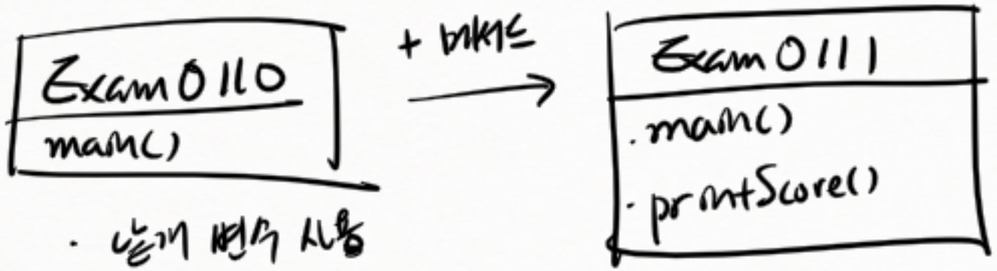
* 클래스 문법 활용 예



- 날개 변수 사용

- 메서드 문법 활용
  ↳ 파라미터 사용

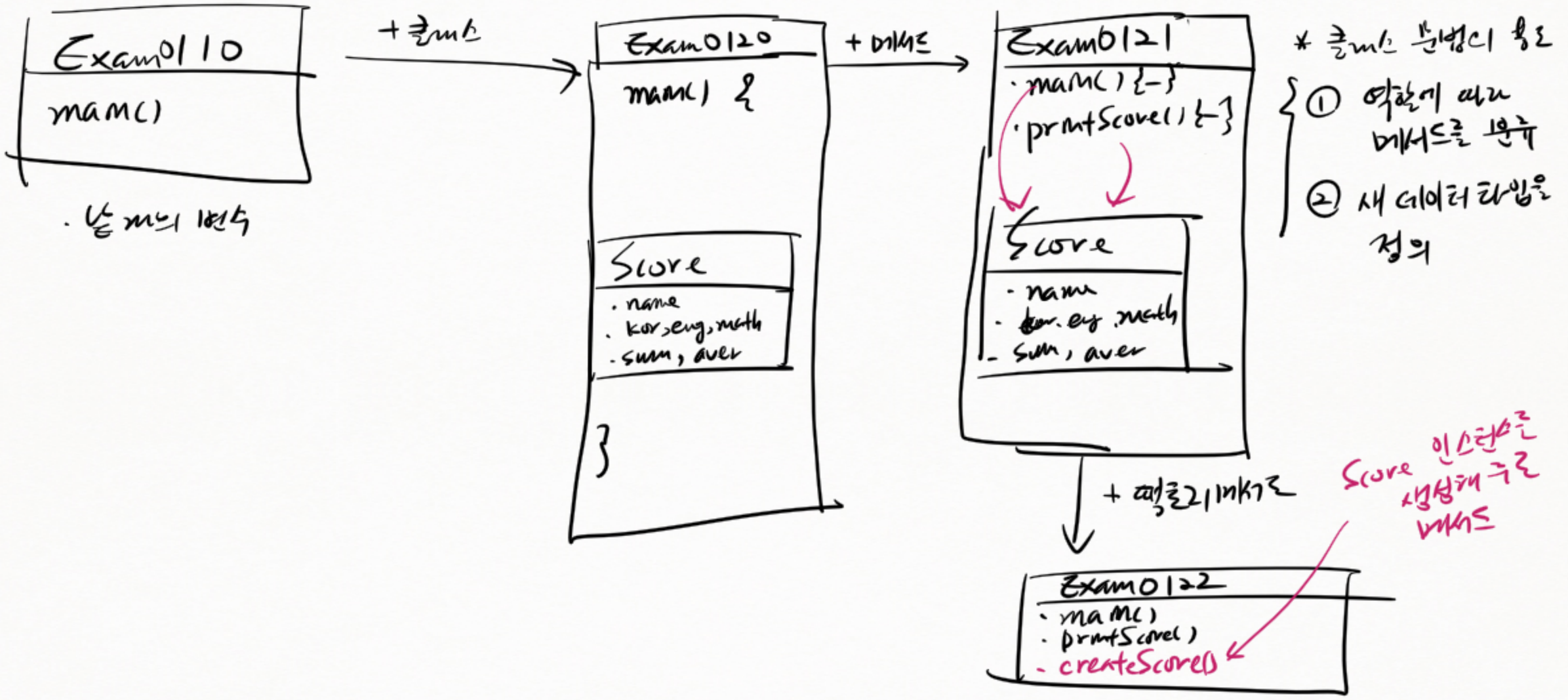
⇓

✓ 중복 코드 제거

↓

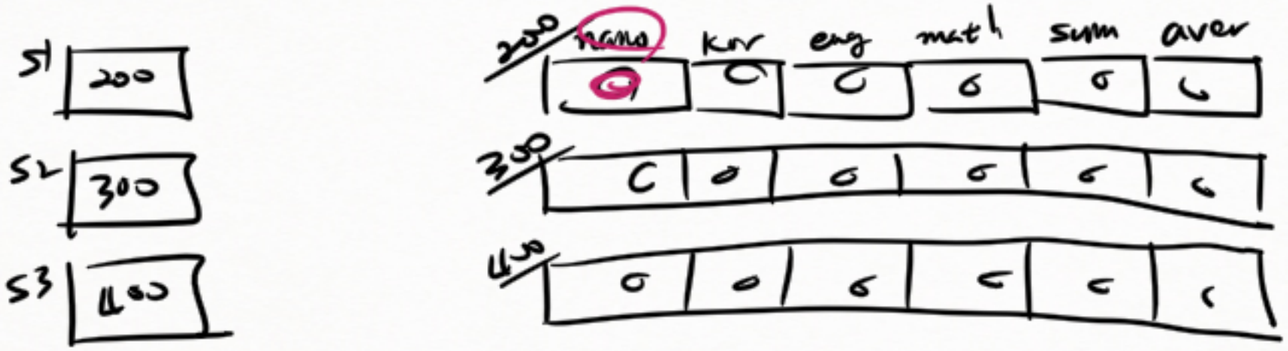
코드 재사용 ↑

✓ 유지 보수가 쉬워짐


Exam0110
main()
+클래스
Exam0120
main() {
Score
. name
. kor, eng, math
. sum, aver
}
+메서드
Exam0121
. main() {-}
. prntScore() {-}
Score
- name
- kor. eng. math
- sum, aver
* 클래스 문법의 목적
① 알맞게 따라 메서드를 호출
② 새 데이터 타입을 정의
+ 팩클리메서드
Exam0122
. main()
. prntScore()
- createScore()
Score 인스턴스를 생성해 주는 메서드

