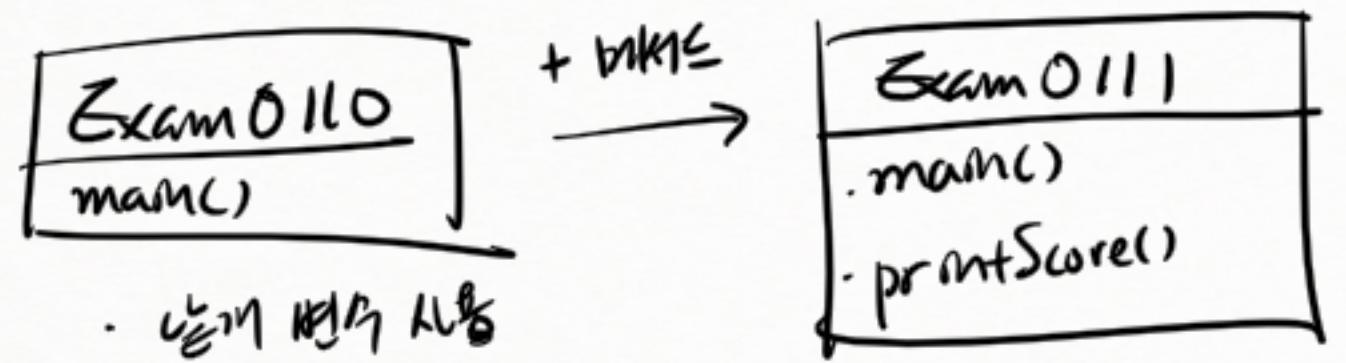


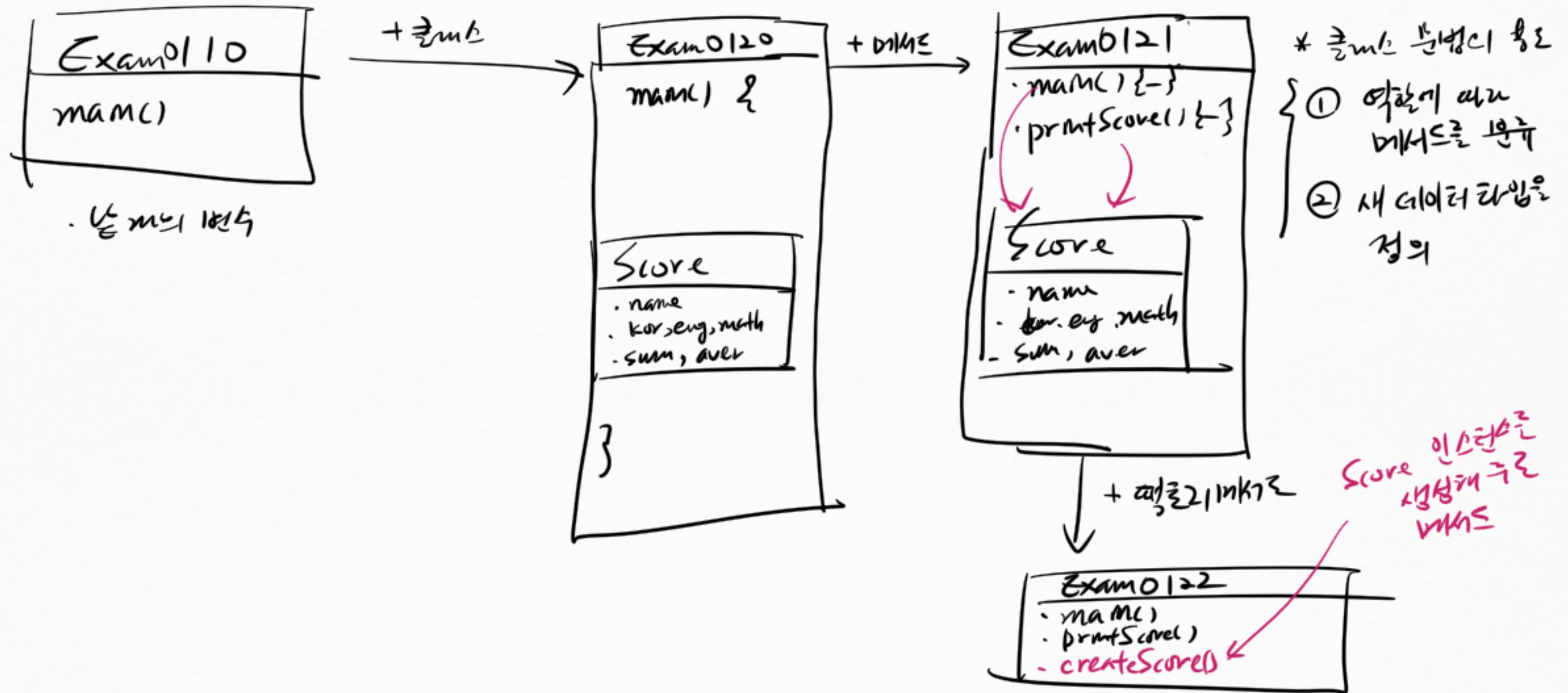
* 재미스 문법 활용 예



• 놓기 허용 사용

- DPLS 문법 활용
- DPLS 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 = "—"'

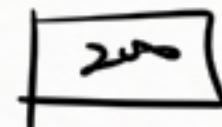
s1. kor = 100 -

:

* 예외처리 10주차

Score[] arr = new Score[3];

arr



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

arr[0] = new Score();

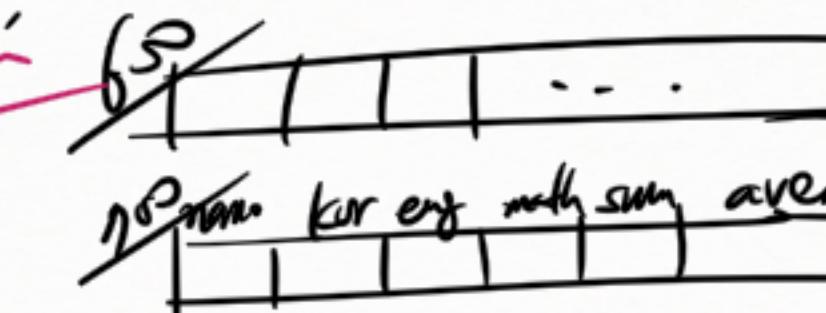
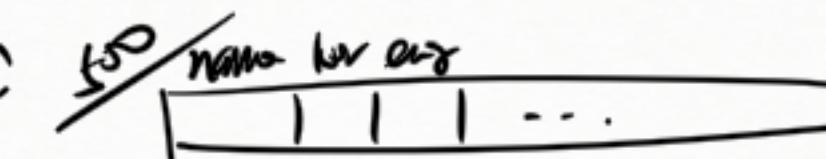
* 예외 처리 ← 자동으로 null로 초기화된다
* 접근할 수 있는 멤버가 초기화되면 좋다.

arr[1] = new Score();

arr[2] = new Score();

arr[3] = new Score();

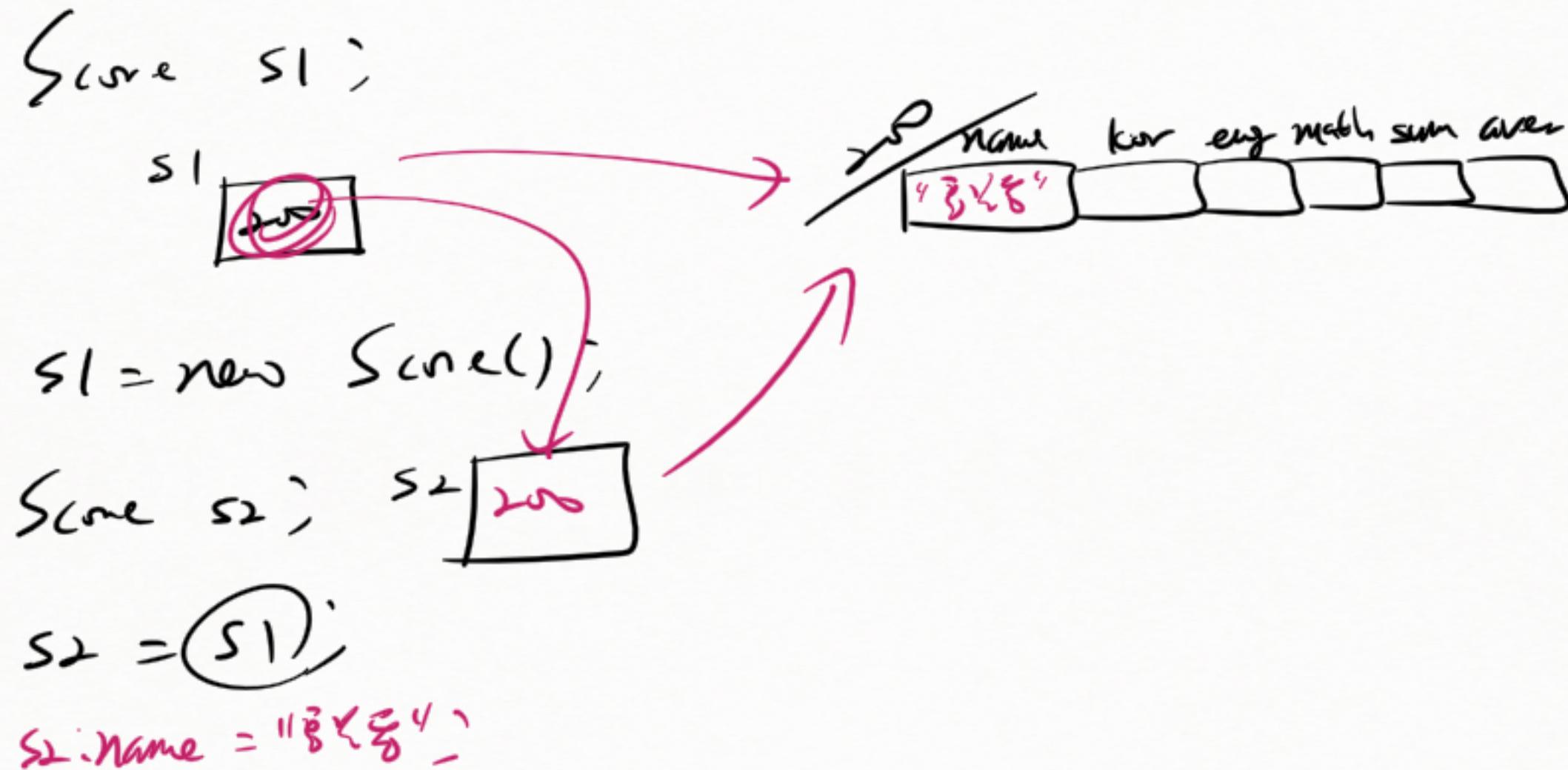
* ArrayIndexOutOfBoundsException



new Score();

Score는 null이 선언된 때
Heap에 할당되는 순간
기억으로 만든다.

