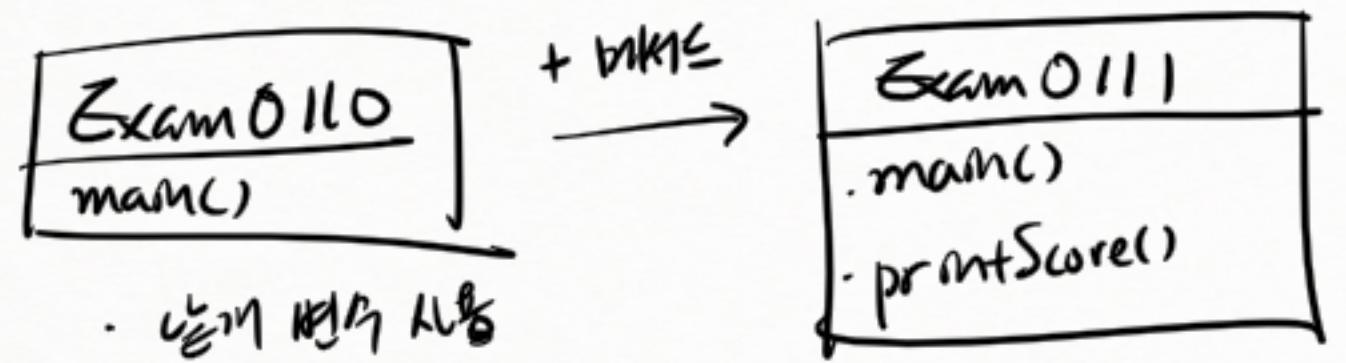


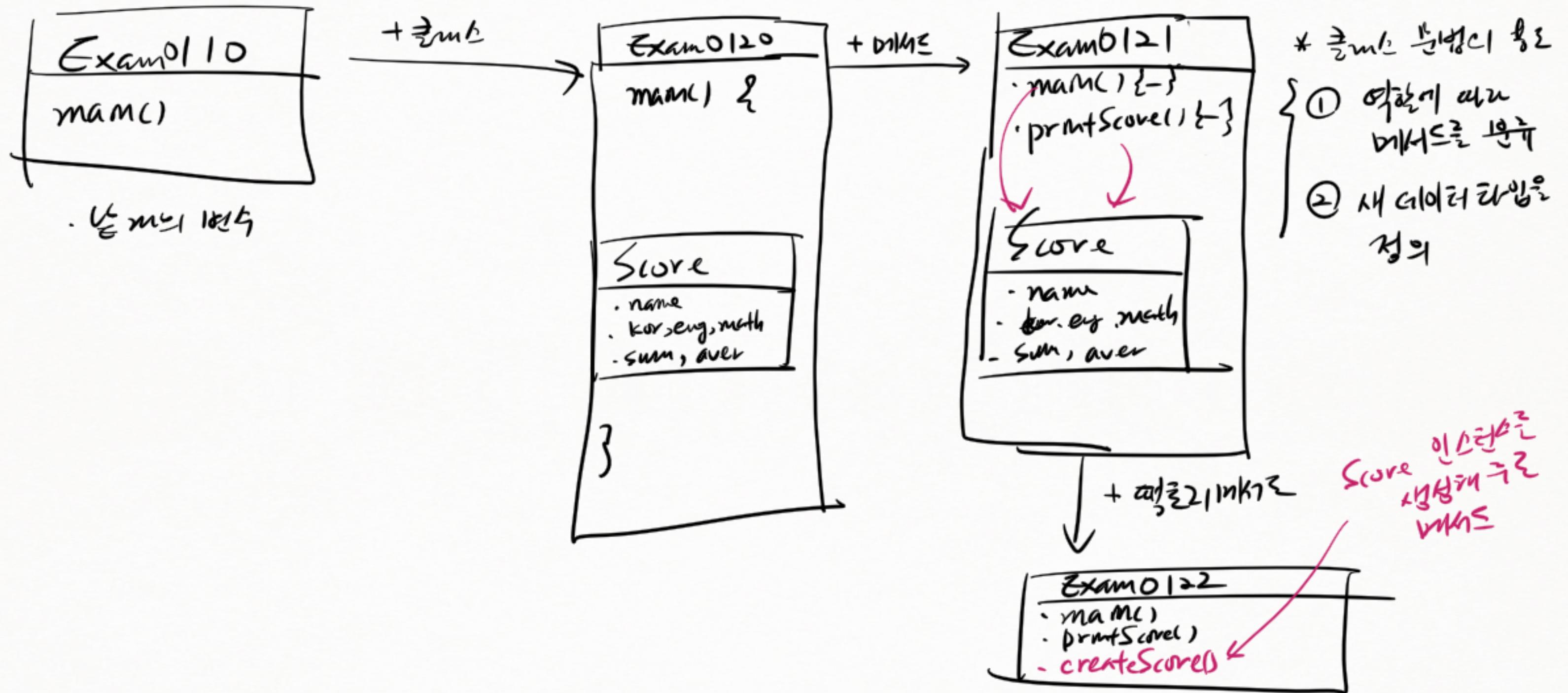
* 재미스 문법 활용 예



- 놓기 허용 사용

- Incls 문법 활용
↳ 자바에서 사용

↓
✓ 정복 코드 세기
↓
코드 처리율 ↑
✓ 유지 보수가 쉬워짐



* 데이터를 101로 → 여러 모의 인스턴스를 다루기

Score s1, s2, s3

s1
200

s2
300

s3
1100

200	name	kor	eng	math	sum	aver
200	○	○	○	○	○	○
300	C	○	○	○	○	○
1100	○	○	○	C	○	○

s1. name = "—"'

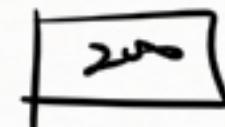
s1. kor = 100 -

:

* 예외처리 10주차

Score[] arr = new Score[3];

arr



null?
- null이면 오류!
- 접근할 수 있는 멤버가 0으로 초기화되었을 때.

arr[0] = new Score();

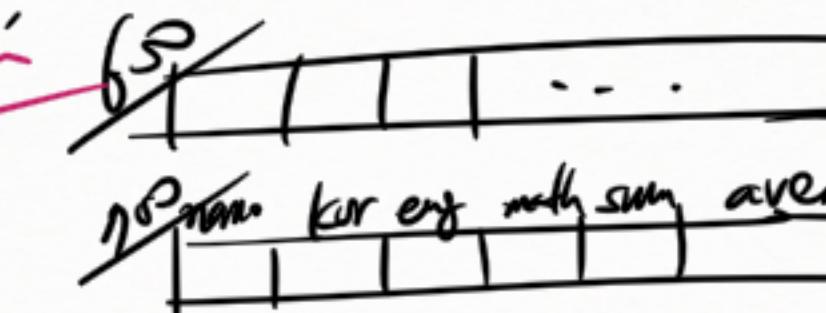
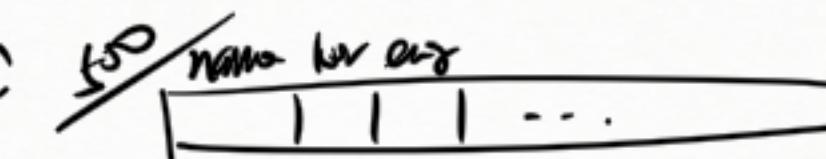
* 예외 처리 ← 자동으로 null로 초기화된다
* 접근할 수 있는 멤버가 초기화되면 좋다.

arr[1] = new Score();

arr[2] = new Score();

arr[3] = new Score();

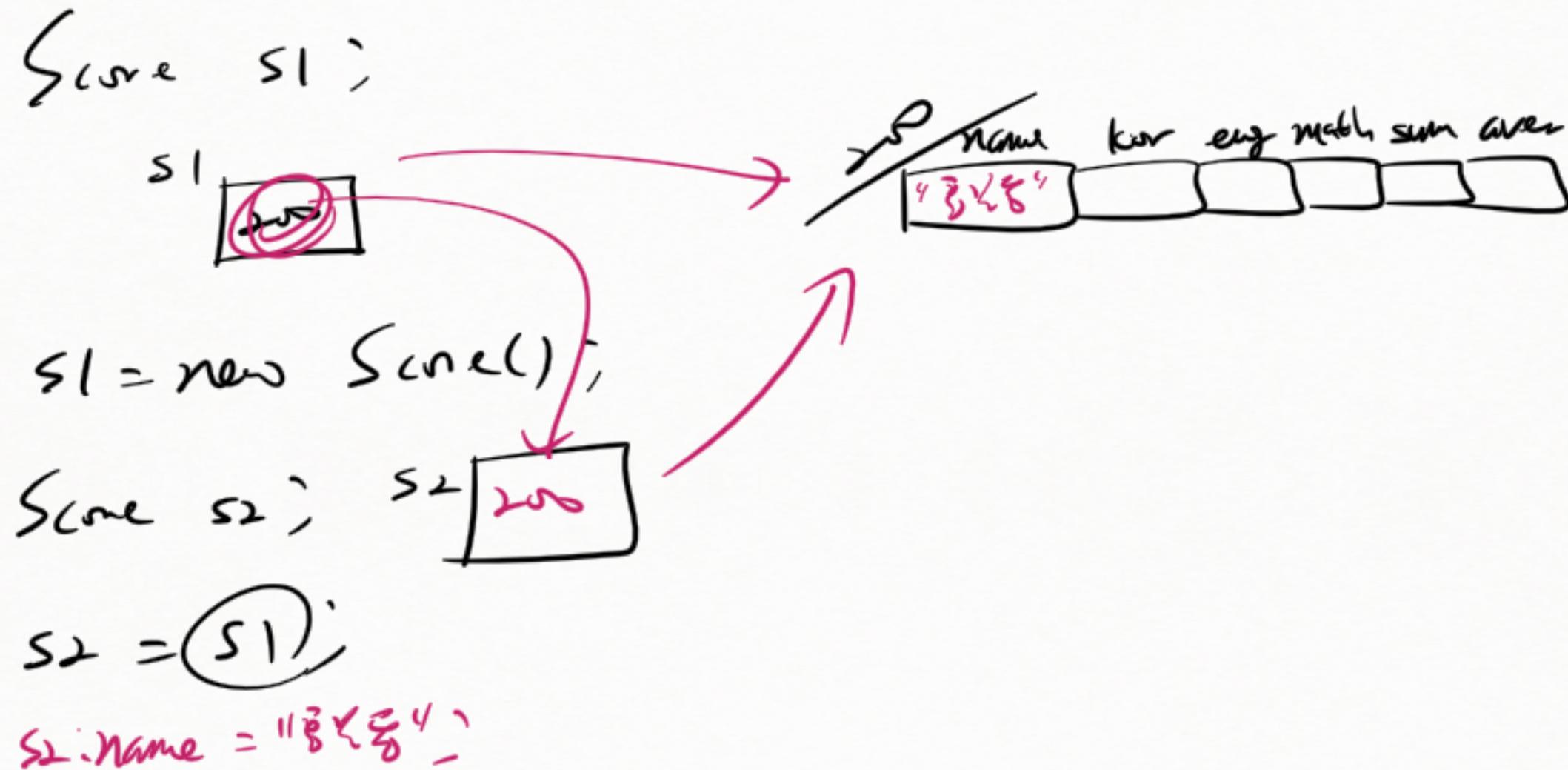
* ArrayIndexOutOfBoundsException



new Score();

Score → null 선언
Heap에 차례로 초기화
기억으로 만든다.

* 리터럴과 인스턴스



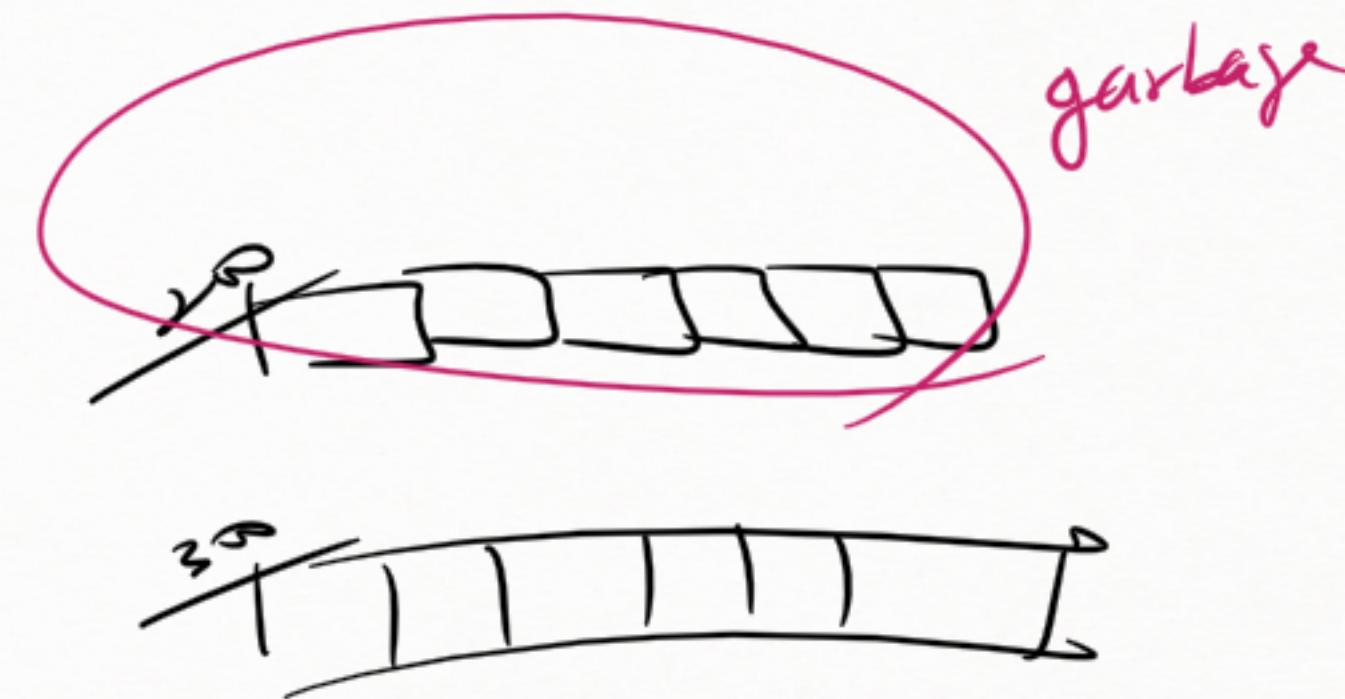
* 7110121 (garbage)

Score s1;



s1 = new Score();

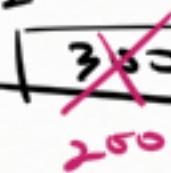
s1 = new Score();

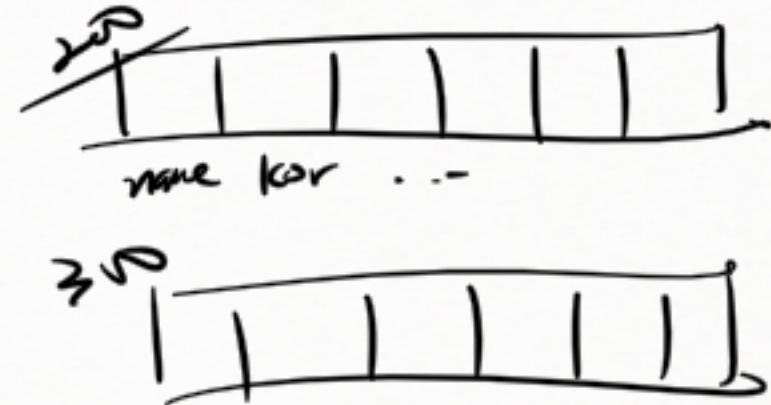


* 리터럴은 카운트와 관계

```
Score s1, s2;  
s1 = new Score();  
s2 = new Score();  
s2 = s1;
```

s1

s2




JVM이 처리

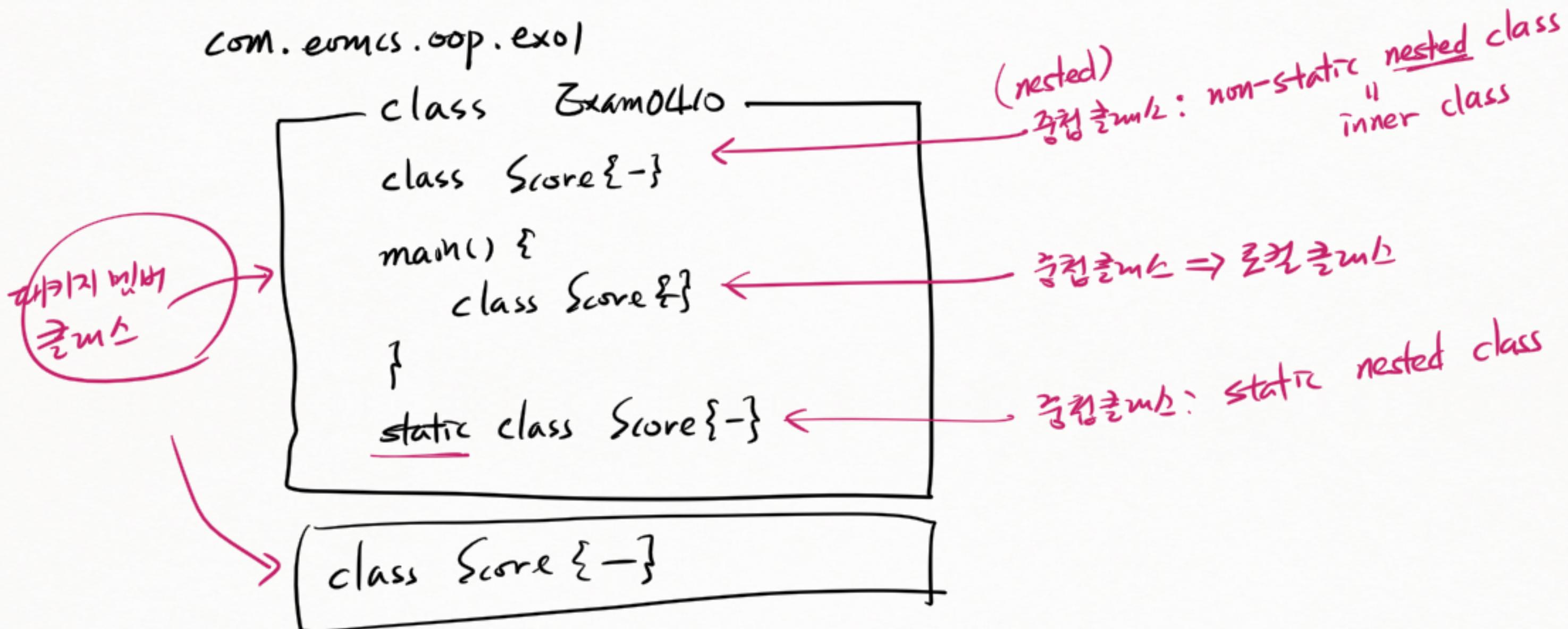
리터럴은 카운트 관계

리터널은 카운트가
0인 경우
"garbage" 가
된다.

