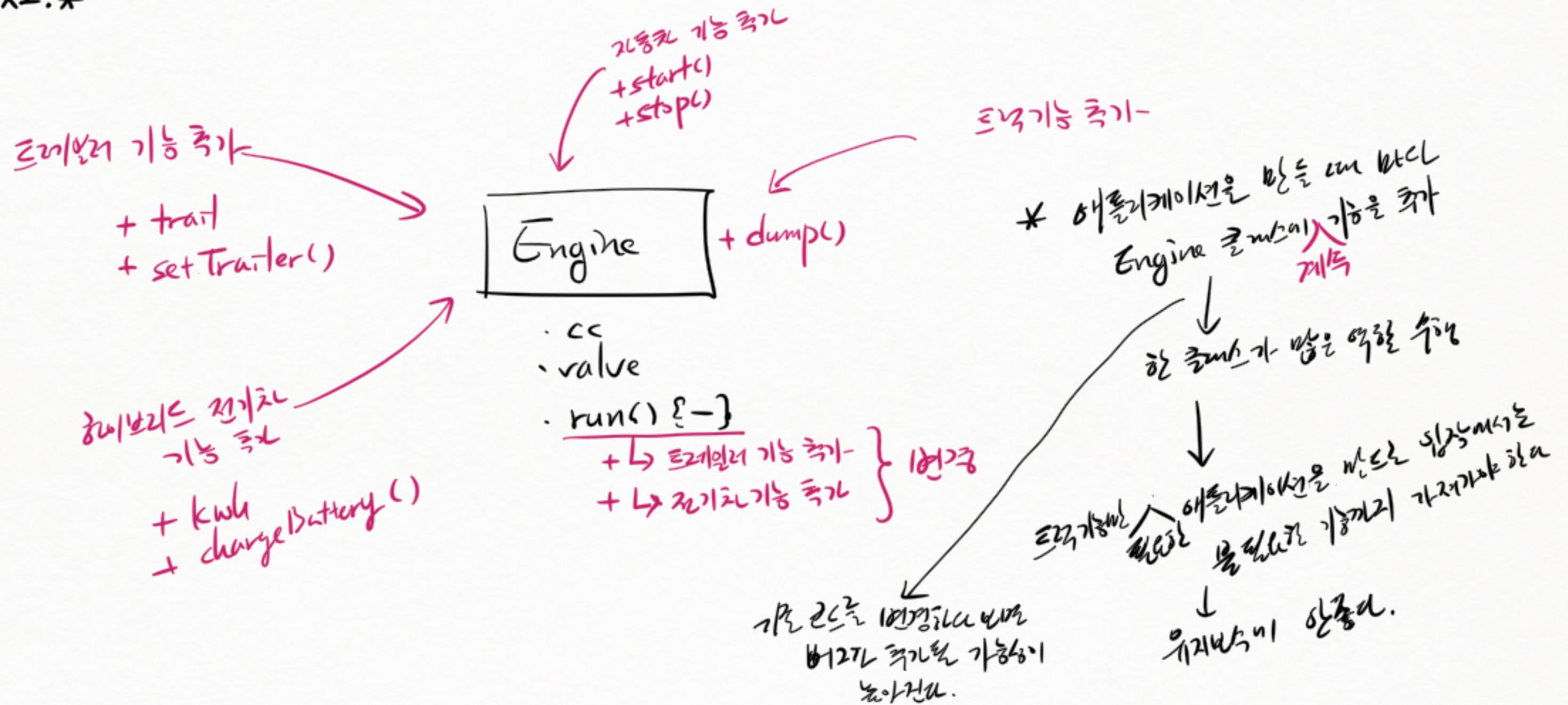


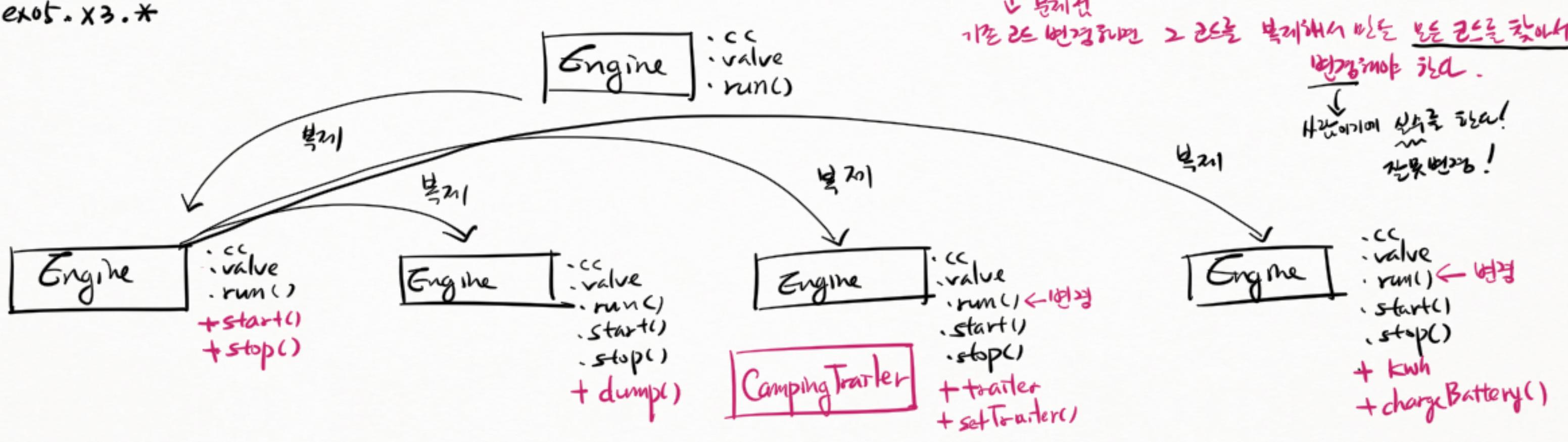
\* 가능학습 방법 | - 가능학습 방법 | 가능학습 방법 |

00P.0X05.X2.\*



## \* 기능 확장 방법 2 - 복제한 코드에 새 기능 추가

oop.ex05.\*



① 자동차 만들기  
app1

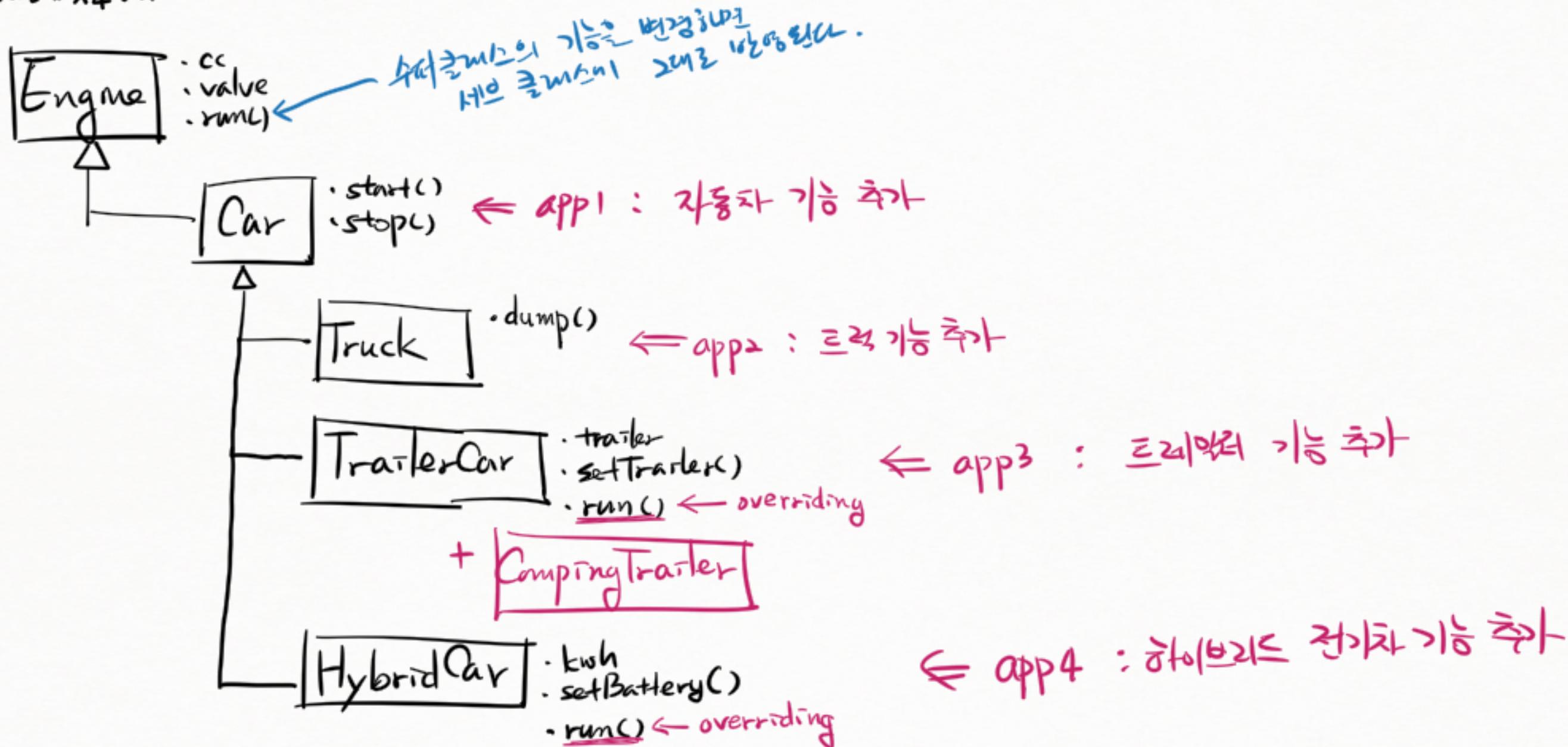
② 트레일러 만들기  
app2

③ 캠핑카 만들기  
app3

④ 차량 모터드 전기차 만들기  
app4

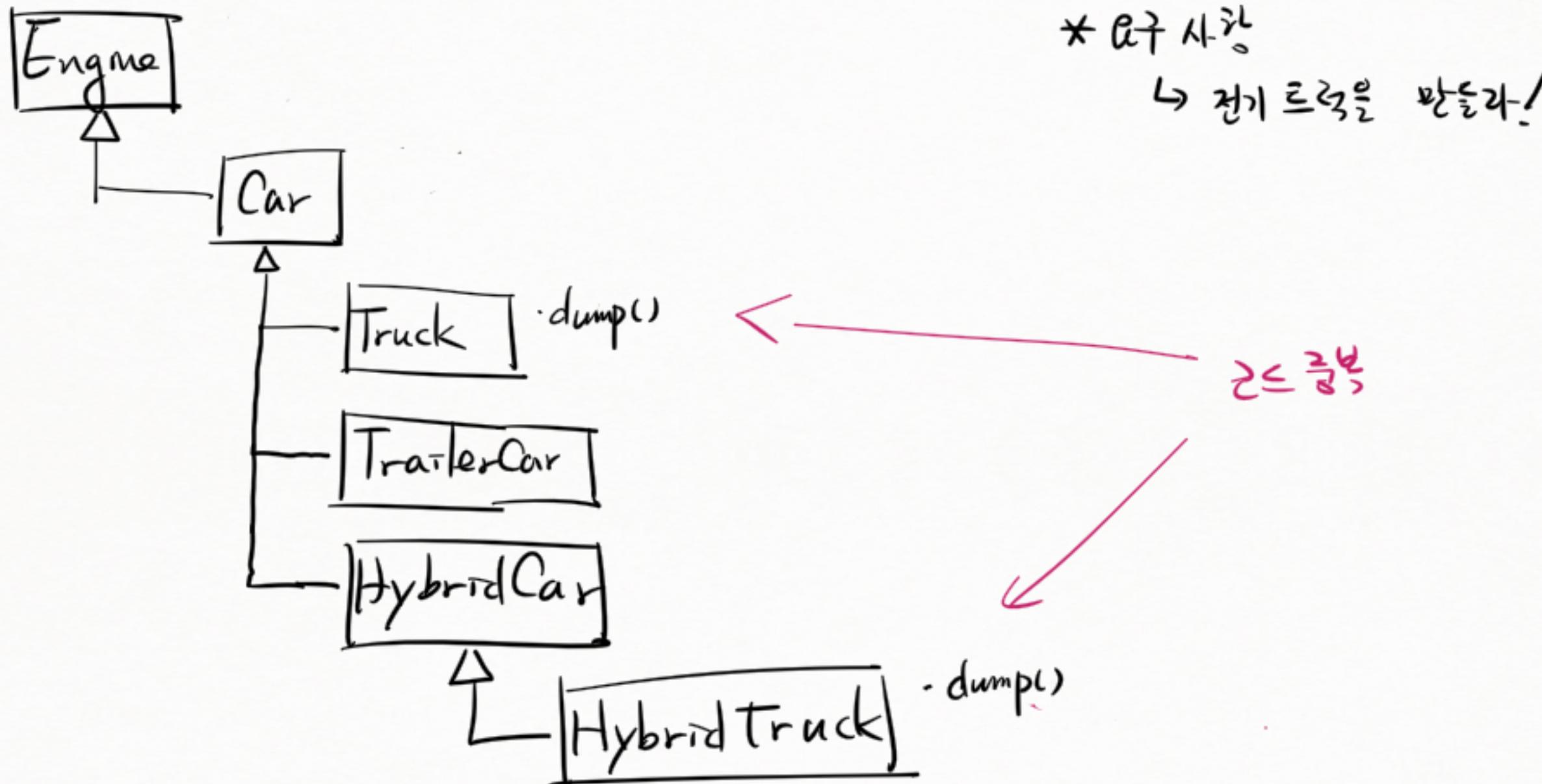
\* 기능 확장 방법 3 - 상속을 이용한 확장.