* 리터럴과 인스턴스



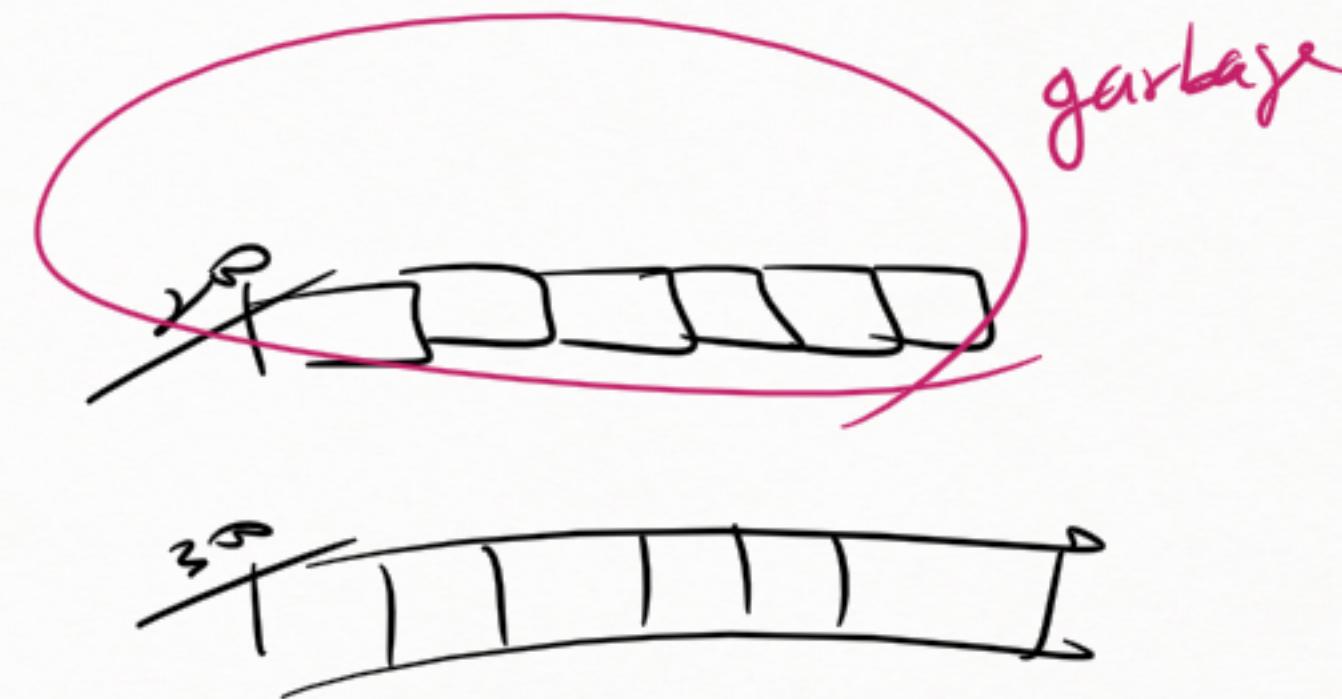
* 7110121 (garbage)

Score s1;



s1 = new Score();

s1 = new Score();

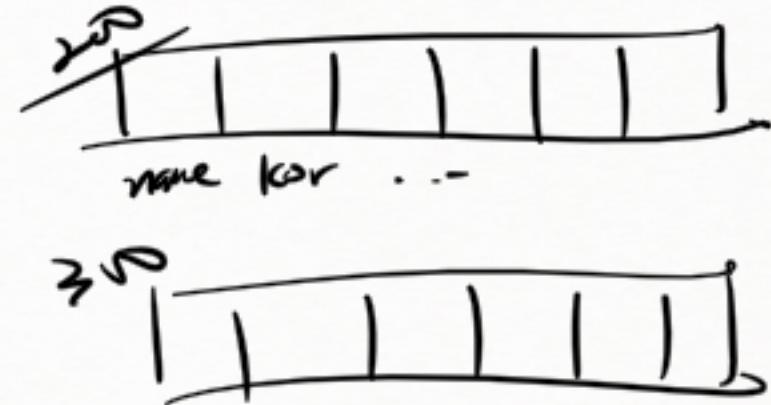


* 리터럴은 카운트와 관계

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

s1

s2

JVM이 처리

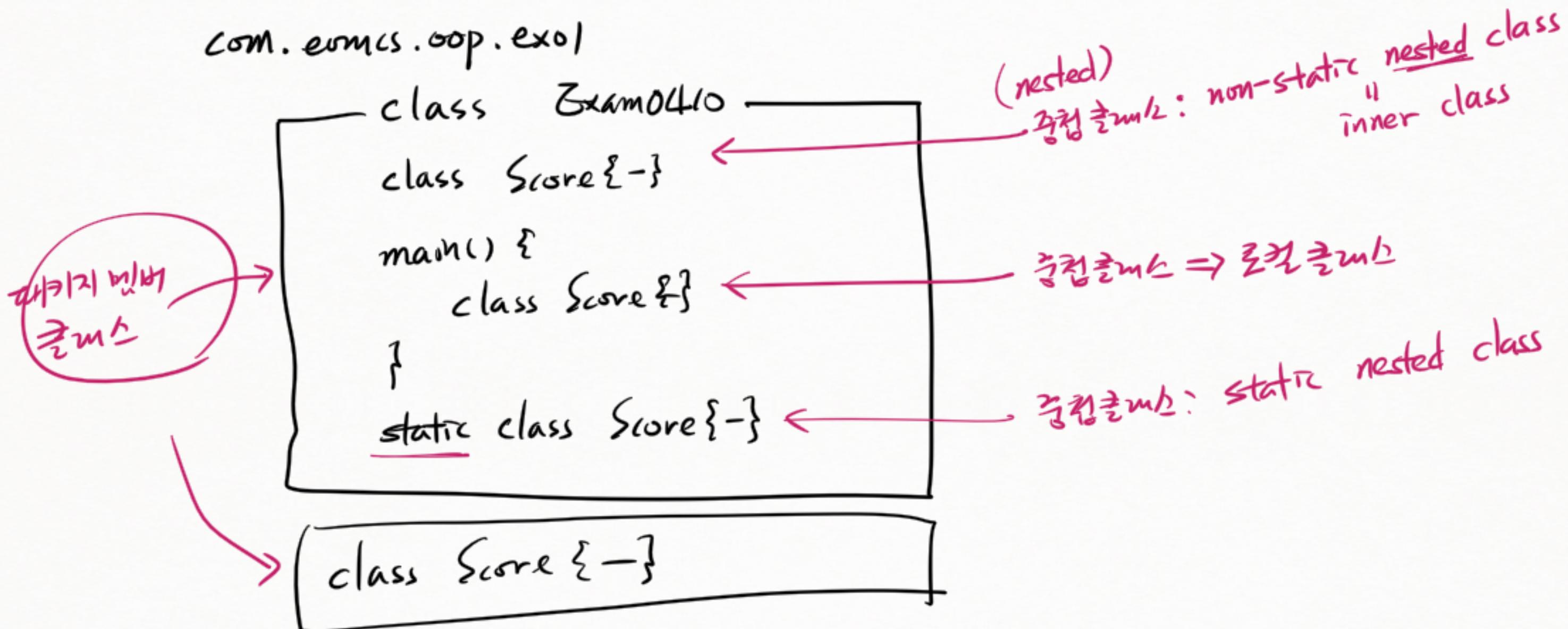
리터럴은 카운트 관계

리터널은 카운트가
0인 경우
"garbage"라
한다.