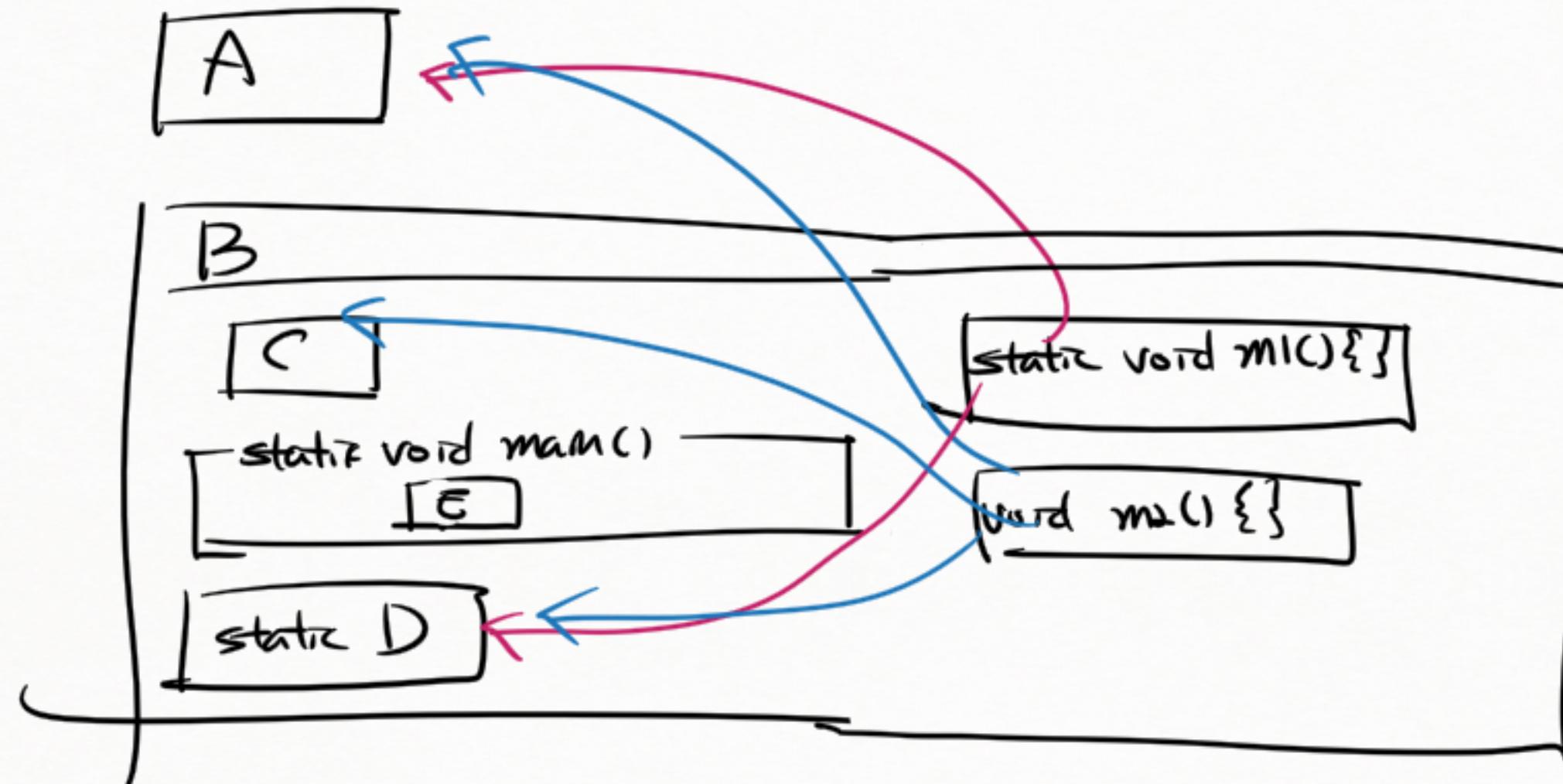
인스턴스	참조 횟수
200	X 2
300	X 0

* 클래스 구조

com.eunics.oop.ex01

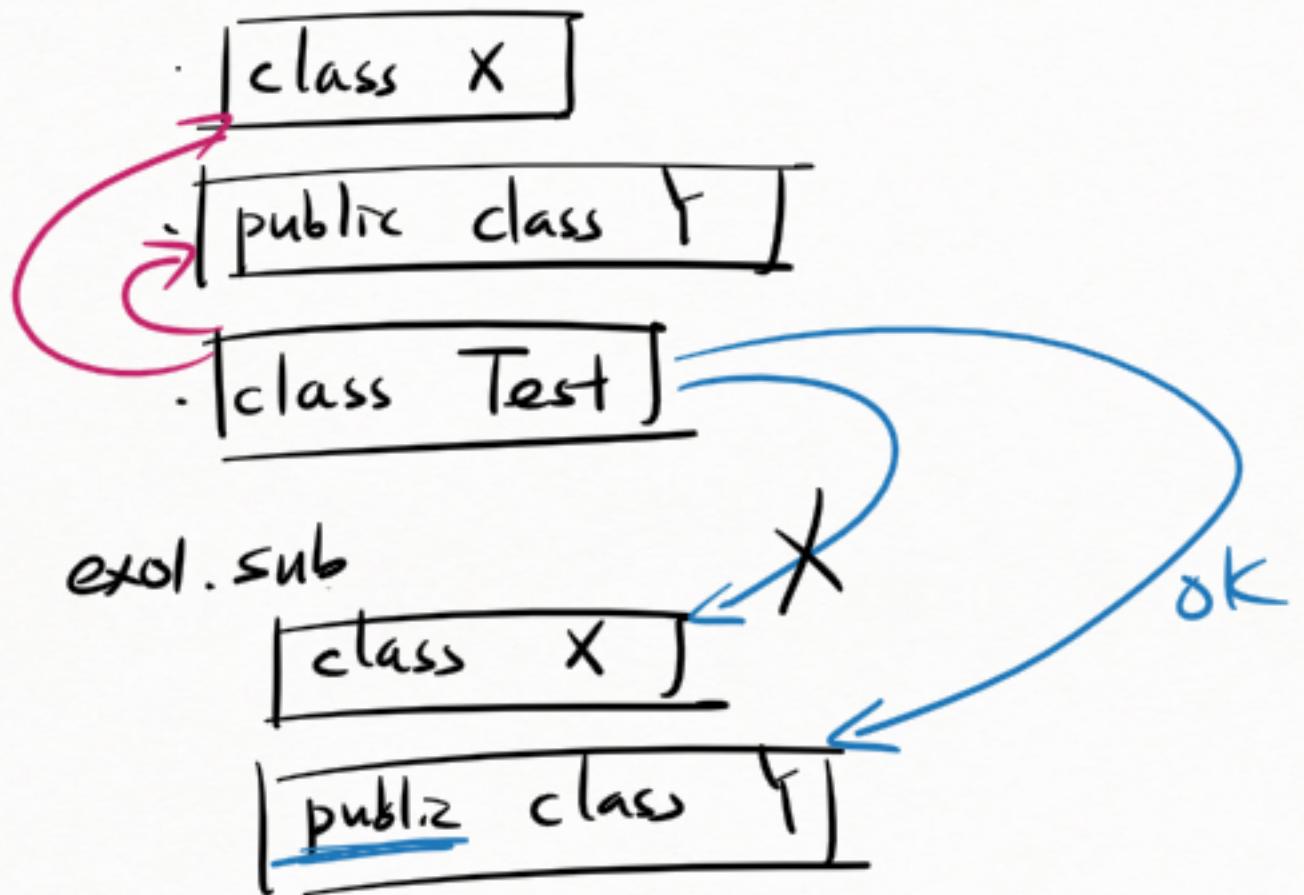


* static member non-static member



* 공개 멤버 필드

ex01



* 클래스 문법의 활용 예: ① 사용자 정의 데이터 타입을 만드는 용도
User-defined Data Type
기본자료형



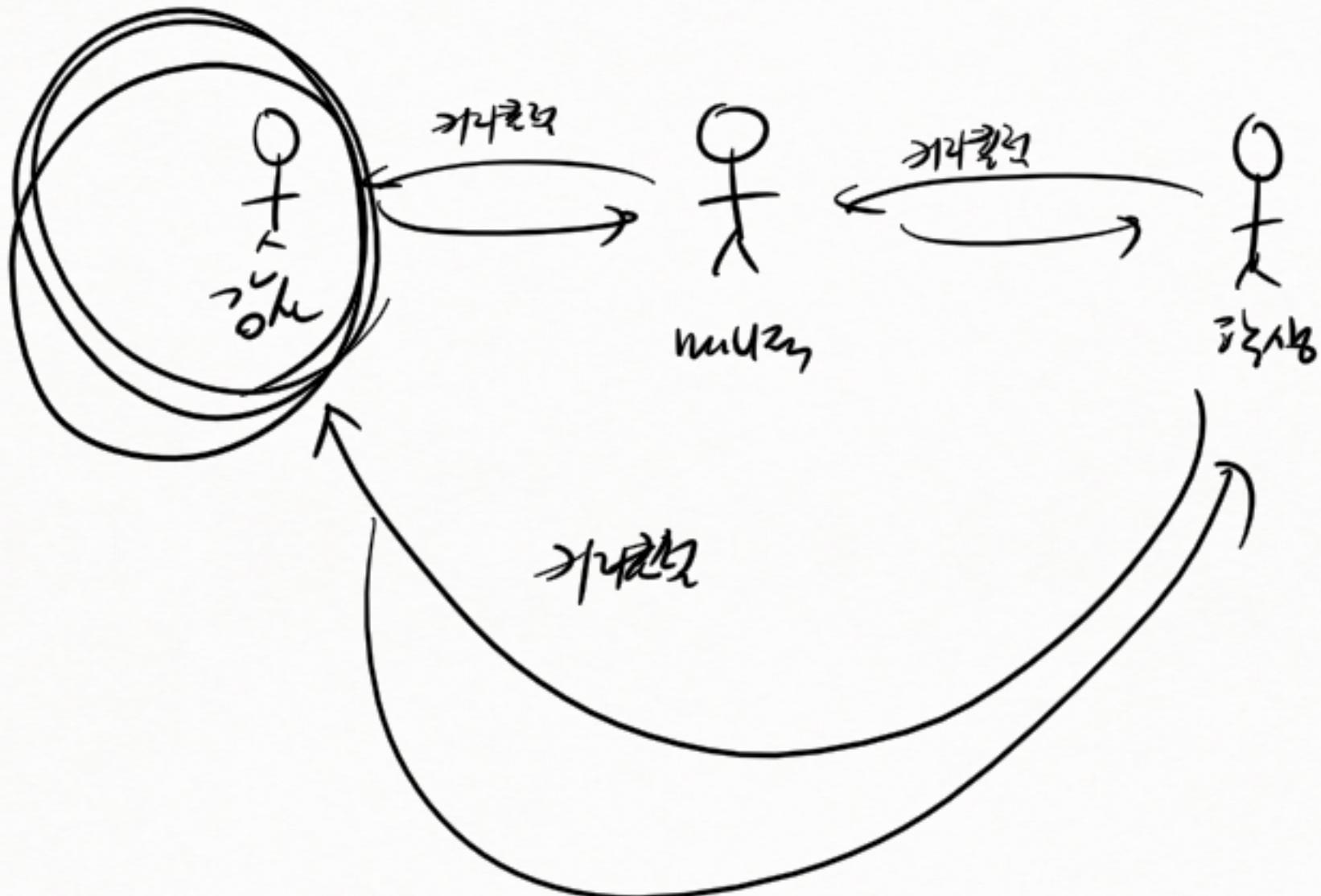
s1.name = "홍길동"

* optimizing(최적화) vs refactoring(재구조화)

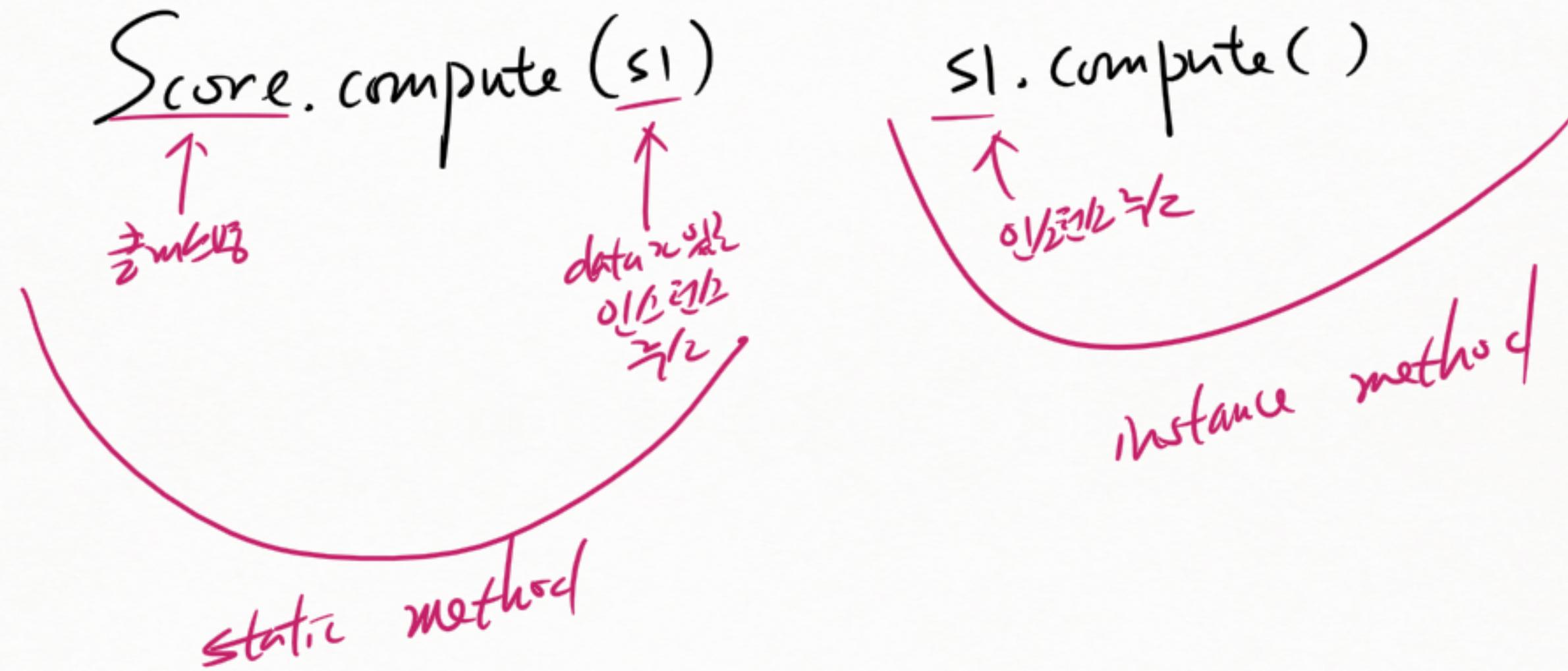
- 속도↑
- 유지보수 품질↑
- 속도↓

- ① s1 리퍼런스에 저장된 주소로 총이어서 해당 인스턴스의 name 변수 —
- ② s1 리퍼런스가 가리키는 인스턴스의 name 변수 —
- ③ s1 인스턴스의 name 변수 —
- ④ s1 객체의 name 변수 (필드)
- ⑤ s1의 name 필드 (변수)

✗ GRASP : 훌륭스며 책임 있는 의사 결정을 +



* static 데일리에 인스턴스 변수



* 인스턴스 메서드와 인자

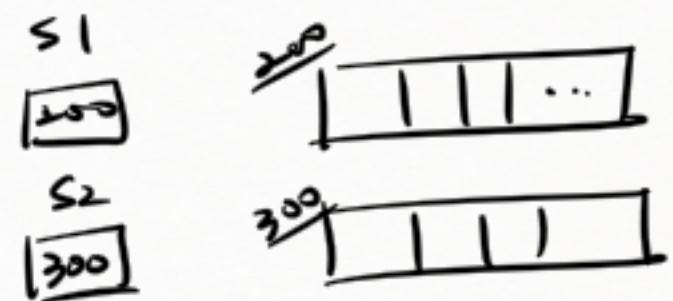
j ++ ;
 operand
 (인자)
 operator (연산자)

j ++;

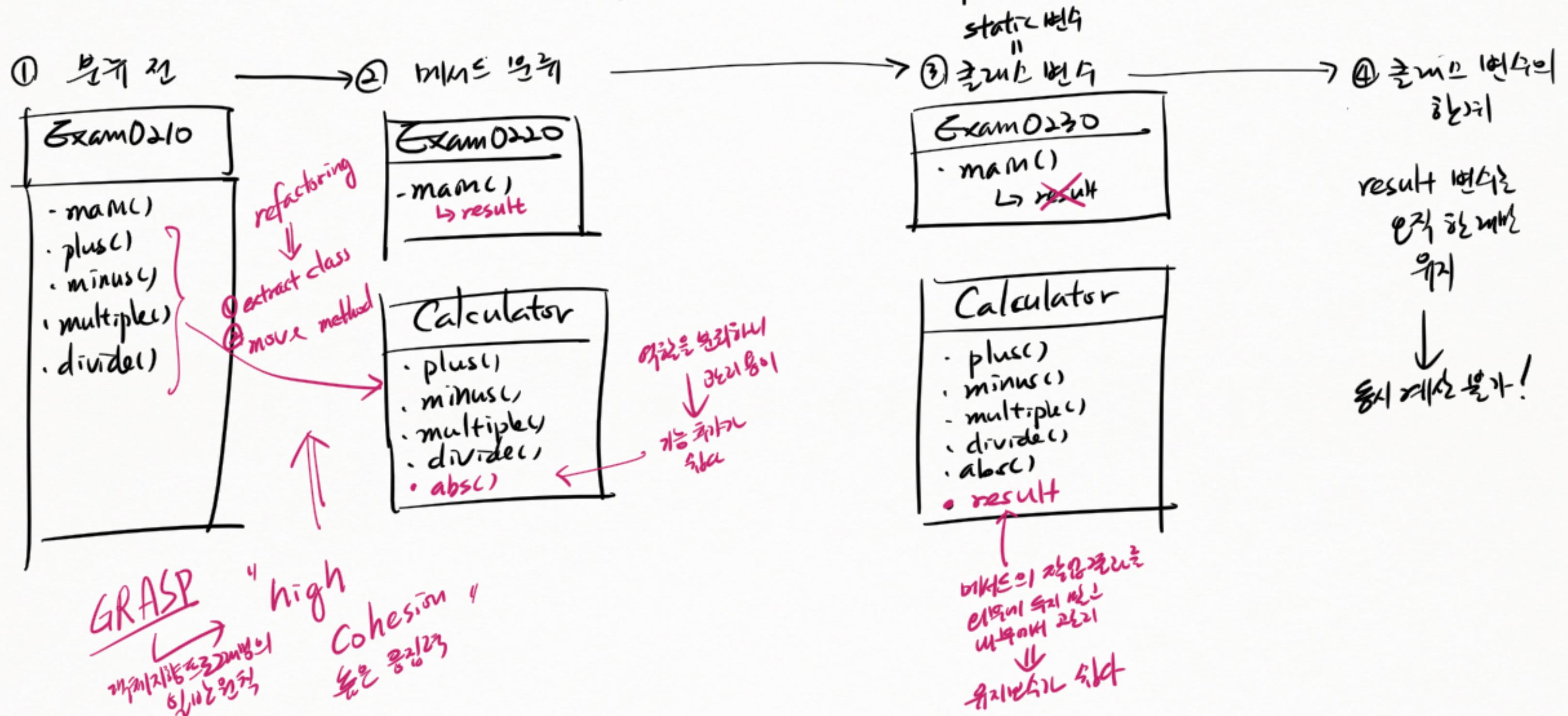
Score s1, s2 ;
 s1 = new Score();
 s2 = new Score()

인자
 ↓
 s1. compute()
 ↑ operand
 ↑ operator

s2. compute()

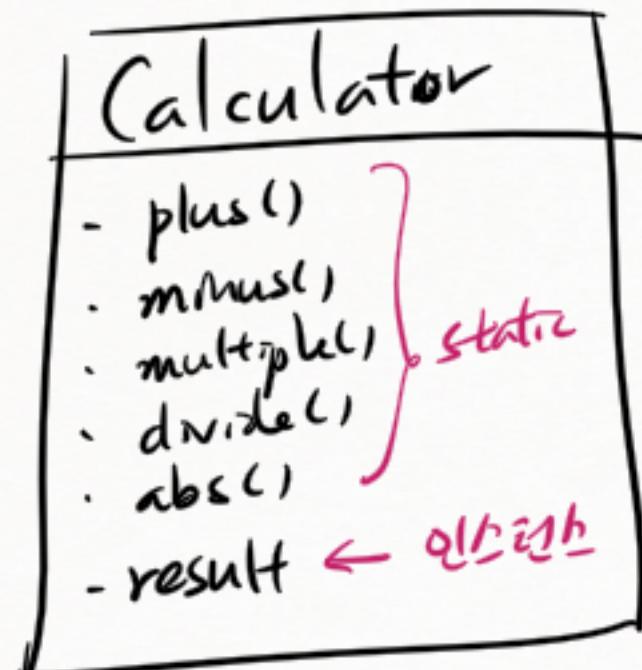
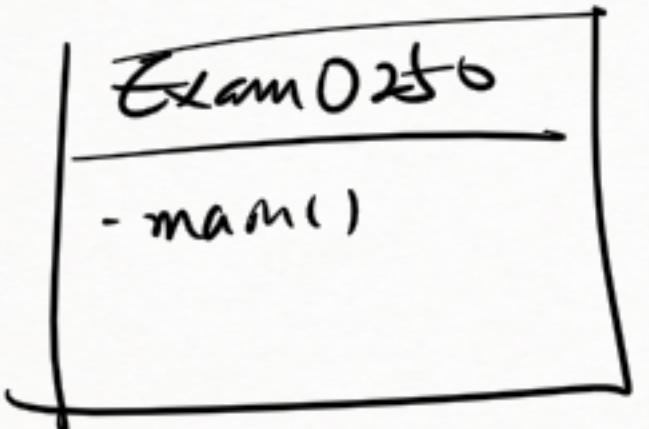


* 3단계 분기점: MMR을 끝난 분기점이



* 구현은 문법입니다 : 인수는 제한이 있지 않습니다

→ ⑤ 인스턴스와 스택



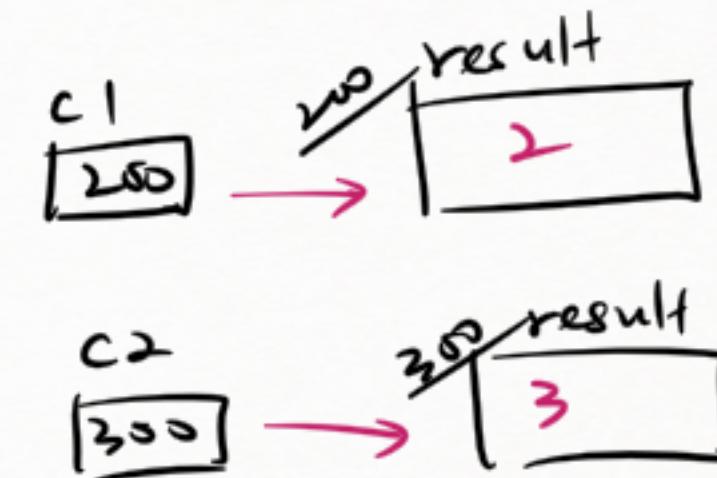
```
Calculator c1 = new Calculator();
Calculator c2 = new Calculator();
```

```
Calculator.plus(c1, 2);
```

```
Calculator.plus(c2, 3);
```

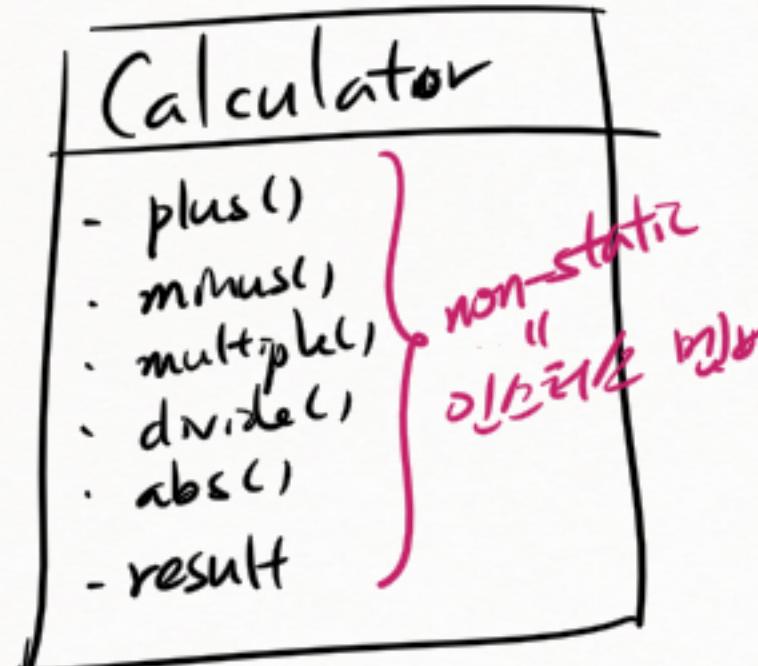
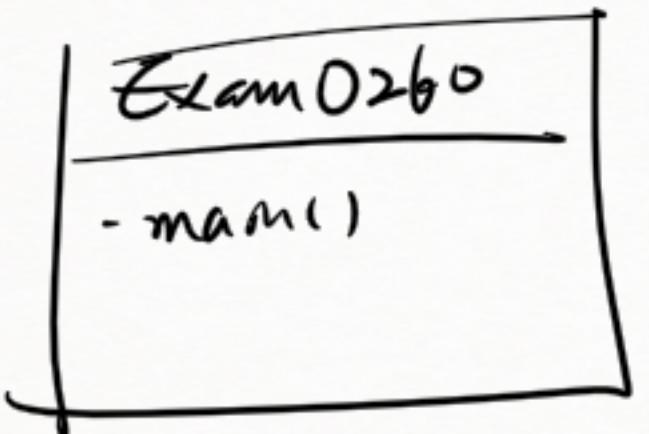
:

↑
각 인스턴스는 각각의
result 변수를 갖습니다.
이스턴스마다 결과가
다릅니다.