oop.ex05.x4.\*



\* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한계

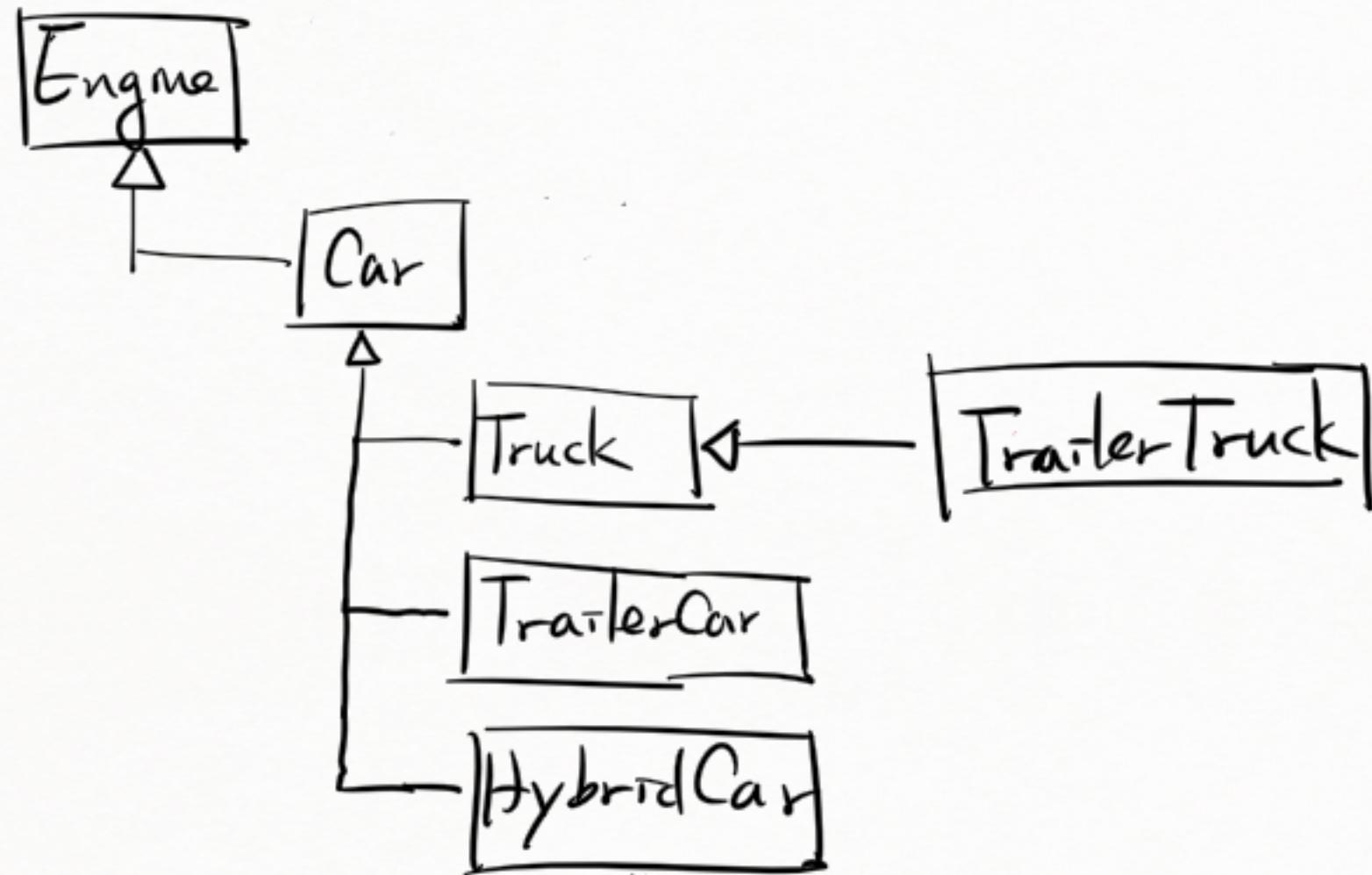
oop.ex05.x4.\*



\* 기능 확장  
↳ 전기 트럭을 만들라!

\* 기능 확장 방법 3 - 상속을 통한 기능 확장의 한가지

oop. ex05. x4.\*



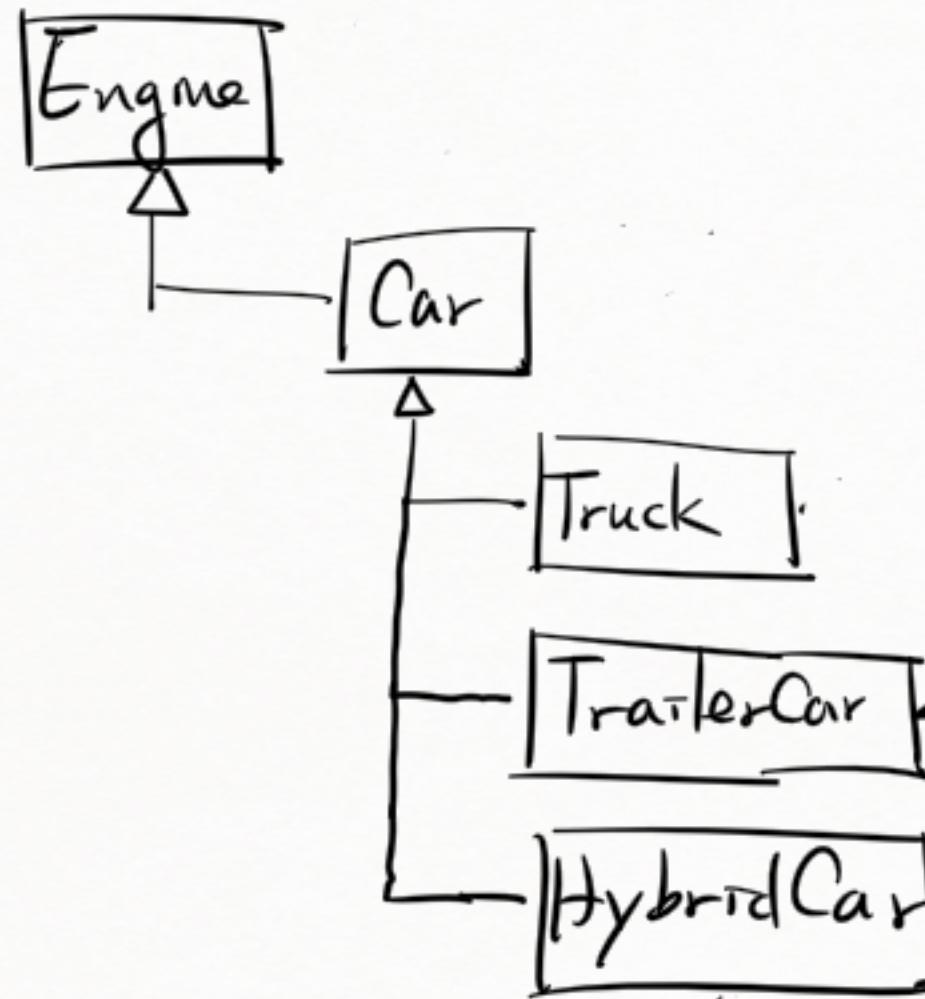
\* 추상화

↳ 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 퍼블릭 메소드가 대체된다

- kwh  
setBattery()

