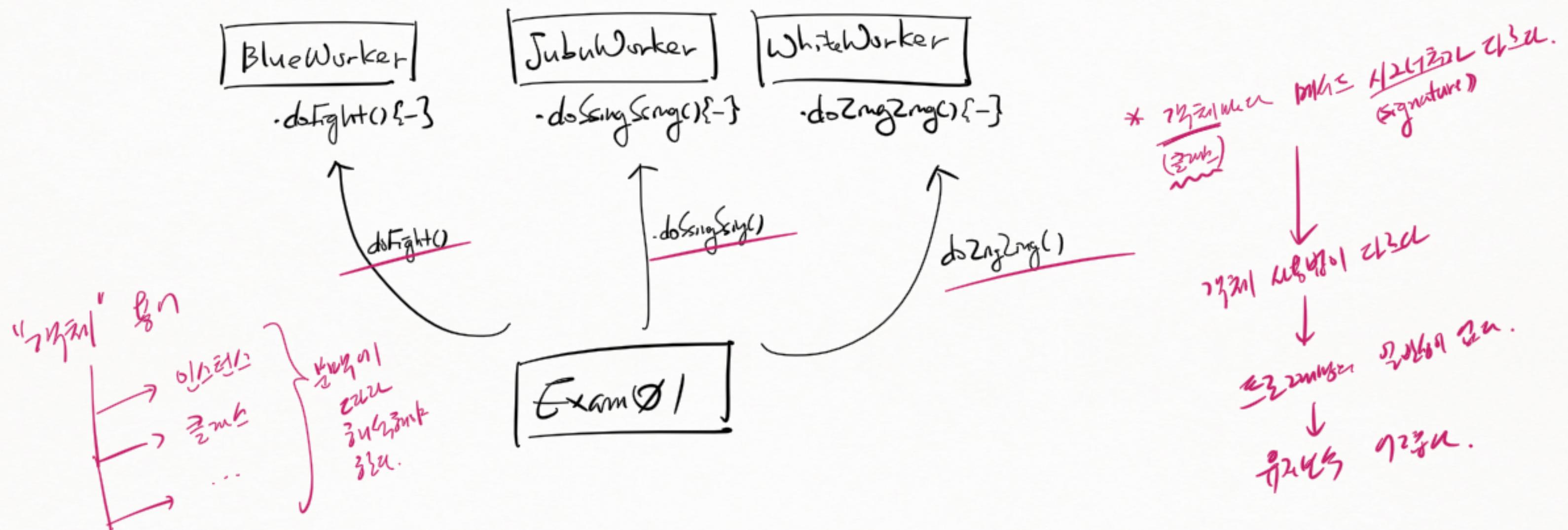
抽象 추상 메서드는 implements interface!

* 추상클래스 만든 인터페이스를 끌 때

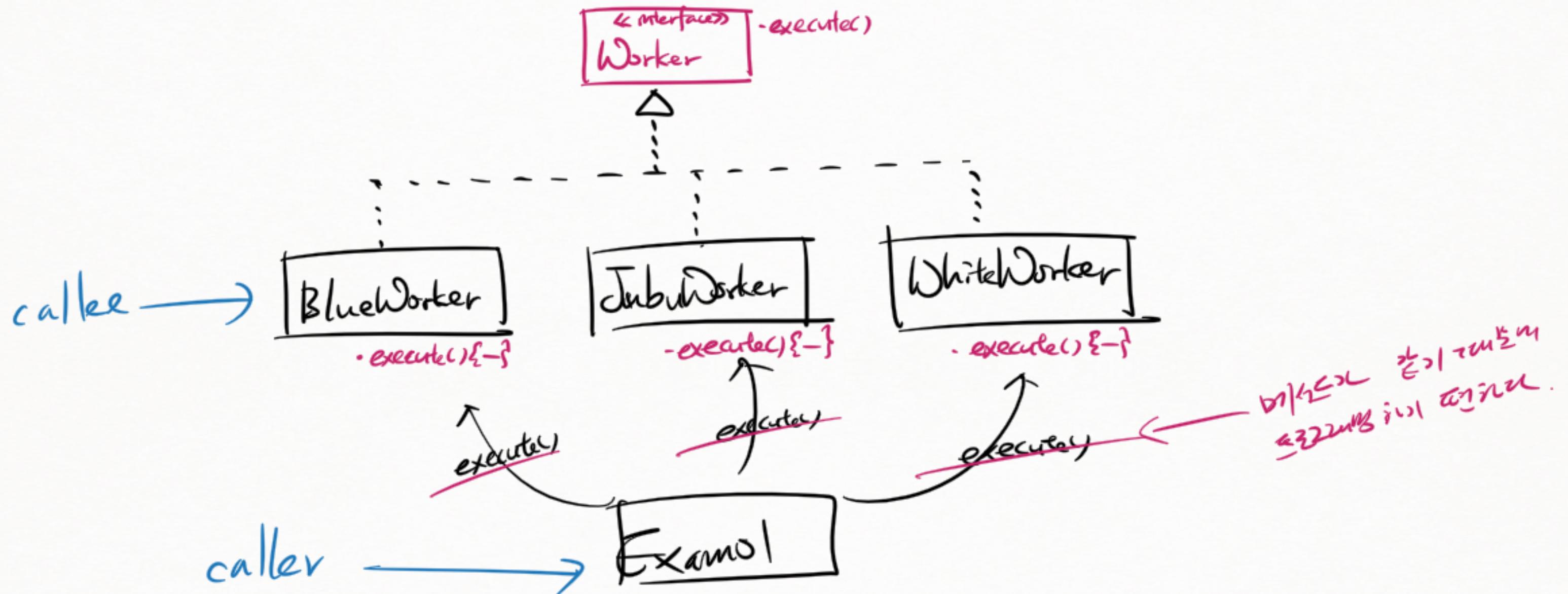
⑤ 추상클래스 만든 인터페이스로 만들기



* 인터페이스 사용 : - oop. ex9. al. before



* 인터페이스 사용 ③ : - oop. ex09. al. after

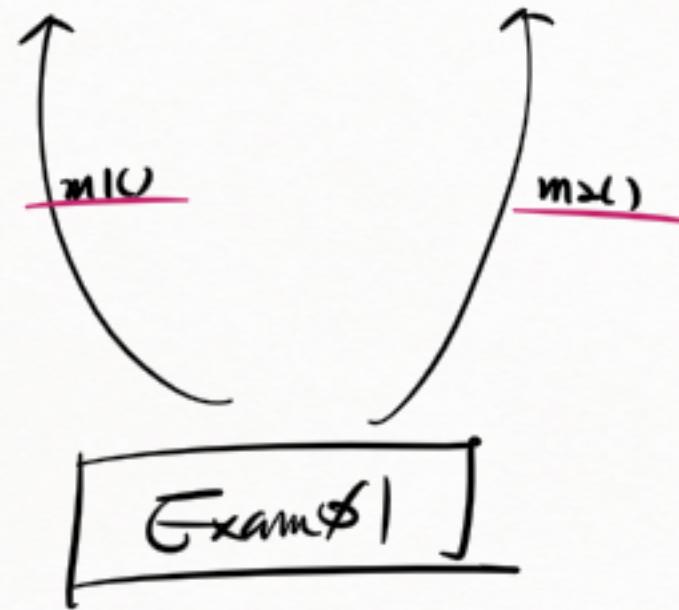
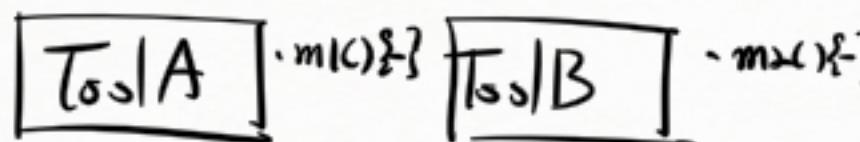


