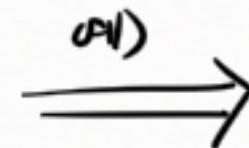


\* 클래스 문법

- 
- ```
graph LR; A[클래스] --> B[① 데이터 타입 정의  
~~~~~  
개별화]; A --> C[② 메서드 분류]
```
- ① 데이터 타입 정의 (User-defined Data Type)  
~~~~~  
개별화
- ② 메서드 분류

\* ↗ 클래스 문법 - 데이터 타입 정의

class 데이터타입 {  
 변수선언  
 :  
}

예) 

class Contact {  
 String name;  
 String email;  
 String tel;  
 String company;  
}

← 메모리  
설정

||  
~~new~~ 명령을 실행하면  
클래스에 정의된 대로  
변수가 준비된다.

\* 클래스를 이용하여 새 데이터 타입의 변수 만들기

Contact c = new Contact();

Contact 클래스의 주소를 저장하는  
리퍼런스(reference)

리터리터링 = 클래스명

클래스 선언에 따라 메모리 할당 => Heap 영역

인스턴스(instance) = 개체(object)

c | 200

200 | name email tel company

## \* 레퍼런스 배열

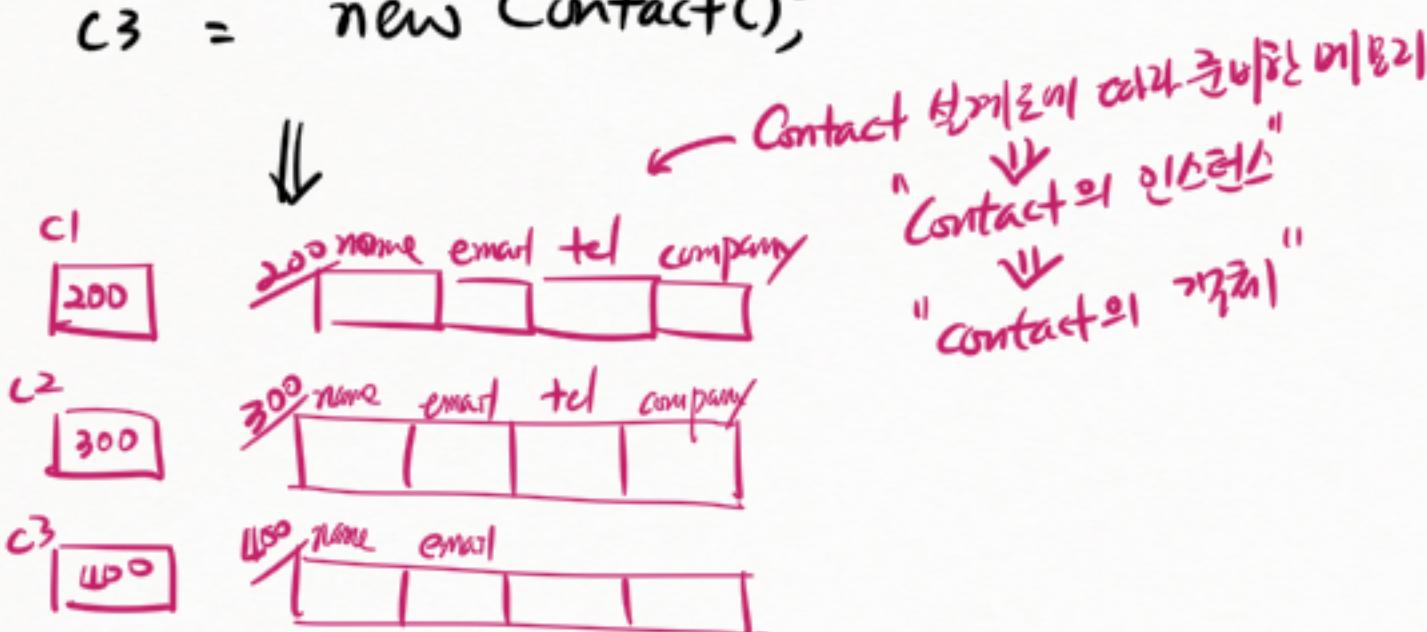
## ① 배포 사용 전

① 초기화 사용 선  
Contact c1, c2, c3;      ← 인스턴스의 주소를 저장하는 변수  
  "리퍼런스"  
  "포인터(pointer)"

```
c1 = new Contact();
```

```
c2 = new Contact();
```

```
c3 = new Contact();
```

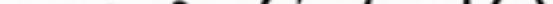


## ② 배포 사용 흐름

Contact[] arr = new Contact[3];  
레퍼런스를 3개 만드는 명령



```
arr[0] = new Contact();
```



A hand-drawn diagram of a Contact object. It consists of a horizontal line with five boxes. The first box is crossed out with a large red X. To its right are four boxes labeled 'name', 'email', 'tel', and 'company' respectively.



```
arr[1] = new Contact();
```



```
arr[2] = new Contact();
```

~~5~~ | name smart tel company

## \* 레퍼런스와 인스턴스 변수

Contact c = new Contact()



① 인스턴스 변수가 있는 저장

c.name = "홍길동";

인스턴스 주소를  
알고 있는  
레퍼런스

↑  
인스턴스  
변수

c.email = "hong@";

c.tel = "1111";

c.company = "비즈";

② 인스턴스 변경

c = new Contact()



c.name = "이재정";

c.email = "leem@";

c.tel = "2222";

c.company = "캐논";

기존 인스턴스의  
주소를 알고 있어  
레퍼런스가 한 개로 고정된  
“garbage”가 된다.

### Method Area

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

### JVM Stack

```
Score s;
```

s 200

↑  
Score의 레퍼런스

### Heap

```
new Score()
```



200 name kor eng math sum aver

↑  
Score의 인스턴스  
(기록체)

\* com.eomcs.oop.exam.Exam0114

Method Area

```
class Exam0114 {  
    public static void main(String[] args) {  
        System.out.println("Hello Java");  
    }  
}
```

```
class Score {  
    public static void main(String[] args) {  
        System.out.println("Hello Java");  
    }  
}
```

JVM Stack

```
main()  
args [ ] s [ ]
```

Heap

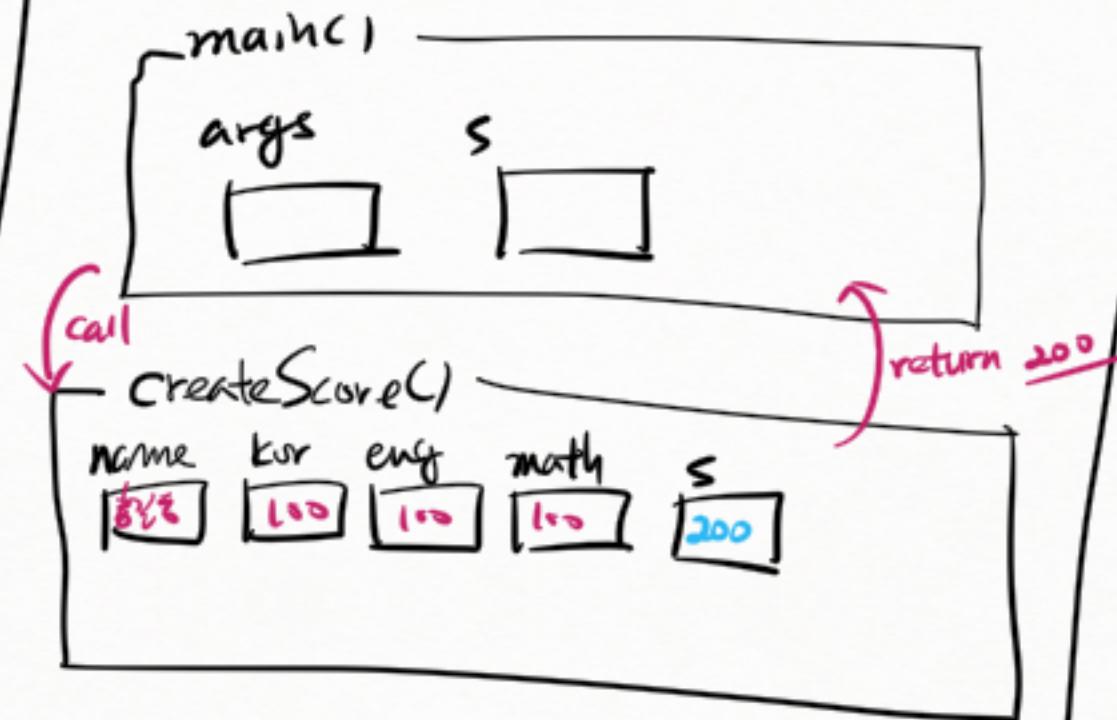
\* com.eomcs.oop.exam. Exam0114

Method Area

```
class Exam0114 {
    public static void main(String[] args) {
        Score s = new Score();
        System.out.println("Score is " + s);
    }
}
```

```
class Score {
    private String name;
    private int kur;
    private int eng;
    private int math;
    private int sum;
    private double aver;
}
```

JVM Stack



Heap

name	kur	eng	math	sum	aver
200	100	100	100	300	100

name	kur	eng	math	sum	aver
323	100	100	100	300	100

\* com.eomcs.oop.exam. Exam0114

Method Area

```
class Exam0114 {  
    public static void main(String[] args) {  
        System.out.println("Hello Java");  
    }  
}
```

```
class Score {  
    public static void main(String[] args) {  
        System.out.println("Hello Java");  
    }  
}
```