(상속으로 대상은 기능이 조합된  
개체로 만든다면 가능)

↳ 유지보수는 어렵지만...

\* 기능학적 특성 4 : 디자인에서 기초 기능

기능적

Sedan

Truck

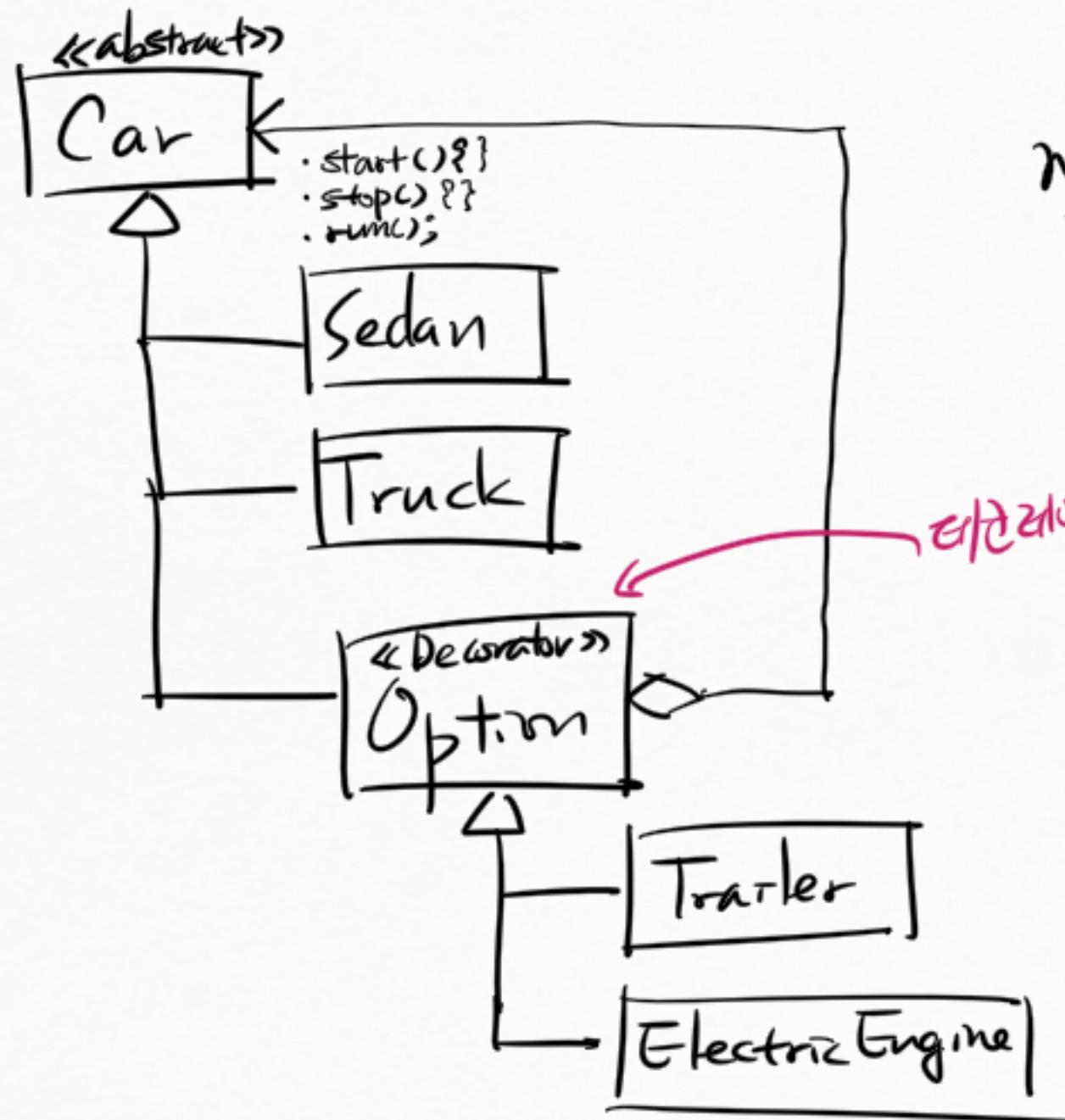
선택적

Gas Engine

Electric Engine

Trailer

\* 테러리아 러닝 기법



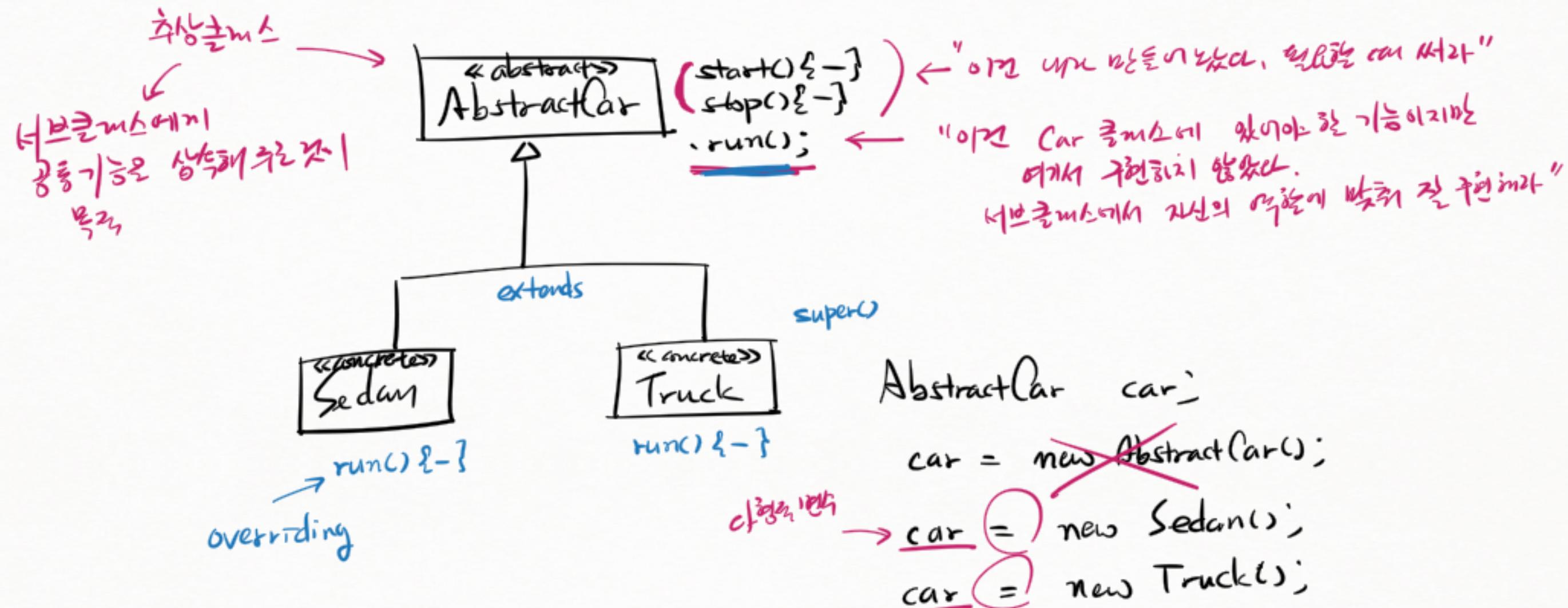
new Optim(new Sedan(new Optim(new ElectricEngine)))

new Optim(new Truck())

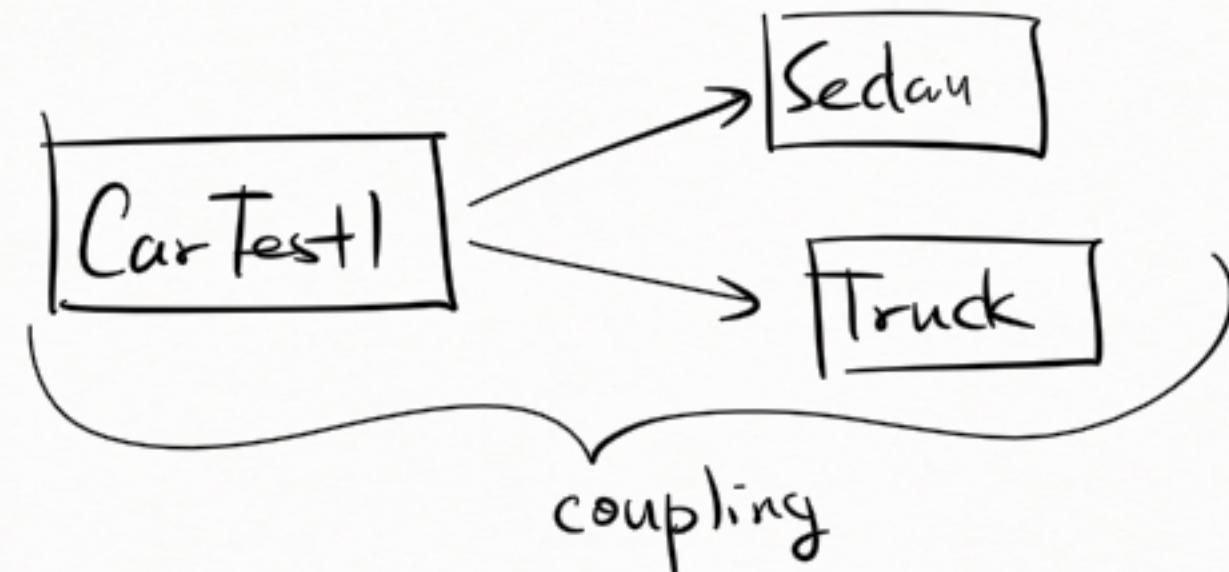
기존  
클래스  
추가  
보다.