* 인터페이스 사용 전 / 후

① 사용전

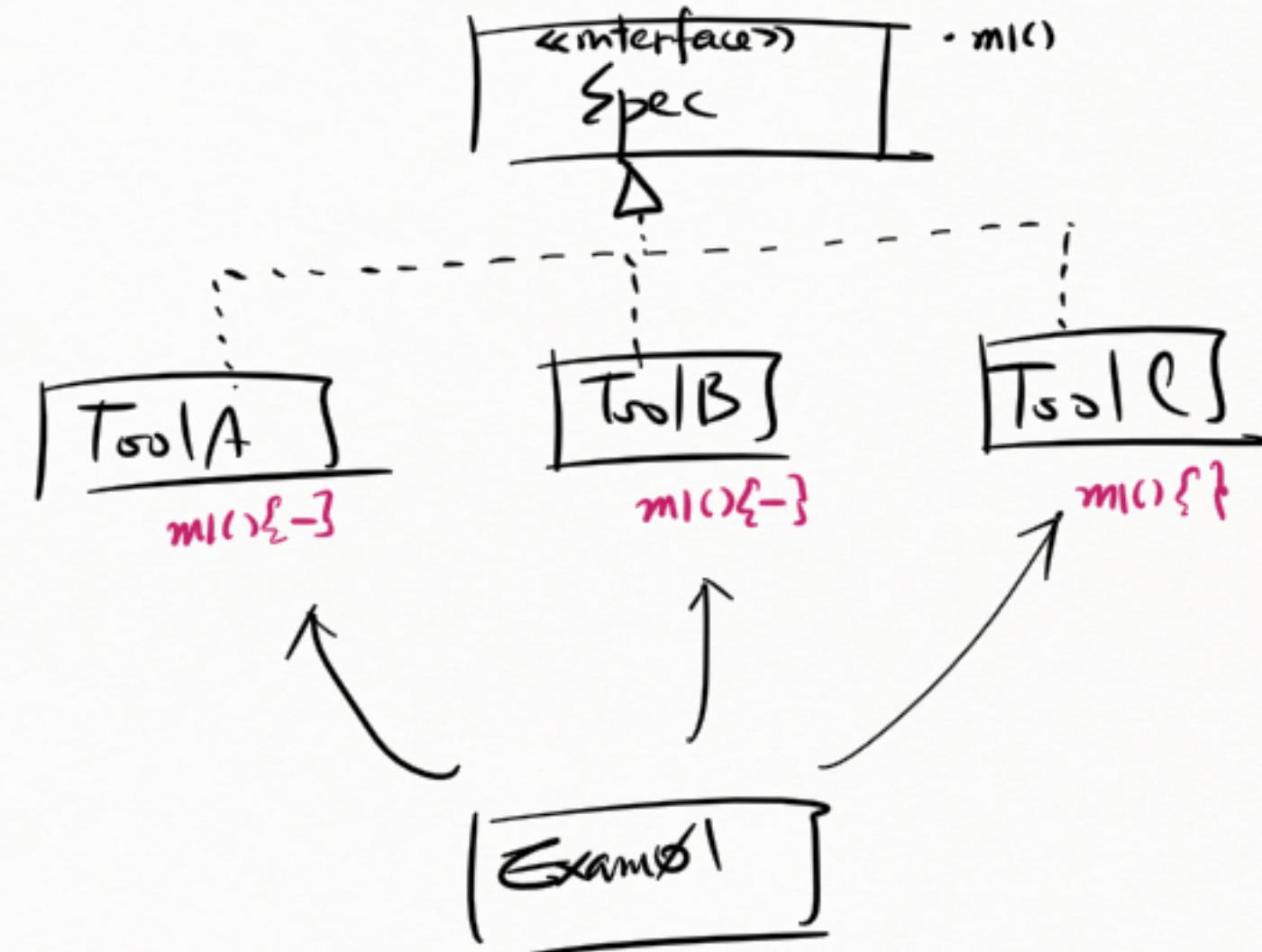
개선

② 사용 후

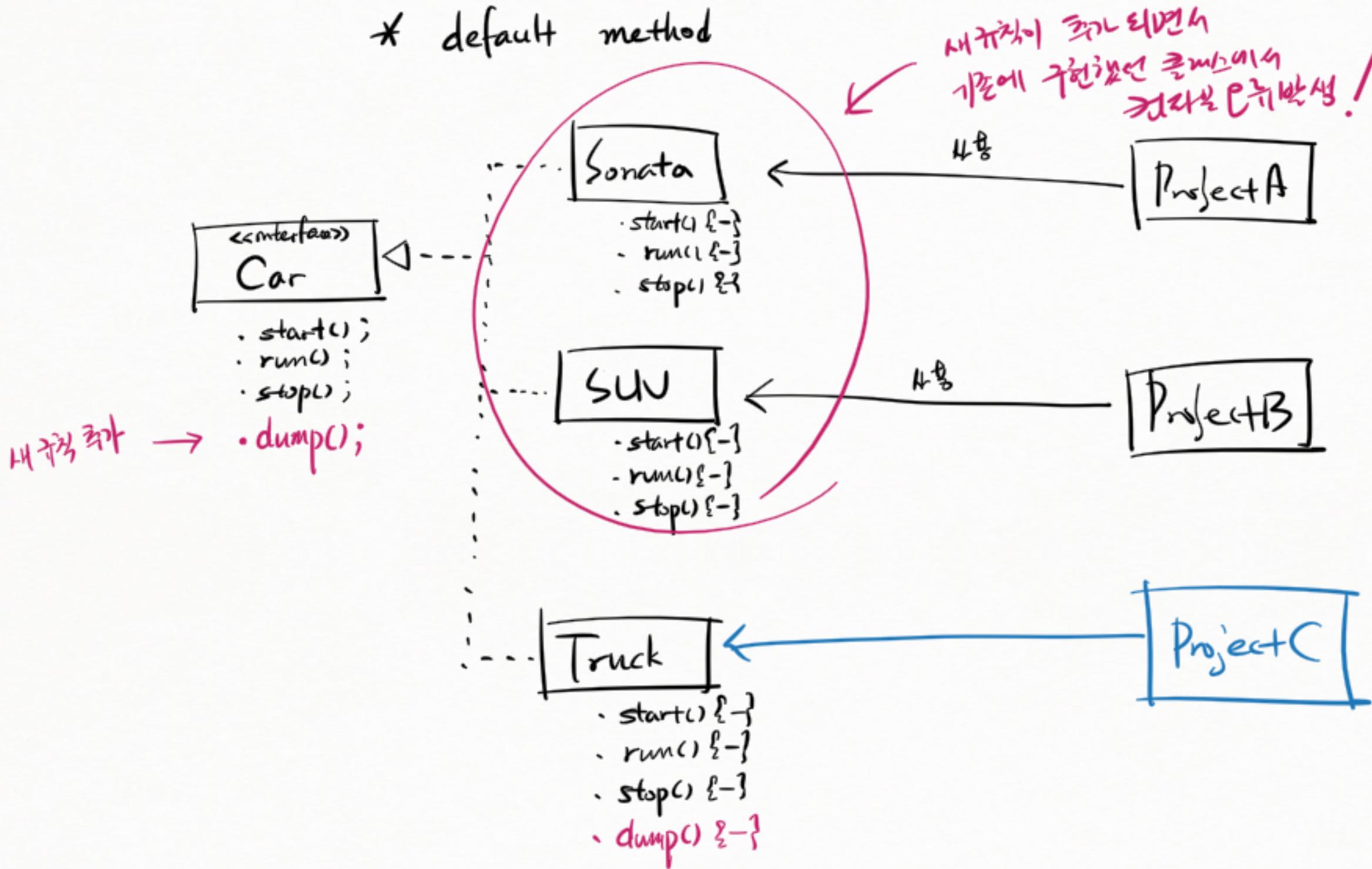


단점
m1(), m2()가 두 클래스에 모두 포함되어 있어 관리가 번거롭다.