* 인스턴스 멤버 변수: 인스턴스마다 다른 값을 갖다

→ ⑥ 인스턴스 변수

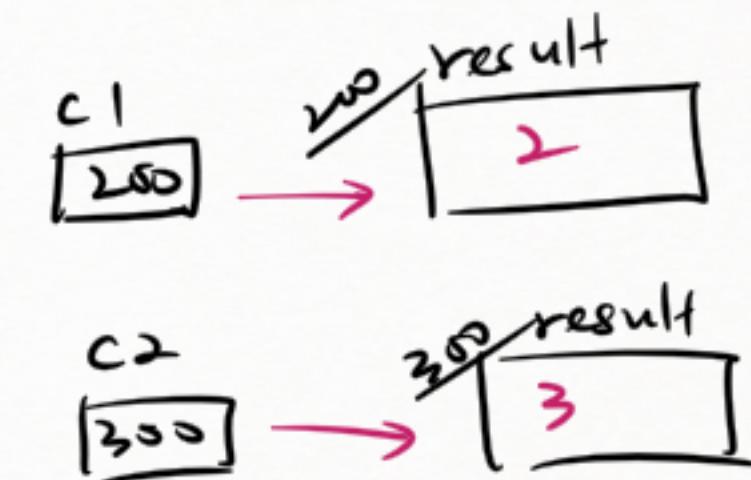


Calculator c1 = new Calculator();
Calculator c2 = new Calculator();

c1.plus(2);
c2.plus(3);

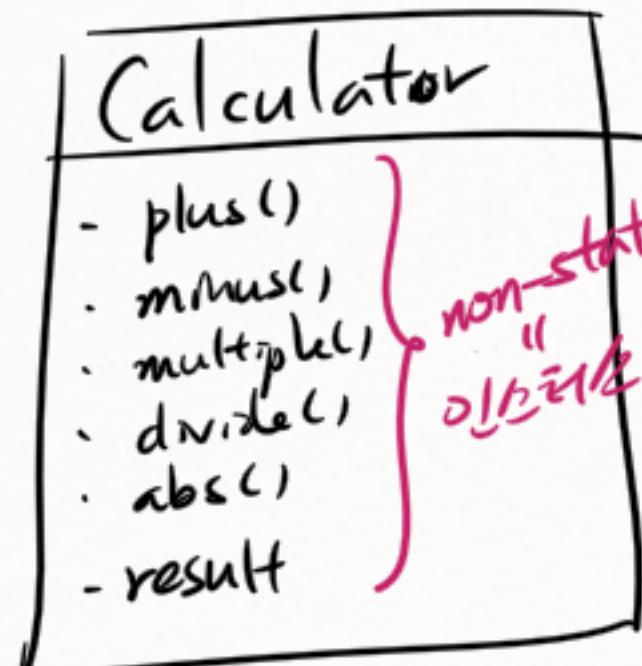
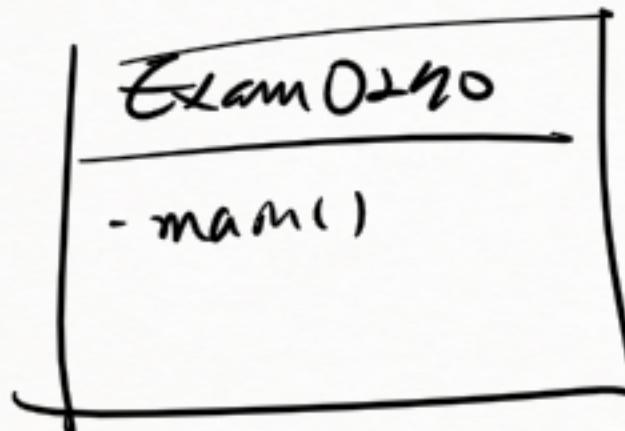
인스턴스 변수

인스턴스 변수
c1.plus(2)
+ 2

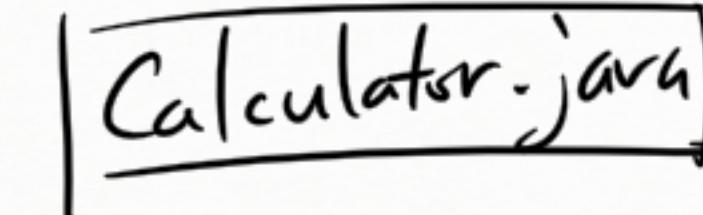
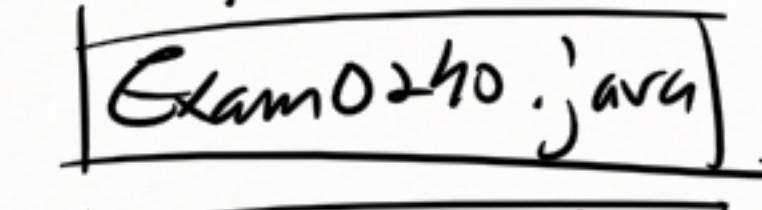


* \Rightarrow ml² ပုံမှန် ရှိခဲ့ပါ : မြတ်သွေးကို ဖြန့်မျေး

→ ① အော်လုပ်မှု \Rightarrow ml² → ② အော်လုပ်



com.eomcs.oop.ex02.



com.eomcs.oop.ex02.Exam0240

com.eomcs.oop.ex02.util.Calculator

import com.eomcs.oop.ex02.util.Calculator;

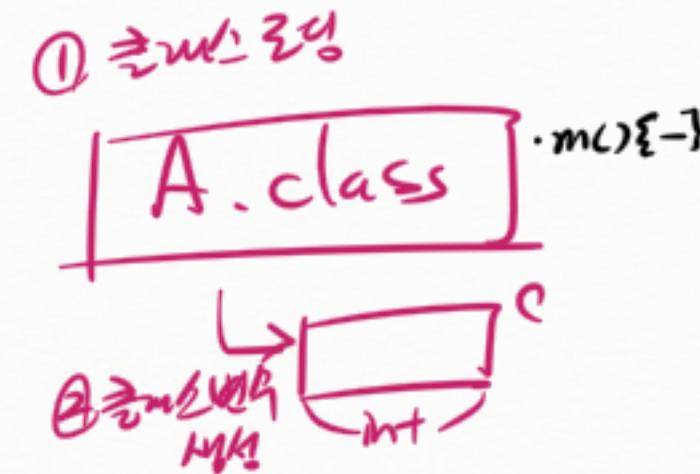
\Rightarrow အော်လုပ် ပုံမှန် ရှိခဲ့ပါ။
အော်လုပ် ပုံမှန် ရှိခဲ့ပါ။

* static 멤버 와 인스턴스 멤버

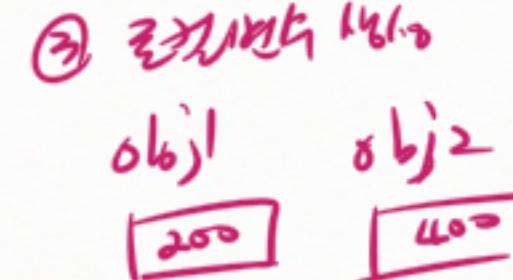
```
class A {
    int a;
    int b;
    static int c;
    void m() { }
}
```

```
A obj1 = new A();
A obj2 = new A();
```

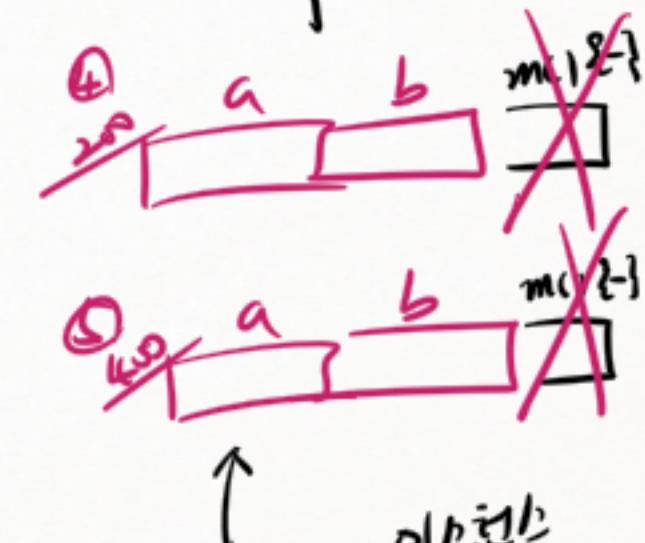
Method Area



JVM Stack



Heap



\uparrow \downarrow
 A 클래스의 인스턴스화

인스턴스화 \Rightarrow 복제.

* 클래스 멤버와 인스턴스 멤버

```
class Calculator {
    int result;
    void plus(int v) {
        result += v;
    }
}
```

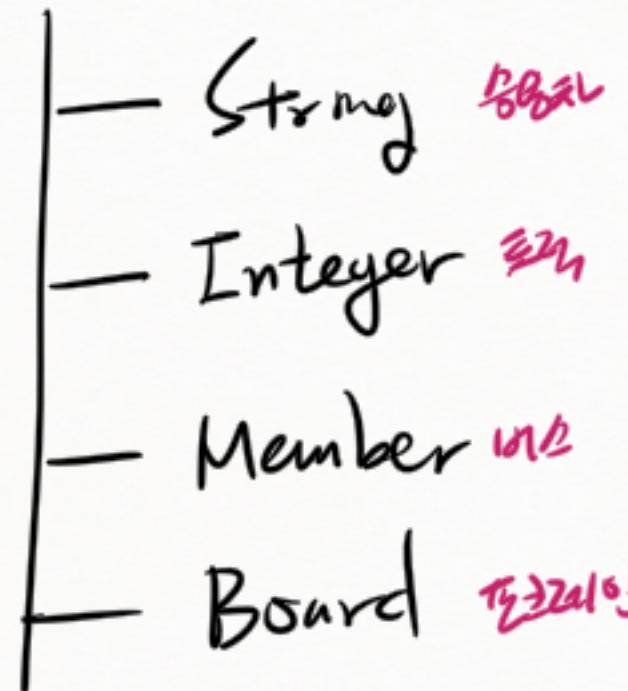
클래스 멤버는
인스턴스 멤버를
다룬다.
연산자!
(인스턴스)

클래스 멤버는
인스턴스 멤버를
다룬다.
연산자!
(인스턴스)

* Object 클래스
↳ 자바의 최상위 클래스

java.lang.

Object 사용

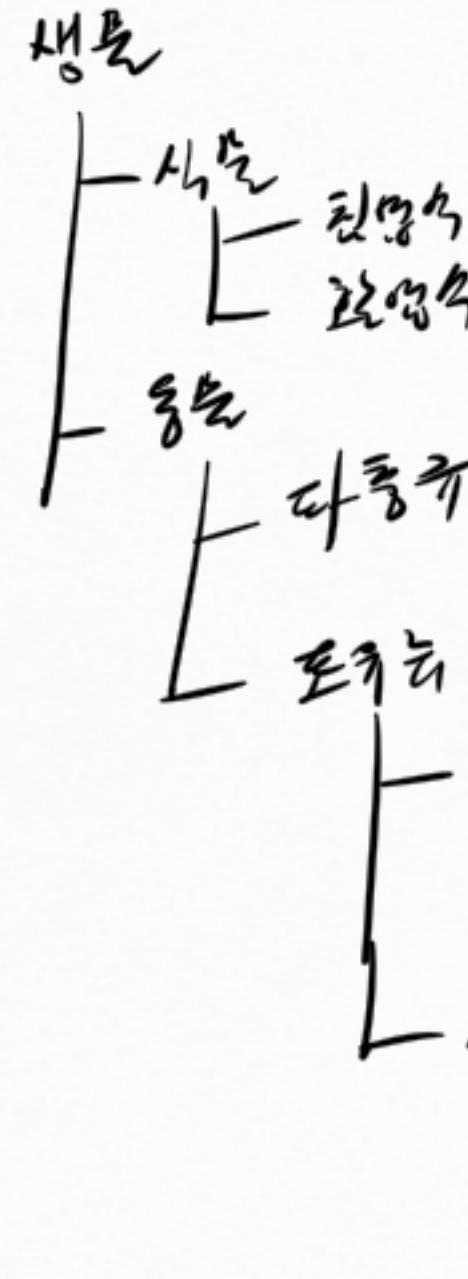


자바의 모든 것은 Object의
자식 클래스이다.

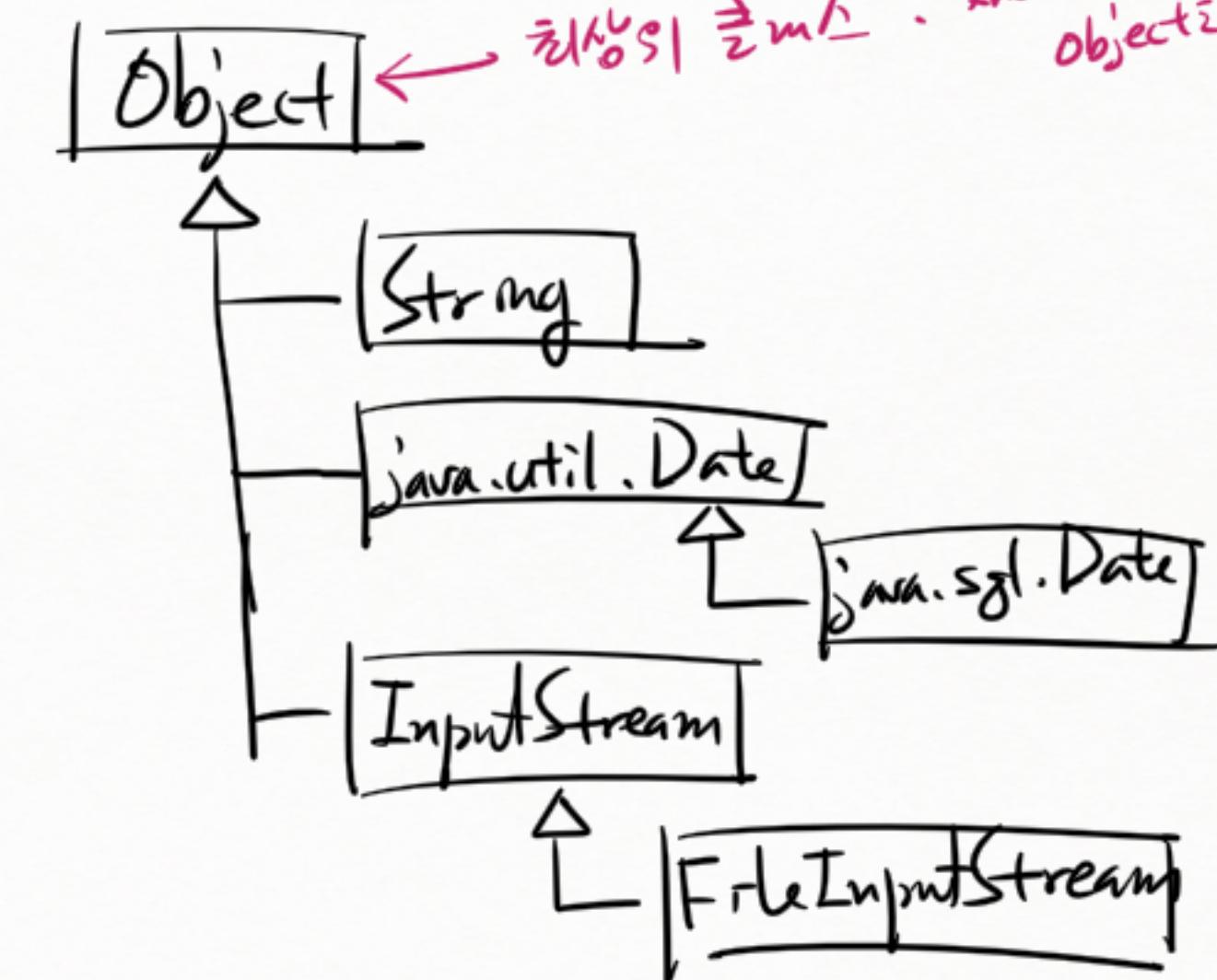
자식 (sub)

* 물류와 출판 분야

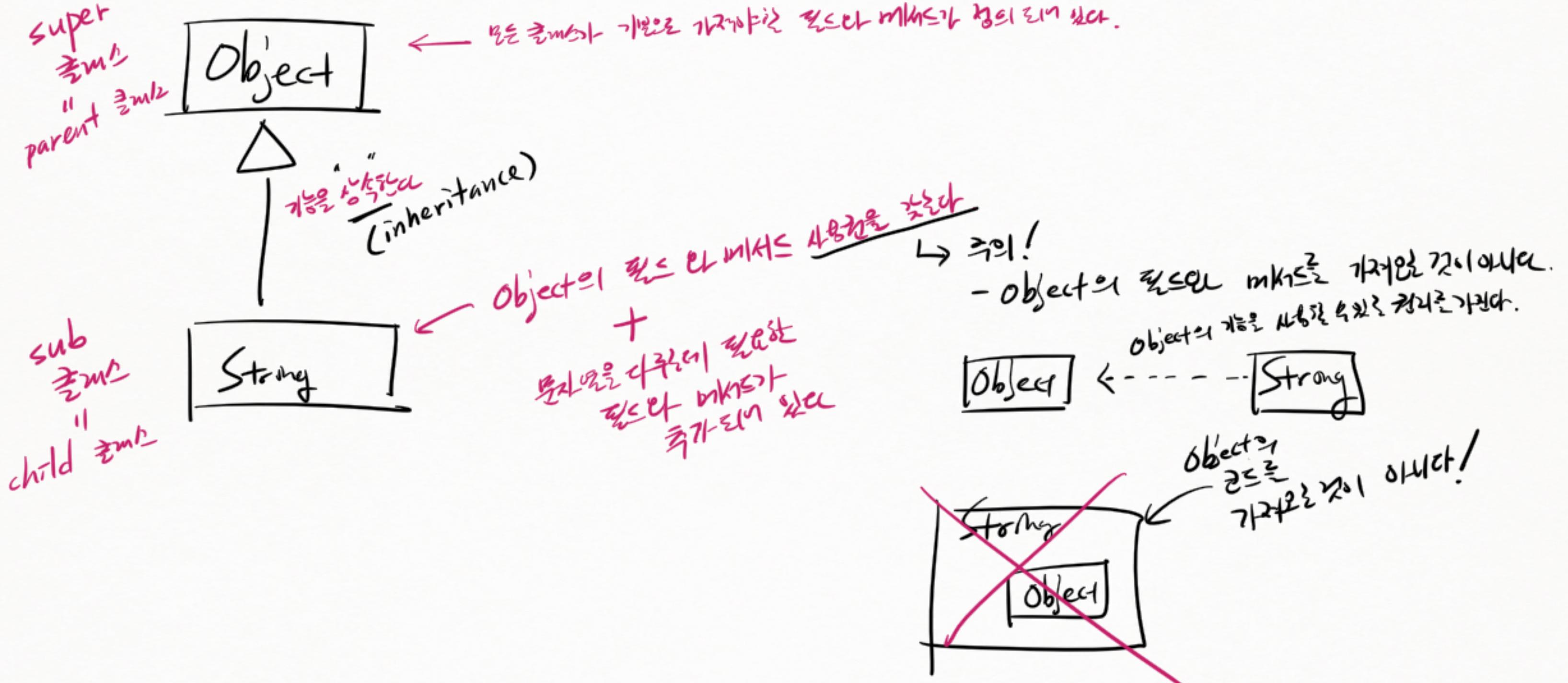
1877
MBS



제작된 파일로 살펴보면
제작을 구성하는 있다.