JVM Stack



Heap

name	kor	eng	math	sum	aver
200	32.5	100	100	100	300

\* com.eomcs.oop.exam. Exam0114

Method Area

```
class Exam0114 {  
    public static void main(String[] args) {  
        Score s = new Score();  
        s.printScore();  
    }  
}
```

```
class Score {  
    public void printScore() {  
        System.out.println("Hello Score");  
    }  
}
```

JVM Stack



Heap

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

200 name kor eng math sum aver

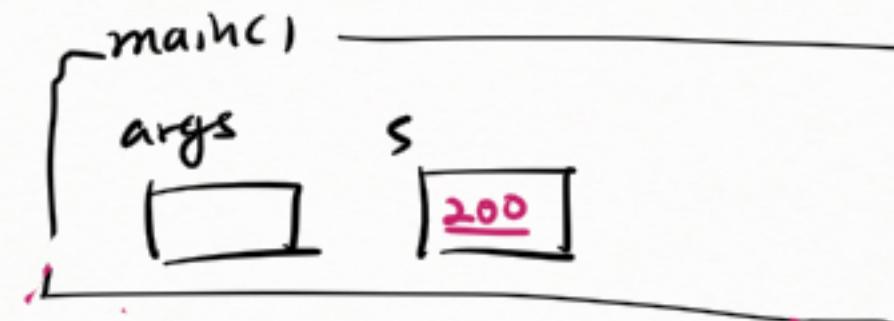
\* com.eomcs.oop.exo1. Exam0114

Method Area

```
class Exam0114 {  
    public static void main(String[] args) {  
        System.out.println("Hello Java");  
    }  
}
```

```
class Score {  
    public static void main(String[] args) {  
        System.out.println("Hello Java");  
    }  
}
```

JVM Stack



Heap

name	kor	eng	math	sum	aver
200	72	100	100	300	100

\* com.eomcs.001.ex01 . Exam0210

Score  $s_1, s_2, s_3$ :

$s_1$  | 200

$s_2$  | 300

$s_3$  | 1100

200	name	kur	...

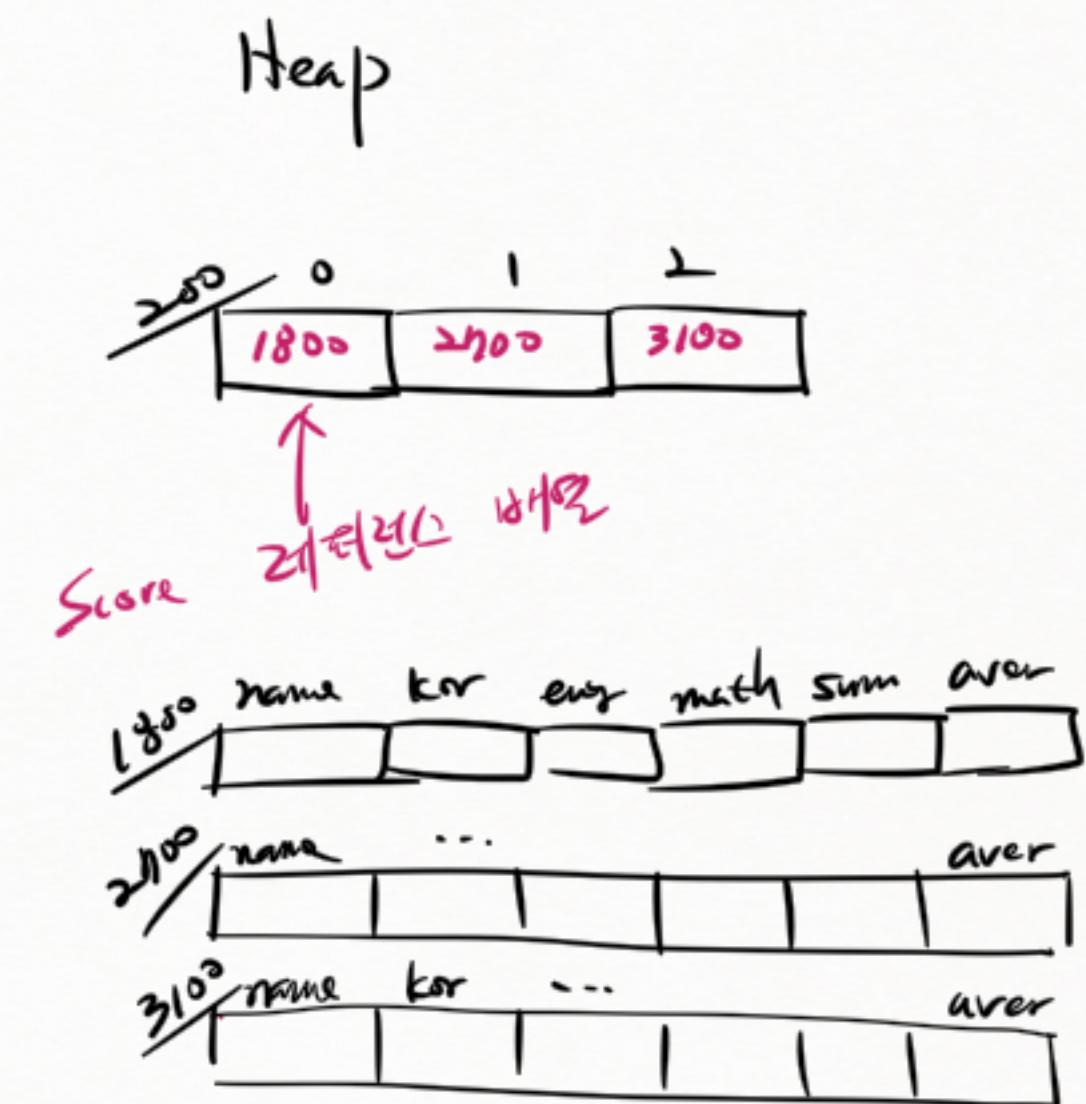
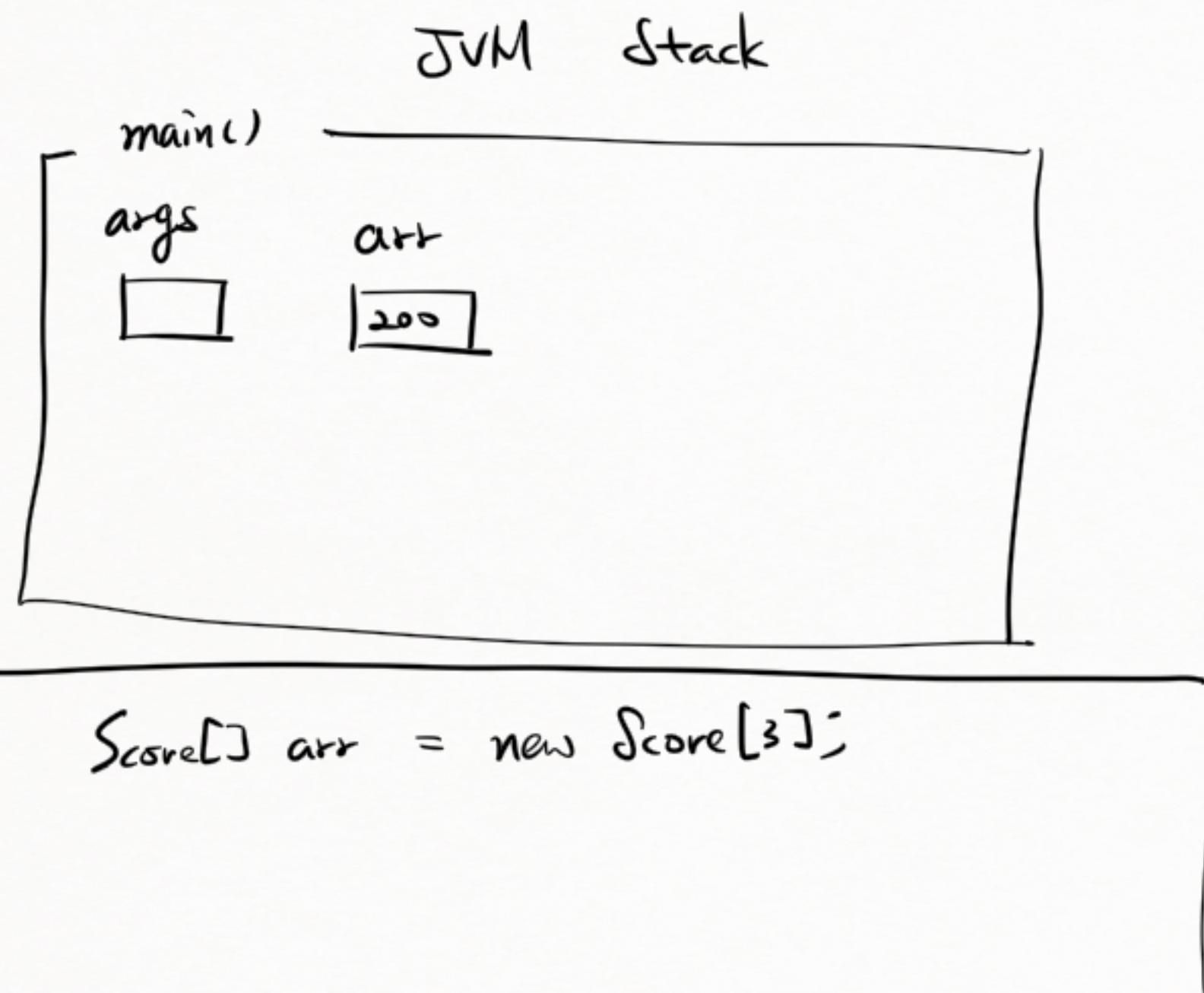
  

