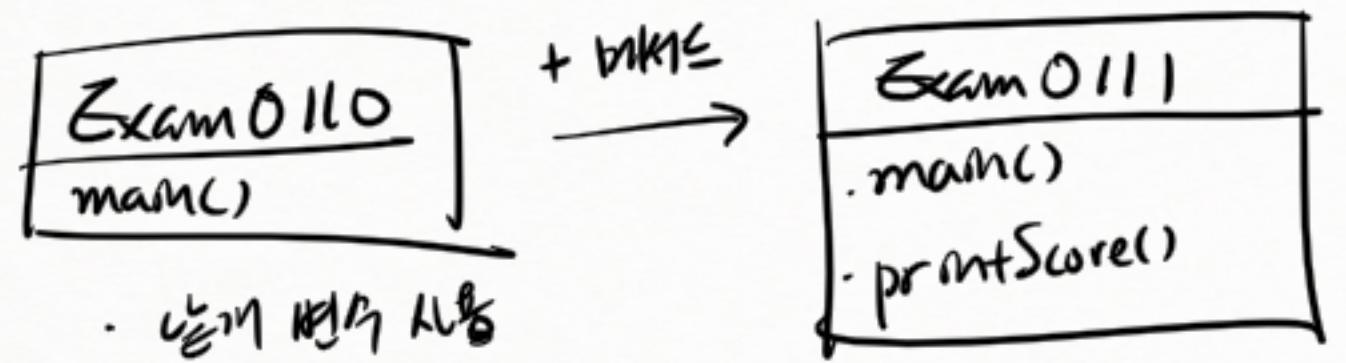


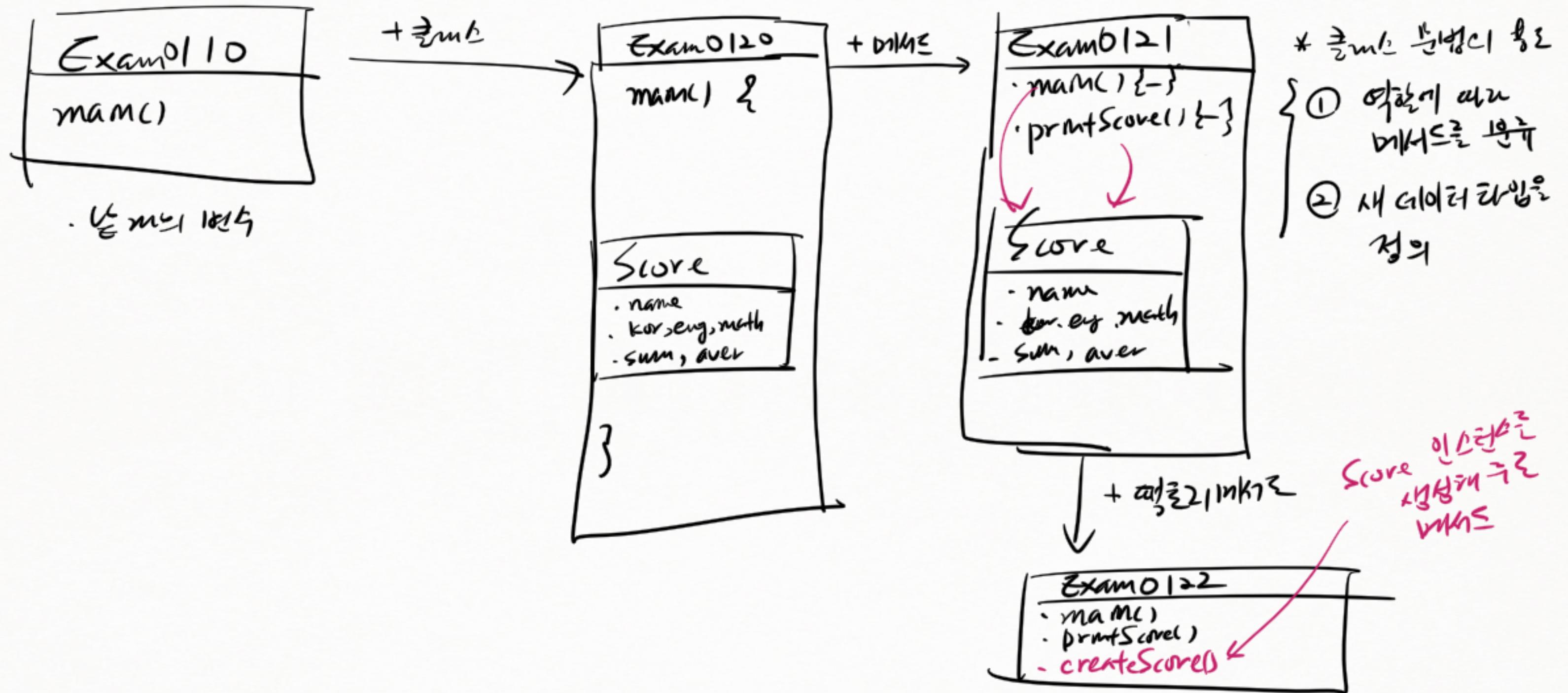
* 재미스 문법 활용 예



• 놓기 허용 사용

- 인라인 문법 활용
↳ 인라인화 사용

- ↓
✓ 정복 코드 세기
↓
코드 처리 사용 ↑
✓ 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 = "—"'

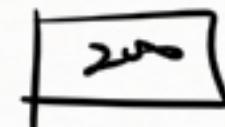
s1. kor = 100 -

:

* 예외처리 10주차

Score[] arr = new Score[3];

arr



null?
- null이면 오류!
- 접근할 수 있는 배열의 범위를 초과하는 경우.

arr[0] = new Score();

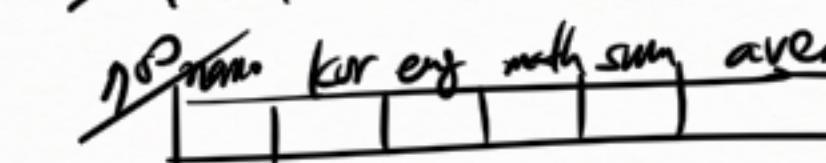
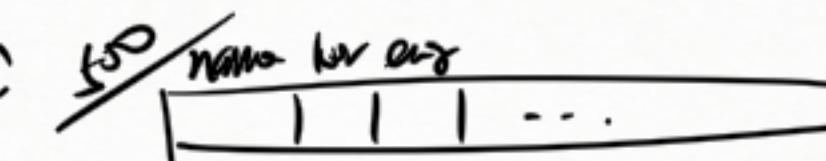
Index 2 ← 자동으로 null 값으로 초기화 된다
* 접근 범위는 초기화되어 있다.

