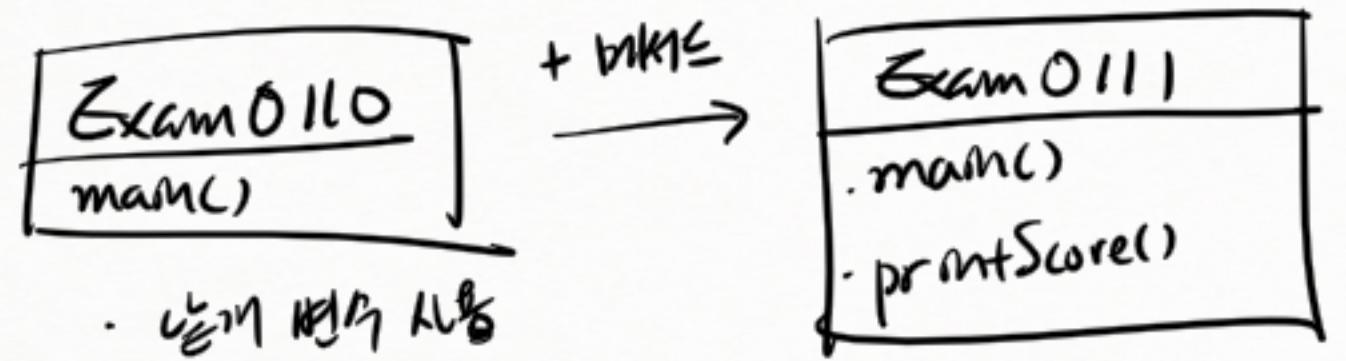


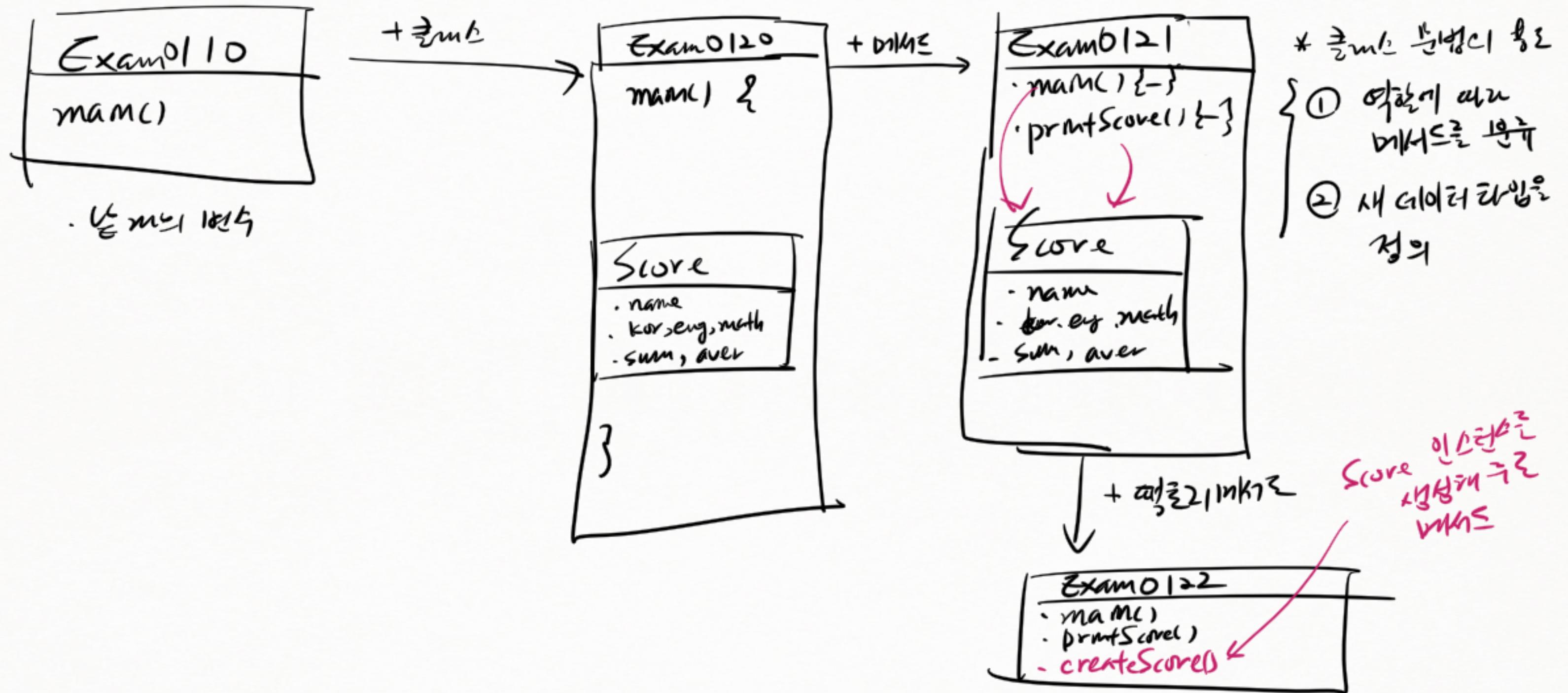
* 재미스 문법 활용 예



- 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 = "—"'

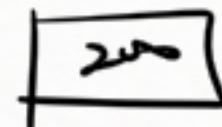
s1. kor = 100 -

:

* 예외처리 10주차

Score[] arr = new Score[3];

arr



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

arr[0] = new Score();

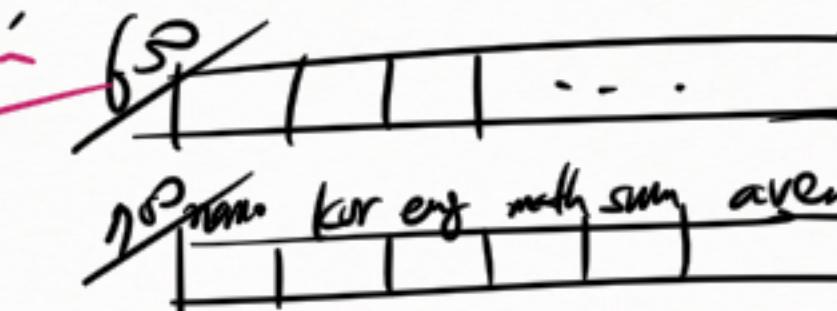
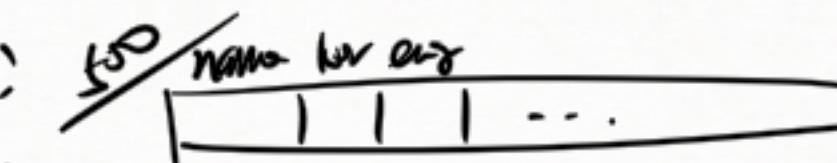
* 예외 처리 ← 자동으로 null로 초기화된다
* 접근할 수 있는 멤버가 초기화되면 좋다.

arr[1] = new Score();

arr[2] = new Score();

arr[3] = new Score();