* 상위 클래스와 하위 클래스 : 같은 공유를 위한!



* 향수 예법: 근드 중의 향법 중 하나.



* 상속 문법과 다형성

- ↳ 대형화 범위 *
- ↳ 오버로딩 (overloading)
- ↳ 오버라이딩 (overriding) *

Car c;

```
c = new Car();
c = new Sedan();
c = new Truck();
c = new Trailer();
c = new Dump();
```

↳ 대형화 범위

Truck t;

```
t = new Car();
t = new Sedan();
t = new Truck();
t = new Trailer();
t = new Dump();
```

상속 | 출현 | 리턴값이
하나 | 출현 | 이전은 가능
하지만 수 있다
 ↓
구현 | 출현 | 가능
가지 않을 수 있다.

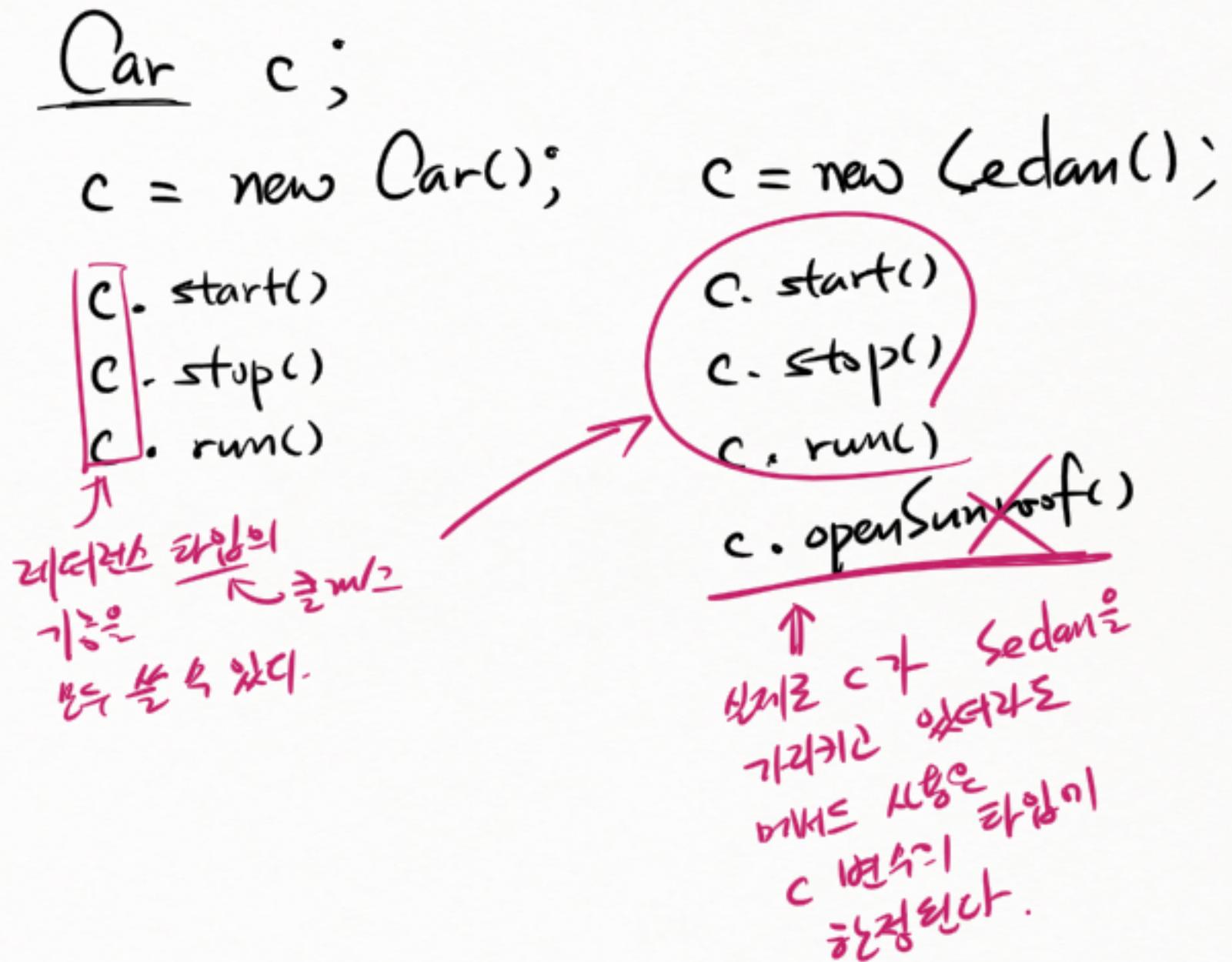
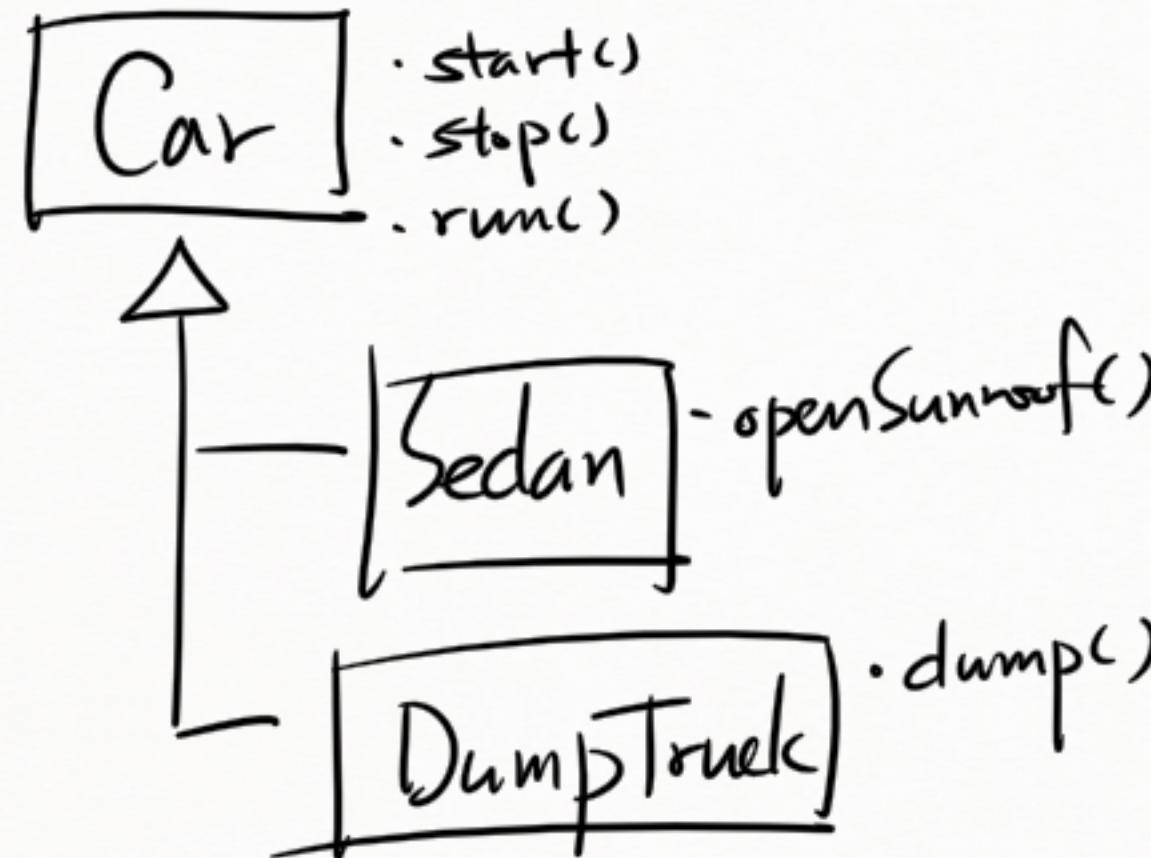
* 쿨러스 정의와 오버라이드 사용하기

```
class Board extends Object {  
    =  
}
```

(생략가능!)

```
class Member {  
    =  
}  
↑  
↑ 멤버를 2정의할 때  
↑ 이를로 Object를  
↑ 멤버로 정한다.
```

* 상속과接口



$$\text{End}_d = \left(\frac{\mathbb{Z}}{2^m} \right)_{\text{primitive type}}$$

Sedan s

```
s = new Sedan()
```

s. start()
s. stop()
s. run()

s. openSurfaces

수퍼 출판의
기본은 판권법
이며.

s. start()
s. stop()
s. run()

~~s.open("root");~~

Sedum \cap Car.

Sedum \cap Hipp.

Sedum \cap Agave

Sedum \cap Hosta

Sedum \cap Yucca

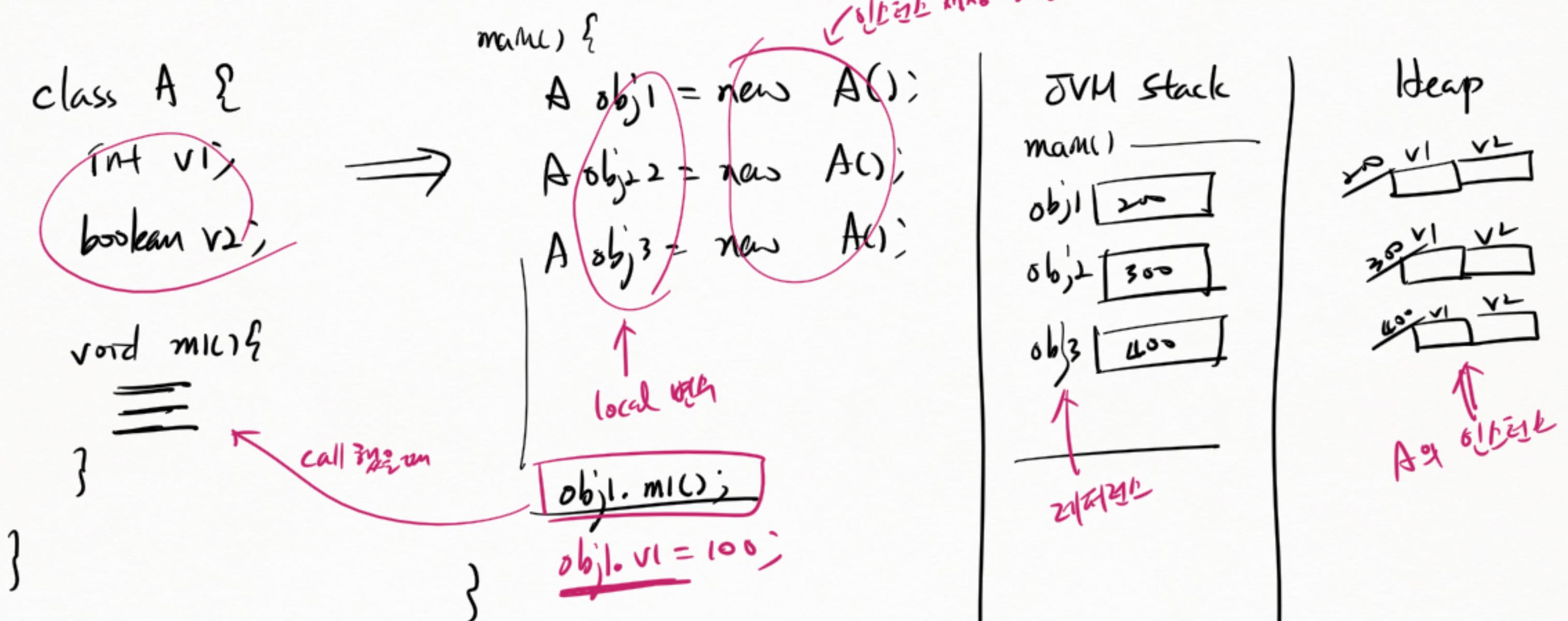
Sedum \cap Stachys

○ 이런 잘못된 시기로
○ 이런 초기에 1052년에
○ 이런 초기에 1051년에
○ 이런 초기에 1050년에
○ 이런 초기에 1051년에
○ 이런 초기에 1050년에
○ 이런 초기에 1050년에

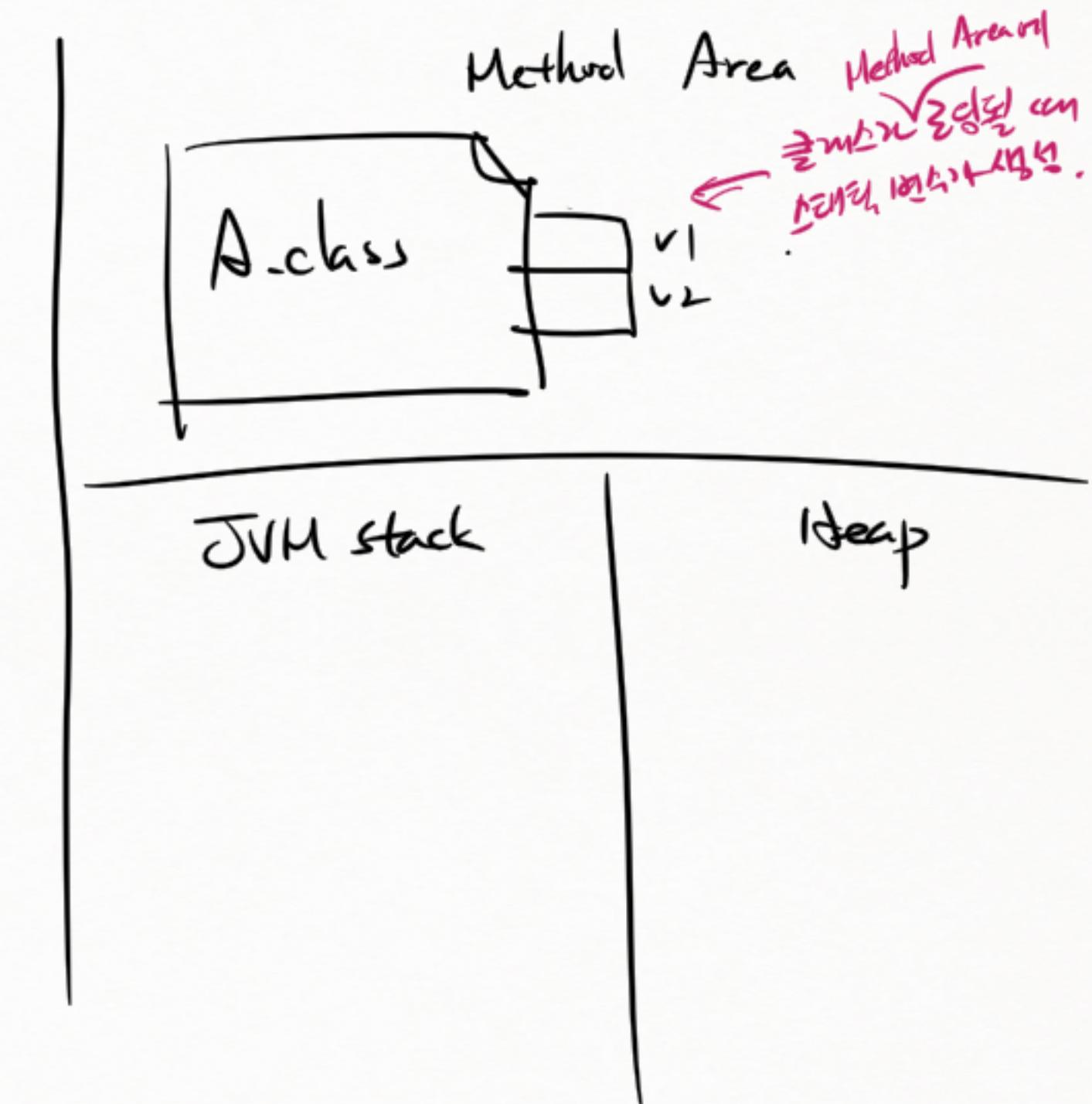
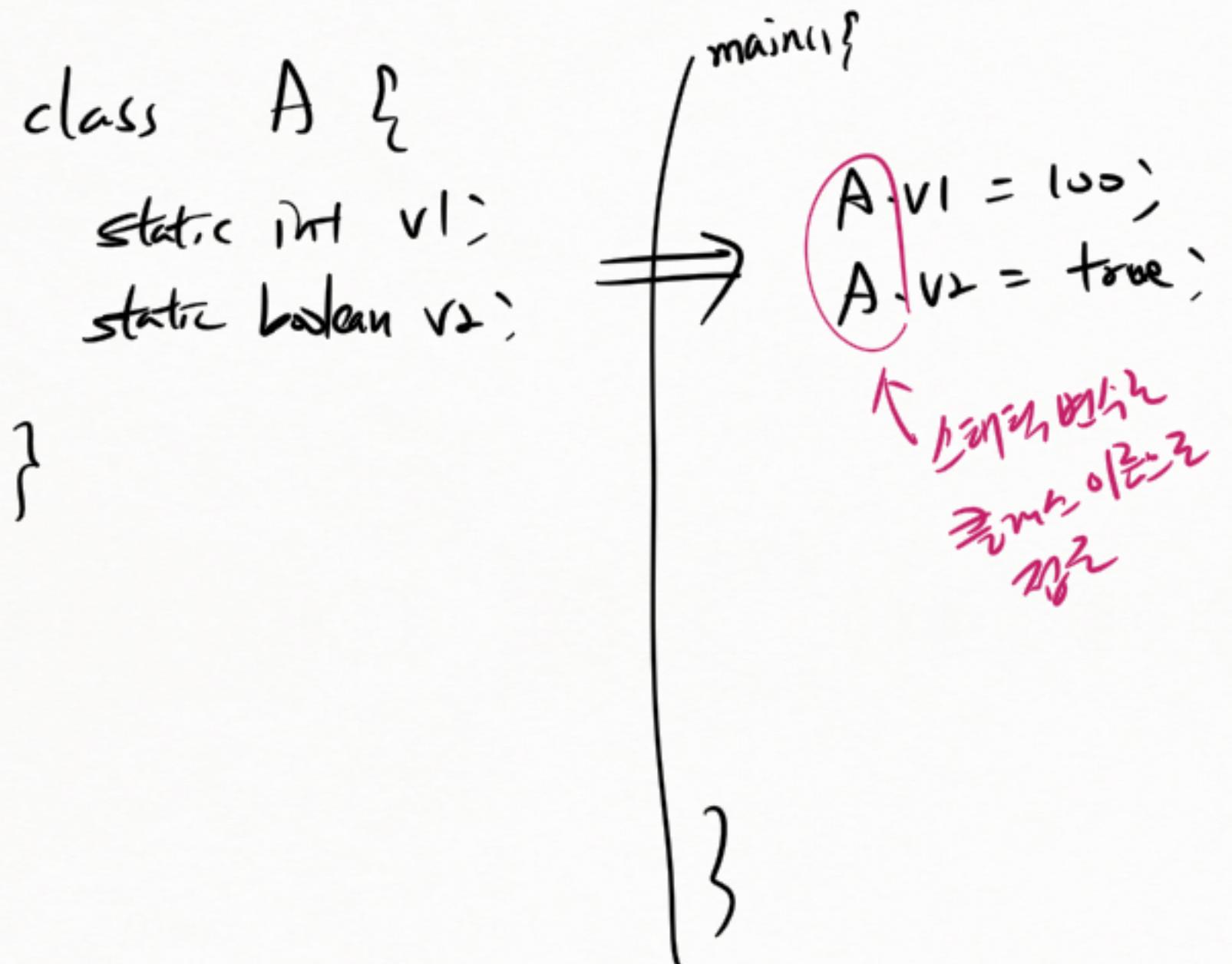
* static 필드와 non-static 필드

```
class A {  
    int a; // non-static 필드(변수) ← Heap ← Garbage Collector가  
           // 관리할 영역  
    static int b; // static 필드(변수) ← Method Area  
}
```

