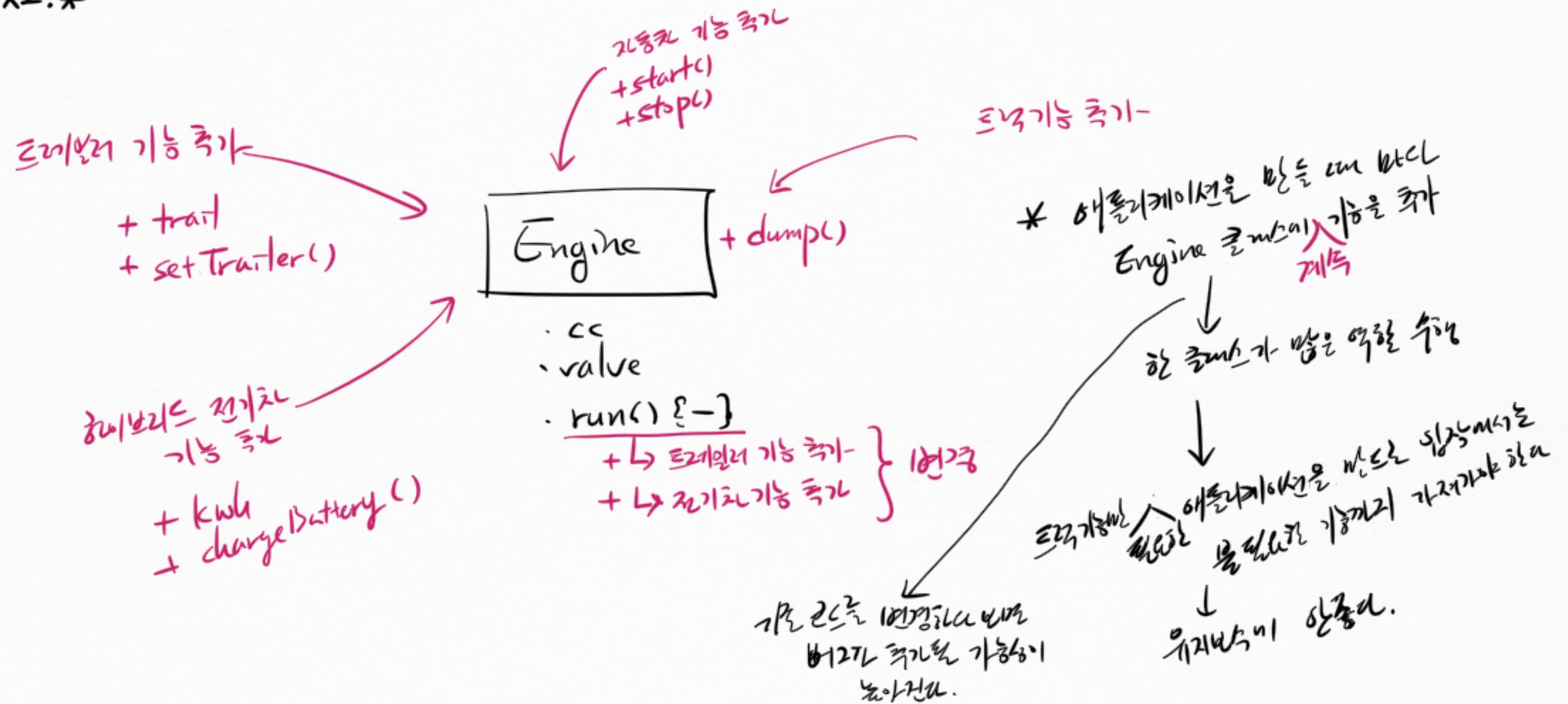


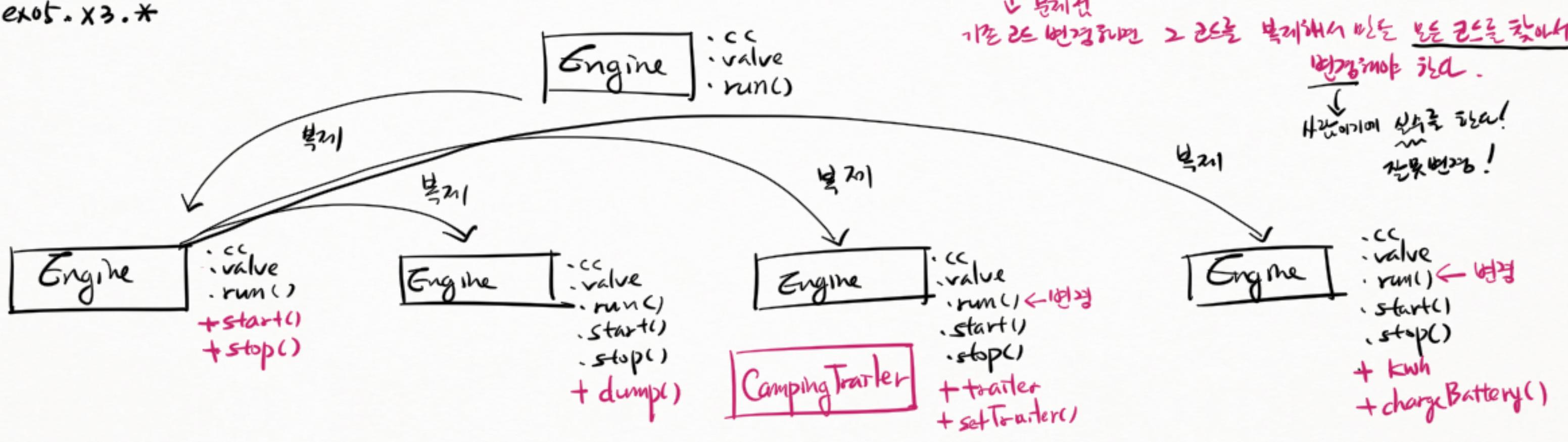
* 기능 확장 방법 1 - 기존 코드에 새 기능 추가

oop.ex5.x2.*



* 기능 확장 방법 2 - 복제한 코드에 새 기능 추가

oop.ex05.*



① 자동차 만들기
app1

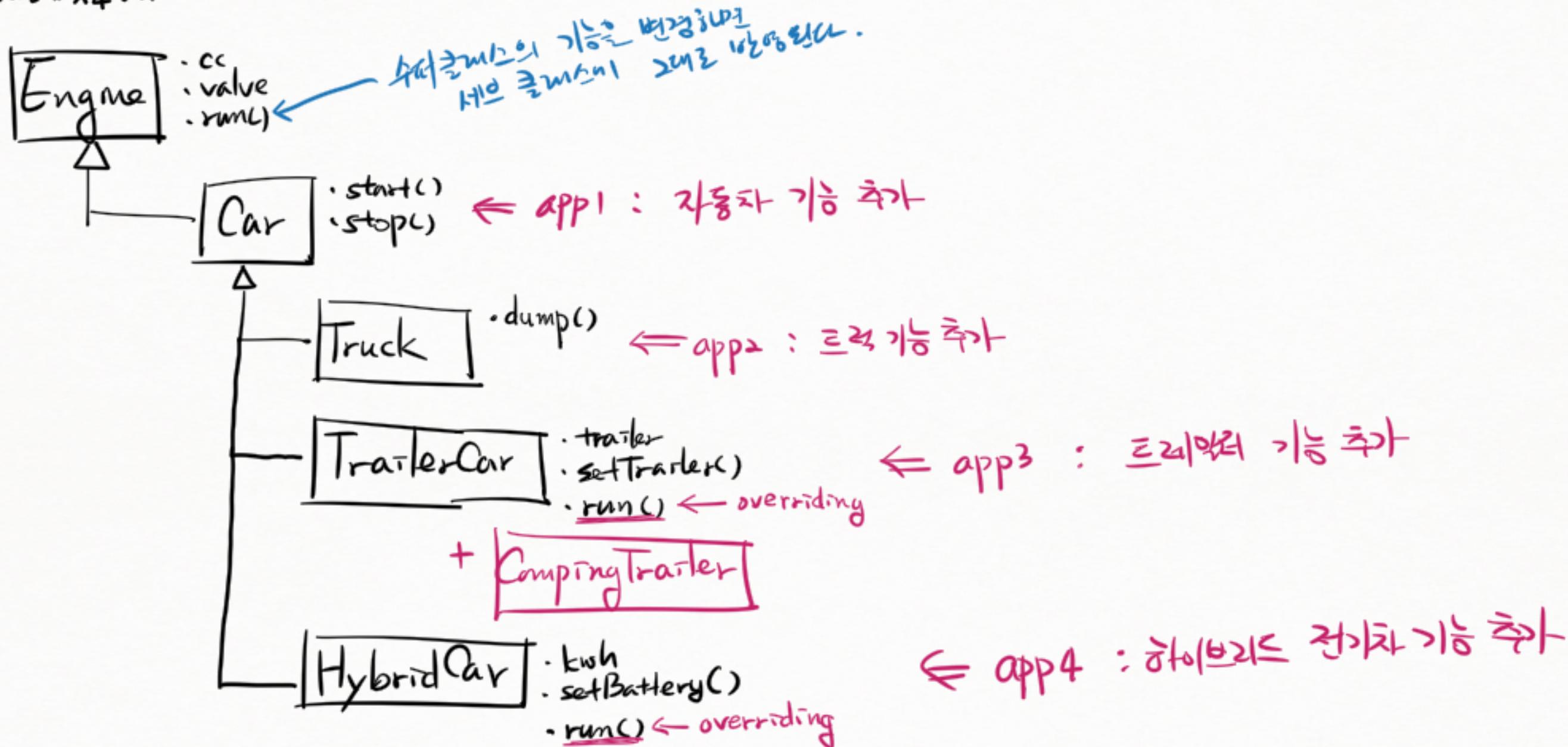
② 트레일러 만들기
app2

③ 캠핑카 만들기
app3

④ 차량 모니터 전기차 만들기
app4

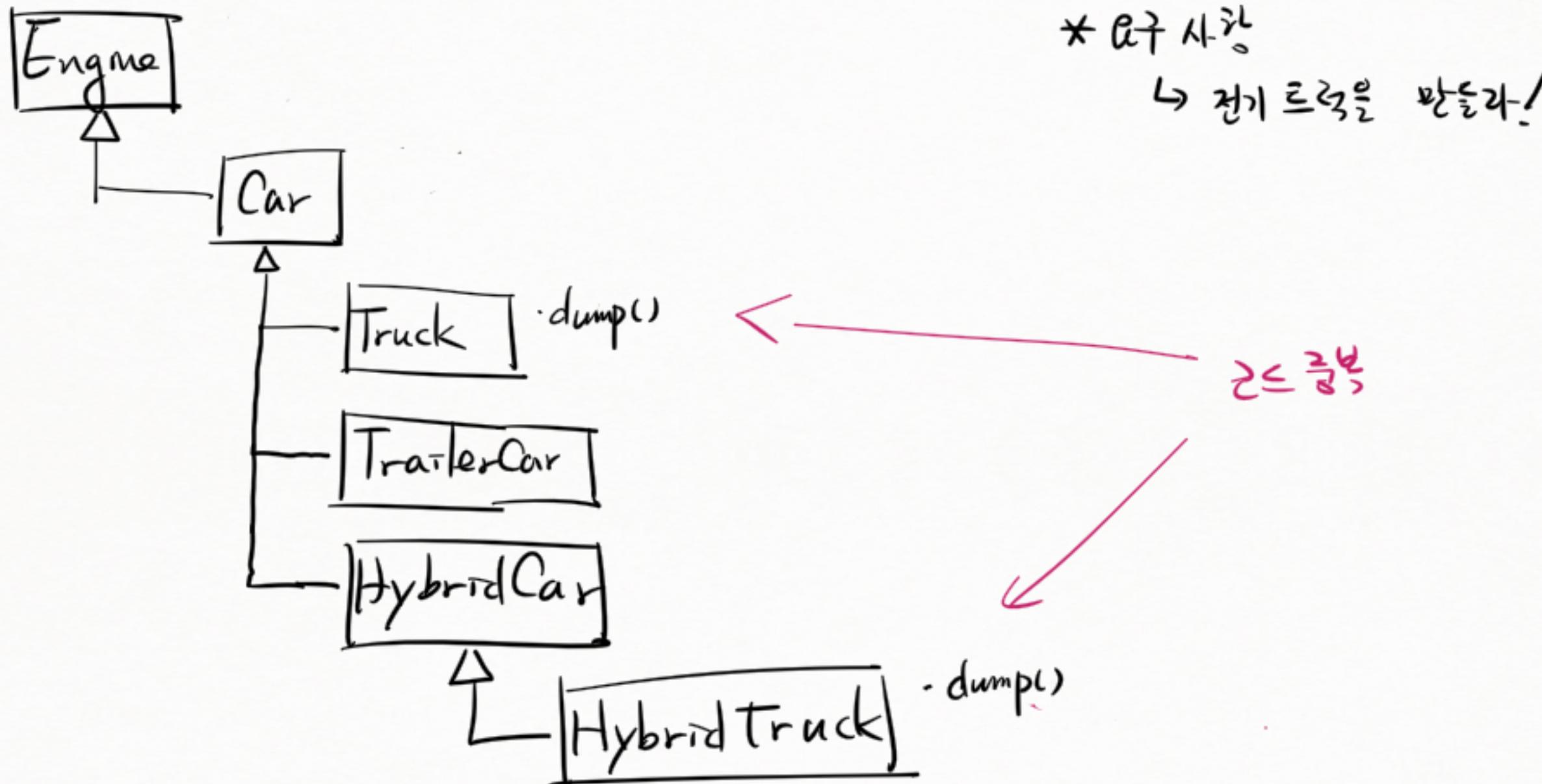
* 기능 확장 방법 3 - 상속을 이용한 확장.