arr[1] = new Score();

arr[2] = new Score();

arr[3] = new Score();

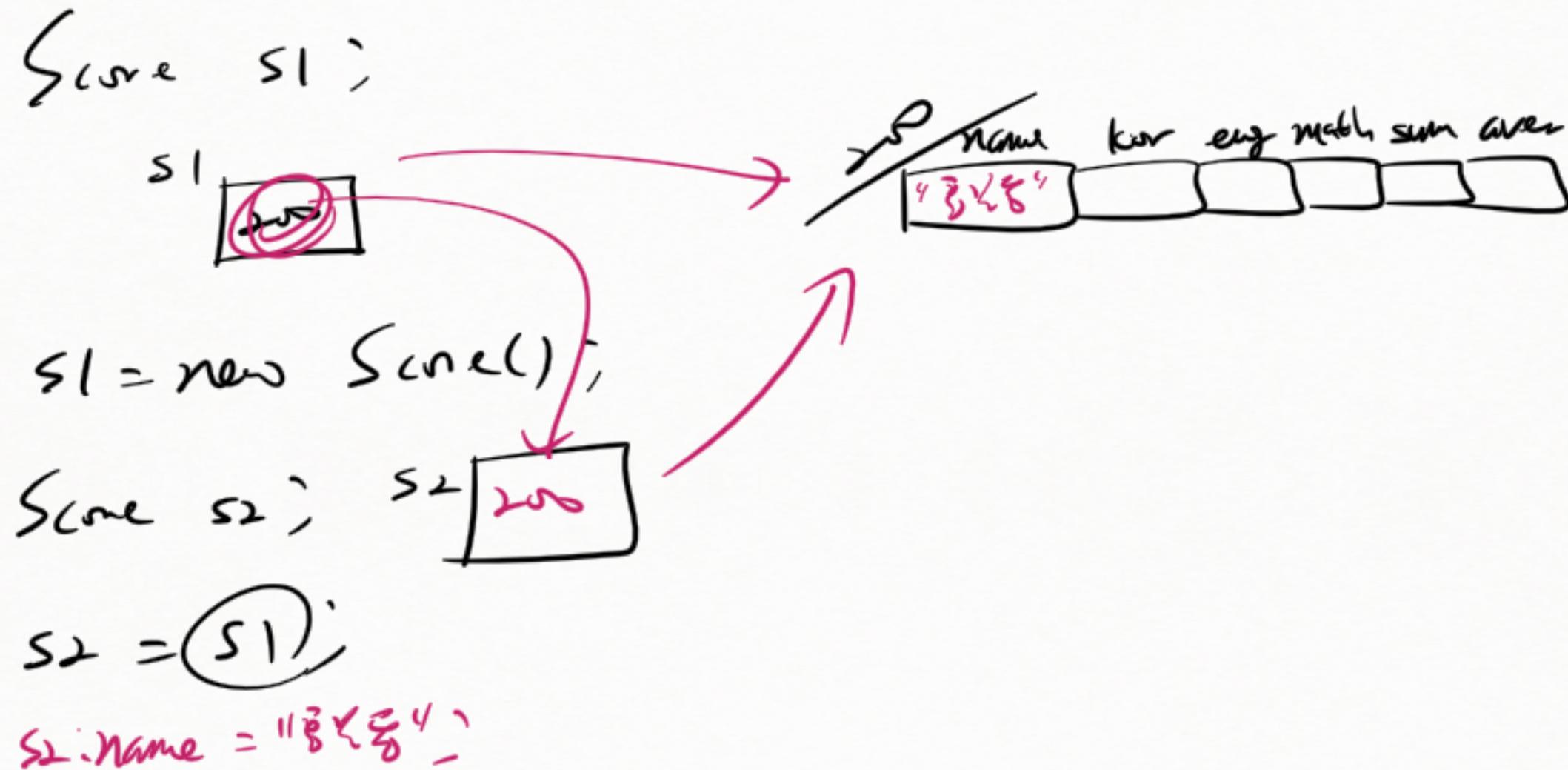
* ArrayIndexOutOfBoundsException



new Score();

Score ⇒ null 선언한 위치
Heap에 차례로 쌓아온다.
기억해두면 좋다.

* 리터럴과 인스턴스



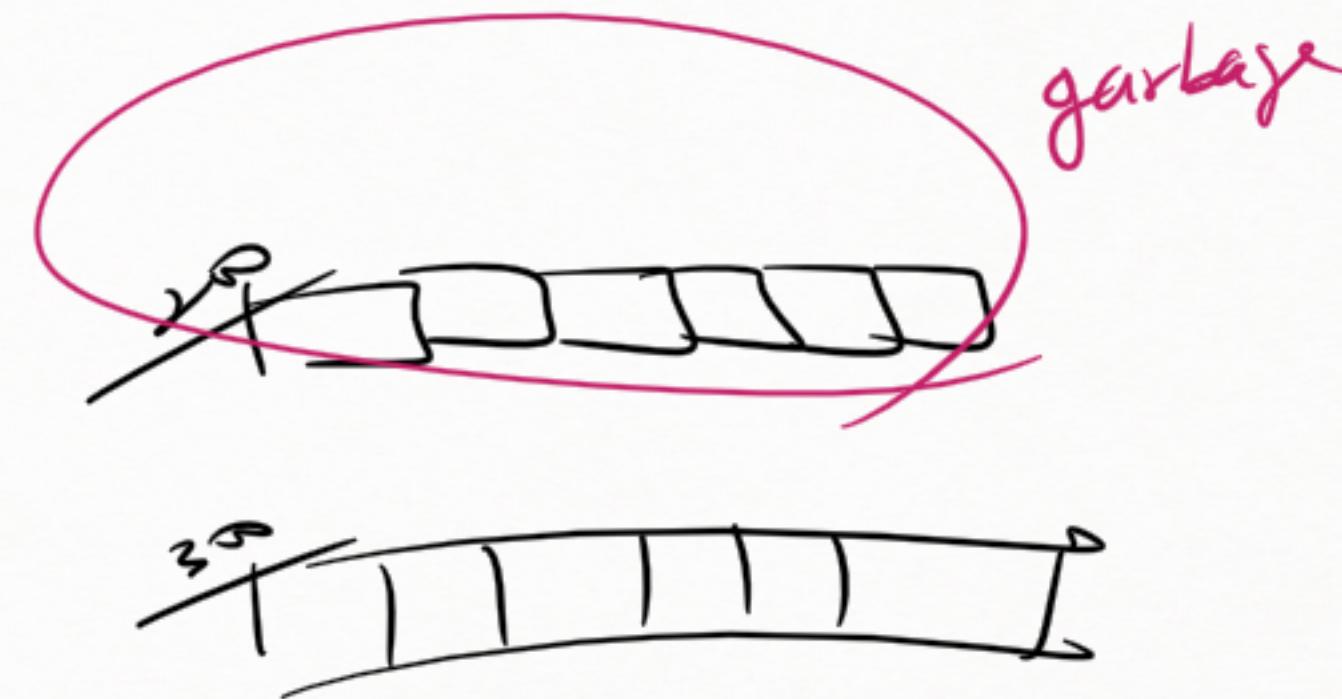
* 7110121 (garbage)