300	name	kur	

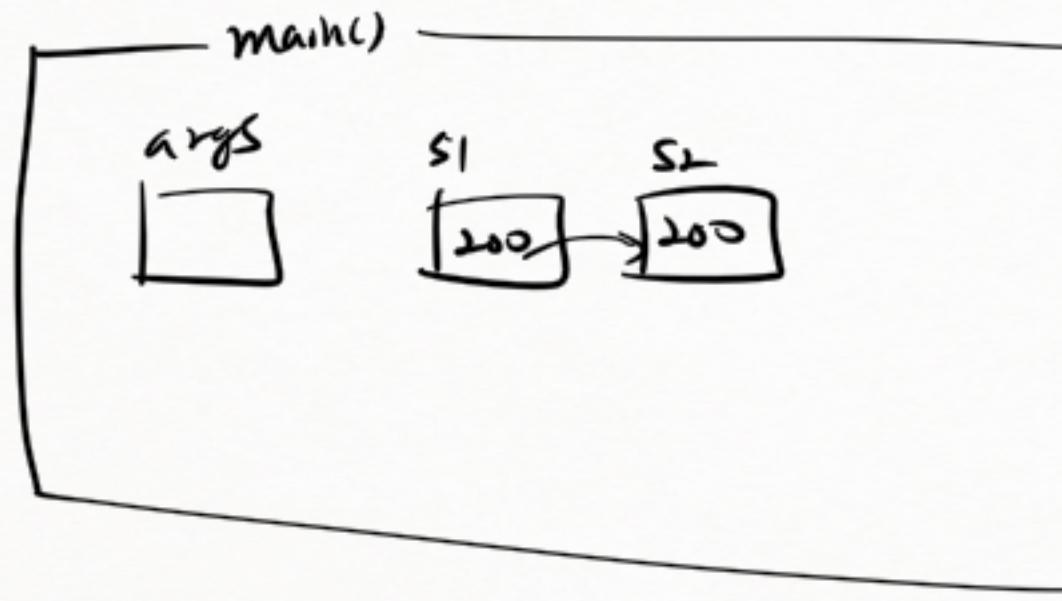
1100	name	kur	

\* com.eomcs.oop.ex01.Exam0220

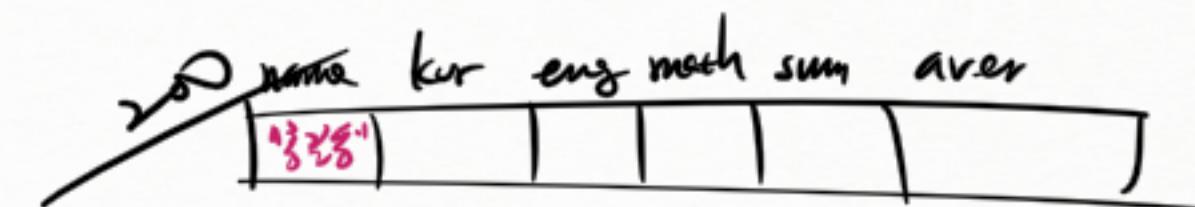


\* com.eomcs.007. ex01. Exam0310

## JVM Stack

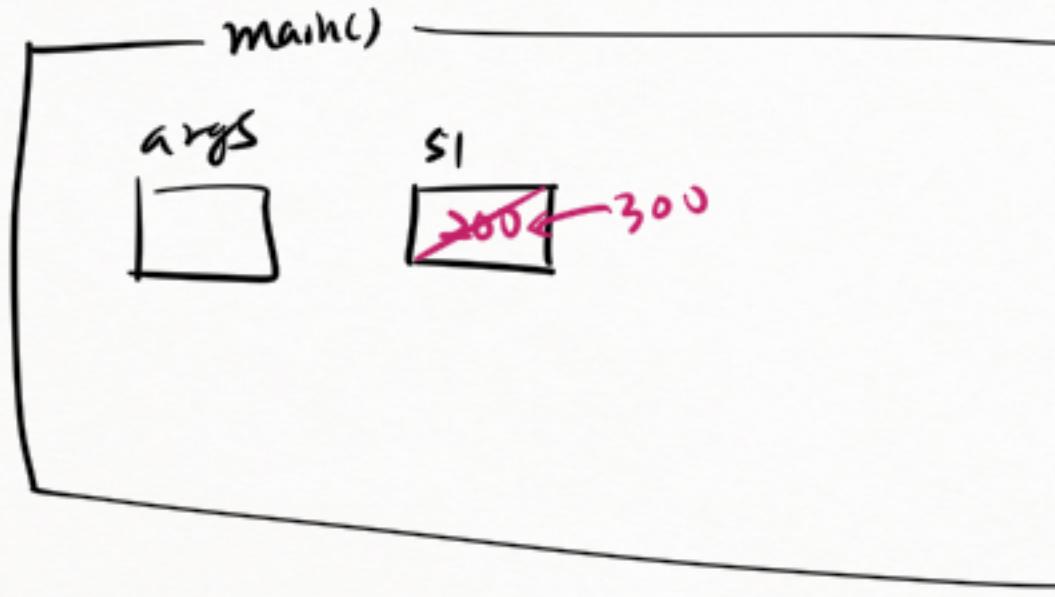


Heap



\* com.eomcs.06p. ex01. Exam0320

JVM Stack



Score s1;

s1 = new Score();

s1 = new Score();

Heap

name	kor	eng	math	sum	aver
null	0	0	0	0	0.0

↑  
인스턴스를 생성하면 각 필드 기본값이 설정된다.

- 객체변수 = null
- 정수변수 = 0
- 부동소수점 변수 = 0.0
- 냄비변수 = false

name	kor	eng	math	sum	aver
300	0	0	0	0	0.0

이 인스턴스의 주소를 갖고 있는  
리터럴스가 단 한개도 없어  
내용은 쓰일 수 없다  
(적어도 한 개 써야함)  
"garbage" 라  
부른다.

# \* com.eomcs.opp.ex01.Exam0330

## JVM Stack



Score s1 = new Score();

Score s2 = new Score();

s2 = s1;

인스턴스 주소는  
200으로 초기화된다.



## Heap

name	kor	eng	math	sum	aver
null	0	0	0	0	0.0

인스턴스를 생성하면 각 필드 기본값이 설정된다.

