

class

* 팀스 운영의 패턴

- ① 멤버는 문구 : MemberHandler
Prompt
- ② 새 아이디 태입을 확장 : Score, Member

* 새 데이터 타입 정의

① class 정의

```
class Score {  
    String name;  
    int kor;  
    int eng;  
    int math;  
    int sum;  
    float aver;  
}
```

"인스턴스 변수" (field)

④ 인스턴스 초기화

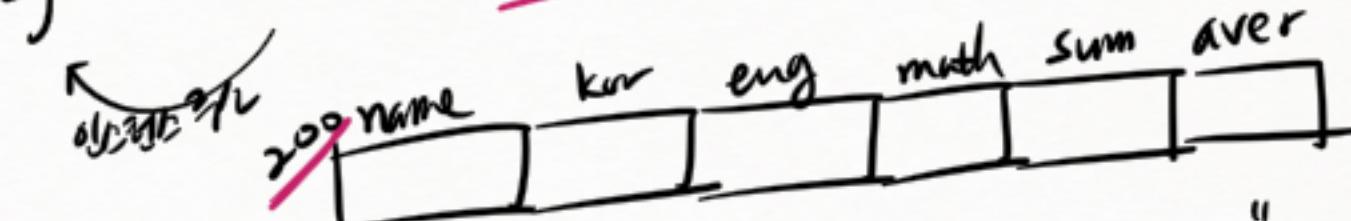
obj. name = "홍길동"

② 인스턴스 할당

```
Score obj;
```

obj
200

obj = new Score();