Score s1;



s1 = new Score();

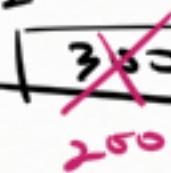
s1 = new Score();

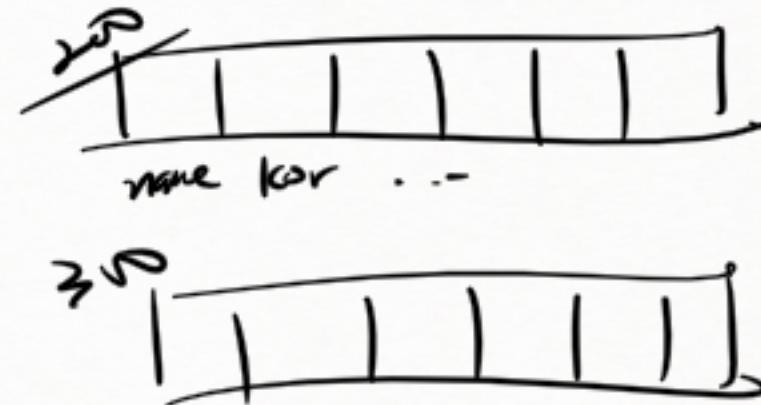


* 리터럴은 카운트와 관계

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

s1

s2




JVM이 처리

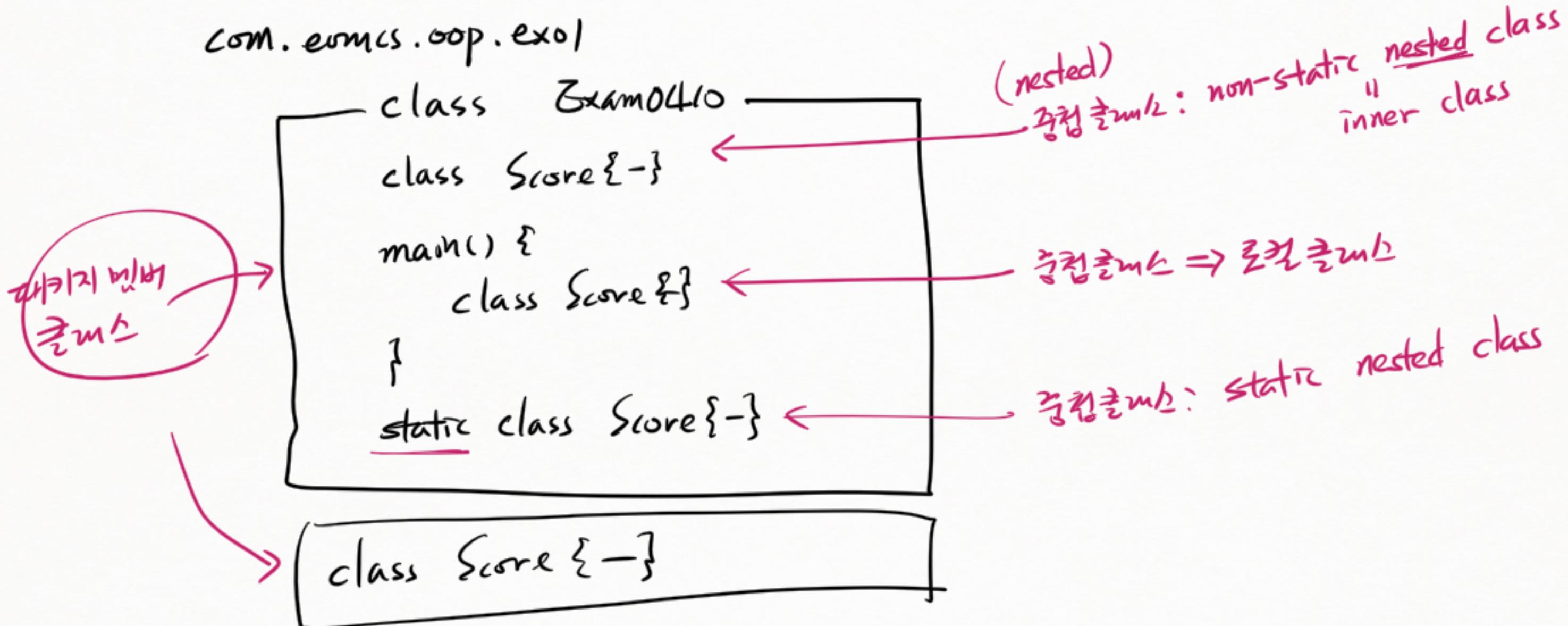
리터럴은 카운트 관계

리터널은 카운트가
0인 경우
"garbage"라
한다.

