class

* 클래스 분류의 목적

① 비즈니스 분류 : MemberHandler
Prompt

② 데이터 타입을 정의 : Score, Member

★ 새 데이터 타입 정의

① 클래스 정의

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

"인스턴스 변수 (field)"

인스턴스를 만들 때 생성된 ‖ new 명령을 실행할 때 생성된.

② 레퍼런스 선언

```
Score obj;
```

"reference"

↑ Score의 인스턴스 주소를 저장할 변수

③ 인스턴스 생성

```
obj = new Score();
```

obj → 200

new Score(); → 생성되는 변수들을 Heap 영역에 만듦.

200 name | kor | eng | math | sum | aver

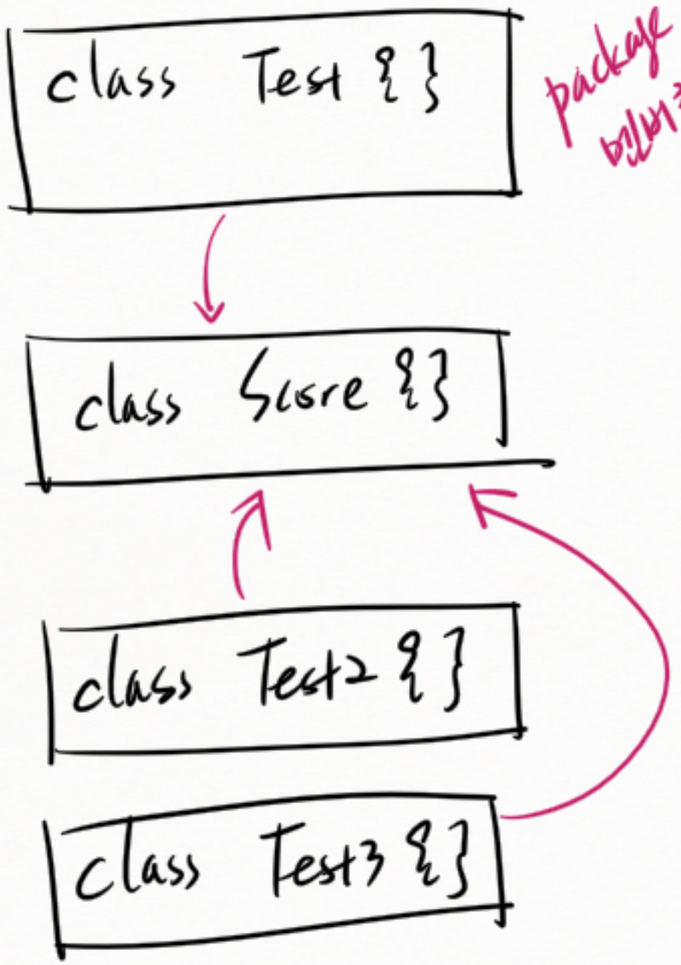