* 레퍼런스 변수 ⟶ 여러 개의 인스턴스를 다룸

Score s1, s2, s3 ;



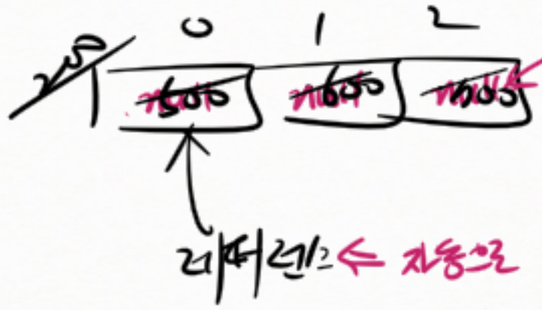
s1.name = "";

s1.kor = 100;

⋮

* 레퍼런스 배열

```
Score[] arr = new Score[3];
```

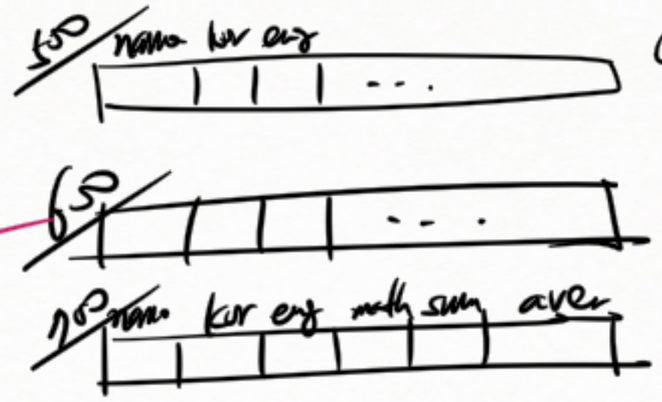


null ?
- 클래스가 아니다!
- 참조변수가 0으로 초기화되었음을 뜻한다.

레퍼런스 ⇐ 자동으로 null 값으로 초기화된다
*로컬변수는 초기화되지 않는다.

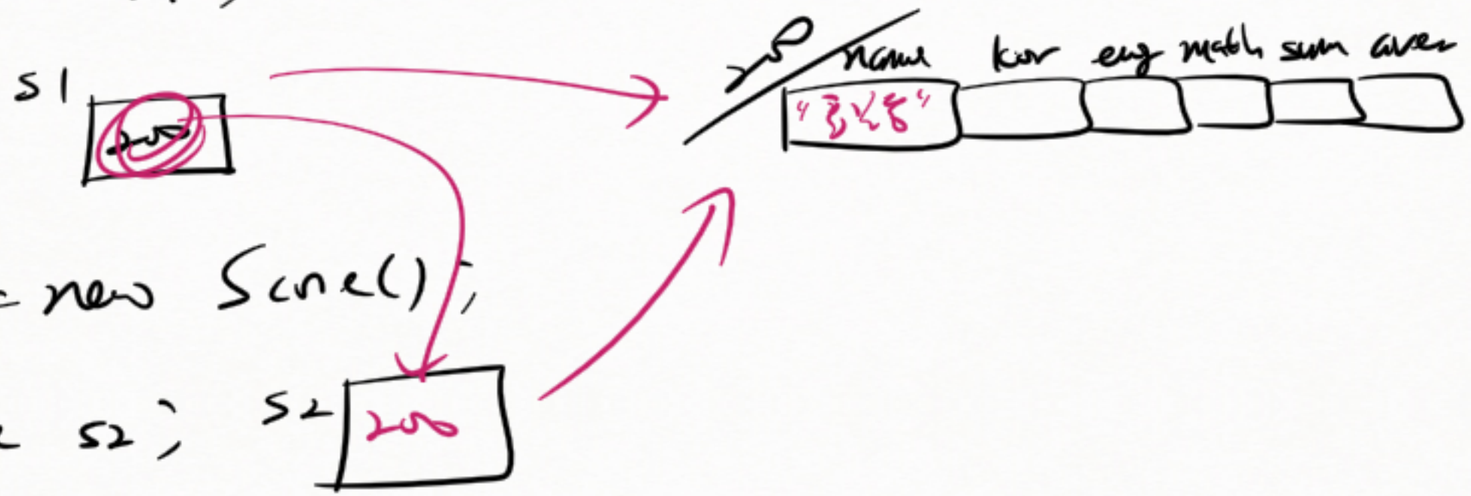
```
arr[0] = new Score();
arr[1] = new Score();
arr[2] = new Score();
arr[3] = new Score();
```

*ArrayIndexOutOfBoundsException



* 레퍼런스 와 인스턴스

Score s1;



s1 = new Score();

Score s2; s2 200

s2 = s1;