oop.ex05.x4.*



* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한계

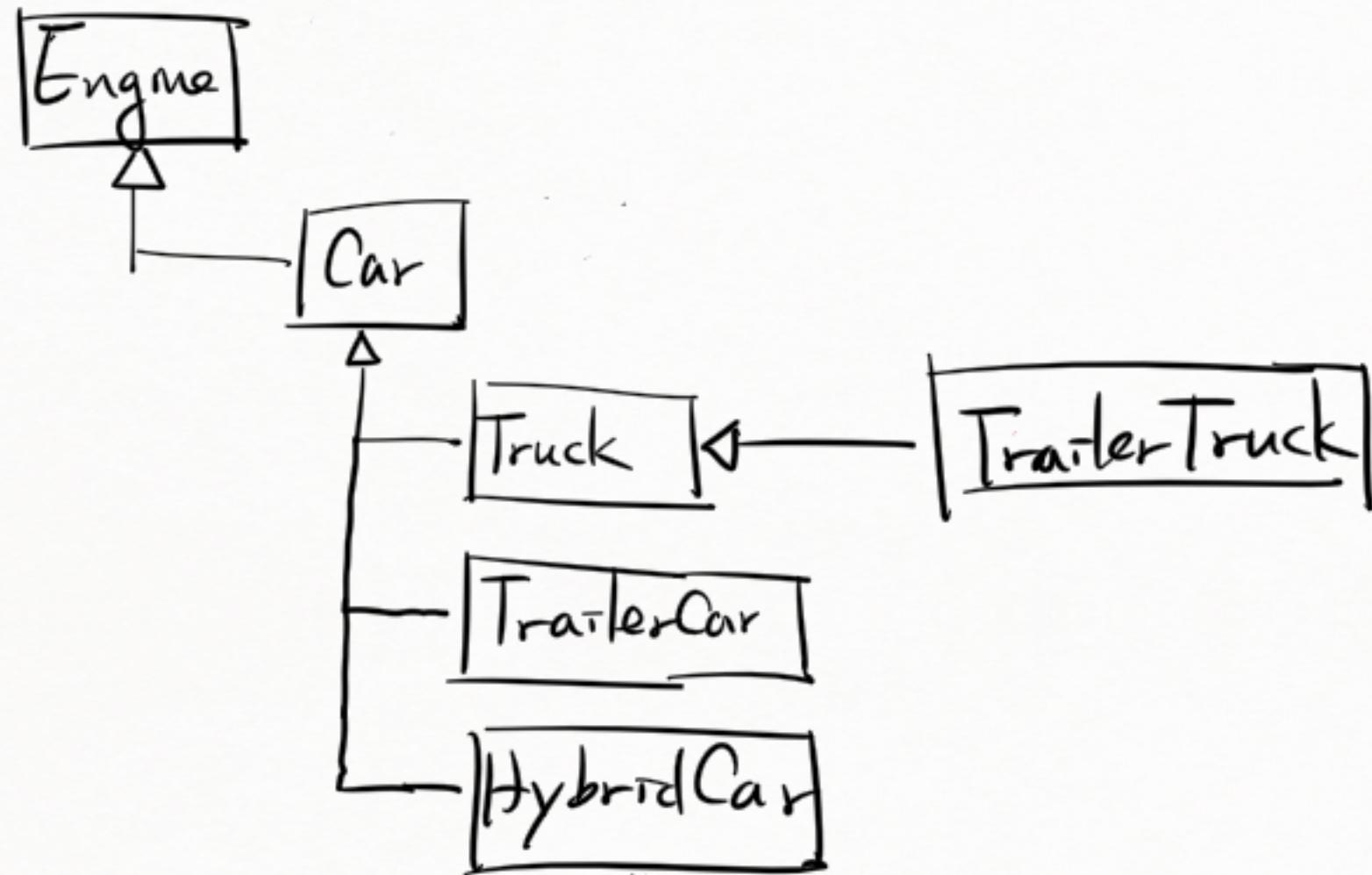
oop.ex05.x4.*



* 기능 확장
↳ 전기 트럭을 만들라!

* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한가지

oop. ex05. x4.*



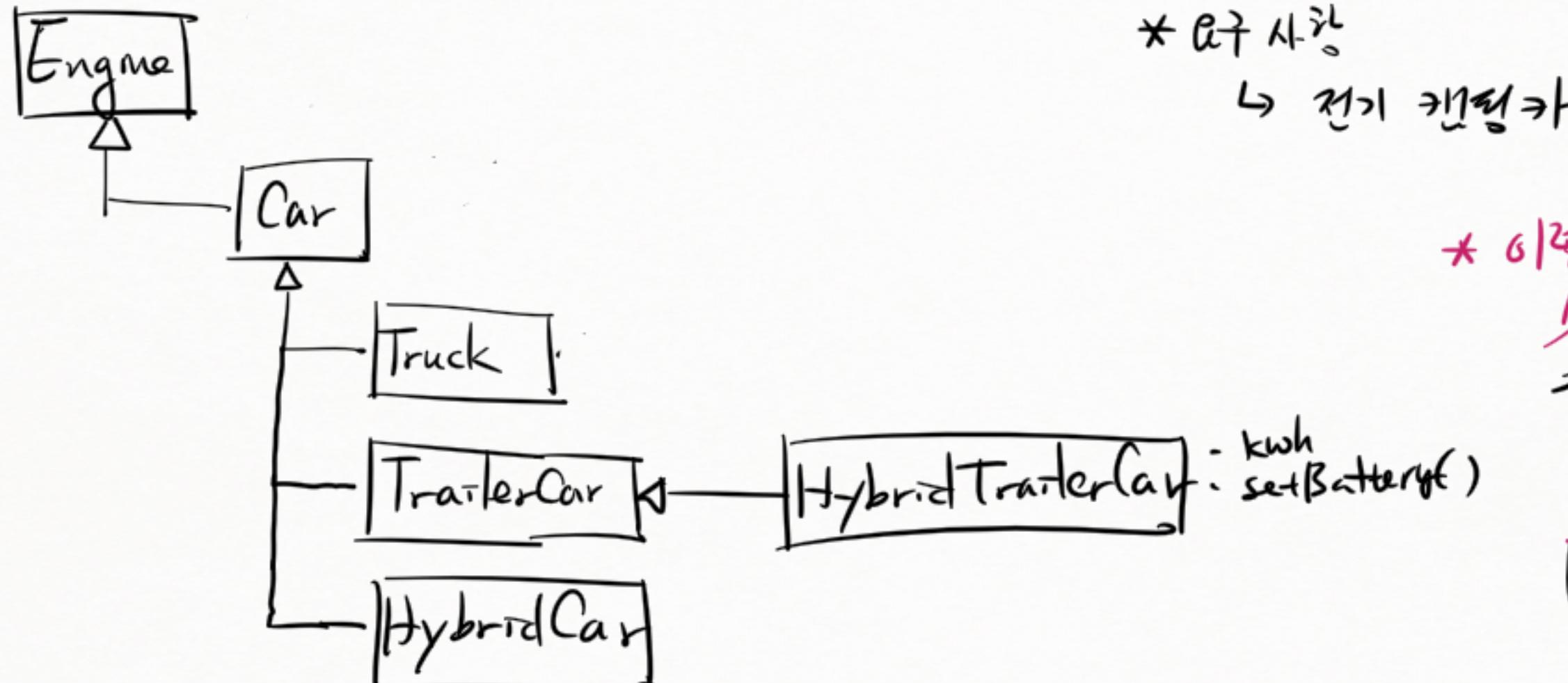
* @구사장

↳ 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

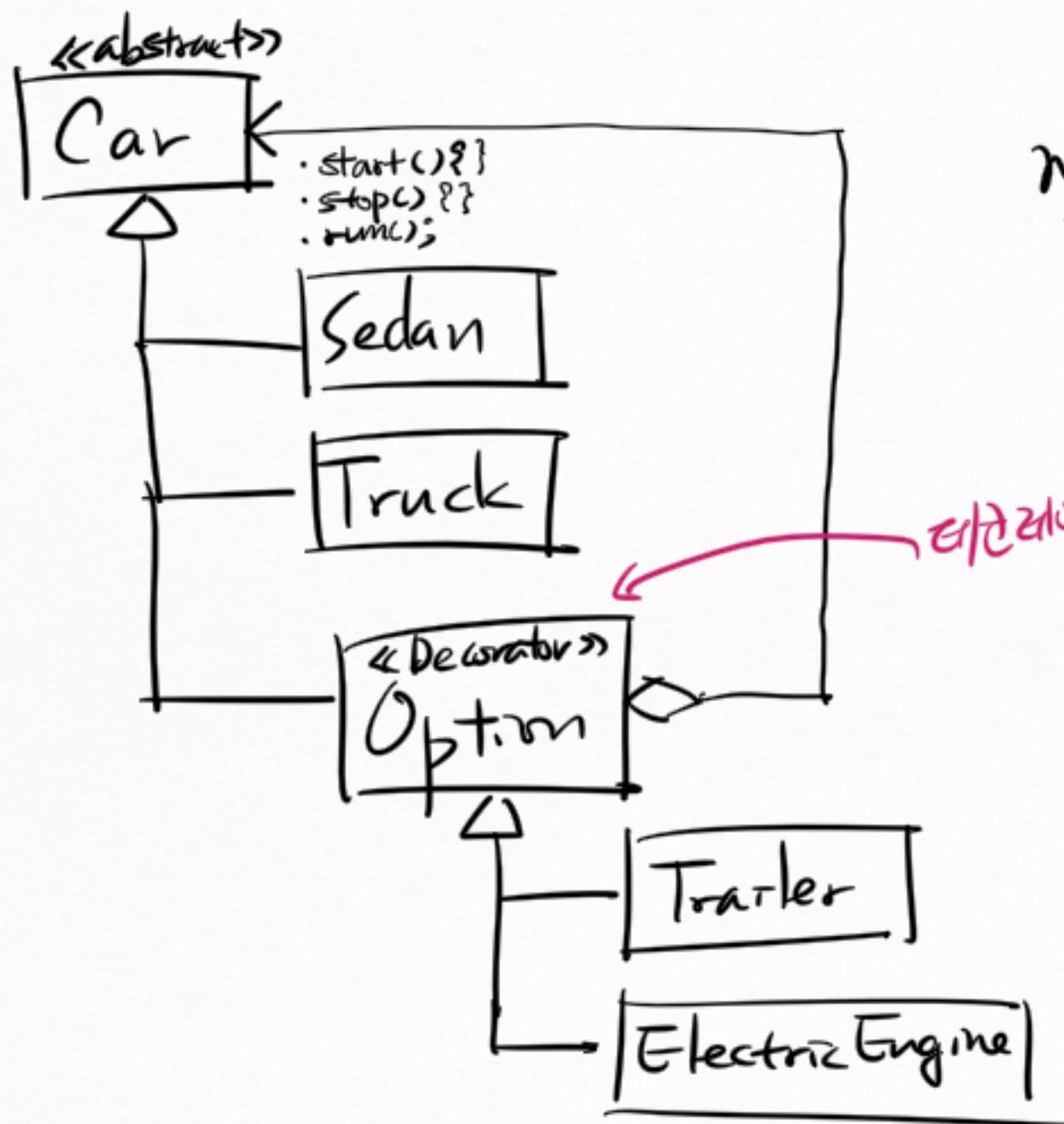
선택적

Gas Engine

Electric Engine

Trailer

* 테러리아 키미 기획

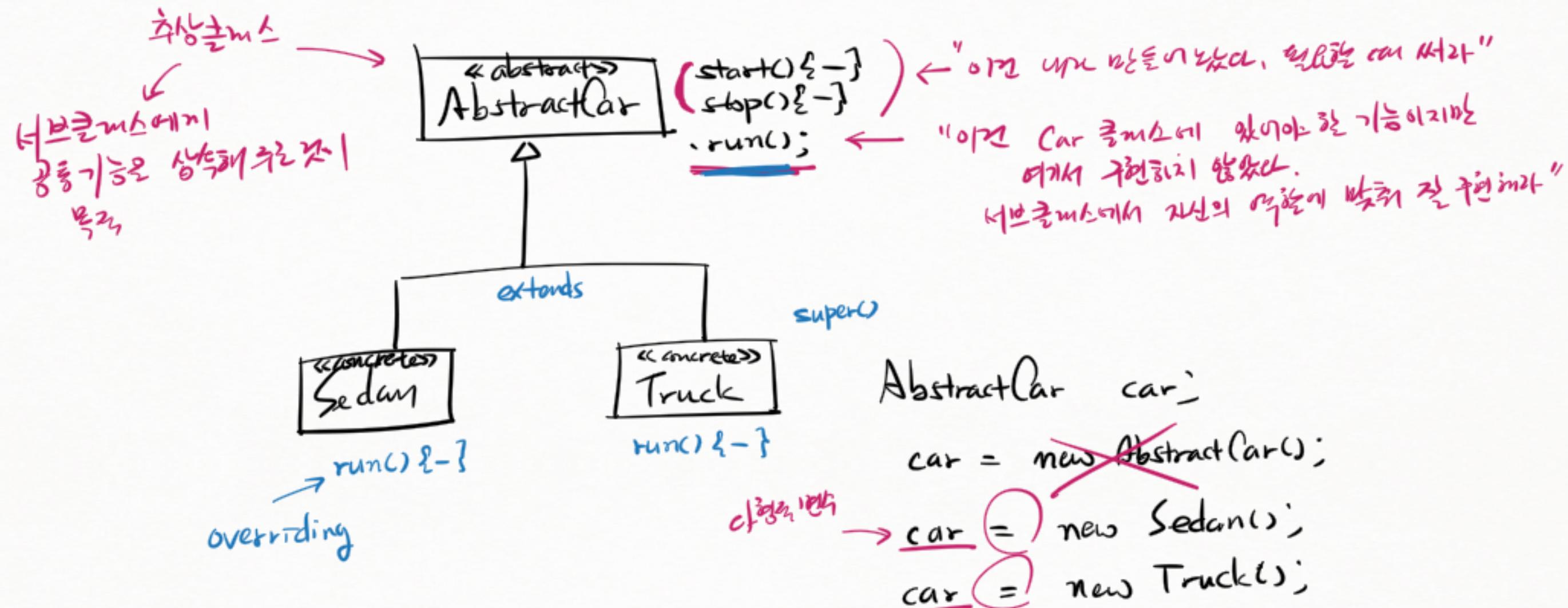


구례여러 : 각 구체에게 봄이로 선택 가능

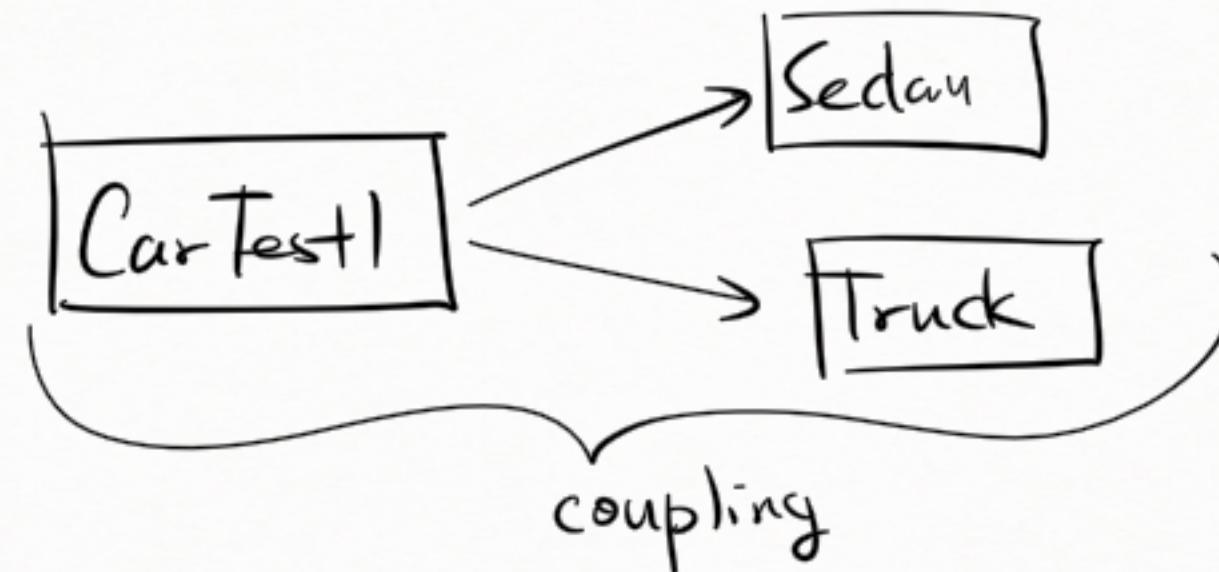
nas 8m3 (2km)

기 32
Korean 03
한국어 03
03d.

* 추상클래스와 구현



* 실전 프로그래밍 1 단계



coupling

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

Sedan

Truck

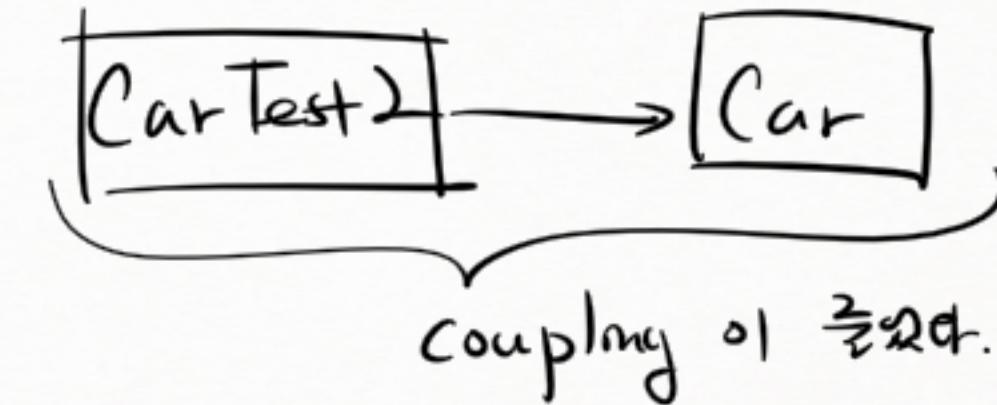
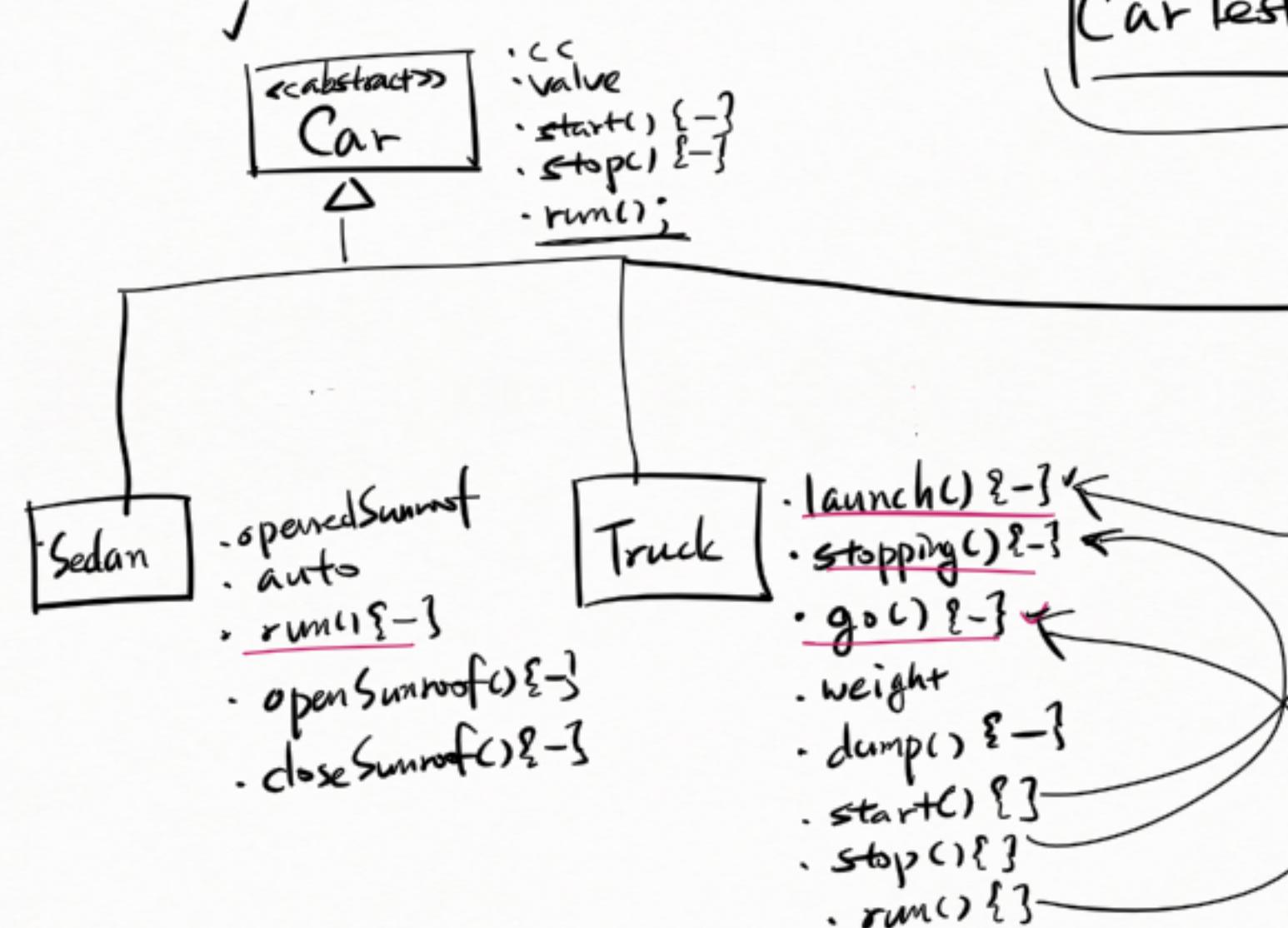
Object