인스턴스

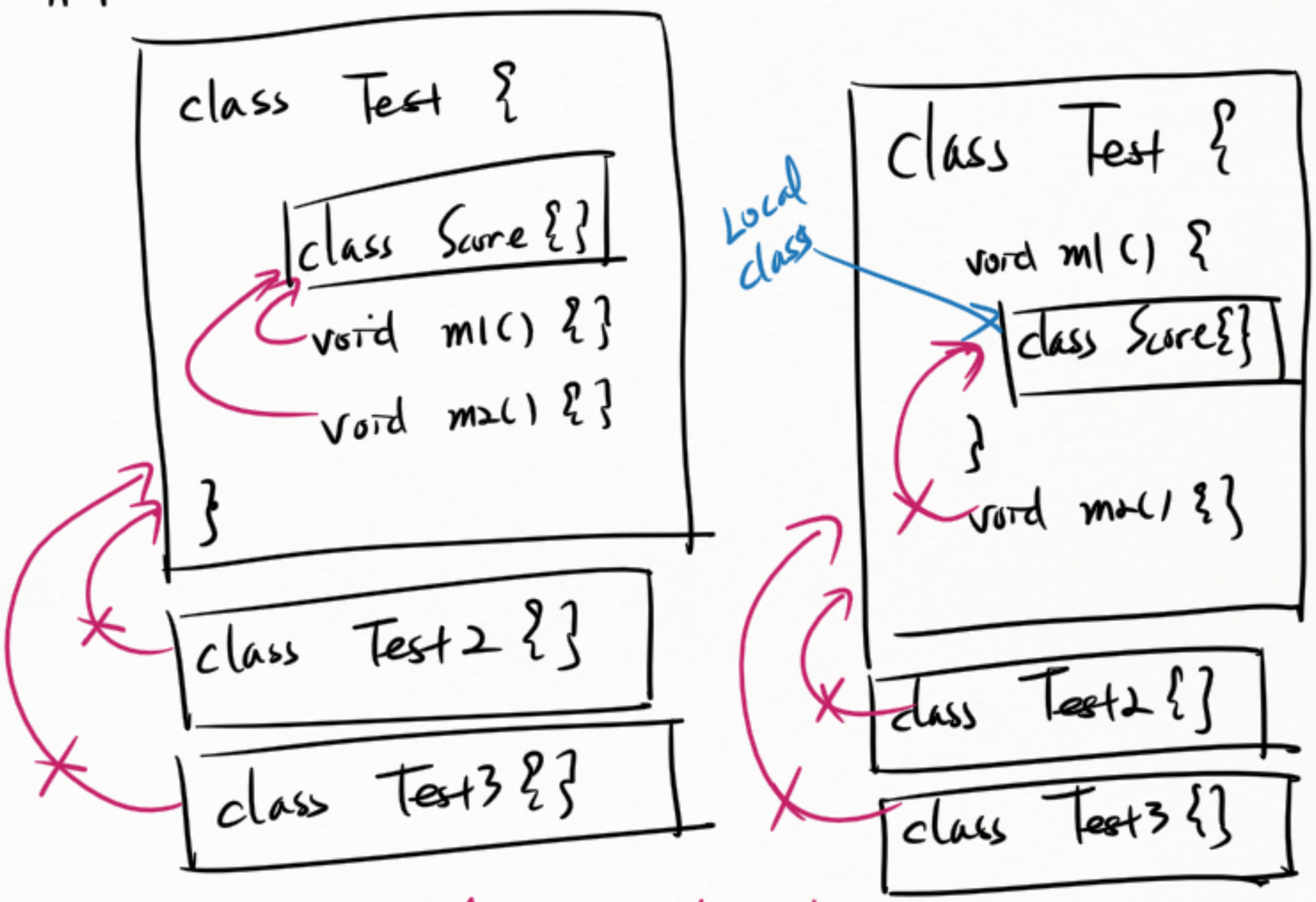
"Score의 인스턴스" instance

④ 인스턴스에 값 저장

```
obj.name = "홍길동";
```

* 클래스 정의의 위치

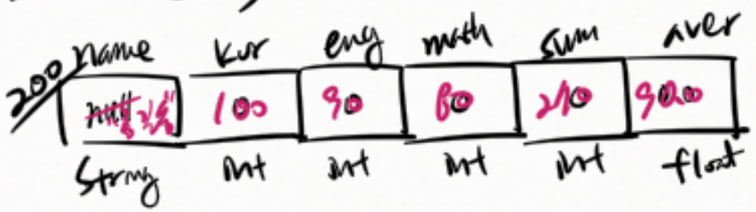




"Nested class"

* 인스턴스 생성, 레퍼런스, call by reference

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




```
s.name = "홍길동";
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 값

```
printScore(Score s) {
    System.out.printf();
}
```

200

* 메서드에서 인스턴스 생성 후 리턴