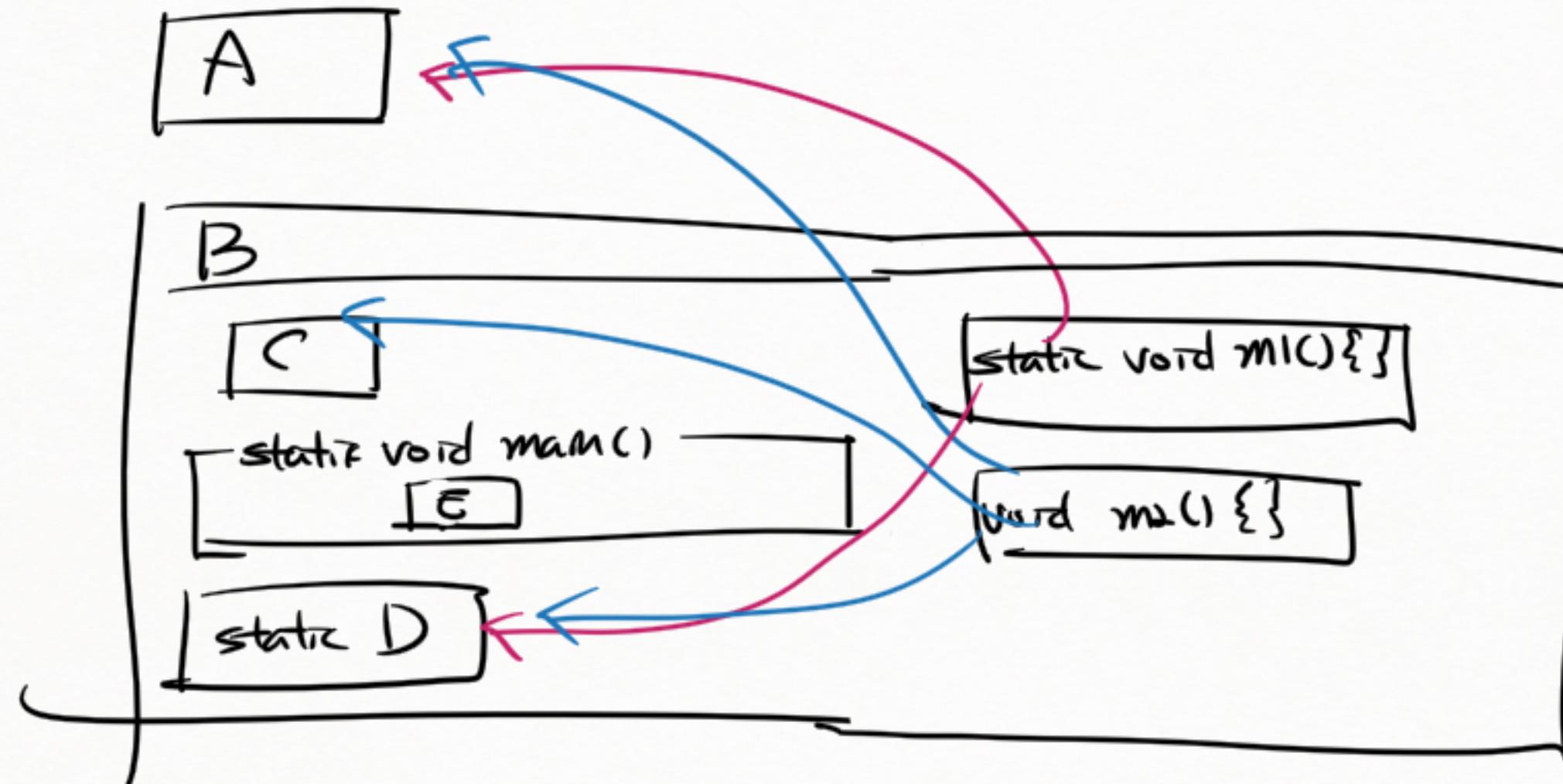
인스턴스	참조 횟수
200	X 2
300	X 0

* 클래스 구조

com.eunics.oop.ex01

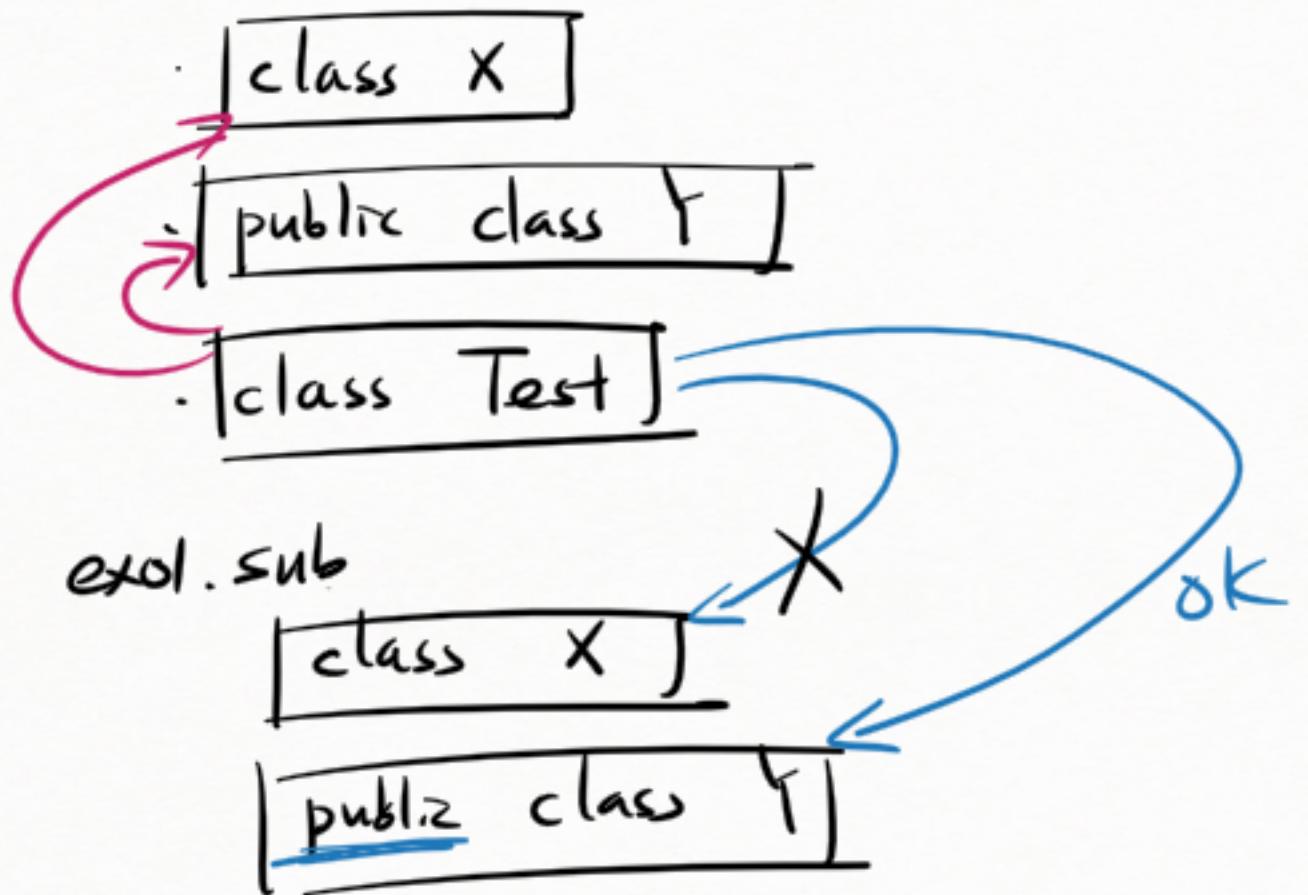


* static member non-static member



* 공개 멤버 필드

ex01



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



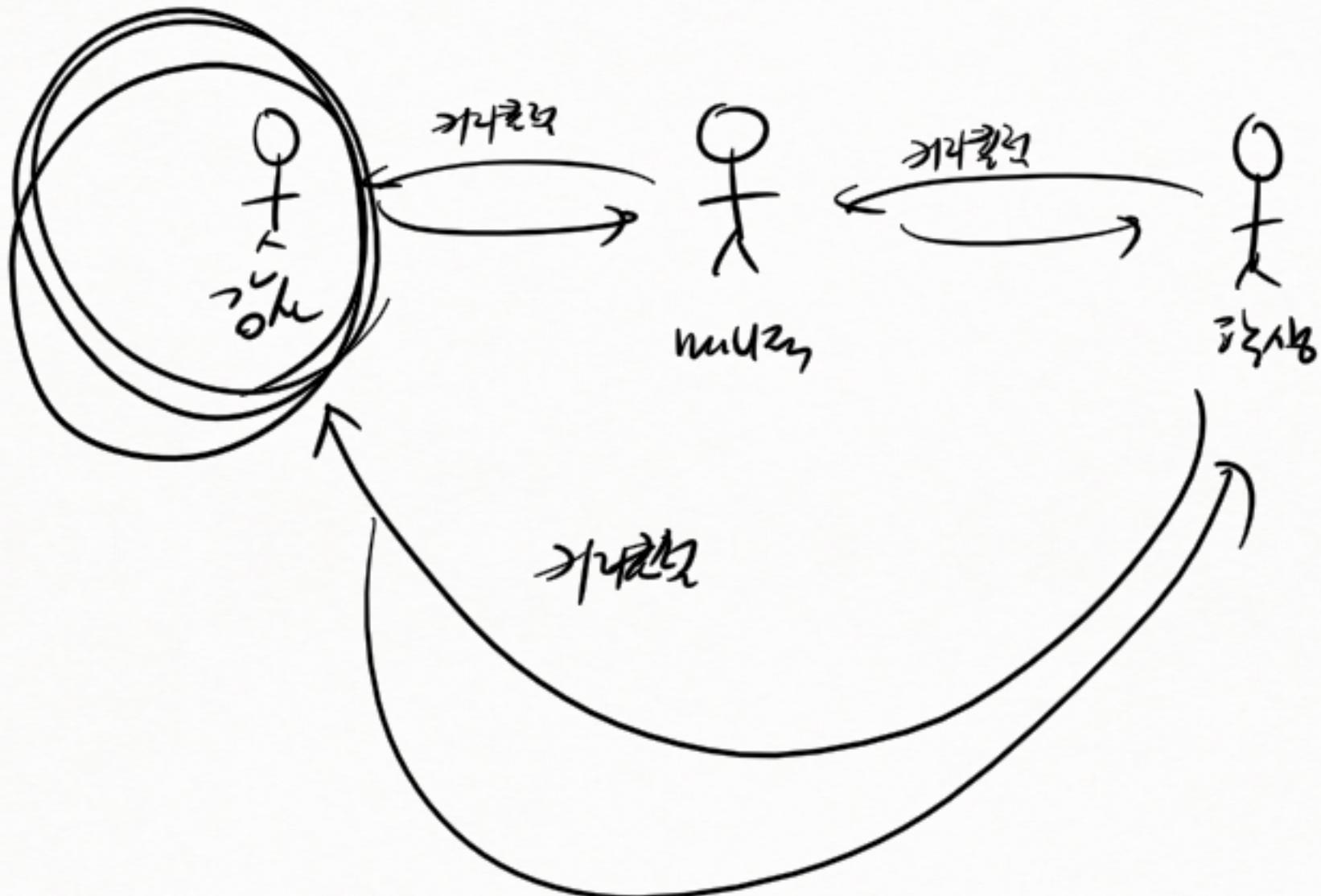
s1.name = "홍길동"

* optimizing(최적화) vs refactoring(재구조화)