③ 인스턴스 사용

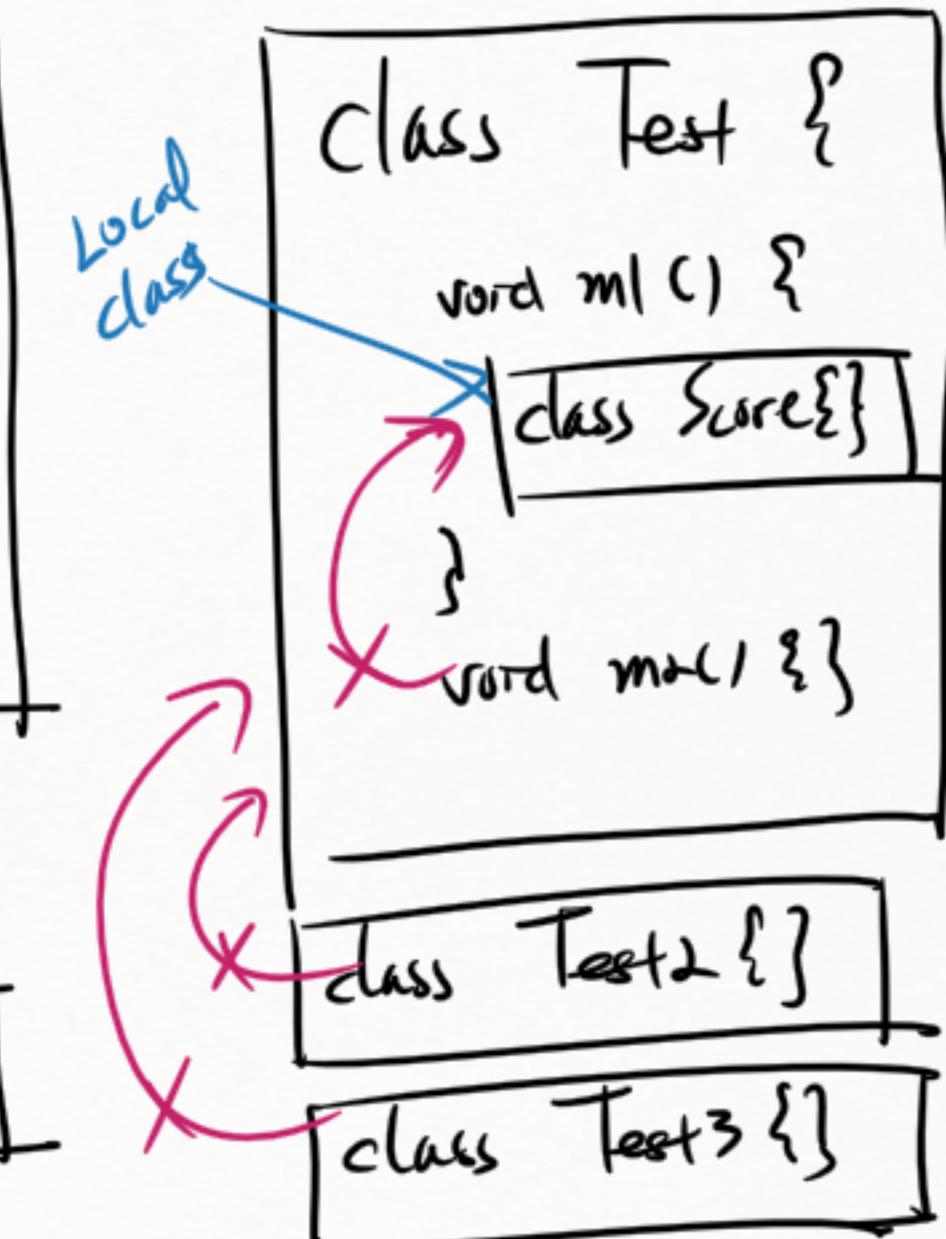
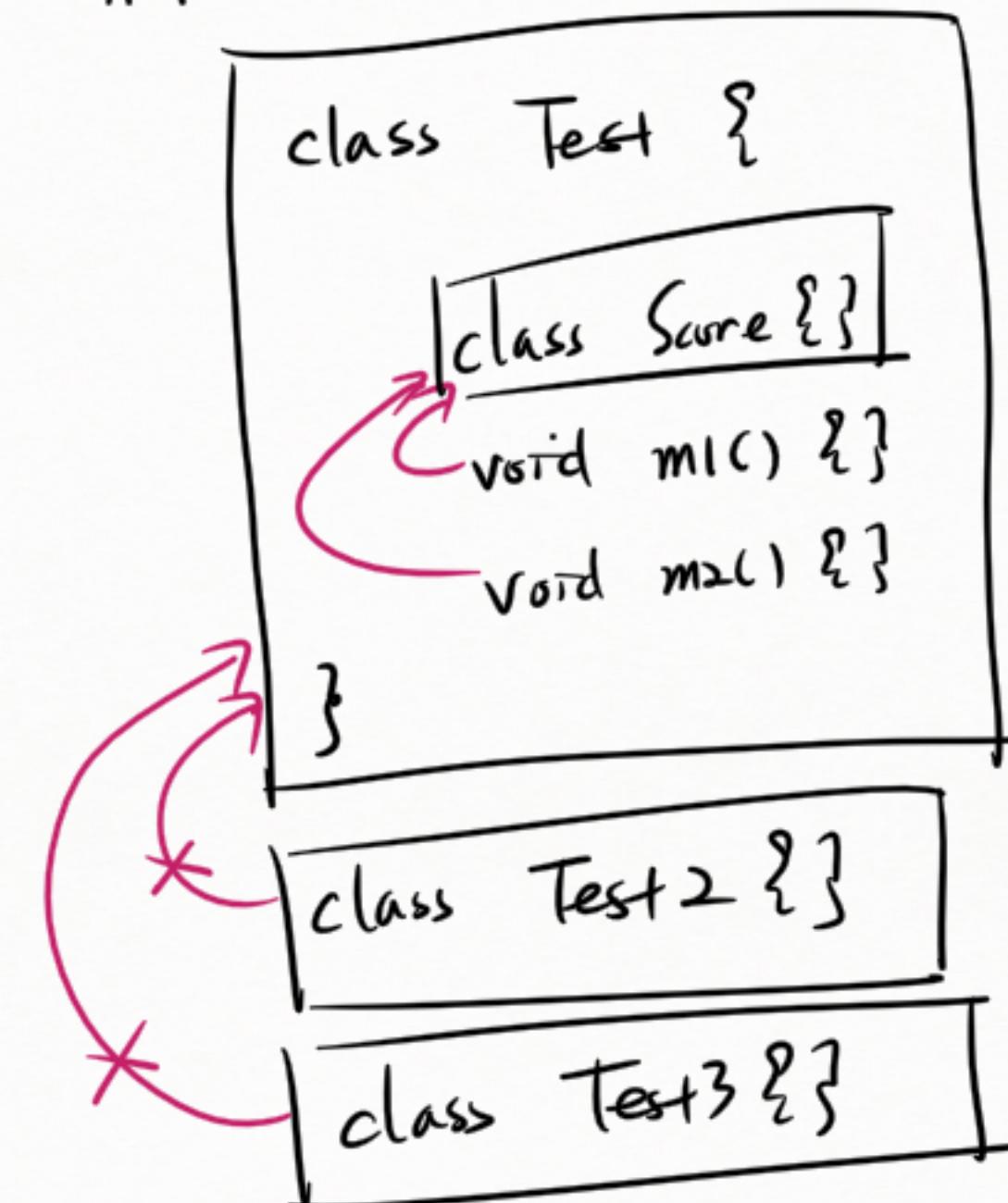
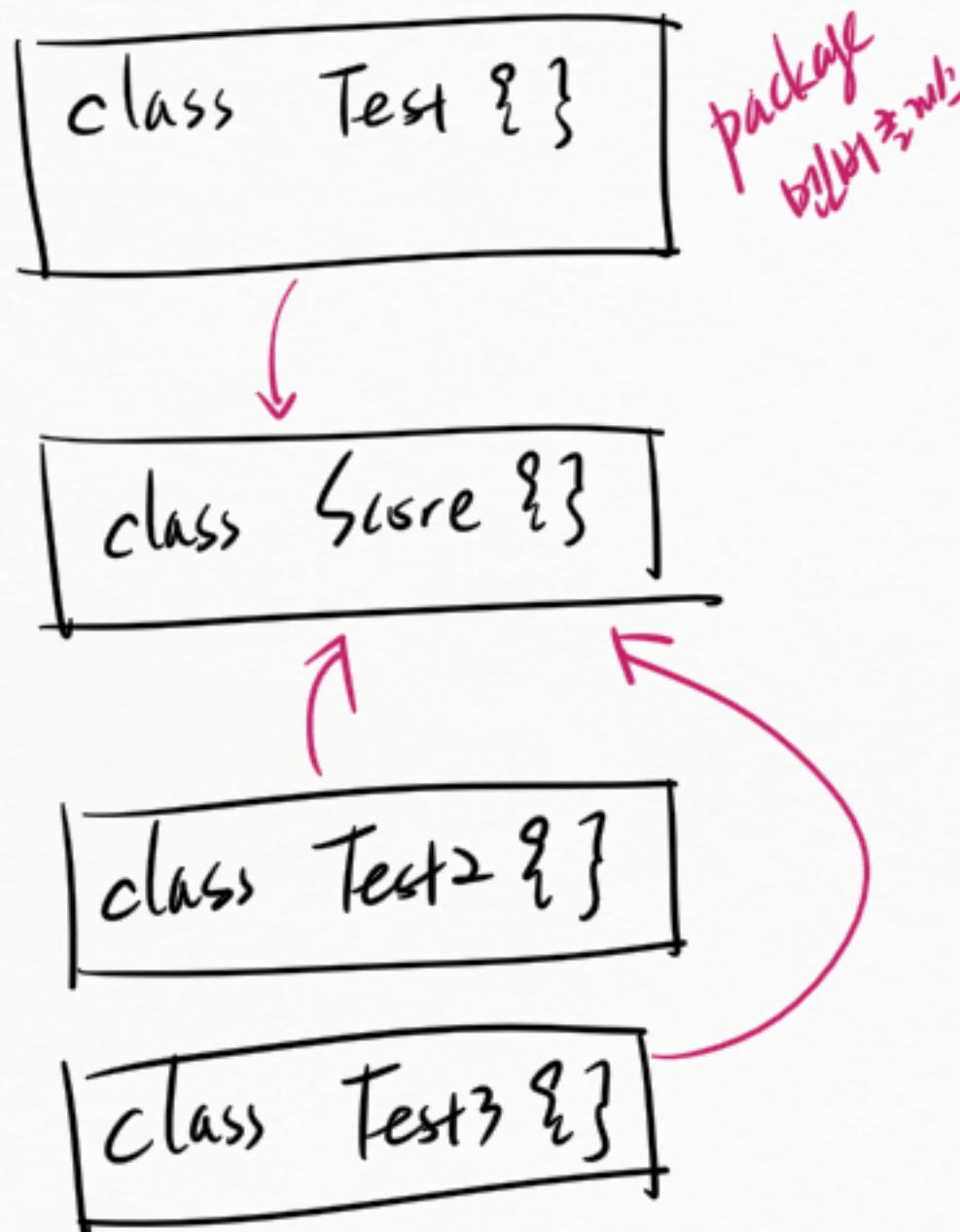
"Score의 인스턴스"
instance