```
Score s = createScore("홍길동", 100, 100, 100);
printScore(s);
```

s → 200

```
createScore(String name, int kor, int eng, int math) {
    Score s = new Score();
    s.name = name;
    s.kor = kor;
    s.eng = eng;
    s.math = math;
    return s;
}
```

s.sum = ;

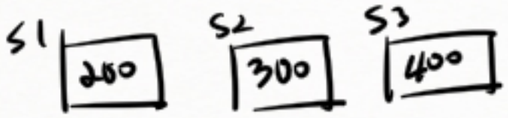
s.aver = ;

s → 200

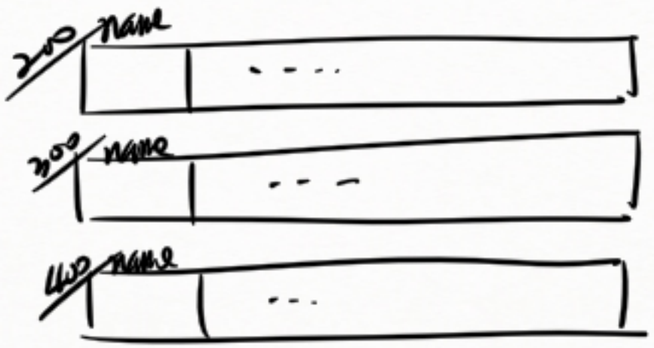
200	name	kor	eng	math	sum	aver
	홍길동	100	100	100	300	100.0

* 레퍼런스 배열 사용 전

