

* 클래스 문법

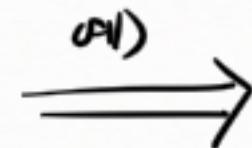
클래스

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

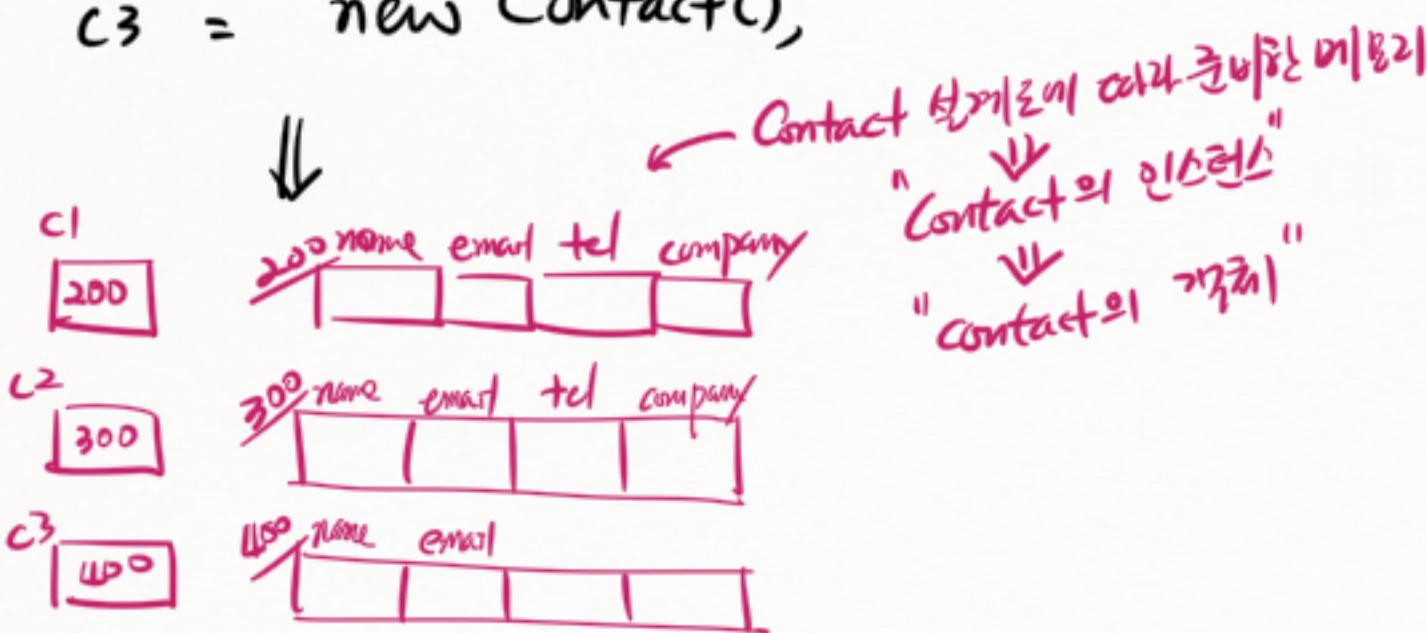
* 레퍼런스 배열

① 배포 사용 전

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

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

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



② 배열 사용 흐름

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



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

~~arr~~^{no} | name email tel company



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



arr[2] = new Contact();

* 레퍼런스와 인스턴스 변수

Contact c = new Contact()



① 인스턴스 변수가 없는 저장

c.name = "홍길동";

인스턴스 주소를
알고 있는
레퍼런스

↑
인스턴스
변수

c.email = "hong@";

c.tel = "1111";

c.company = "비트";

② 인스턴스 변경

c = new Contact()



c.name = "이꺽정";

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

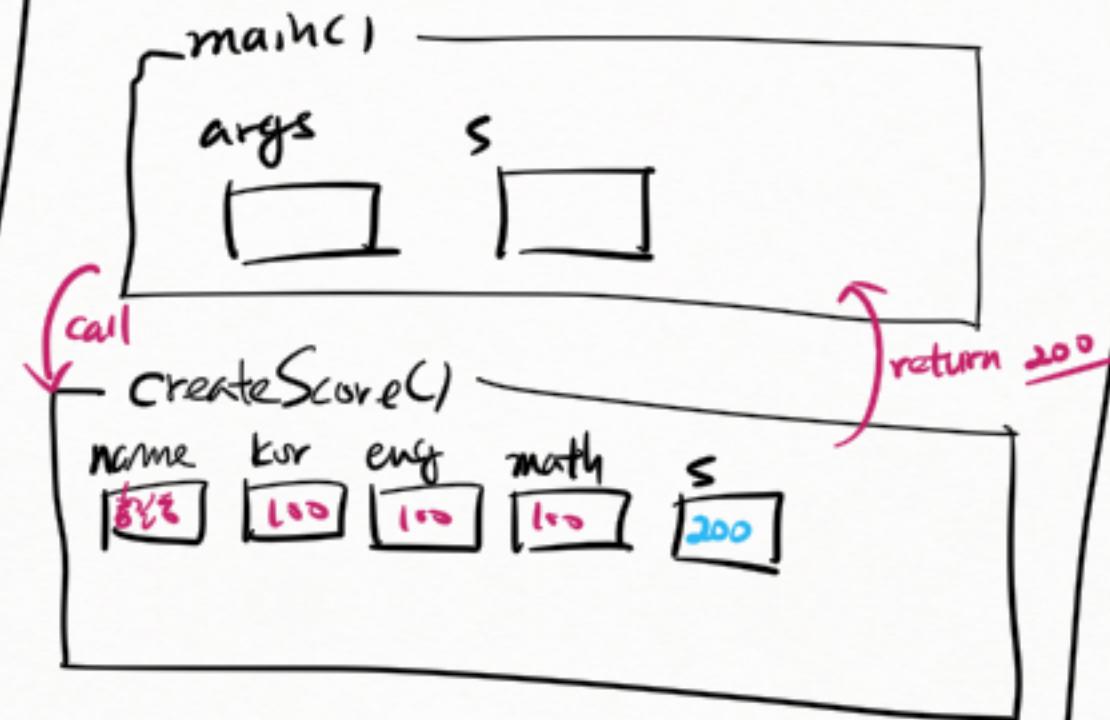
* com.eomcs.oop.exam. Exam0114

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

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

* 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	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	72.5	100	100	100	300

name	kor	eng	math	sum	aver
200	72.5	100	100	100	300

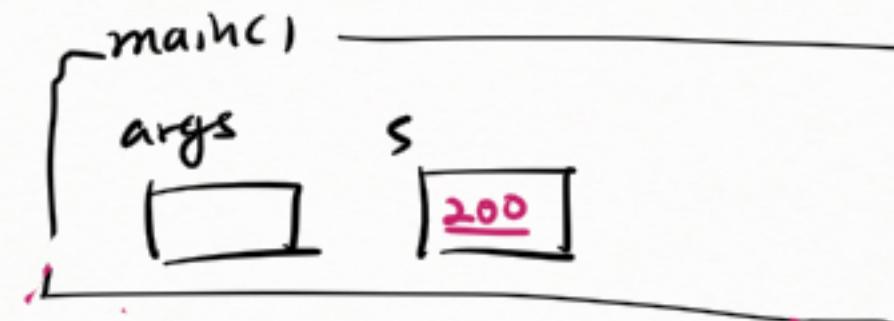
* com.eomcs.oop.exo1. Exam0114

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

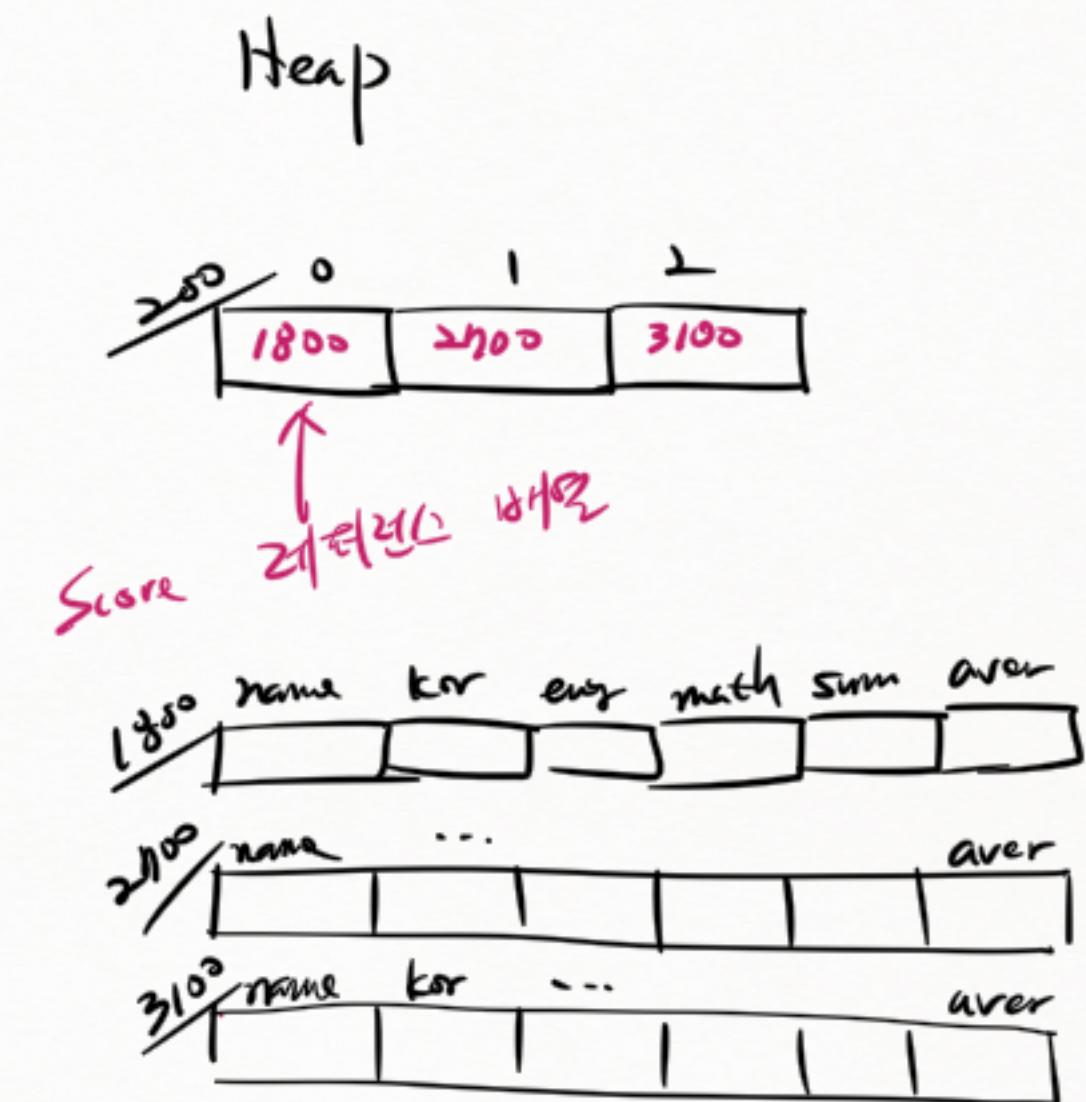
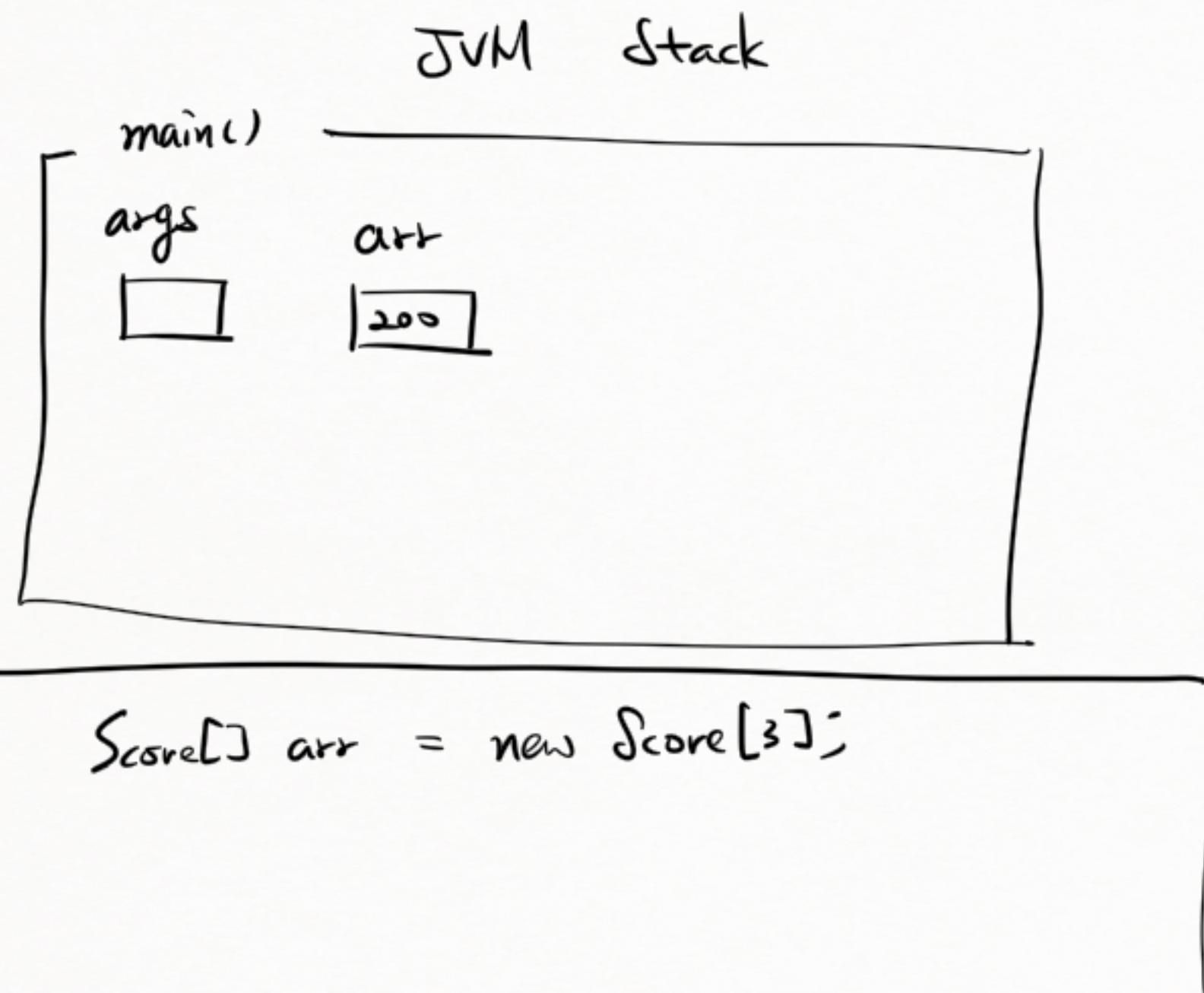
s_3 | 1100

200	name	kur	...