* Object (non-static) 풀드



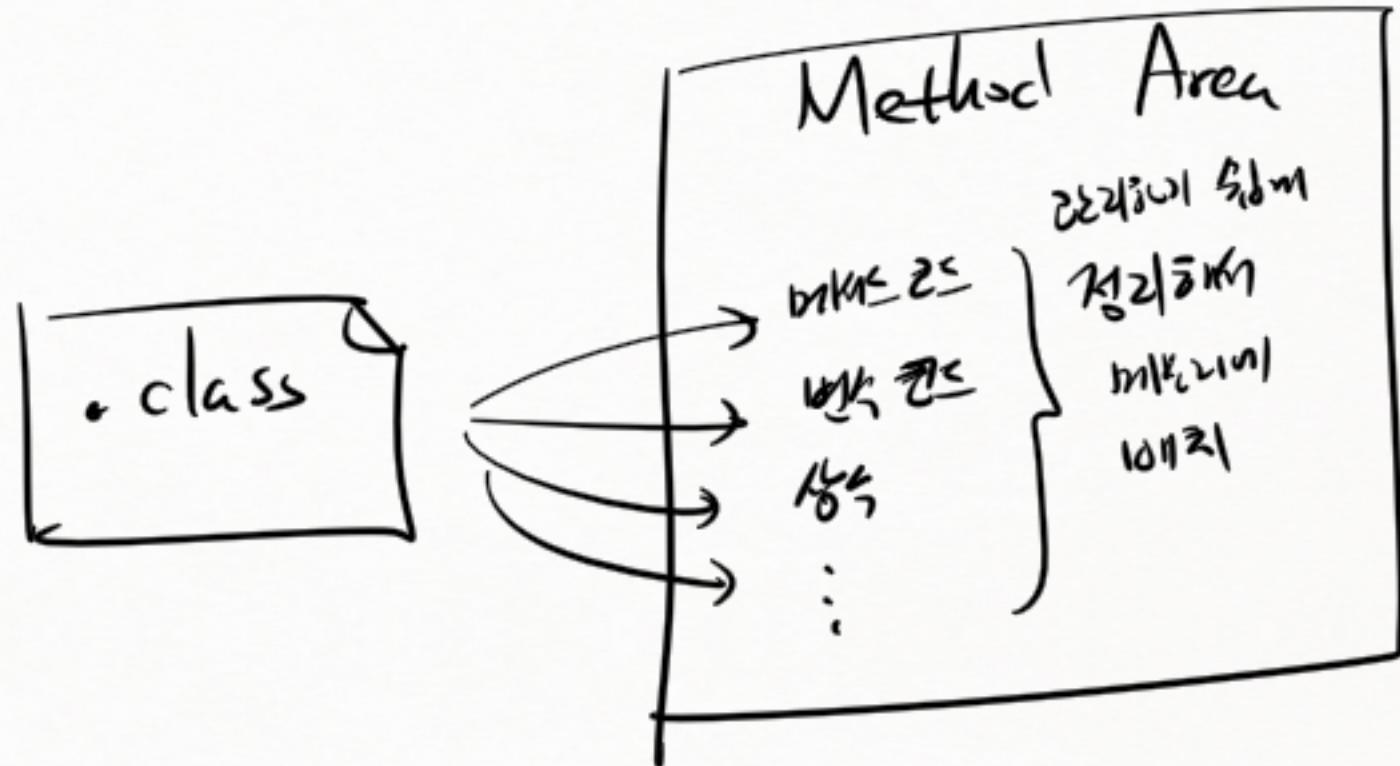
* 주소(static) 필드(값)



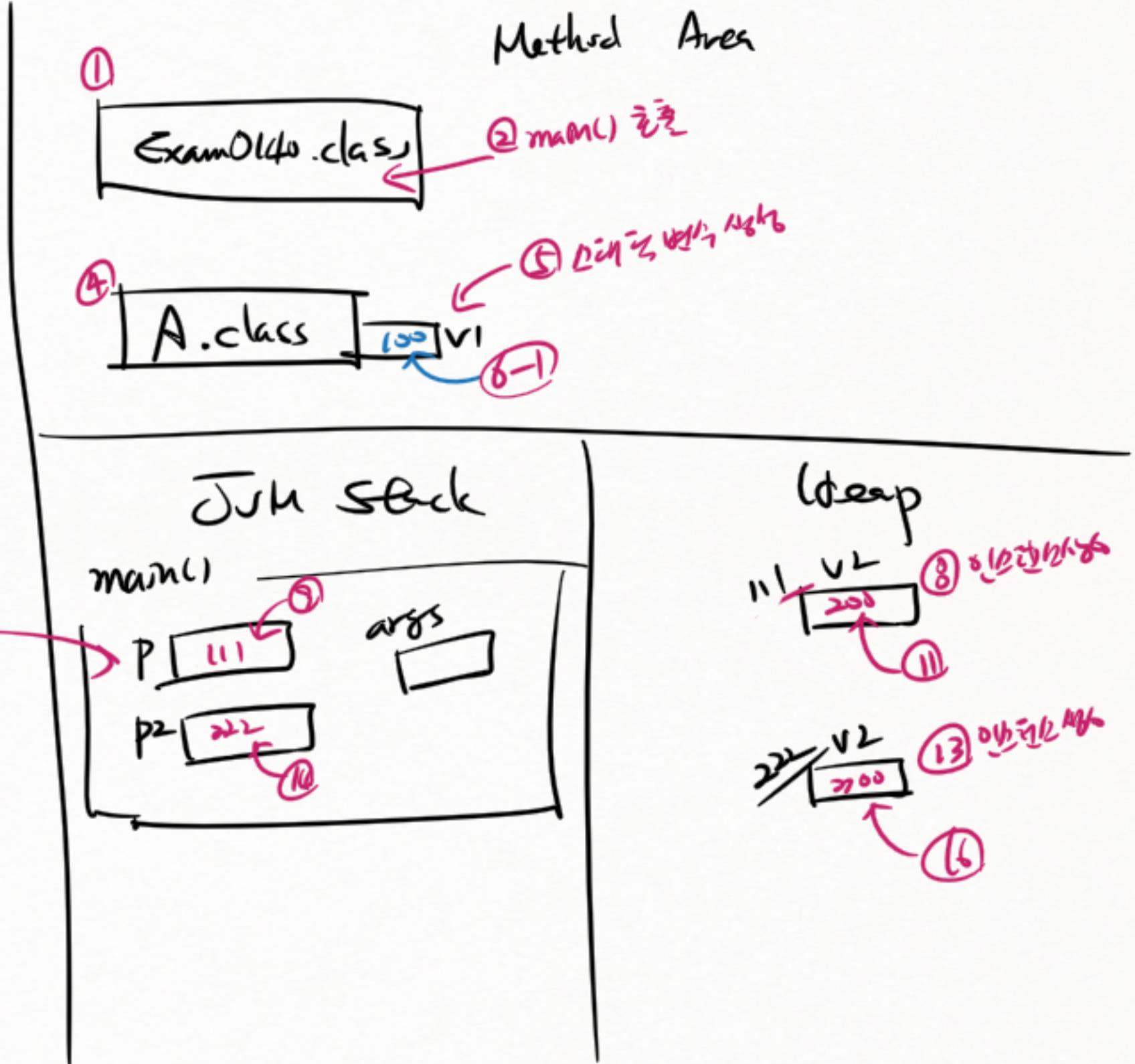
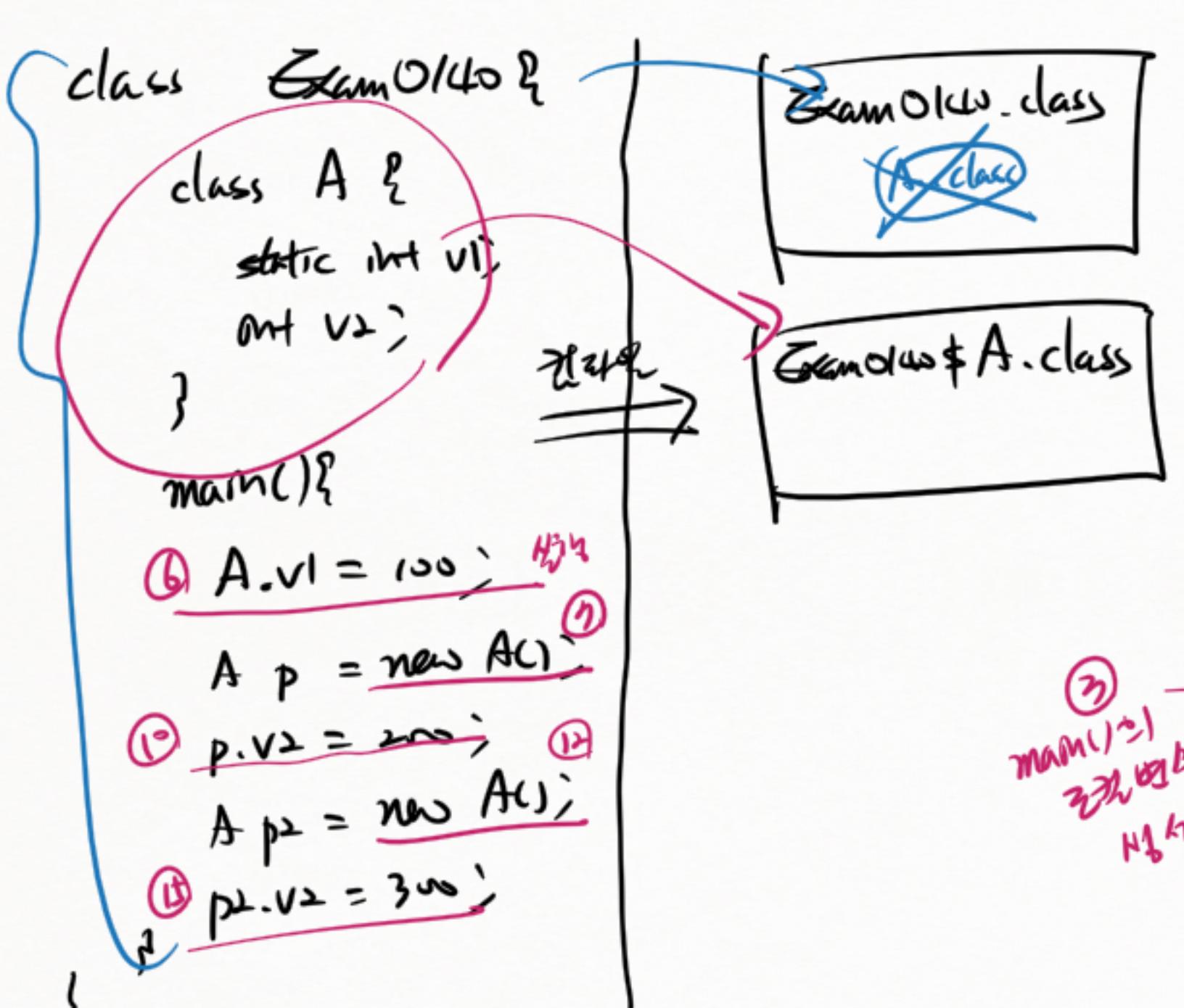
* JVM의 로딩과 실행

\$ java Hello

- ① Hello.class 찾는다
- ② Bytecode 검증
- ③ Method Area에 로딩 \Rightarrow
- ④ 스레蚀 필드 생성
- ⑤ 스레蚀 블록 실행
- ⑥ main() 호출

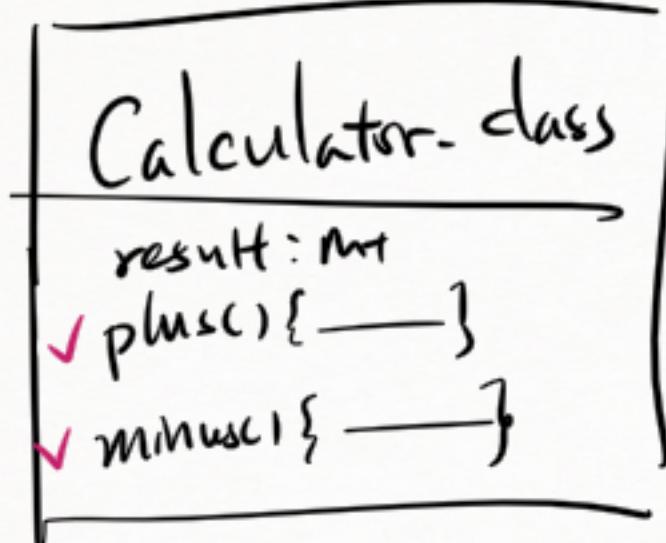
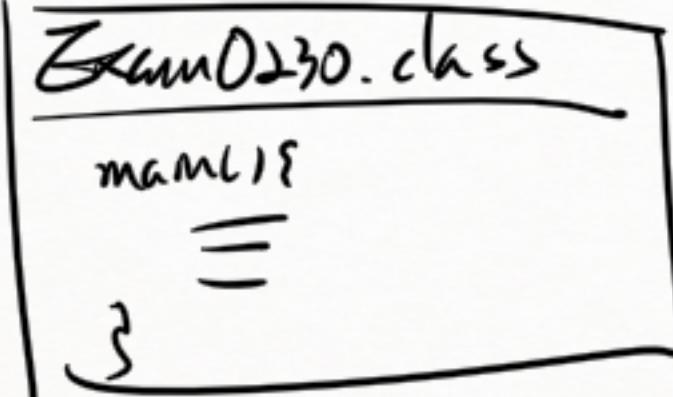