```
Score s1, s2, s3;
```




```
s1 = new Score();
s2 = new Score();
s3 = new Score();
```



* 레퍼런스 배열 사용 후

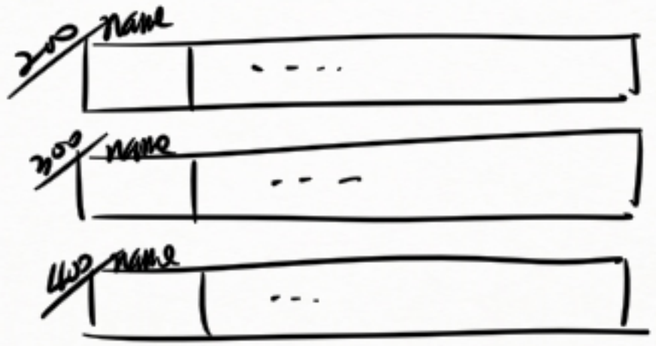
```
Score[] scores = new Score[3];
```

배열레퍼런스 — 레퍼런스들의 배열

scores

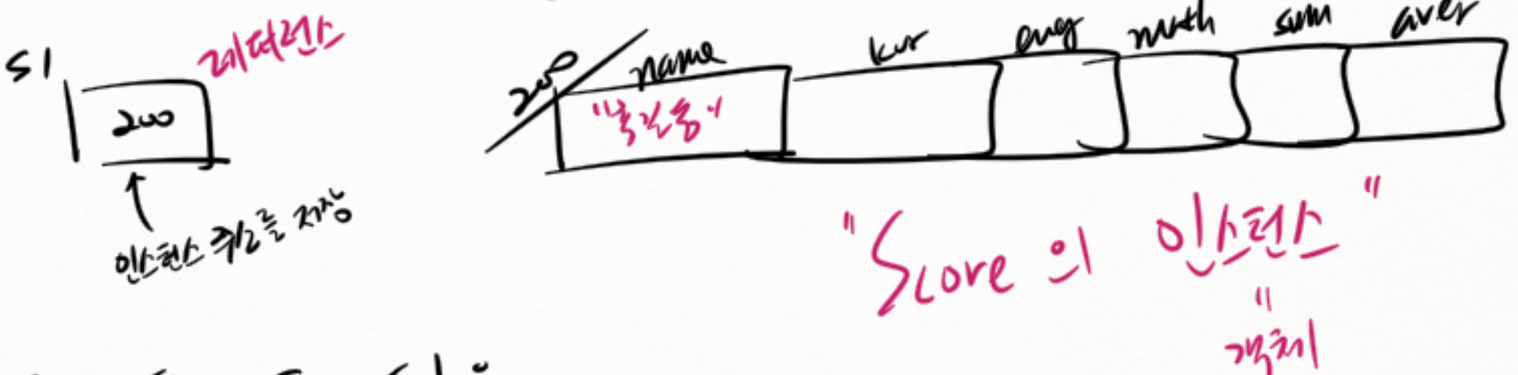
1700


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



*인스턴스 와 레퍼런스

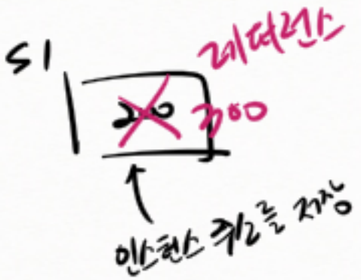