"reference"

↑
Score의 인스턴스 주소를 저장하는 변수

Heap
obj → 200
new Score()
name → 홍길동
kor eng math sum aver

* 클래스 정의의 유형



"Nested class"

* 인스턴스 생성, 메모리, call by reference

Score s = new Score();

s
200

200	name	kur	eng	math	sum	aver
	String	int	int	int	int	float
	한국어	100	90	80	270	90.0

s.name = "한국어";

s.kor = 100;

s.eng = 90;

s.math = 80;

s.sum = s.kor + s.eng + s.math;

s.aver = s.sum / 3f;

printScore(s);

인스턴스의 주소

printScore(Score s) {

}

System.out.printf();

}

* 디자인한 인스턴스 생성 후 출력

```
Score s = createScore("김민수", 100, 100, 100);    createScore (String name, int kor, int eng, int math) {  
    Score s = new Score();  
    s.name = name;  
    s.kor = kor;    s.sum = _____;  
    s.eng = eng;    s.aver = _____;  
    s.math = math;  
    return s;  
}  
s | 200  
  |  
  +-----+-----+-----+-----+-----+-----+  
  | name | kor | eng | math | sum | aver |  
  | "김민수" | 100 | 100 | 100 | 300 | 100.0 |
```

* 커스텀 클래스 사용 예

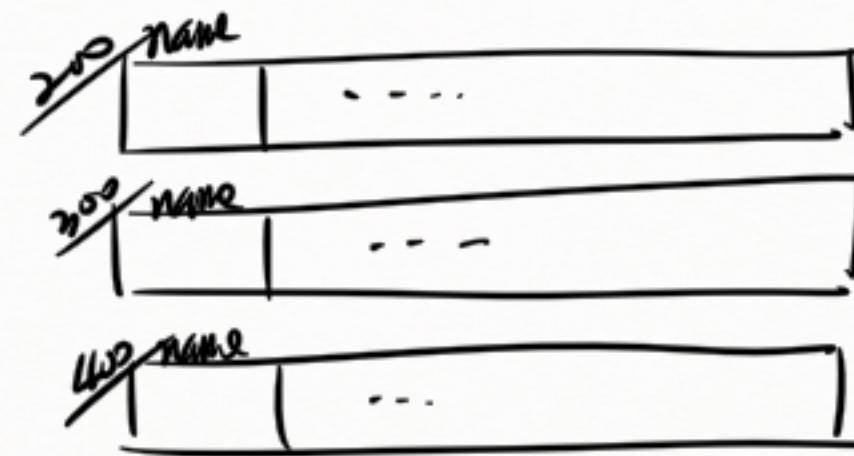
Score s1, s2, s3;

s1 | 200 s2 | 300 s3 | 400

s1 = new Score();

s2 = new Score();

s3 = new Score();



* 리퍼런스 변수 사용 후
 ↗️ 리퍼런스 변수
 ↗️ 리퍼런스들의 대체

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

`scores`

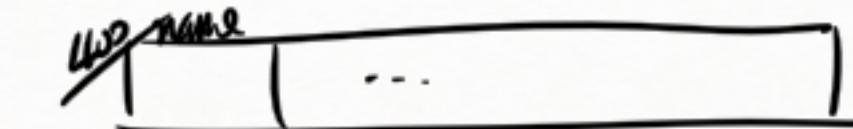
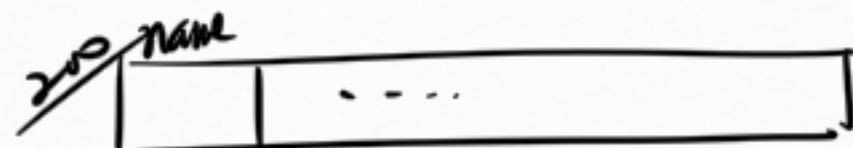
`1700`



`scores[0] = new Score();`

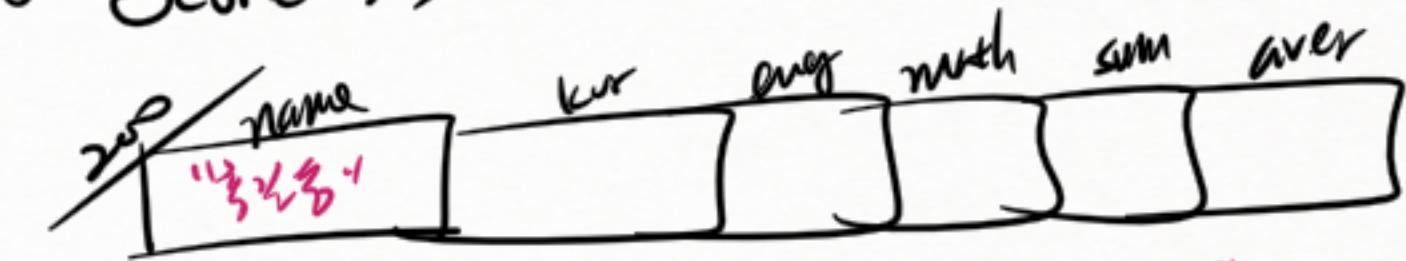
`scores[1] = new Score();`

`scores[2] = new Score();`



* 인스턴스와 메서드:

Score s1 = new Score();



"Score의 인스턴스"
개념

Score s2 = s1;