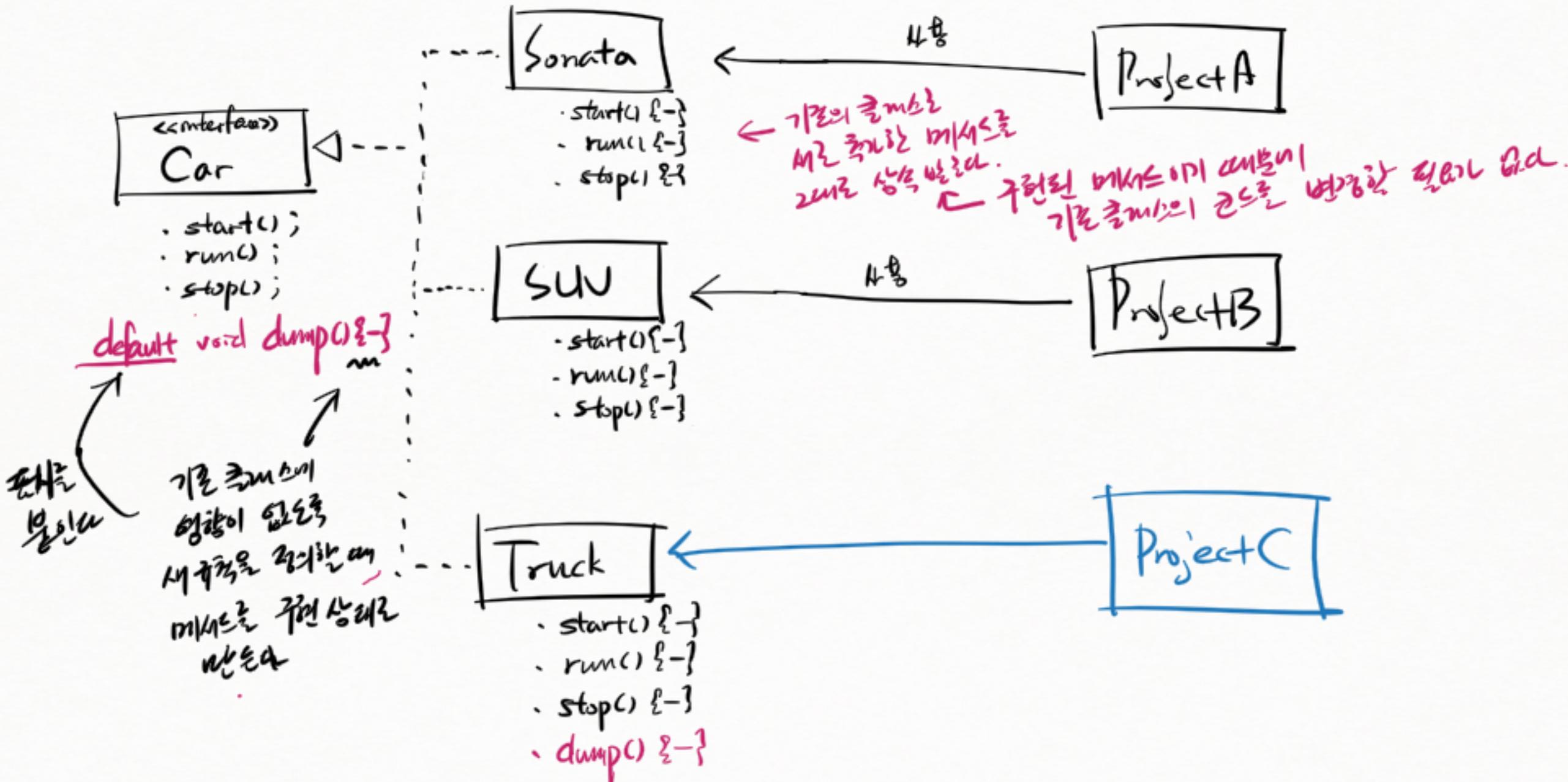
제거
단점
유지보수에 불편성이 있다.