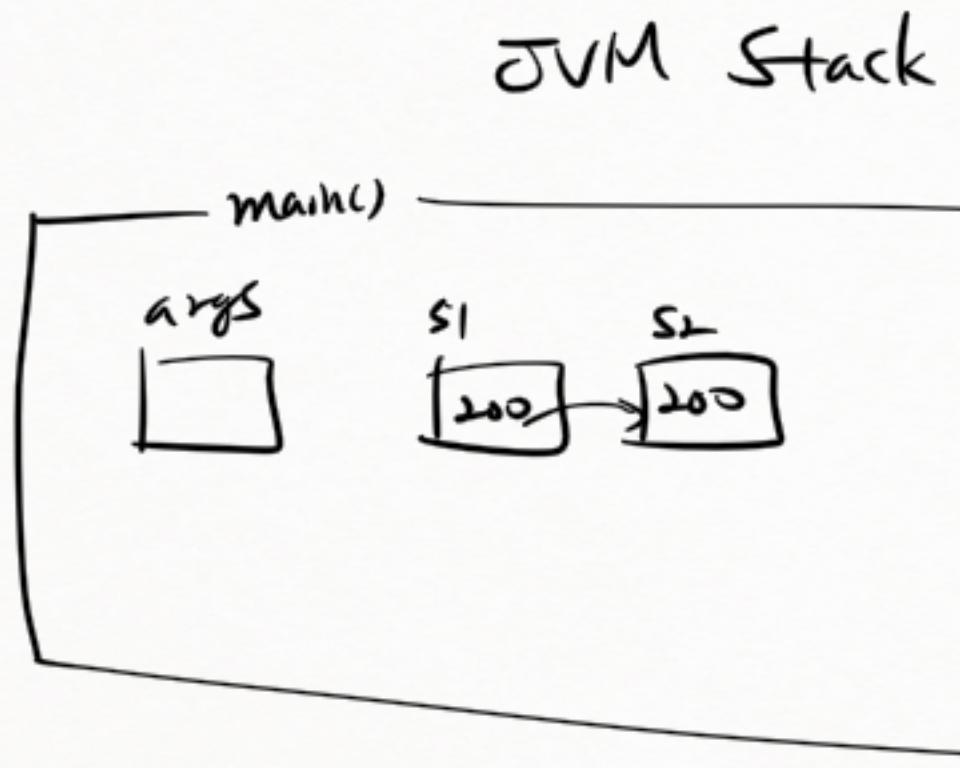
300	name	kur	

1100	name	kur	

* com.eomcs.oop.ex01.Exam0220

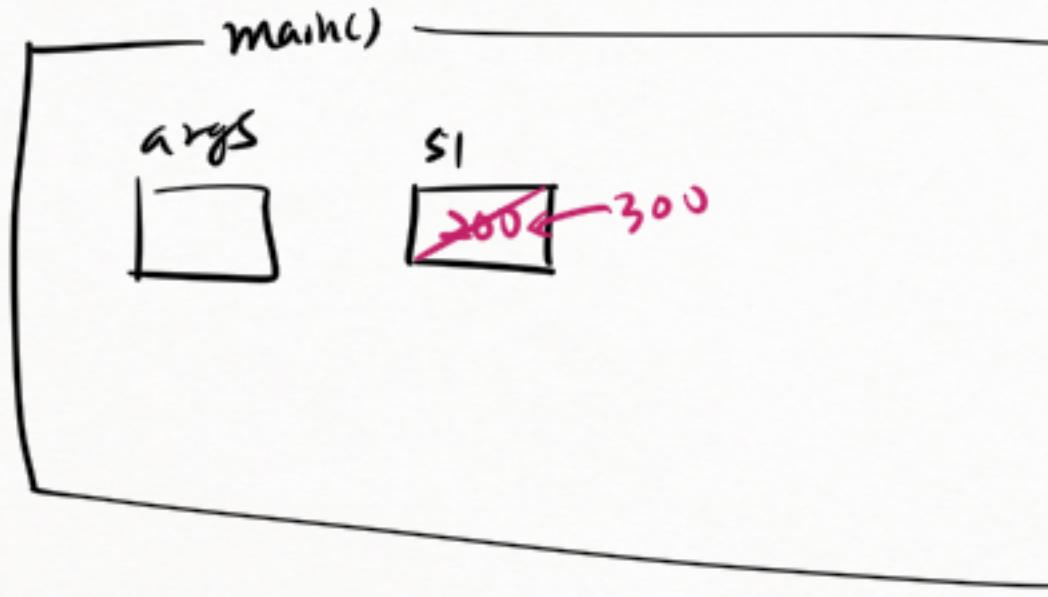


* com.eomcs.007. ex01. Exam0310



* 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

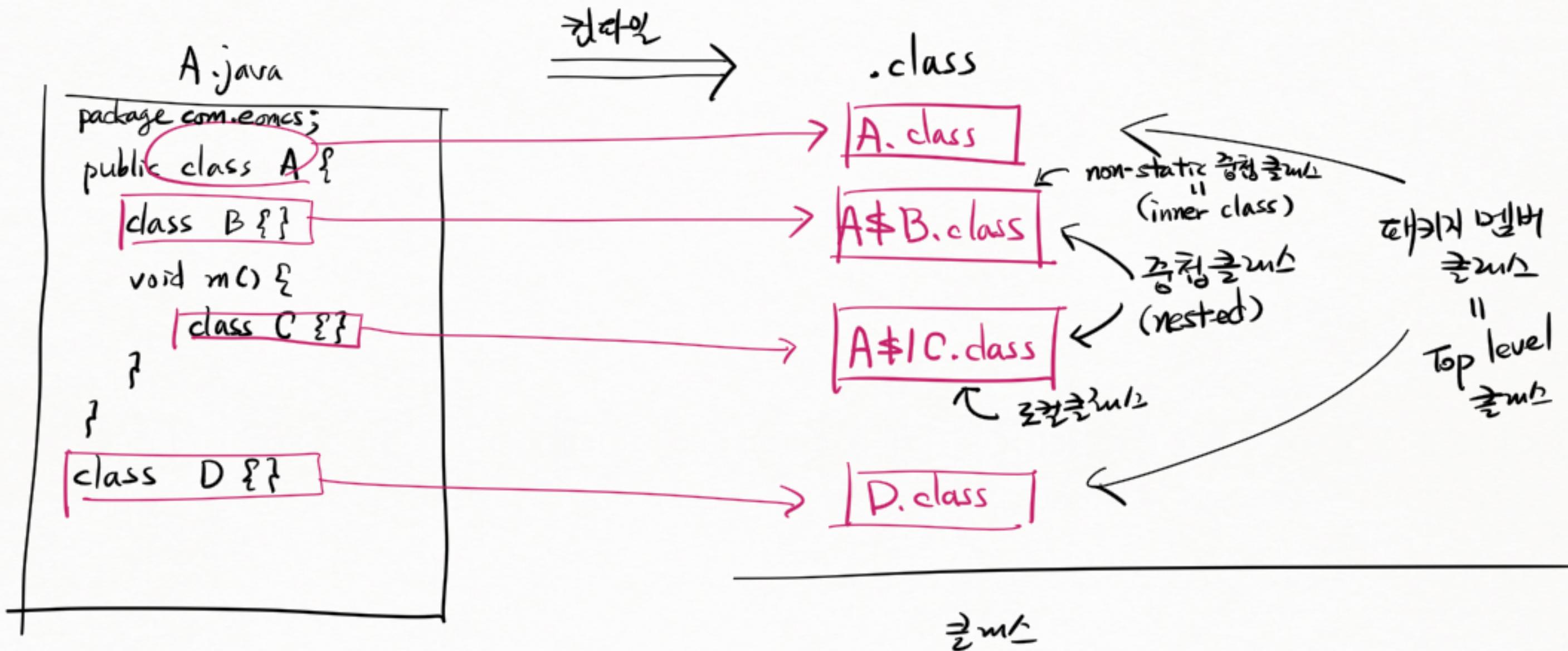
- 부동소수점 변수 = 0.0

- 논리 변수 = false

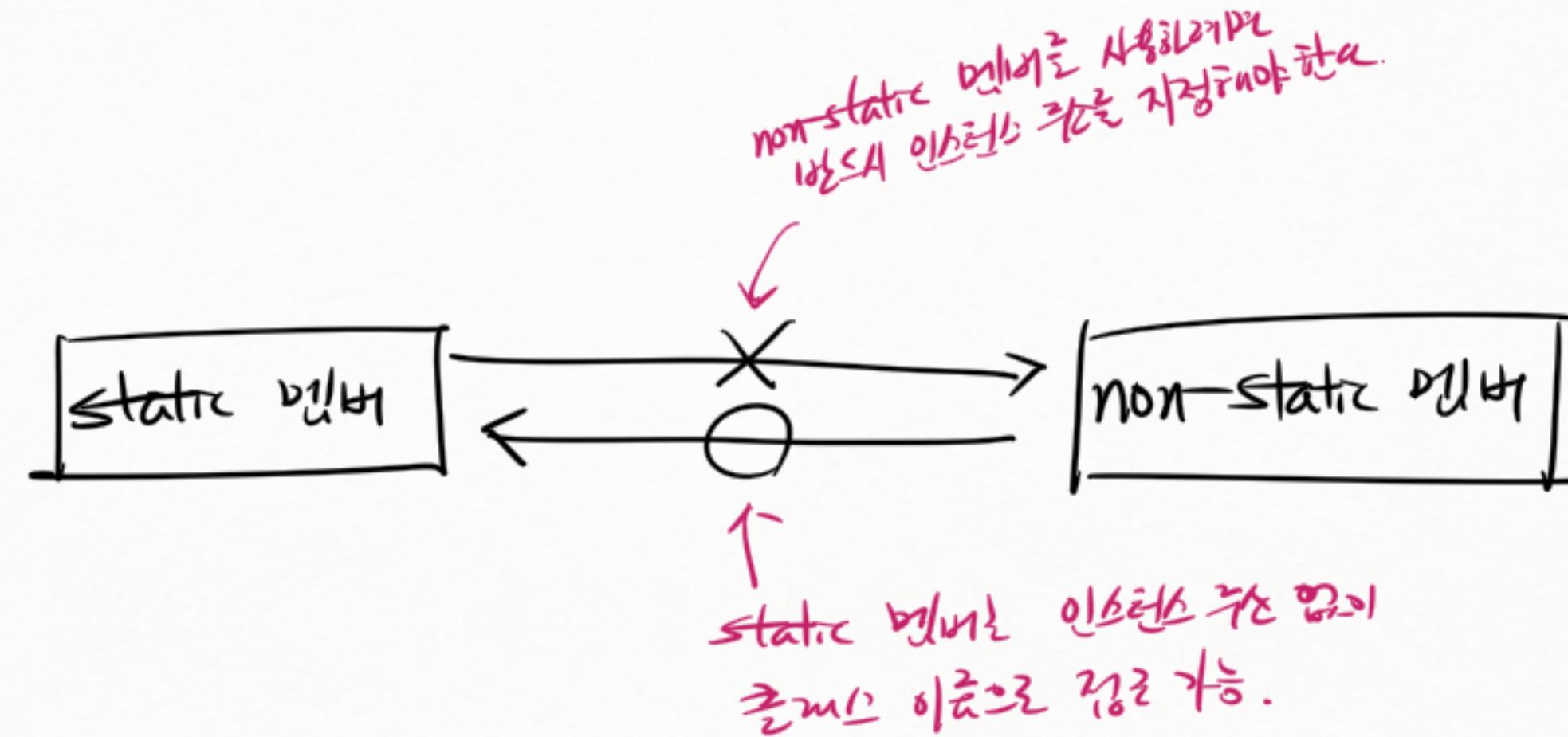
name	kor	eng	math	sum	aver
null	0	0	0	0	0.0

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

* 클래스 복수와 .class 파일



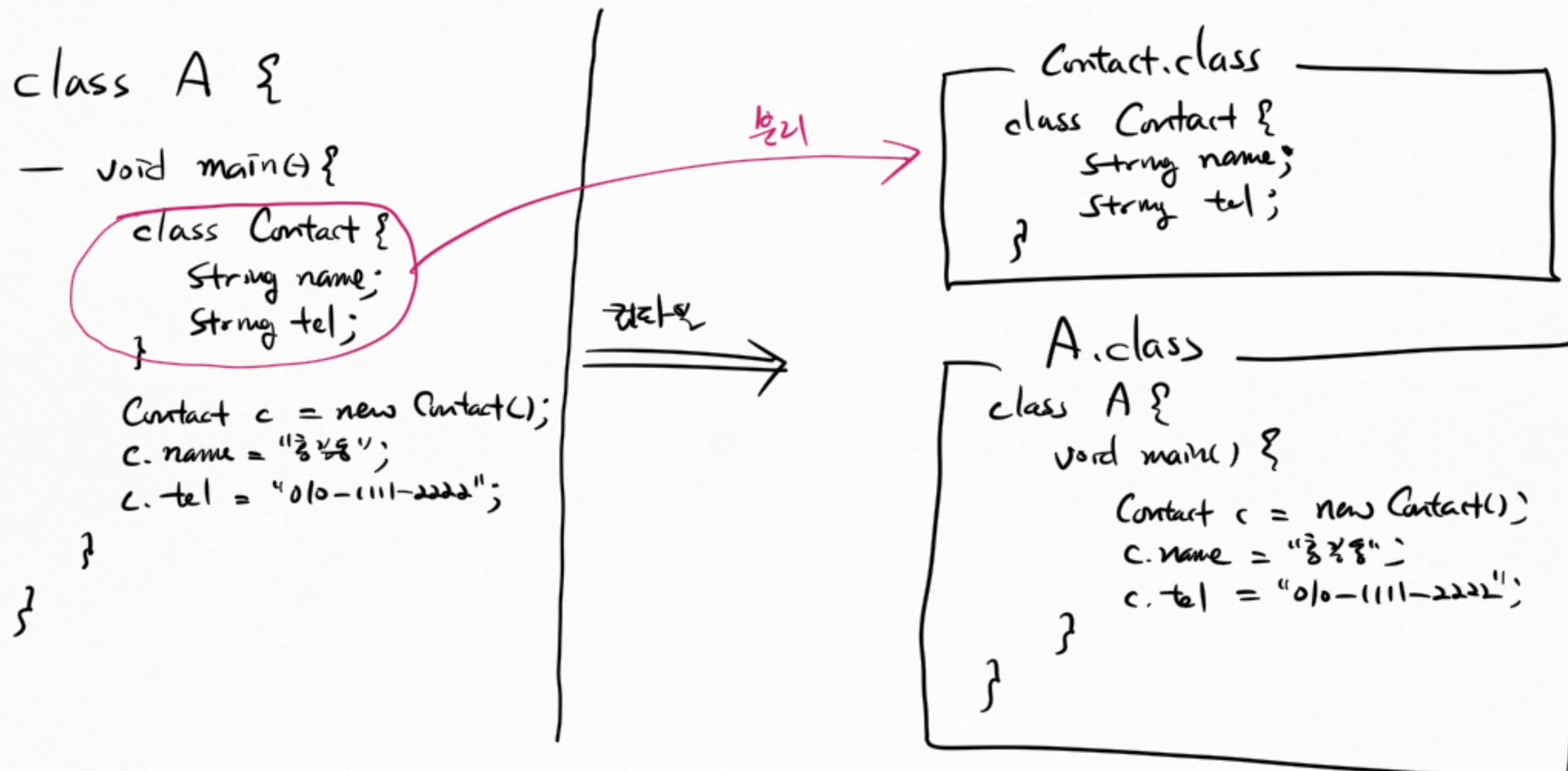
* static 멤버와 non-static 멤버 간의 접근 규칙



* 대기지 멤버 구조 : (default) vs public



* 디자인 원칙과 연결성을 정의



* 클래스 문법 - 사용자 정의 데이터 타입

① 메모리 유형설정

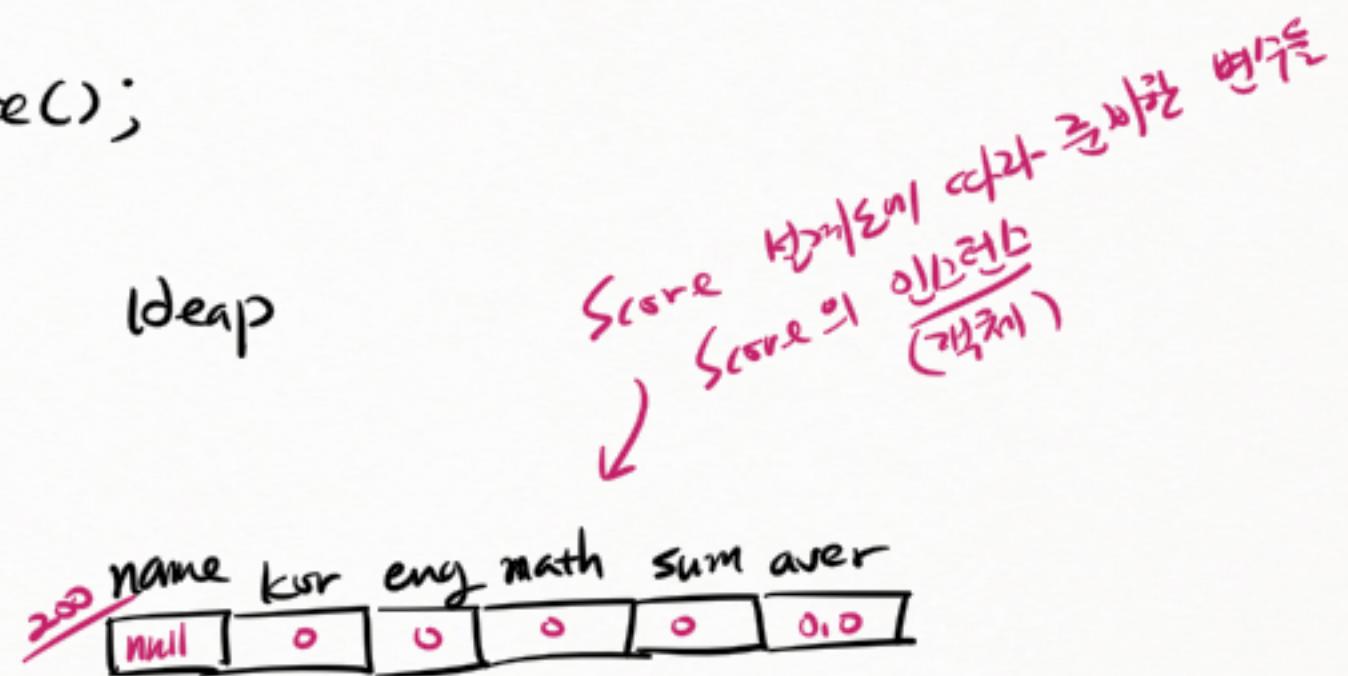
class Score {

```
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  인산자
```

Score s = new Score();

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

Score.calculate(s);

설계문장 → 인산자(Operator) 대(연산자)

설계문장 → 메서드(method) = 함수(function)

설계문장 → 메시지(message)