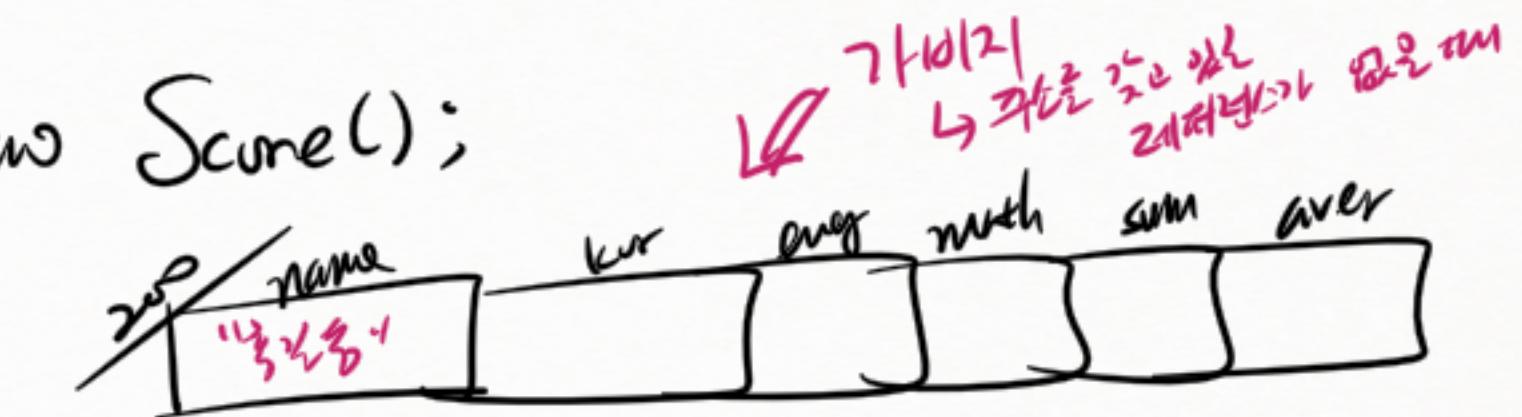


* 111011 (Garbage)

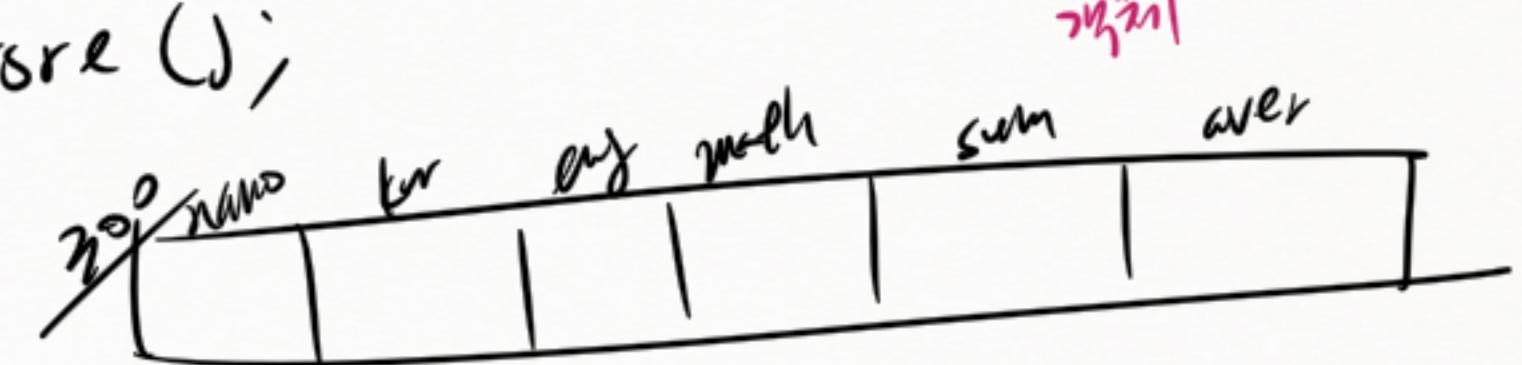
Score s1 = new Score();



s1 = new Score();



"Score의 인스턴스"