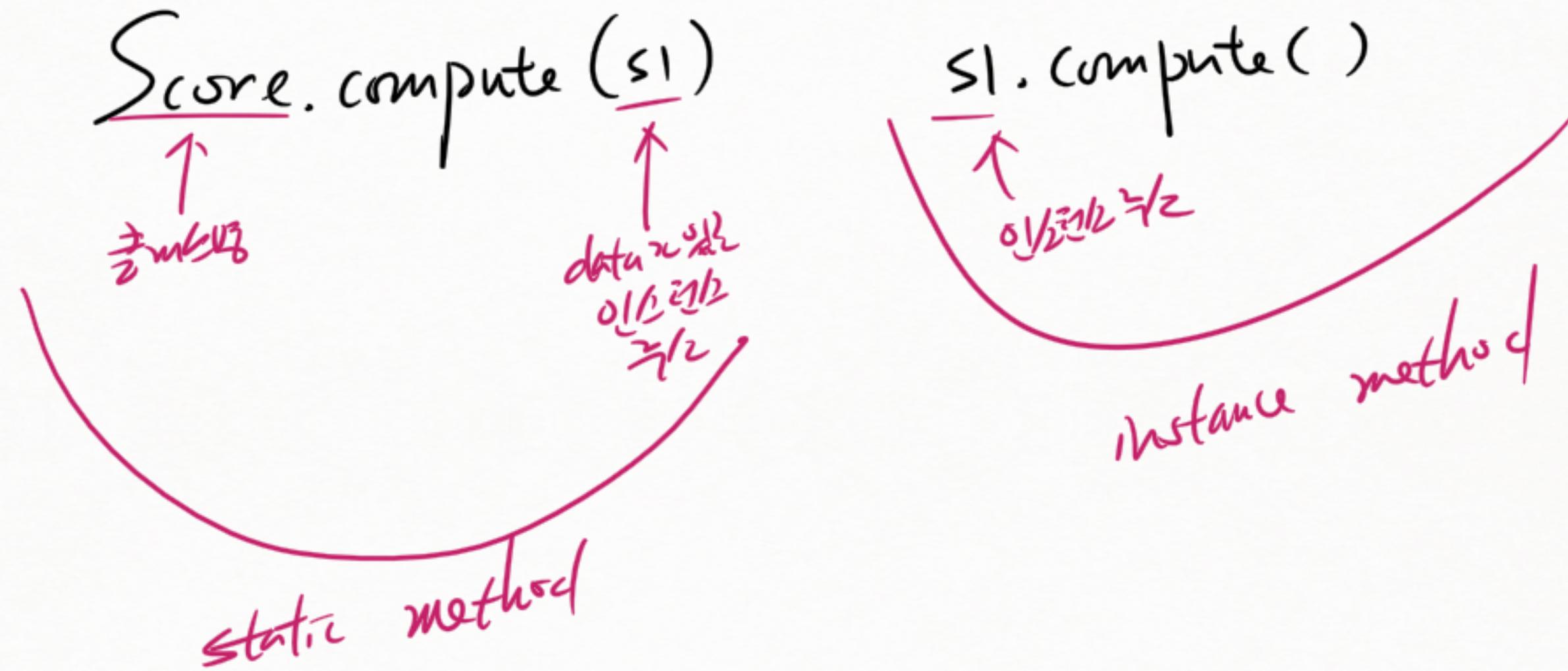
- 속도↑
- 유지보수 품질↑
- 속도↓

- 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

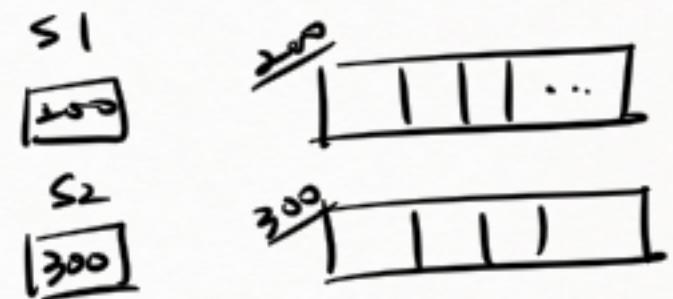
j ++ ;
 operand
 (인자)
 operator (연산자)

j ++;

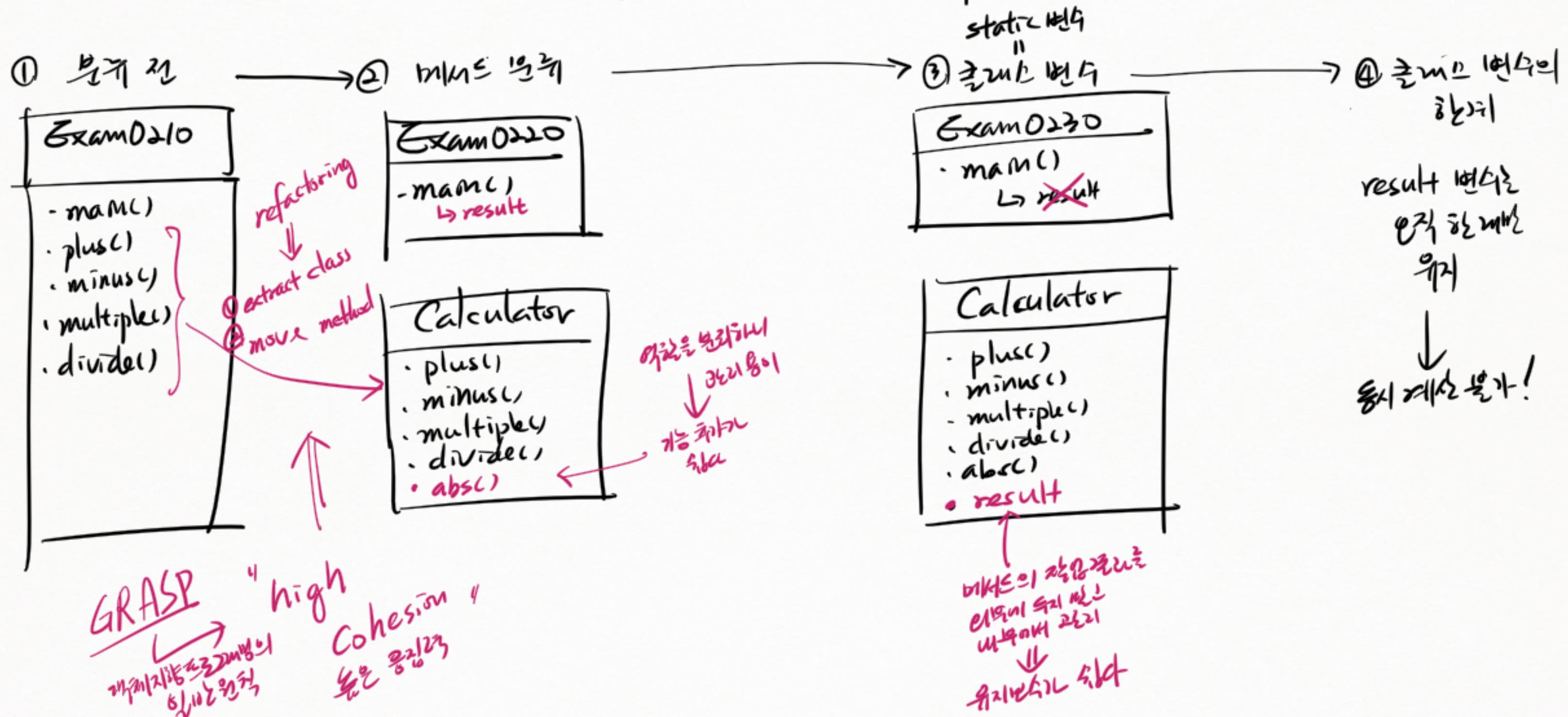
Score s1, s2 ;
 s1 = new Score();
 s2 = new Score()

인자
 ↓
s1. compute()
 ↑
 operand
 ↑
 operator

s2. compute()

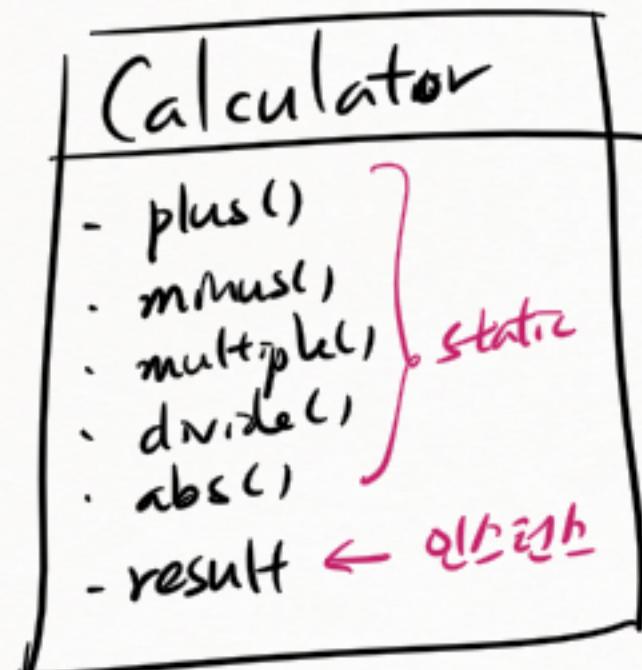
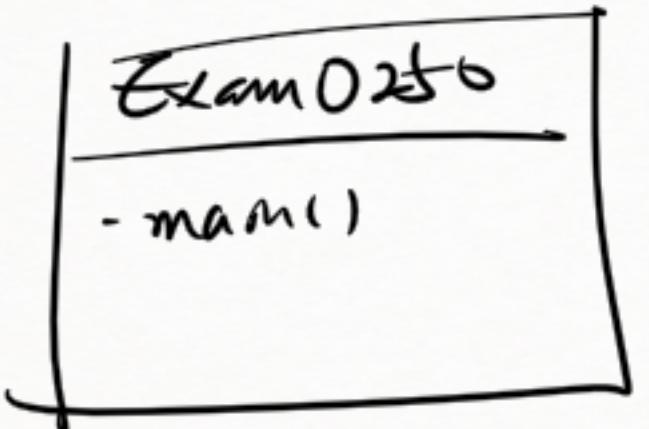


* 3단계 분기점: MMR을 끝난 분기점이



* 인스턴스 멤버 함수: 인스턴스마다 다른 결과를 반환

→ ⑤ 인스턴스 멤버 변수



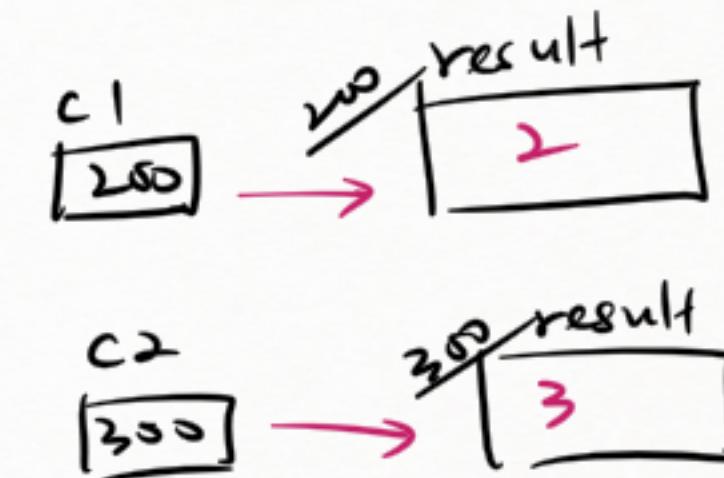
```
Calculator c1 = new Calculator();
Calculator c2 = new Calculator();
```

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

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

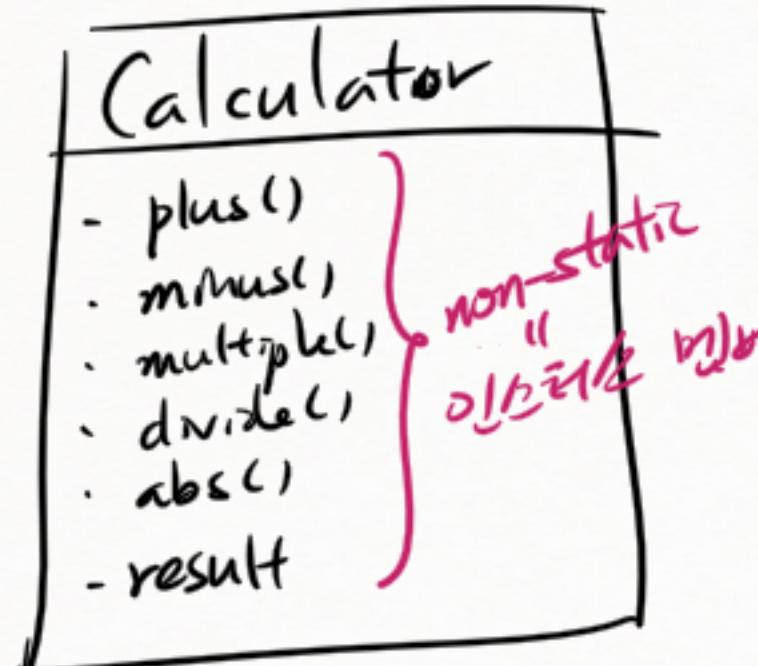
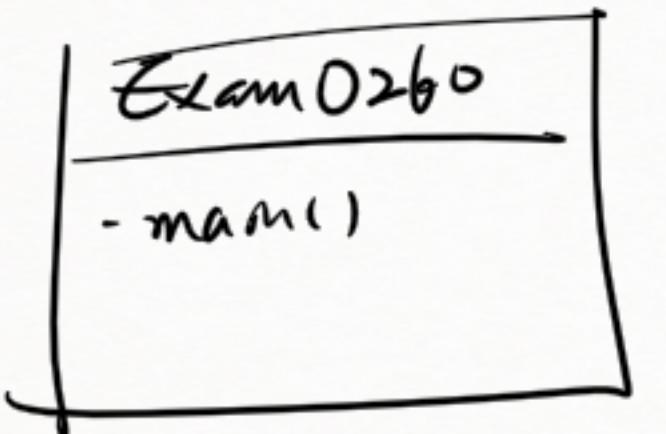
:

↑
각 인스턴스는 자신의
result 멤버 변수에
0으로 초기화된다.