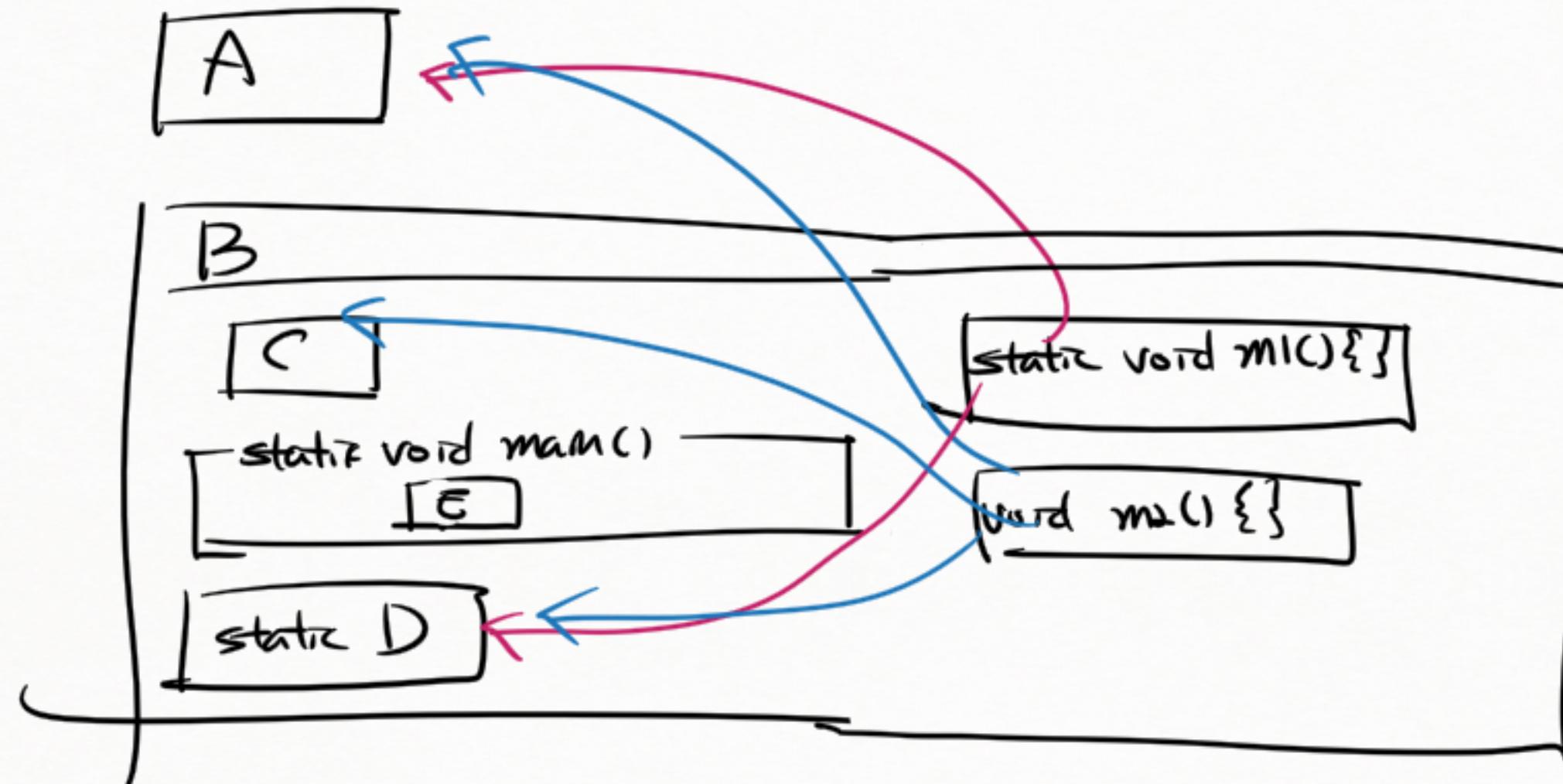
인스턴스	참조 횟수
200	X 2
300	X 0

* 클래스 구조

com.eunics.oop.ex01

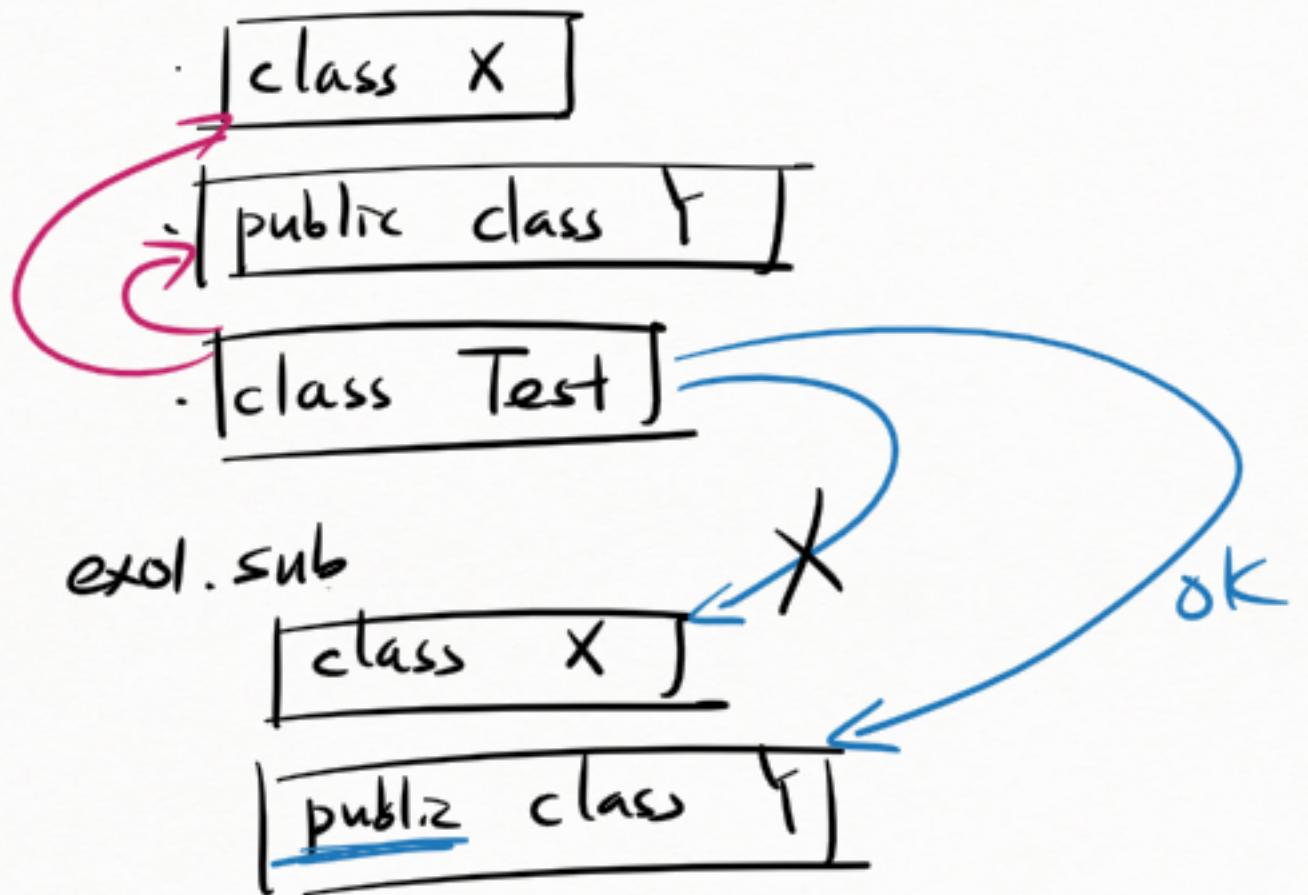


* static member non-static member