- 리퍼런스 변수 = null

- 정수 변수 = 0

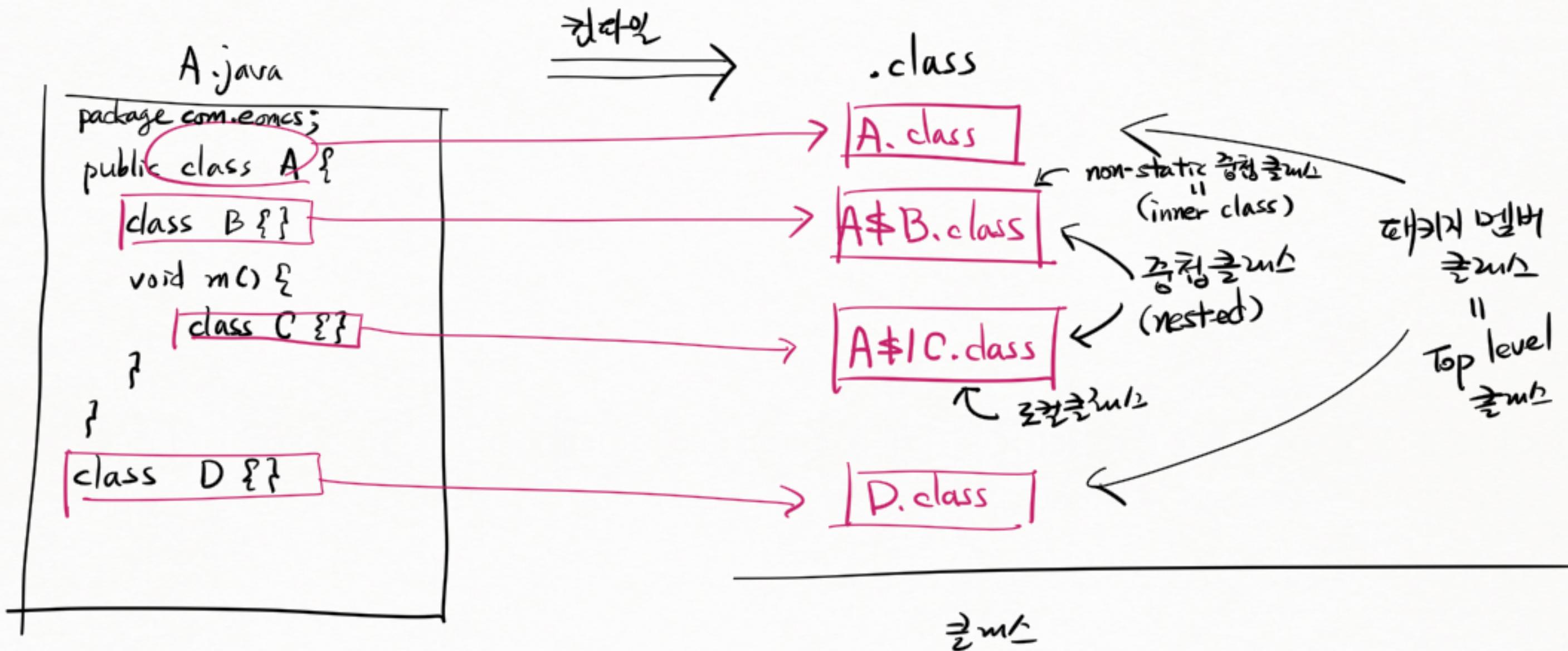
- 부동소수점 변수 = 0.0

- 끝나 변수 = false

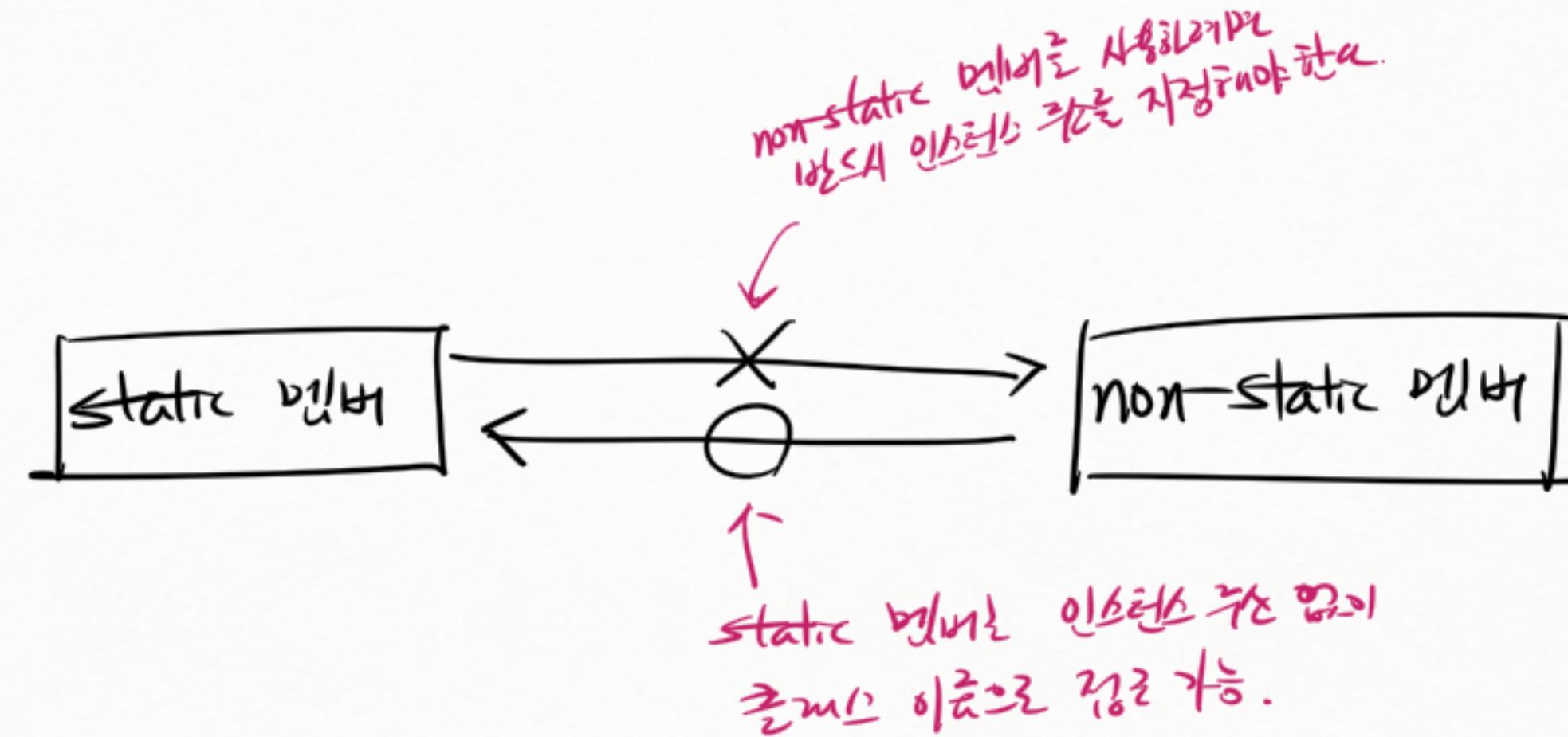
name	kor	eng	math	sum	aver
null	0	0	0	0	0.0

리퍼런스 카운트 개수가 0 이면  
"가arbage(Garbage)" 가 된다.