* 퀸을 조망, 스파워 월드 및 인터넷 월드 사용

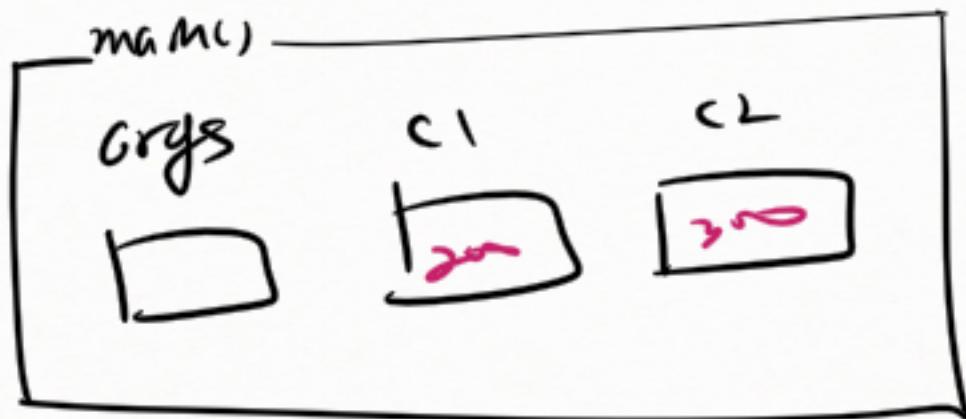


* 인스턴스 변수와 인스턴스 Method 둘다 9/21

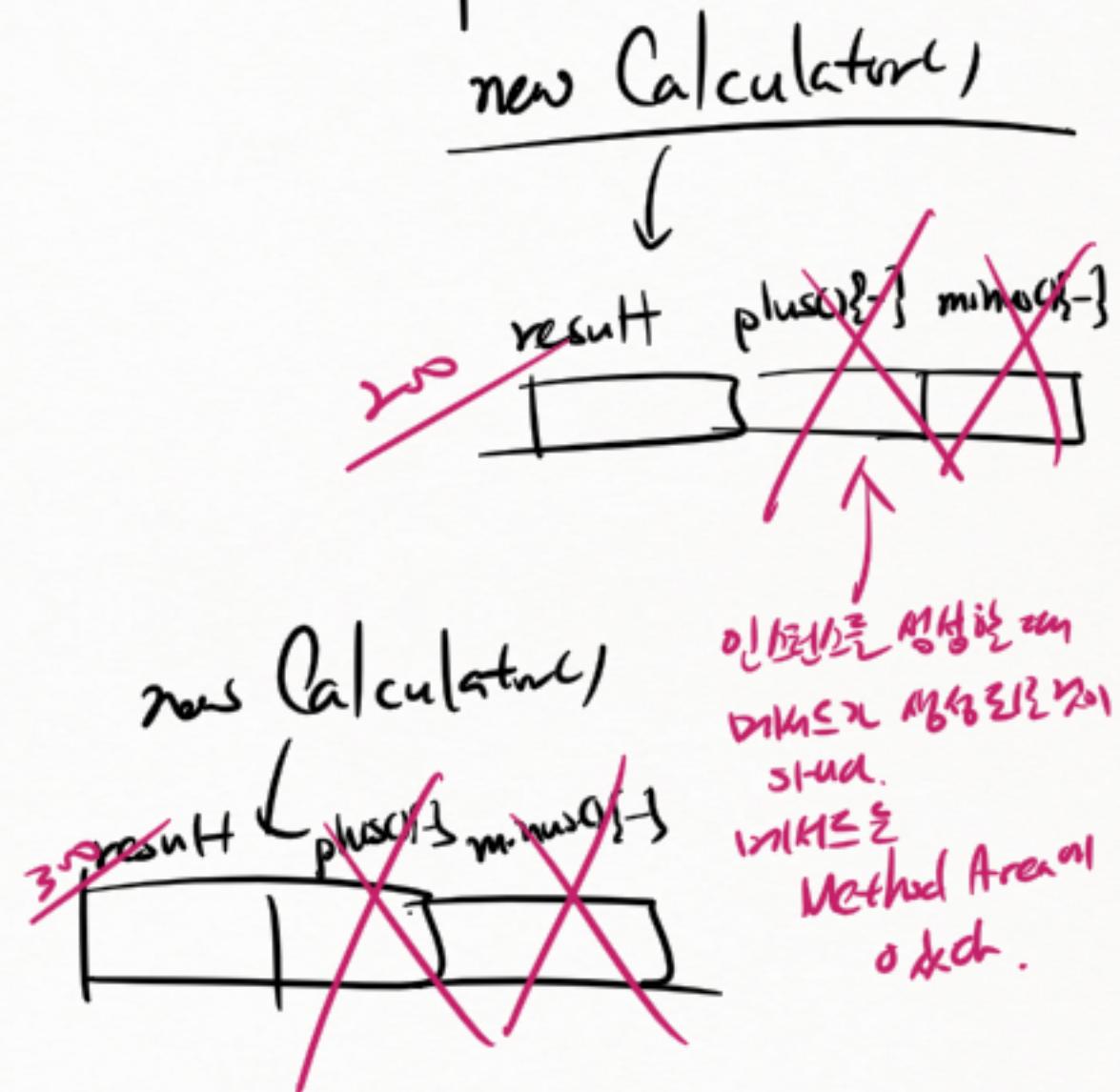
Method Area



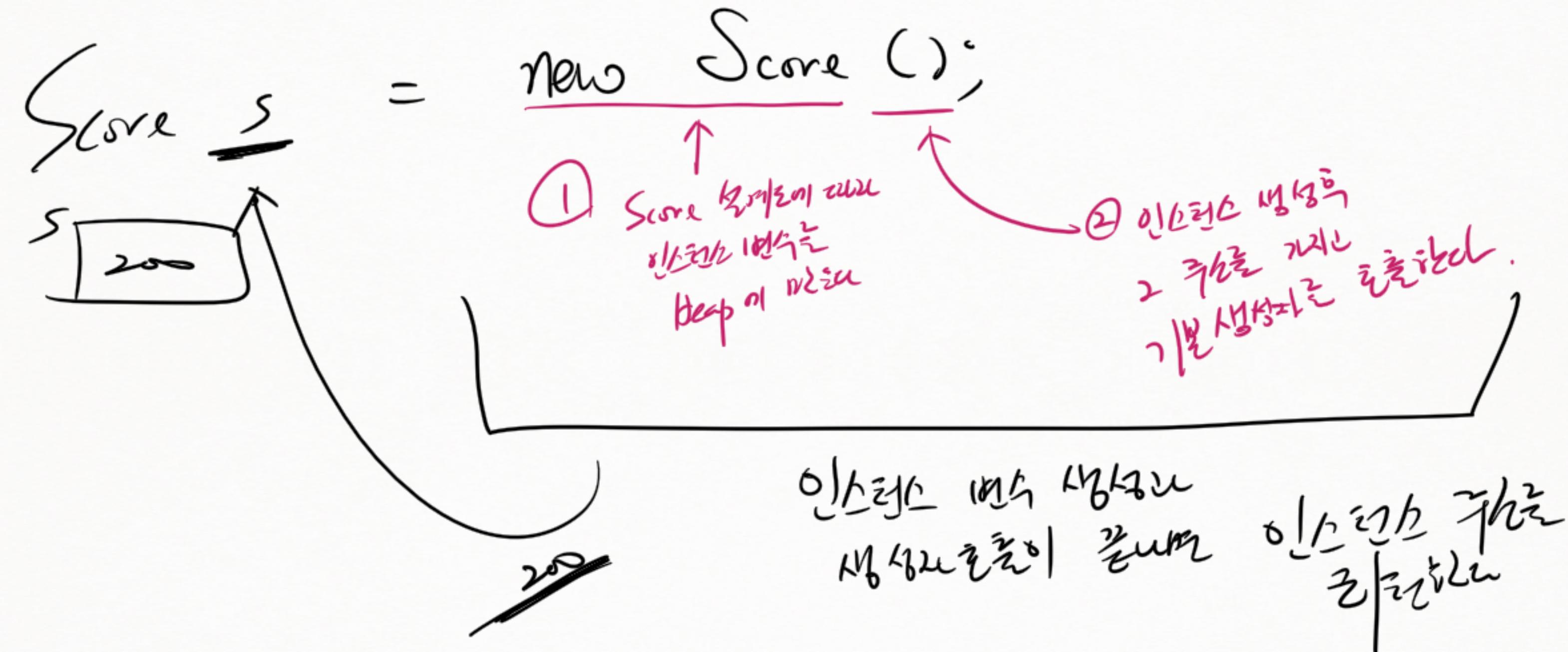
JVM Stack

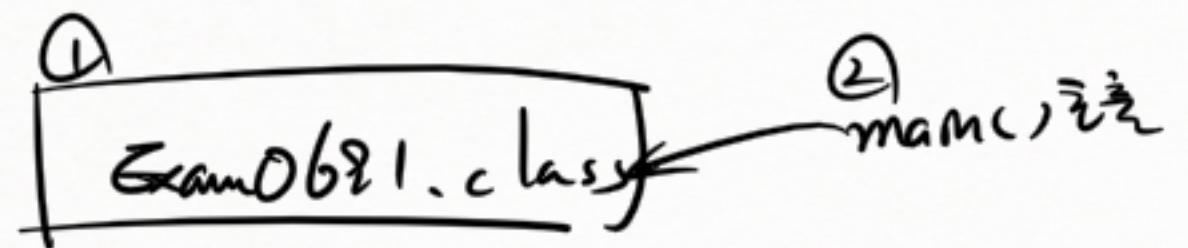


Decp



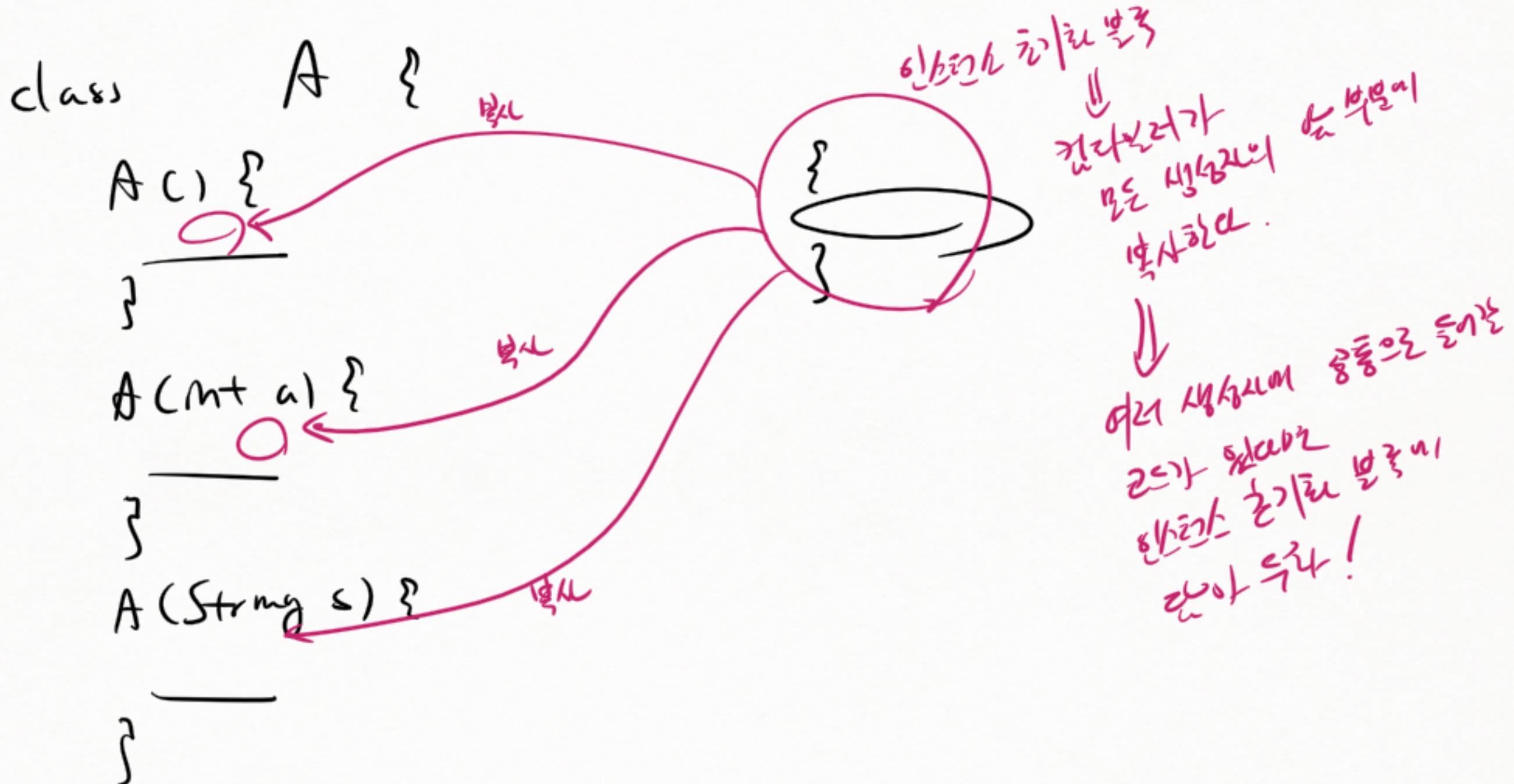
* 인스턴스 생성과 사용



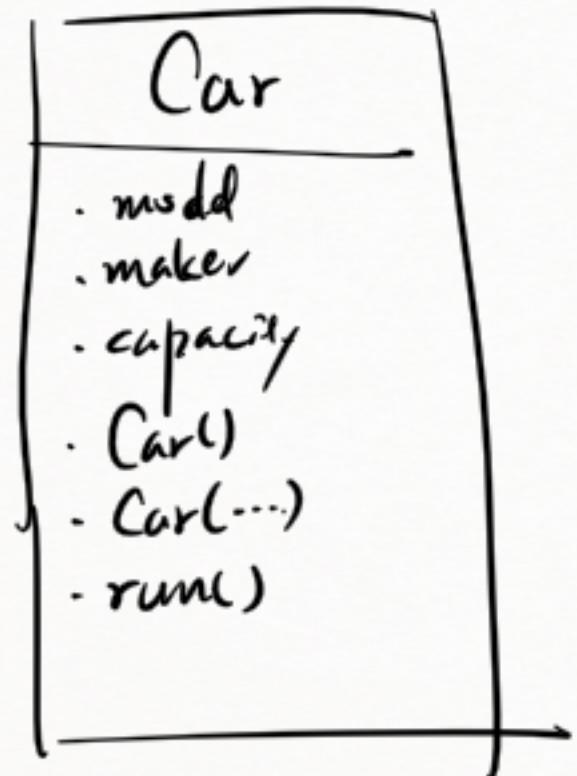


- ① A.static{}
② B.static{}
3)
2)

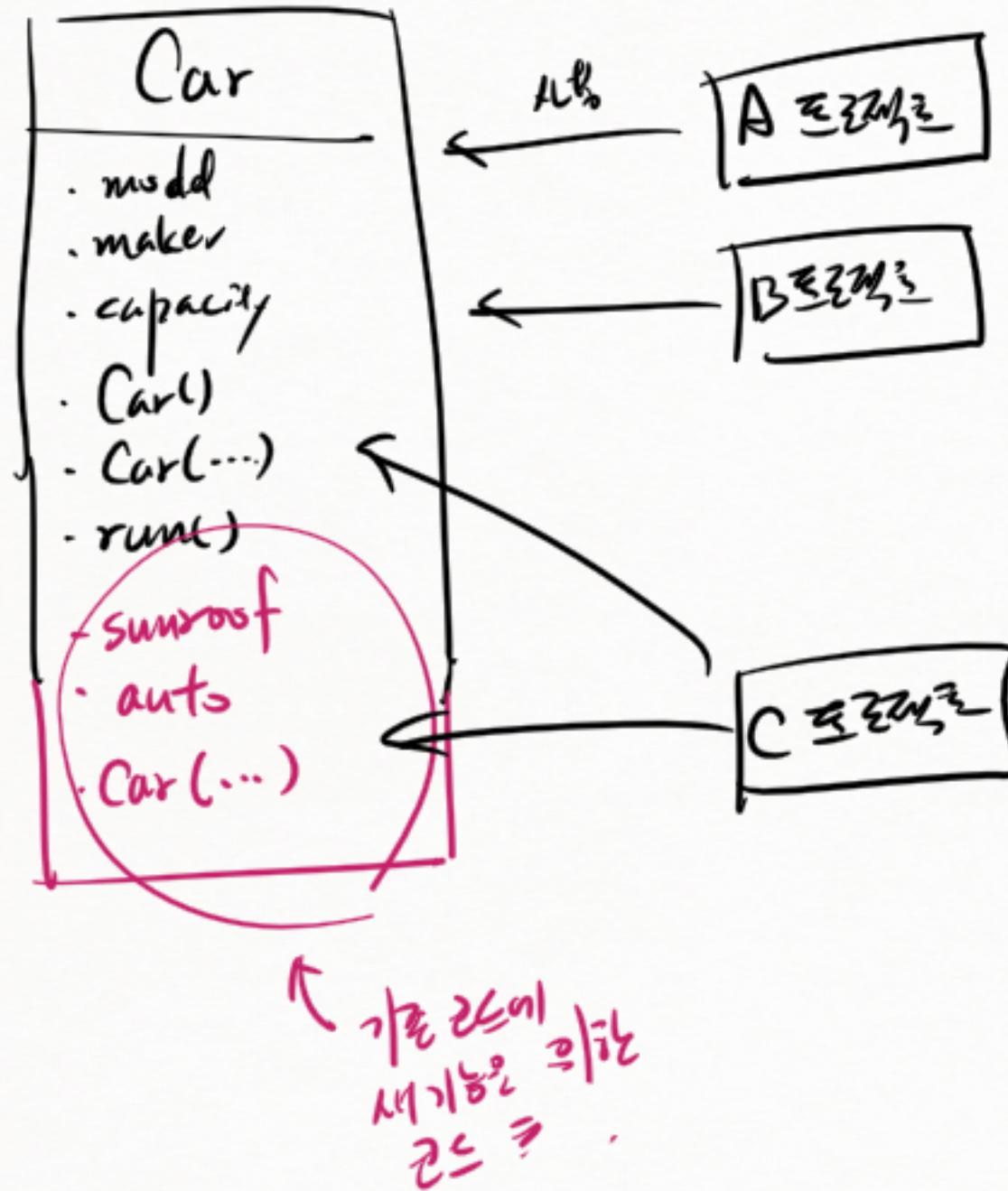
* 인스턴스 초기화 (instance initializer)



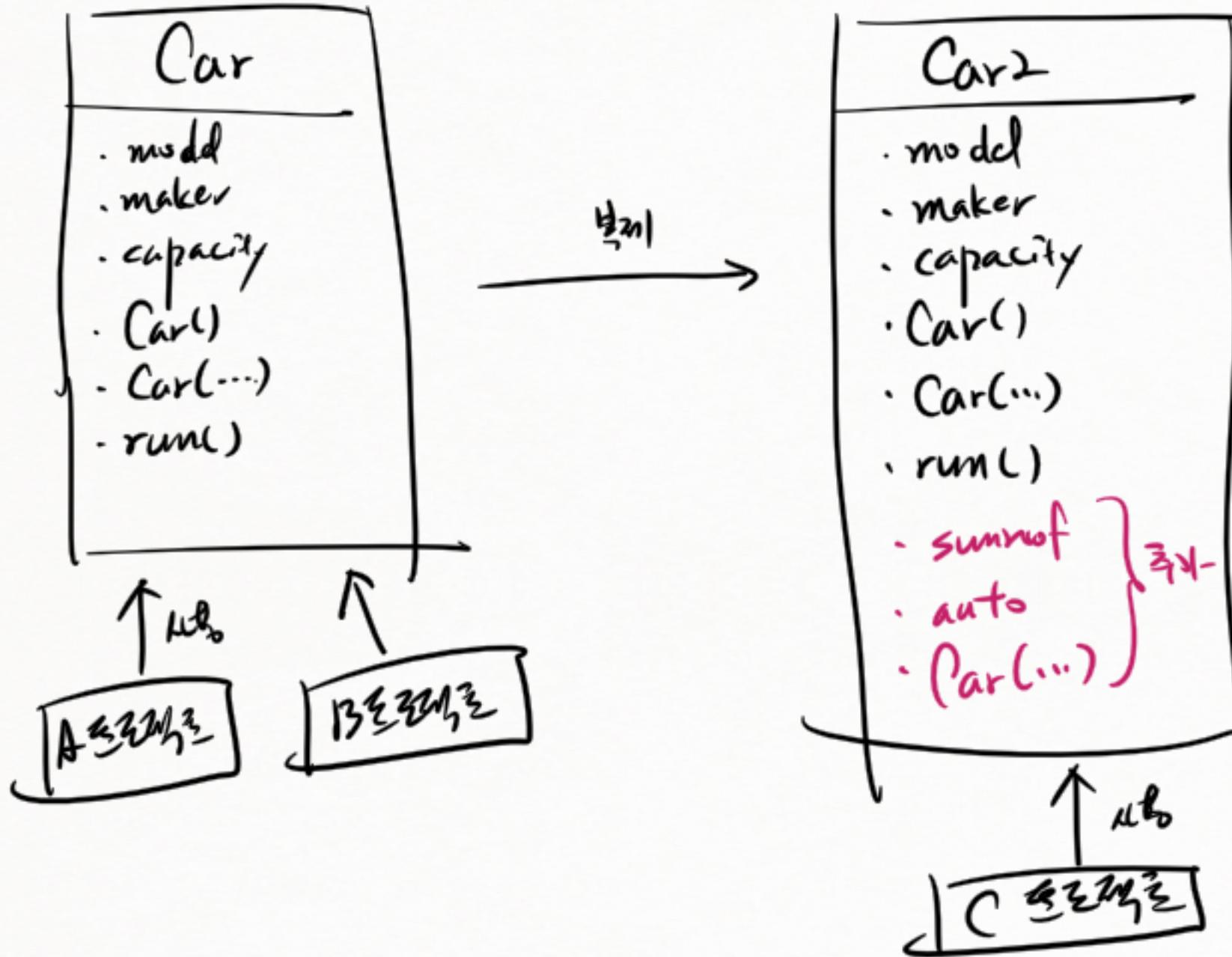
* 자료 구조 예제



* 기능을 확장하는 방법 - ① 이를테면 면적



* 자동차 클래스의 상속 - ② 자료 속성을 복제하는 경우



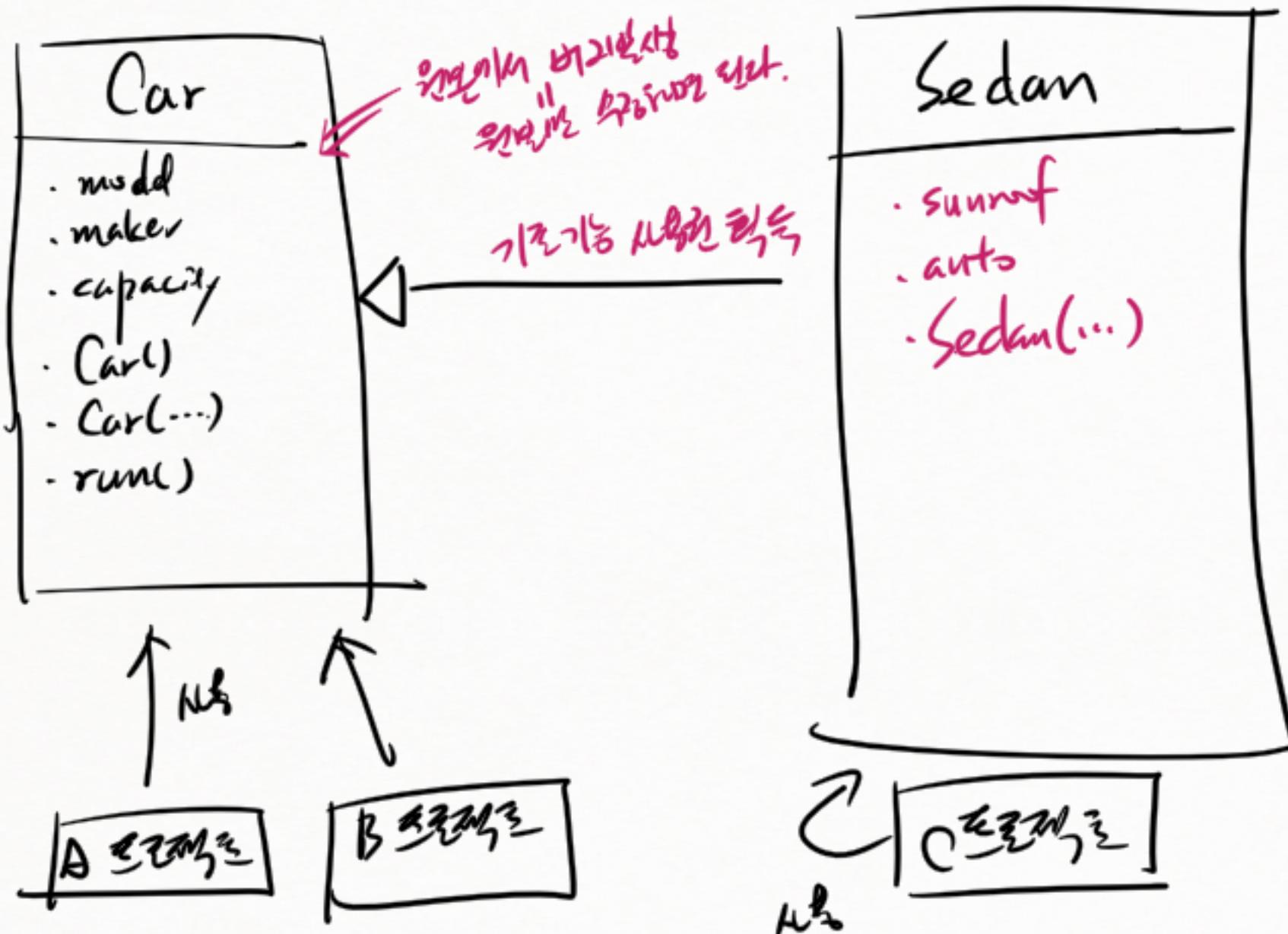
특징

- ① 자료 속성을 헤게이트로
기존 프로토 타입 패턴을 확장화.
→ 중복되는 책임

단점

- ① 복제 → 중복되는 책임
→ 가능하지 않다 → 자체적인 책임
→ 비슷하지 않다 → "
유지보수를 한다."

* 기능을 확장하는 방법 - ③ 상속을 이용



특징!

- 기존 코드를 통해 상속.
↓
이전 프로그램에 영향을 가지지 않아야
- 기존 코드를 재사용 → 비용 절감
→ 비용 초기 가격 ↓

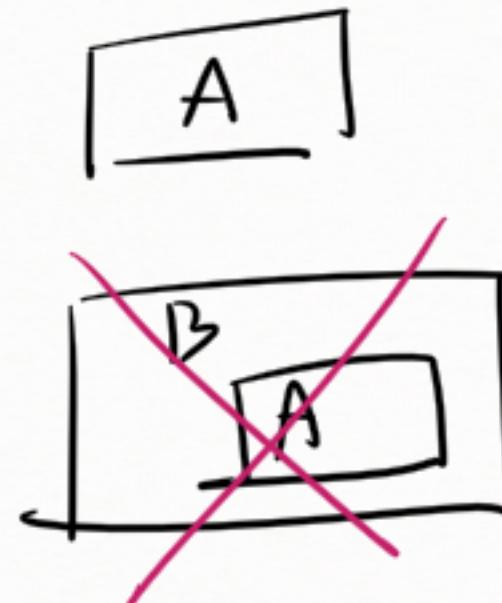
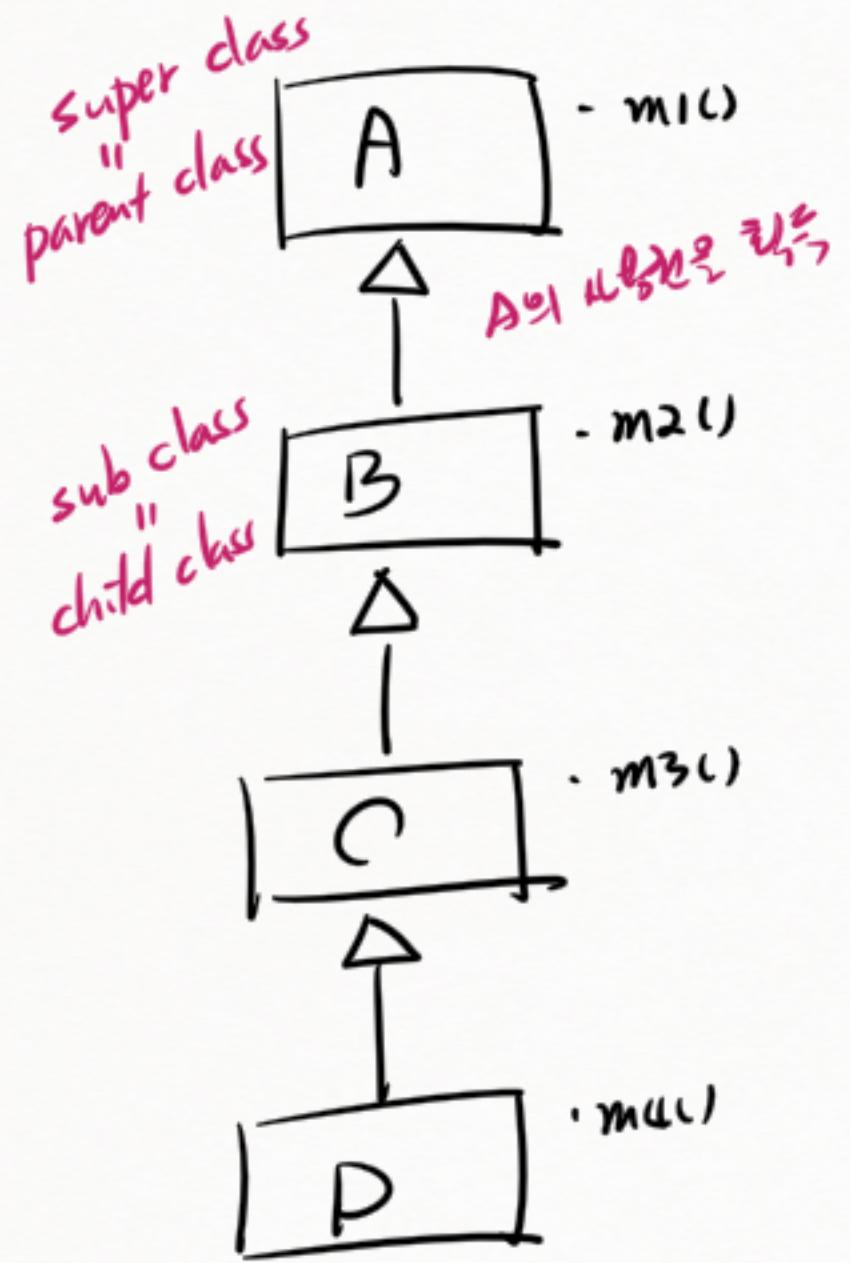
단점!