* 공개 멤버 필드

ex01



ex01.sub

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



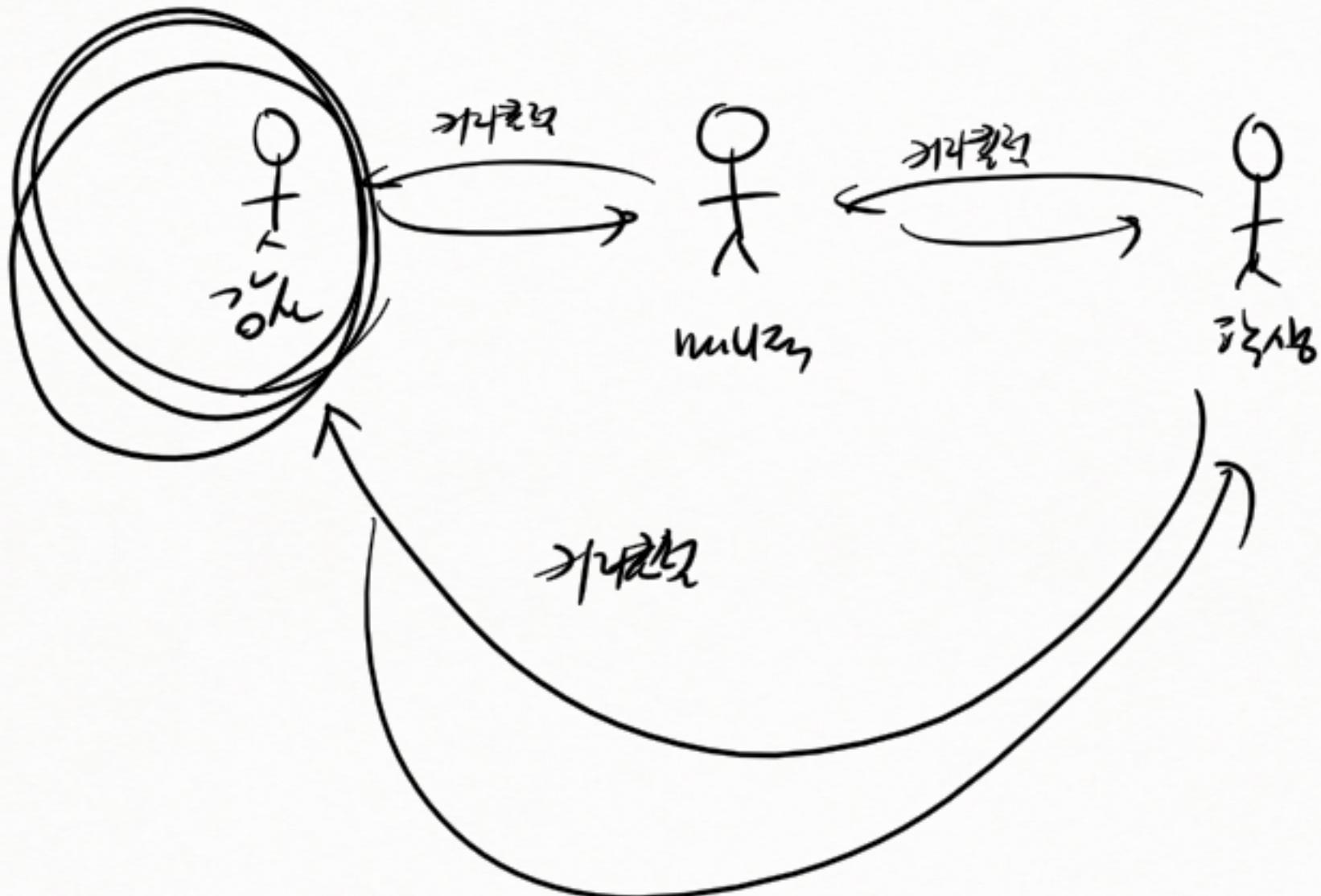
s1.name = "김길동"

* optimizing(최적화) vs refactoring(재구조화)