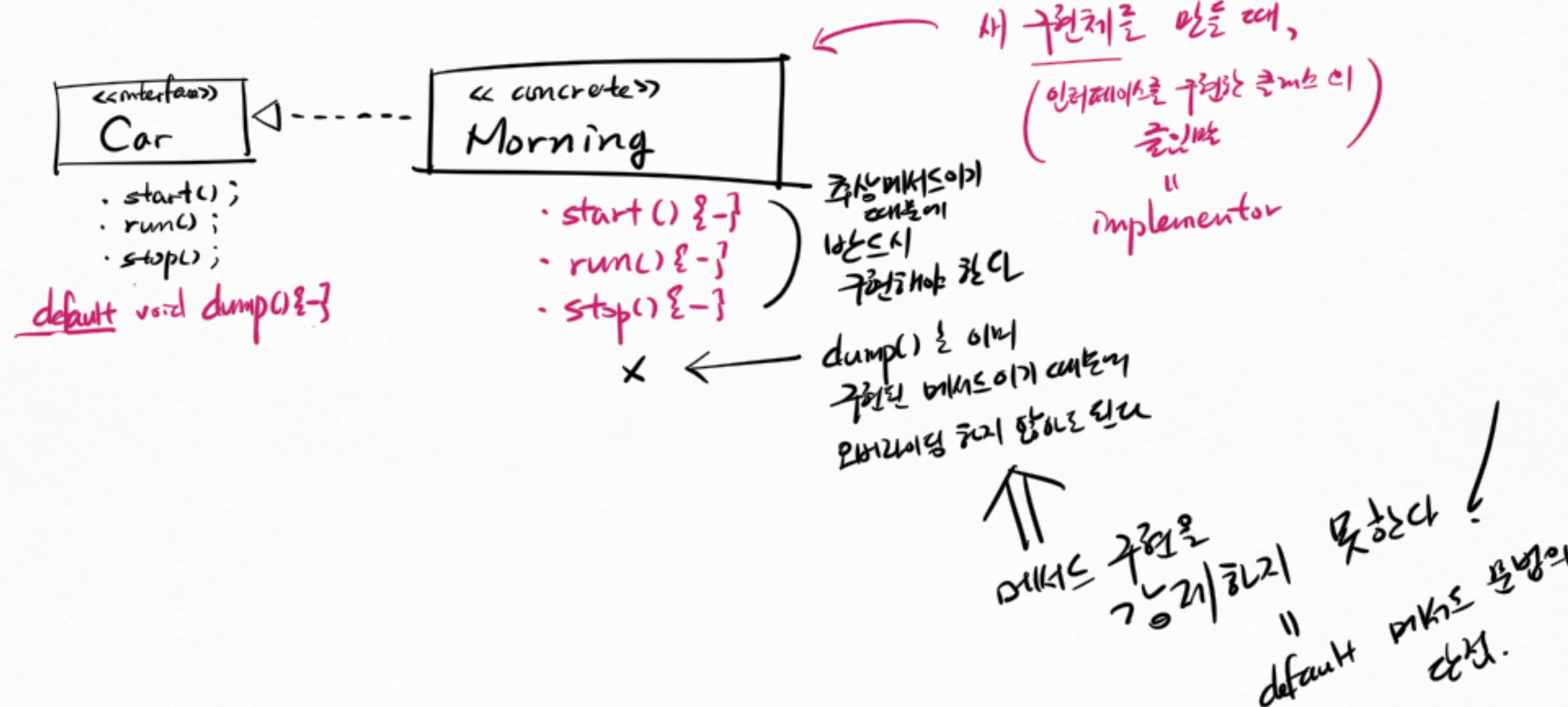
* default method



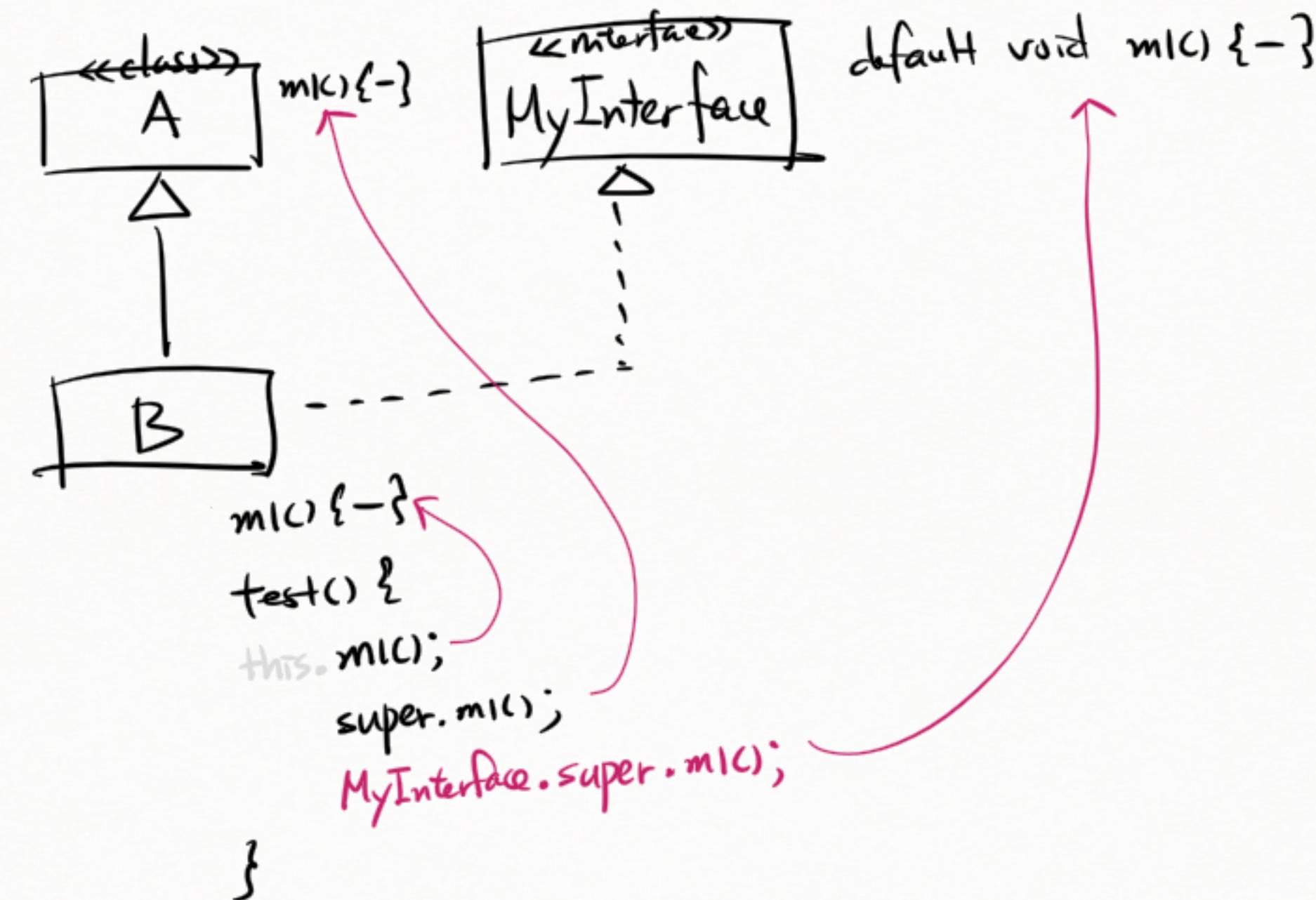
* default method ← 기존의 구현 클래스의 상황을 유지하면서
새 규칙을 추가하고 싶을 때,