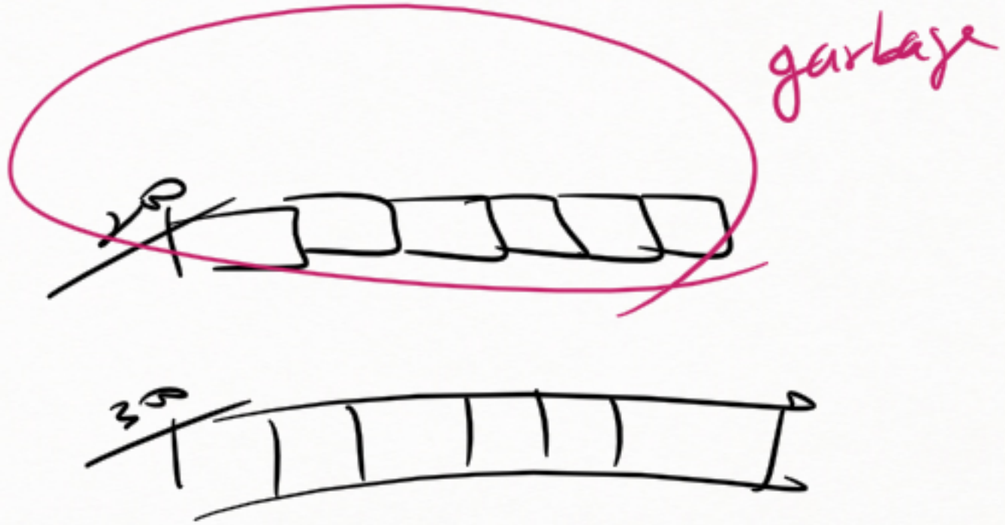
s2.name = "홍길동";

* 가비지 (garbage)

Score s1;

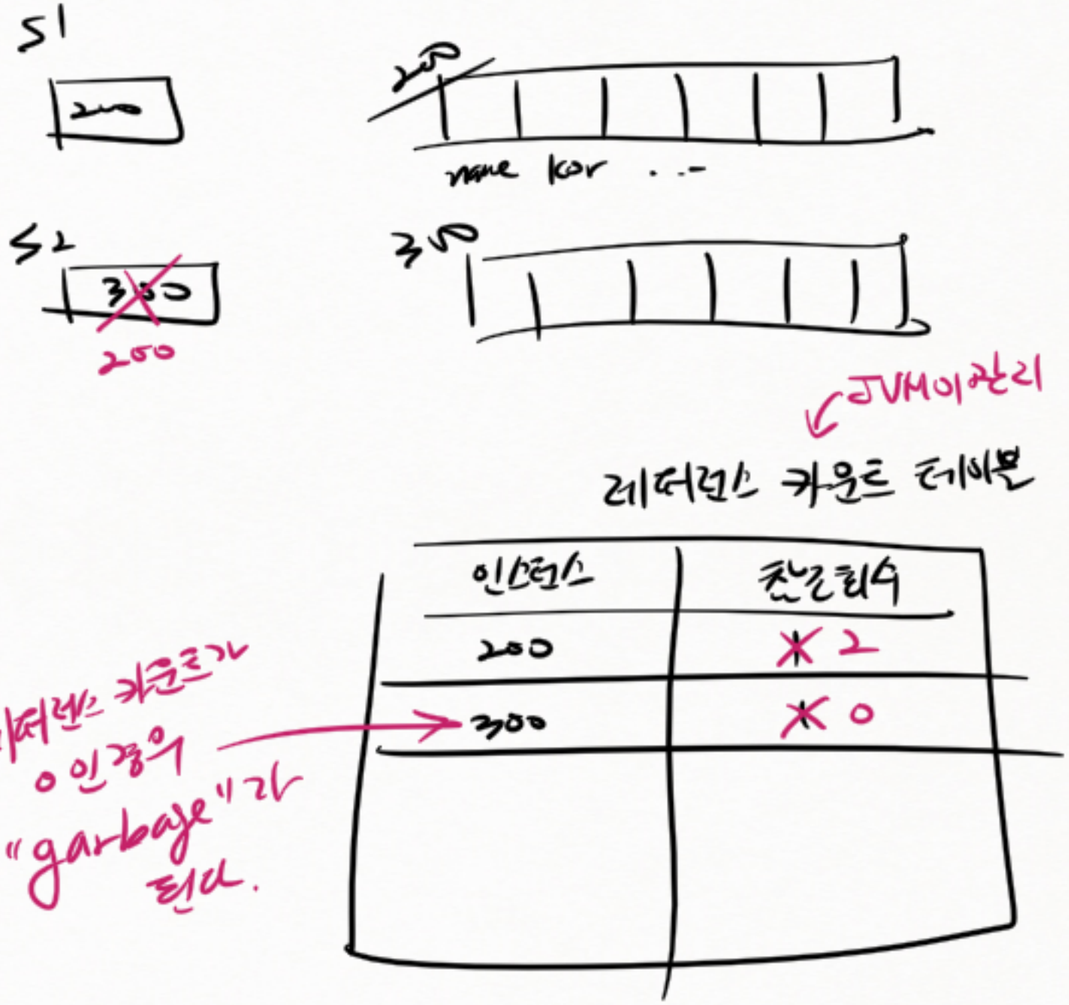
s1 = new Score();

s1 = new Score();