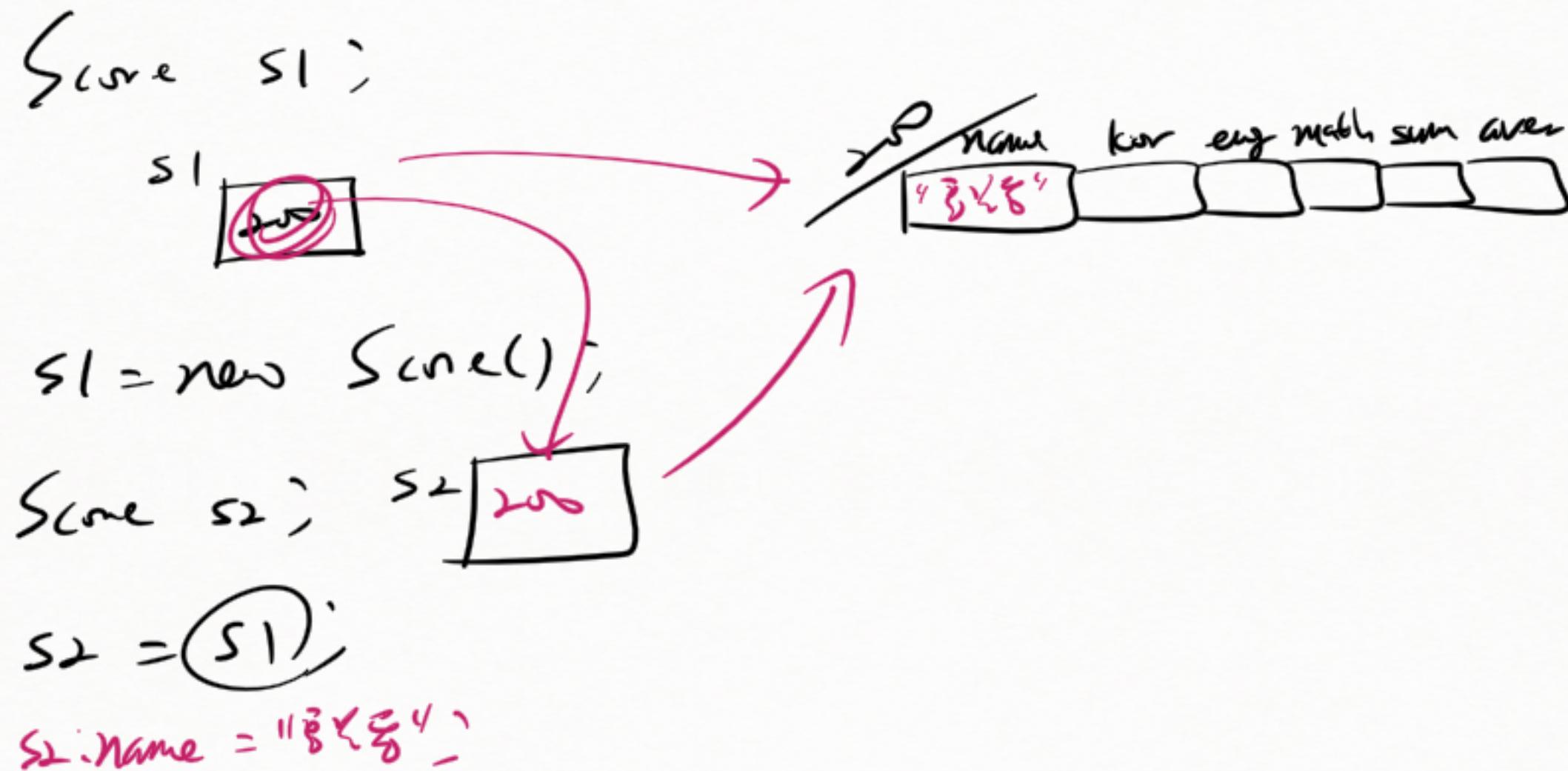
* ArrayIndexOutOfBoundsException



new Score();

Score → null 선언 했을 때