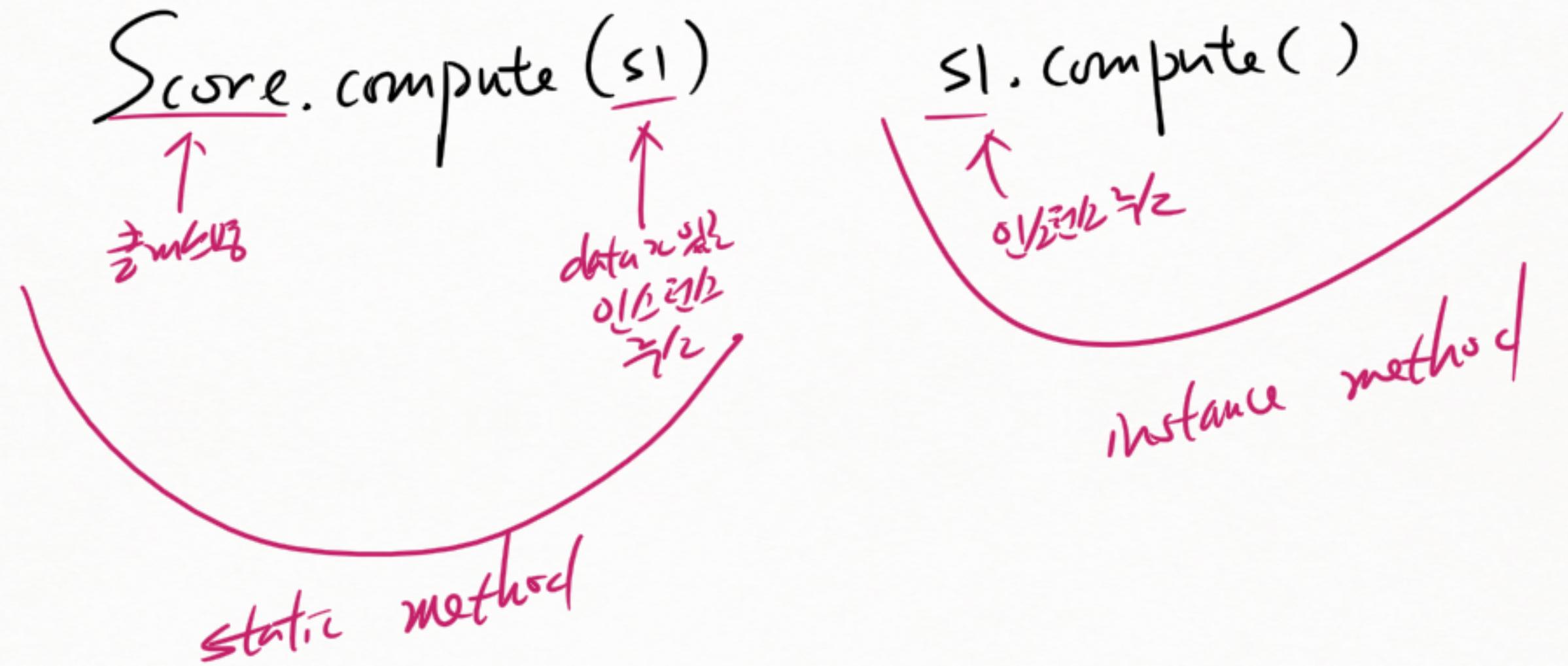
- ↓
- 속도↑
- 유지보수 품질↑
- 속도↓

- ① s1 리퍼런스에 저장된 주소로 총칭해서 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

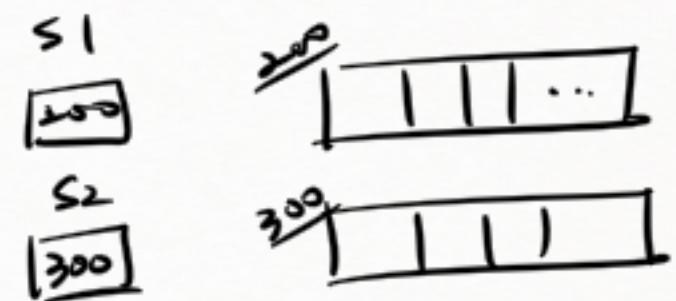
j ++ ;
 operand
 (인자)
 operator (연산자)

j ++;

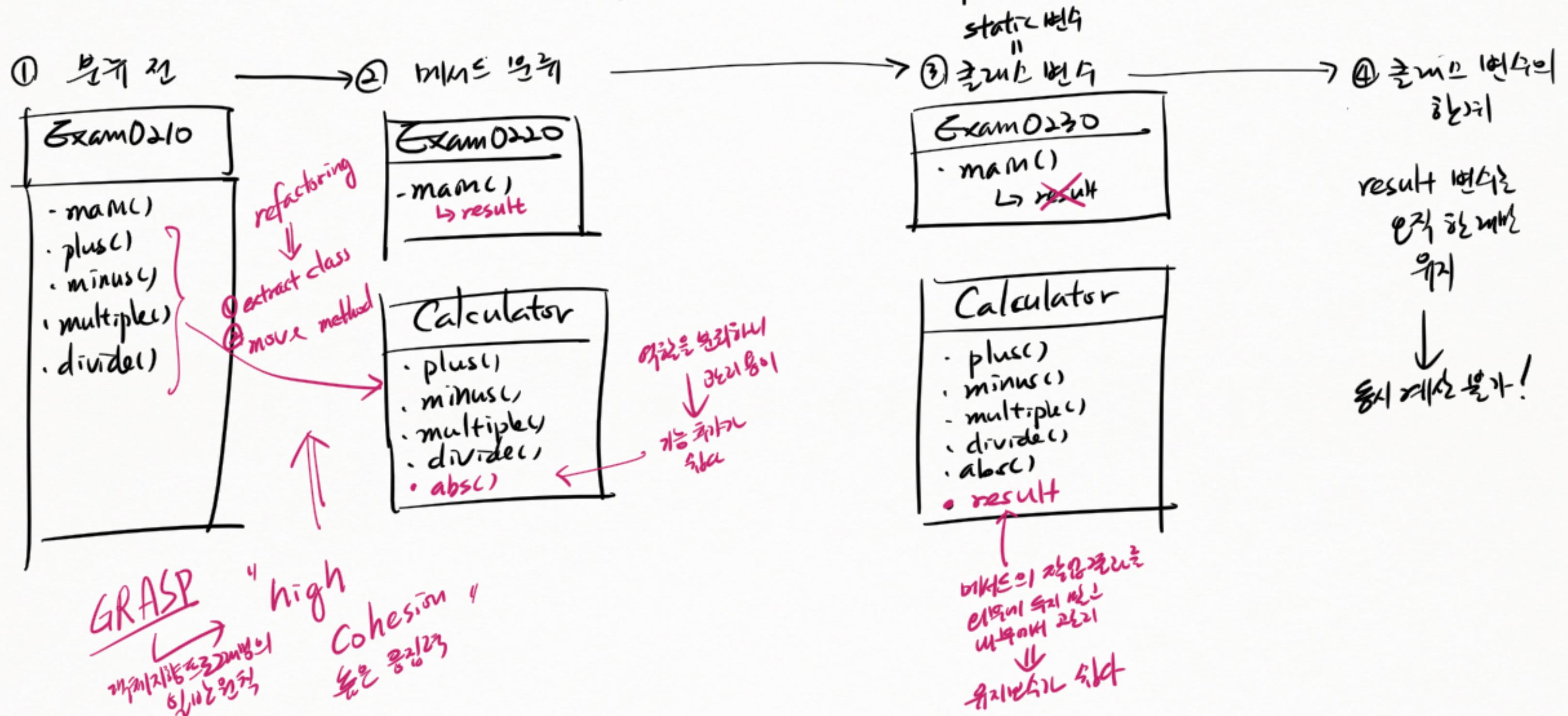
Score s1, s2 ;
 s1 = new Score();
 s2 = new Score()

인자
 ↓
 s1. compute()
 ↑ operand
 ↑ operator

s2. compute()

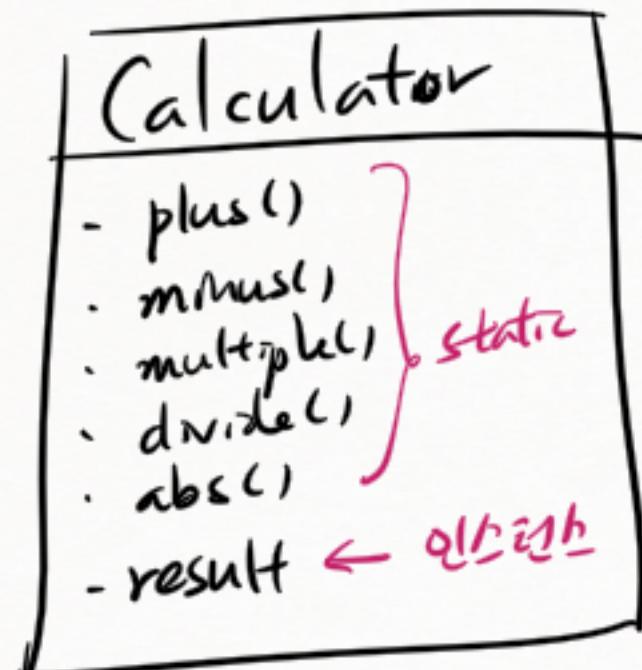
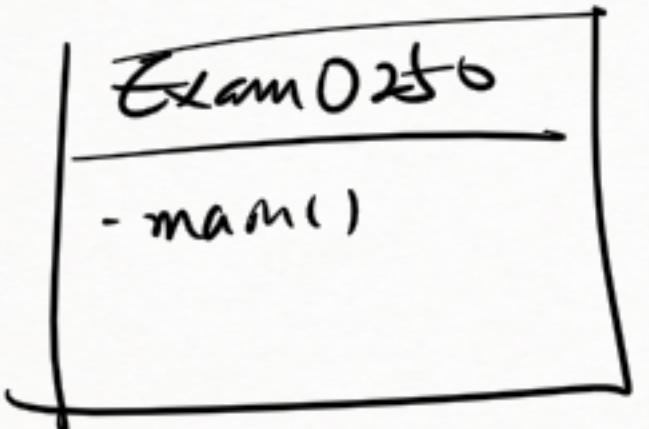