\* 클래스 복수와 .class 파일



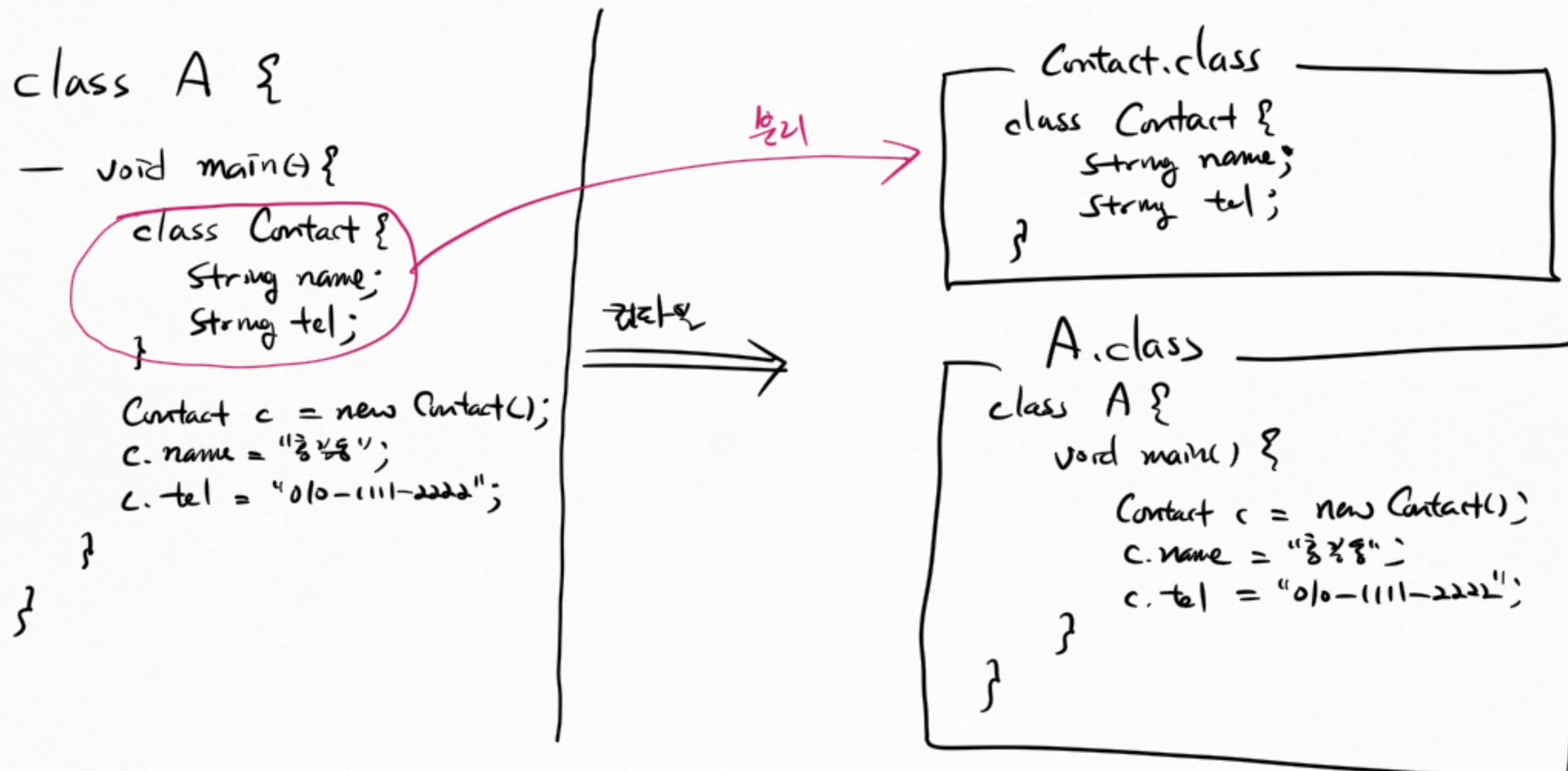
\* static 멤버와 non-static 멤버 간의 접근 규칙



\* 대기지 멤버 구조 : (default) vs public



\* 디자인 원칙과 연결성을 정의



\* 클래스 문법 - 사용자 정의 데이터 타입

① 메모리 유형설정

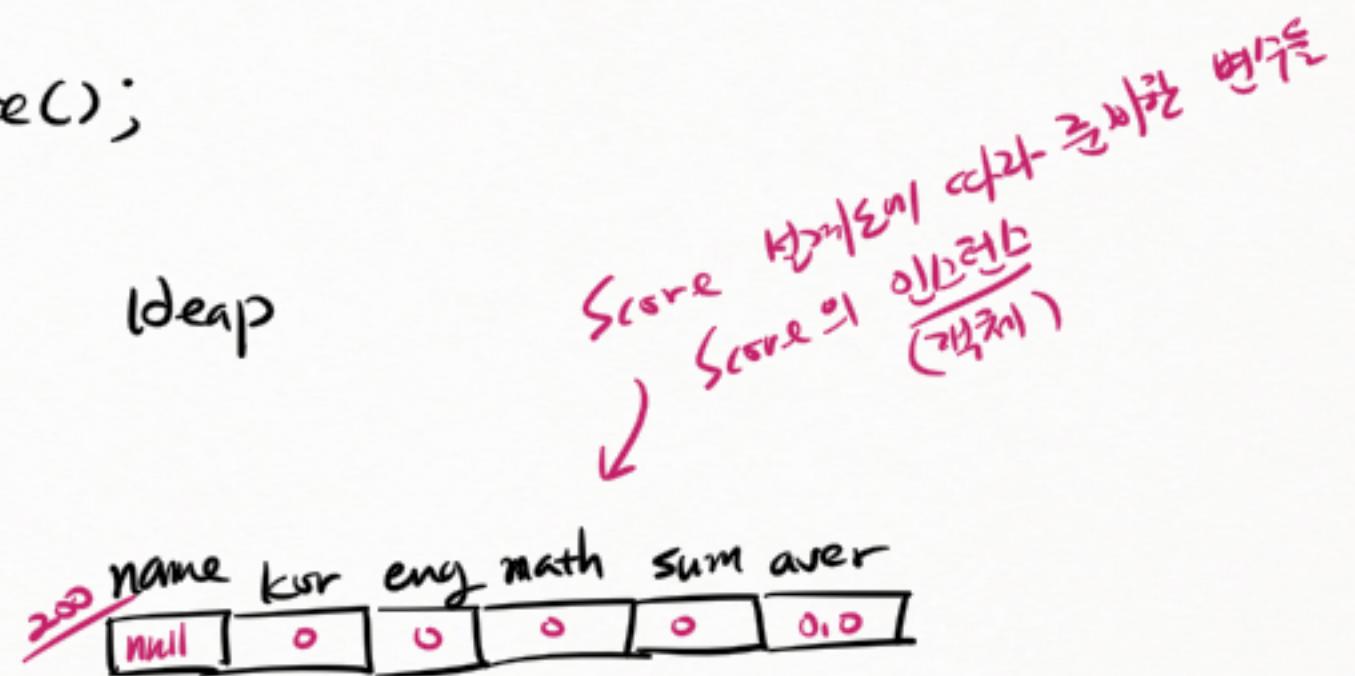
class Score {

```
String name;  
int kor;  
int eng;  
int math;  
int sum;  
float aver;
```

}



Score s = new Score();



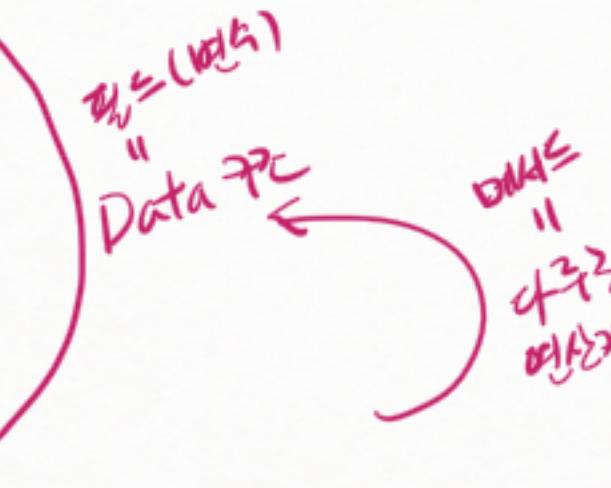
\* 클래스 문법 - 사용자 정의 데이터 타입

② 데이터 구조를 설계하고 그 데이터를 다루는 인산자를 정의  
스라워 메서드

class Score {

```
String name;  
int kor;  
int eng;  
int math;  
int sum;  
float aver;
```

```
static void calculate(Score score){  
    score.sum = score.kor + score.eng + score.math;  
    score.aver = score.sum / 3f;  
}
```



```
int a;  
a = -100;  ⇔  
a++;  
↑ ↑  
데이터 인산자
```

Score s = new Score();

```
s.name = "홍길동";  
s.kor = 100;  
s.eng = 90;  
s.math = 80;
```

Score.calculate(s);

설계문장 → 인산자(Operator) 대입문자

설계문장 → 메서드(method) = 함수(function)

설계문장 → 메시지(message)

\* 클래스 문법 - 사용자 정의 데이터 타입

③ 데이터 구조를 설계하고 그 데이터를 다루는 인수를 정의

non-static 메서드  
(인스턴스)

class Score {

```
String name;
int kor;
int eng;
int math;
int sum;
float aver;
```

필드(변수)  
Data PC  
인수  
여기서  
여기서

~~static~~ void calculate(~~this~~) {

```
this.sum = this.kor + this.eng + this.math;
this.aver = this.sum / 3f;
```

}

}

int a;
a = -100;  $\leftrightarrow$ 
  
a++;
  
피연수자 연산자

Score s = new Score();

s.name = "홍길동";
s.kor = 100;
s.eng = 90;
s.math = 80;

~~Score.calculate(s);~~

기존의  
연산자를  
사용하는 방법은  
더 비슷하다

피연수자  
↓  
s.calculate();  
연산자  
—  
메서드의 내장 변수 this가 저장됨.