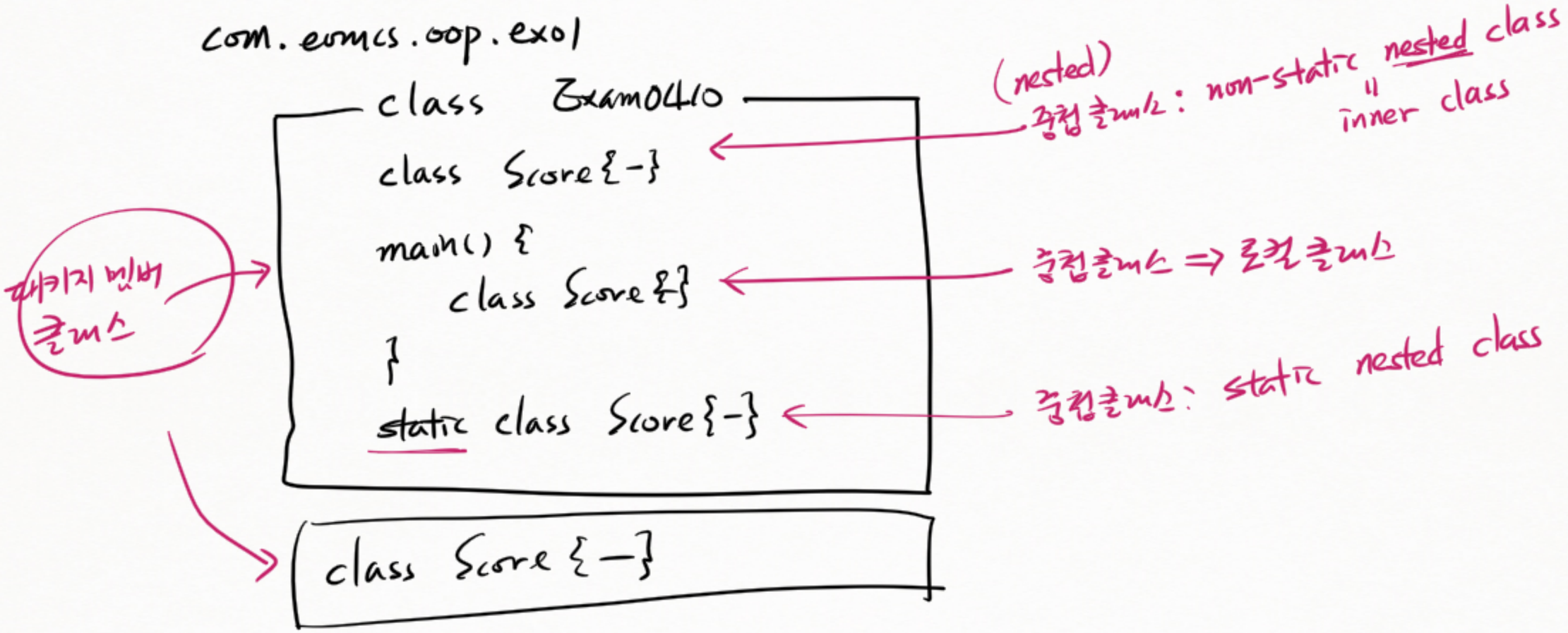
* 레퍼런스 카운트란 가비지

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

s2 = s1;
```

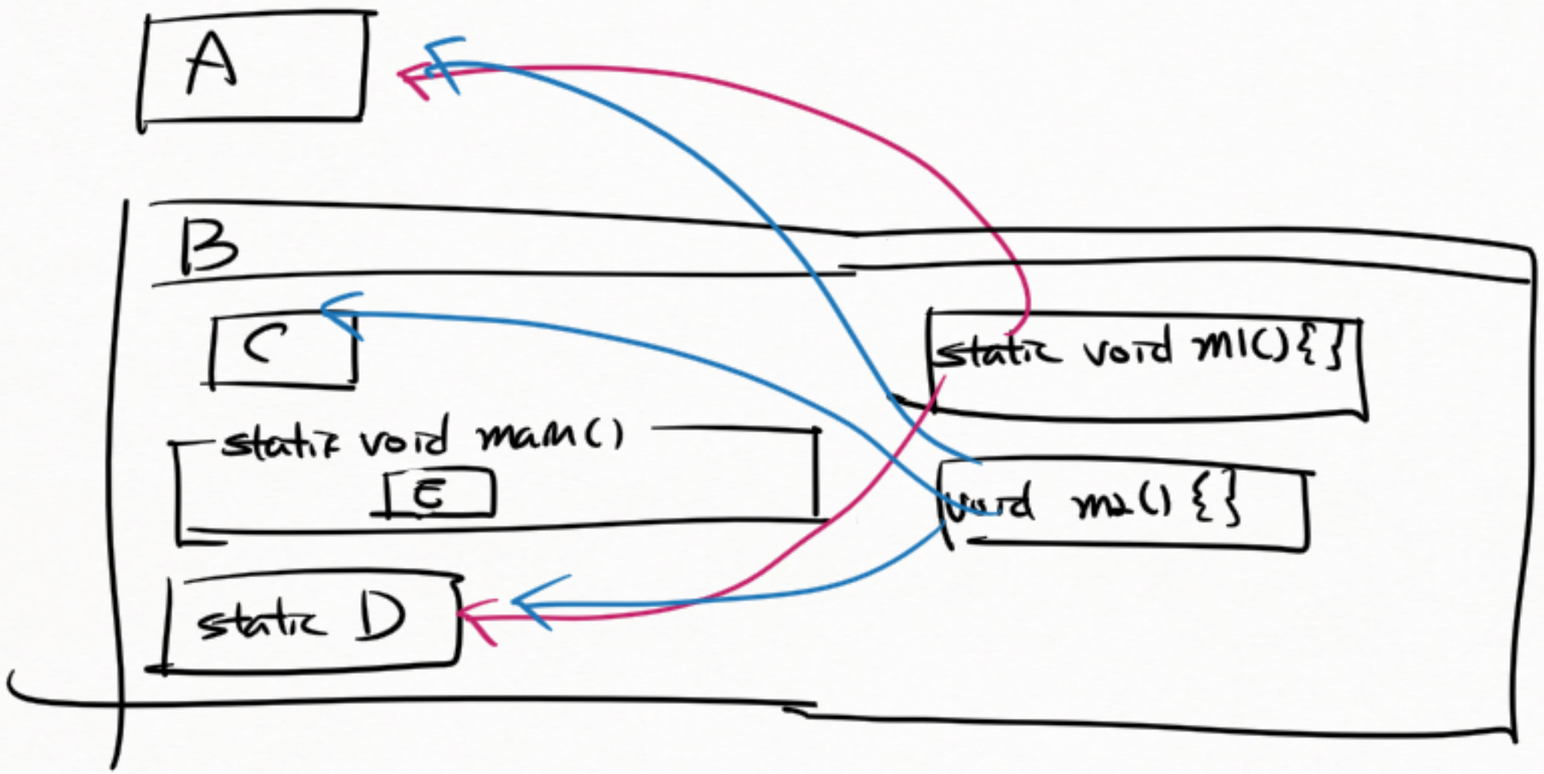


* 클래스 리뷰

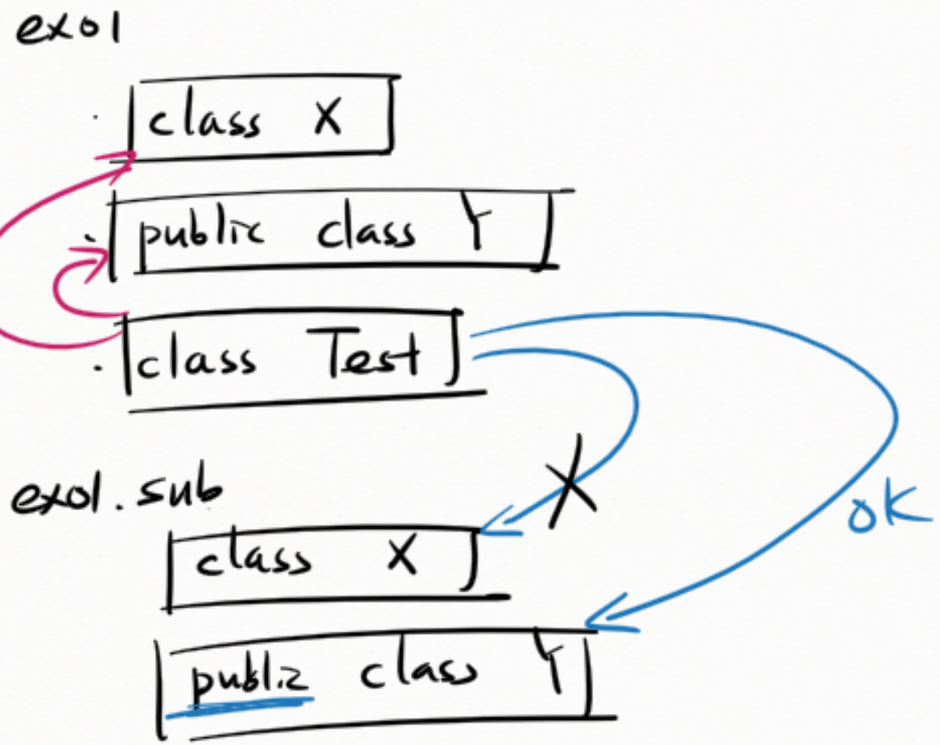
com.eomcs.oop.ex01



* static 멤버와 non-static 멤버



* 공개 클래스와 기본 클래스