* 3단계 분기점: MMR을 끝난 분기점이



* 인스턴스 멤버 함수: 인스턴스마다 다른 결과를 반환

→ ⑤ 인스턴스 멤버 변수



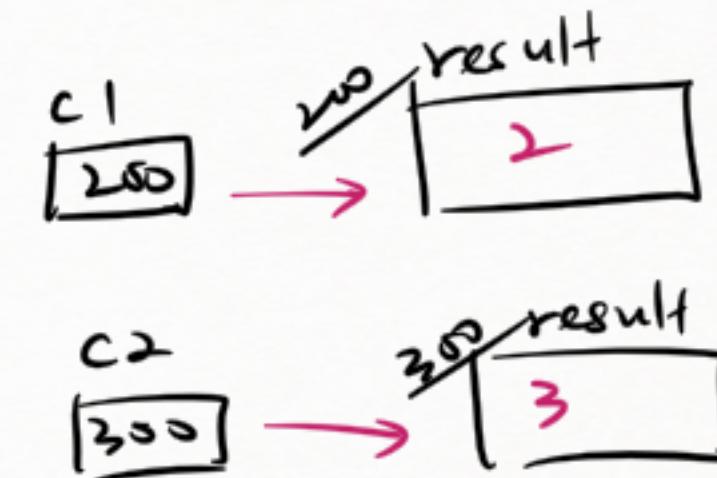
```
Calculator c1 = new Calculator();
Calculator c2 = new Calculator();
```

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

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

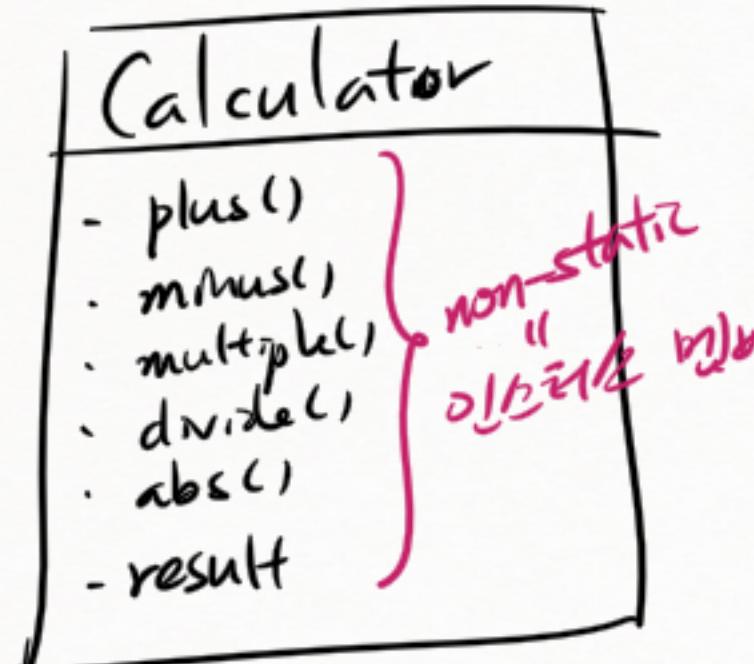
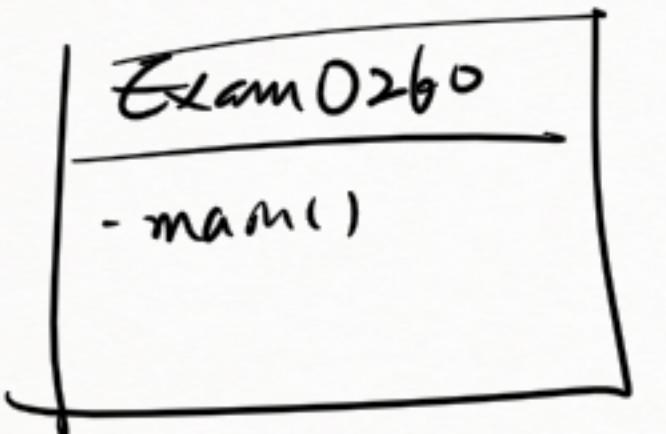
:

↑
각 인스턴스는 자신의
result 멤버 변수에
0으로 초기화된다.



* 인스턴스 멤버 변수: 인스턴스마다 다른 값을 갖다

→ ⑥ 인스턴스 변수

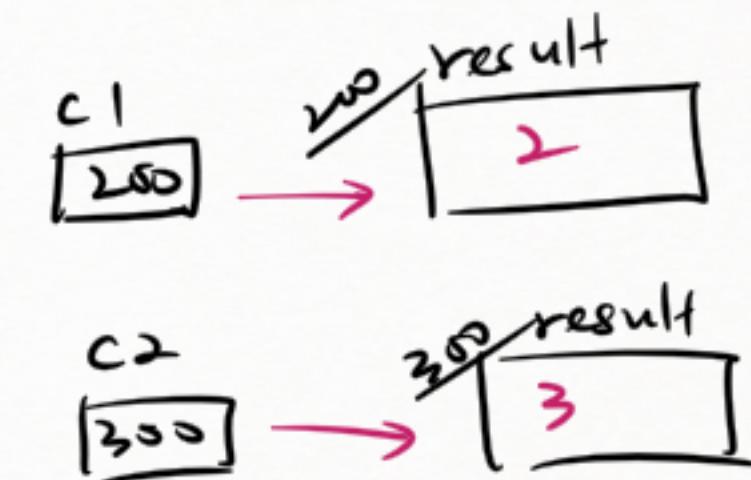


Calculator c1 = new Calculator();
Calculator c2 = new Calculator();