디코레이터: 주제에  
별도로 선택 가능

## \* 추상클래스와 구현



\* 실전 프로그래밍 1 단계



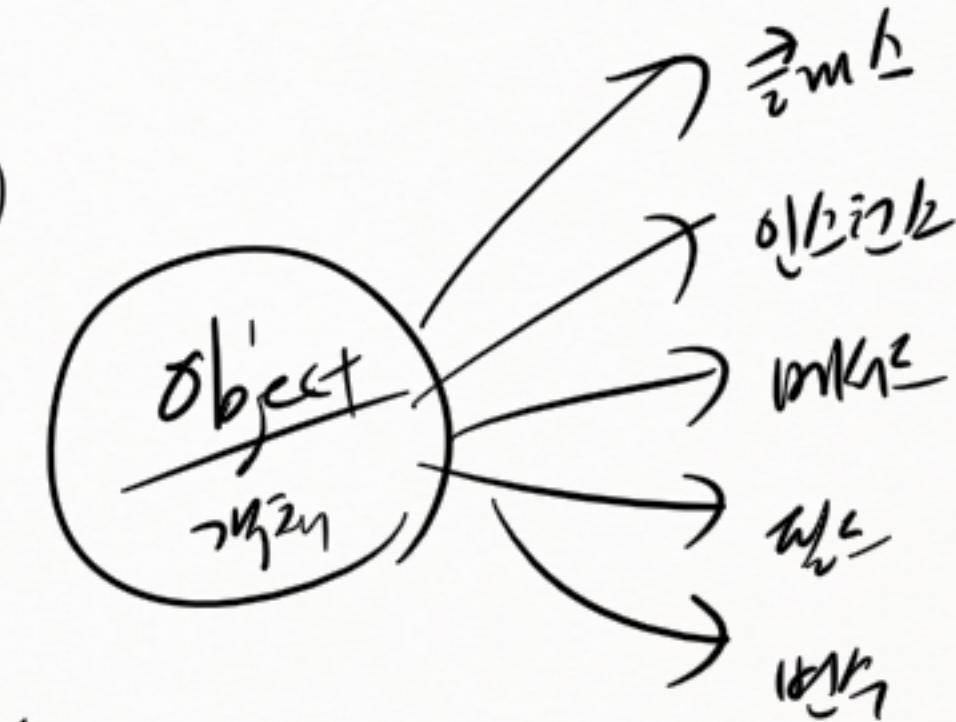
coupling

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

**Sedan**

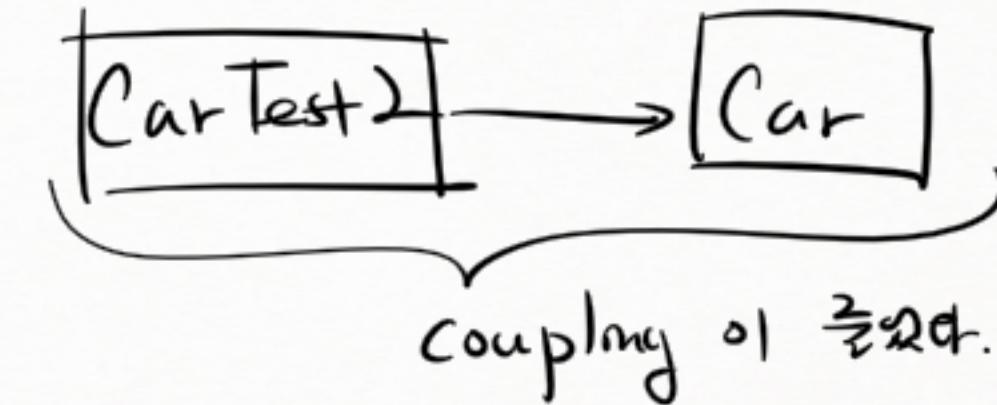
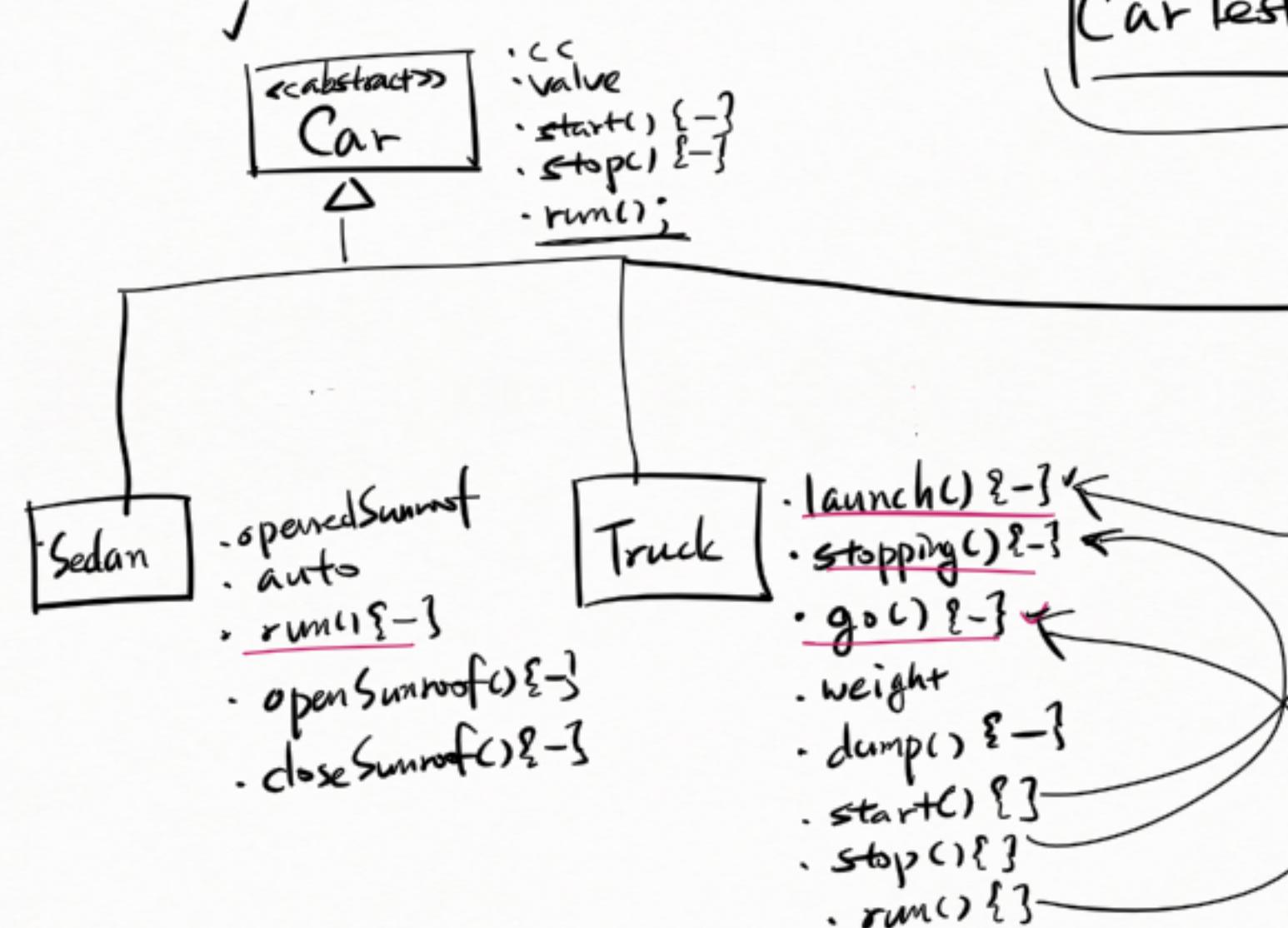
**Truck**