* 클래스 문법의 활용 예: ① 사용자 정의 데이터 타입 을 만드는 용도

User-defined Data Type
개발자

Score s1 = new Score();

s1.name = "홍길동"

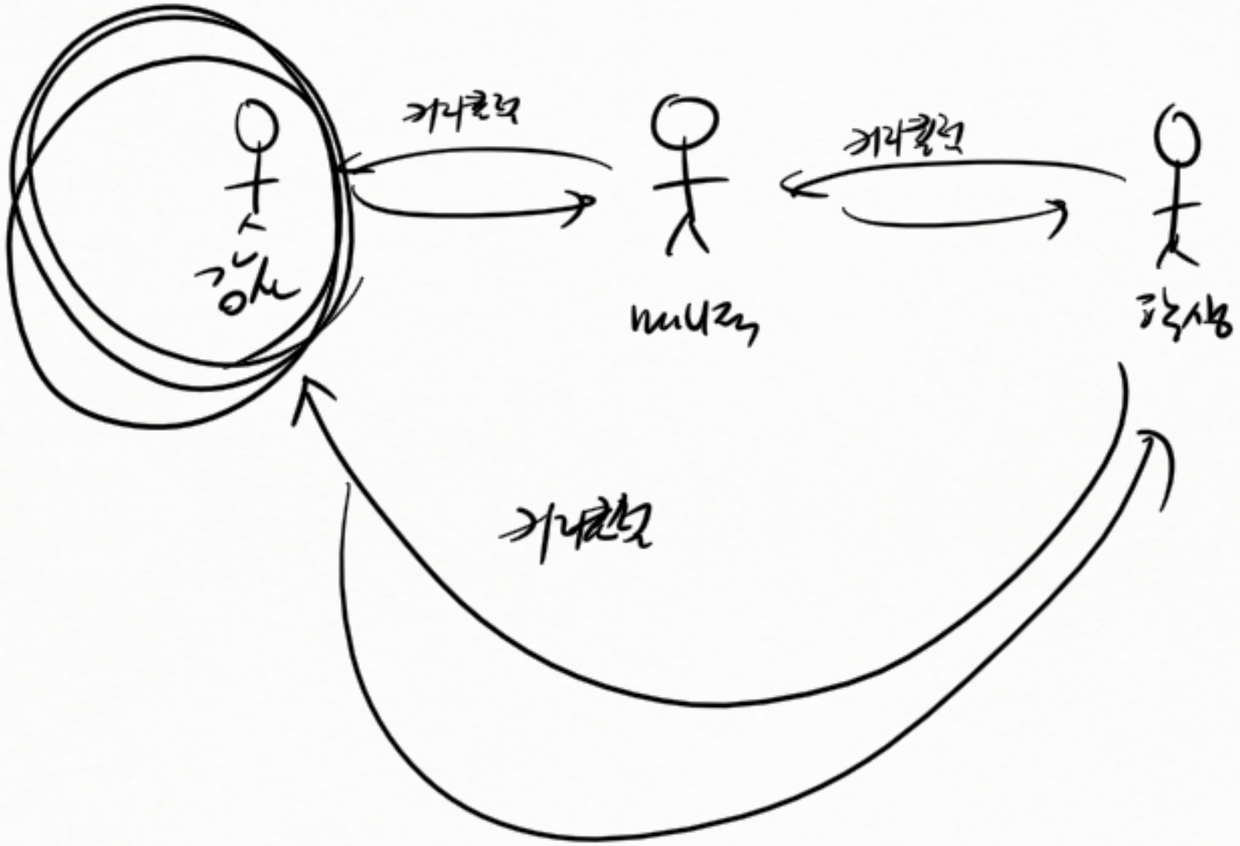
① s1 레퍼런스에 저장된 주소로 찾아가서 해당 인스턴스의 name 변수 —
② s1 레퍼런스가 가리키는 인스턴스의 name 변수 —
③ s1 인스턴스의 name 변수 —
④ s1 객체의 name 변수 (필드)
⑤ s1의 name 필드 (변수)