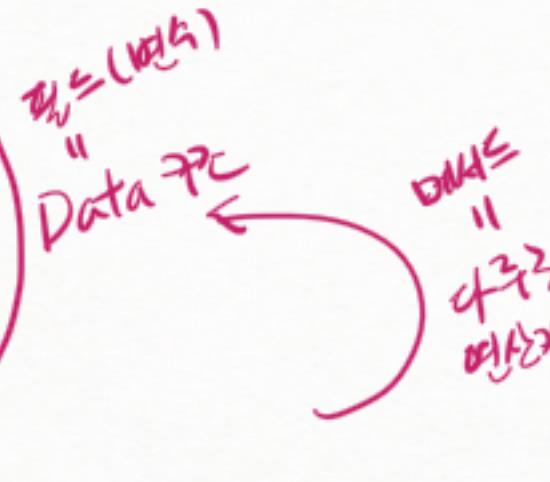
* 클래스 문법 - 사용자 정의 데이터 타입

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;
```



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

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

}

}

int a;
a = -100; ⇔
a++;
↑ ↑
파인스터 인수자

기존의
연수자를
사용하는 방법은
더 비슷하다

Score s = new Score();

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

~~s.calculate(s);~~

파인스터 ↓
s.calculate(); 인수자
 ↓
 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

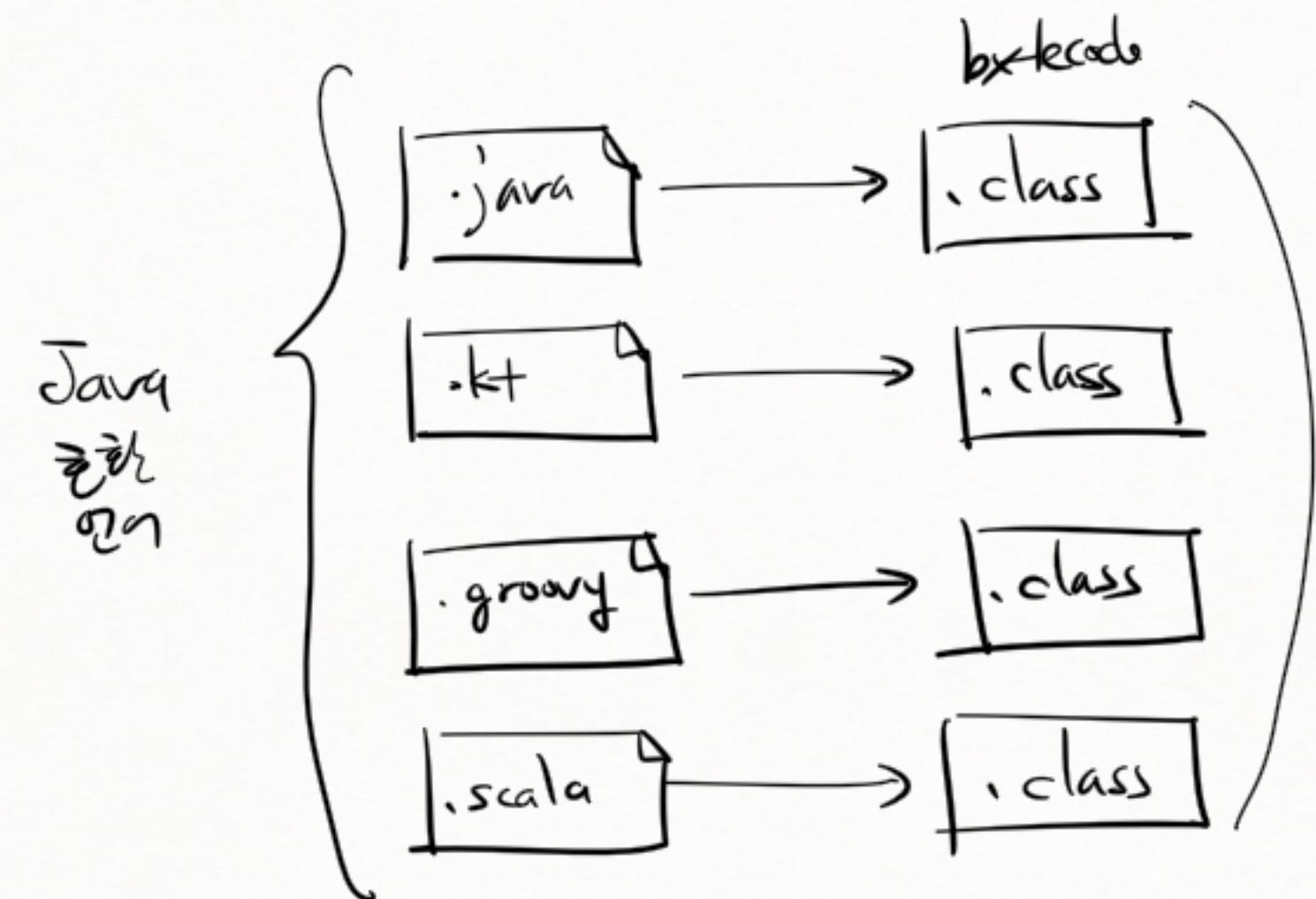
* modifier

final
static
private
⋮

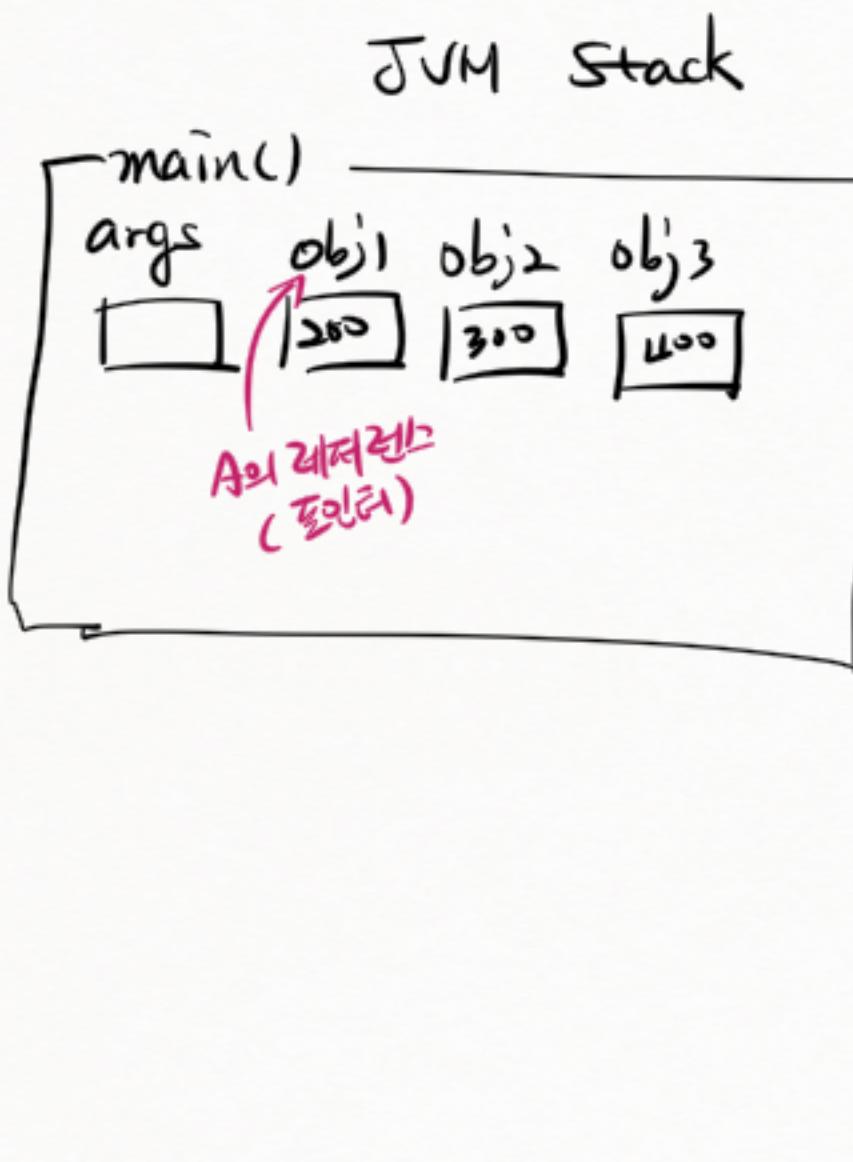
int $i = 100;$

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

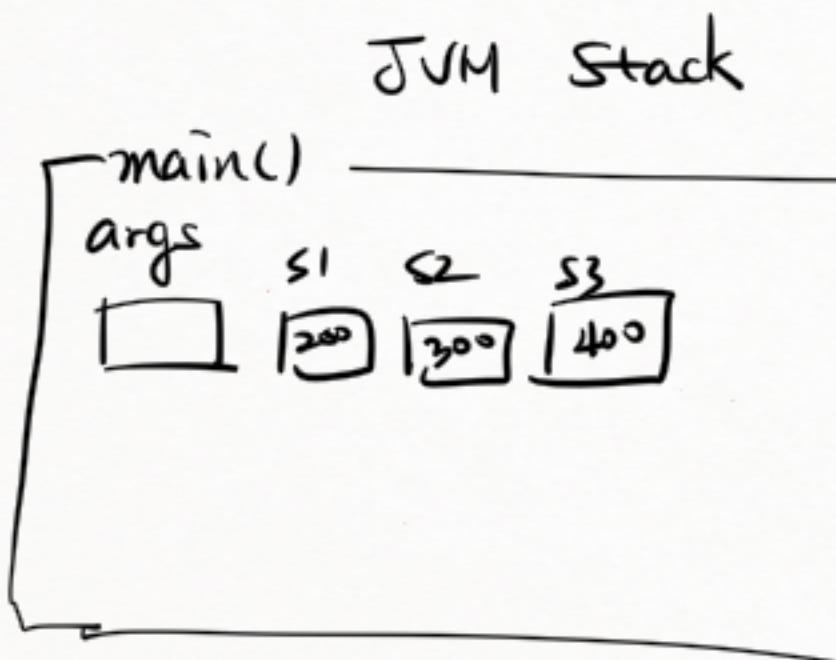
* 자바 향한 언어



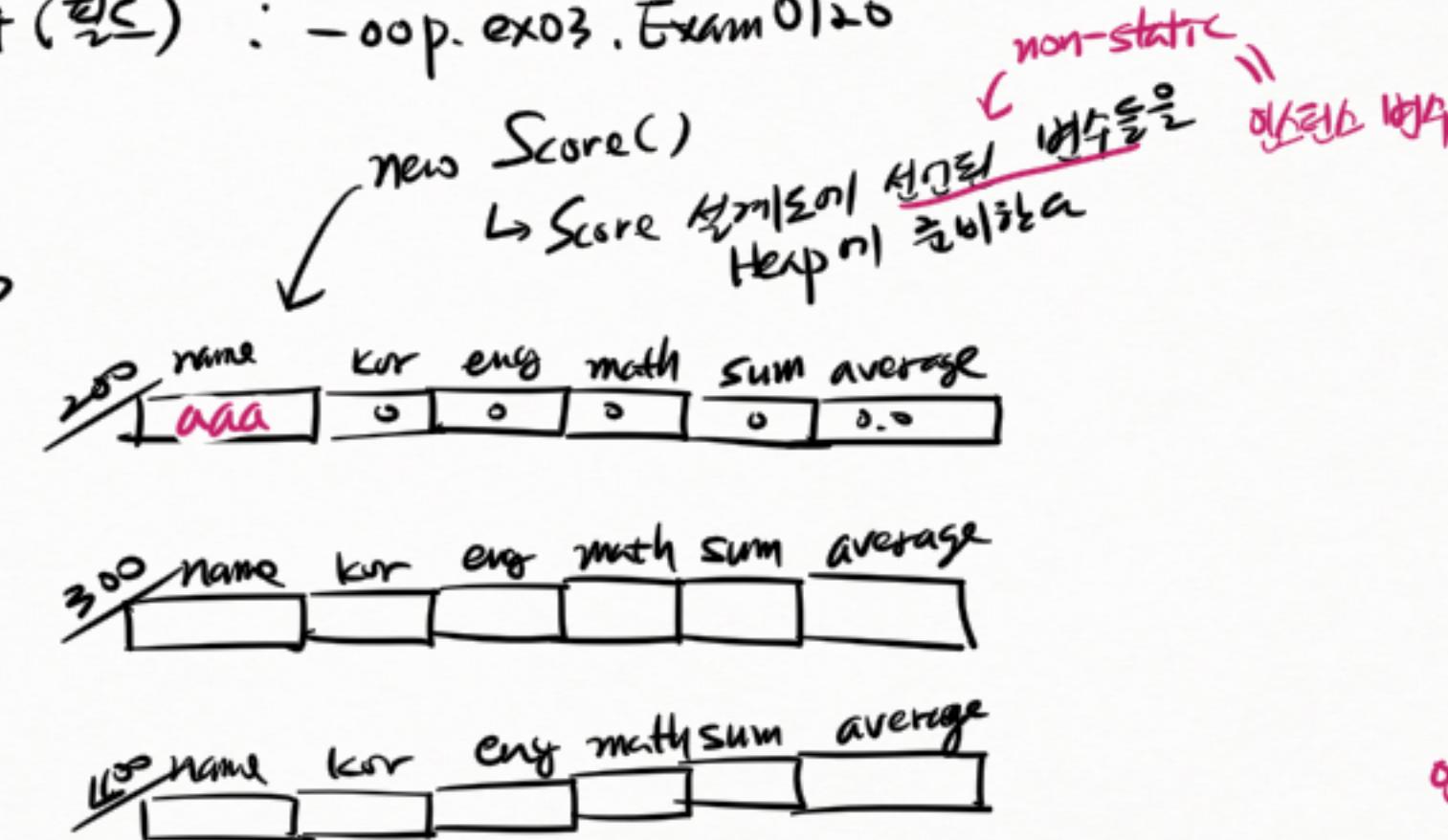
* 인스턴스 변수(필드) : - oop. ex03, Exam 0110
non-static



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

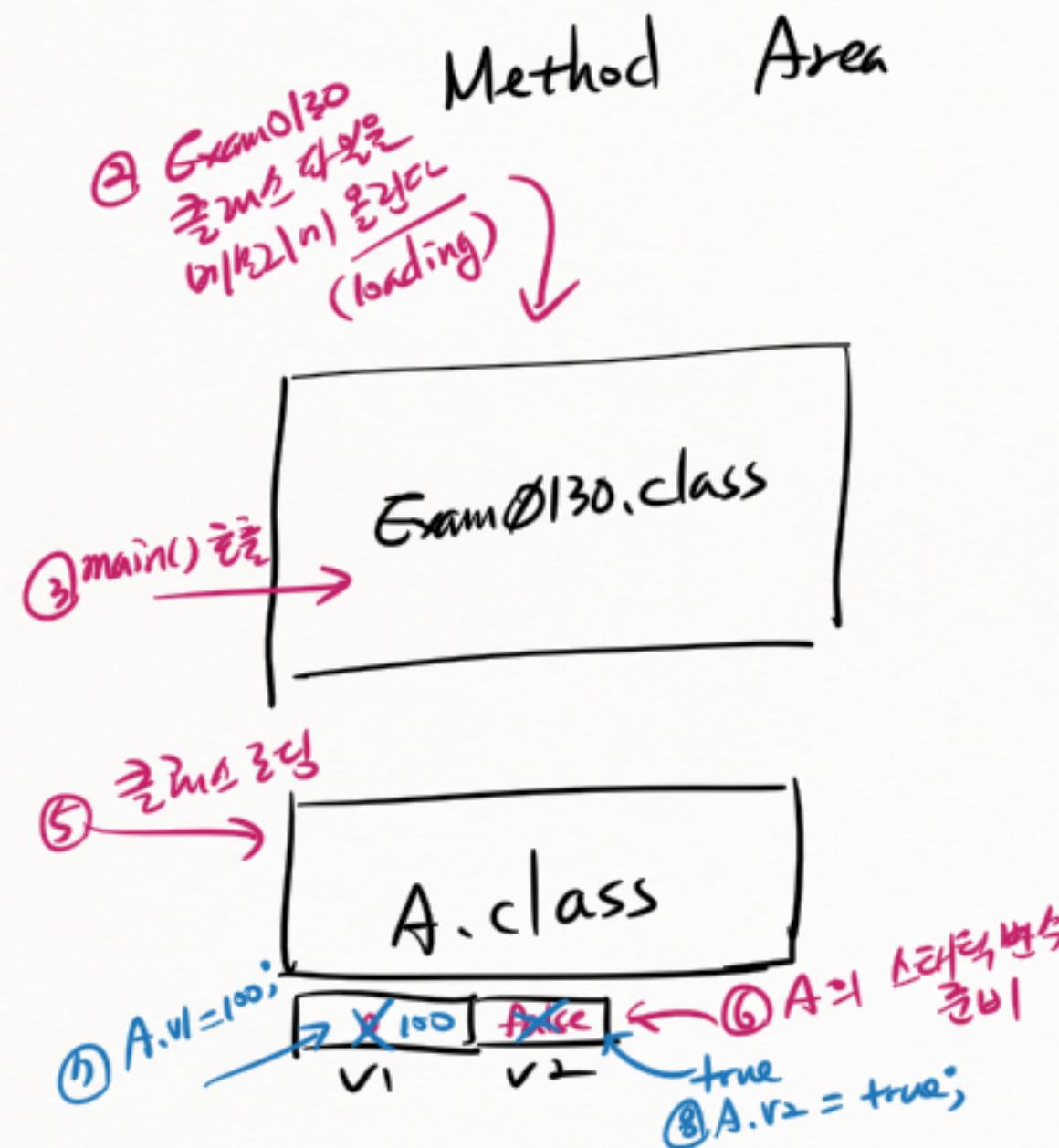


Heap



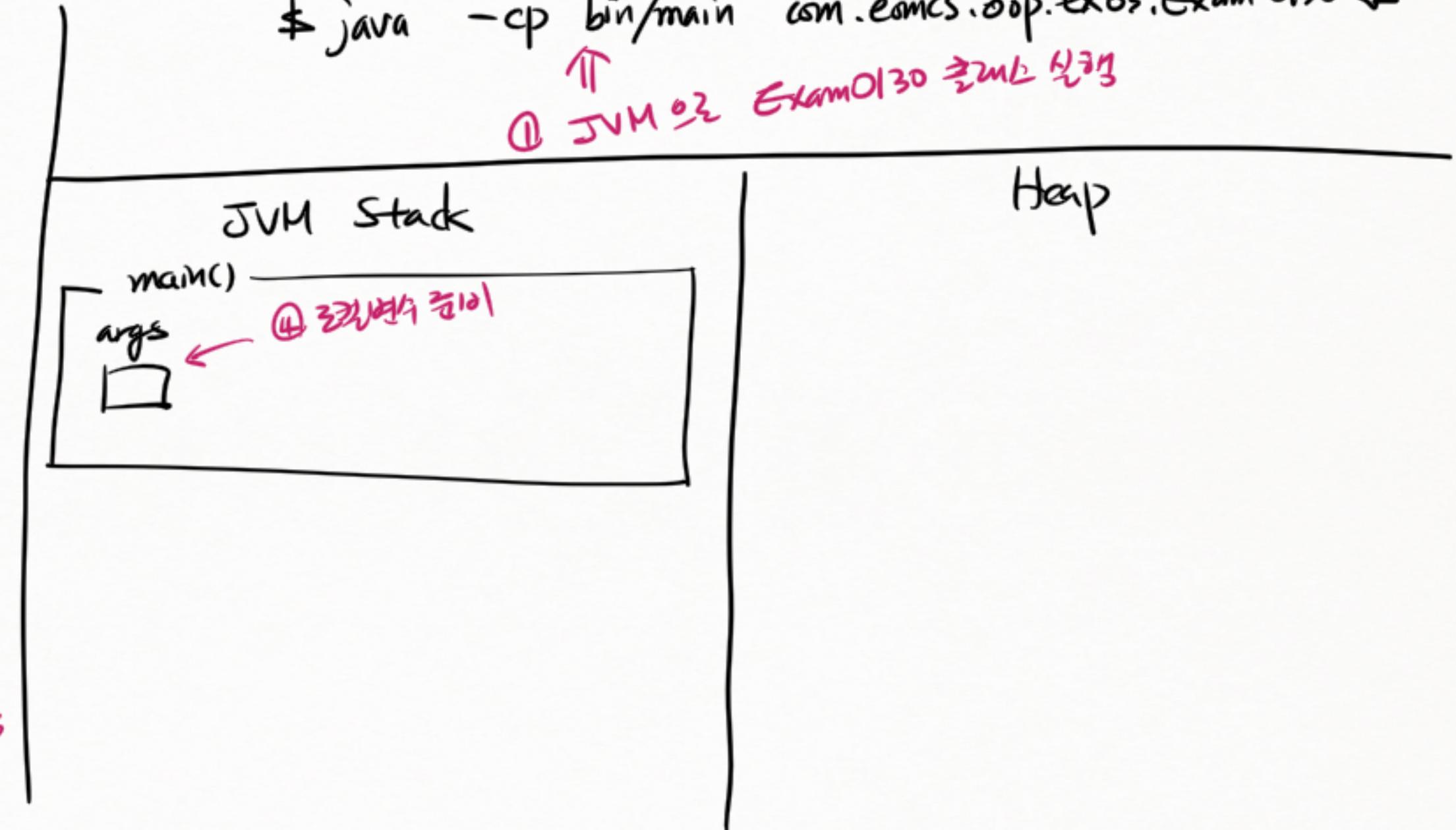
s1.name = "aaa"
 nm
 200
 ↑
 인스턴스 주소

* 정적 변수(필드) : - oop.ex03.Exam0130
static

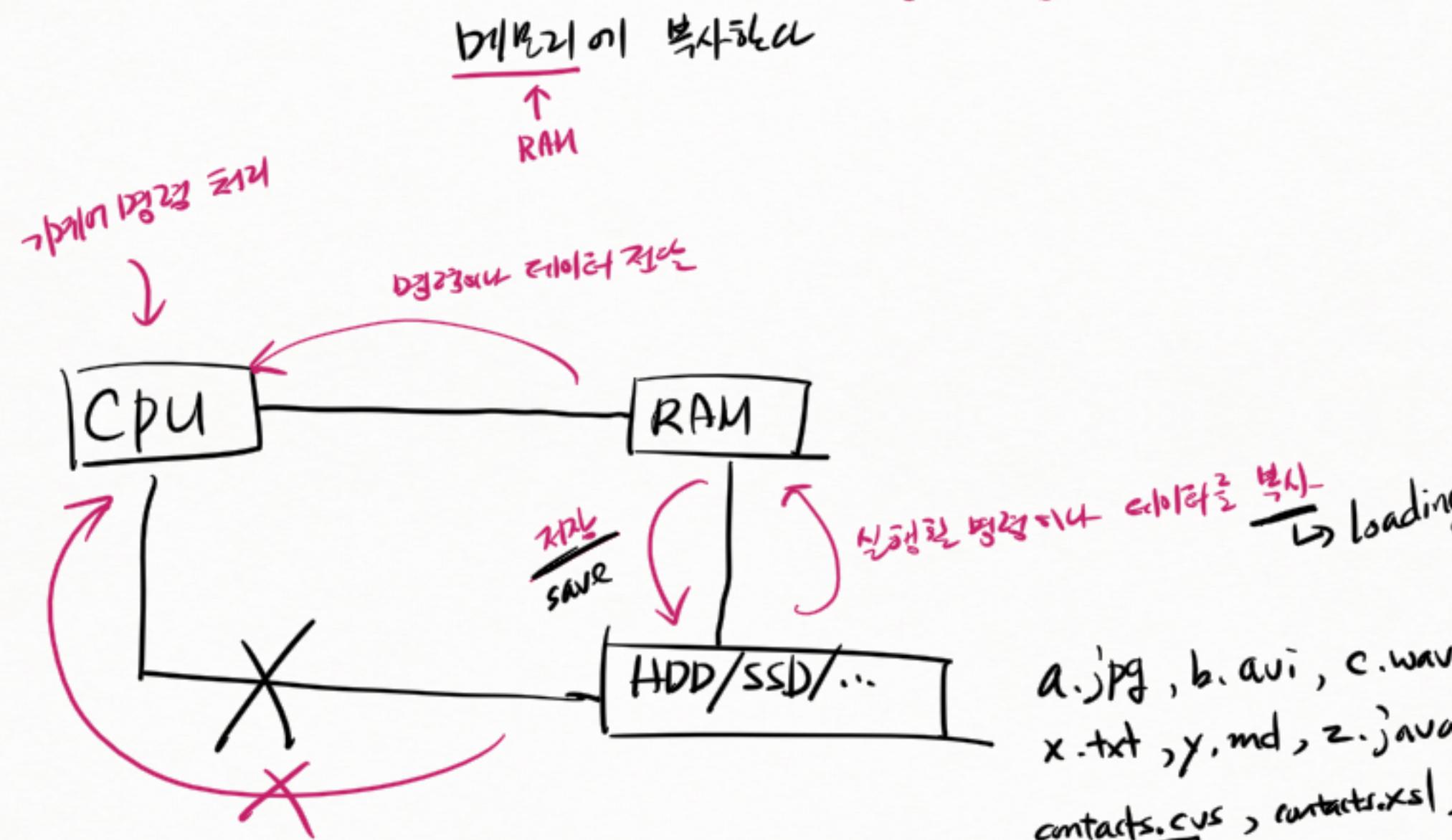


\$ java -cp bin/main com.eomcs.oop.ex03.Exam0130 ↴

① JVM으로 Exam0130 정적 변수 실행 ↴



* 클러스터링 (clustering)