Heap에 차례로 쌓아온다.
기억해두면 좋다.

* 리터럴과 인스턴스



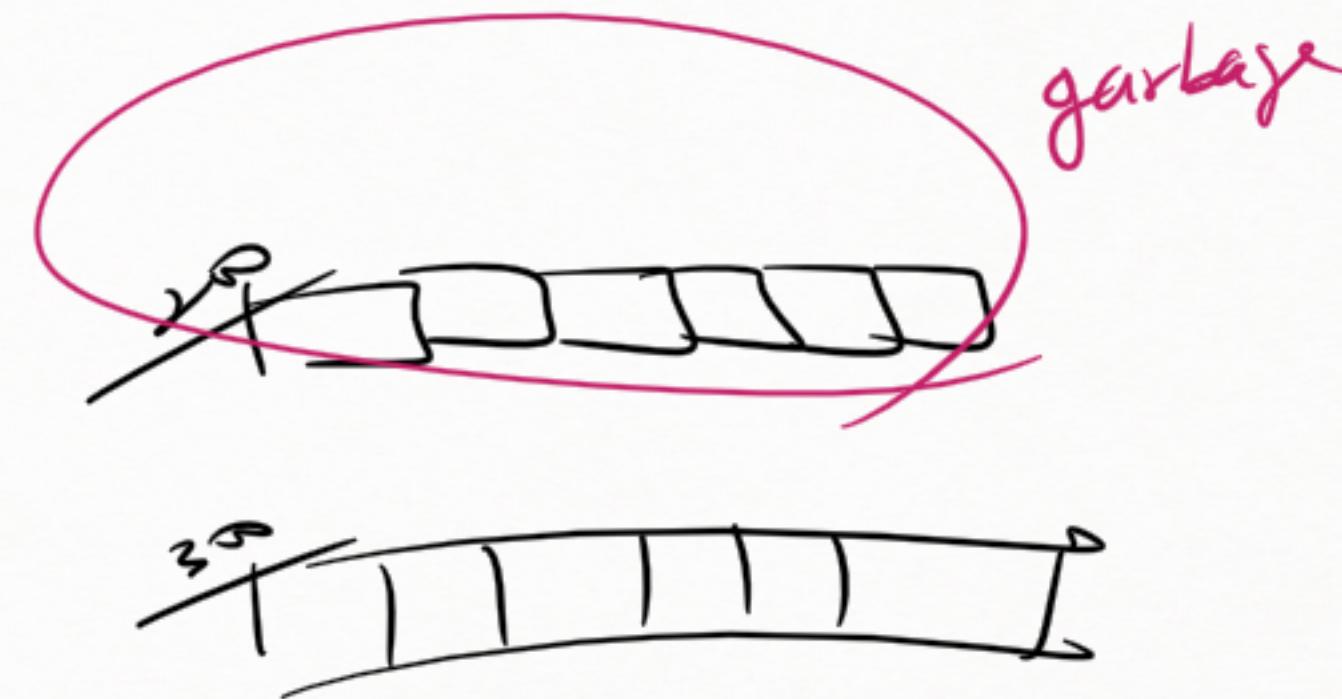
* 7110121 (garbage)