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

제작
이동
제작
제작
제작
제작

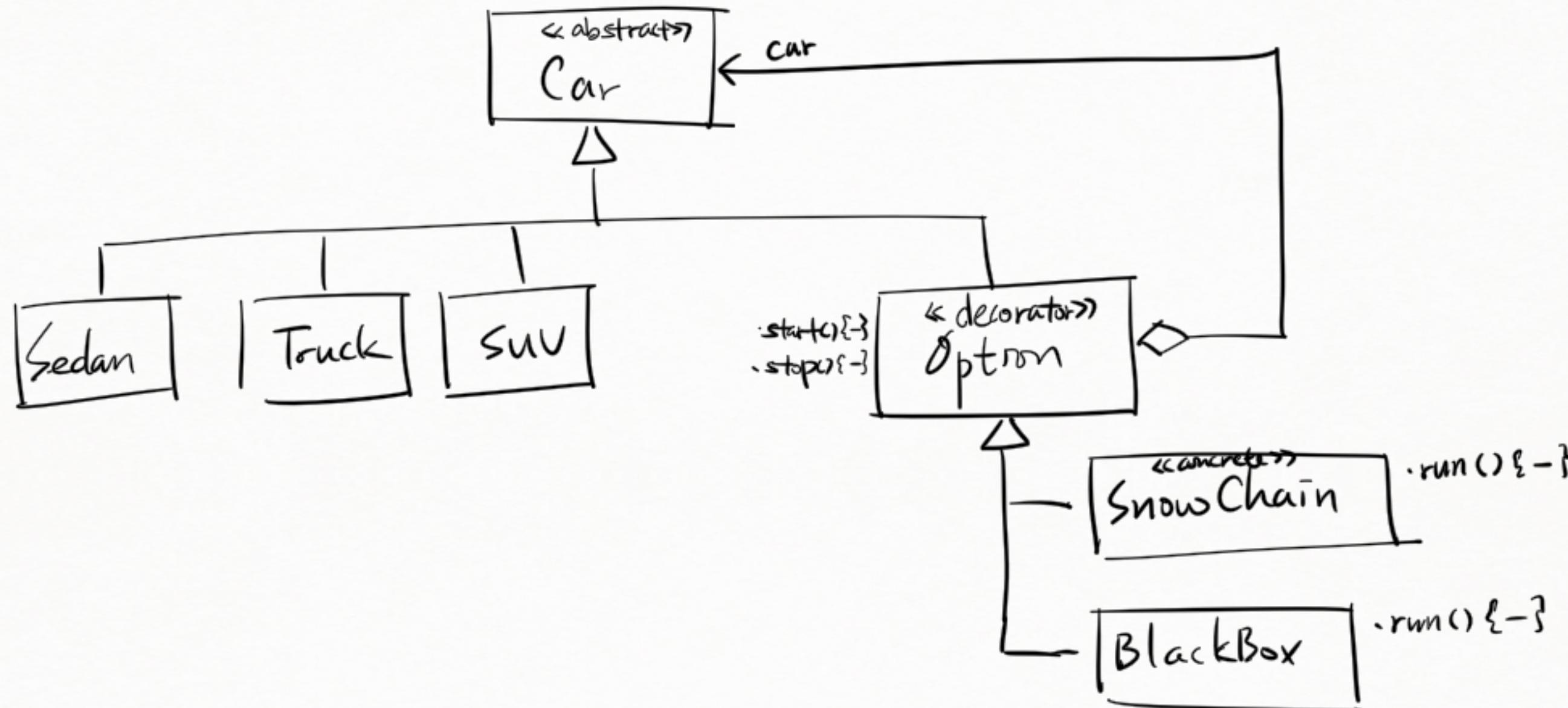
* 상 하 한 한 한 한 한 - 품종, 품질, 성능

\hookrightarrow : generalization

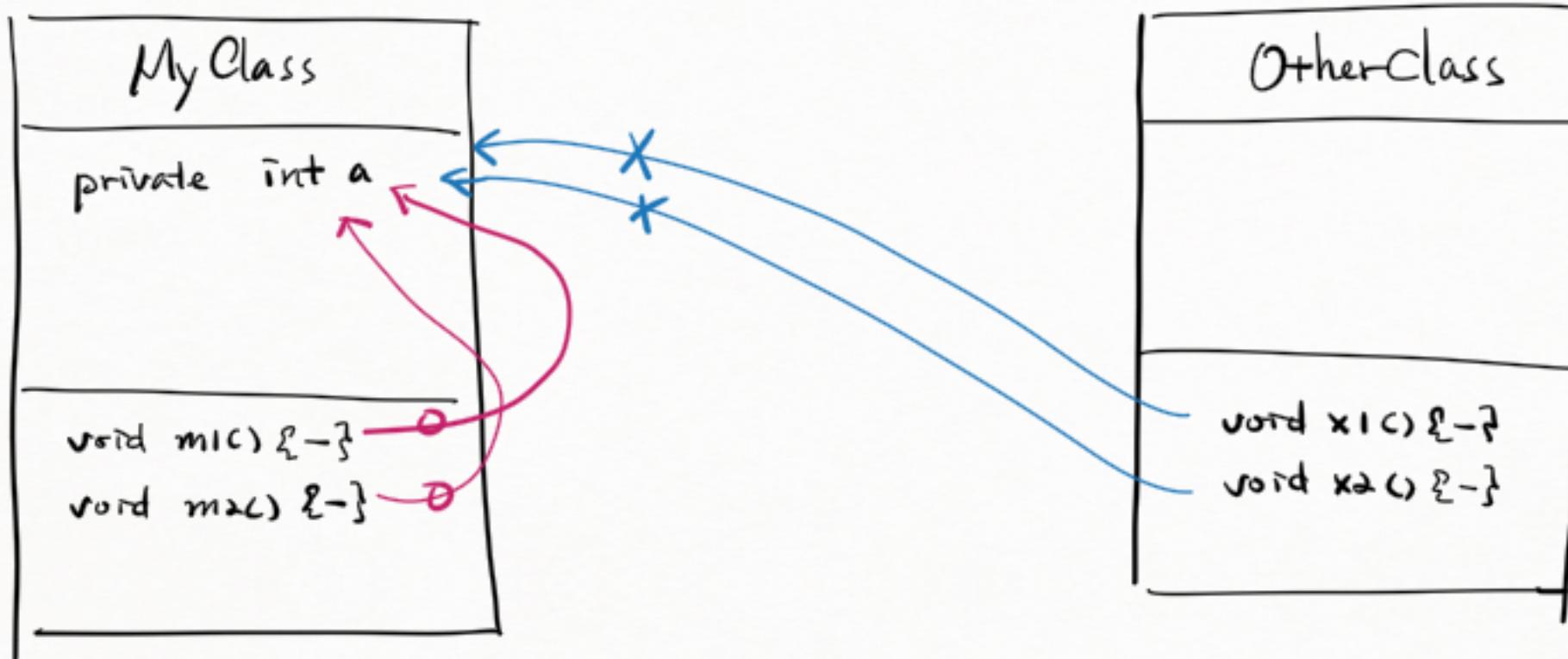


- enableTow: boolean
- activateTow(boolean)
- run() { }

* 차선을 3개로 나누면 차량이 1개로 통합

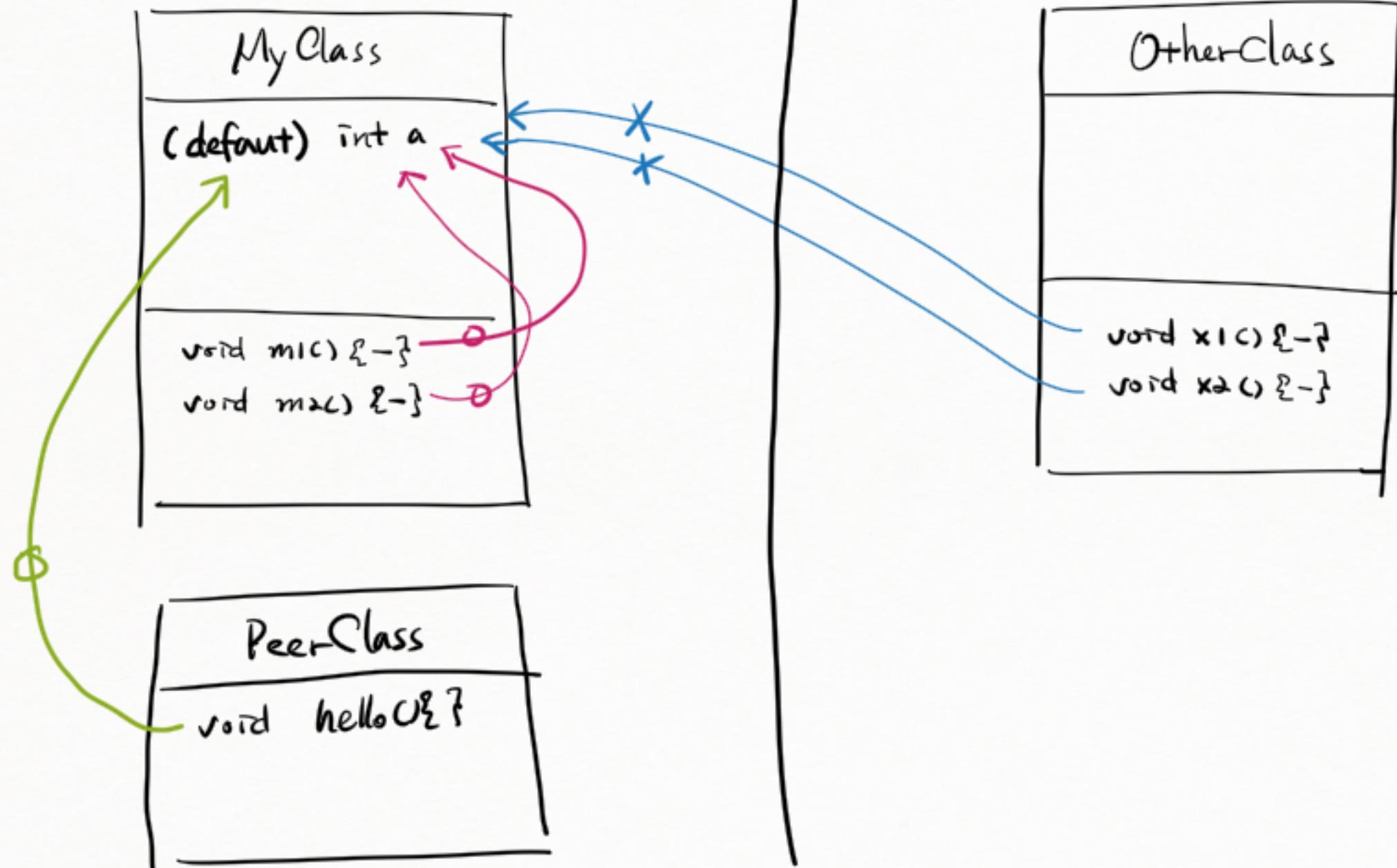


* private - 같은 클래스의 멤버만 접근 가능



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

com.eomcs.test1



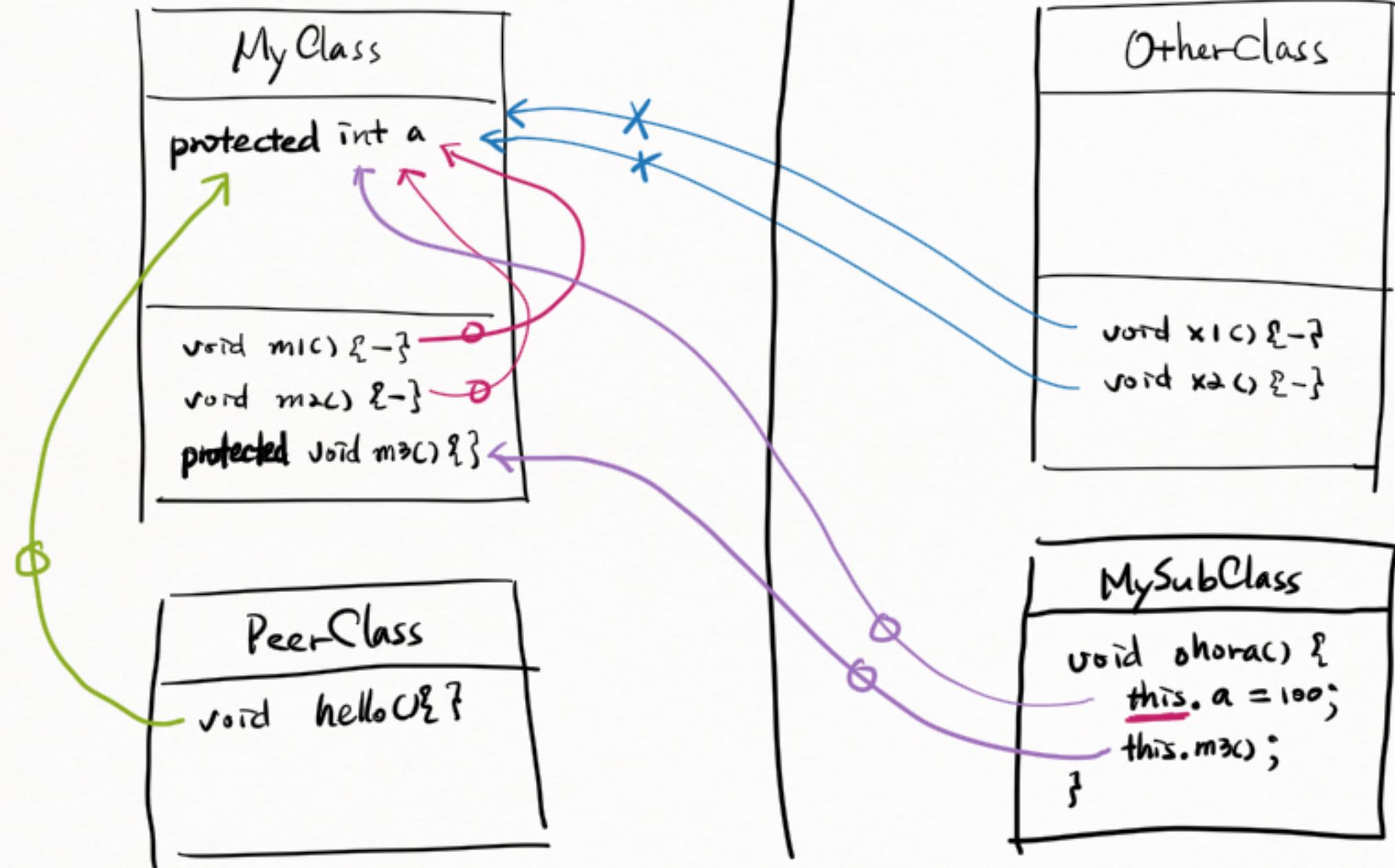