* 인스턴스 멤버 변수: 인스턴스마다 다른 값을 갖다

→ ⑥ 인스턴스 변수

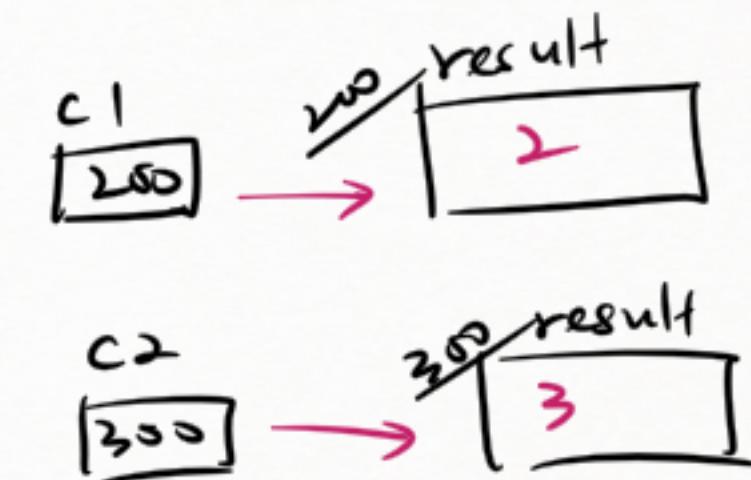


Calculator c1 = new Calculator();
Calculator c2 = new Calculator();

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

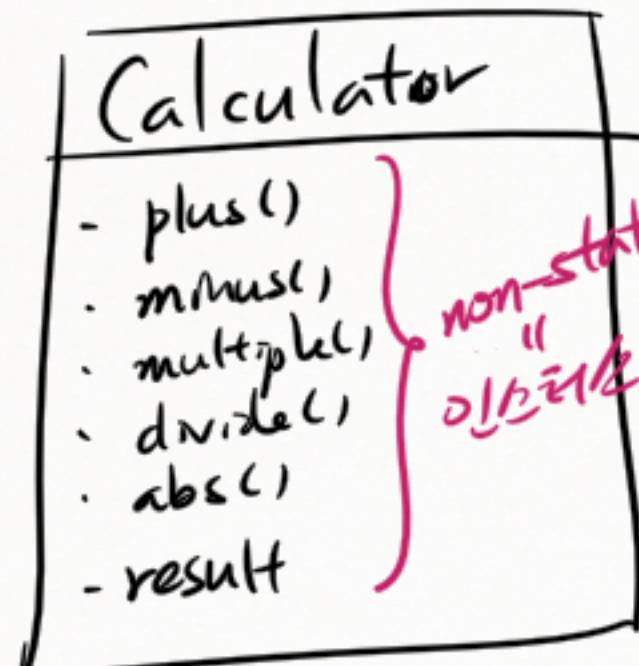
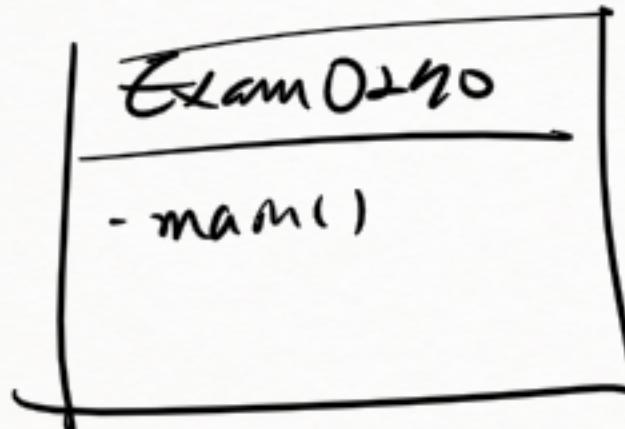
인스턴스 변수

인스턴스 변수
c1.plus(2)

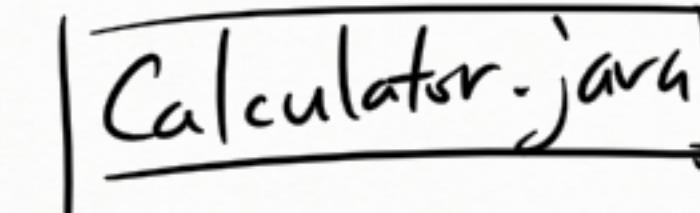
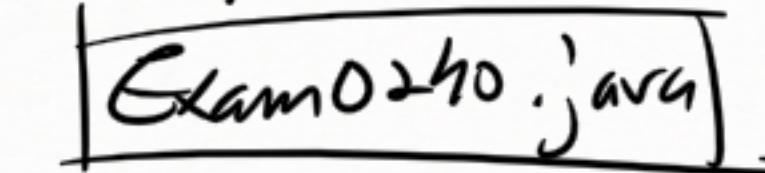


* \Rightarrow ml² ပုံမှန် ရှိခဲ့ပါ : မြတ်သွေးကို ဖြန့်မျက်

→ 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;

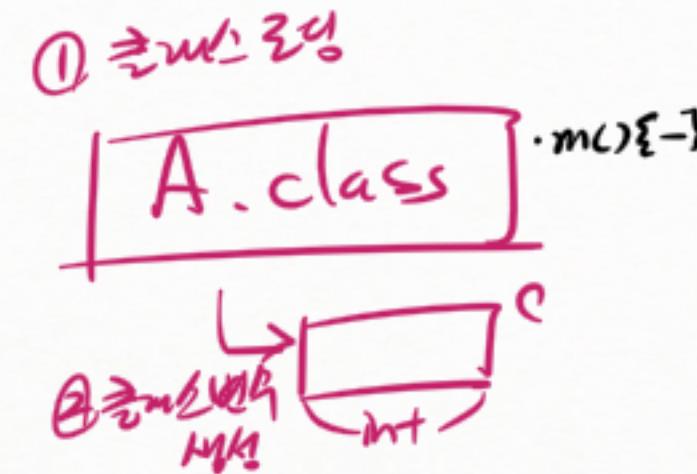
\Rightarrow အော်လုပ် ပုံမှန် ရှိခဲ့ပါ။
အော်လုပ် ပုံမှန် ရှိခဲ့ပါ။

* static 멤버 와 인스턴스 멤버

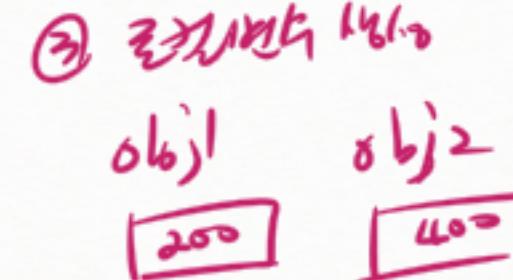
class A {
 인스턴스 멤버
 int a;
 int b;
 static int c;
 void m() {}
}

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

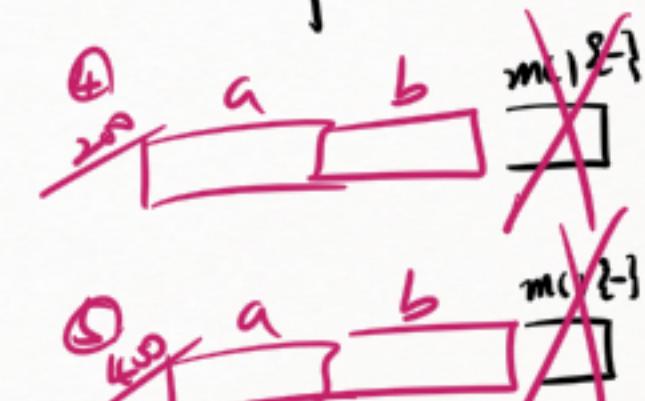
Method Area



JVM Stack



Heap



↑
 A 클래스의 인스턴스

인스턴스 멤버가 2nd.

* 클래스 멤버와 인스턴스 멤버

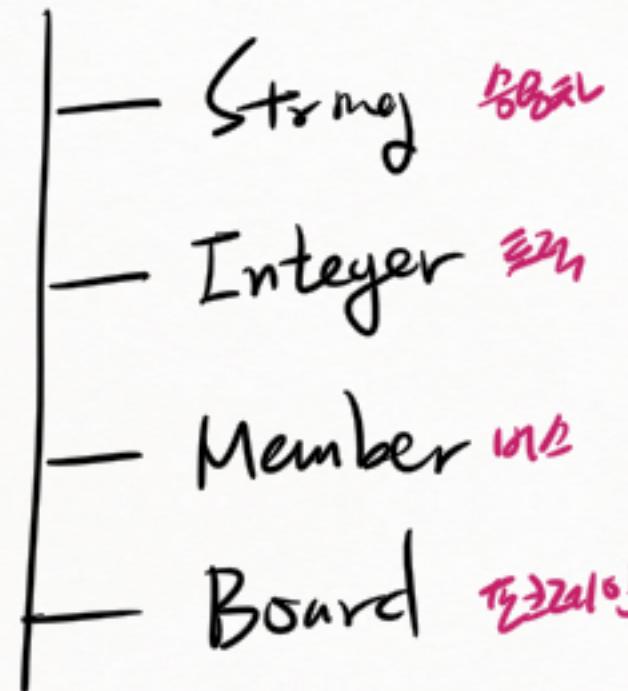
```
class Calculator {  
    int result;  
    void plus(int v){  
        result += v;  
    }  
    ...  
}
```

클래스 멤버
인스턴스 멤버를 다룰 때
연산자!
(인스턴스)

* Object 클래스
↳ 자바의 최상위 클래스

java.lang.