※ optimizing(최적화) vs refactoring(코드 깨끗)

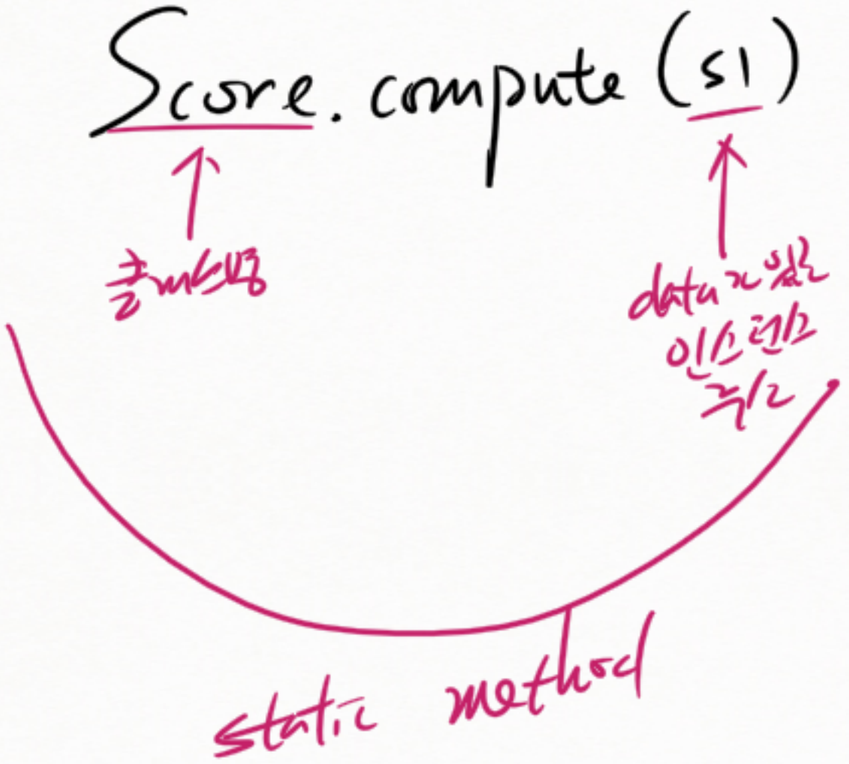
optimizing(최적화)
↓
. 속도↑
. 코드 난해 → 유지보수↓

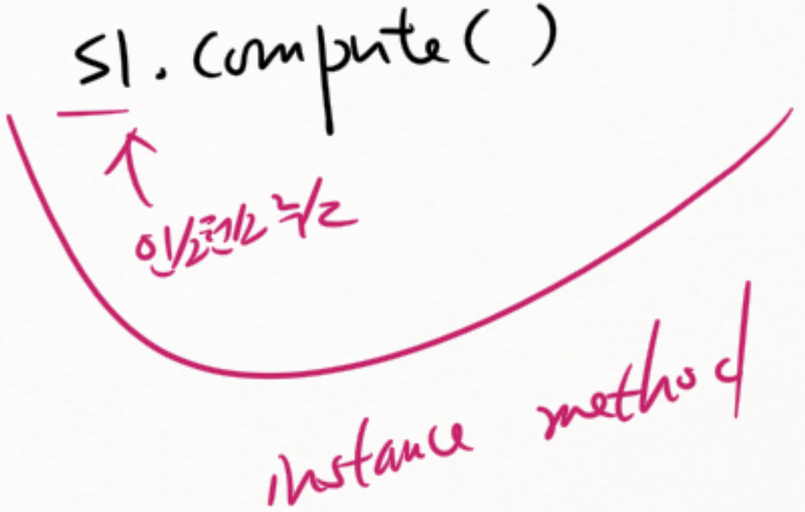
refactoring(코드 깨끗)
↓
. 유지보수 용이성↑
. 속도↓

x GRASP : 클래스의 책임 및 협력을 +



* static 메서드 vs 인스턴스 메서드





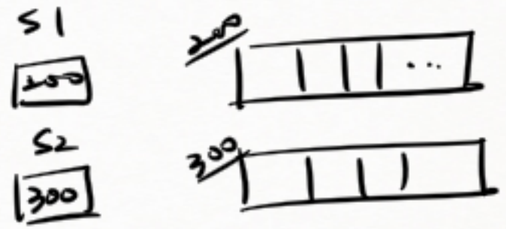
* 인스턴스 메서드 와 연산자