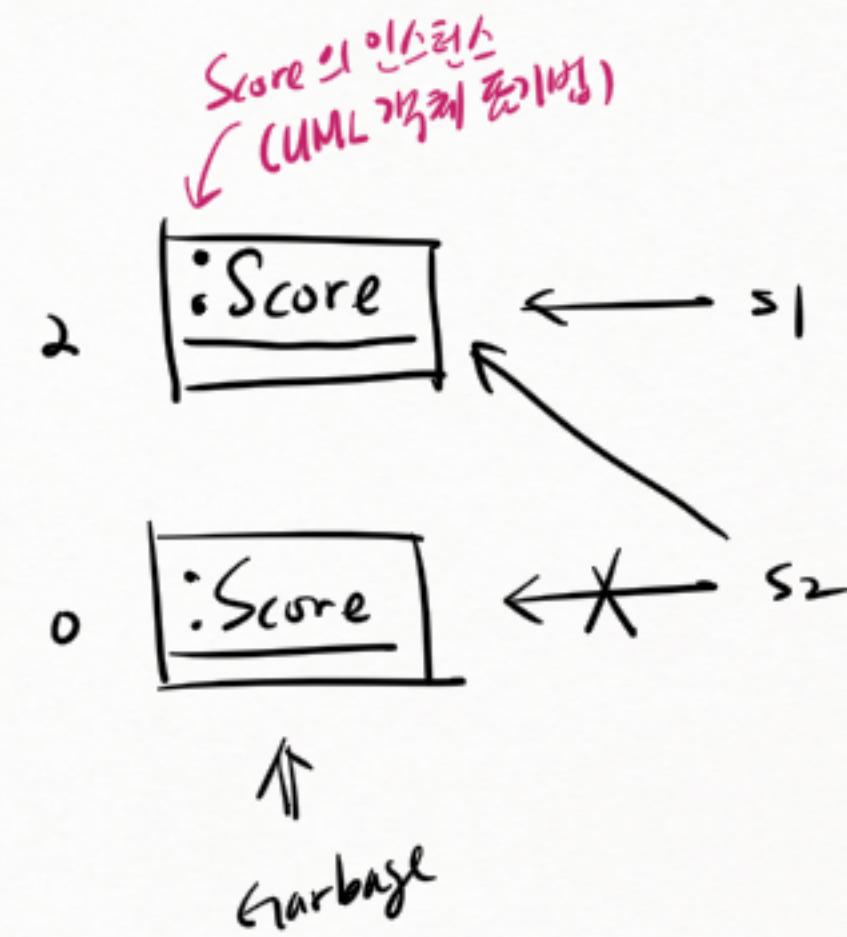


* 인스턴스와 리퍼런스 차운트

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

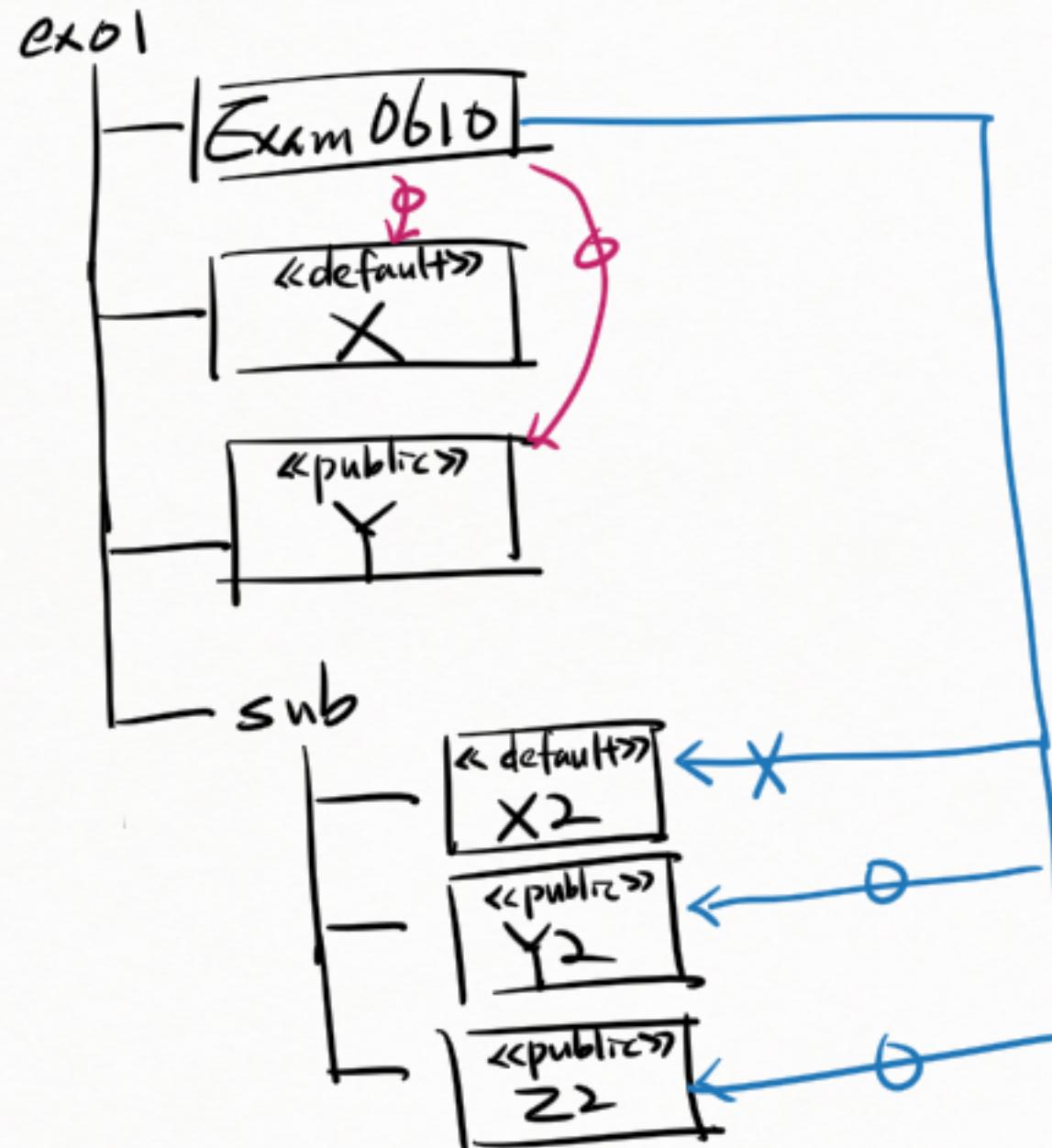
```
Score s2 = new Score();
```

```
s2 = s1;
```



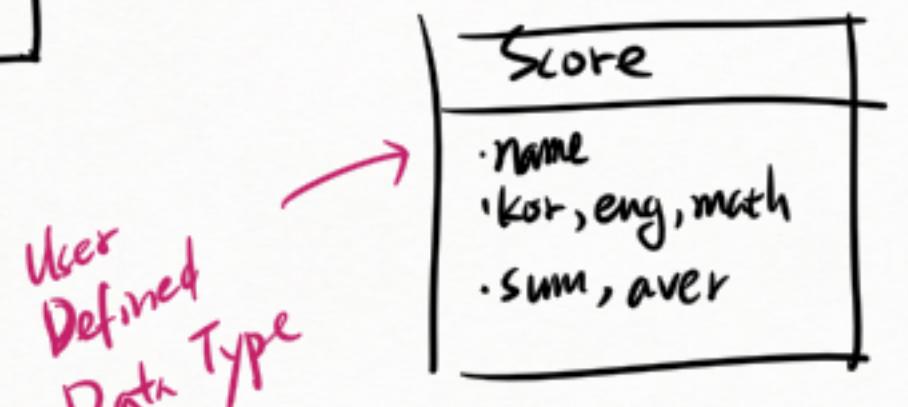
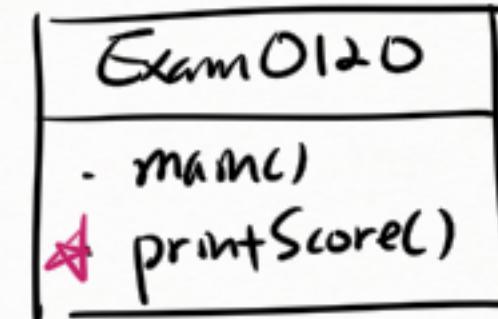
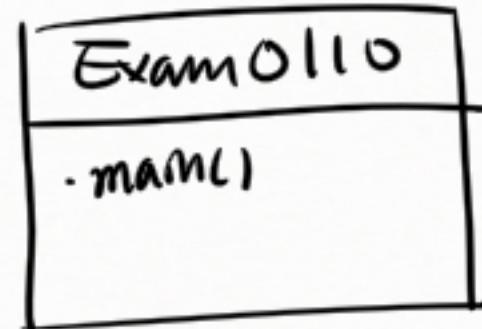
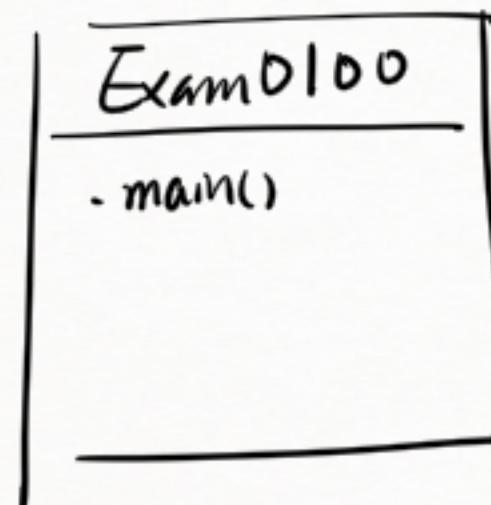
* public $\frac{3}{2}$ mLcf