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



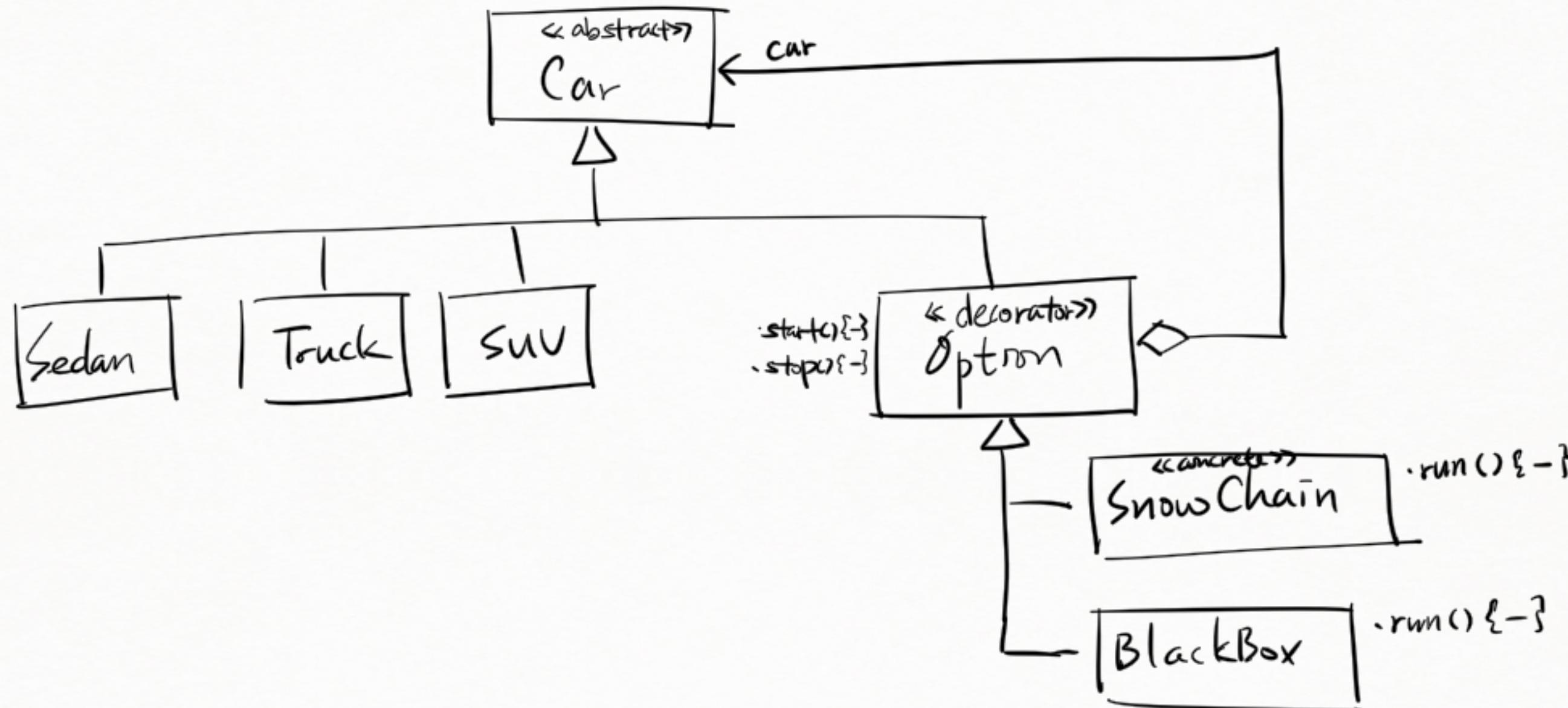
\* 상 하 한 한 한 한 한 - 품종, 품질, 성능

$\hookrightarrow$ : generalization

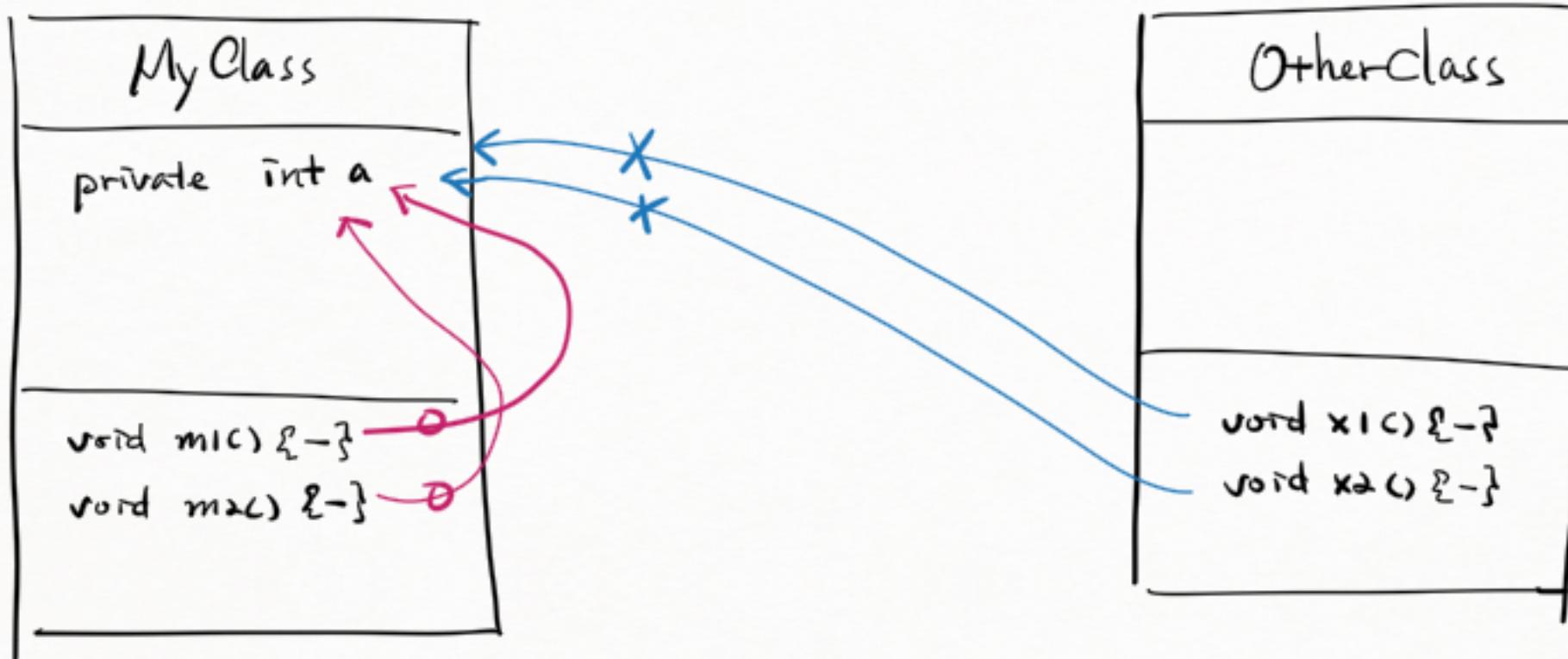


- `enabledTow: boolean`  
 - `activateTow(boolean)`  
 - `run() {}`

\* 차선을 3개로 나누면 차량이 1개로 통합

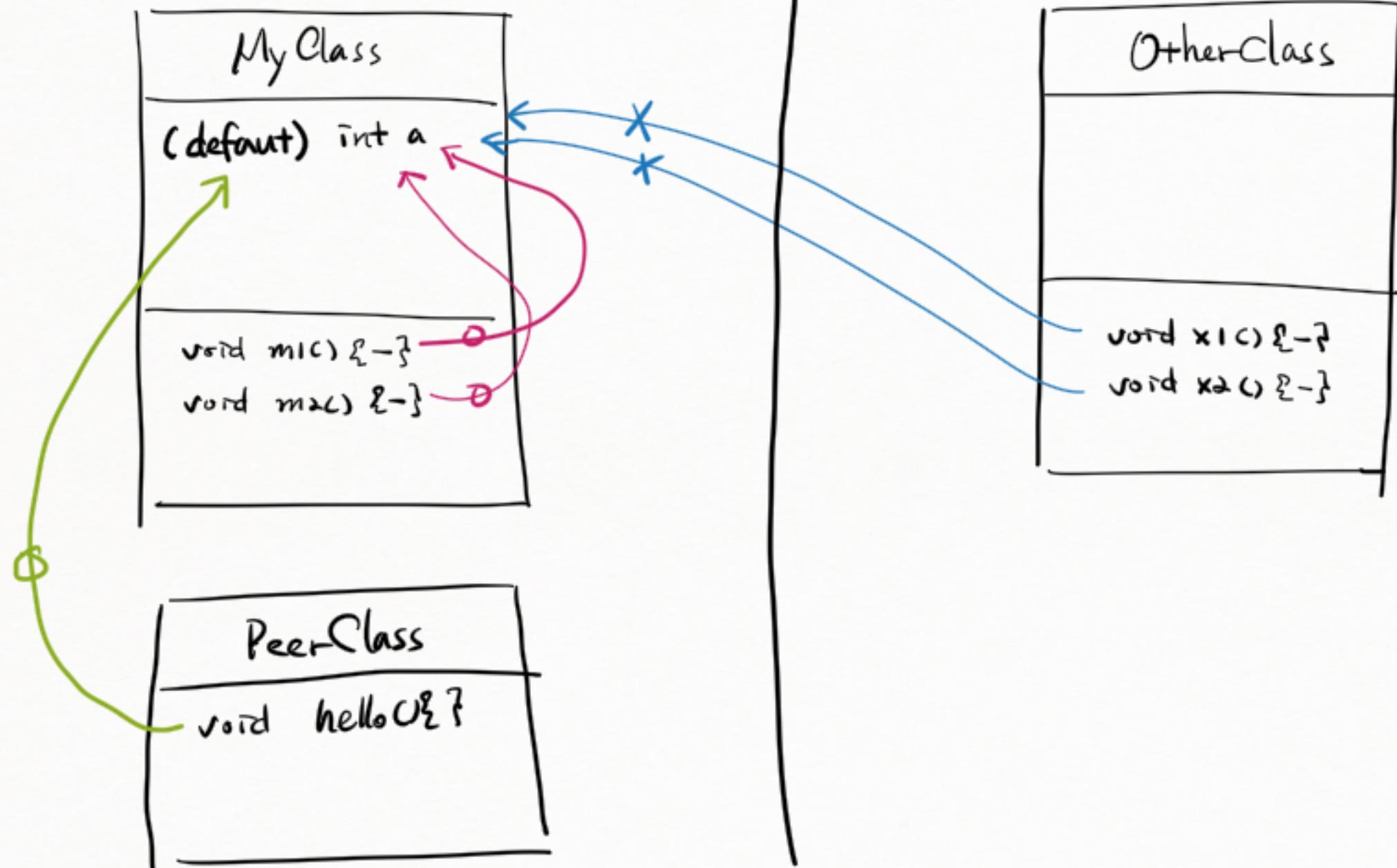


\* private - 같은 클래스의 멤버만 접근 가능



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