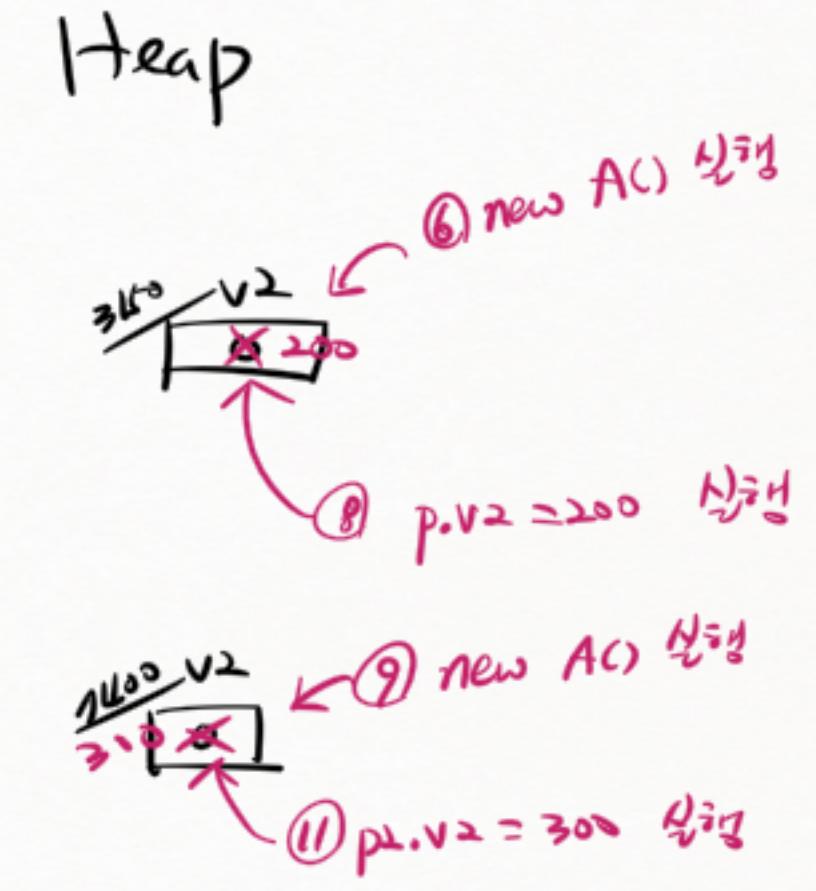
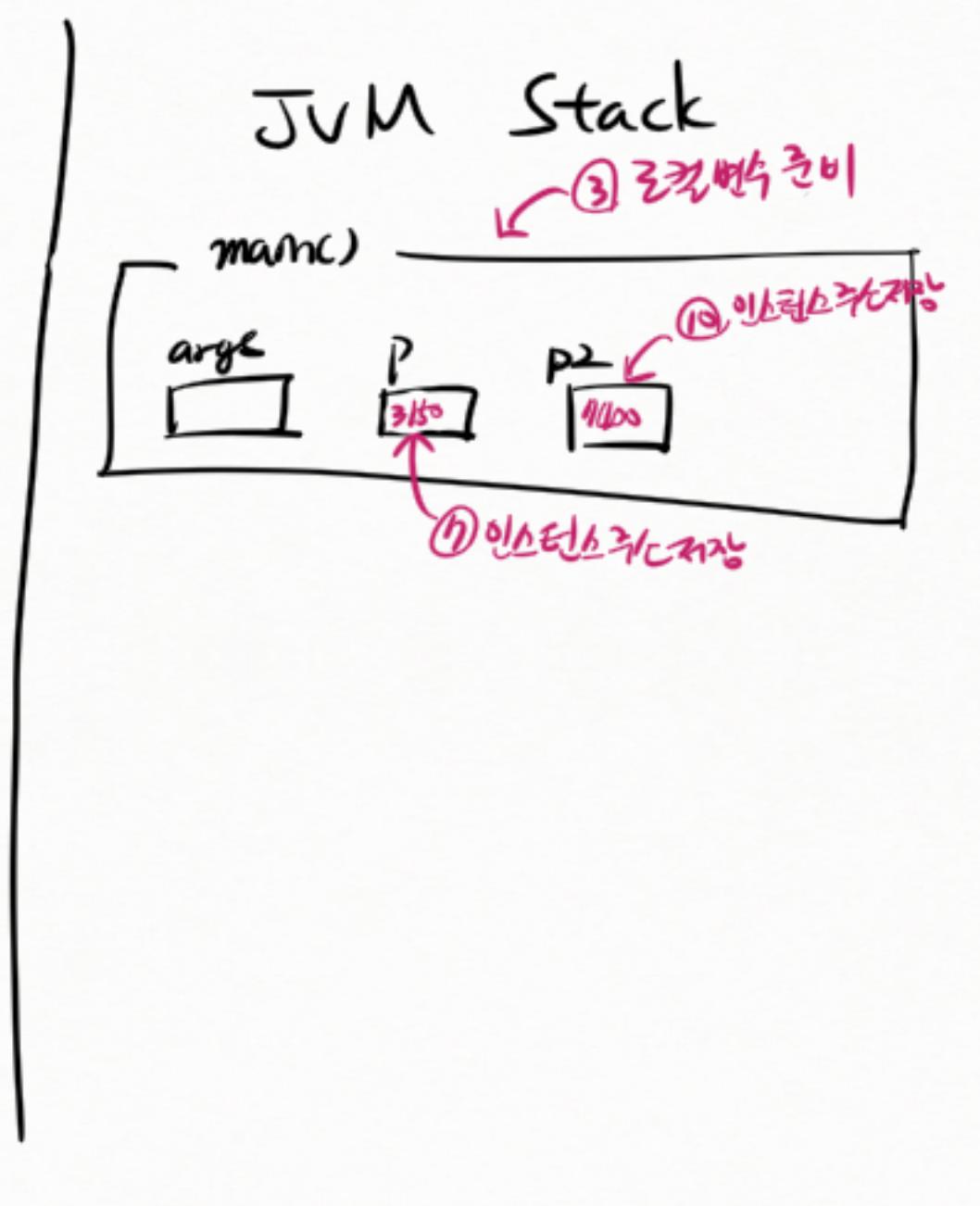
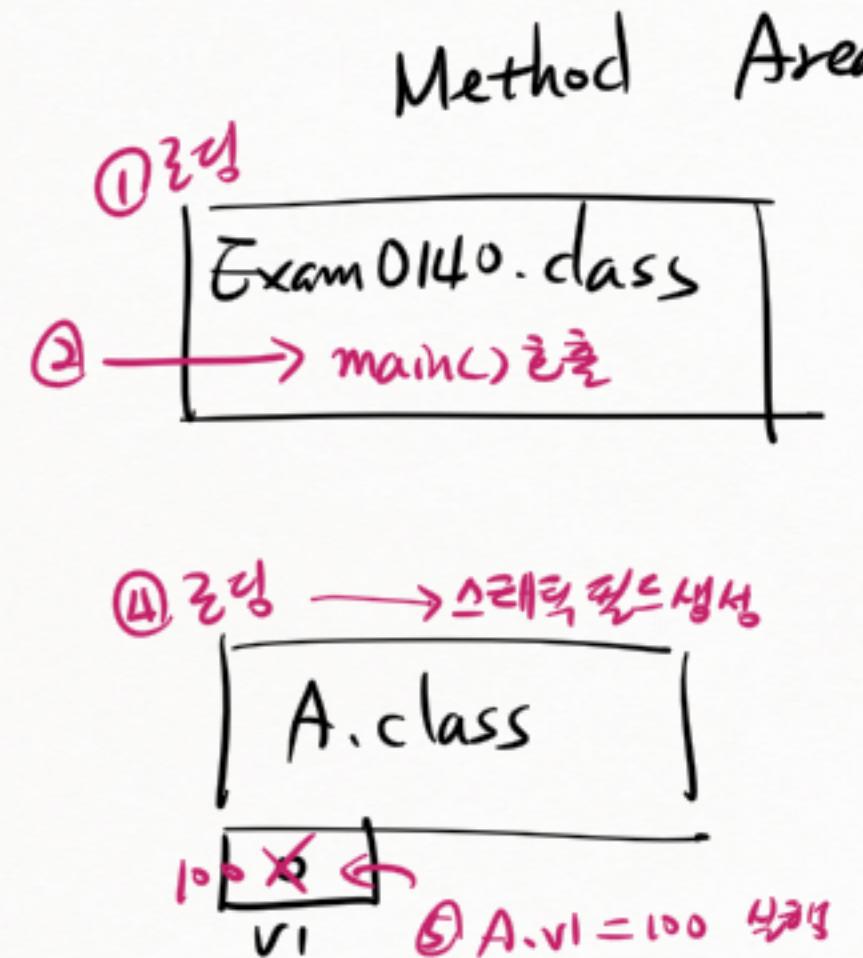


$y_2z = 14$ \leftarrow $\text{color}^2 \frac{2}{2}$ 부서

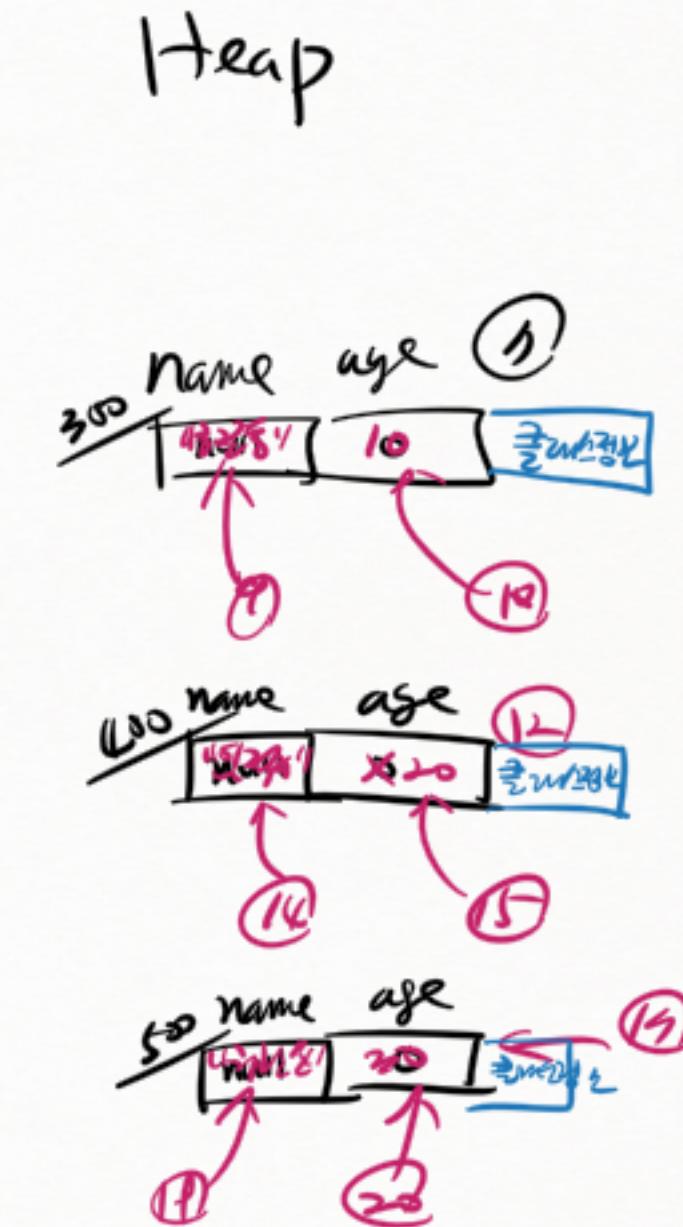
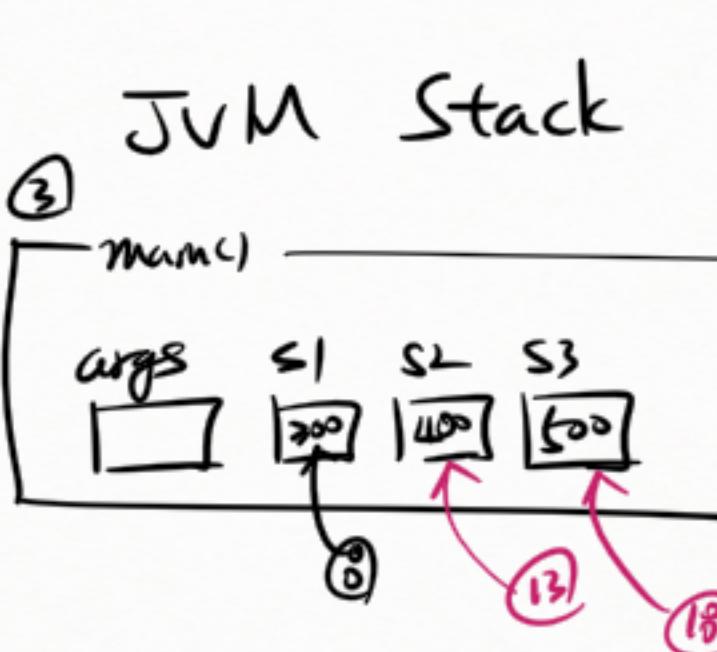
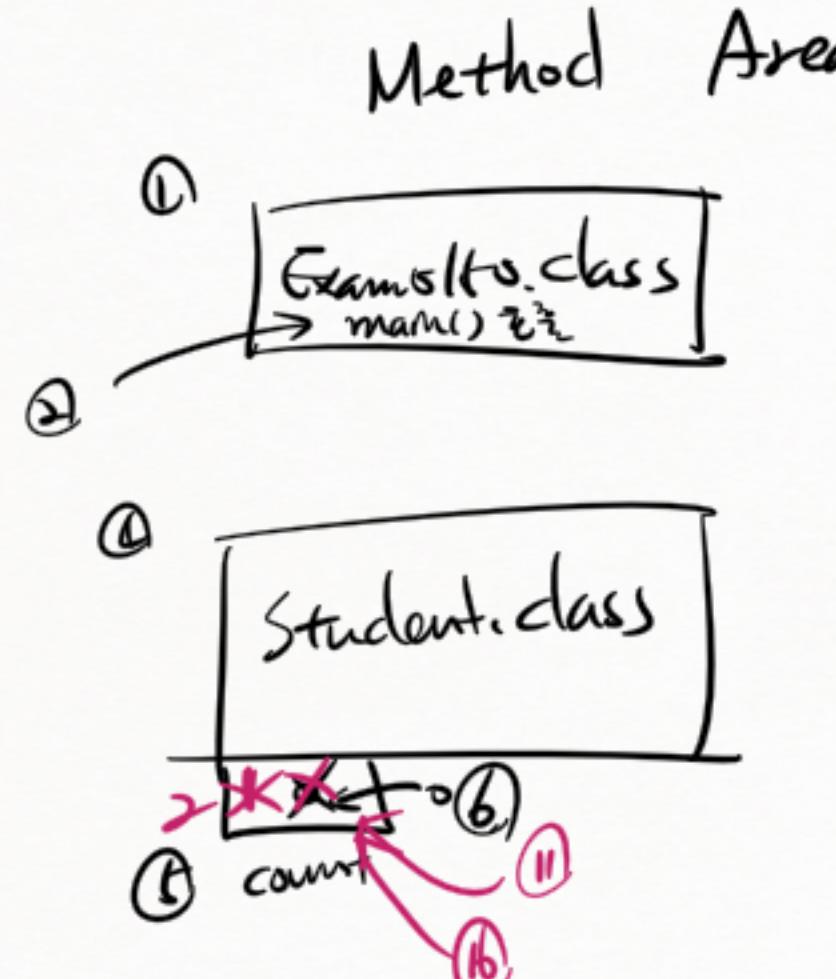
↳ loading

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

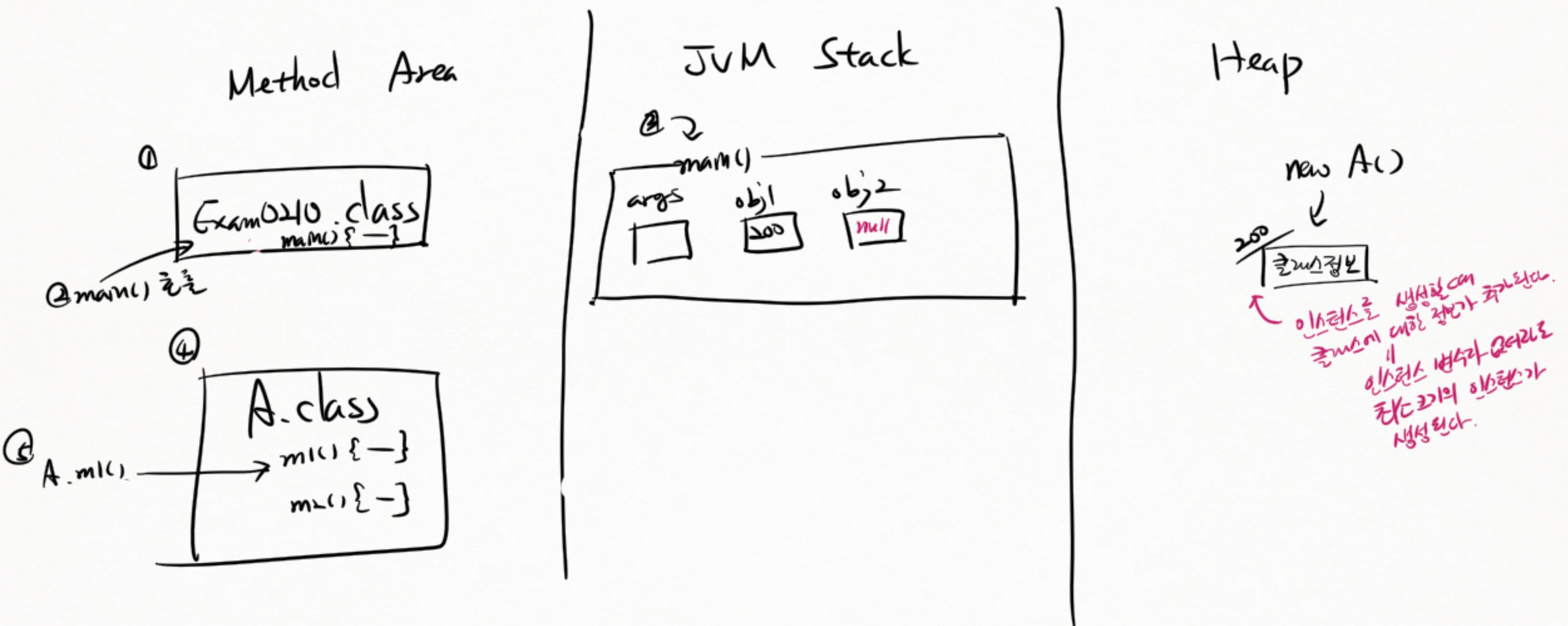
* 글래스 변수와 인스턴스 변수 : oop. ex 3. Exam0140



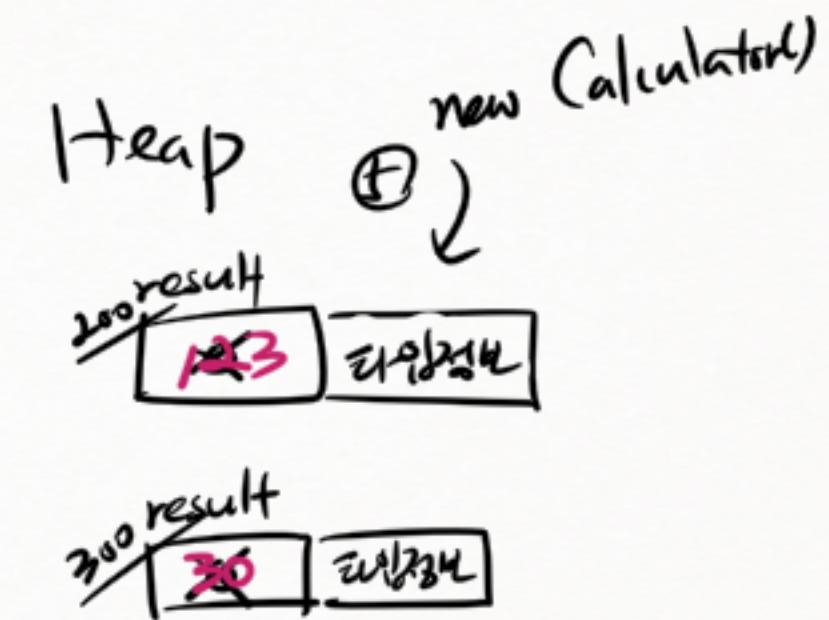
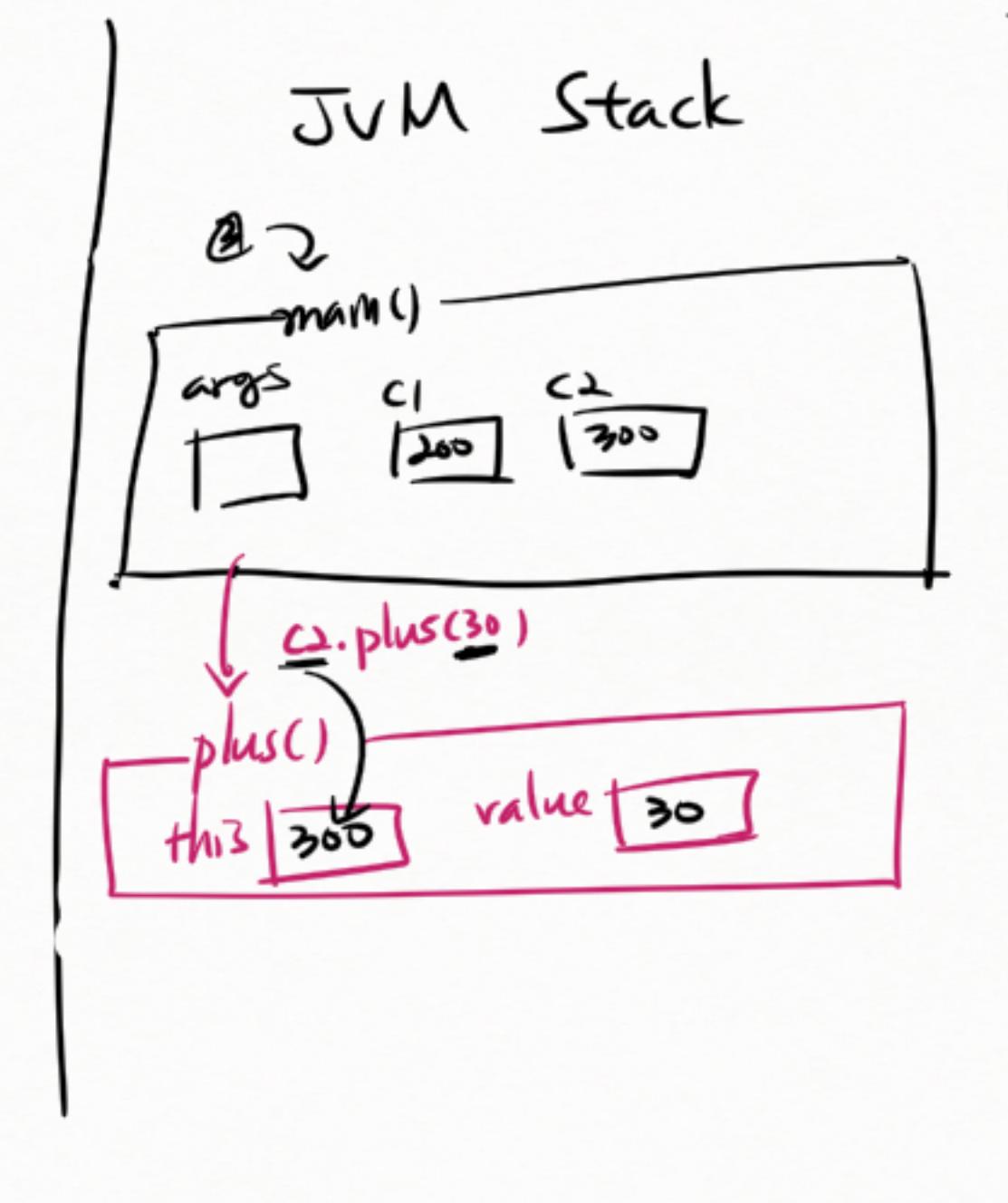
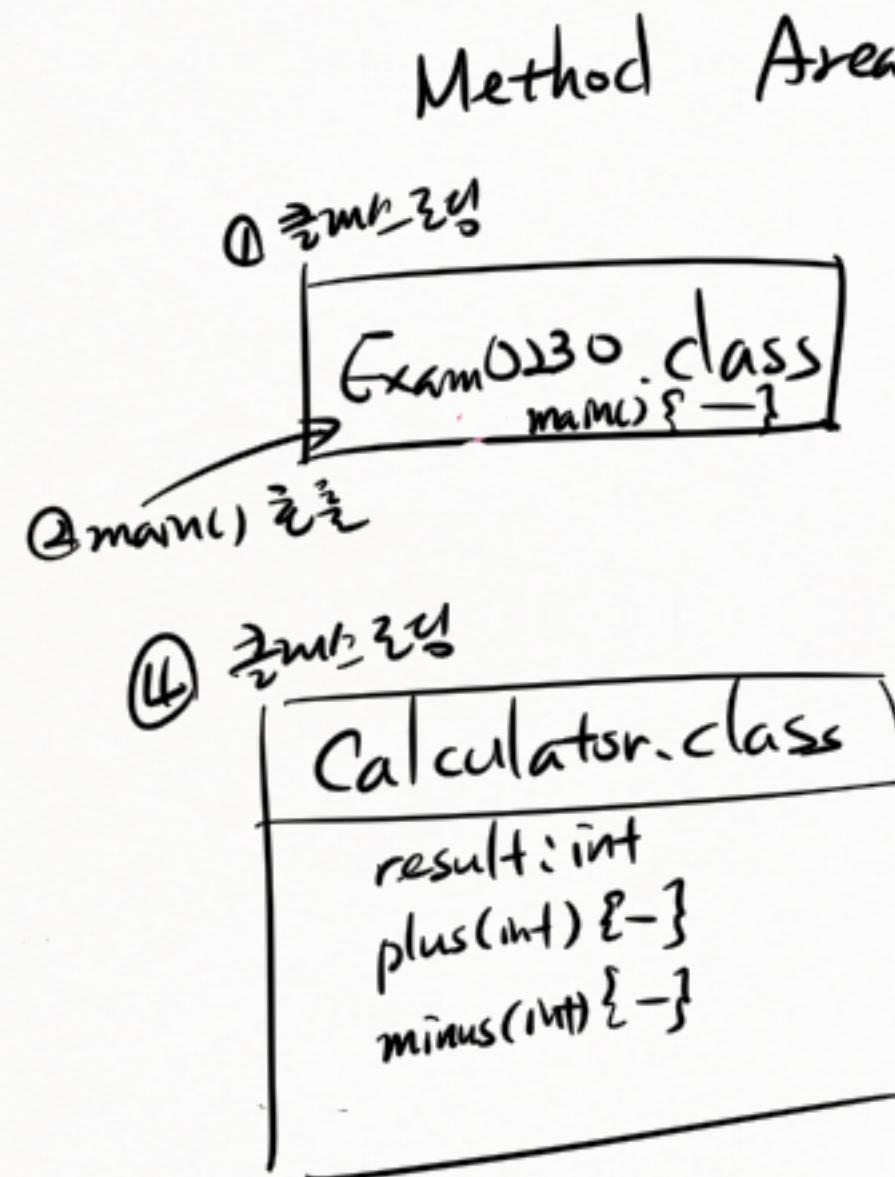
* 주소 변수와 인스턴스 변수 : oop. ex03. Exam0150



* 접근스 박스와 인스턴스 메소드 : oop. ex03. Exam0210



* 디스터스 멤버(변수와 메서드) : oop. ex03. Exam0230



* 생성자

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

~~Score()~~

) 생성자(Constructor)

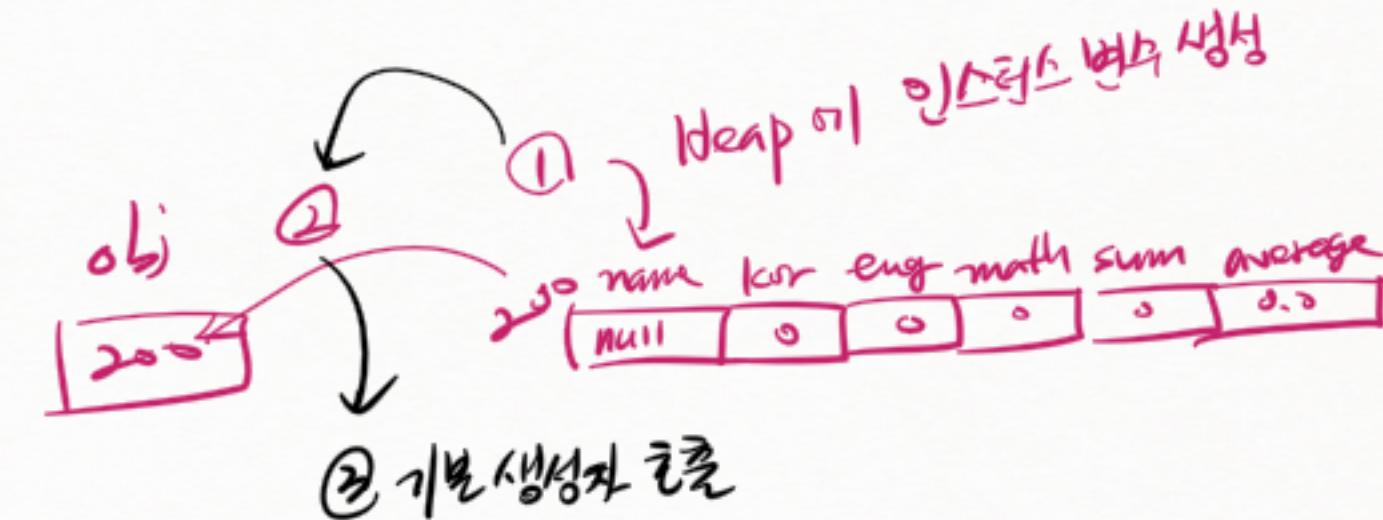
자바비전가 같은 생성자
"default constructor"

Score obj = new Score();

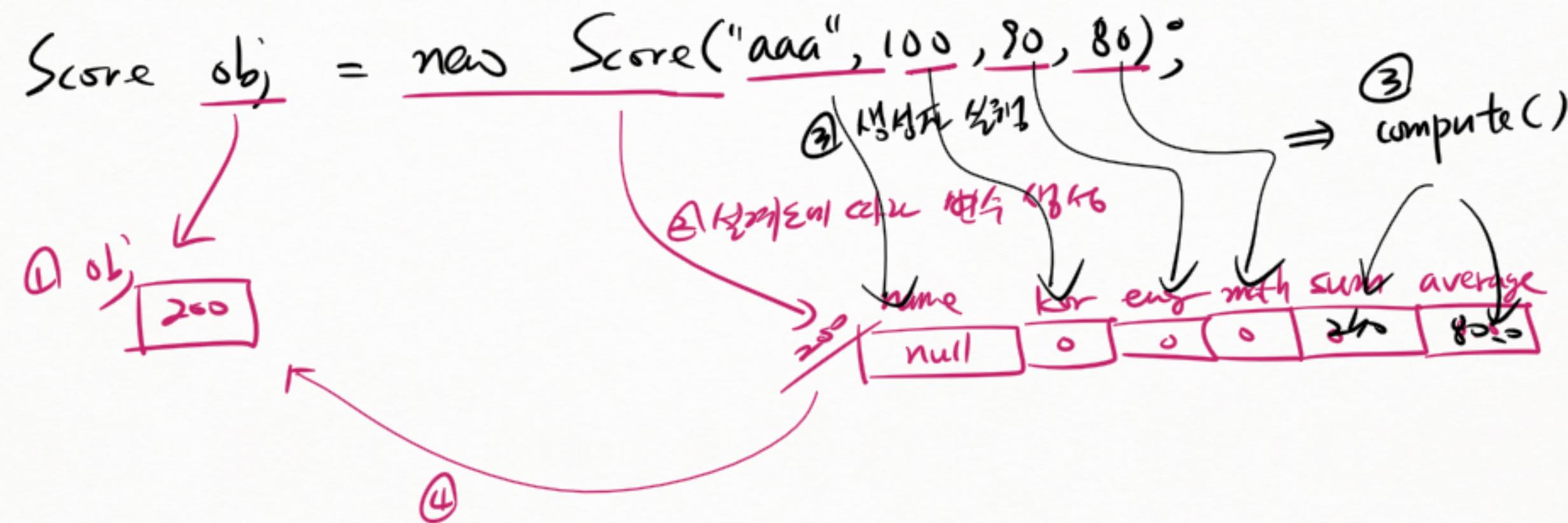
↑
클래스명
↑
자동으로 호출될 생성자를 지정.

타입 primitive type
class interface enum

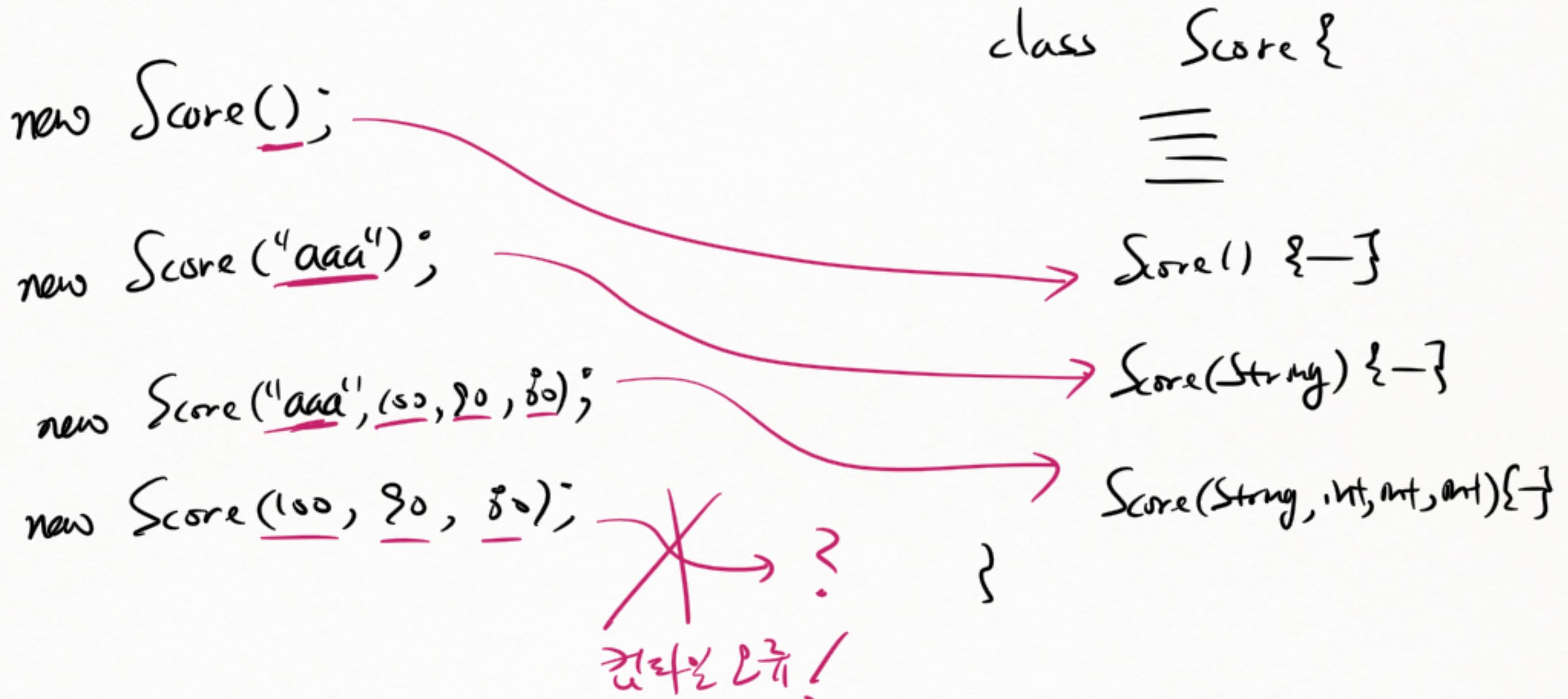
리퍼런스
" (포인터)



* 생성자 활용 \Rightarrow 인스턴스 변수를 초기화시킨다.

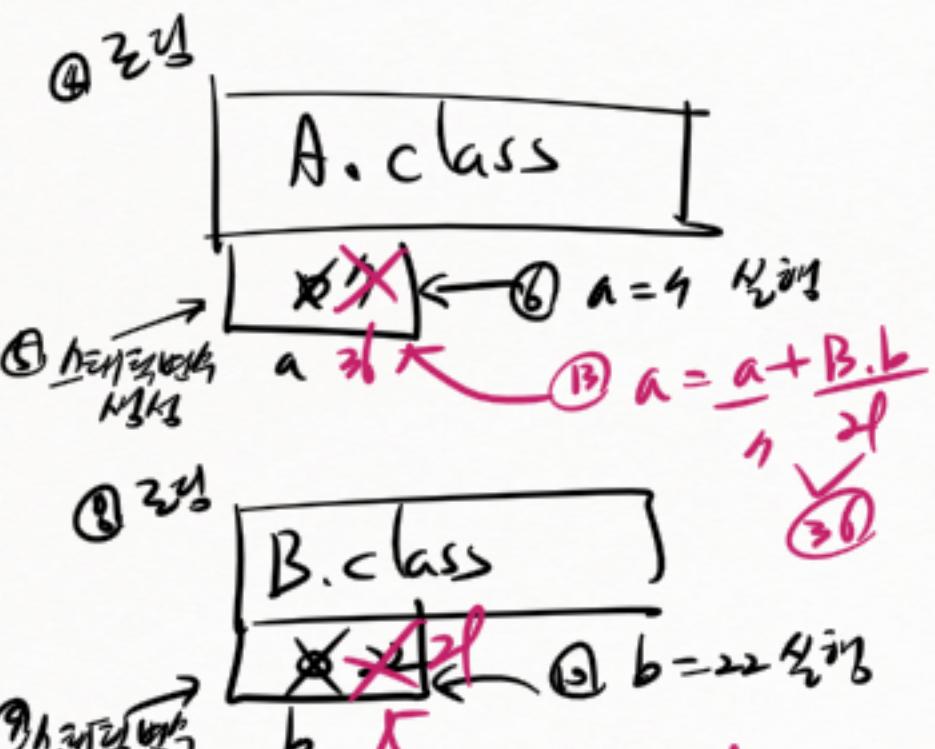
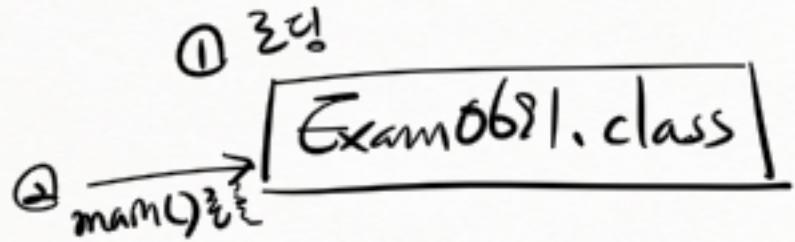


* 헤더 파일 생성자를 결정하는 방법