* protected - 같은 패키지 내의 멤버 접근 가능

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

com.eomcs.test1

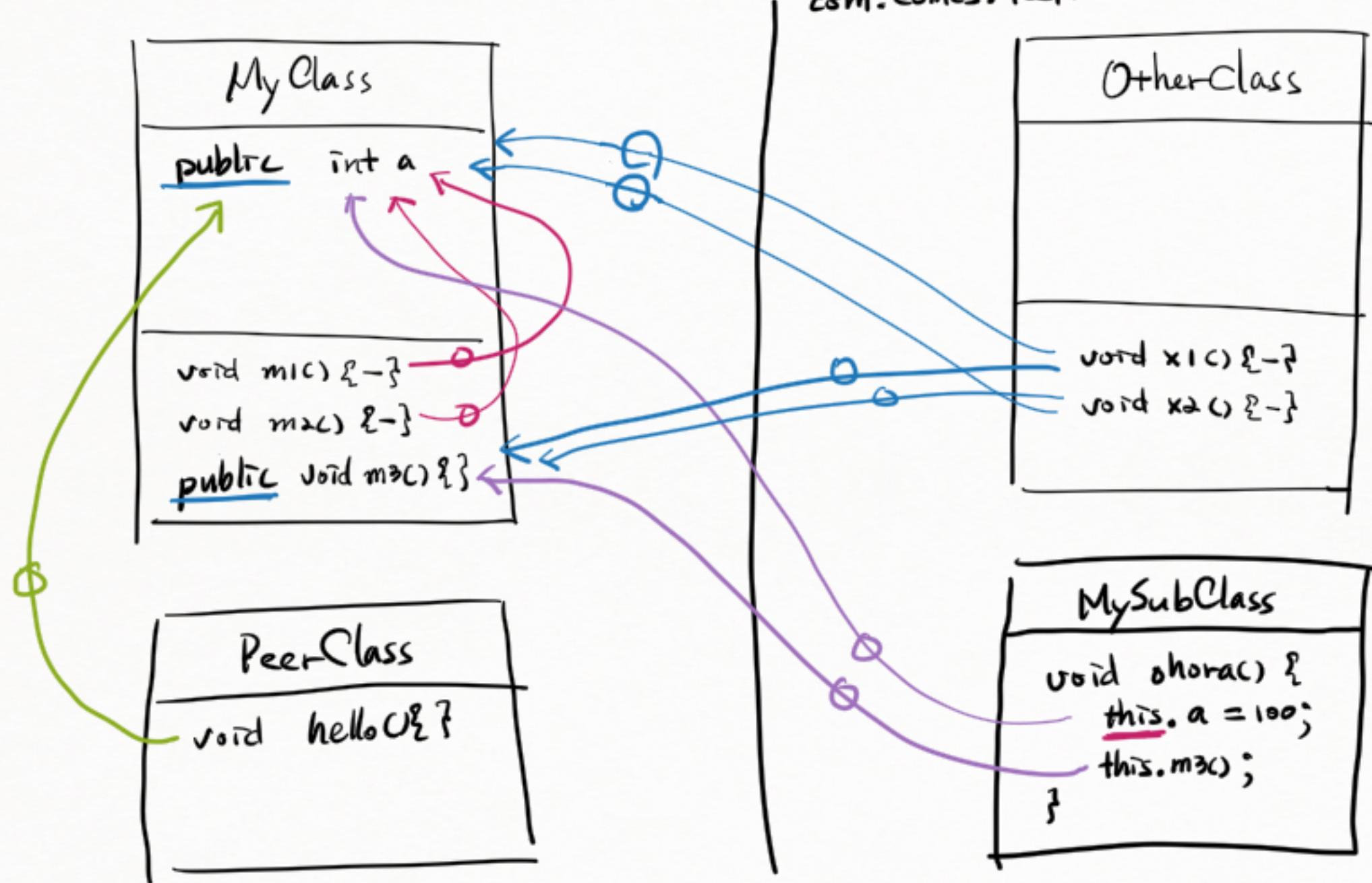
com.eomcs.test2

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

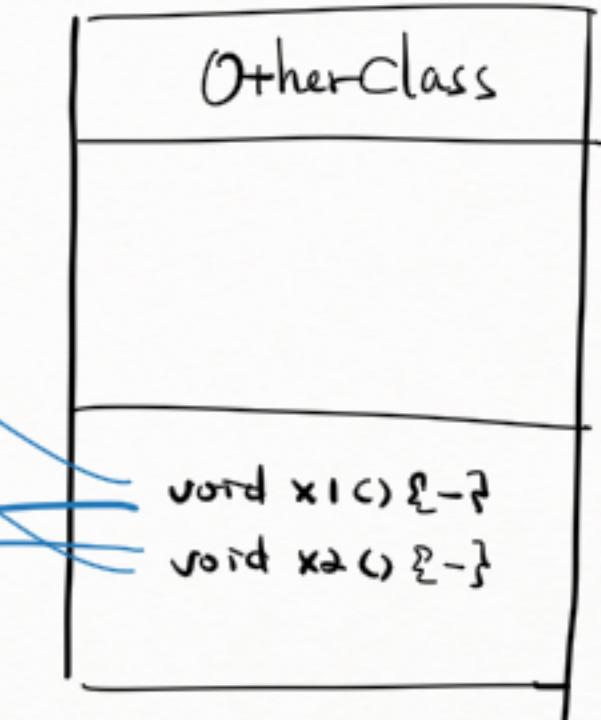


* public - 모든 멤버 접근 가능

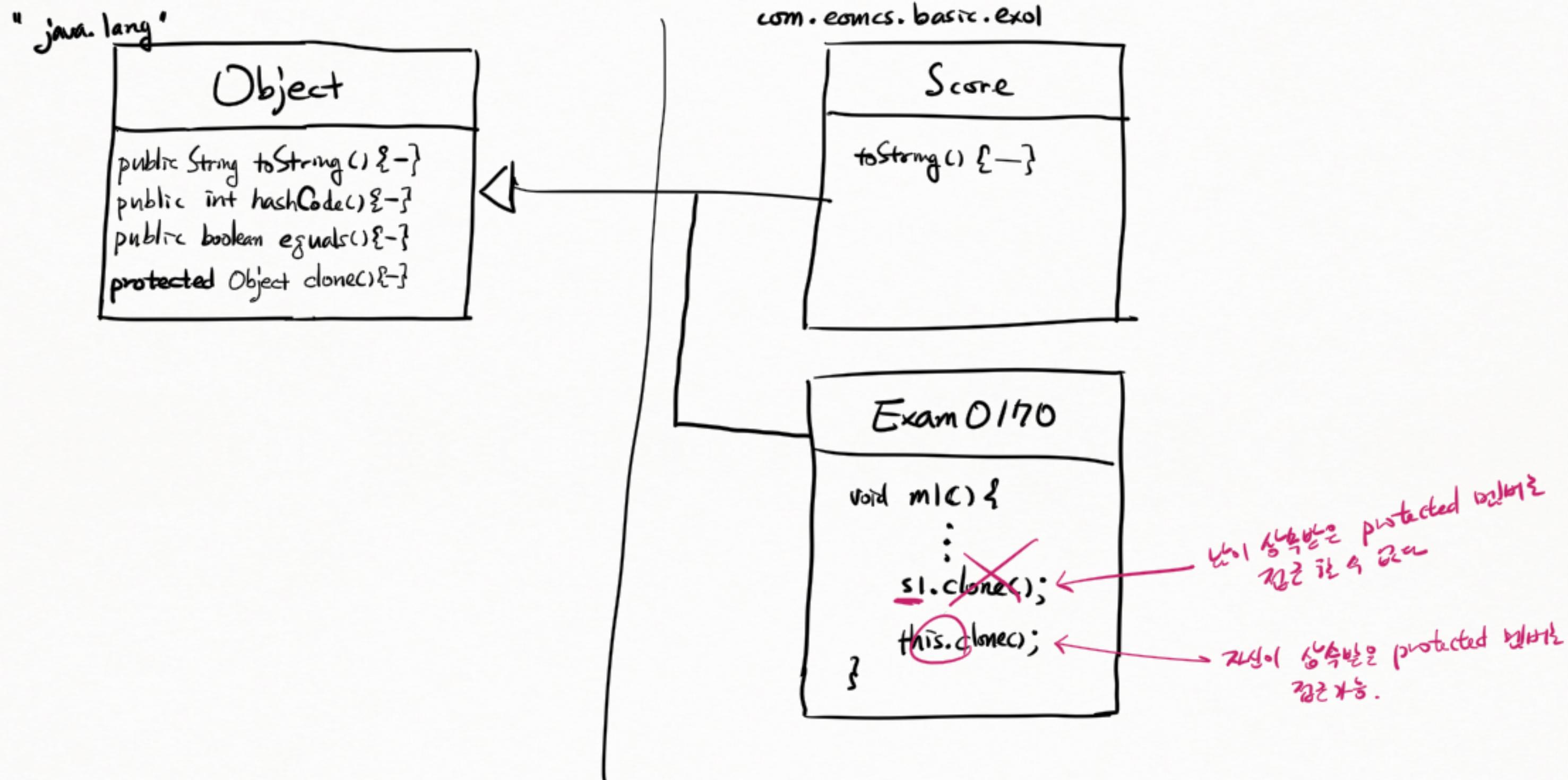
com.eomcs.test1



com.eomcs.test2

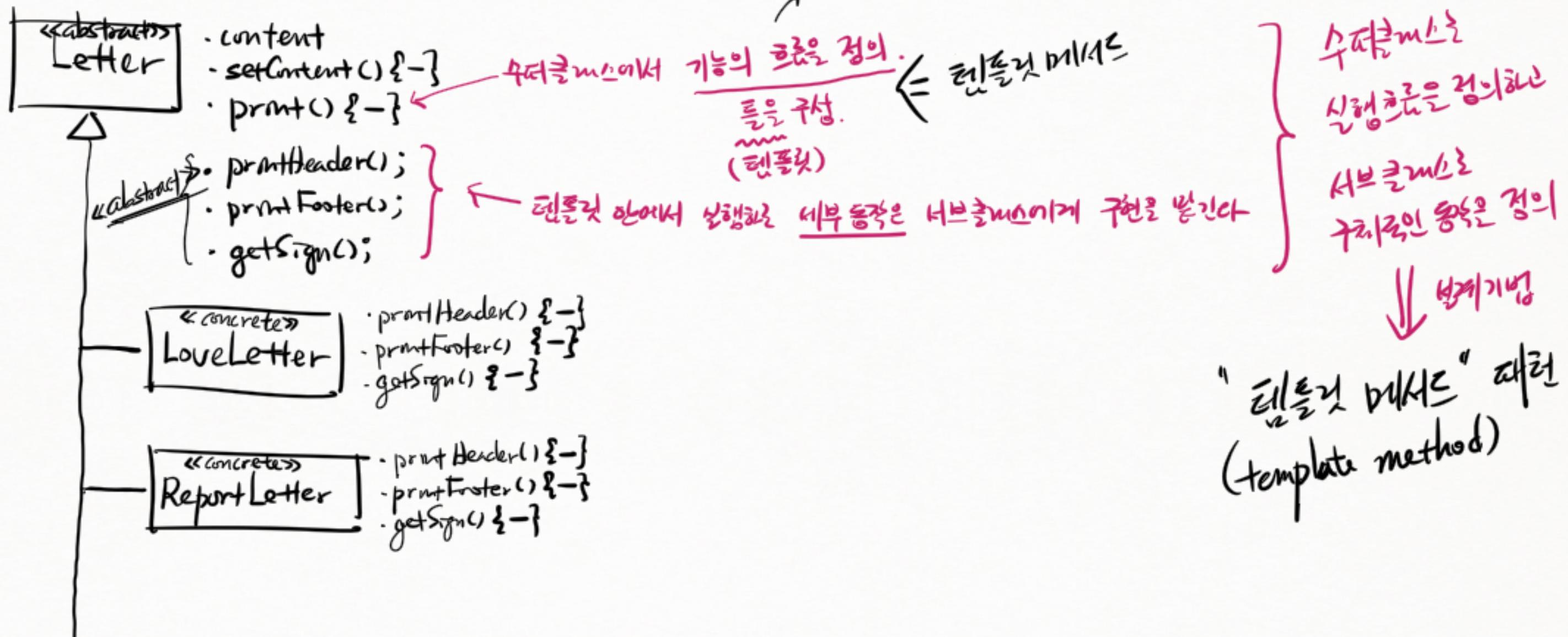


* clone 데일



* 주제 퀴즈

the skeleton of an algorithm



* 추상 클래스와 추상 메서드의 활용

① 각각 단일의 클래스 사용



• run()