default $\frac{3}{2}$ mL
(package private class)



* com. cs. oop. ex02. Exam01xx

① 놓개 변수 사용 → ② class, 블법: 새 데이터 타입 정의 → ③ method 블법: 중복코드 제거



- 메모리와 인스턴스
- ↓
 new
 ↓
 Heap 영역
 ↓
 garbage
 ↓
 garbage collector

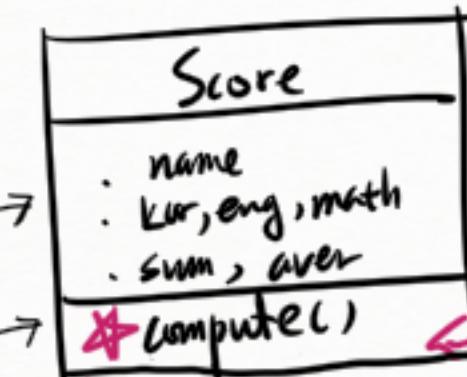
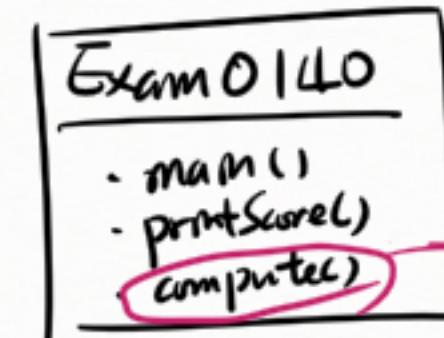
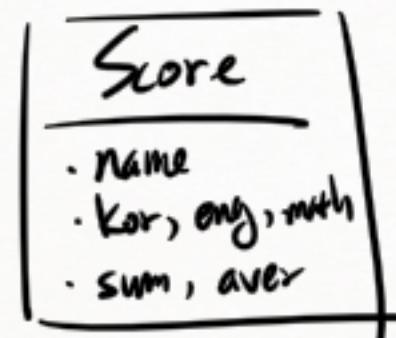
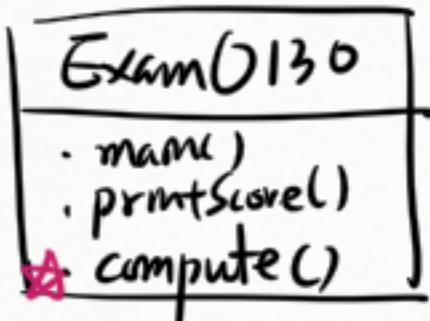


④ 리팩토링: 1기능 → 1 메서드

⑤ 리팩토링: 멤버드 이동

⑥ 인스턴스에 더 쉽게 접근하는 법: 인스턴스 메서드

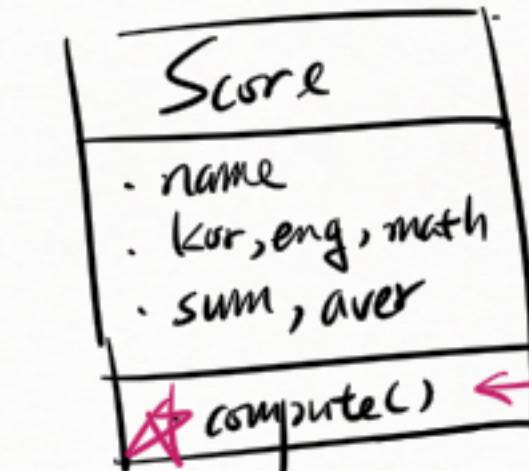
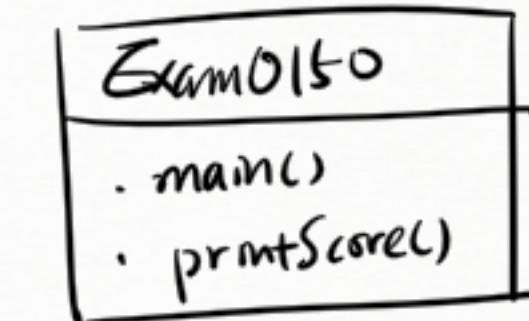
non-static



class
속성
↓
data를 정의한
그 데이터를 다루는
operator를 만든다.

GRASP의
Information
Expert

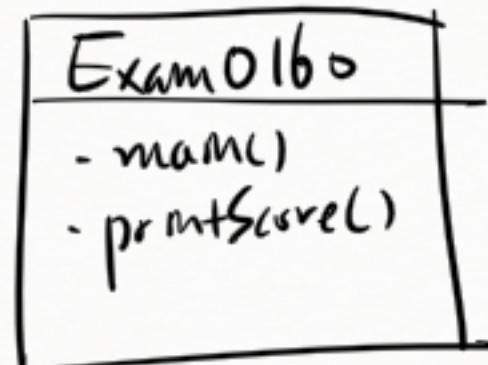
이동



non-static 으로
변경

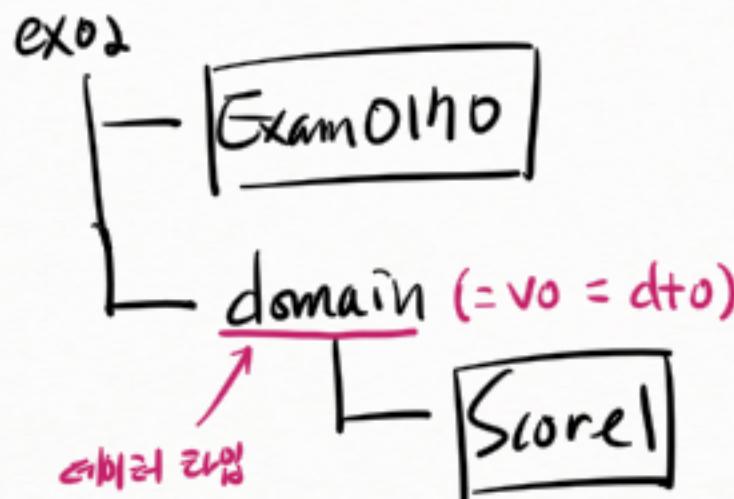
이전 방식
Score. compute(인스턴스주소) → 변경 후
인스턴스주소. compute()