com.eomcs.test1



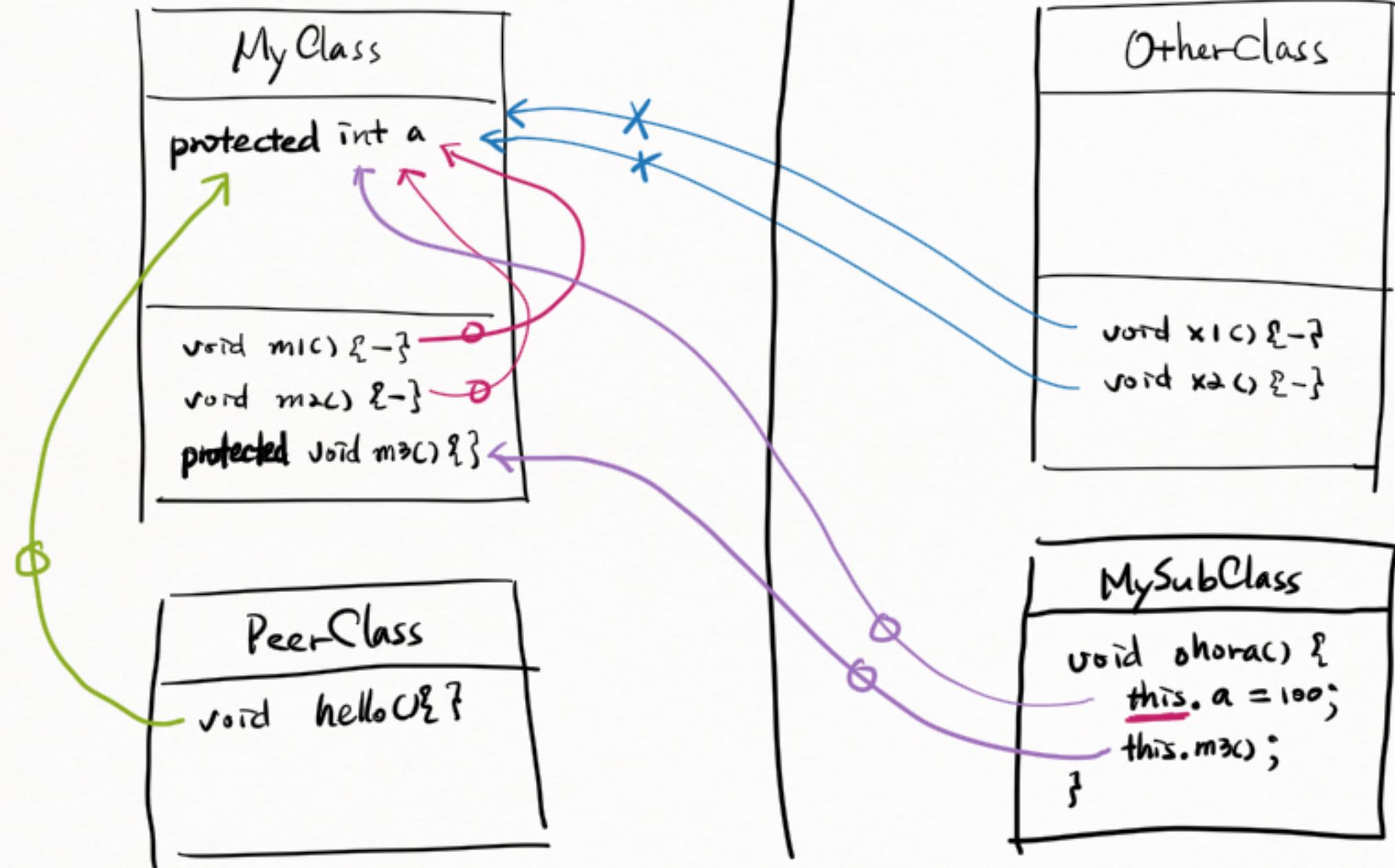
\* protected - 같은 패키지 내의 멤버 접근 가능

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

com.eomcs.test1

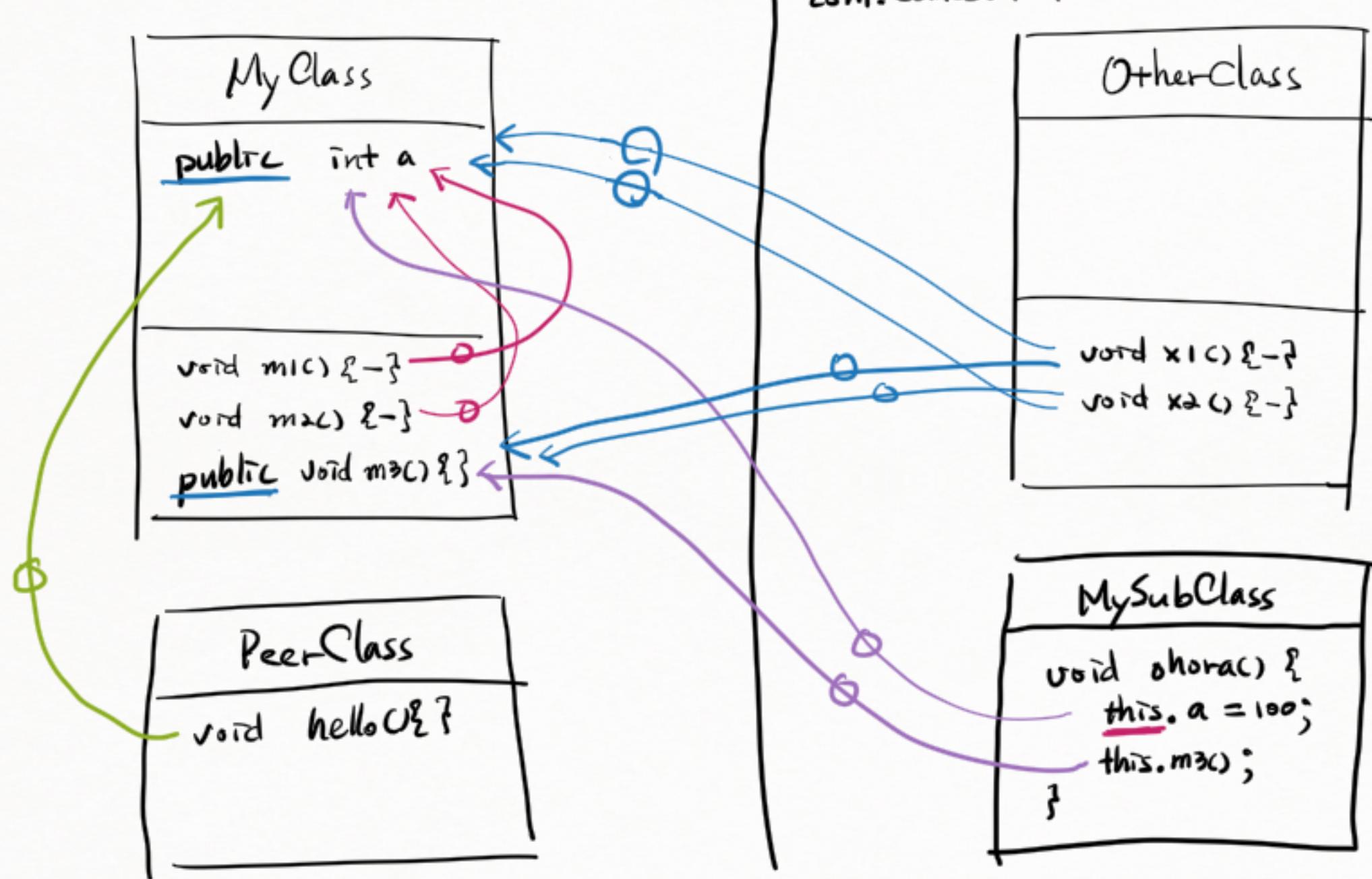
com.eomcs.test2

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

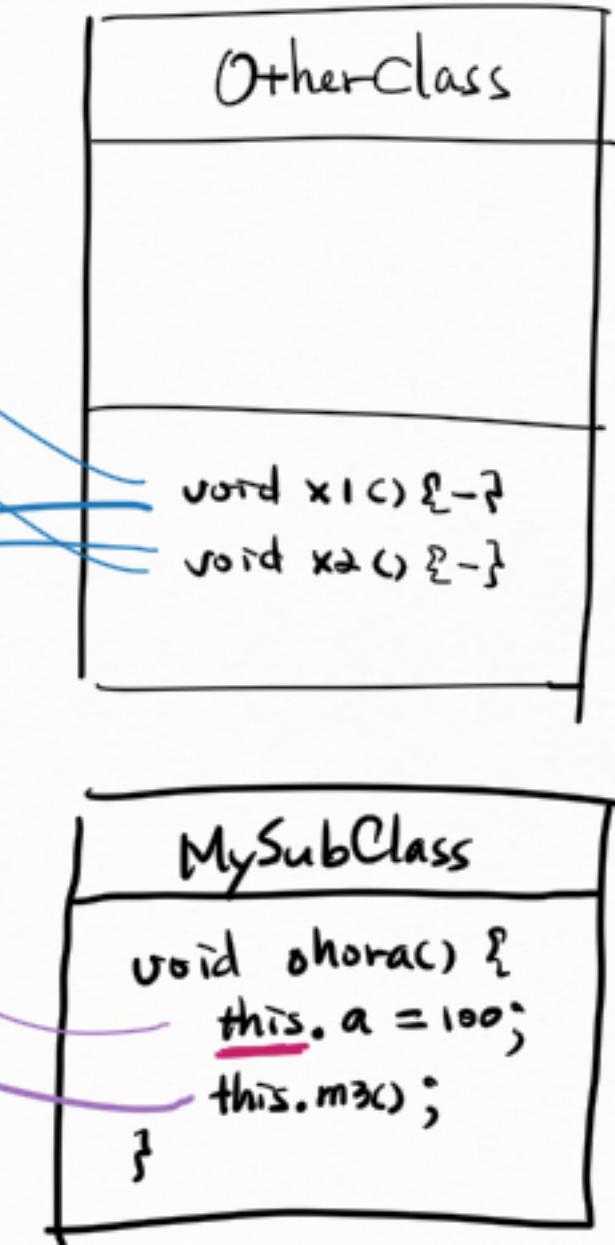


\* public - 모든 멤버 접근 가능

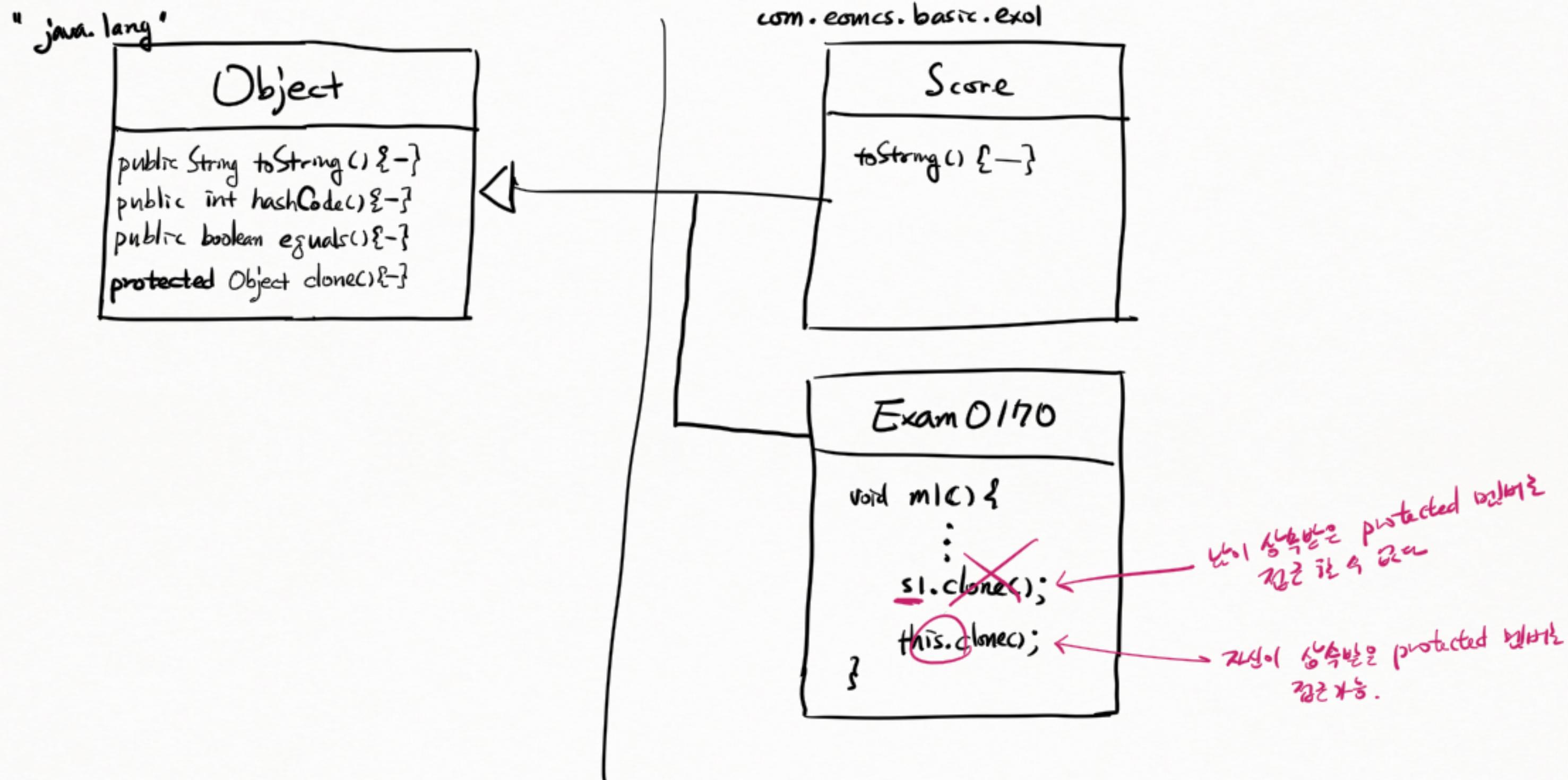