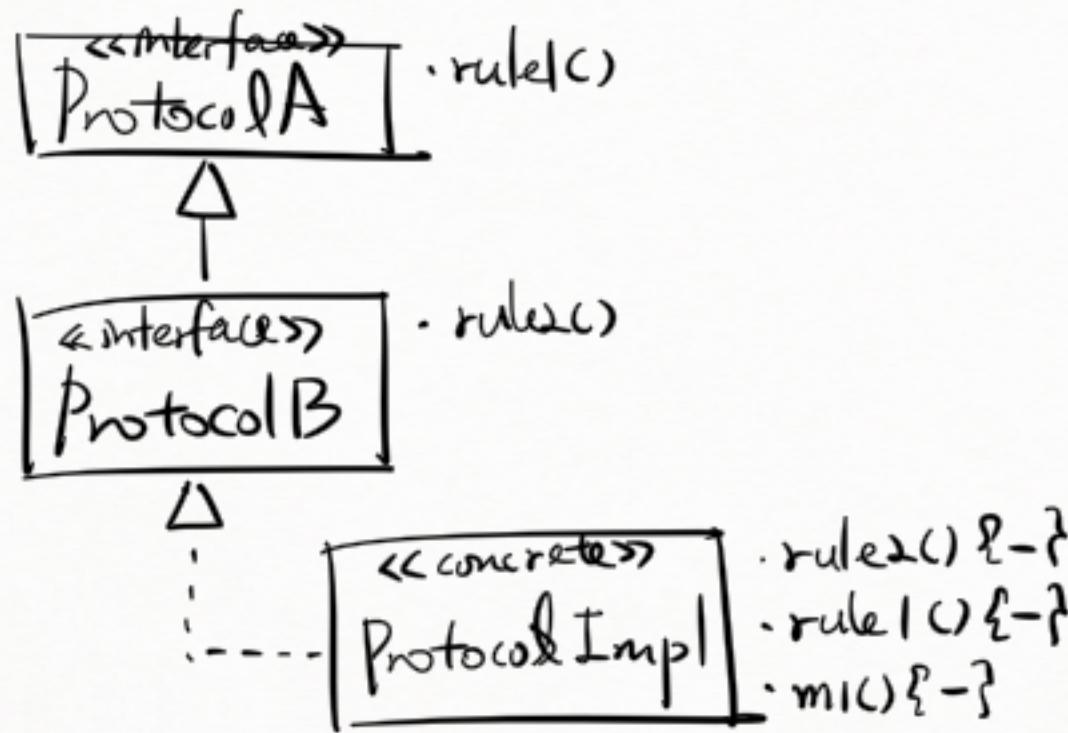
* default method of Σ



* super el. 인터페이스.super



* 인터페이스 상속 - oop.ex09.c.*

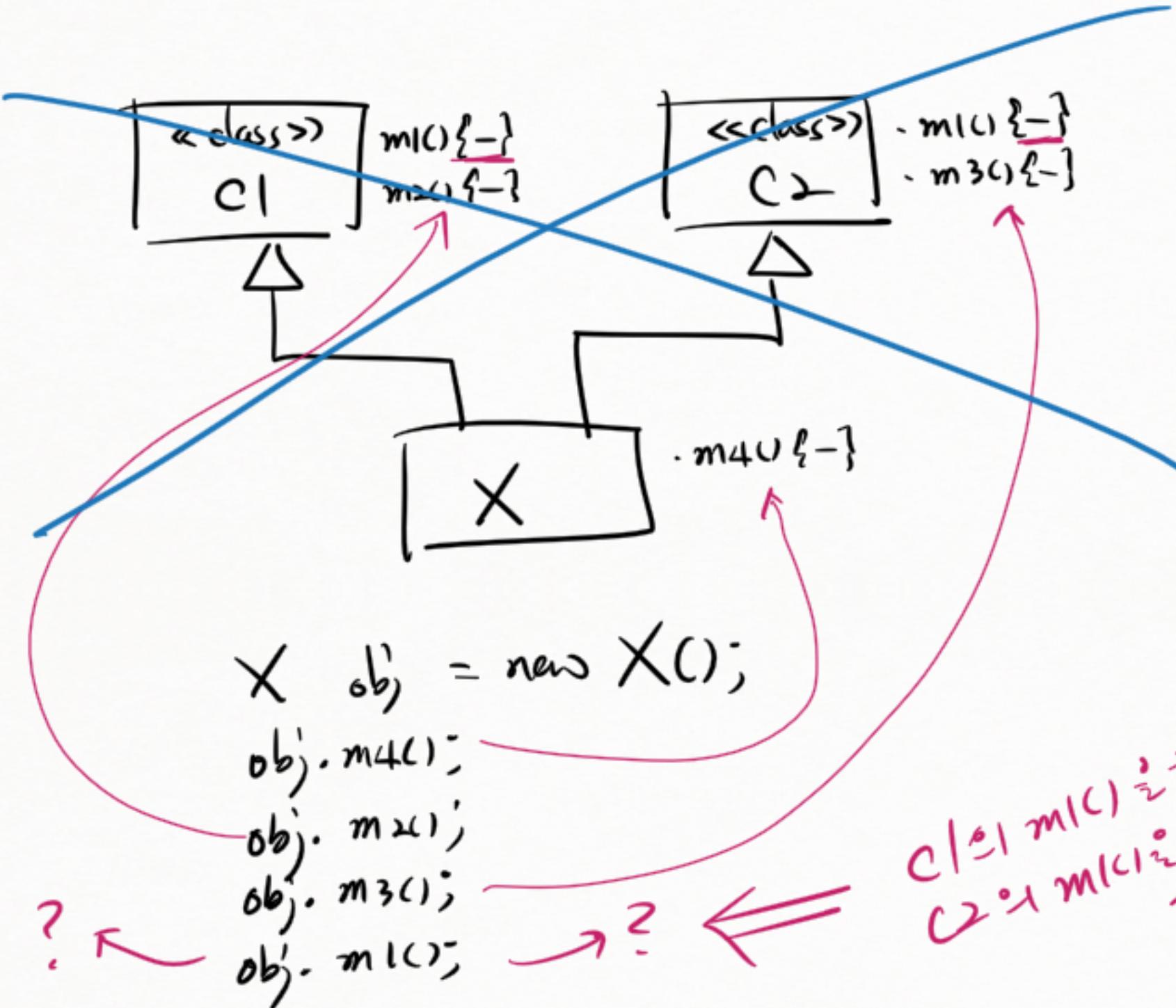


ProtocolImpl obj = new ProtocolImpl();
obj.m1(); // ok
obj.rule2(); // ok
obj.rule1(); // ok