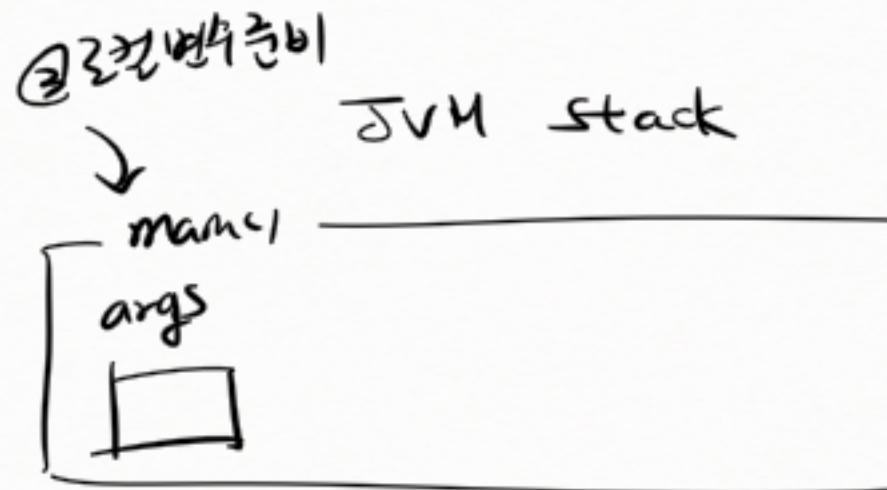


oop. ex03. Exam0910

Method Area



$$\begin{aligned} & \text{⑪ } b = \cancel{22} + \frac{A.a}{\cancel{22}} \\ & \quad \checkmark \end{aligned}$$



console \rightarrow

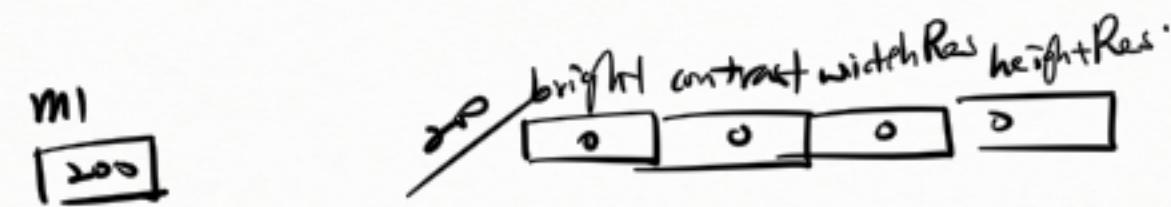
① \hookrightarrow $A.\text{static}\{ \}$

② $\xrightarrow{\text{static }} \text{ static } \frac{\text{정적 }}{\text{변수}}$

⑪ \hookrightarrow $B.\text{static}\{ \}$

⑫ $\xrightarrow{\text{static }} \text{ static } \frac{\text{정적 }}{\text{변수}}$

`new Monitor()`



m1.display();

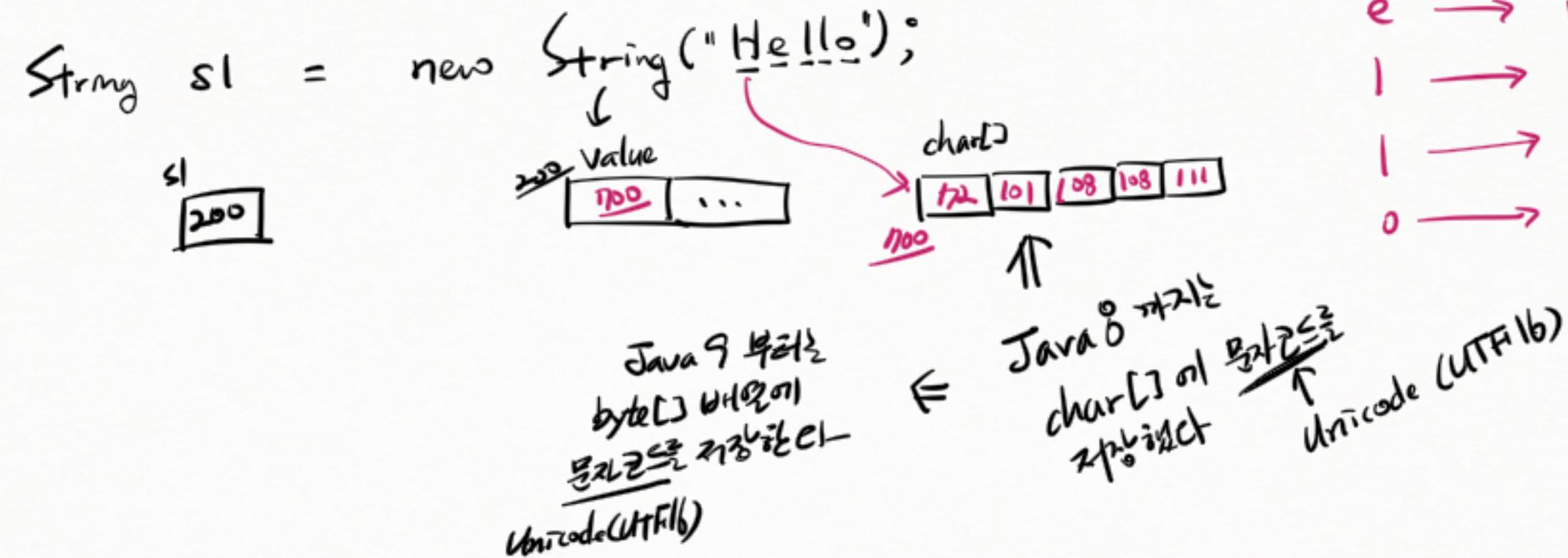
200

this



* String 클래스의 생성자

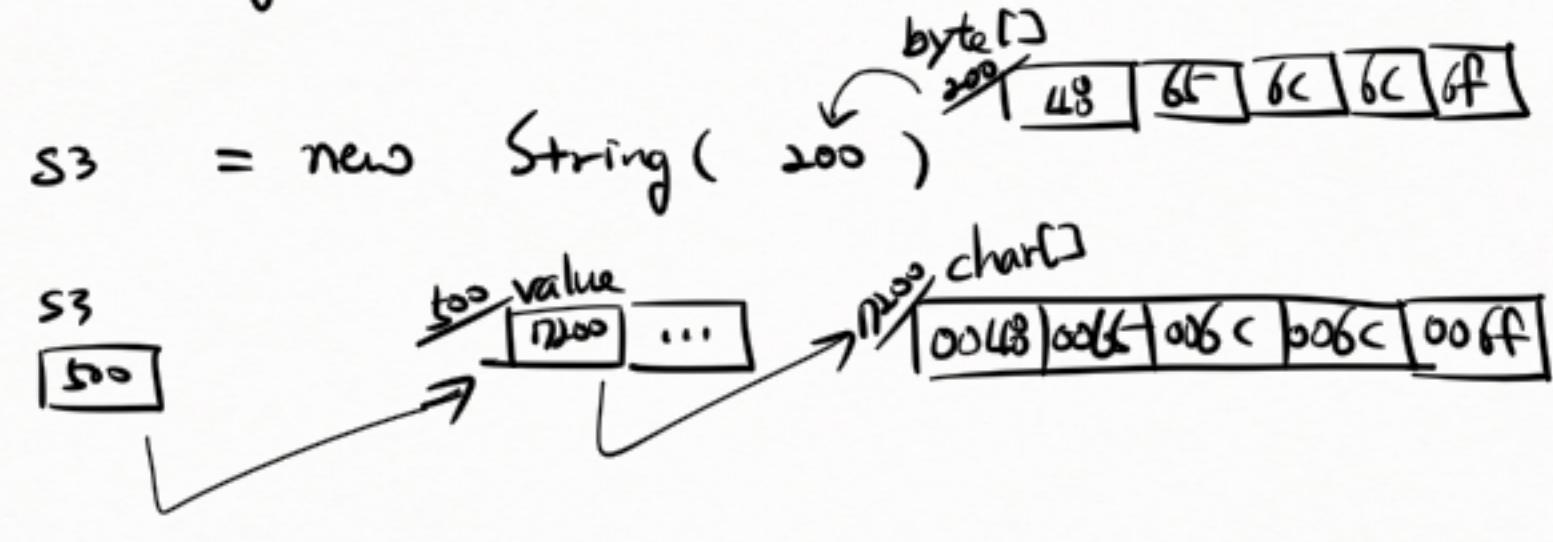
String s0 = new String();
기본 생성자 (default constructor)



문자	JVM이 사용하는 (charset)	Unicode 값 (UTF-16)
H	→	72 (0x48)
e	→	101 (0x65)
l	→	108 (0x6C)
l	→	108 (0x6C)
o	→	111 (0x6F)

* String 클래스의 생성자

String s3 = new String("Hello")



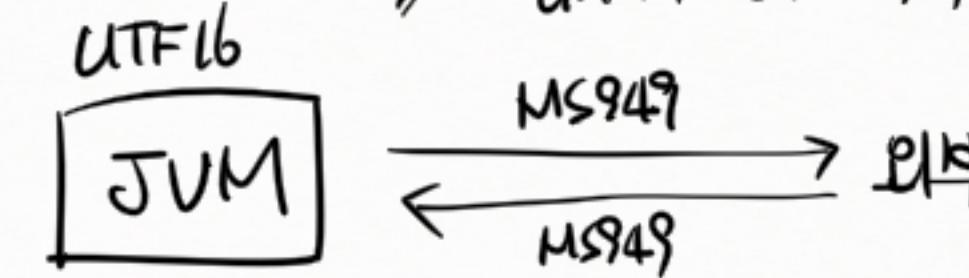
1 byte	2 byte Unicode
48	0048 (H)
65	0065 (e)
6c	006c (l)
6c	006c (l)
6f	006f (o)

* JVM 와 UTF16

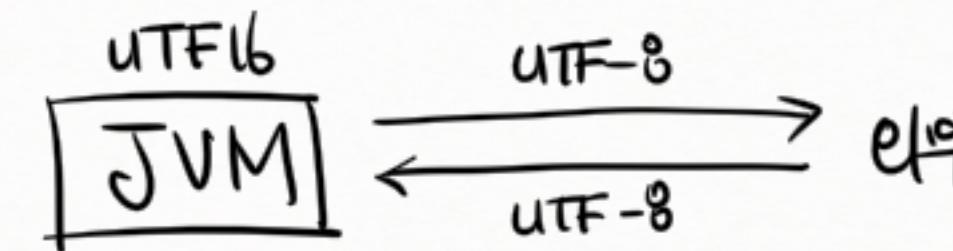
default characterset

↳ JVM이 외부의 문자코드를 내부문자로 변환할 때 사용하는 charset-
기본값은 charsets (문자변환기체)

Windows \Rightarrow

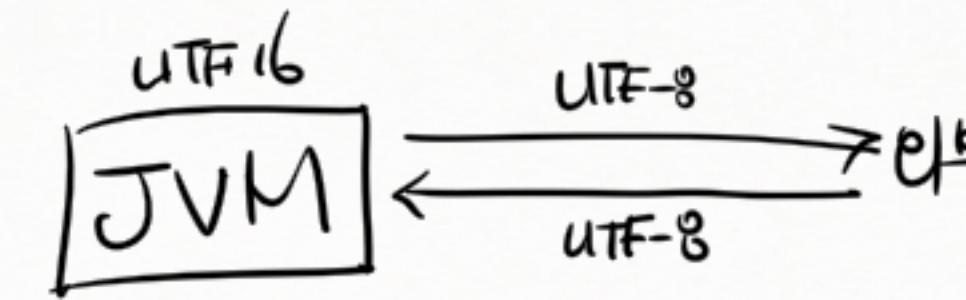