com.eomcs.test1



com.eomcs.test2

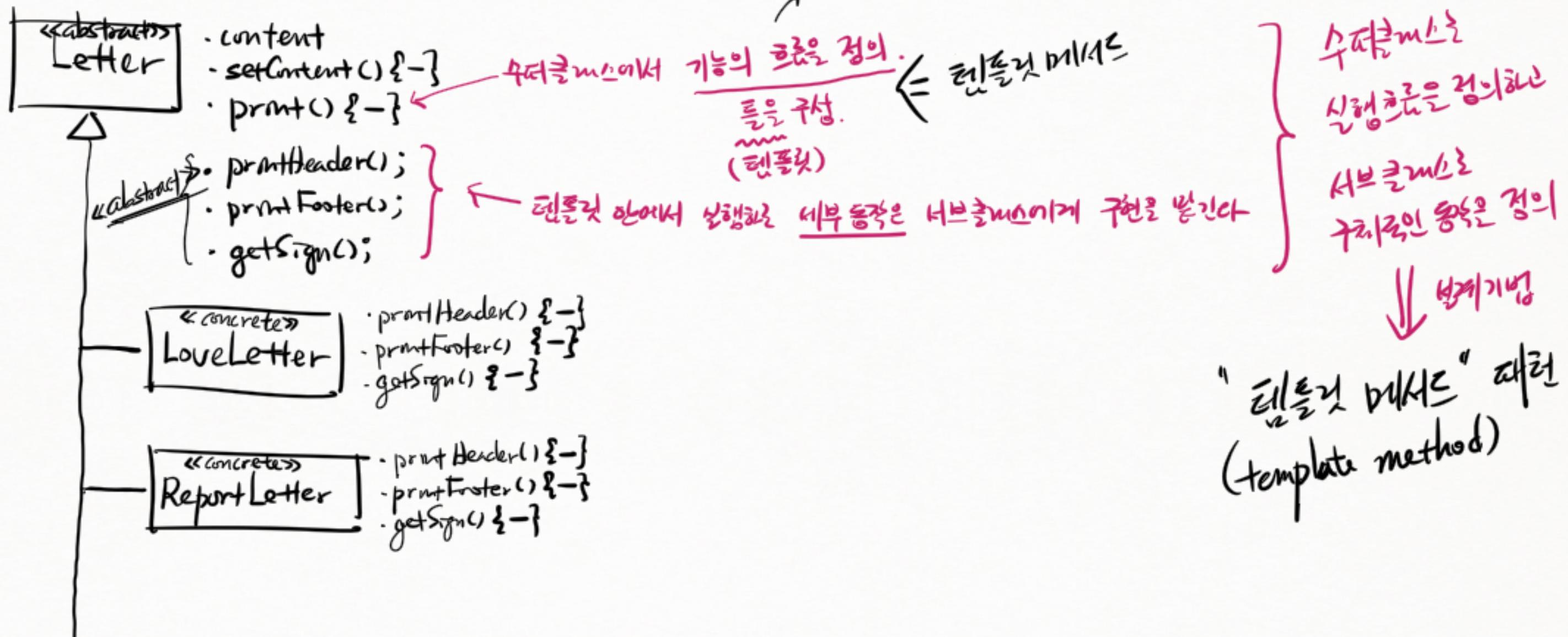


## \* clone 데일



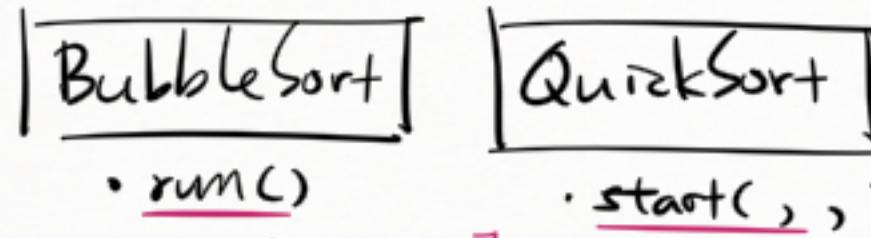
## \* 주제 퀴즈

the skeleton of an algorithm



## \* 추상 클래스와 추상 메서드의 활용

① 각각 알고리즘의 클래스 만들기



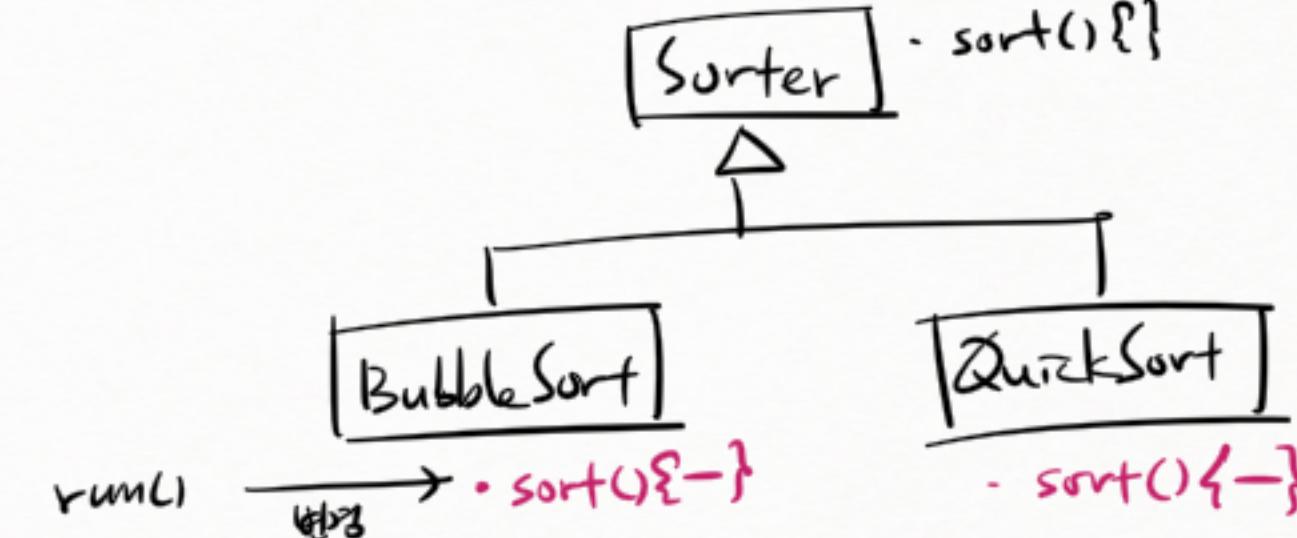
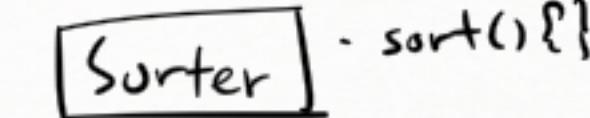
• run()

$\uparrow$   
두 개의 정렬 클래스를 같은 부류가 아니기 때문에  
같은 `print()` 메서드를 사용할 수 없다.  
모든 알고리즘은 `area`.