인스턴스 주소를 메서드에 전달

## \* 클래스 정의

데이터를 저장할  
메모리 구조 설계 ← 인스턴스 필드 선언

+  
새 데이터 유형을 다른  
연산자 정의 ← 메서드 정의

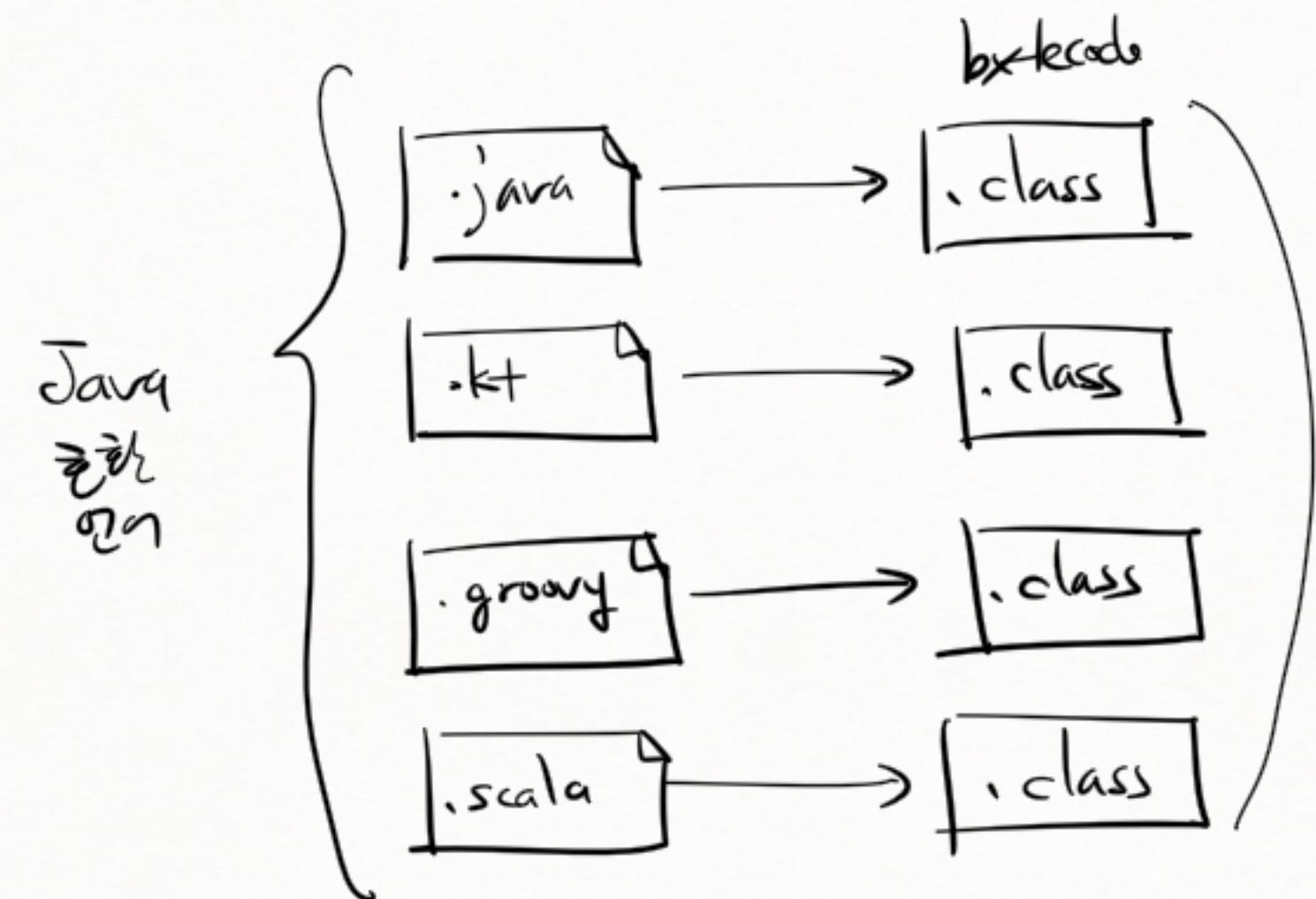
## \* modifier

final  
static  
private  
⋮

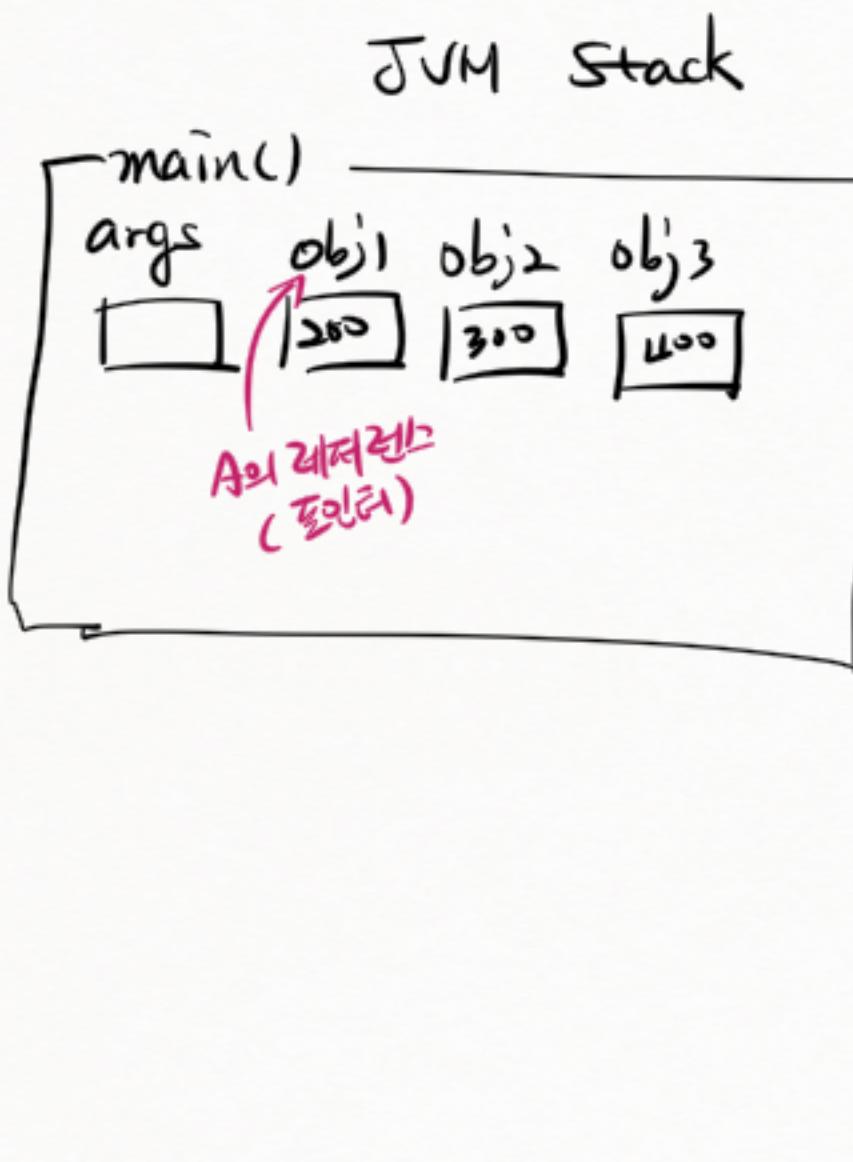
int  $i = 100;$

이들이 무엇을 추가하느냐에 따라-  
위에 선언한 변수(또는 메서드, 클래스)의  
성질(특성)을 바꾼다  
||  
변경을 가하는 명령 (modifier)  
변경자 | 흡정자 | 제한자-