Unix/Linux \Rightarrow



Eclipse에서 \Rightarrow

App. 실행



* String 클래스와 생성자: oop.ex04.Exam0112

실행環境: Eclipse环境下

JVM이 외부 문자코드를 읽을 때
UTF-8이라 가정한다

String s

200

= new String();

200 value ...
IP 00bo 00a1 00b0 00a2 00b6 00ca 00b6 00cb
char[]

byte[]
'가' '나' '들' '을'의 EUC-KR 코드
문자

문자	EUC-KR
가	B0A1
나	B0A2
들	B6CA
을	B6CB

String 클래스
내부로 배열에 들어가는 문자코드가 UTF-8 이라고
assumes.

각각한다.

↓ 문제 발생!

각각의 Unicode로 변환

한글이 깨진다.

System.out.println(s);

'가'
→ B0A1 (EUC-KR)
→ ACOO (UTF-16)
→ EAB0D0 (UTF-8)

* String 클래스와 생성자: oop.ex04.Exam0112

실행環境: Eclipse SDK 실행

JVM이 외부 문자코드를 읽을 때
UTF-8이라 가정한다

String s

200

= new String(, "EUC-KR");

200 value
1200 ...

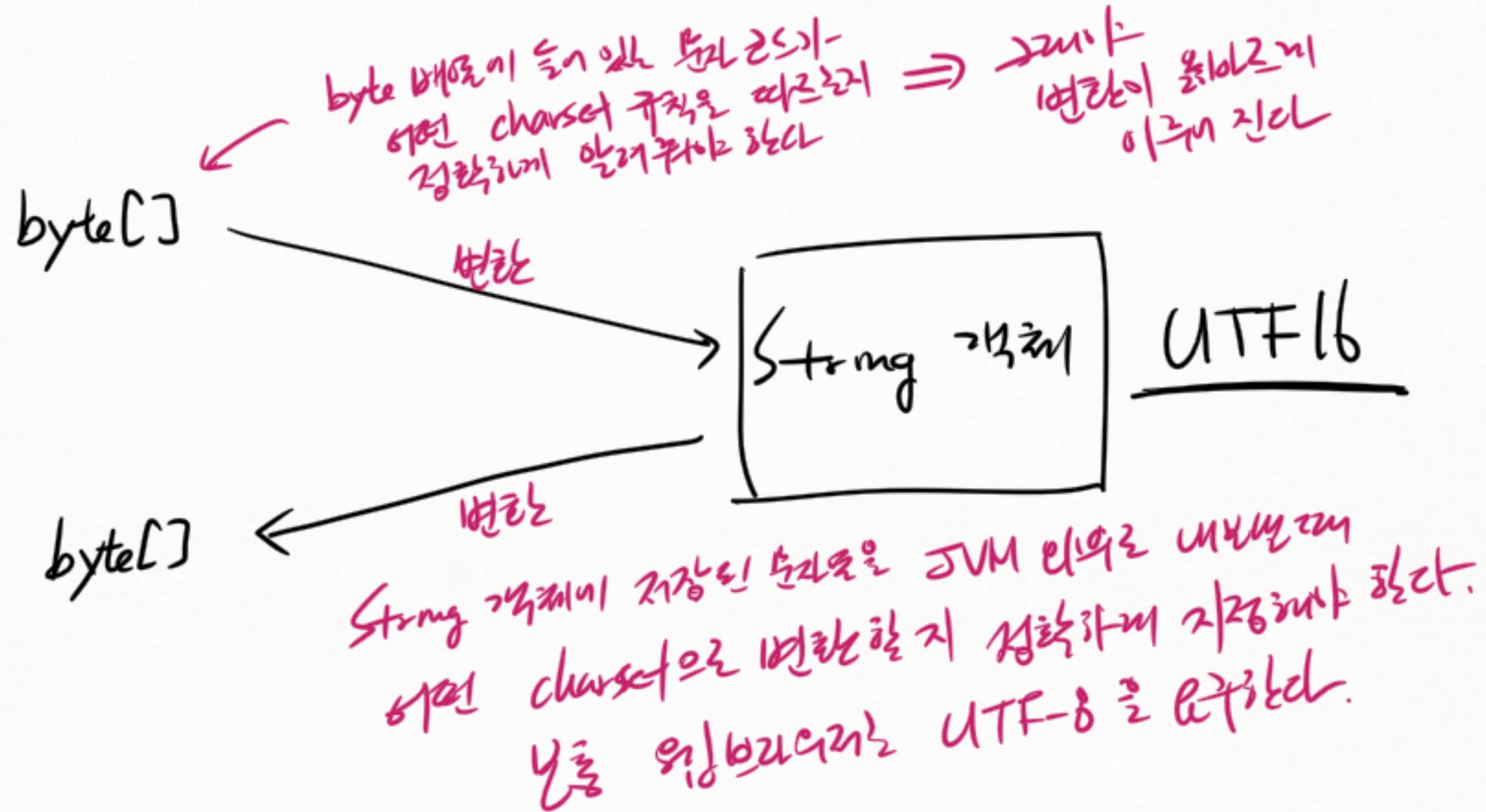
1200
AC00 AC01 B618 B625
char[]

byte[]
b0 a1 b0 a2 b6 ca bc cb
'가족동행'의 EUC-KR 코드
문자코드

byte[] 빼면 이들이 있는 값이
이전 문자집합의 규칙을 따르는지
정확하게 지정된다면
UTF16으로 정상적으로
변환하는 거다

{ EUC-KR \Rightarrow b0 a1 b0 a2 b6 ca bc cb }
UTF16 \Rightarrow AC00 AC01 B618 B625 }

* Java 한글 처리 주의!



① -Dfile.encoding 퀘션의 답변

Windows : MS-949 ✓

Unix/Linux : UTF-8

byte[]

EUC-KR

b0	a1	b0	a2	b6	ca	b6	cb
----	----	----	----	----	----	----	----

"ABC가Charlotte"



EUC-KR, UTF-8, UTF-16
(한글 235자)

(한글 111자)

(한글 X)

현재 byte[] 배열에 저장되어 있는