① 패키지 멤버 클래스



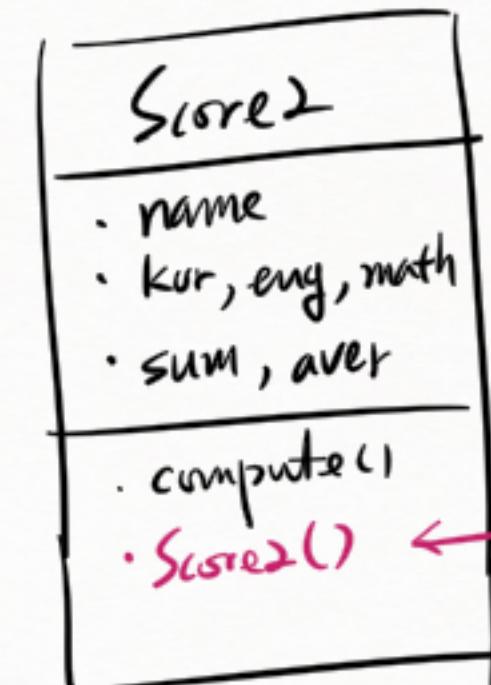
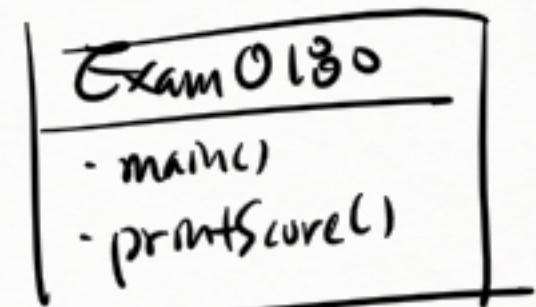
nested class → package member
변경

② 클래스를 패키지로 분리
관리 용이



- 접근제어 특성
- public : 외전용
 - protected : 내부클래스, 같은 패키지
 - (default) : 같은 패키지
 - private : 외용

③ 객체 초기화 방법 : 생성자



* 스태틱 메서드와 인스턴스 메서드

static 메서드 \Rightarrow Score. compute(s1);
||
클래스 메서드

메서드가 소속된 클래스

non-static 메서드 \Rightarrow s1. compute();
||
인스턴스 메서드

메서드가 소속된 클래스의 인스턴스 주는
~~~ 인스턴스를 보다 쉽게 다루는  
메서드 문법

\* 인스턴스 메서드와 this

인스턴스  
메서드를 호출할 때  
this는  
매개변수로  
传여  
된다.  
이쪽에서 넘겨온 인스턴스 주소를 받는  
built-in 로컬 변수

non-static 멤버에만 존재한다.

\* this 가는  
this 멤버 변수에 저장된다

↓  
인스턴스 주소를 블리지  
하고 멤버를 접근할  
수가 있다.

```
void compute() {  
    this.sum = this.kor + this.eng + this.math;  
    :  
}
```

\* 생성자

class Score {

    ↓  
    → Score(int num, ... ) {

        }  
        =

}

## \* 생성자呼び出し

① 이름

```
Score s = new Score();
```

```
s.name = "홍길동";
```

```
s.kor = 100;
```

```
s.eng = 100;
```

```
s.math = 100;
```

```
s.compute();
```

} Score 객체 생성  
이스턴스 생성

생성자 호출

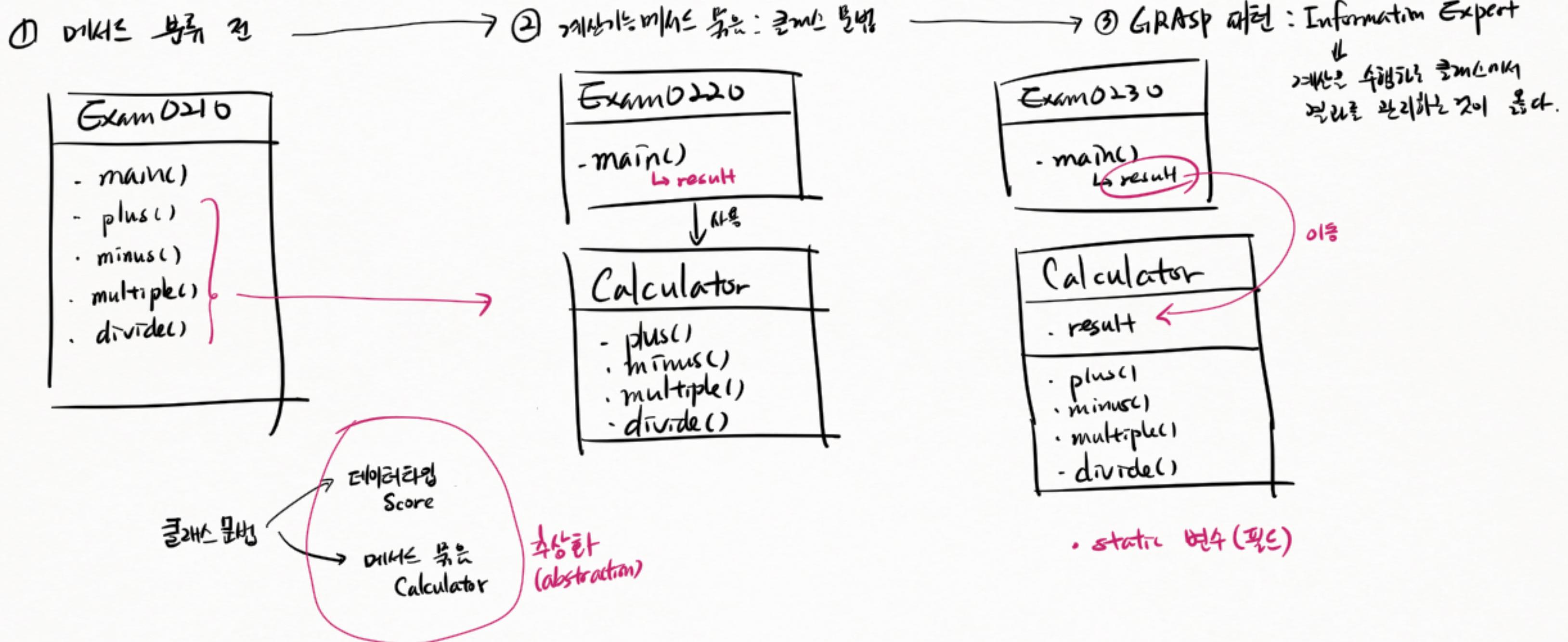
② 내용