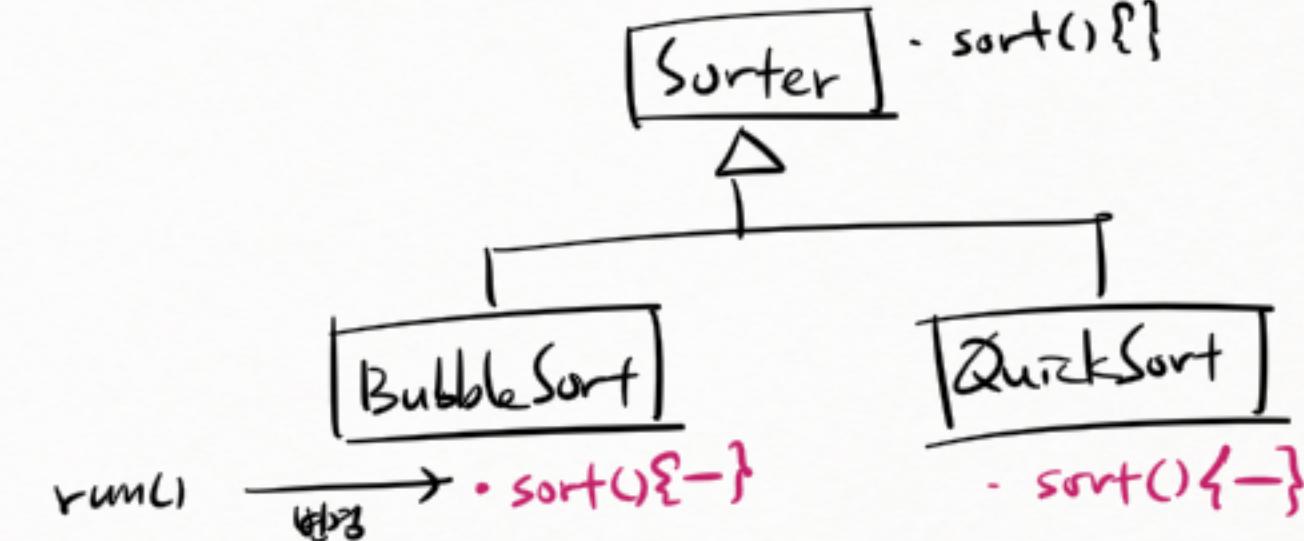
\uparrow
두 개의 정렬 클래스를 같은 부류가 아니기 때문에
한 개의 print() 메서드를 사용할 수 없다.
모든 내용은 각자.

• display(BubbleSort, int[]) { }

• display(QuickSort, int[]) { }

개선

② 같은 단일의 클래스로 보기



run()

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

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

같은 단일화된
print()를 두 클래스를 위한 대신

• display(Surter, int[]) { }

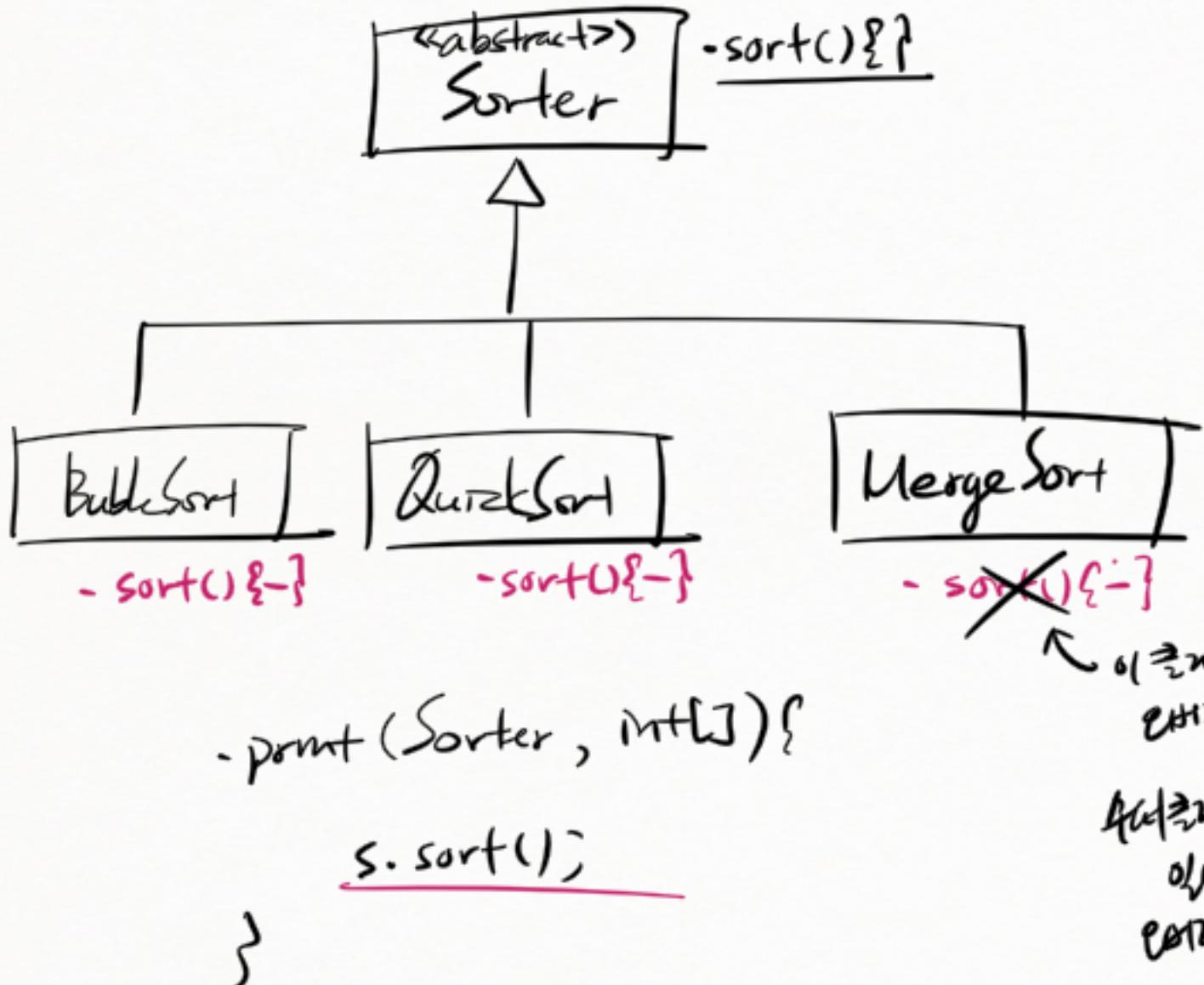
s.sort();

}

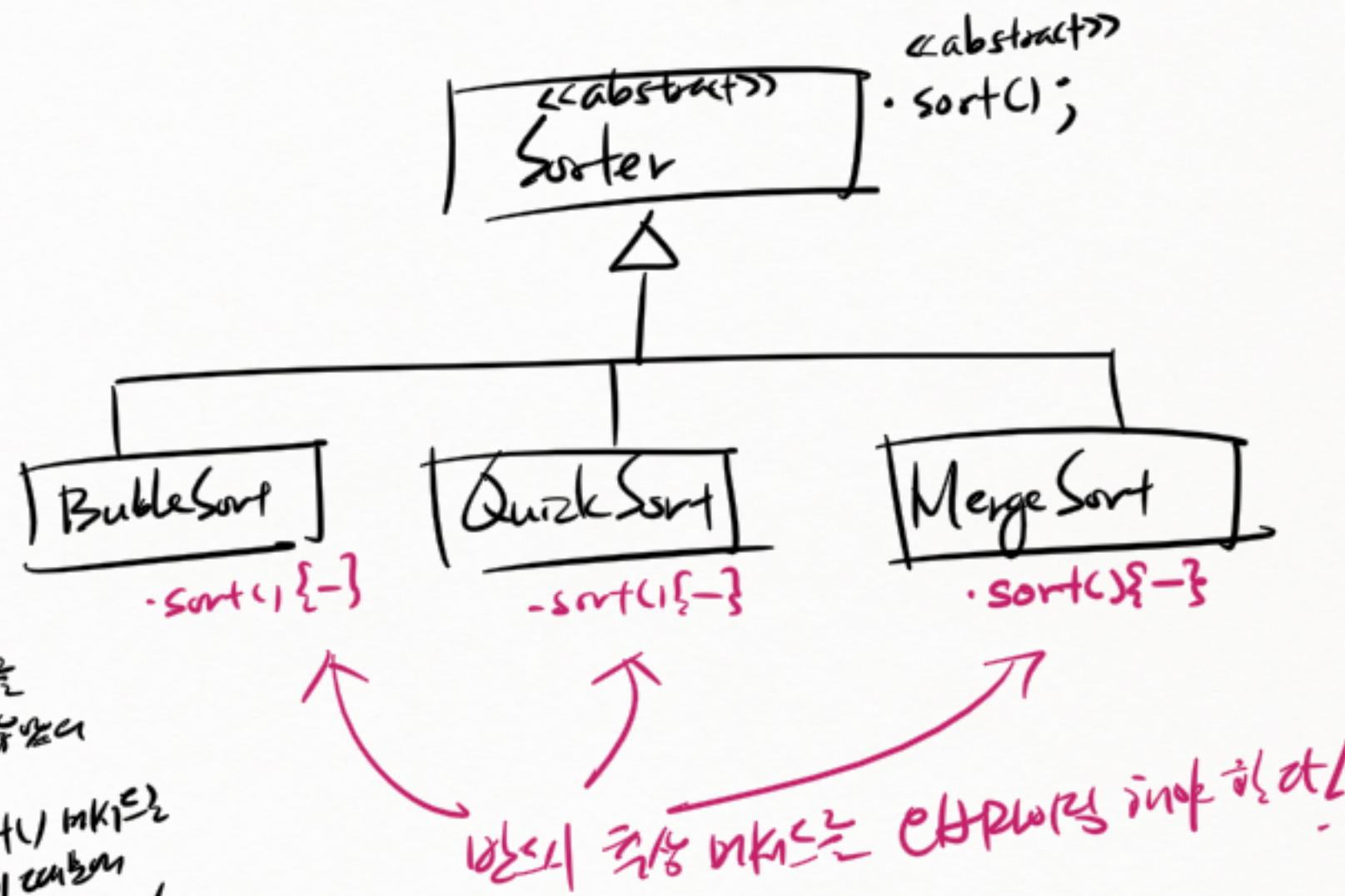
클래스에 상관없이
같은 이름의 메서드 호출
"이름의 사용 가능"