Object 사용

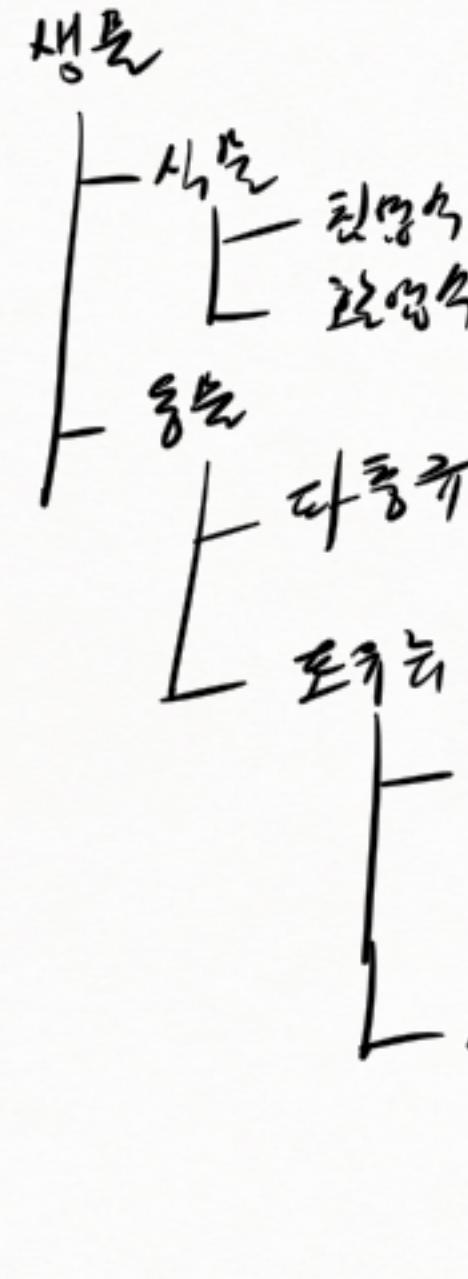


자바의 모든 것은 Object의
자식 클래스이다.

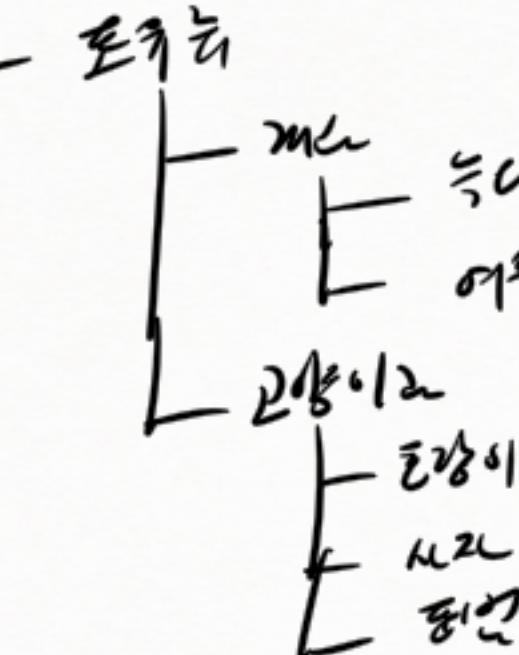
자식 (sub)

* 물류와 출판 분야

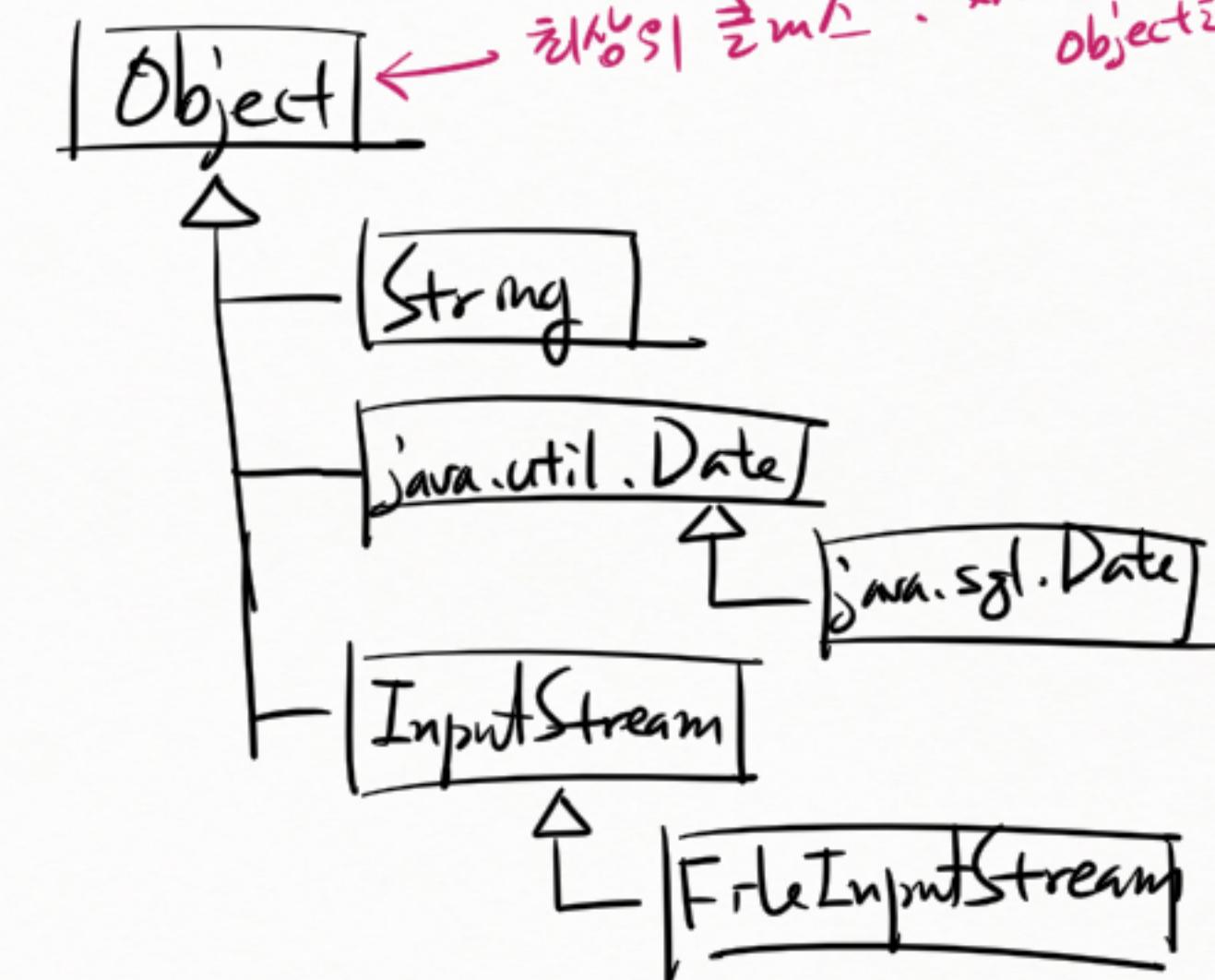
1877
MBS



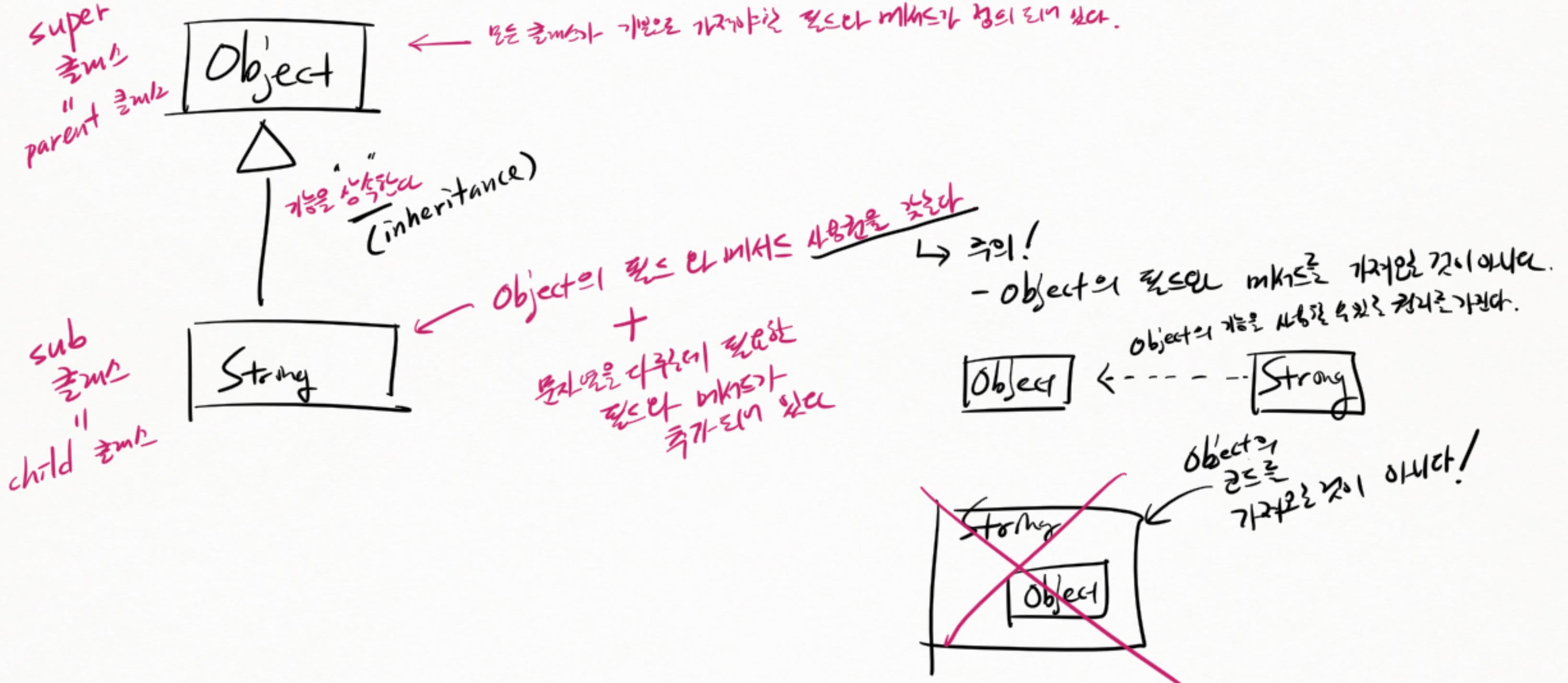
다중체



제작된 물건은 상수, 기능
기능을 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.