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

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

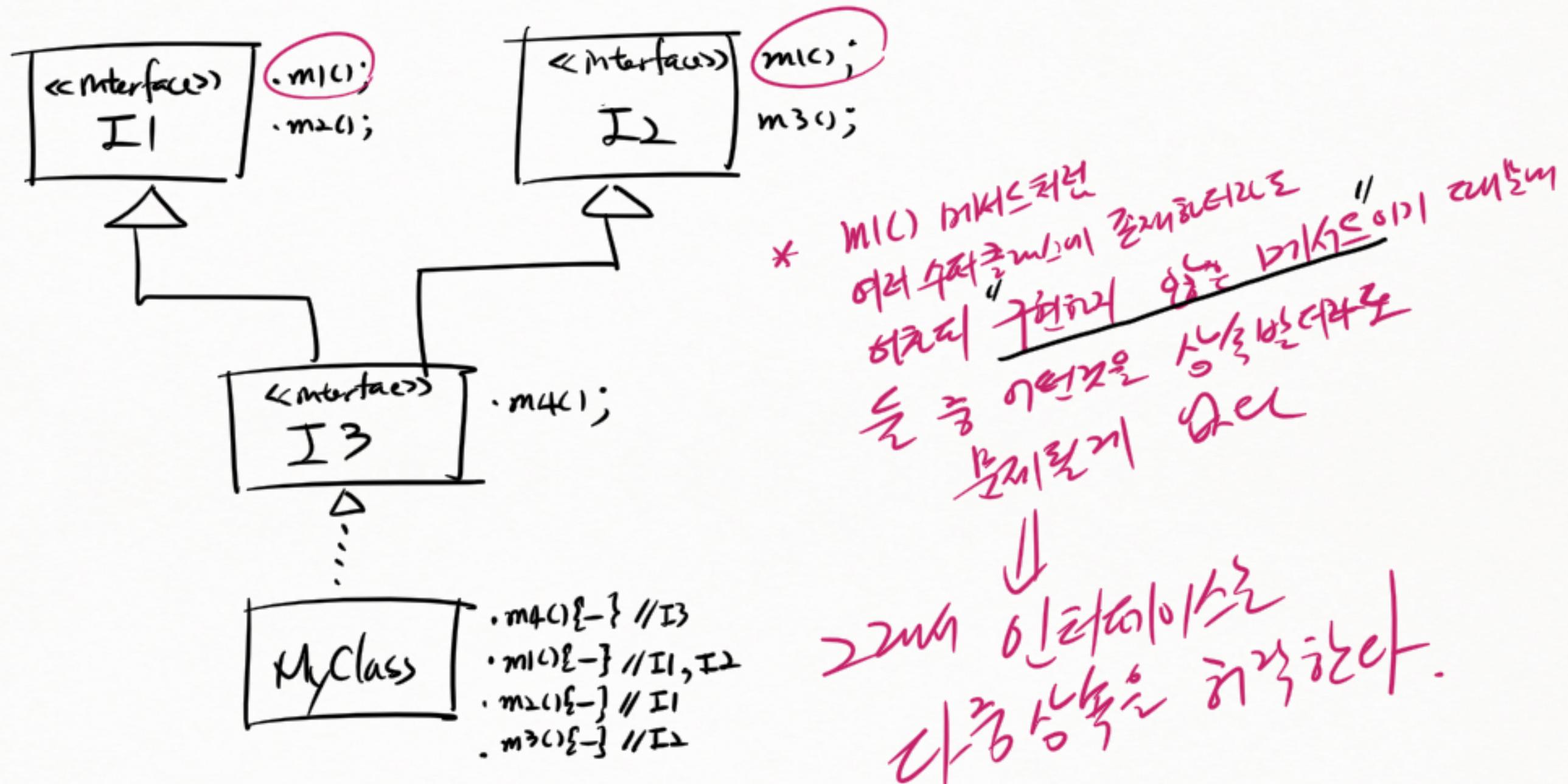
* ~~클래스~~ 다음 상속과 인터페이스 다음 상속



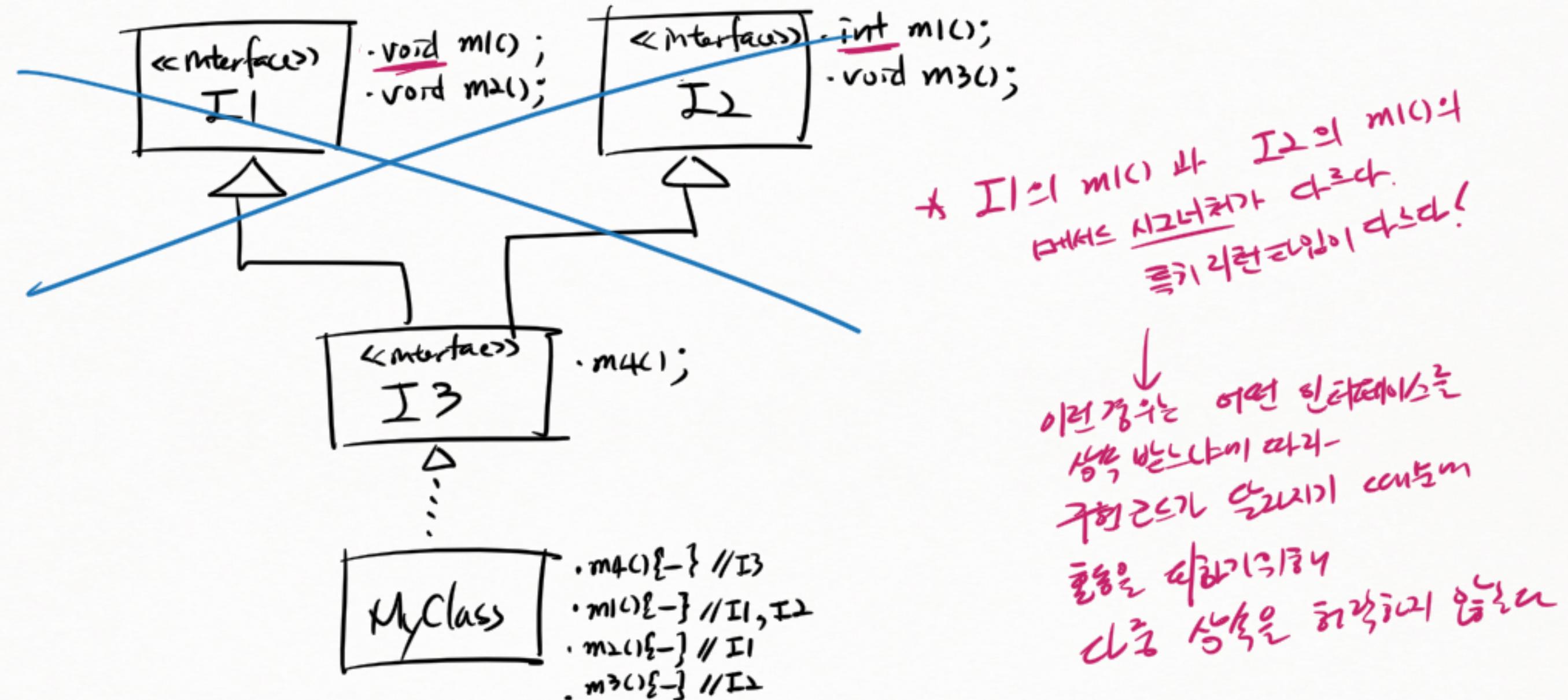
2번쨰 차종 상속은 예상치 않음.

CH?
m1()은 충돌하는가?
m3()은 충돌하는가?