* 추상클래스와 추상메서드의 활용

③ 추상클래스로 추상클래스로 나누기

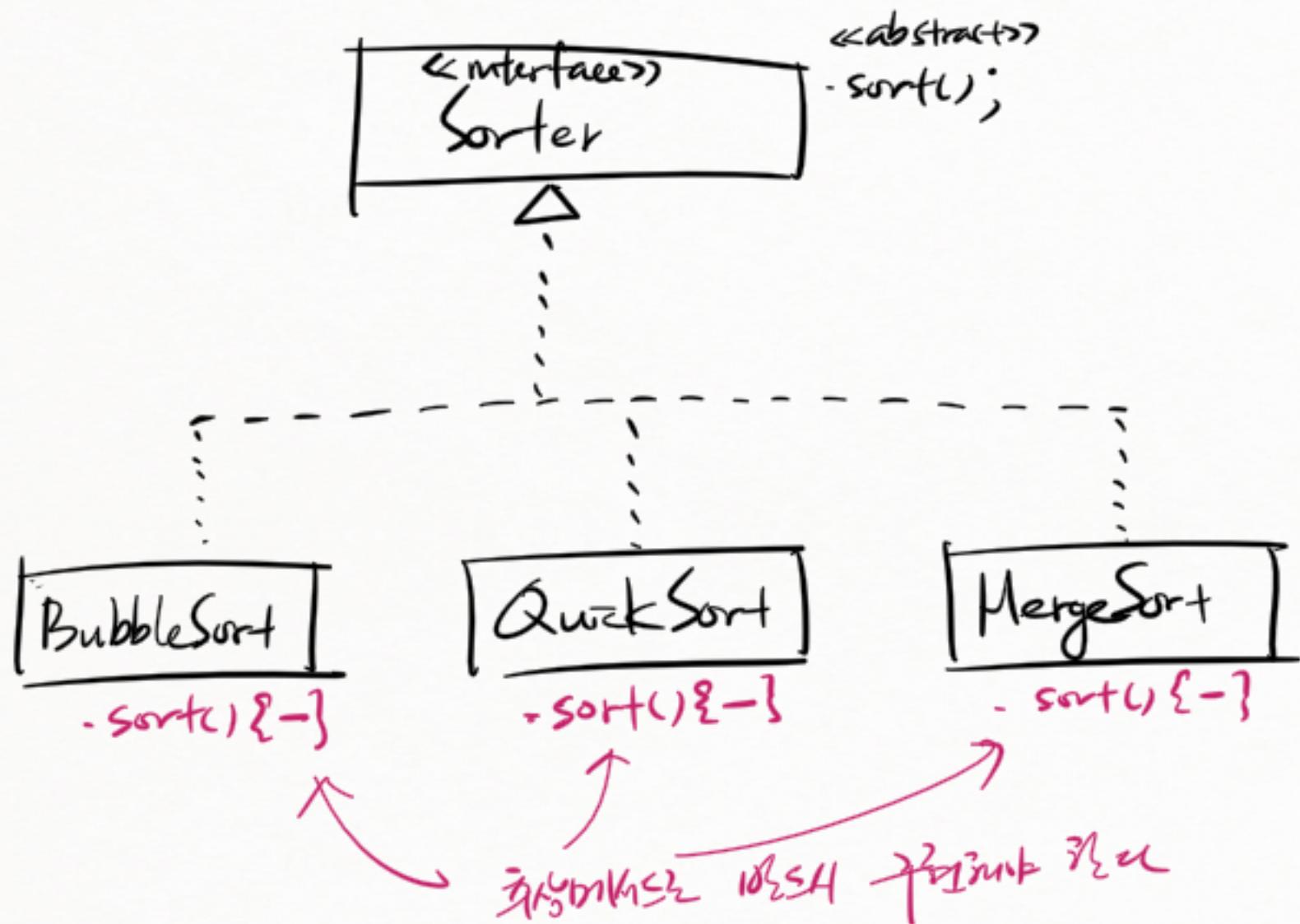


④ 추상메서드로 분리하는 것

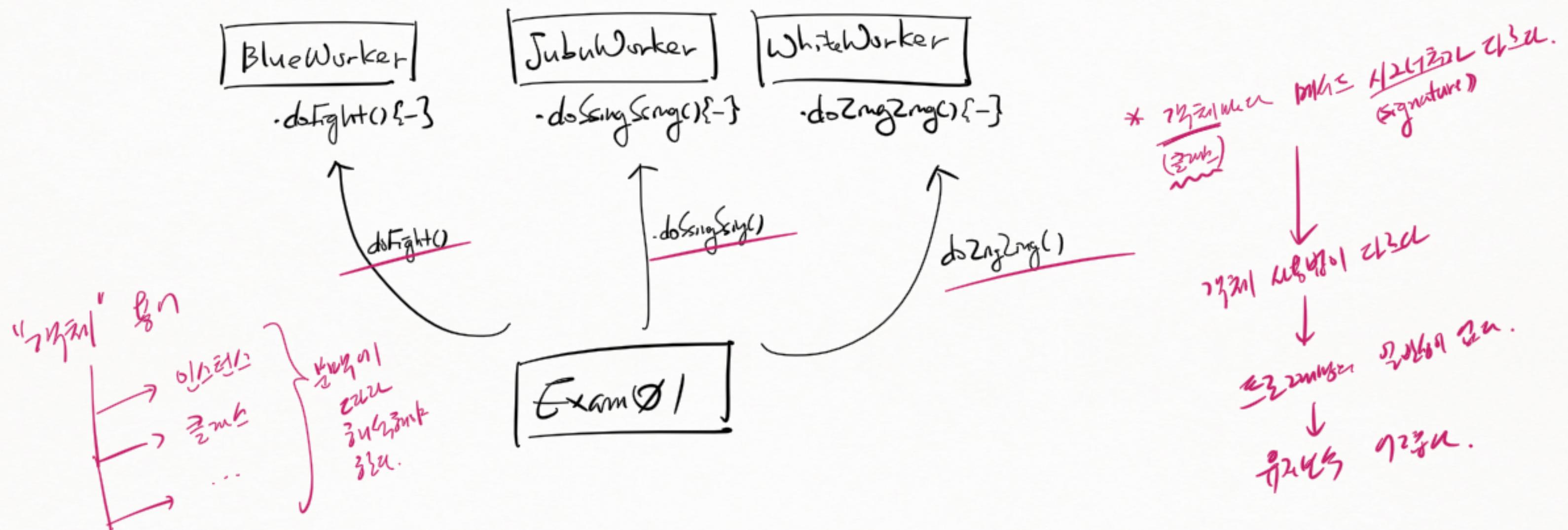


* 추상클래스 만든 인터페이스를 끌 때

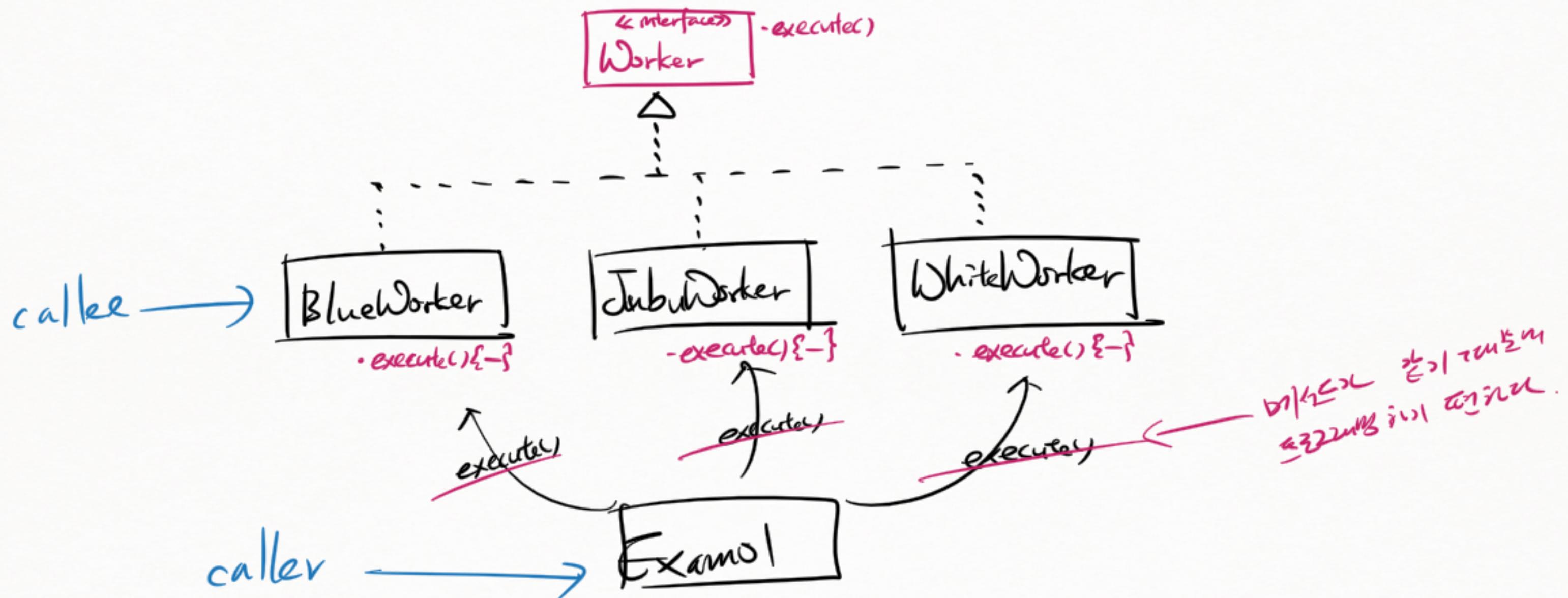
⑤ 추상클래스 만든 인터페이스로 만들기



* 인터페이스 사용 : - oop. ex9. al. before



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

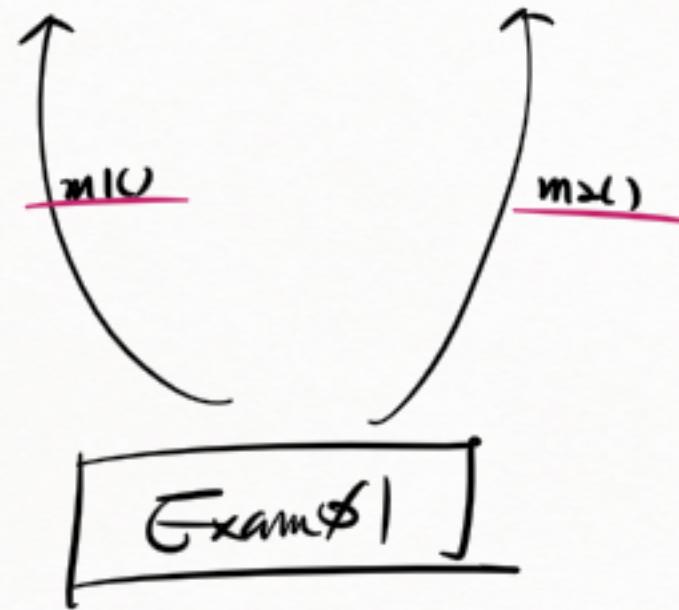
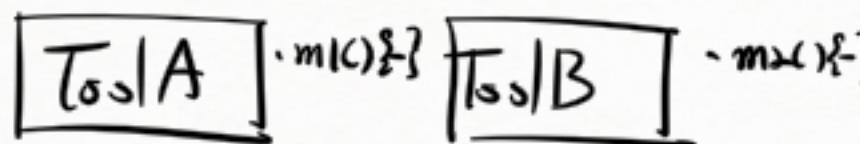