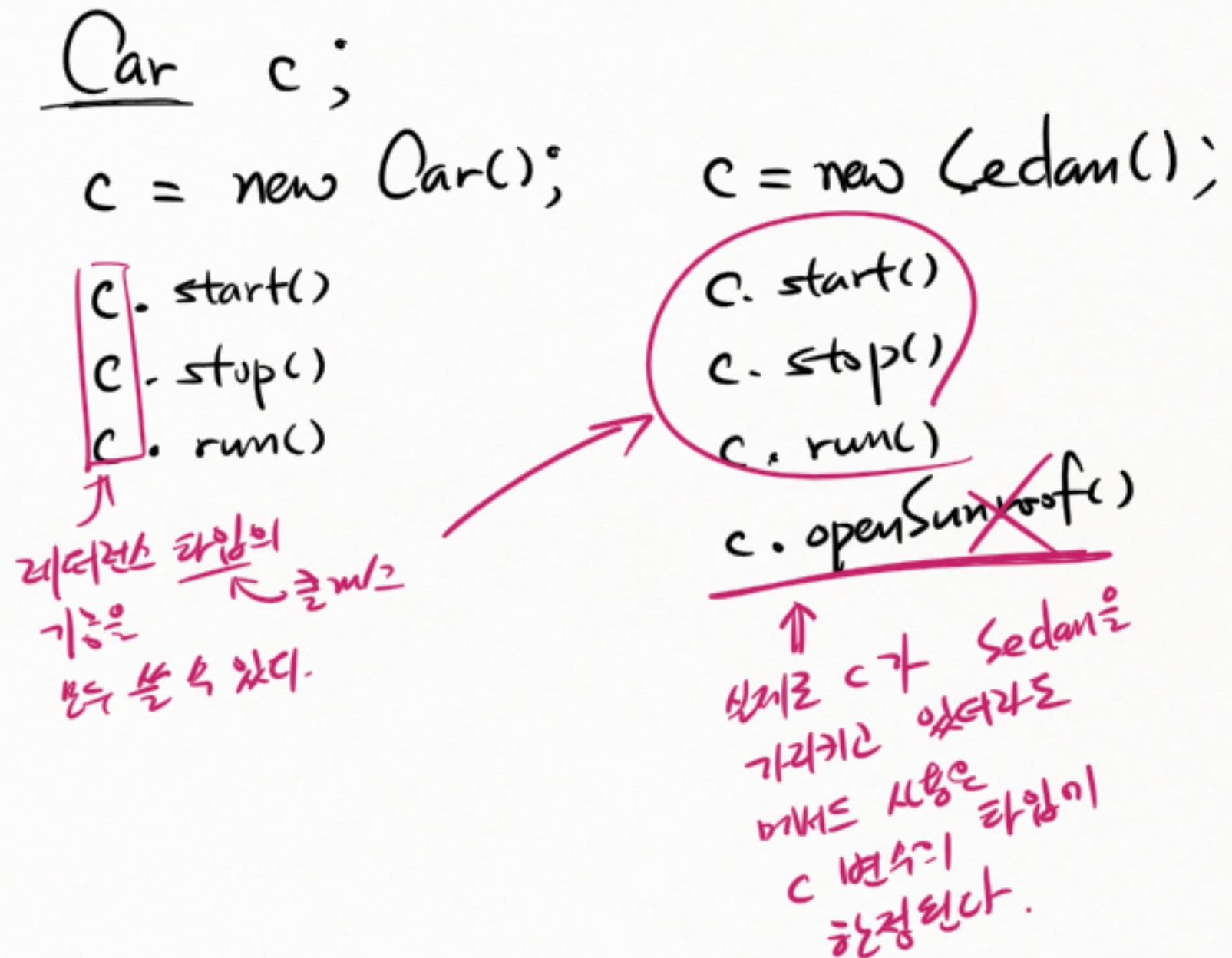
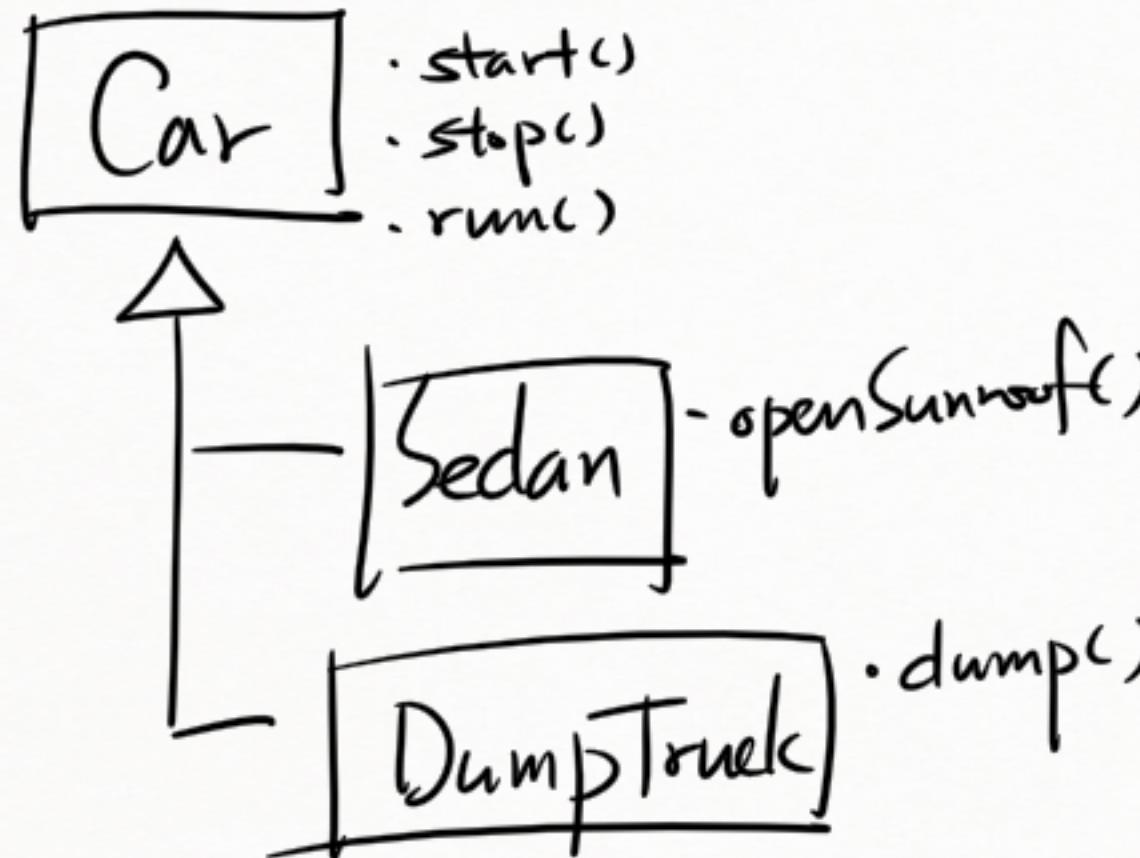
* 쿨러스 정의와 오버라이드 사용하기

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

(생략가능!)

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

* 상속과接口



Sedan s;

s = new Sedan();

s.start()

s.stop()

s.run()

s.openSunroof()

수퍼클래스의
기능은 자식
클래스에
다른.

~~Car = (2m/2
primitive-type)~~

~~s = new Car()~~

s.start()

s.stop()

s.run()

~~s.openSunroof();~~

s의 인터페이스
Sedan의 기능은
Car의 기능과
다른.
Hence Car

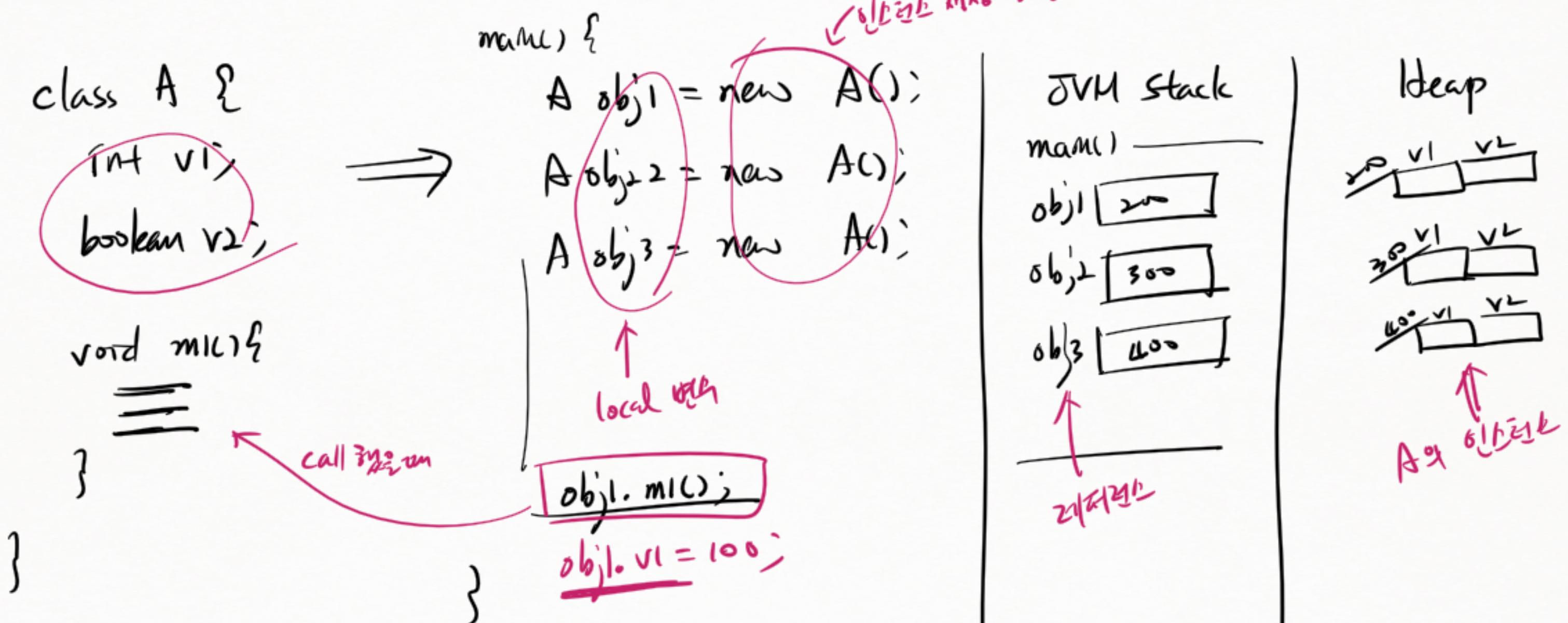
Sedan의 기능은
Car의 기능과
다른.
Hence Car

자연 적용된 API
Car 모델
= Car(2m/2)
= Car(2m/2)
= Car(2m/2)
= Car(2m/2)
= Car(2m/2)
= Car(2m/2)
= Car(2m/2)