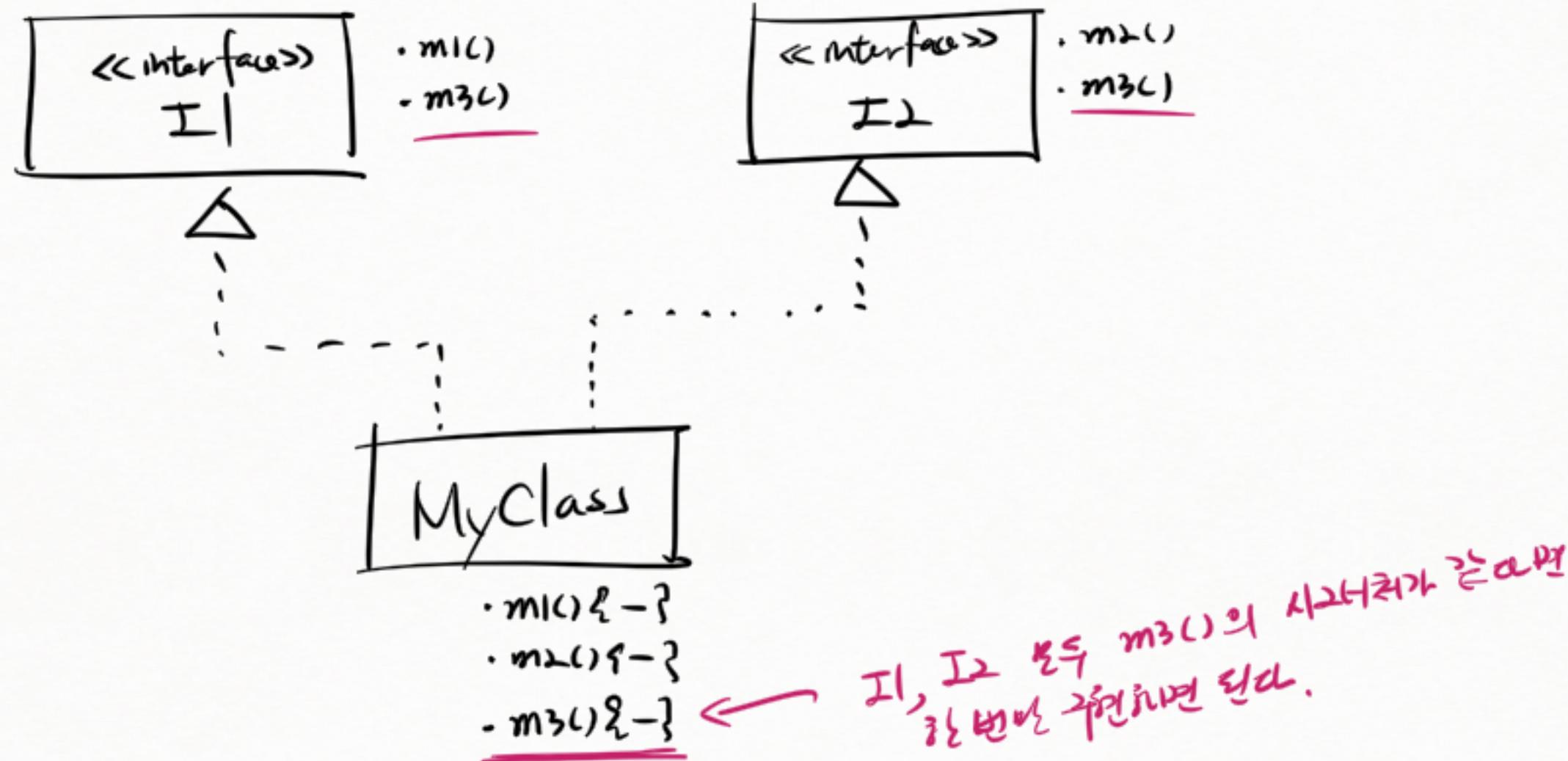
* $\frac{1}{2}$ m1 다중 상속과 인터페이스 다중상속



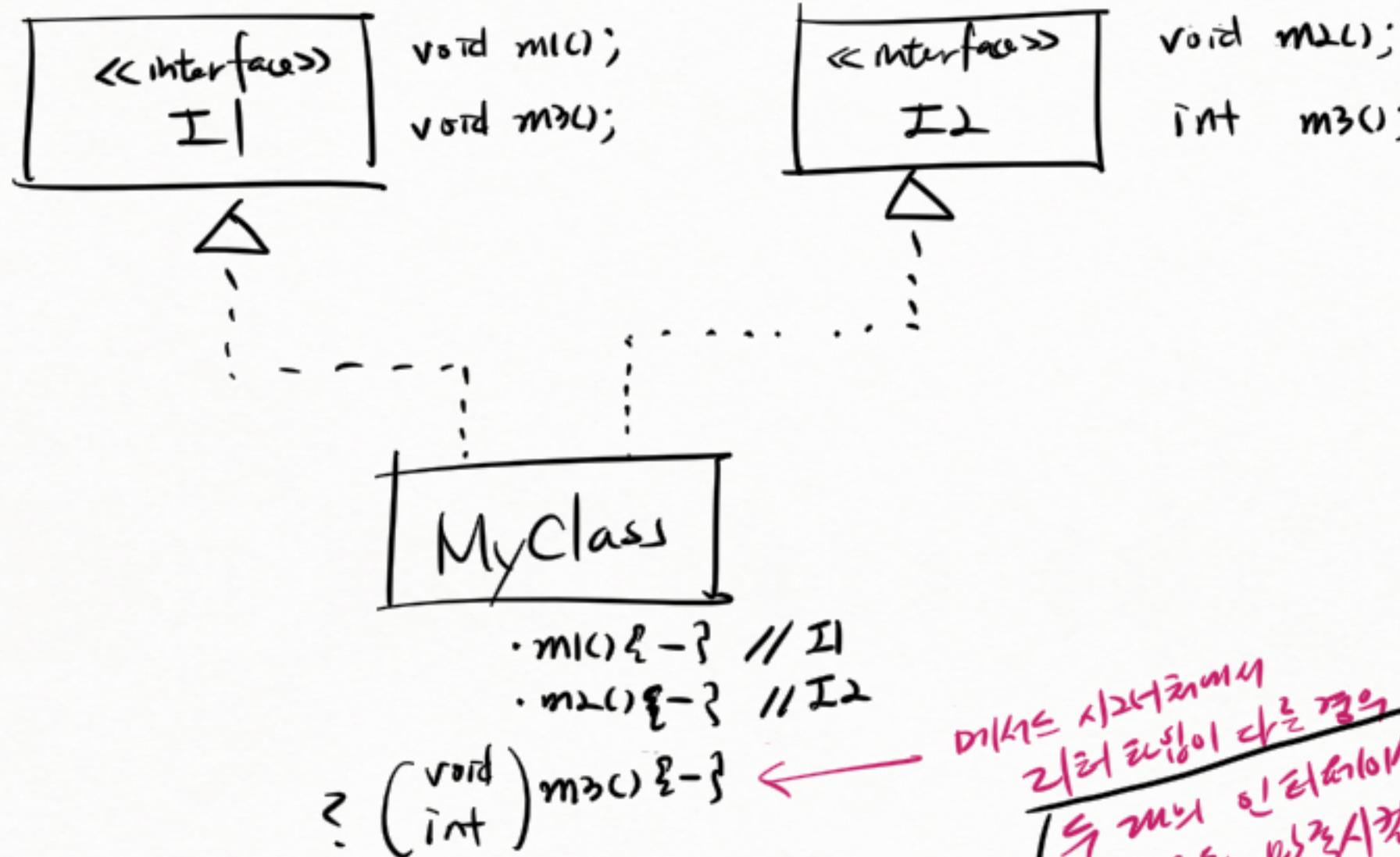
* 인터페이스 다중 상속 불가!



* 인터페이스의 상속 특성

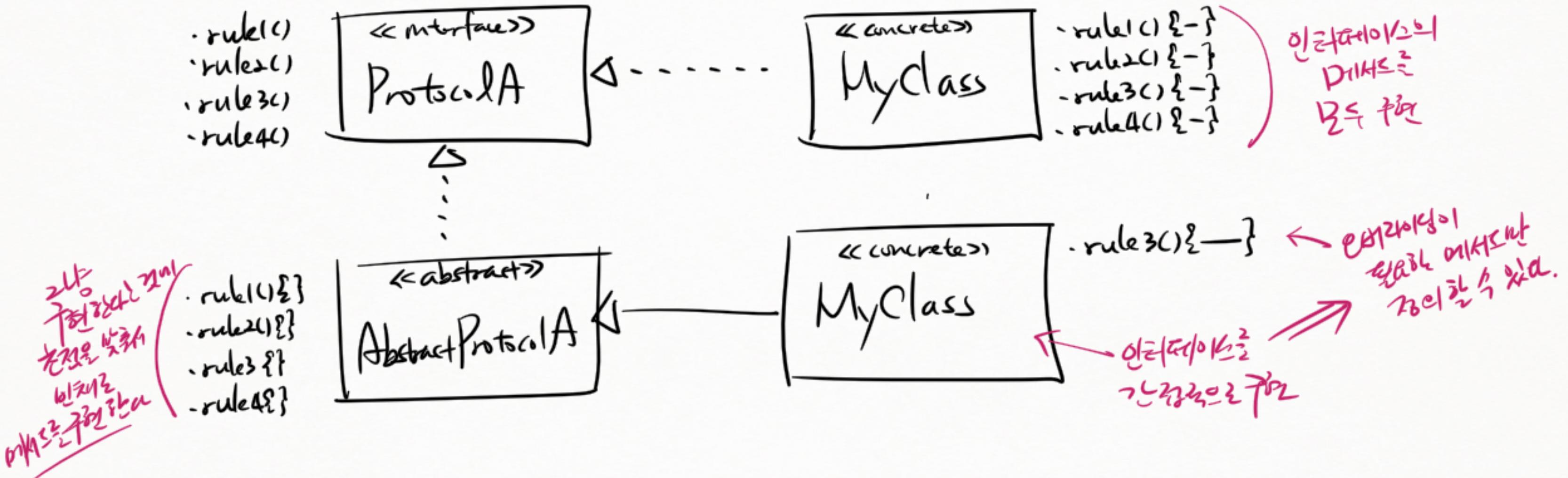


* 인터페이스는 다형성을 위한!

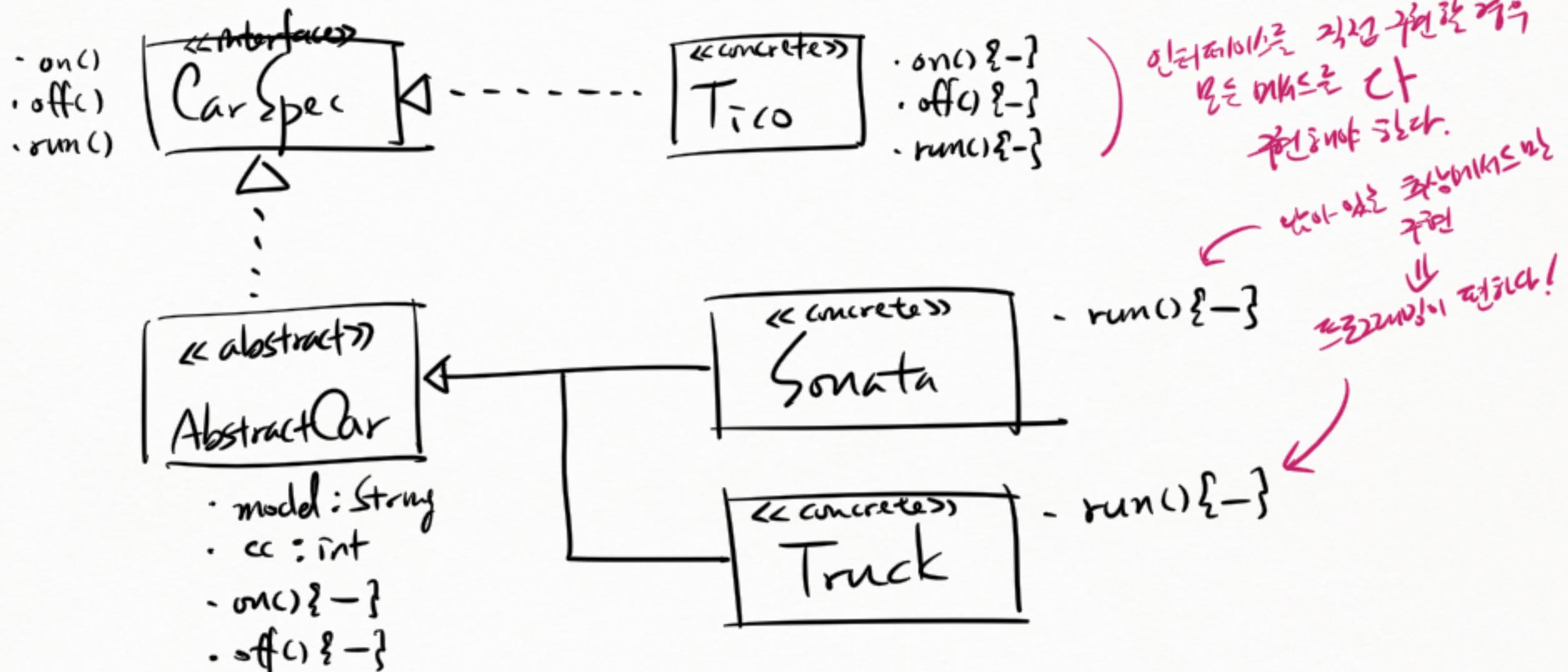


MyClass는 시그니처만 나온다
하지만 이미 다른 경로
MyClass는 인터페이스를
만족하는지 확인하는
방법이 있다!
그러면 다형성을 위한
문제가!