```
i ++ ;
```

operand (피연산자) ↗ ↖ operator (연산자)

```
j ++;
```

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



인스턴스 주소 ↘ ↙ 메서드

```
s1 . compute()
```

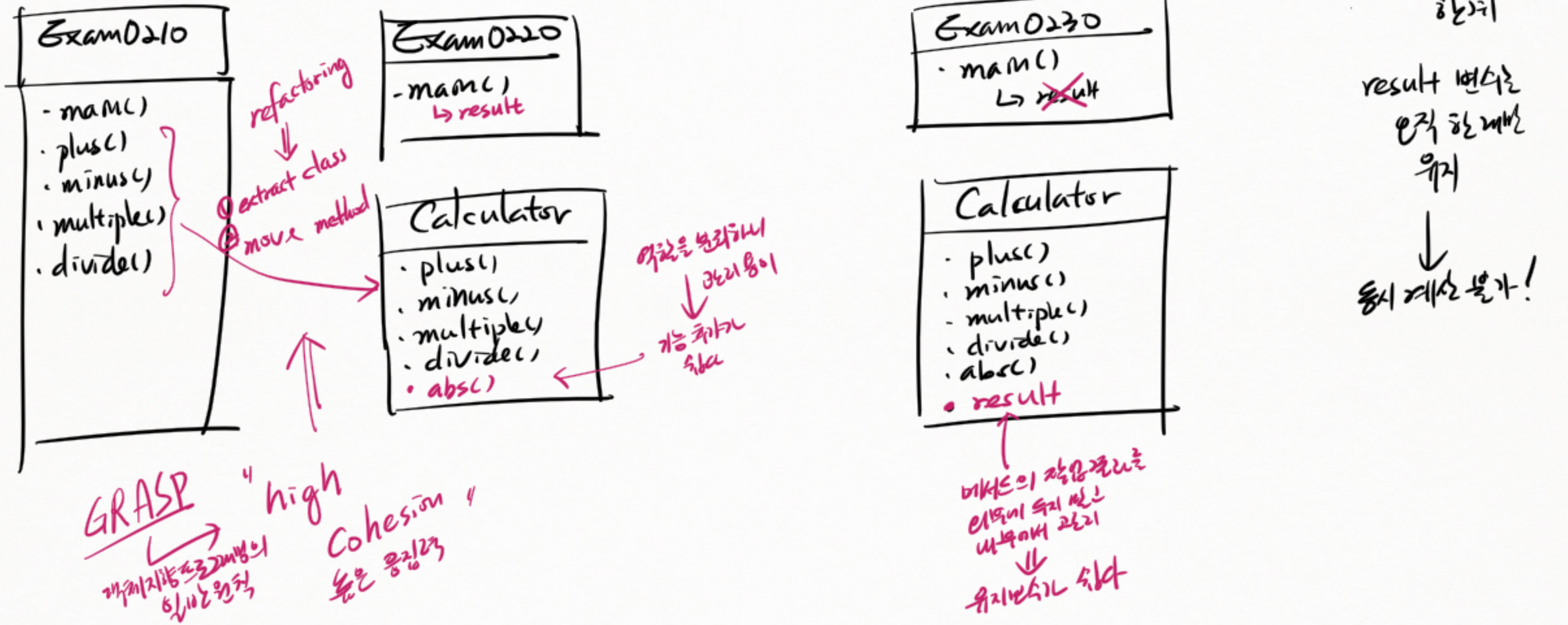
operand ↑ ↑ operator

```
s2 . compute()
```

↑

* 클래스 변경 활용 : 메서드를 위한 분리하기

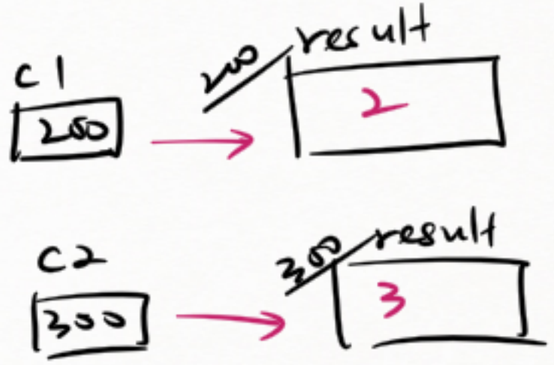
① 분리 전 → ② 메서드 분리 → ③ 클래스 변수 (static 변수 = 클래스 변수) → ④ 클래스 변수의 한계



* 클래스 분리 활용 : 메서드를 묶어 관리하기

→ ① 인스턴스 변수 도입

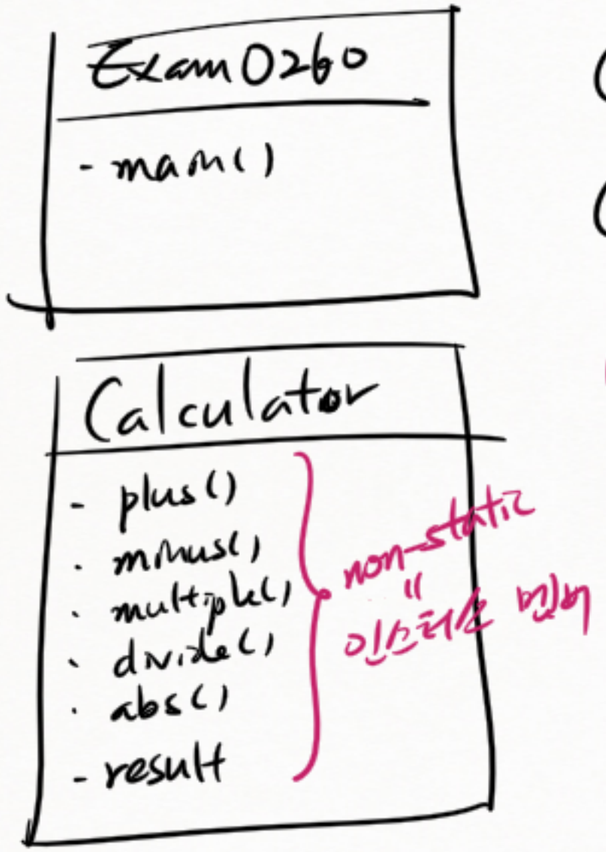

```
Calculator c1 = new Calculator();
Calculator c2 = new Calculator();