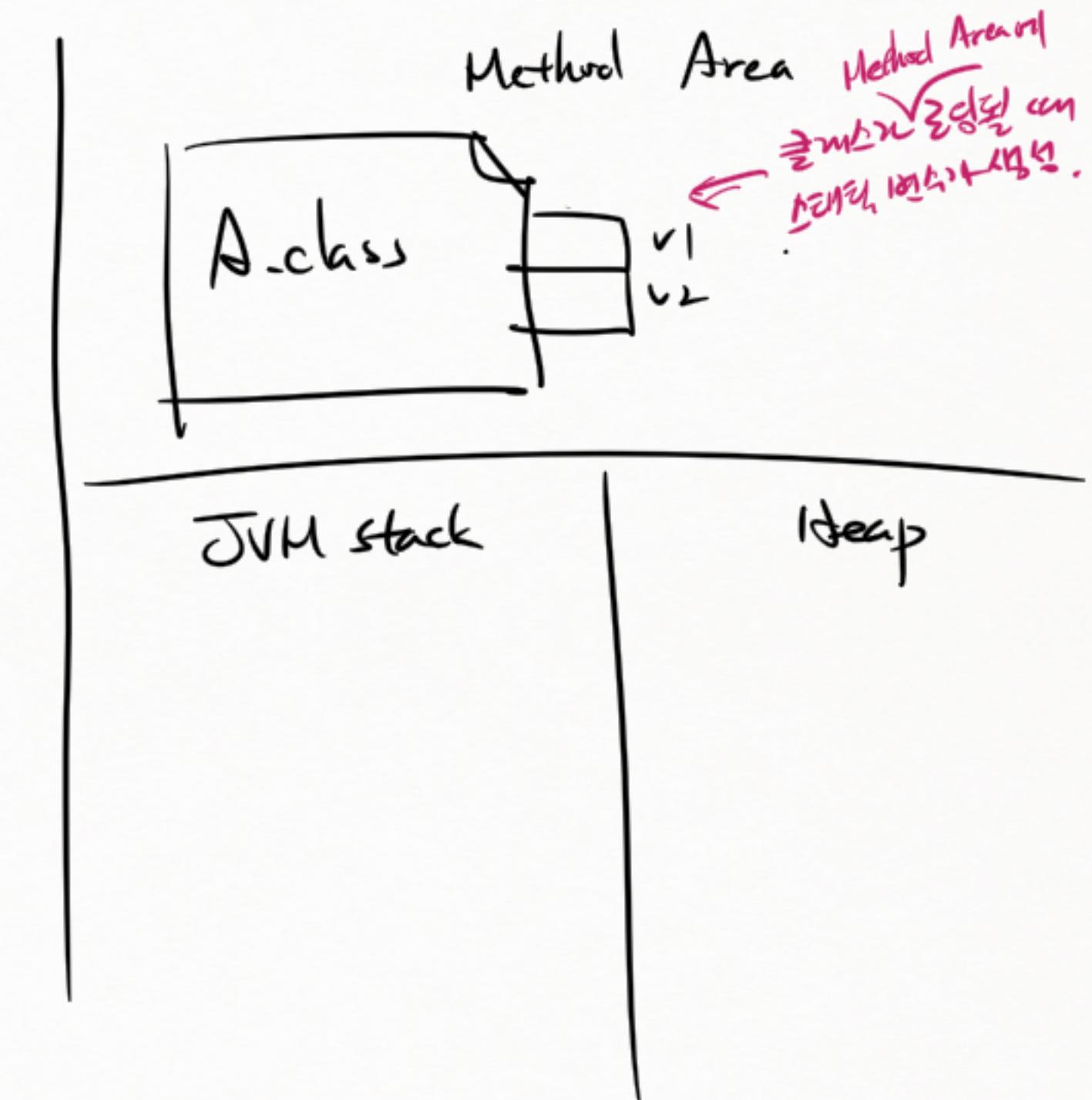
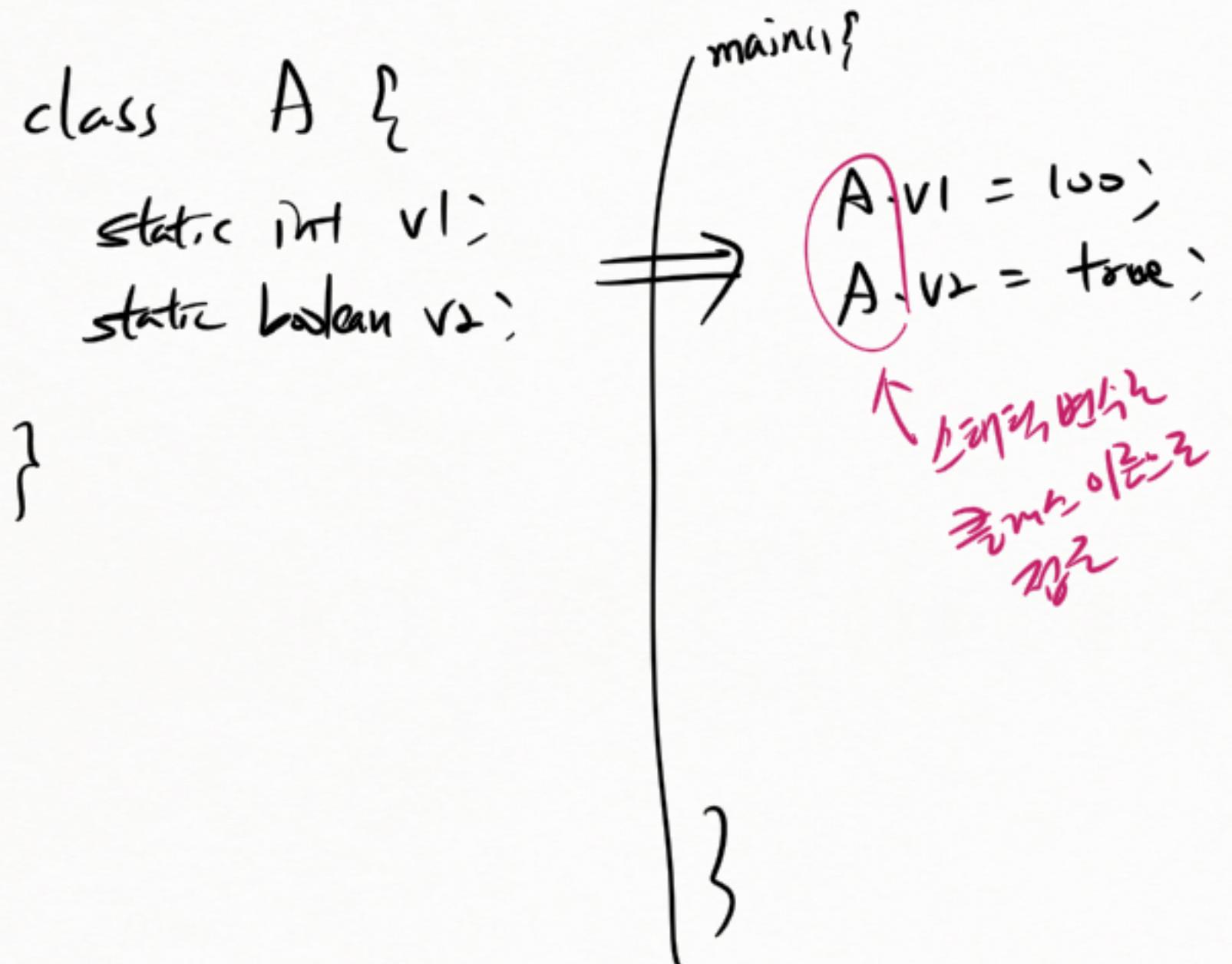
* static 필드와 non-static 필드

```
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



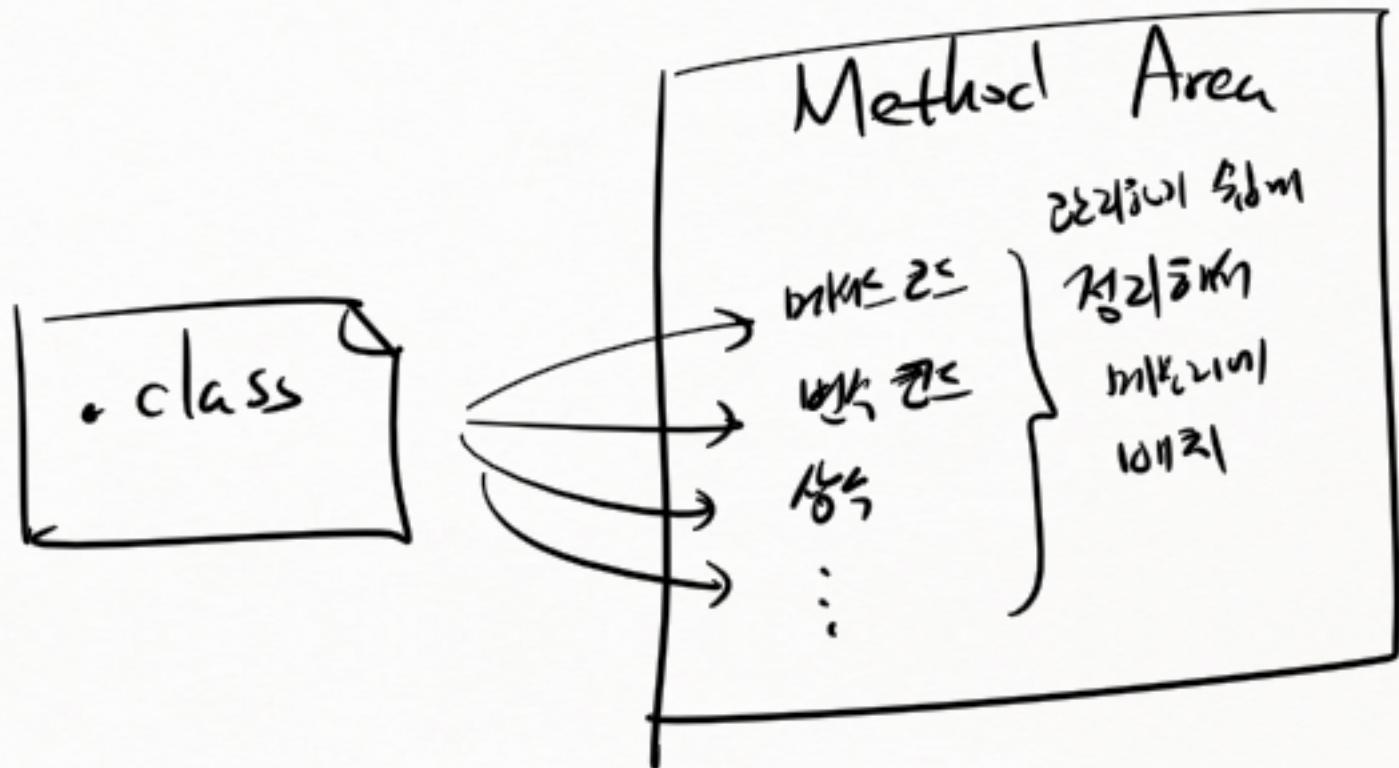
* 주소(static) 필드(값)



* 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



* 예제 2 문제, 소스코드 및 실행 결과 분석