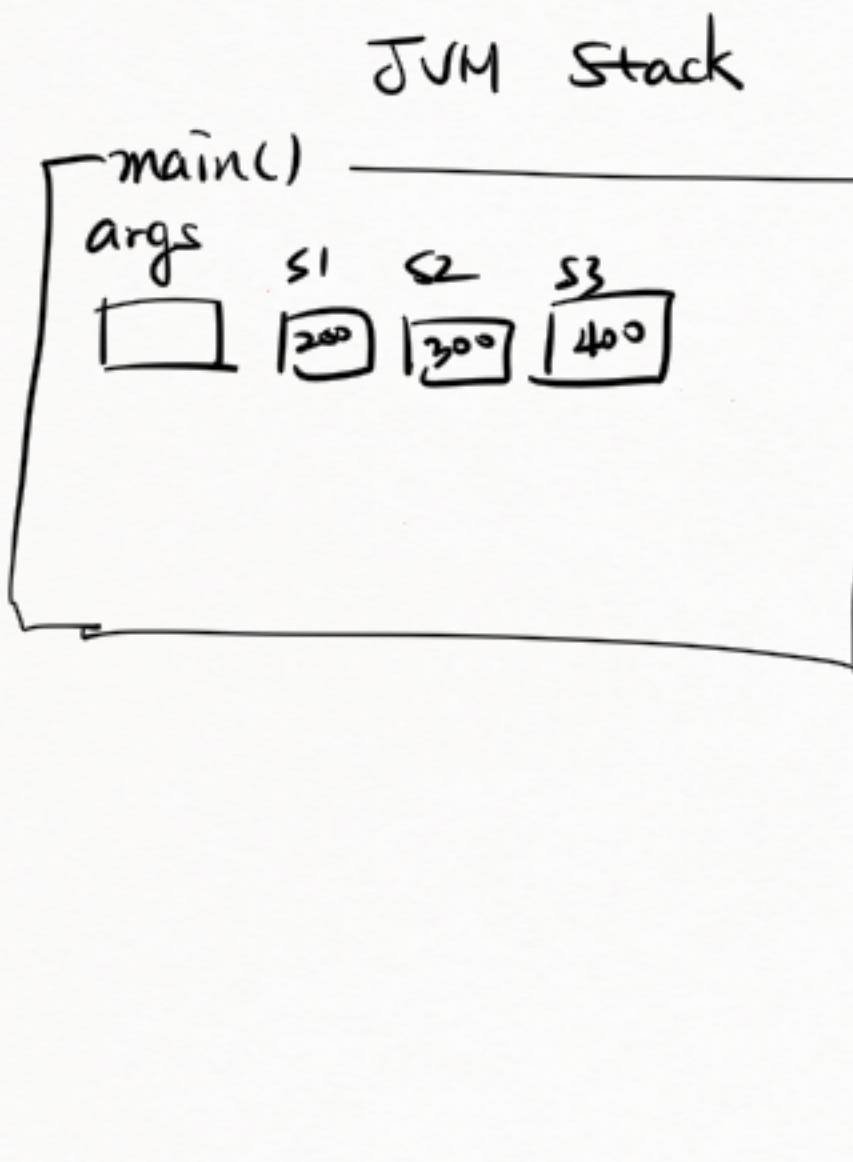
## \* 자바 향한 언어



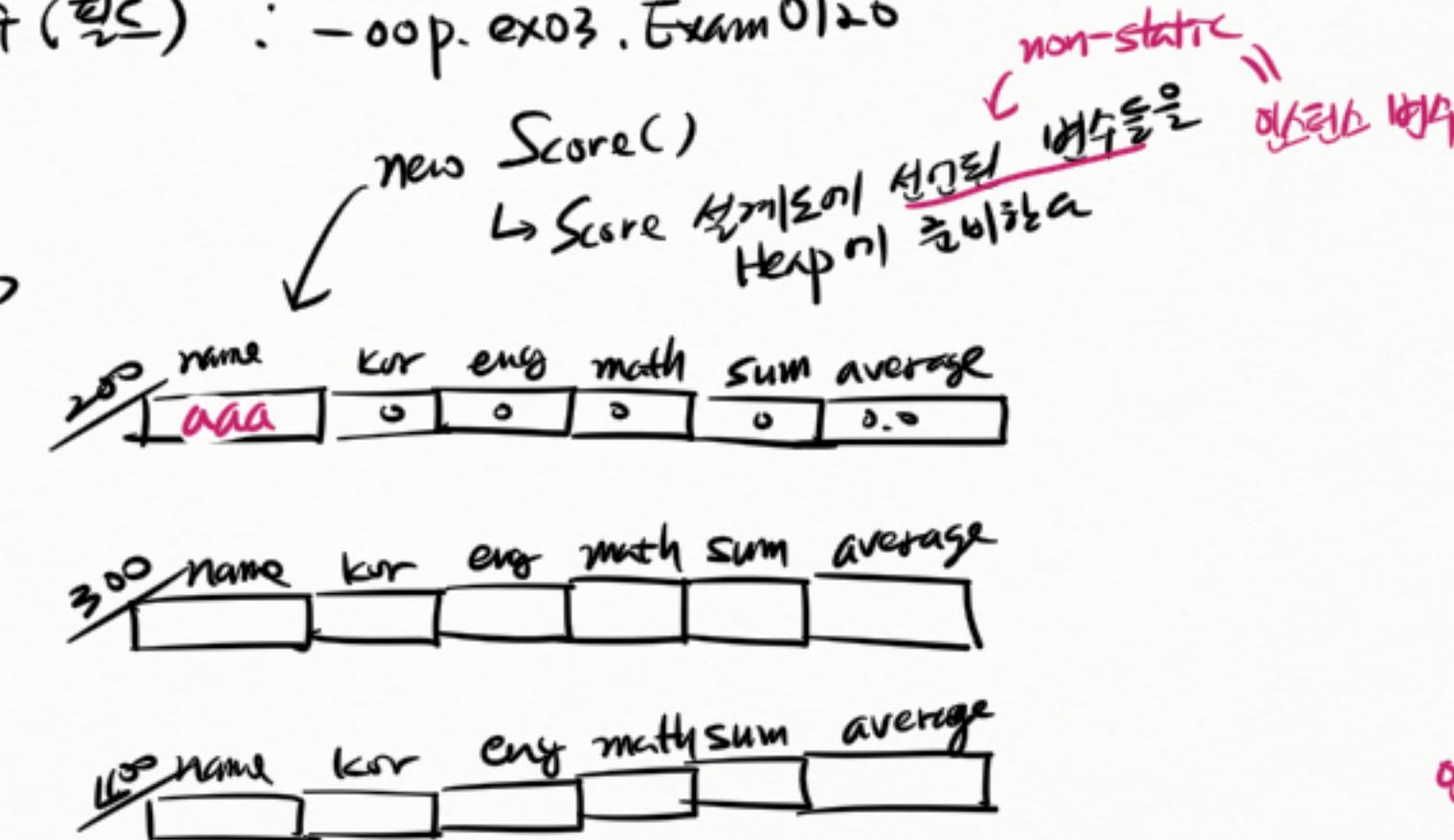
\* 인스턴스 변수(필드) : - oop. ex03, Exam 0110  
non-static



\* 인스턴스 변수(필드) : - oop. ex03, Exam 0120  
non-static



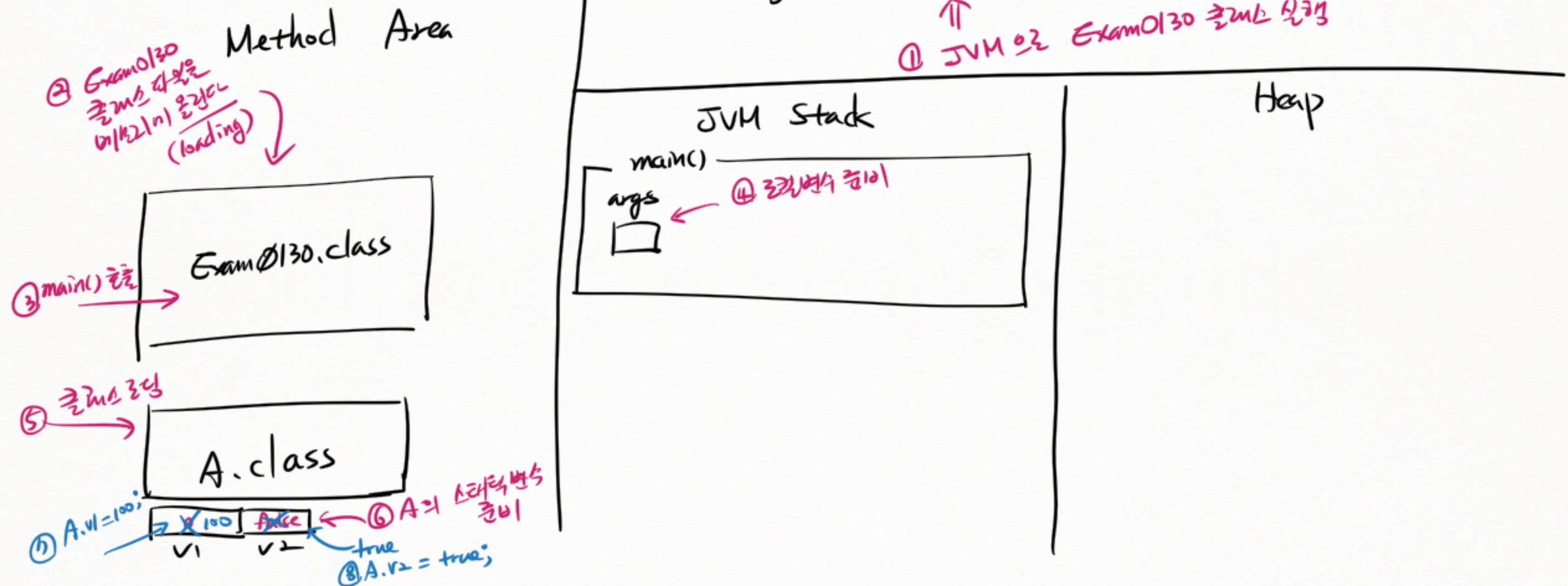
Heap



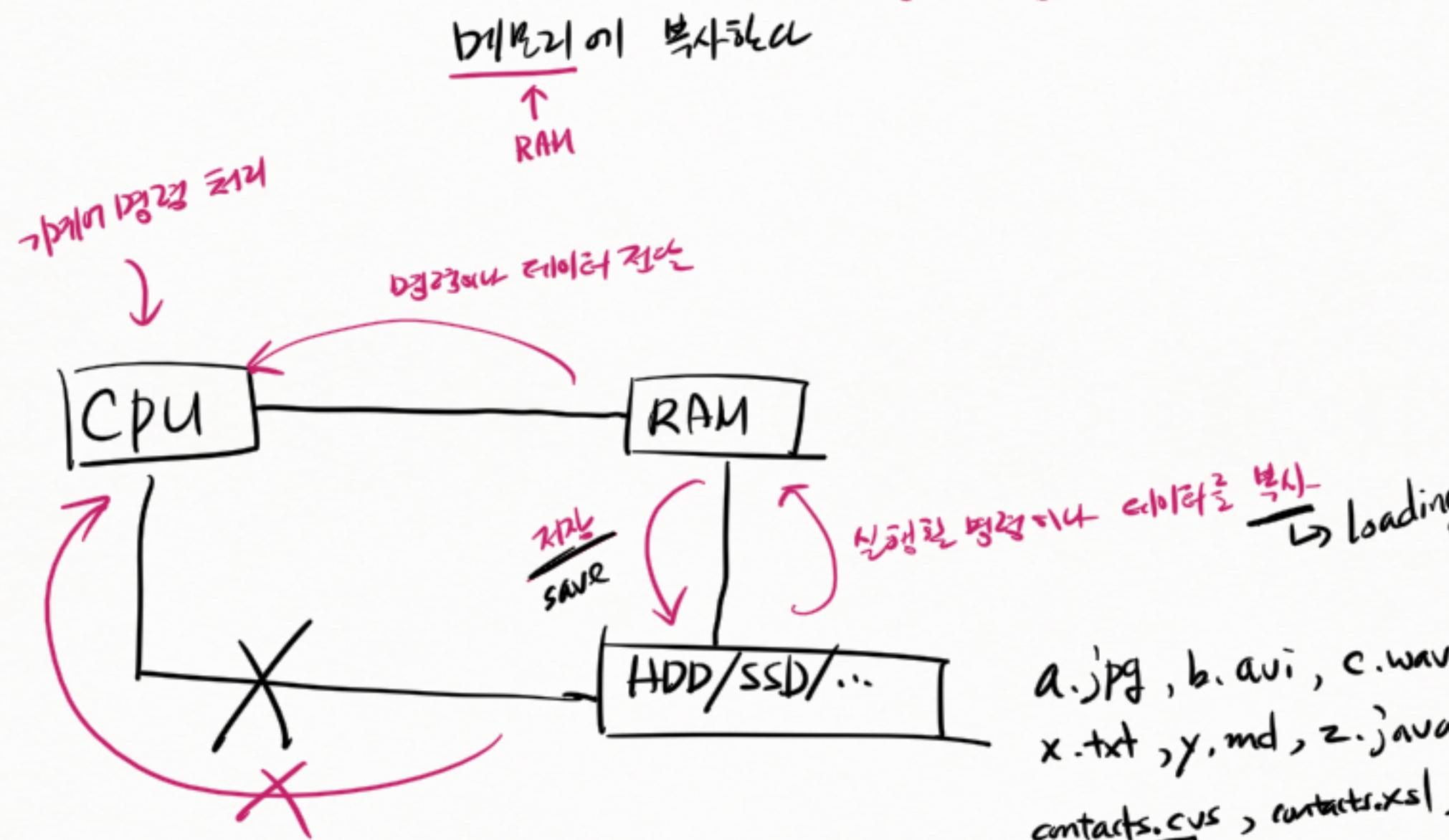
s1.name = "aaa"  
 nm  
 200  
 ↑  
 인스턴스 주소

\* 클래스 변수(필드) : - oop\_ex03\_Exam0130  
(static)

# java -cp bin/main com.eomcs.oop.ex03.Exam0130 ↴  
① JVM 의 Exam0130 클래스 실행



\* 클러스터링 (clustering)