데이터 문자집합은? EUC-K

UTF-16

'A' → 0041

'B' → 0042

'C' → 0043

'가' → AC00

'Charlotte' → AC01

String s = new String()



UTF-16

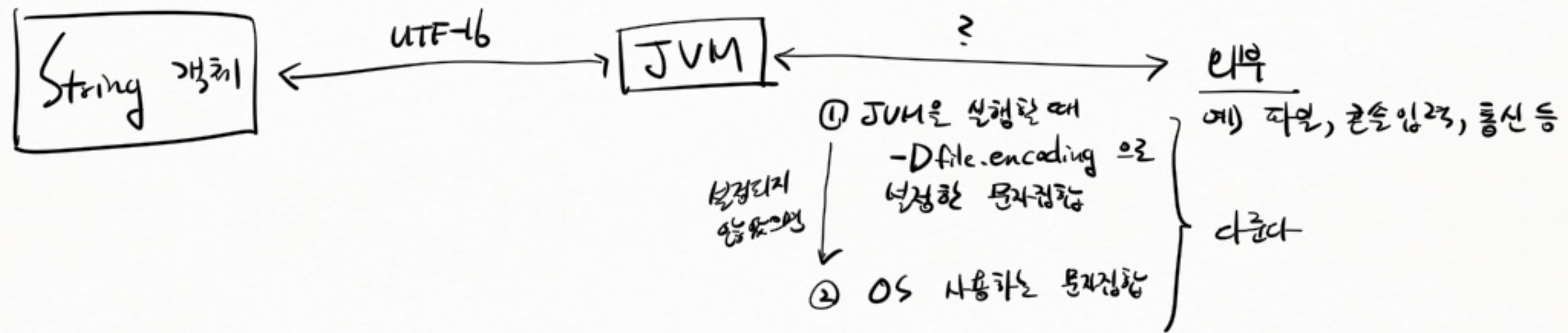
스팅 객체

char[]

--	--	--	--	--	--	--

JVM이 인부에서 문자데이터를 읽거나.
인부로 문자데이터 보낼 때
사용하는 문자변환기체
character set
(문자집합)

* JVM 내 문자 진화법



~~-Dfile.encoding=UTF-8~~ 복정 오류

java -cp bin/main com —

Windows
(MS949)

[MS949 → UTF16] ↗ 사용

byte[] $\xrightarrow{\text{영어}} \text{new String()}$

영어 EUC-KR $\longrightarrow \text{O}$

한글 UTF-8 $\longrightarrow \text{X}$

Unix/Linux
(UTF-8)

[UTF-8 → UTF16]

byte[] $\xrightarrow{\text{한글}} \text{new String()}$

EUC-KR $\longrightarrow \text{X}$

UTF-8 $\longrightarrow \text{O}$

-Dfile.encoding=UTF-8 복정 오류

java -Dfile.encoding=UTF-8 -cp bin/main —

Windows
(MS949)

byte[] $\longrightarrow \text{new String()}$

EUC-KR $\longrightarrow \text{X}$

UTF-8 $\longrightarrow \text{O}$

Unix/Linux
(UTF-8)

byte[] $\longrightarrow \text{new String()}$

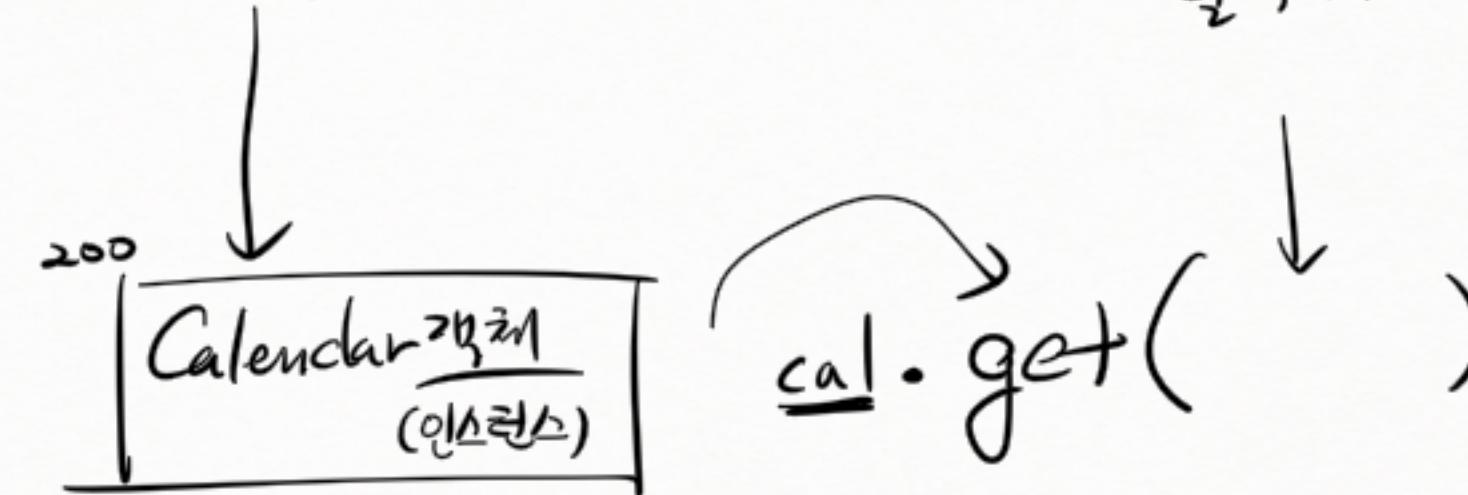
EUC-KR $\longrightarrow \text{X}$

UTF-8 $\longrightarrow \text{O}$

* Calendar 사용법

Calendar cal = Calendar.getInstance()

200



cal.get(1) → 년 을 일 시:분:초:밀리초

cal.get(Calendar.YEAR) → ①년도

값에의 아이디를 얻기위해 카운트