Score s1;



s1 = new Score();

s1 = new Score();

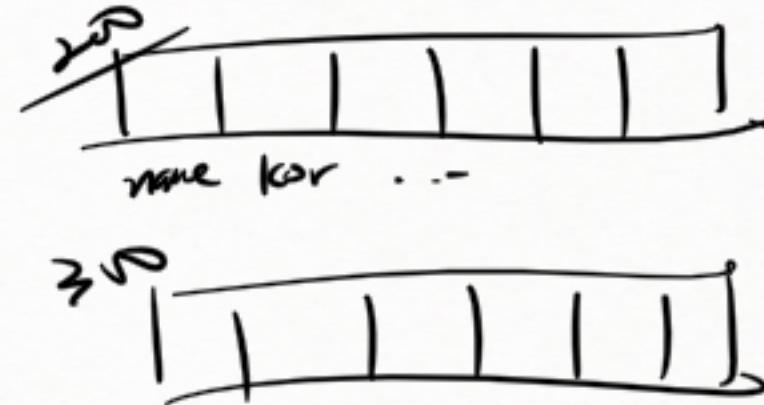


* 리터럴은 카운트와 관계

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

s1

s2
~~300~~
200



JVM이 관리

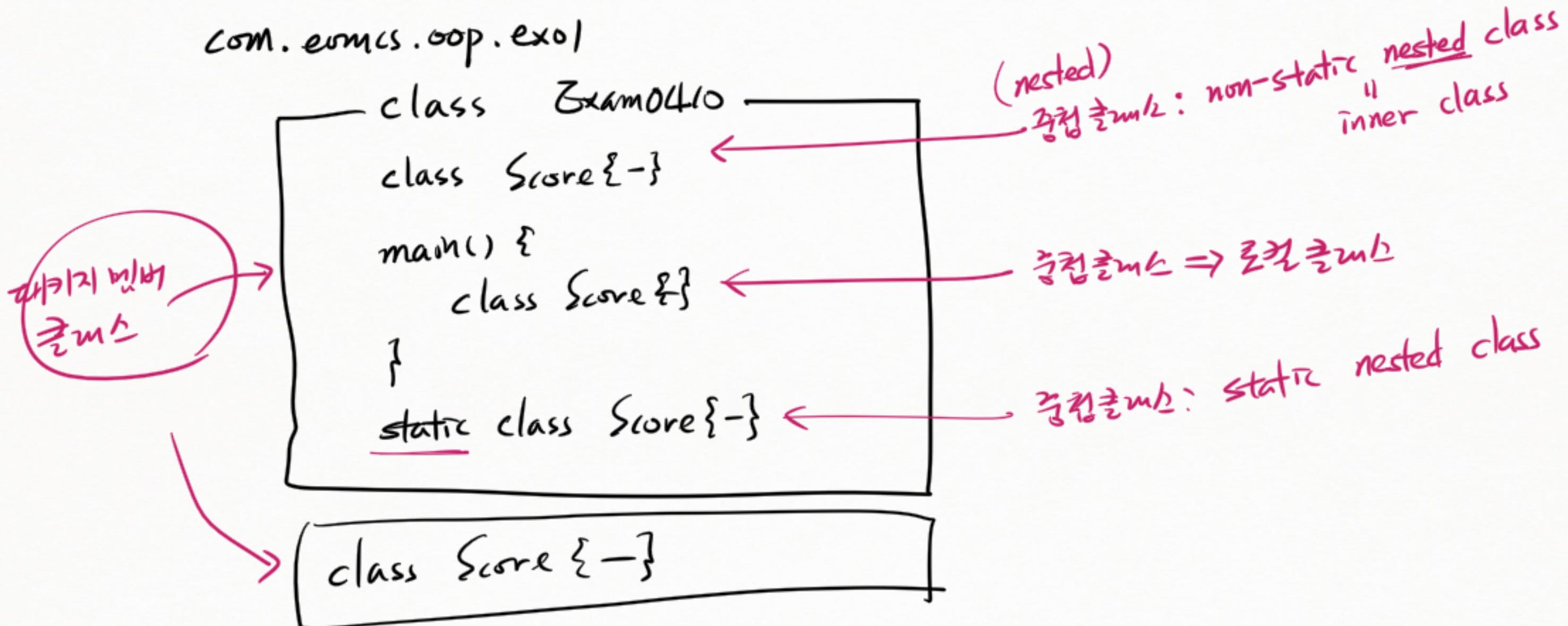
리터럴은 카운트 관계

리터널은 카운트가
0인 경우
"garbage" 가
된다.