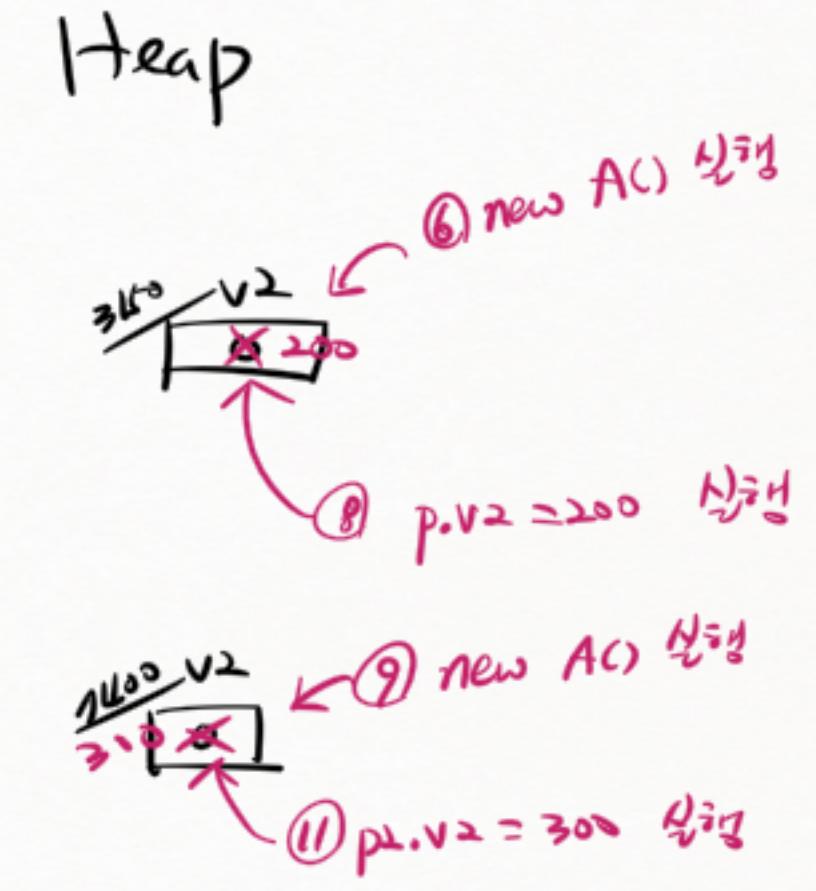
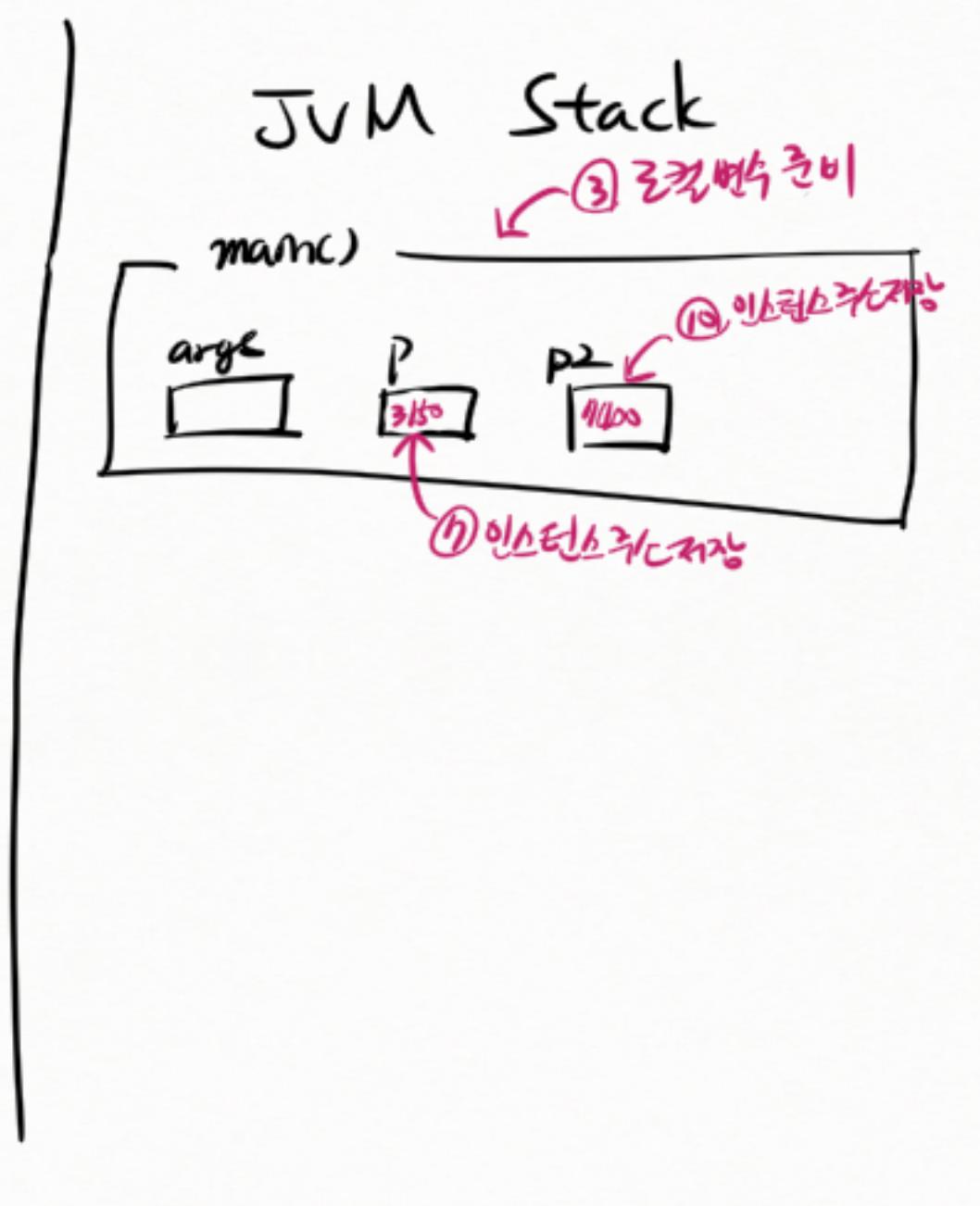
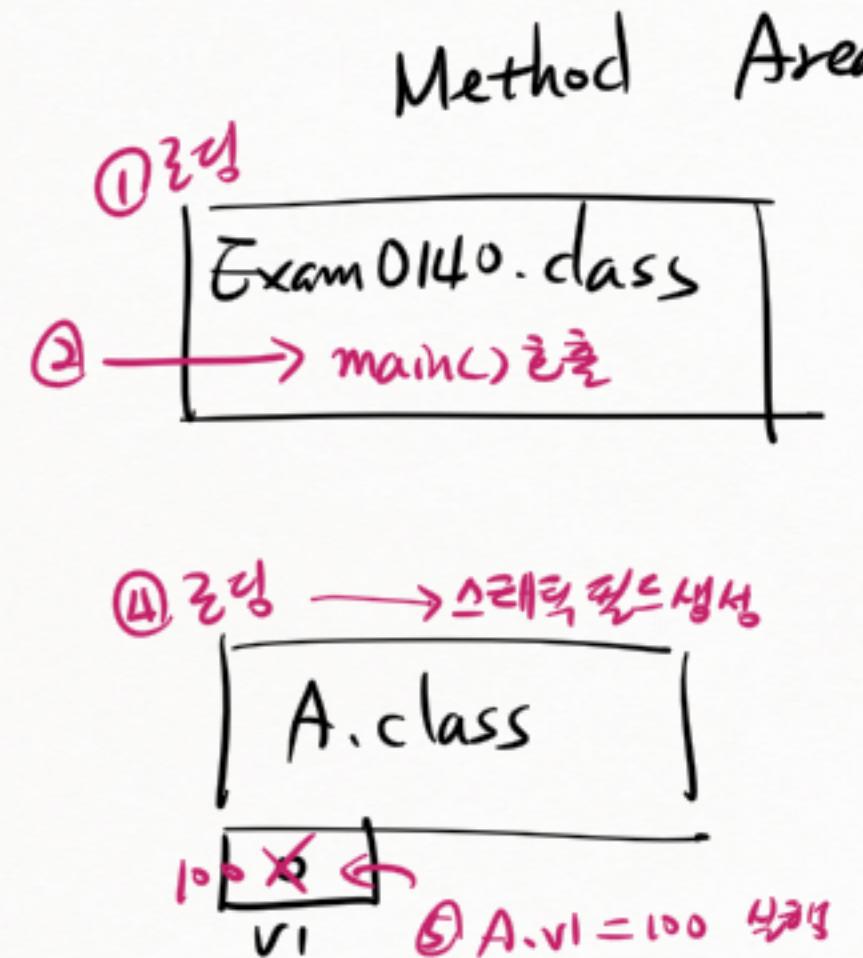


$y_2z = 14$   $\leftarrow$   $\text{color}^2 \frac{2}{2}$  부서

↳ loading

- a.jpg, b.avi, c.wav, d.mp3 ...
- x.txt, y.md, z.java
- contacts.csv, contacts.xls, ...
- .class

\* 글래스 변수와 인스턴스 변수 : oop. ex 3. Exam0140



\* 주소 변수와 인스턴스 변수 : oop. ex 3. Exam0150