Score s1 = new Score();

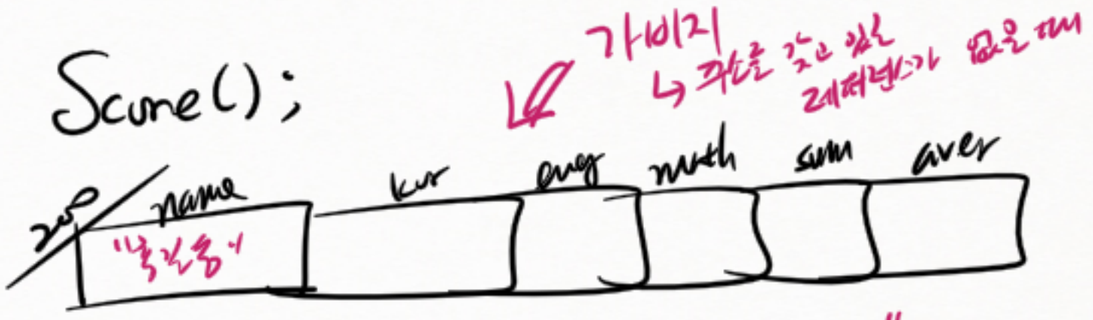


Score s2 = s1;

* 가비지(Garbage)

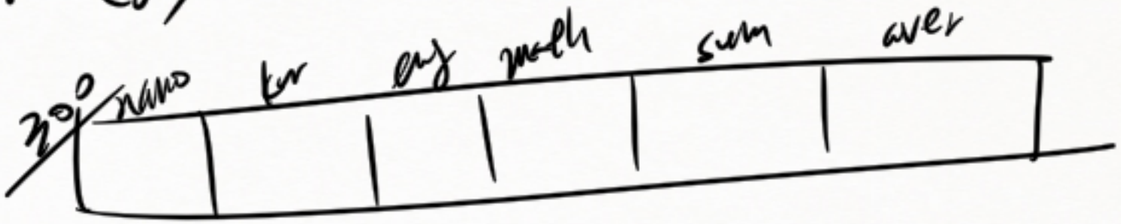
Score s1 = new Score();





"Score의 인스턴스"
"
객체

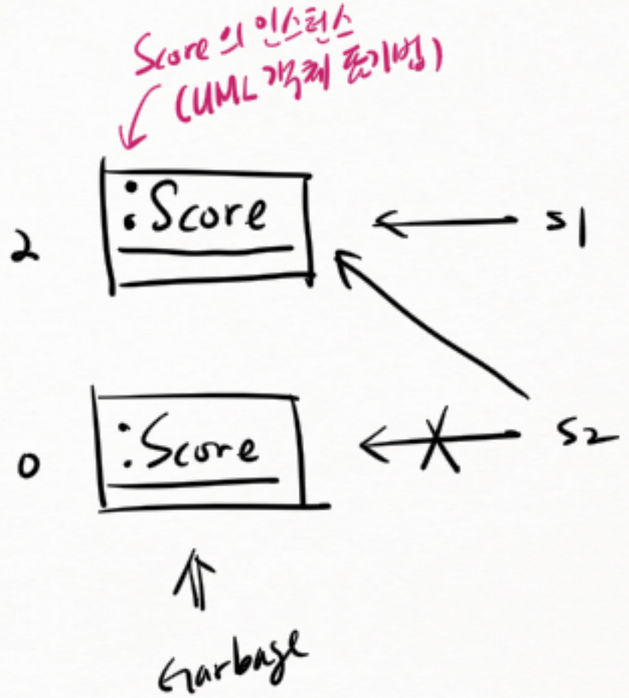
s1 = new Score();



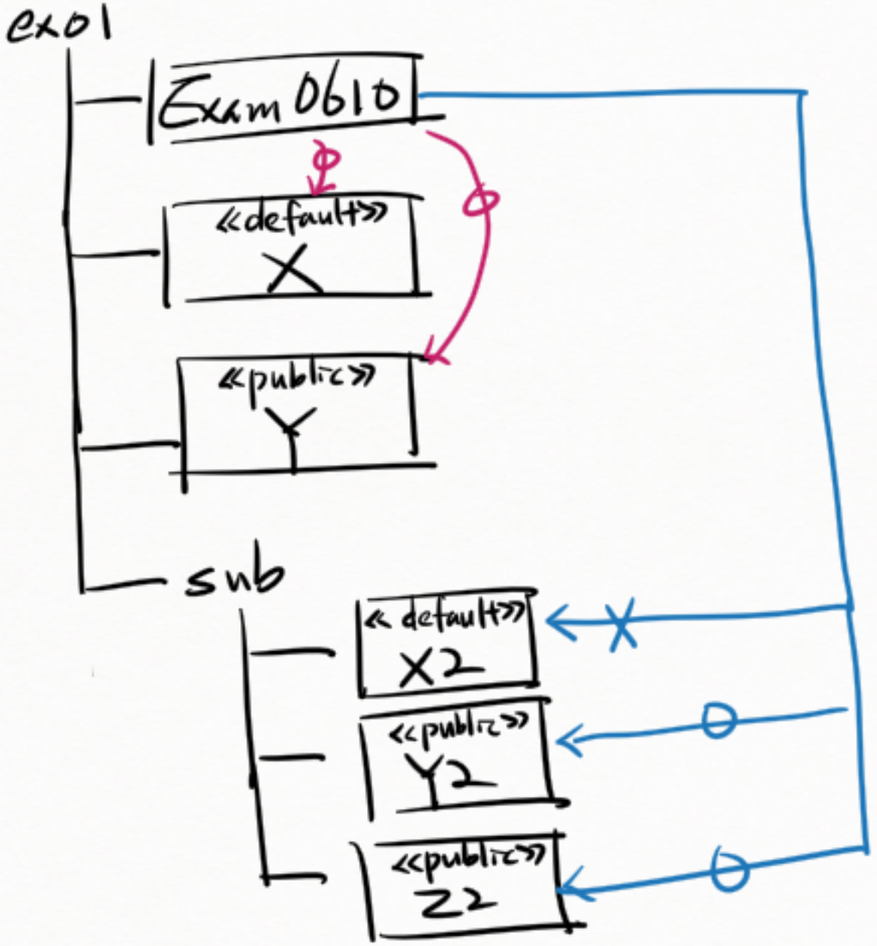
* 인스턴스 레퍼런스 카운트

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

s2 = s1;
```