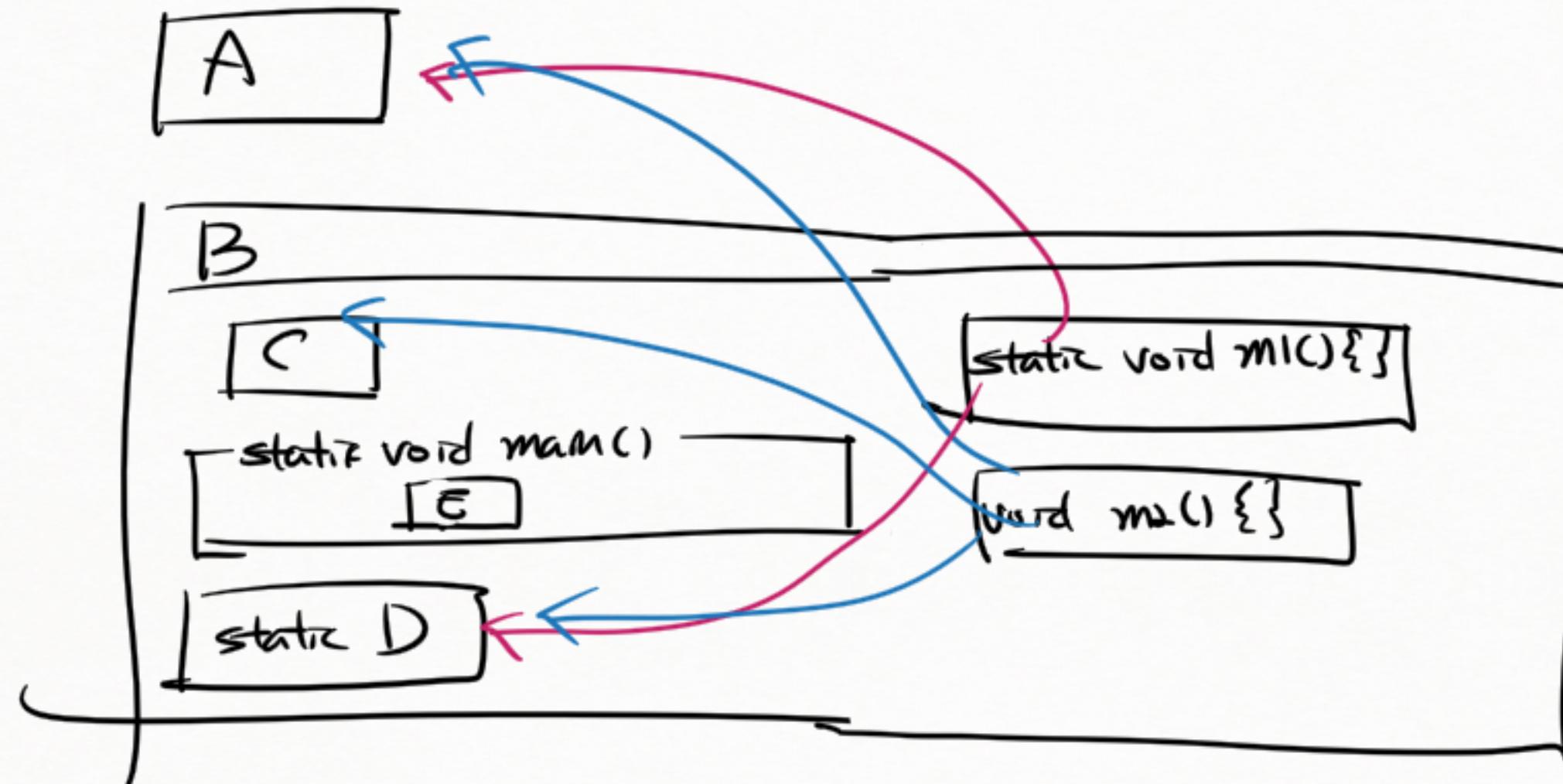
인스턴스	참조 횟수
200	X 2
300	X 0

* 클래스 구조

com.eunics.oop.ex01

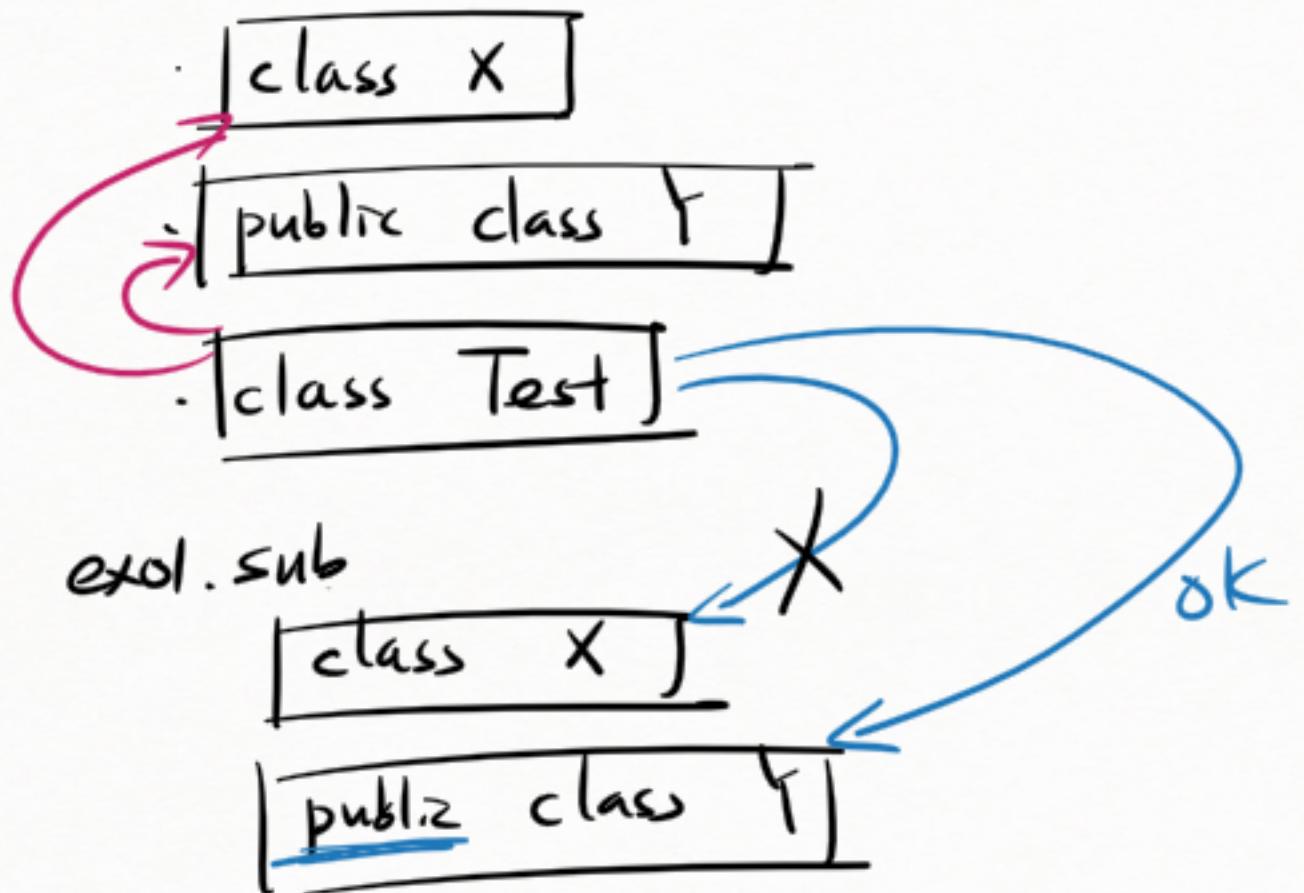