Calculator.plus(c1, 2);
Calculator.plus(c2, 3);
:
```



작업대상은 각각의 result 변수가 있는 인스턴스 주소를 알려줄거.

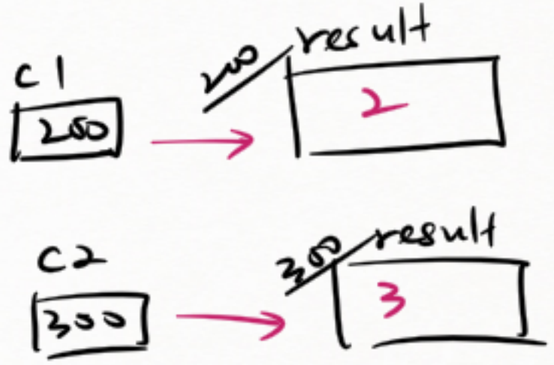
* 클래스 문법 활용 : 메서드를 통해 프로그램

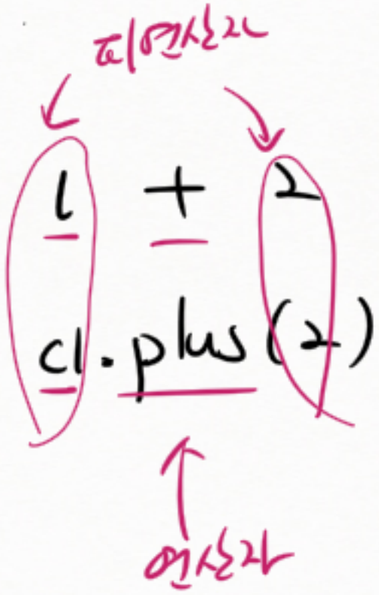
→ ⑥ 인스턴스 메서드 활용




```
Calculator c1 = new Calculator();
Calculator c2 = new Calculator();
c1.plus(2);
c2.plus(3);
```

c2 → 인스턴스 참조값





* 클래스 분배 활용 : 메서드를 묶어 관리하기

→ ① 패키지 없이 클래스

Exam0240
- main()

Calculator
- plus() - minus() - multiple() - divide() - abs() - result

non-static = 인스턴스 변수

com.eomcs.oop.ex02.

Exam0240.java
Calculator.java

→ ② 패키지 분류

com.eomcs.oop.ex02.Exam0240

com.eomcs.oop.ex02.util.Calculator

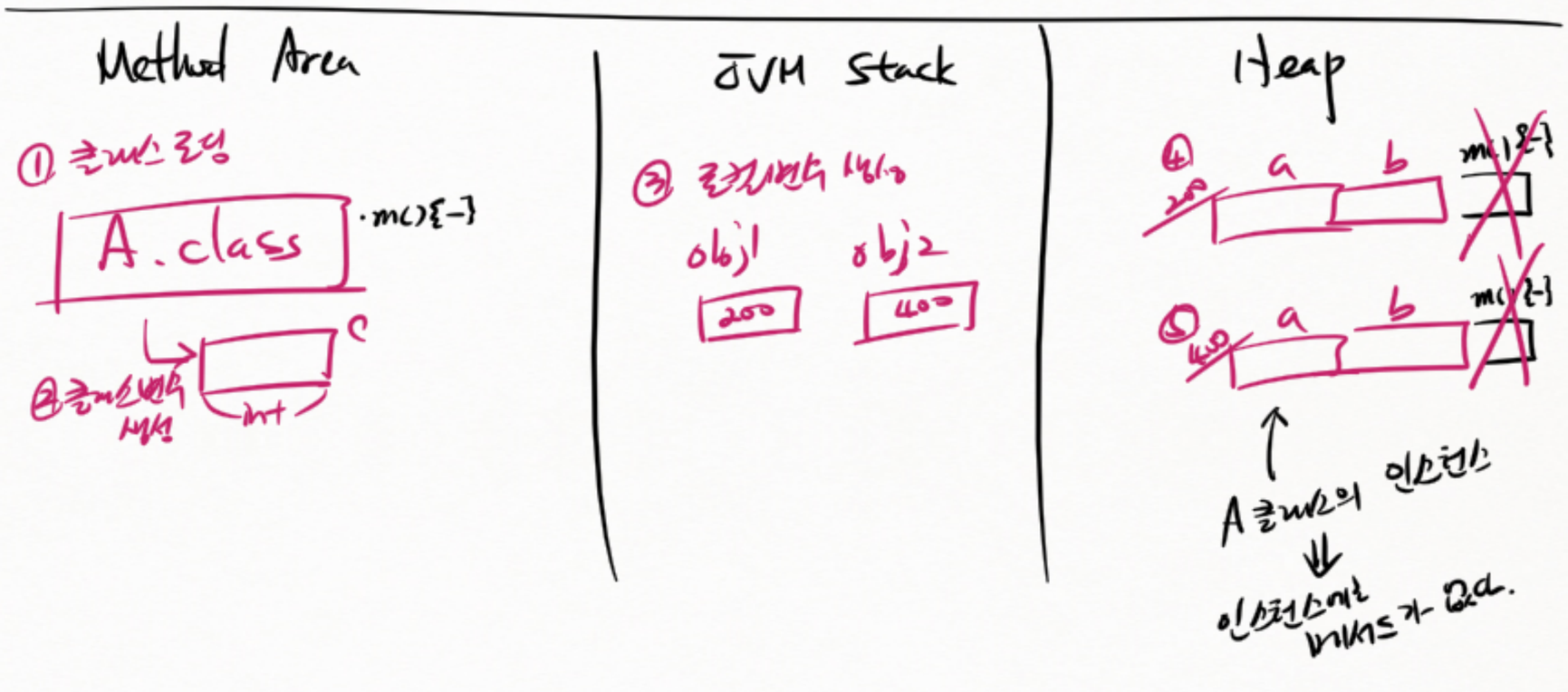
import com.eomcs.oop.ex02.util.Calculator;

클래스를 사용할 때 반드시 선언해 주어야 하는 코드.

* 클래스 변수 vs 인스턴스 변수

static 변수 / non-static 변수

```
class A {
    int a;      // 인스턴스 변수
    int b;      // 인스턴스 변수
    static int c;   // 클래스 변수
    void m() {}
}

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



* 인스턴스 변수 vs 인스턴스 메서드