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

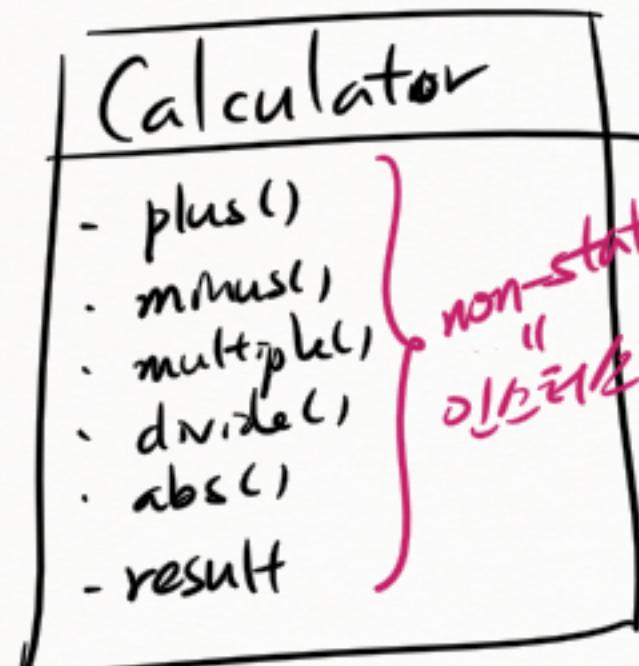
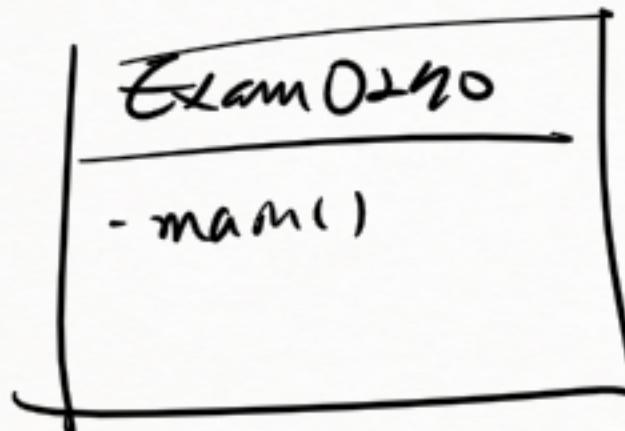
인스턴스 변수

인스턴스 변수
c1.plus(2)

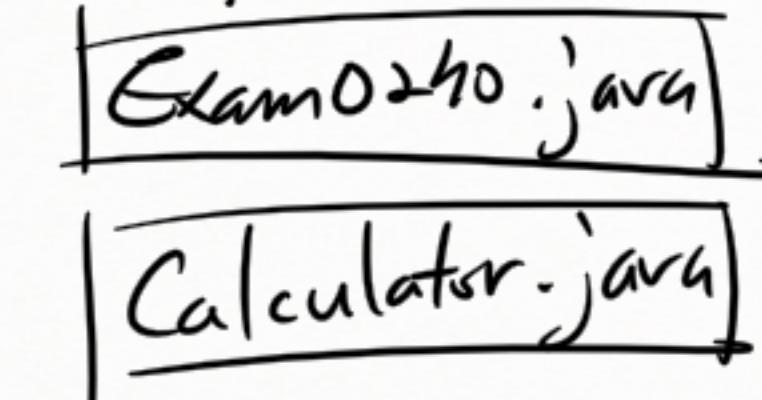


* \Rightarrow ml² ပုံမှန် ရှိခဲ့ပါ : မြတ်သွေးကို ဖြန့်မျက်

→ 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;

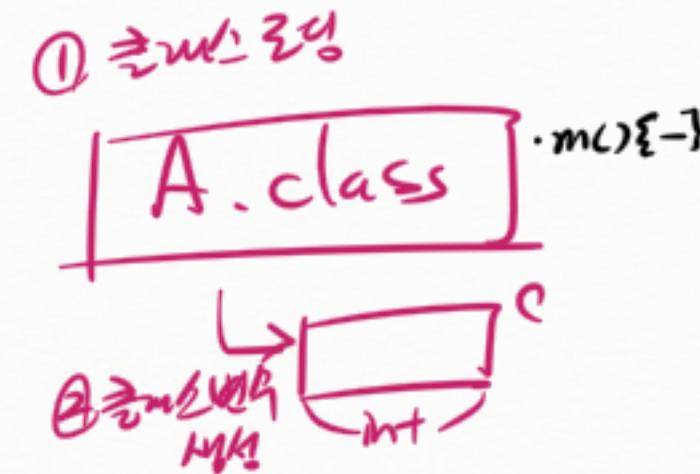
↑
 \Rightarrow ပုံမှန် ပုံမှန် ပုံမှန် ပုံမှန်

* static 멤버 와 인스턴스 멤버

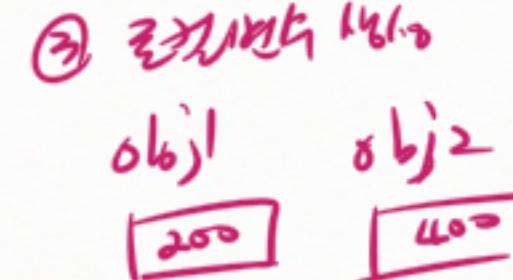
```
class A {
    int a;
    int b;
    static int c;
    void m() { }
}
```

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

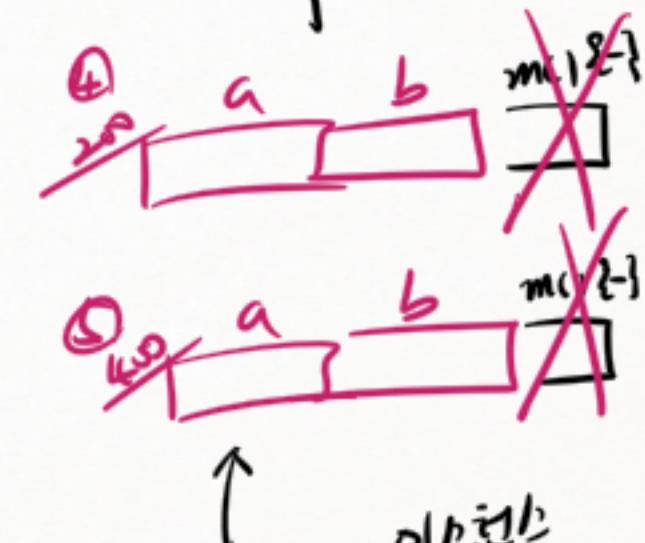
Method Area



JVM Stack



Heap



\uparrow
 A 클래스의 인스턴스화

\downarrow
인스턴스화된 필드가 대체됨.

* 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)