* static member non-static member



* 공개 멤버 필드

ex01



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



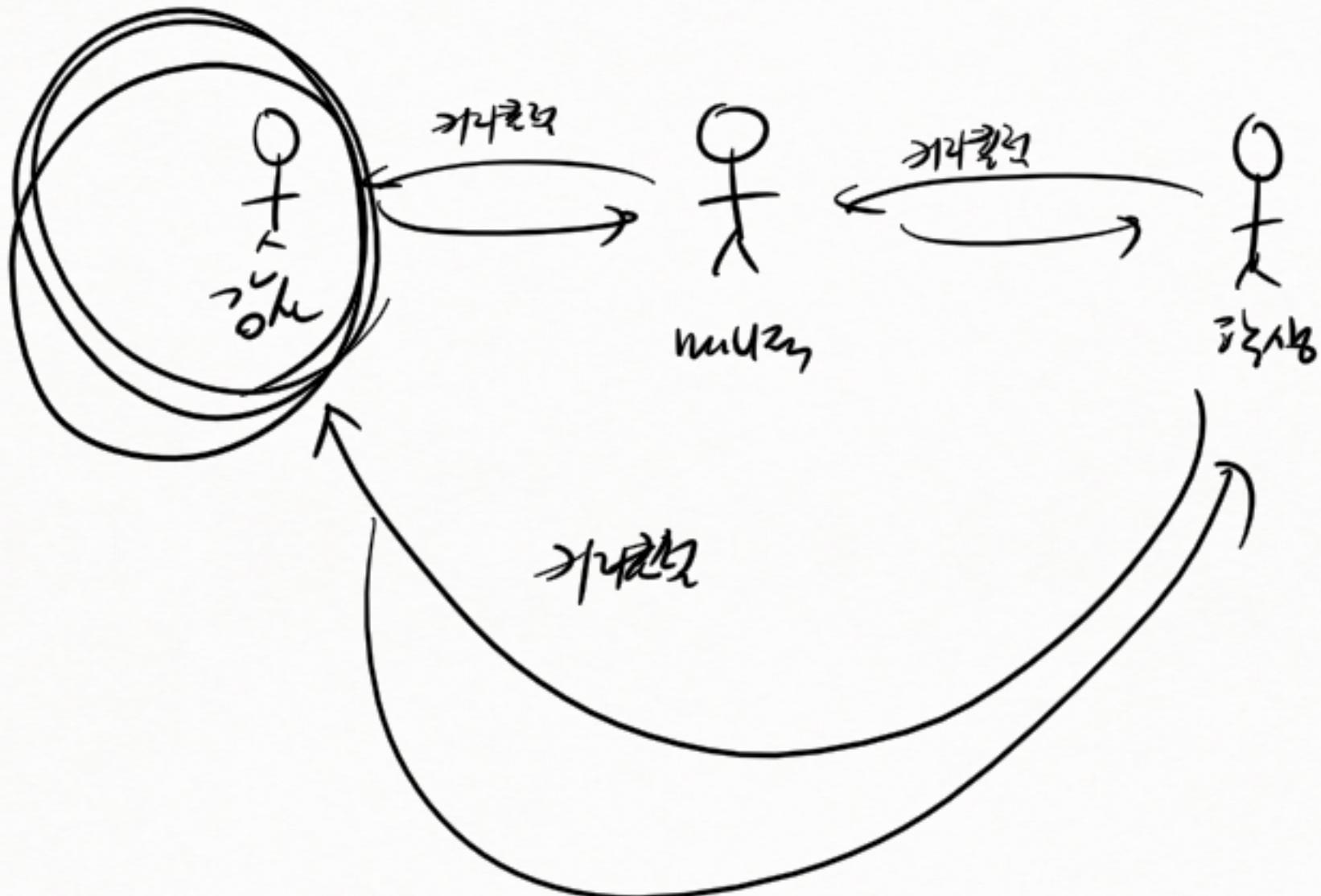
s1.name = "김철수"

* optimizing(최적화) vs refactoring(재구조화)

- 속도↑
- 유지보수 편리↑
- 속도↓

- ① s1 리퍼런스에 저장된 주소로 채워가서 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