```
Score s = new Score("홍길동", 100, 100, 100);
```

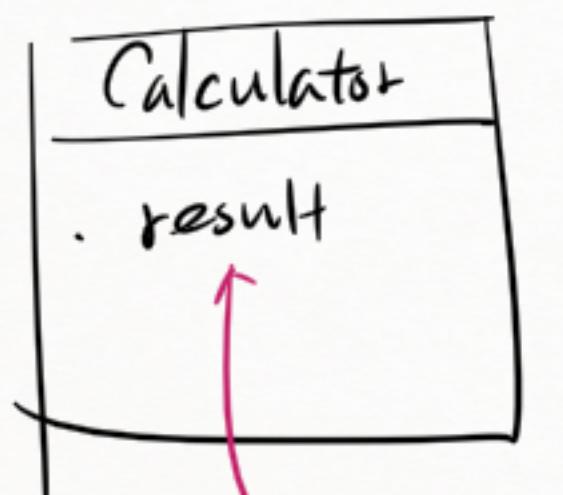
↓ 이스턴스 생성과 즉시 생성자 자동호출

```
Score (String n, int k, int e, int m){  
    this.name = n;  
    this.kor = k;  
    this.eng = e;  
    this.math = m;  
    this.compute();  
}
```

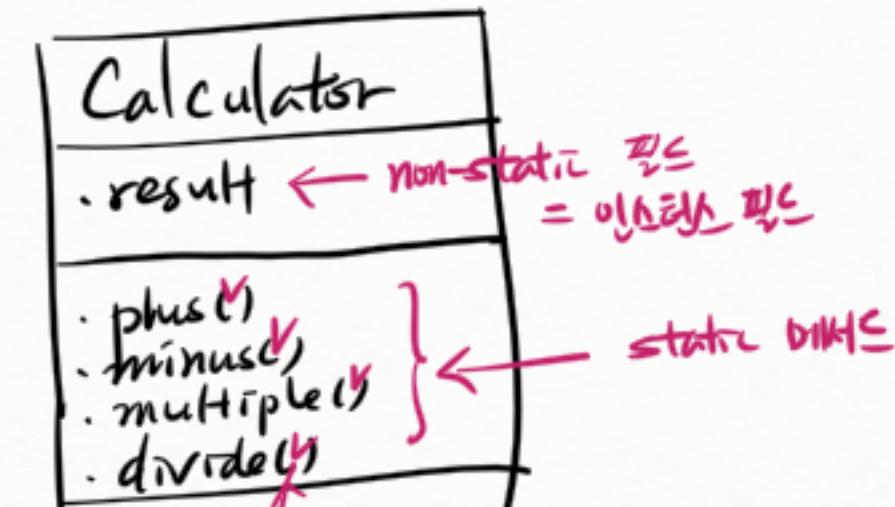
\* 스태틱 필드 → 인스턴스 필드  
 (com.eomcs.oop.p. ex02.Exam02xx)



④  $\Rightarrow$  멤버 변수의 초기화  $\longrightarrow$  ⑤ 인스턴스 변수로 전환



static int result;



```
void plus(Calculator obj, int value) {  
    obj.result += value;  
}
```

Calculator c1 = new Calculator();

c1  
200

result  
\_\_\_\_\_

Calculator c2 = new Calculator();

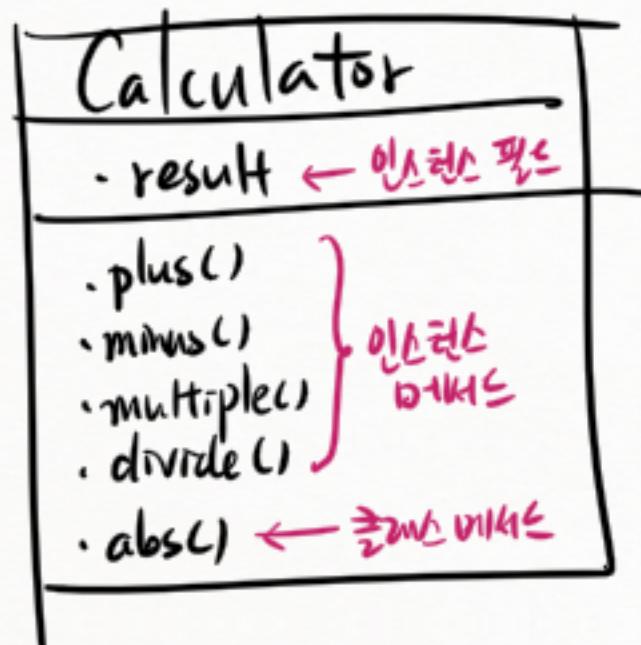
c2  
300

result  
\_\_\_\_\_

Calculator.plus(c1, 2);

Calculator.plus(c2, 3);

⑥ static 멤버드 → 인스턴스 멤버드 정의



```
void plus( int value ) {  
    this.result += value;  
}
```

↑ 인스턴스 멤버드의 Built-in 변수

Calculator c1 = new Calculator();



Calculator c2 = new Calculator();



c1.plus( 2 );

c2.plus( 3 );

① Nested  $\frac{2}{2}m\backslash$  → Package member  $\frac{2}{2}m\backslash$   $\rightarrow$  ②  $\text{util}$   $\frac{2}{2}$