코드 중복을 없애야.
↳ 비용 수정이 필요

단점!

- 여러 단계를 거쳐야 하는
작업은 기능은 강제로 상속받을 경우가 있다

* 부기된 멤버는



A의 재정의 멤버는
허용되지 않아!

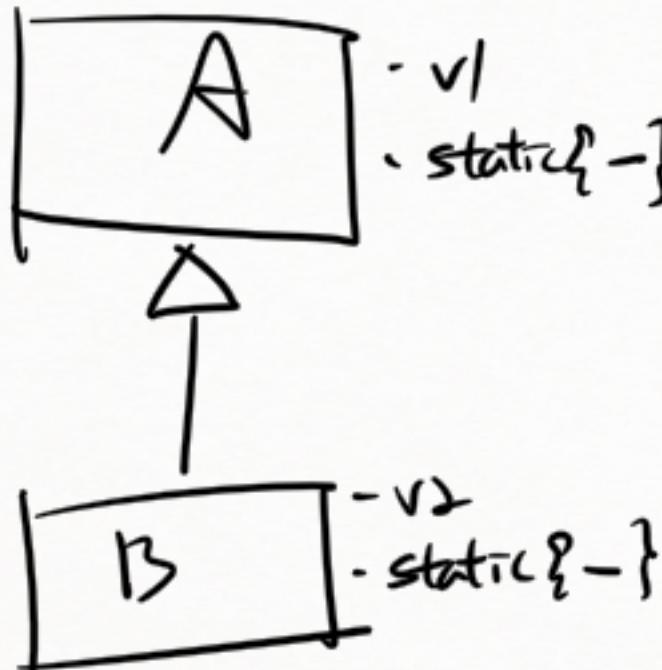
B obj = new B();

obj. m2(); // ok

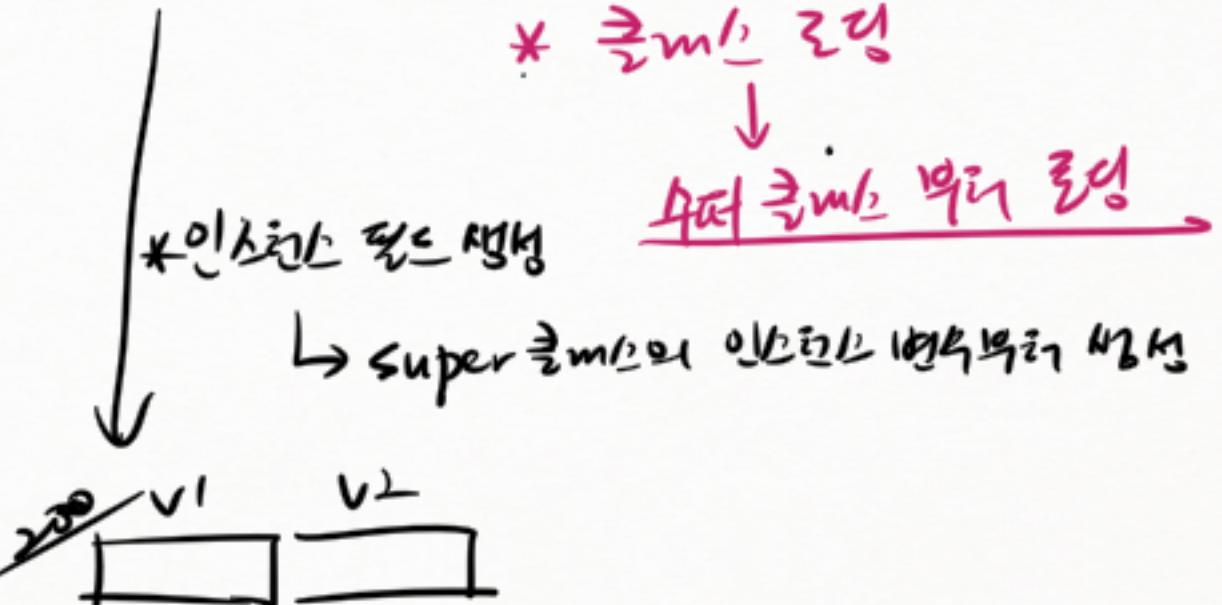
obj. m1(); ← B 클래스를 통한
↑
A의 m1()은
재정의

A 클래스의 멤버는
사용할 수 없다

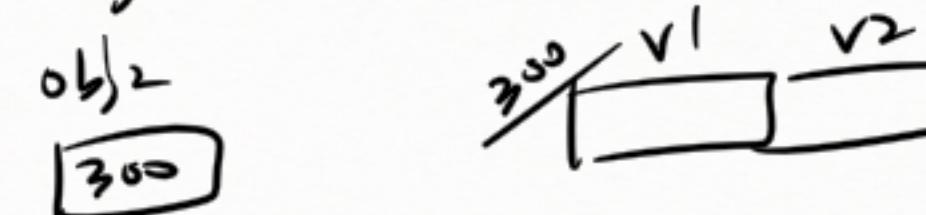
* 상속과 인스턴스 필드(변수)



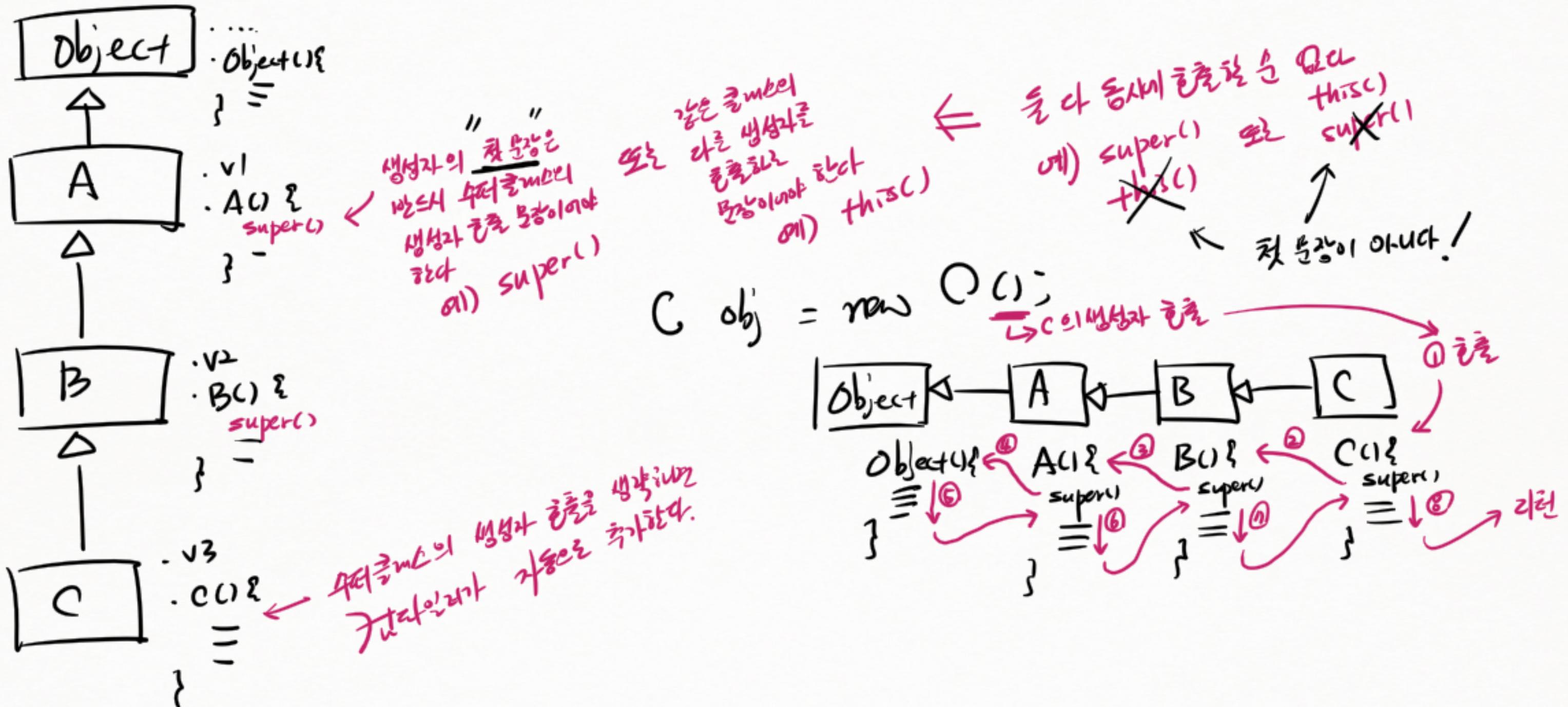
$B \ obj_1 = new B();$



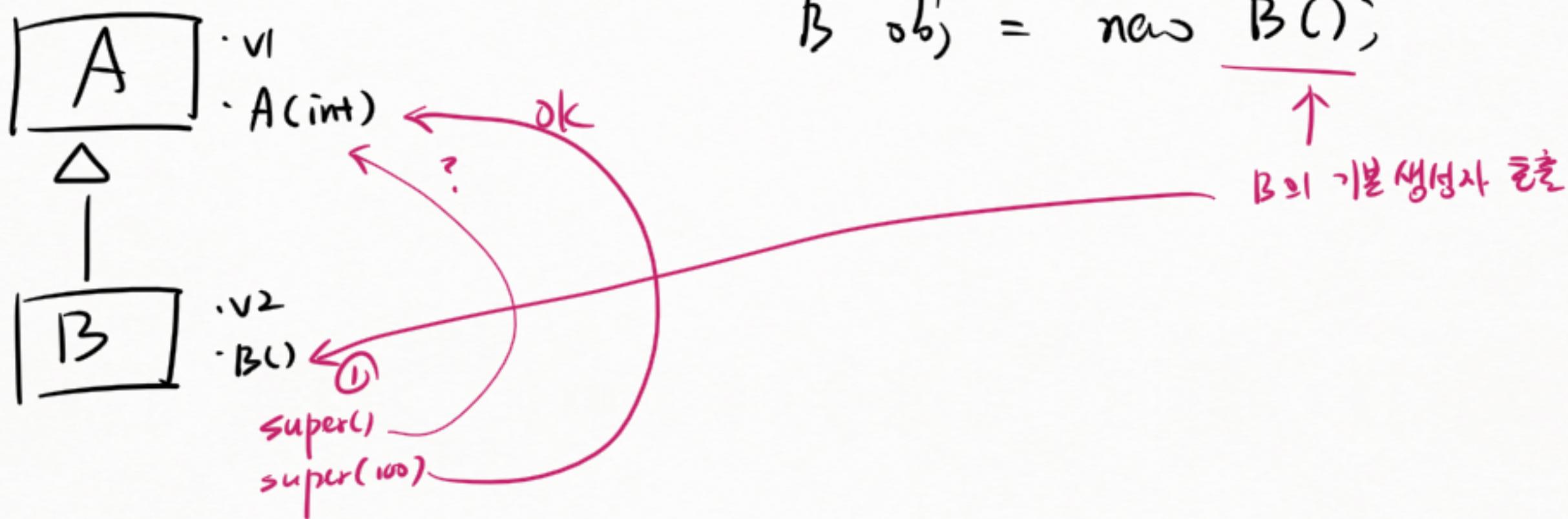
$B \ obj_2 = new B();$

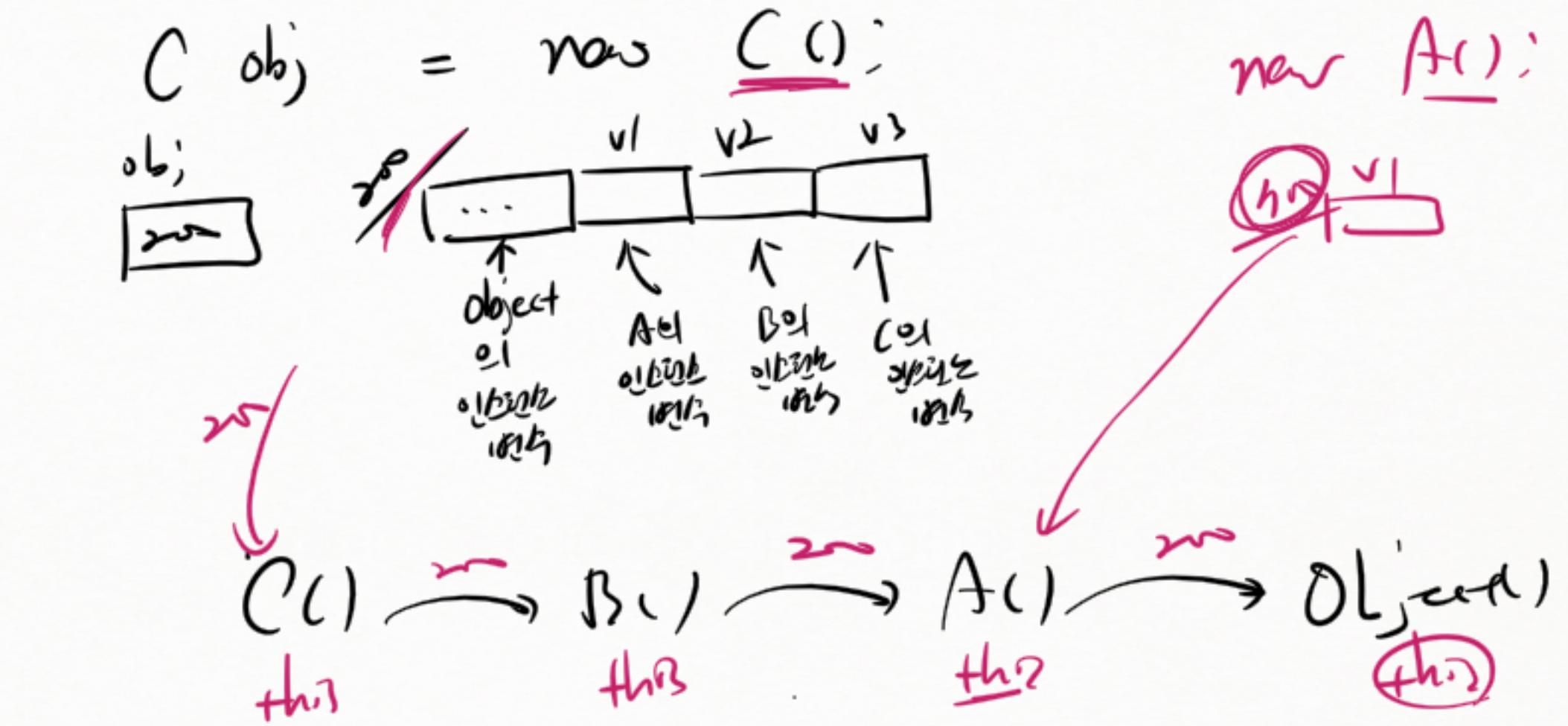


* 생성자 호출 순서

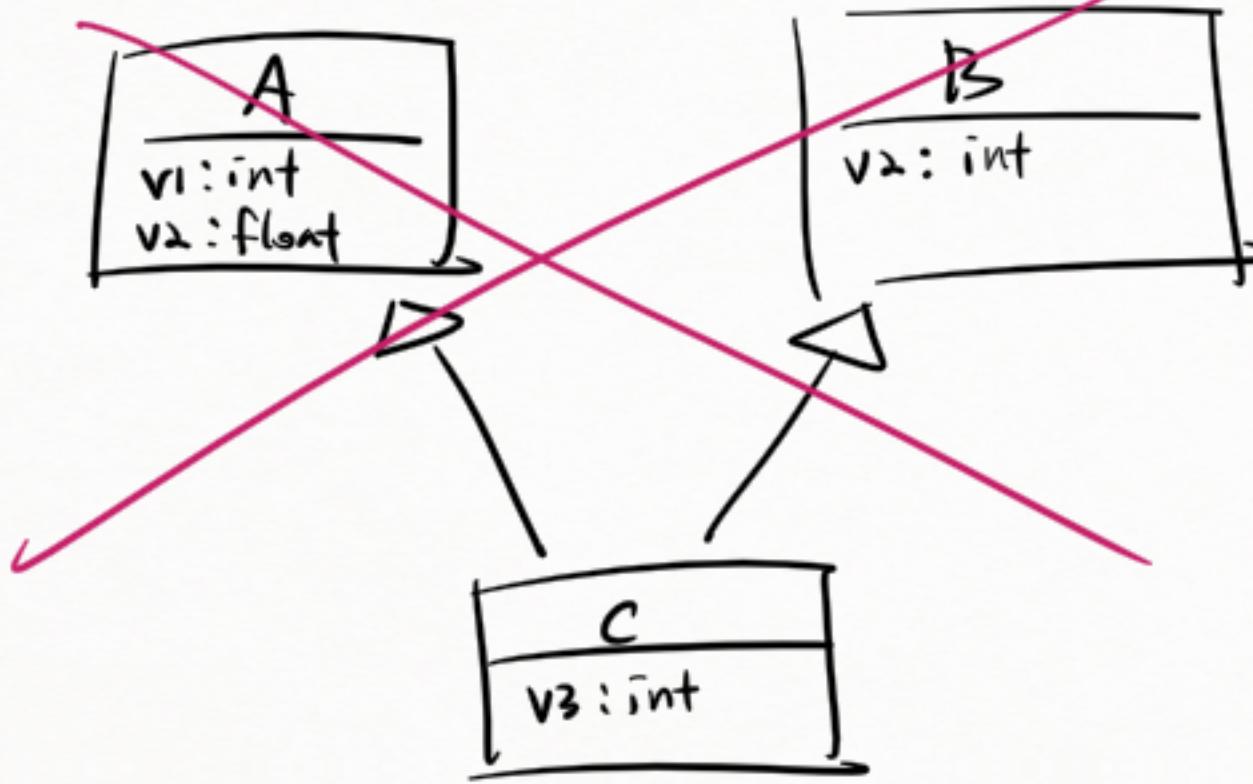


* 생성자 호출 2

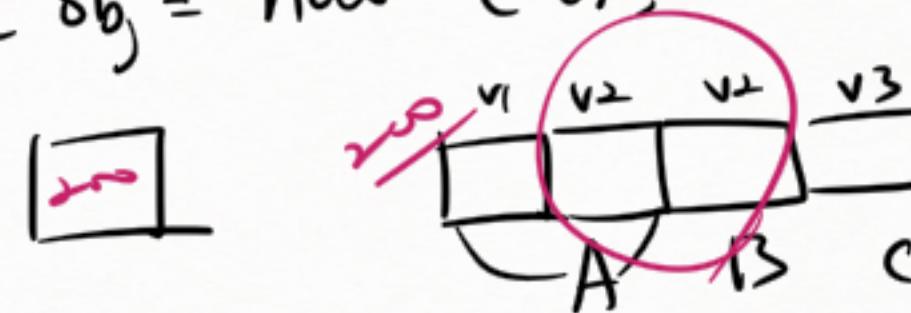




* 다음 상속 \Rightarrow 자원 얻는다.