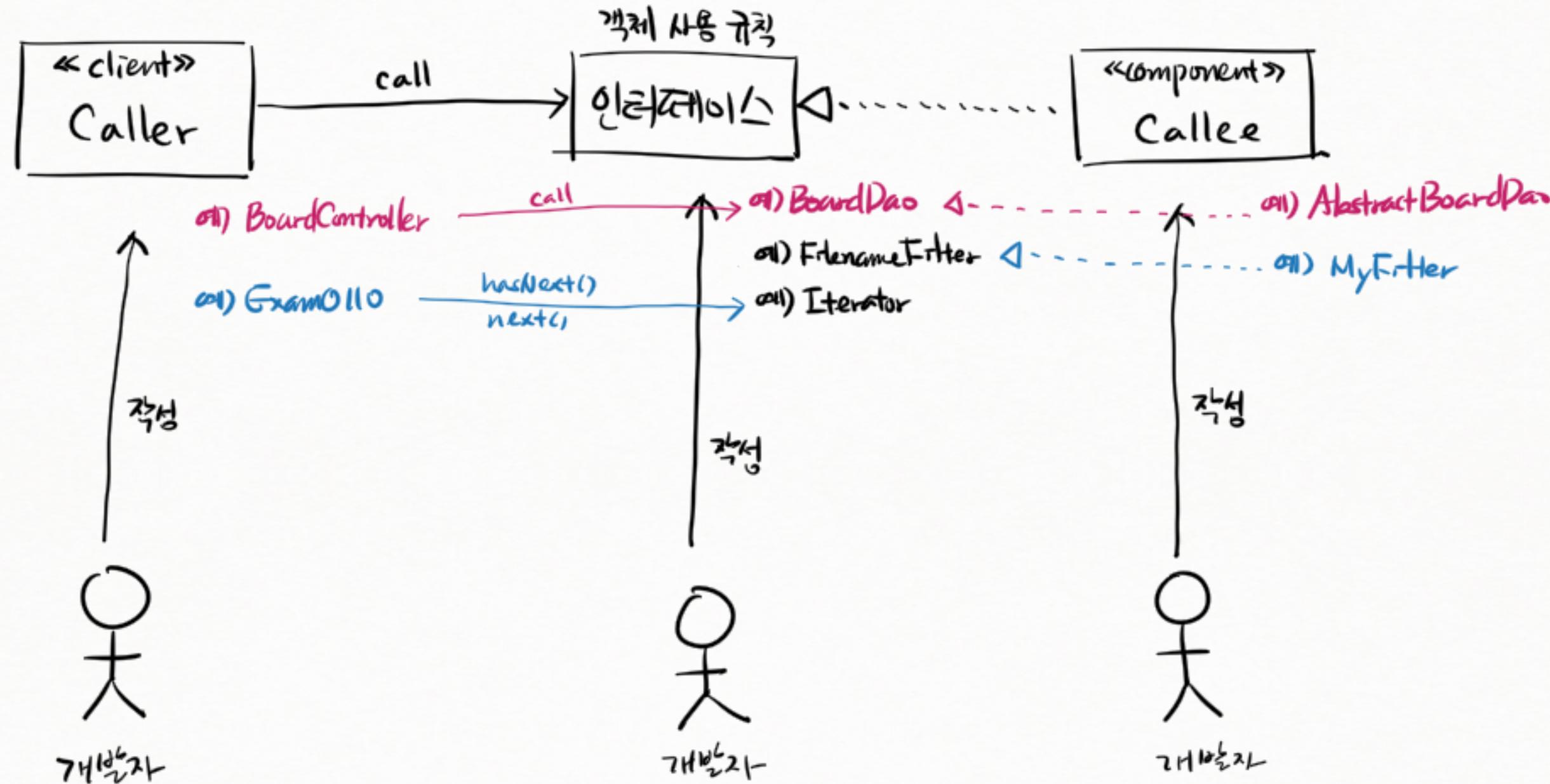
* 인터페이스와 추상 클래스의 혼란



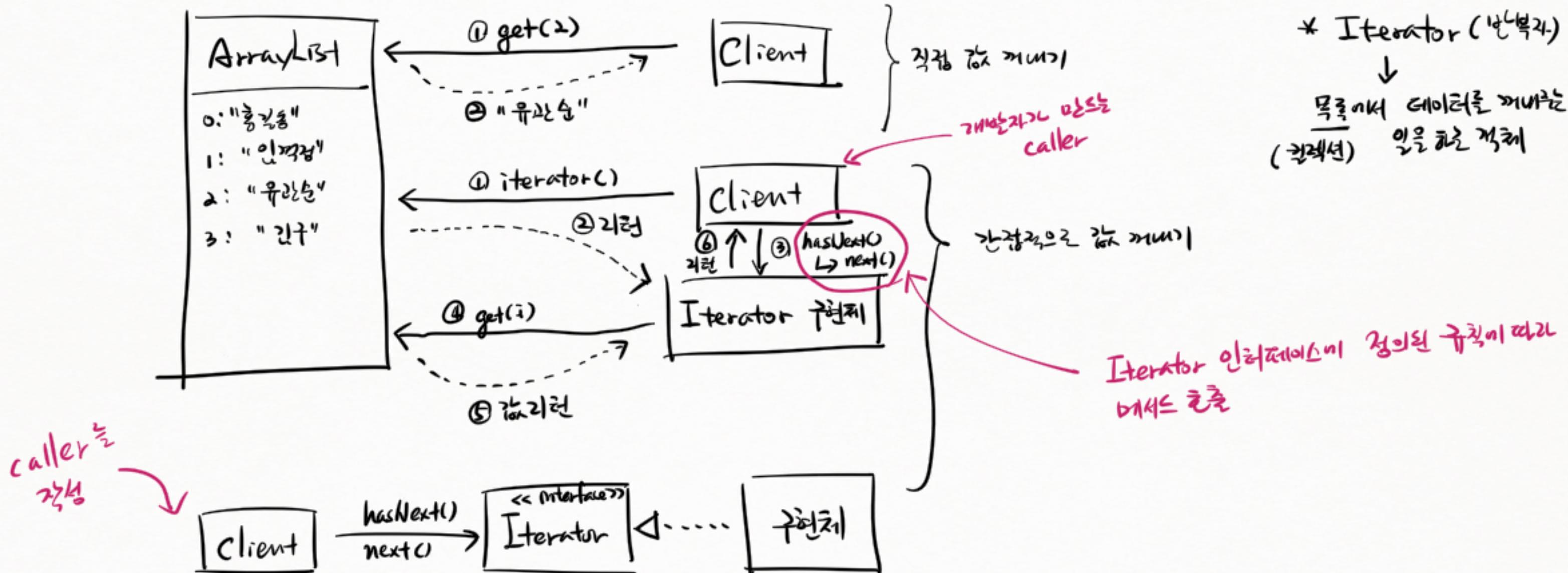
* 인터페이스와 추상클래스 활용 예)



* 인터페이스와 구현체



* Caller 개발 입장



* callee 2 번째 괄호