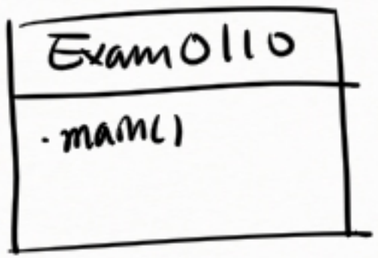


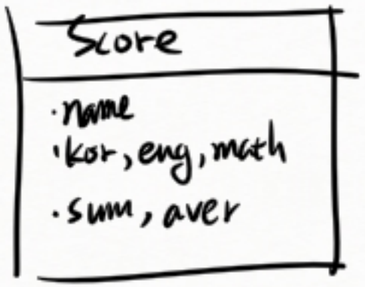
* public 클래스 — default 클래스 (package private class)



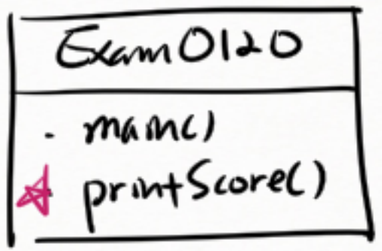
* com.comcs.oop.ex02.Exam01xx

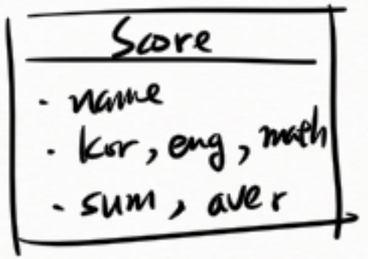
① 날개 변수 사용 → ② class 분법 : 새 데이터 타입 정의 → ③ method 분법 : 중복 코드 추출





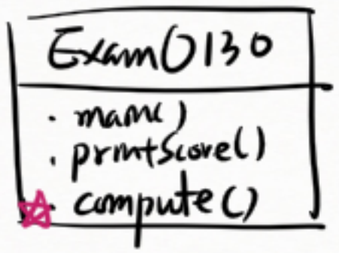
User Defined Data Type



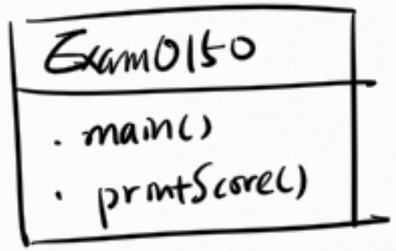


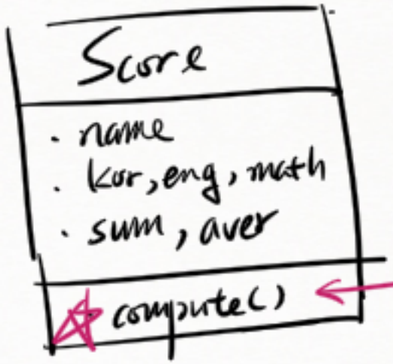
- 레퍼런스 와 인스턴스
  - new
  - Heap 영역
  - garbage
  - garbage collector

④ 리팩토링 : 1기능 → 1메서드 ⑤ 리팩토링 : 메서드 이동 ⑥ 인스턴스에 더 쉽게 접근하는 법 : 인스턴스 메서드 non-static



GRASP 의
Information
Expert





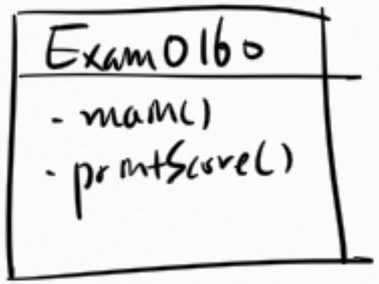
non-static 으로 변경

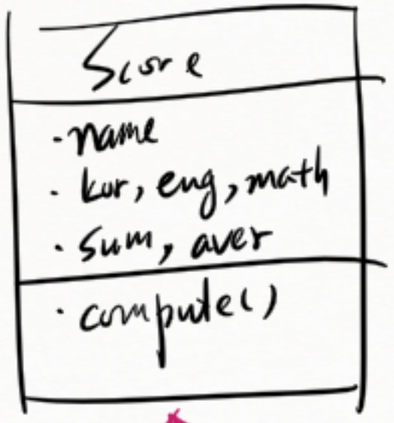
이전 방식
Score. compute (인스턴스주소)

변경 후
인스턴스주소. compute()