`C obj = new C();`



$$\text{obj} \cdot \frac{\sqrt{2}}{4} = 0.2$$

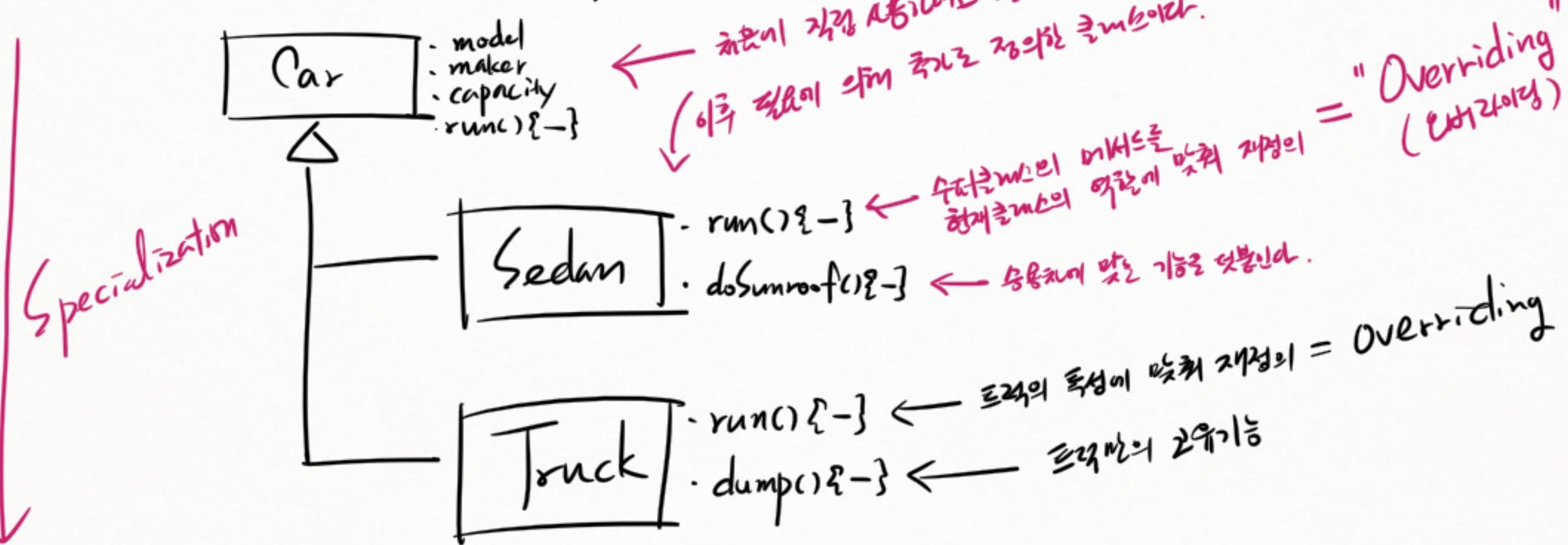
A의 연속인리
B의 1연속인리
구현으로 써 ok.

② 주민등록증
등록증을 추가

연기 10월
11월
전시회 10월

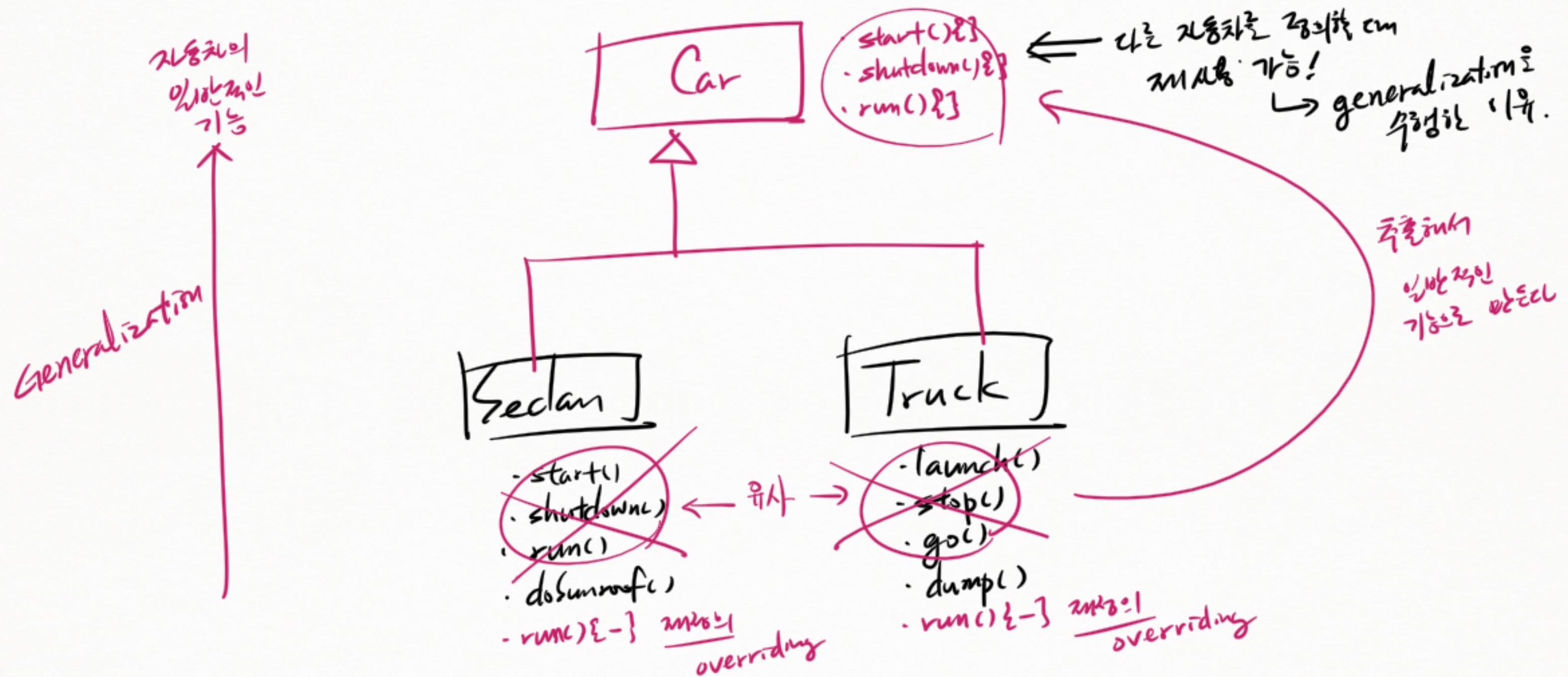
22m⁴
2118m³ 텐트을
2126 3121 3121

* ↗ : specialization (전문화)

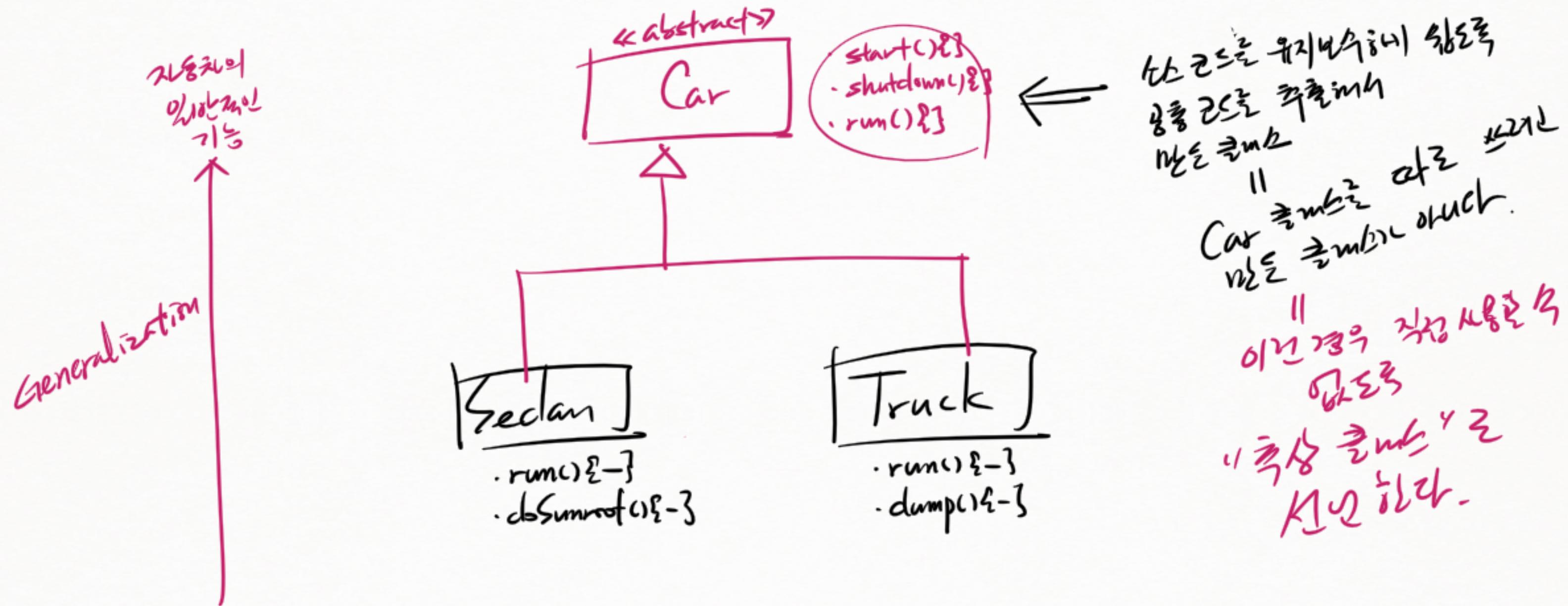


더 특별한
기능을 만들다

* 10: Generalization (일반화)



* 15주년 추상 춤극

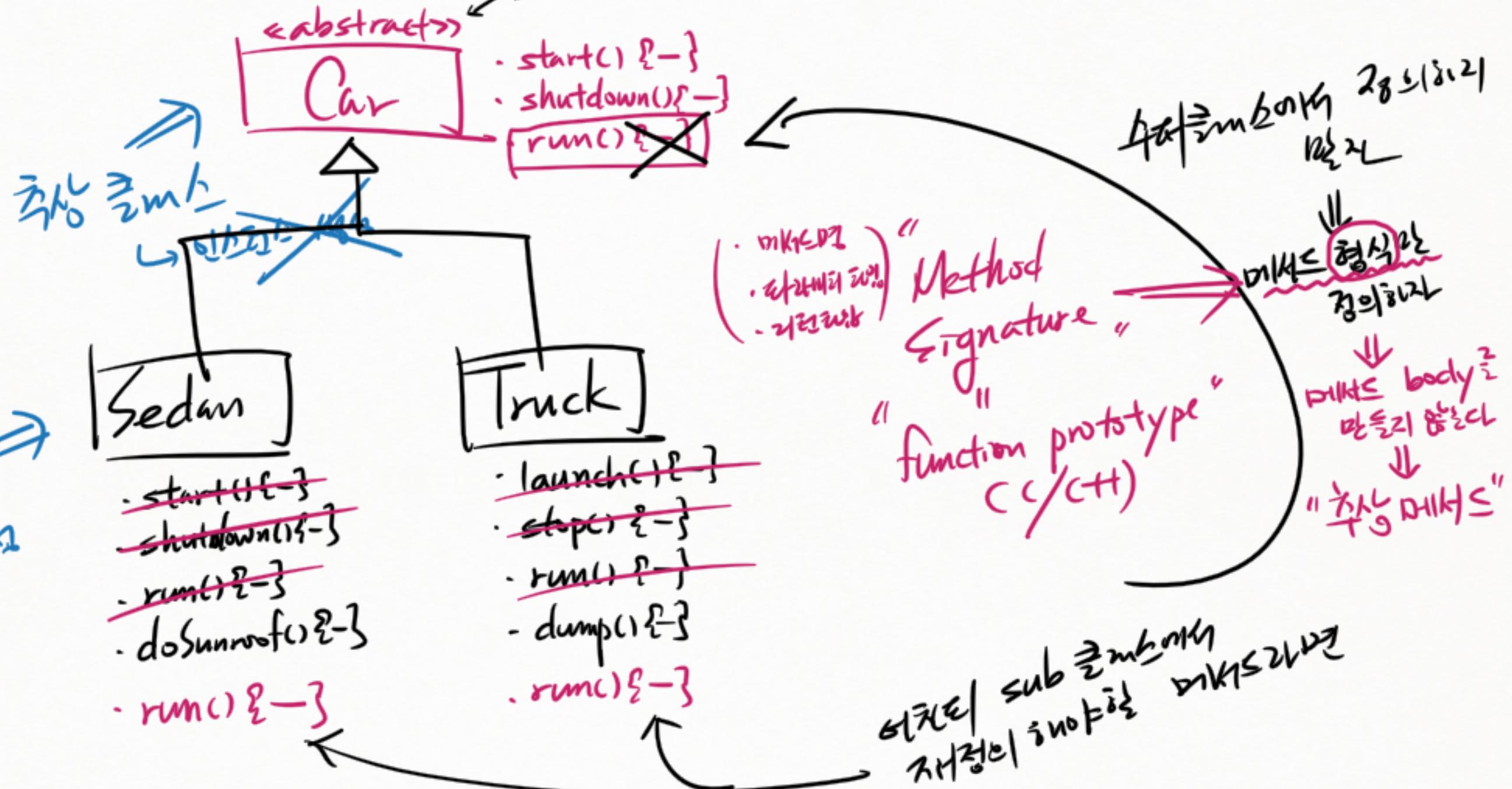


* 성별로 흐상되느 보조명칭 (stereotype)

~~new (any)~~

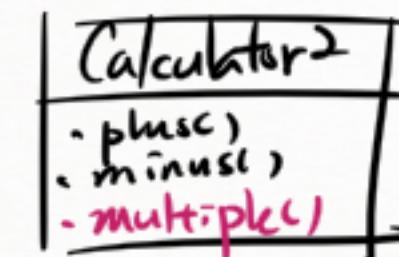
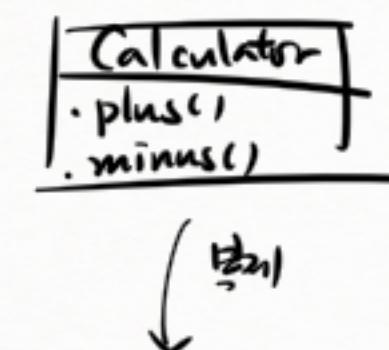
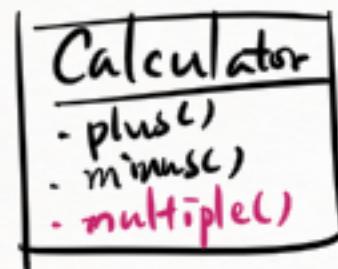
~~د. ج. سعيد~~

Concrete class
"Plain class"
↳ 인터넷 애플리케이션

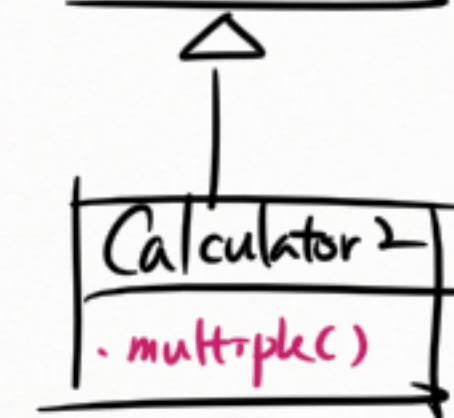
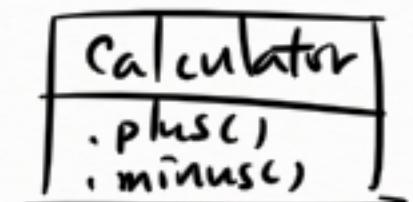


* 기능 학습법 cheat sheet

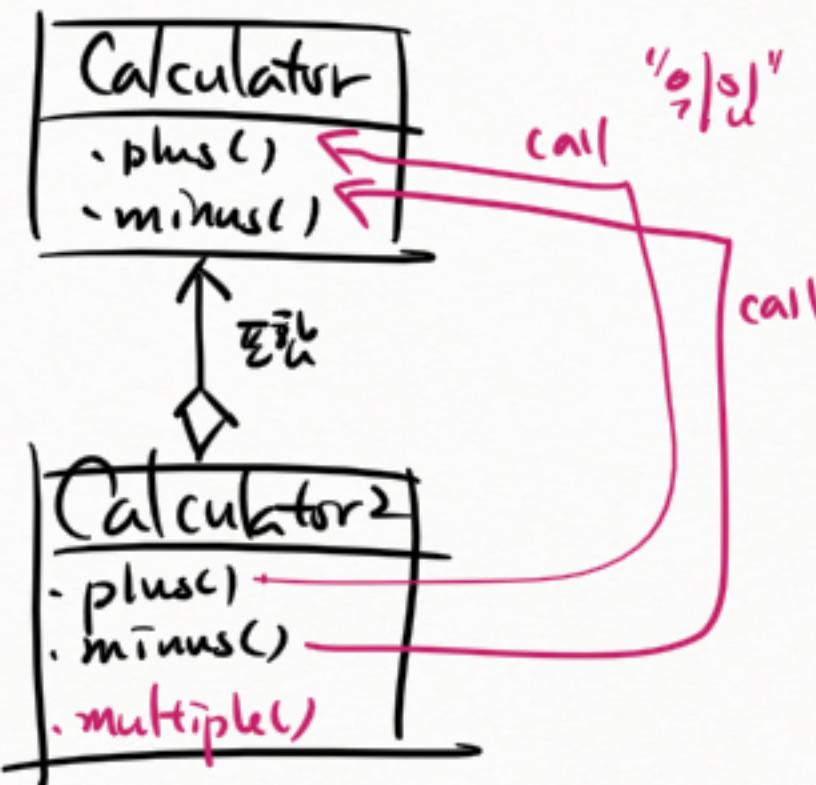
① 기능 재사용



③ 상속

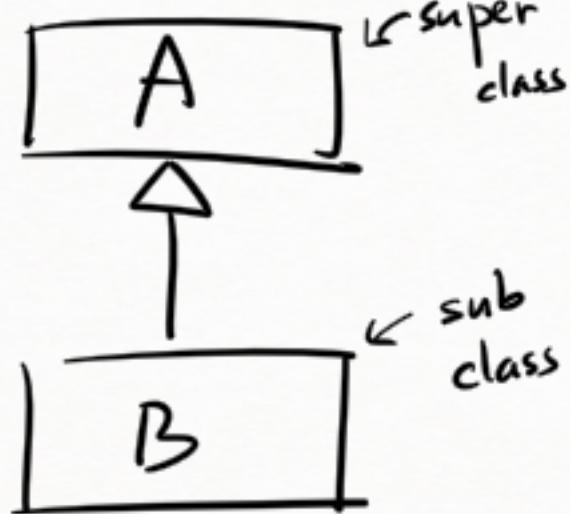


④ 오버라이드



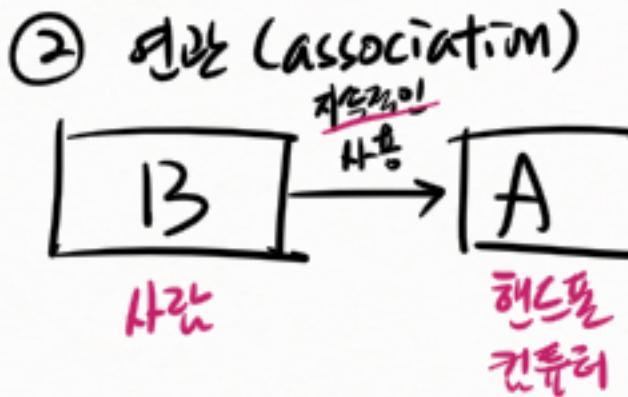
* 클래스 관계 cheat sheet

① 상속 (inheritance)



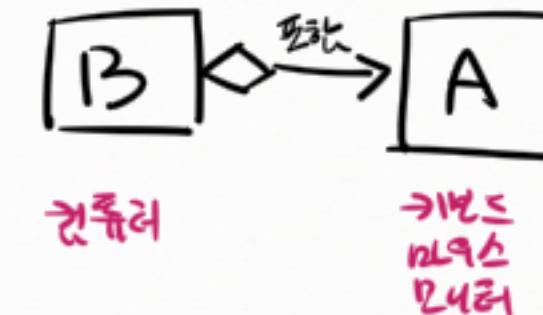
class B extends A {
≡ }

② 연관 (association)



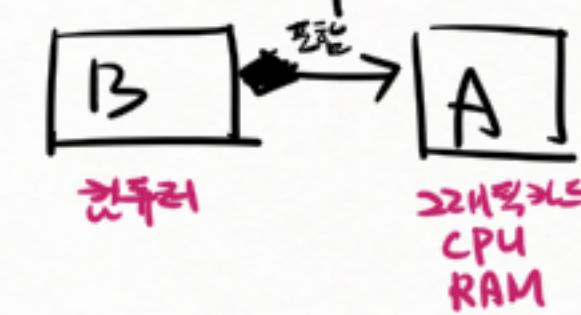
class B {
 A obj;
} ≡

③ 집합 (aggregation)



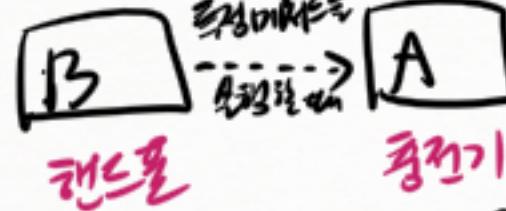
class B {
 A obj;
} ≡

④ 핍심 (composition)



class B {
 A obj;
} ≡

⑤ 의존 (dependency)



class B {
 void m(A obj){
} ≡ }

컴퓨터 ≠ 키보드
마우스
마이크

Lifecycle

컴퓨터 = $\frac{\text{CPU}}{\text{RAM}}$

Lifecycle

* String \Rightarrow 멀티 사용법.

String s1;

s1 = new String("Hello");

String s2 = new String("Hello");

String x = "Hello";

String y = "Hello";

String 풀(Pool)의
String 인스턴스는 같은
"중복 생성하지 않는다"

JVM Stack

s1 [200]

s2 [300]

x [400]

y [500]

String Constant Pool

500 value hash
Hello —

String 풀의
인스턴트

Heap

200 Hello

300 value hash
Hello ...

String 풀의
인스턴트

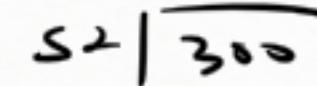
* String.intern()

String s1 = new String("Hello")

String s2 = s1.intern()

String s3 = "Hello"

JVM Stack



↑
s1의 주소를 문자열을 가진
String 객체는 String Pool에서 찾지 못한
이 생성되는 순간

만약 두번,
300

Heap



String Pool