* 인터페이스 사용 전 / 후

① 사용전

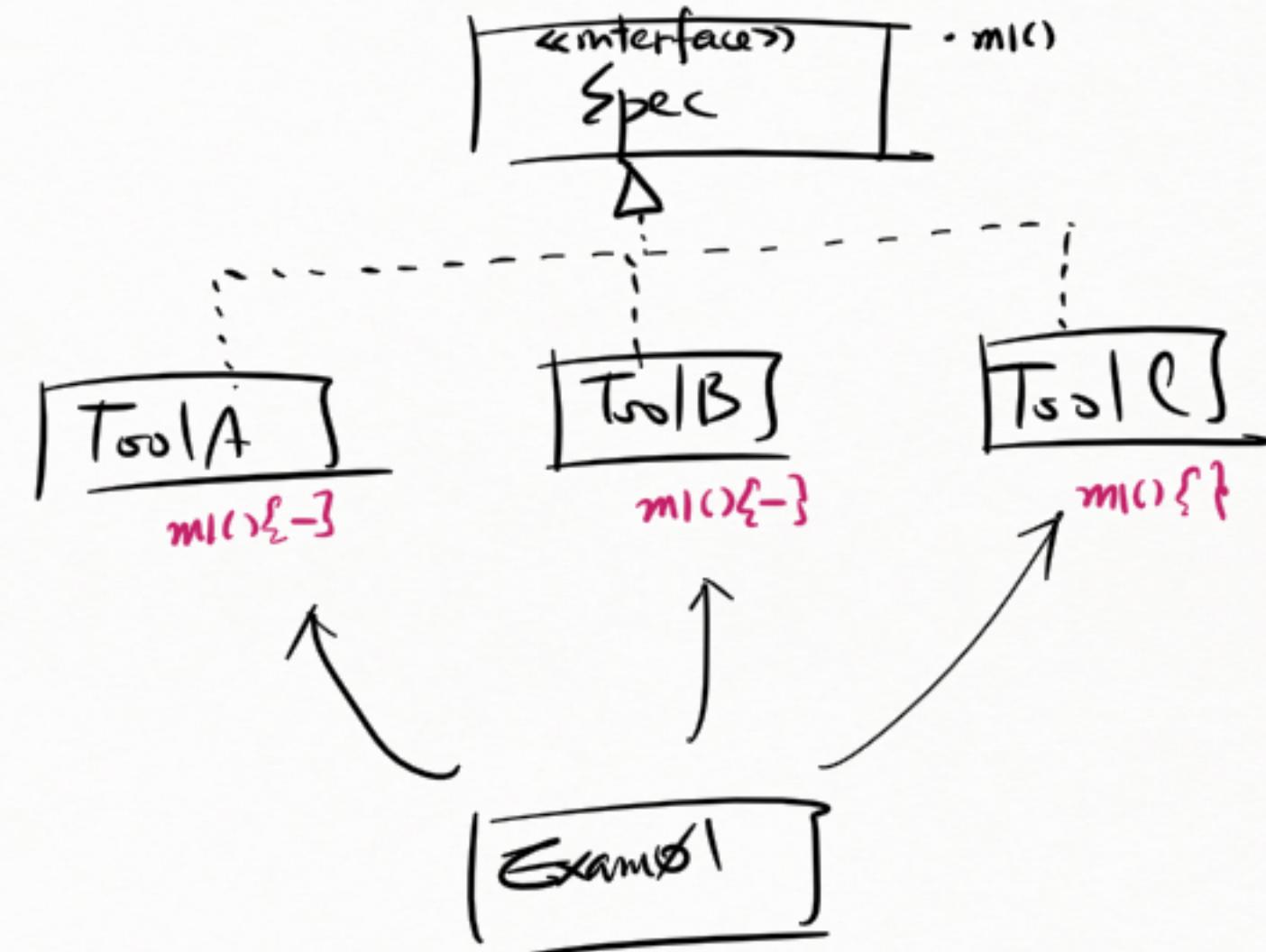
개선

② 사용 후

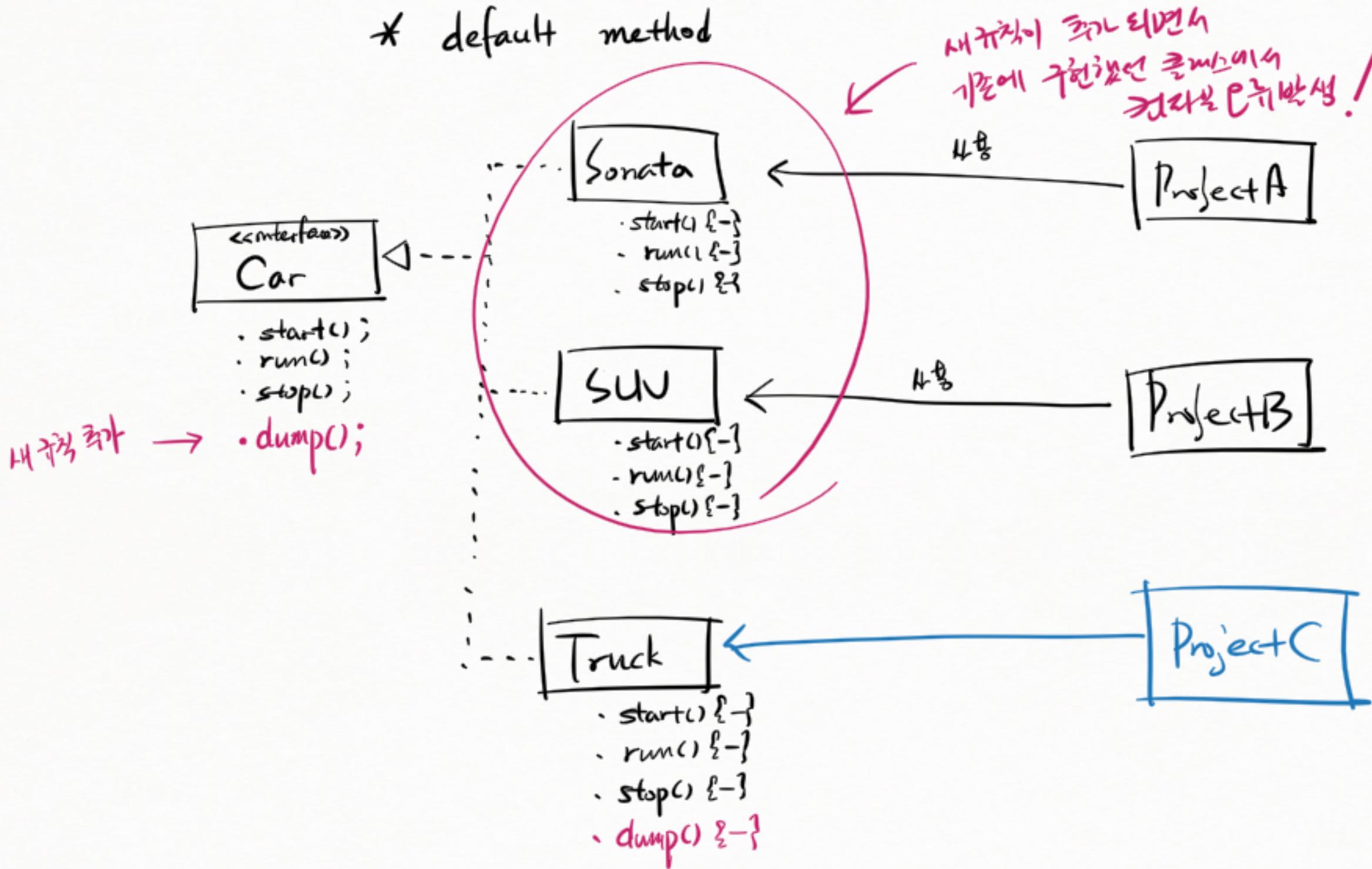


단점
m1(), m2()가 같은 이름

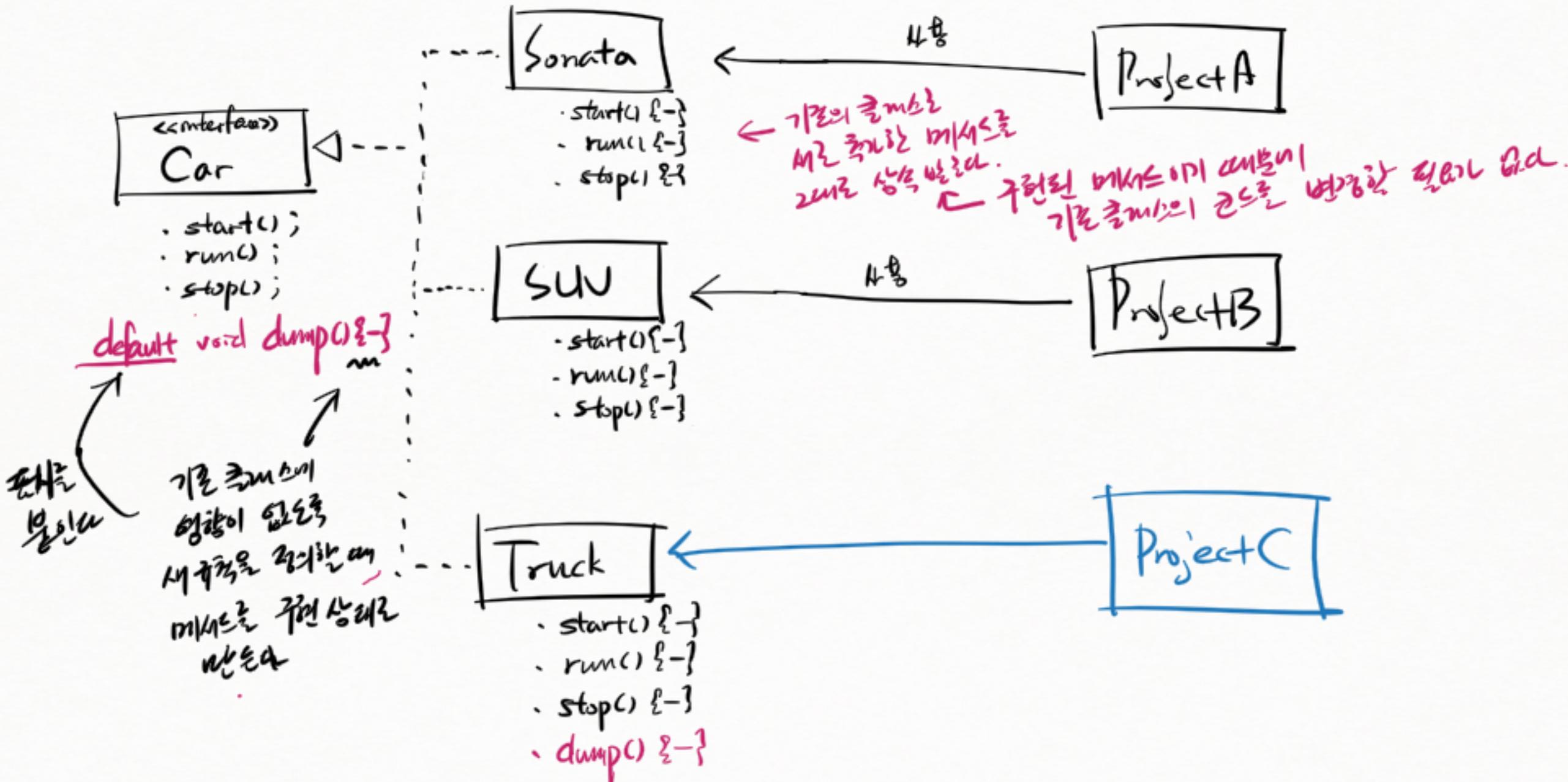
단점
유저에게 혼동.



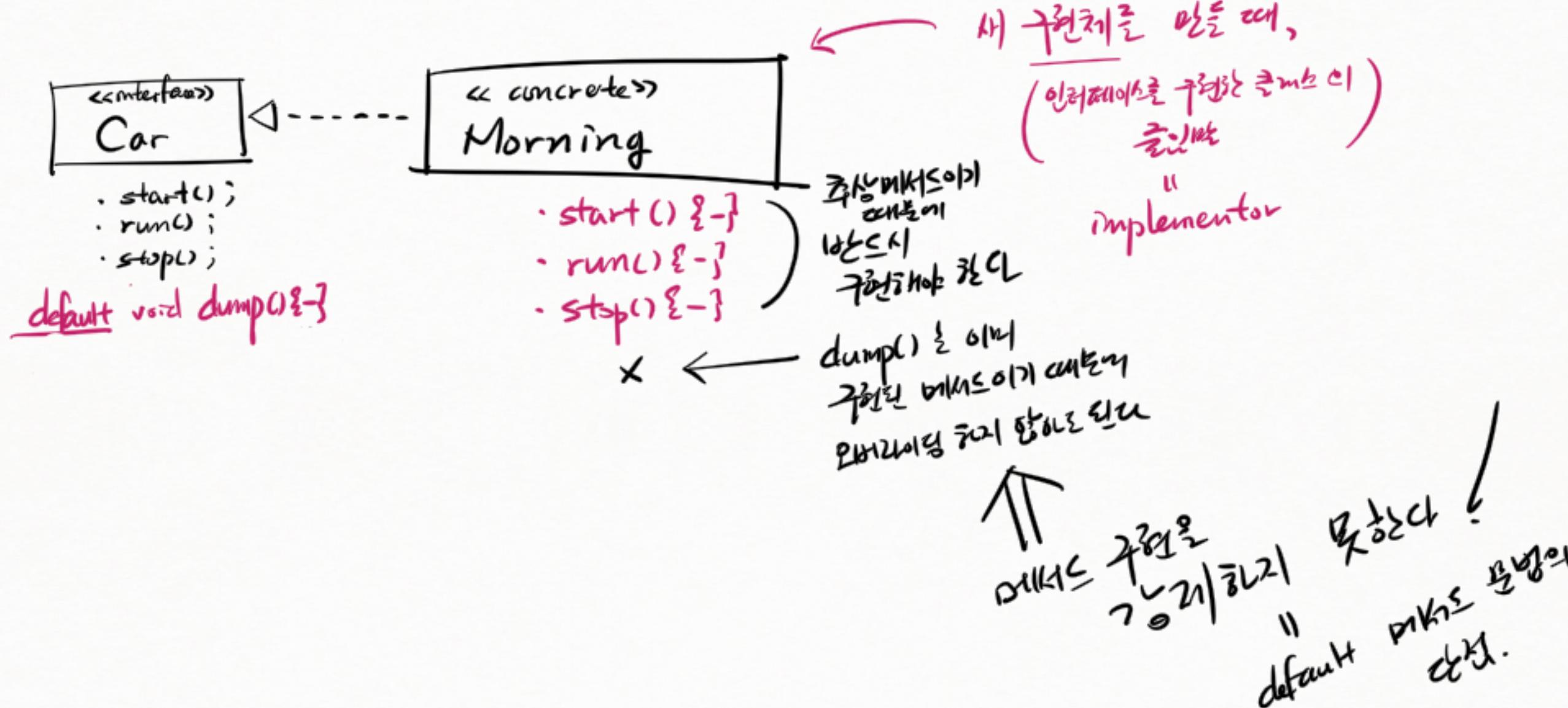
* default method



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



* default method o/a 문제점



* super el. 인터페이스.super