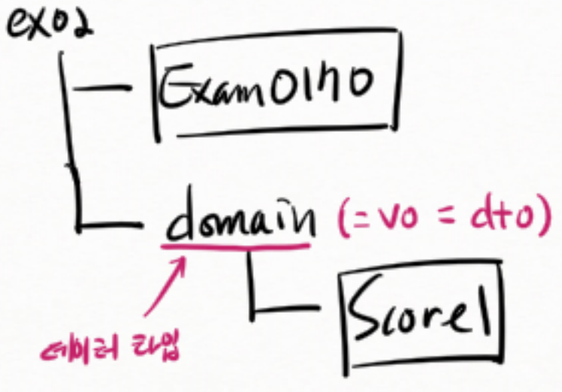
⑦ 패키지 멤버 클래스 → ⑧ 클래스를 패키지로 묶음 → ⑨ 객체 초기화 방법 : 생성자

관리 용이



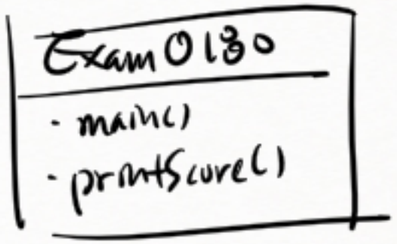


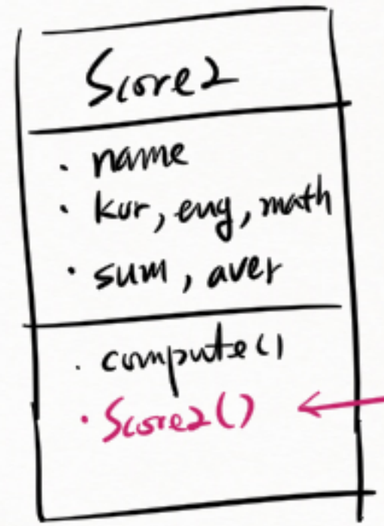
nested class → package member
변경



데이터 타입

접근제어 종류
- public : 완전공개
- protected : 서브클래스, 같은 패키지
- (default) : 같은 패키지
- private : 비공개





생성자 (constructor)

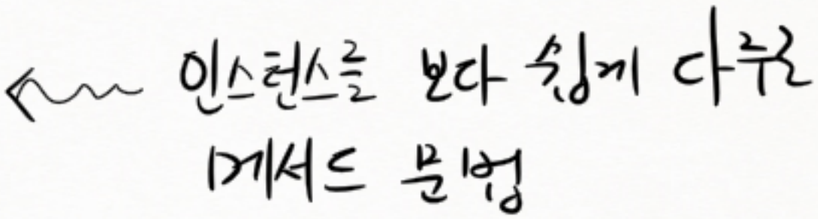
* 스태틱 메서드와 인스턴스 메서드

static 메서드 = 클래스 메서드 ⇒ `Score.compute(s1);`

- `Score` : 메서드가 소속된 클래스
- `s1` : 파라미터 값

non-static 메서드 = 인스턴스 메서드 ⇒ `s1.compute();`

- `s1` : 메서드가 소속된 클래스의 인스턴스 주소



* 인스턴스 메서드와 this

전달

```
s1.compute();
```

메서드를 호출한 cm
왼쪽에서 넘겨준 인스턴스 주소를 받는
built-in 참조 변수
⇓
non-static 메서드에만 존재한다.

* 인스턴스 주소를
this 내장 변수에 저장된다
⇓
인스턴스 주소를 받기 위해
따로 변수를 선언할
필요가 없다.

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

* 생성자

```
class Score {
    Score(파라메터, ...) {
    }
}
```

클래스 이름과 같아야 한다

리턴타입이 없다

* 생성자 사용 전/후

① 사용 전

```
Score s = new Score();
s.name = "홍길동";
s.kor = 100;
s.eng = 100;
s.math = 100;
s.compute();
```

Score 설계도에 따라 인스턴스 생성 ⇓ 생성자 호출

② 사용 후

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

↓ 인스턴스 생성 후 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();
}
```