- `display(BubbleSort, int[]){ }`
- `display(Quicksort, int[]){ }`

개선

→ ② 같은 알고리즘 클래스로 보기



run()

• `sort(){ }`

• `sort(){ }`

같은 알고리즘이나  
`print()`를 두 클래스는 같은 대.

• `display(Sorter, int[]){ }`

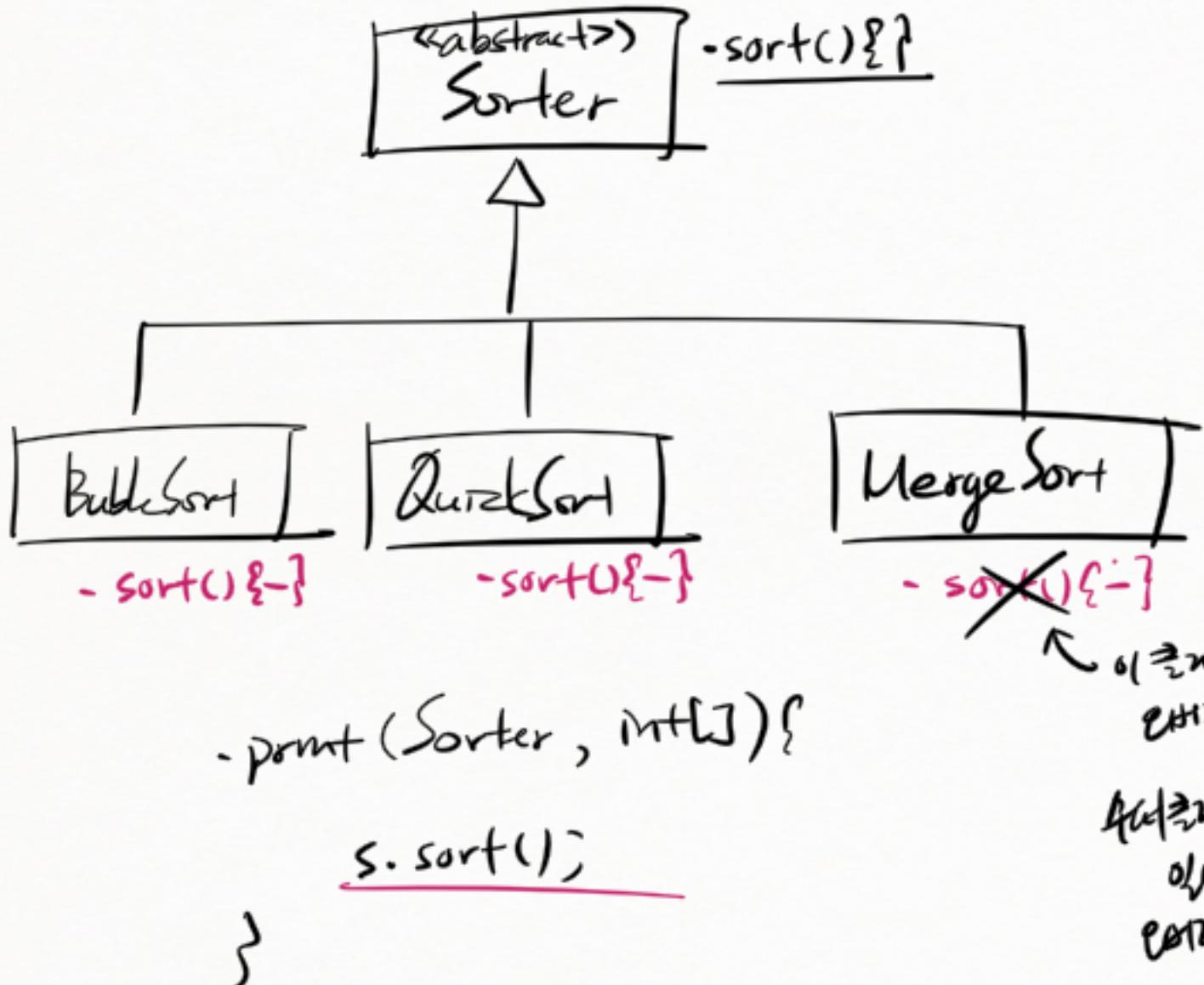
`s.sort();`

}

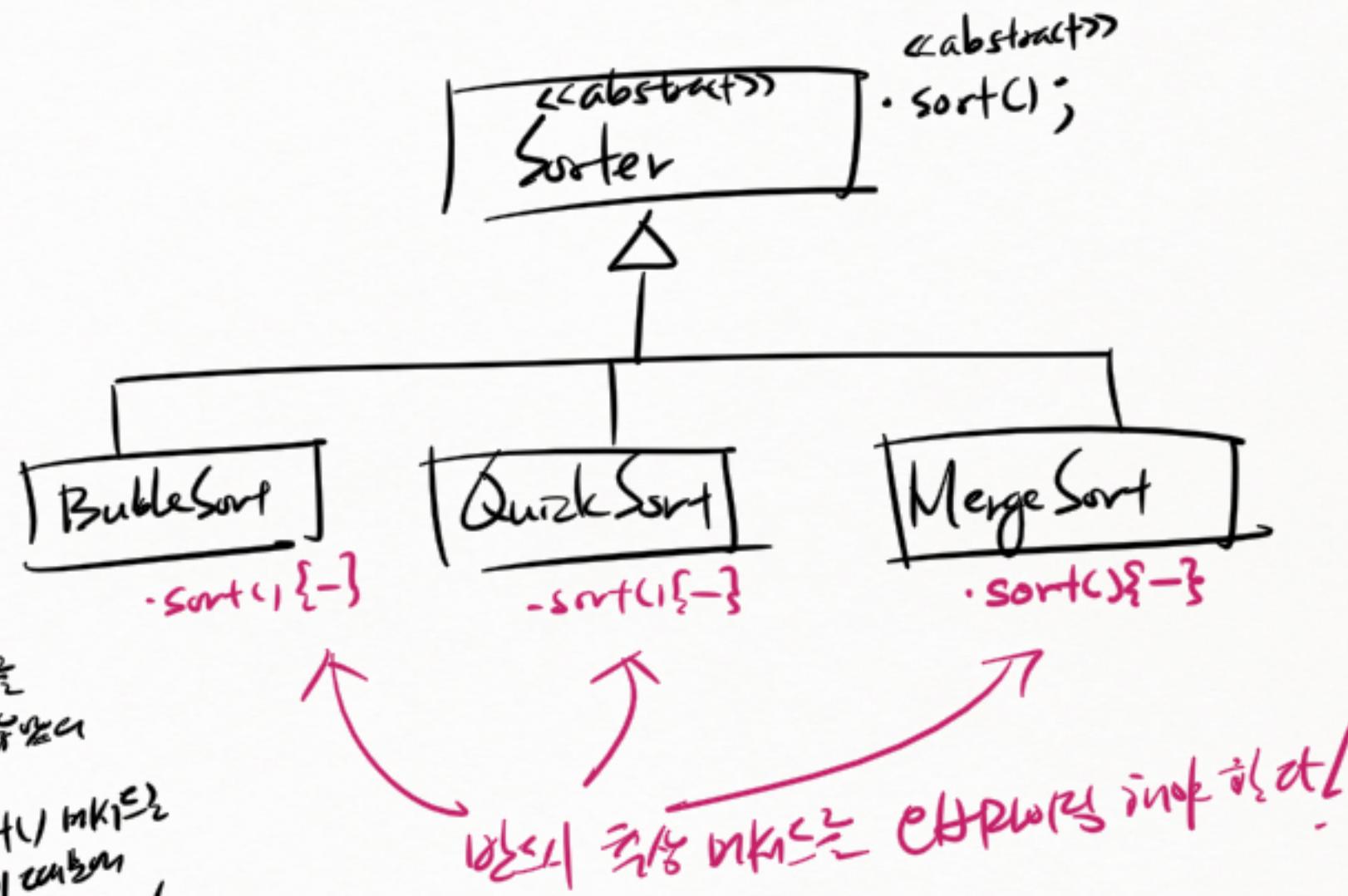
• `정렬하기`  
같은 이름의 메서드 호출  
이름은 `area` 가능

\* 추상클래스와 추상메서드의 활용

③ 추상클래스로 추상클래스로 나누기



④ 추상메서드로 분리하는 것



\* 추상클래스 만든 인터페이스를 끌 때

⑤ 추상클래스 만든 인터페이스로 만들기