그러나 소수점 뒷수에 미리 그 번호를 저장해 두고자.

숫자는 문자이기 때문에 이해하기 수월.

* String : compareTo()

s1.compareTo(s2)
 ↙
 this

Hello;
Hello ← s1
Hello

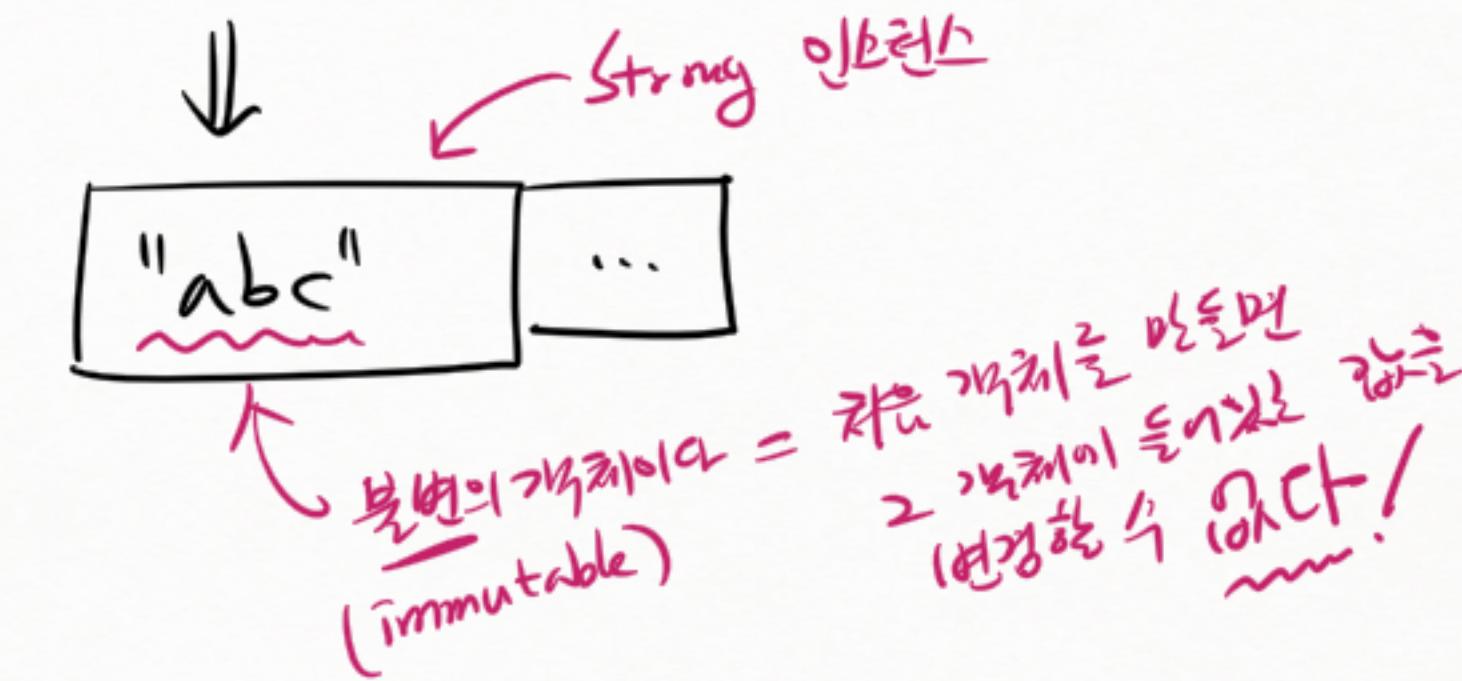
A



z

* Immutable 객체와 Strong

new String("abc")



* primitive type 2 wrapper 풀이

int → java.lang.Integer

byte → " .Byte

short → " .Short

long → " .Long

float → " .Float

double → " .Double

boolean → " .Boolean

char → " .Character

(
자료형 대체 가능
자동으로 만들 수 있음)
Wrapper

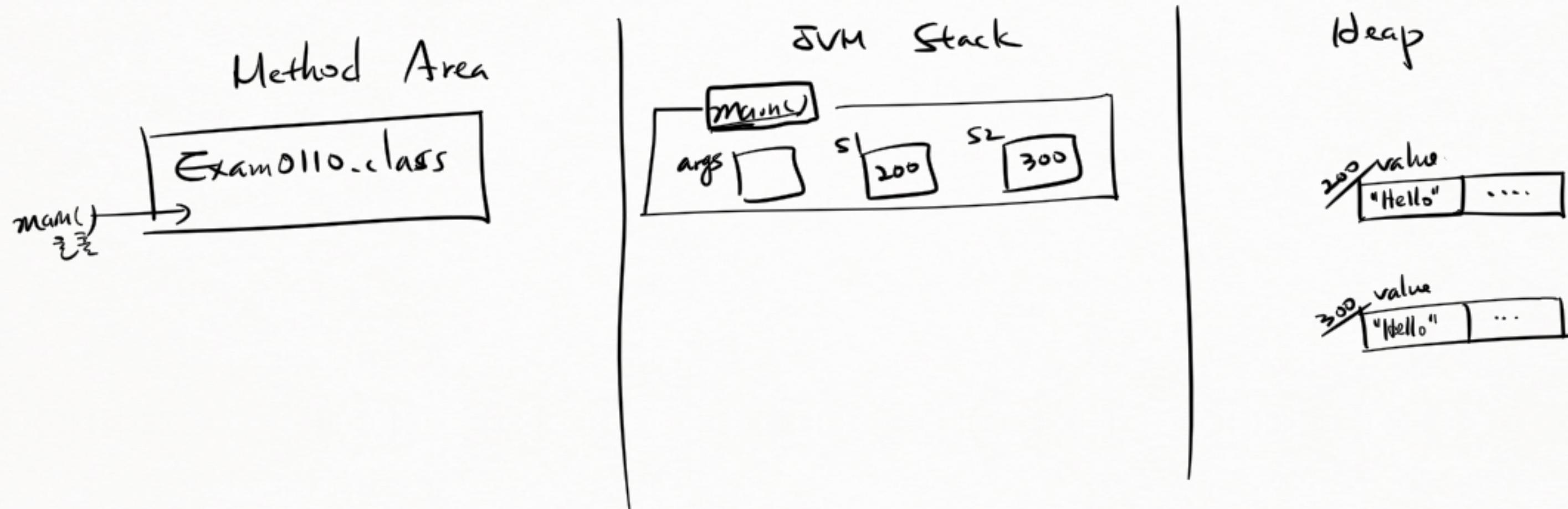
int → Integer

10

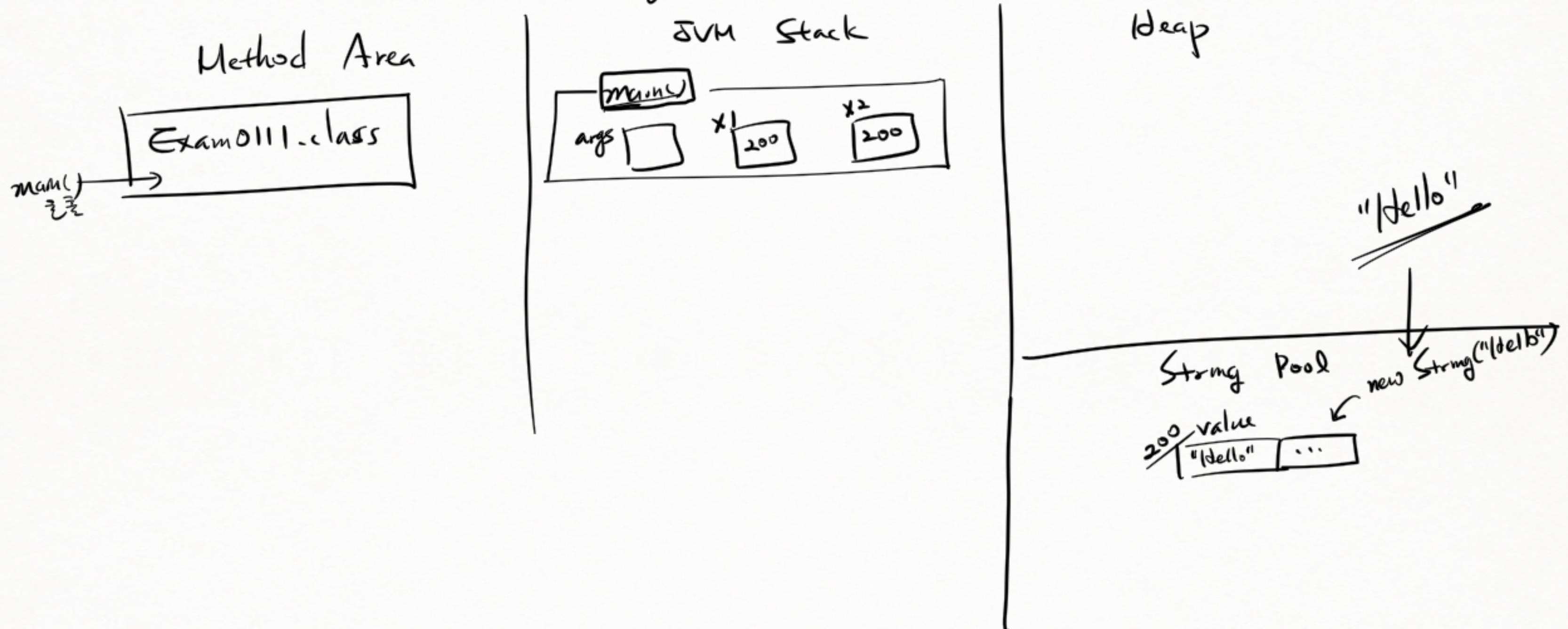
value
10 ...

인스턴스 변수가 있는 것임

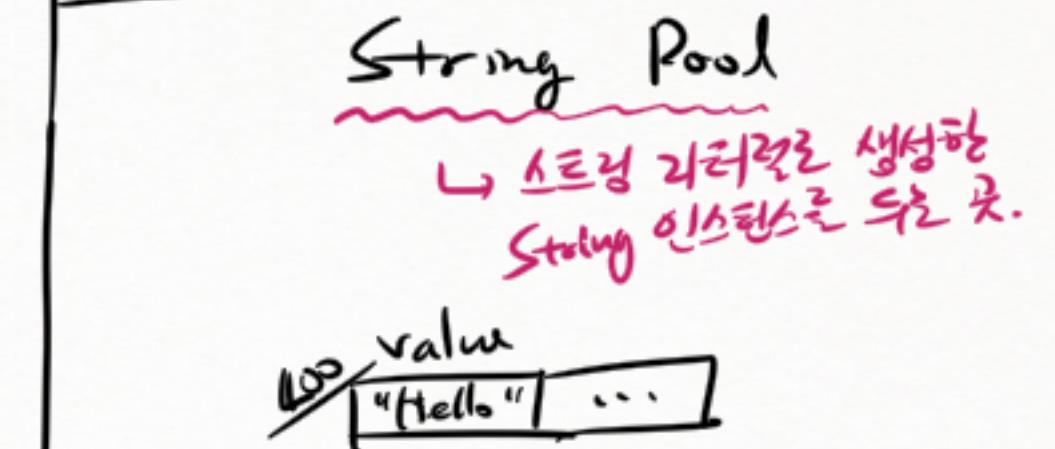
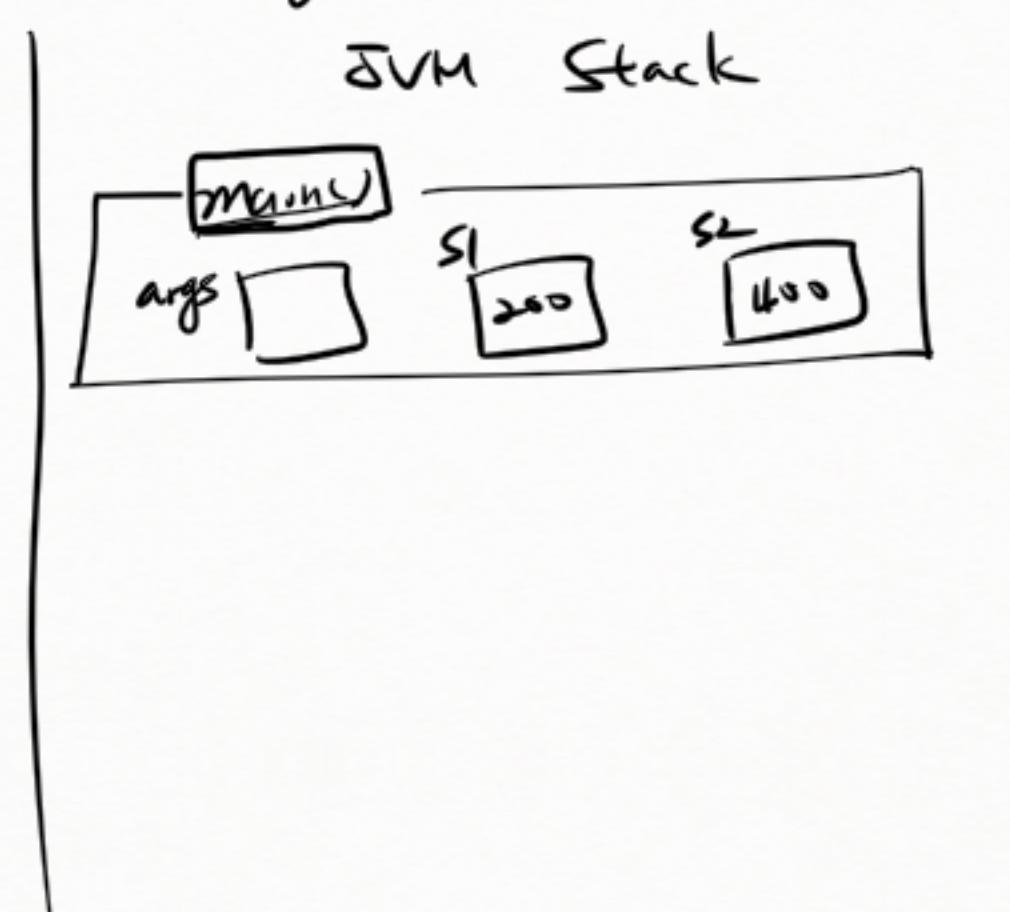
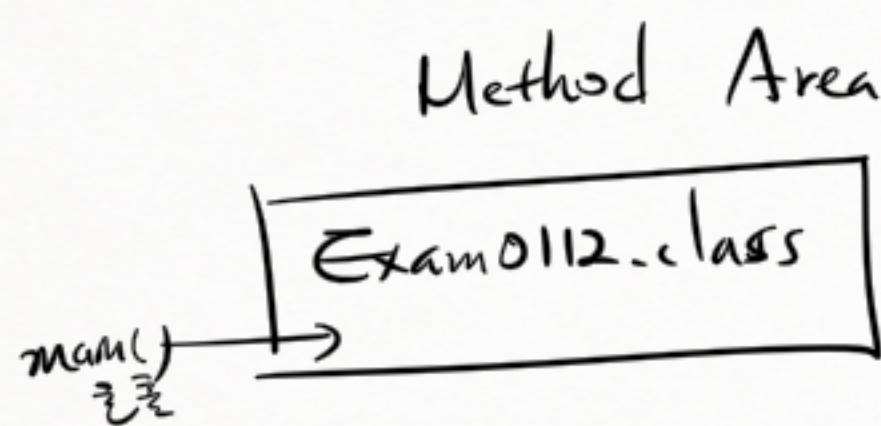
* lang. ex02 - Exam0110 - String 之 m< n & 之



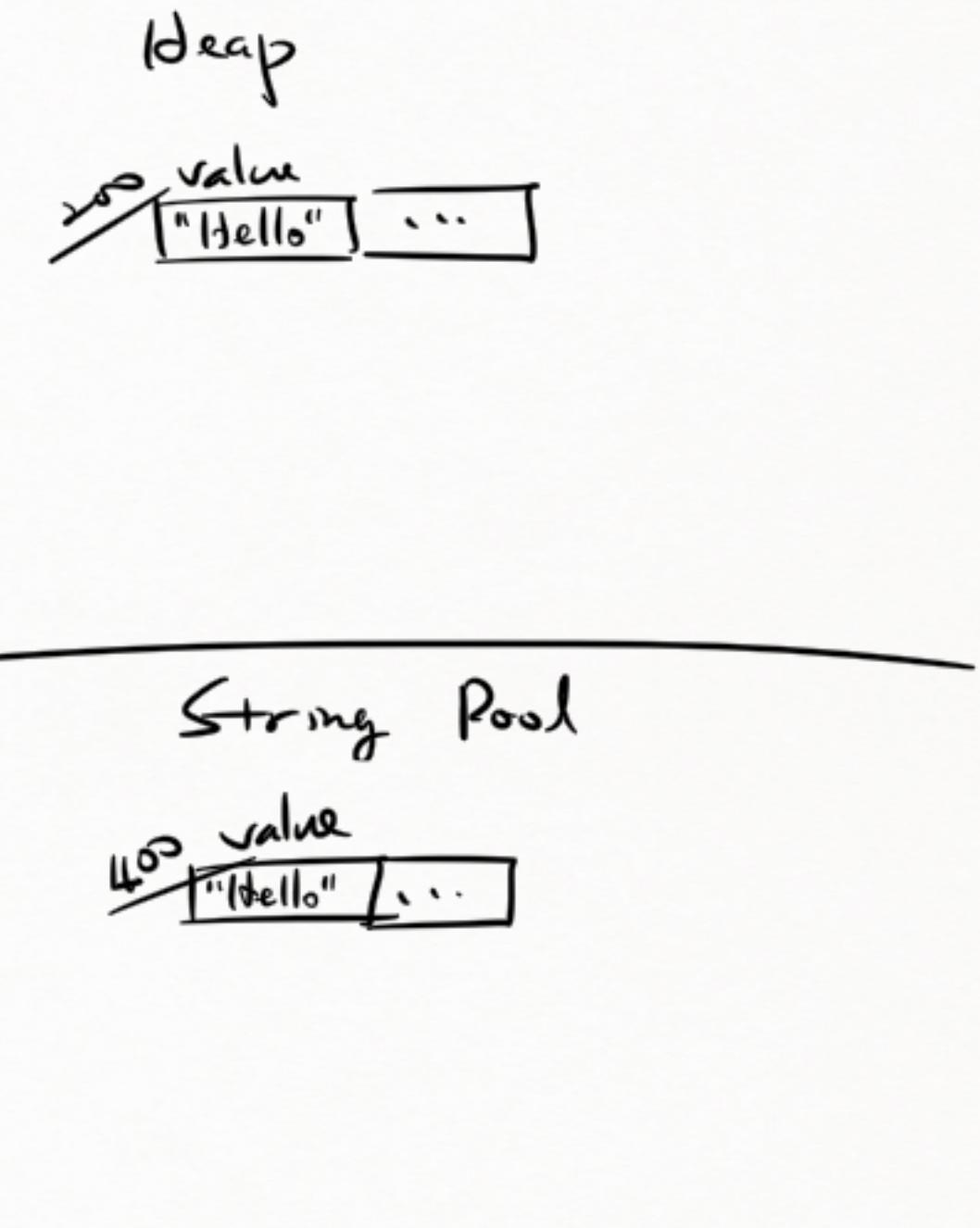
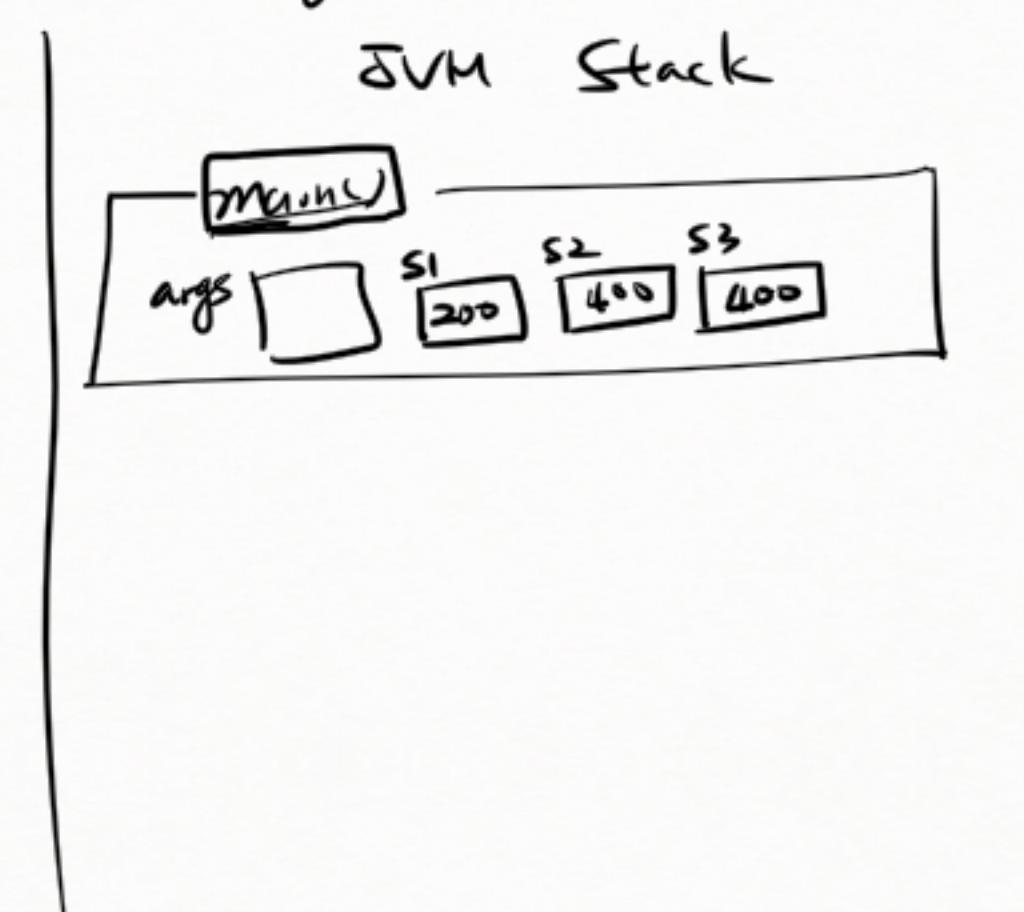
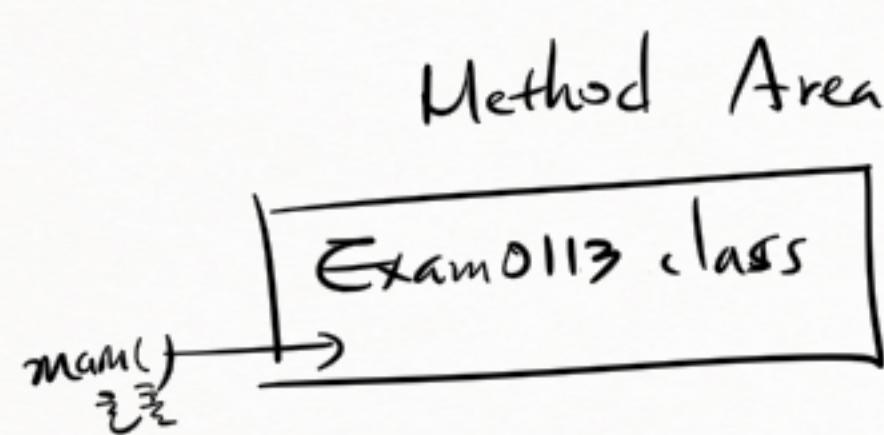
* String 은 mt. No법 - 스트링 리터럴로 인스턴스 생성하기
lang.ex02.Exam0111



* String 초기화 방법 - new vs "—"
lang.ex02.Exam0112

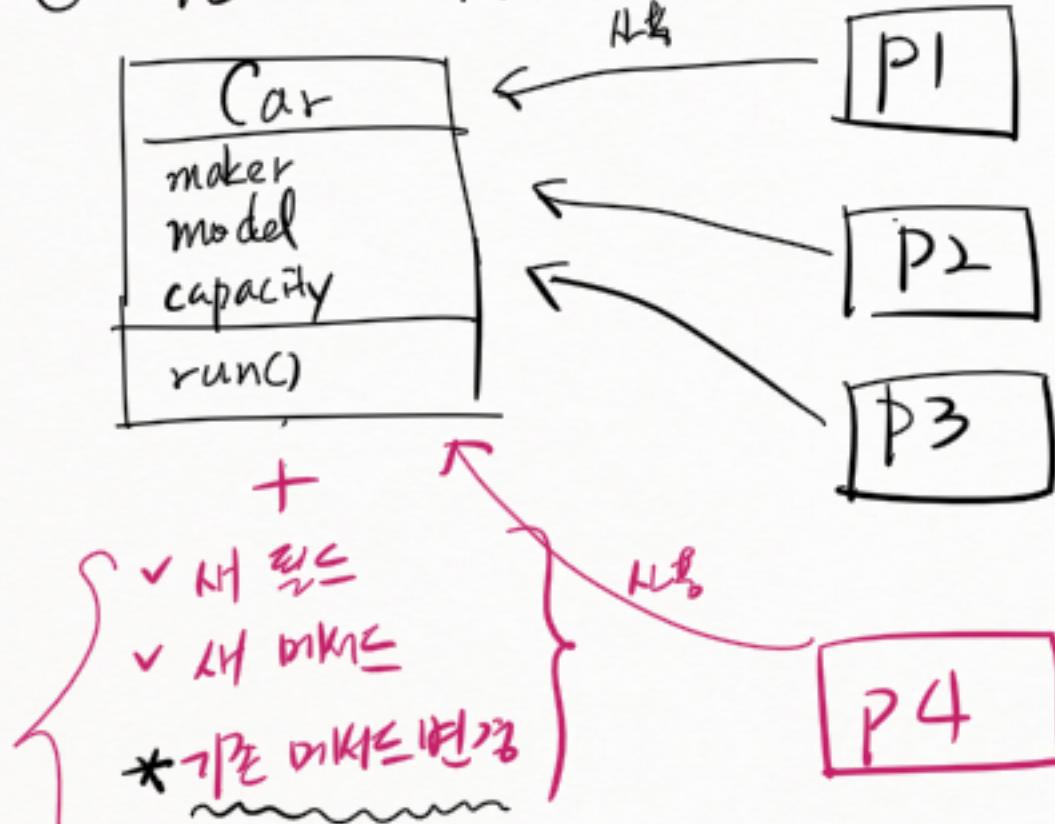


* String အဲမှု မှုပါး - intern()
lang.ex02.Exam0113

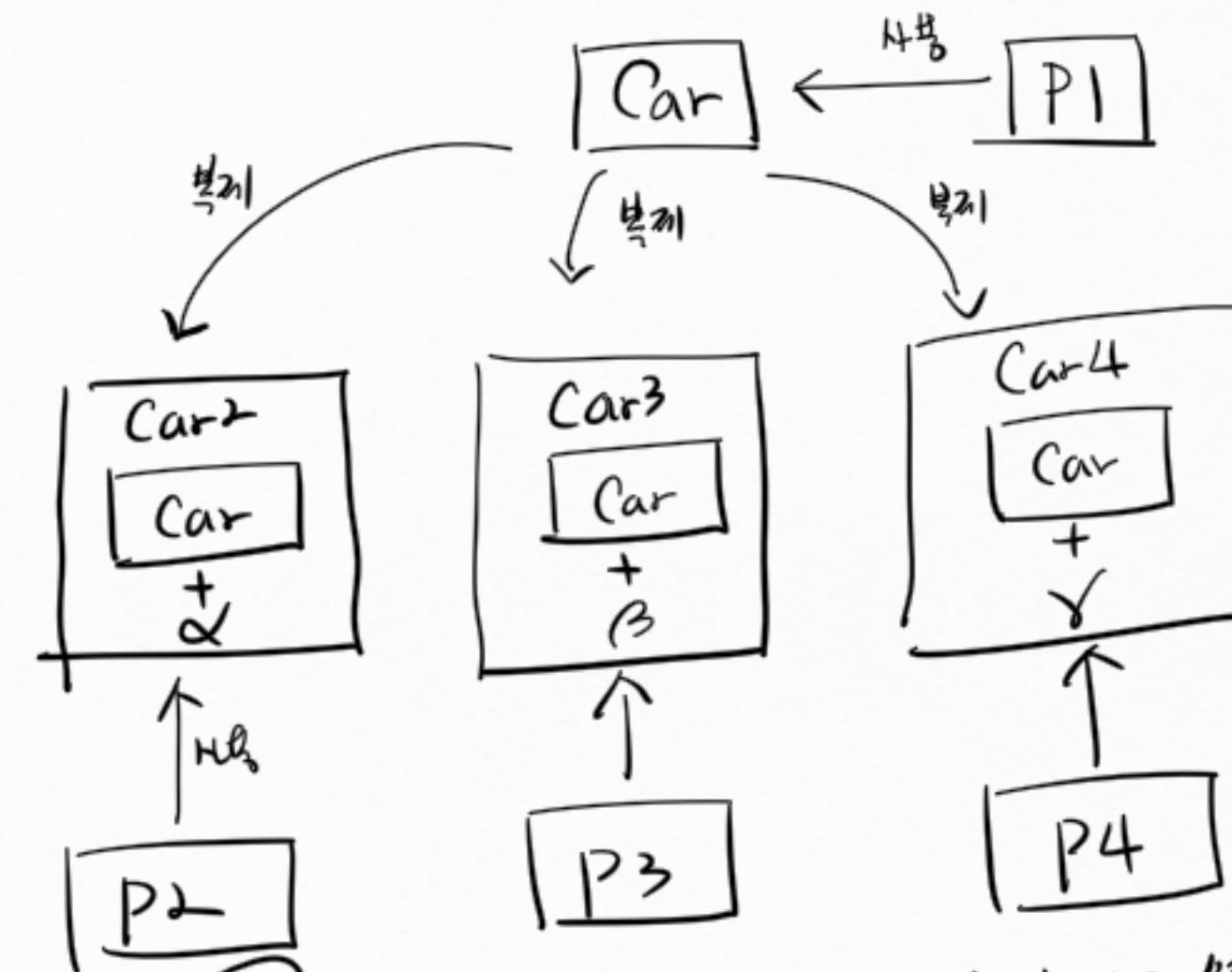


* 상속 - 기능 확장은 위한 방법 (oop.ex05)

① 기존 클래스에 새 기능 코드를 추가하는 방법



② 기존 코드를 복제하여 새 기능을 추가하는 방법



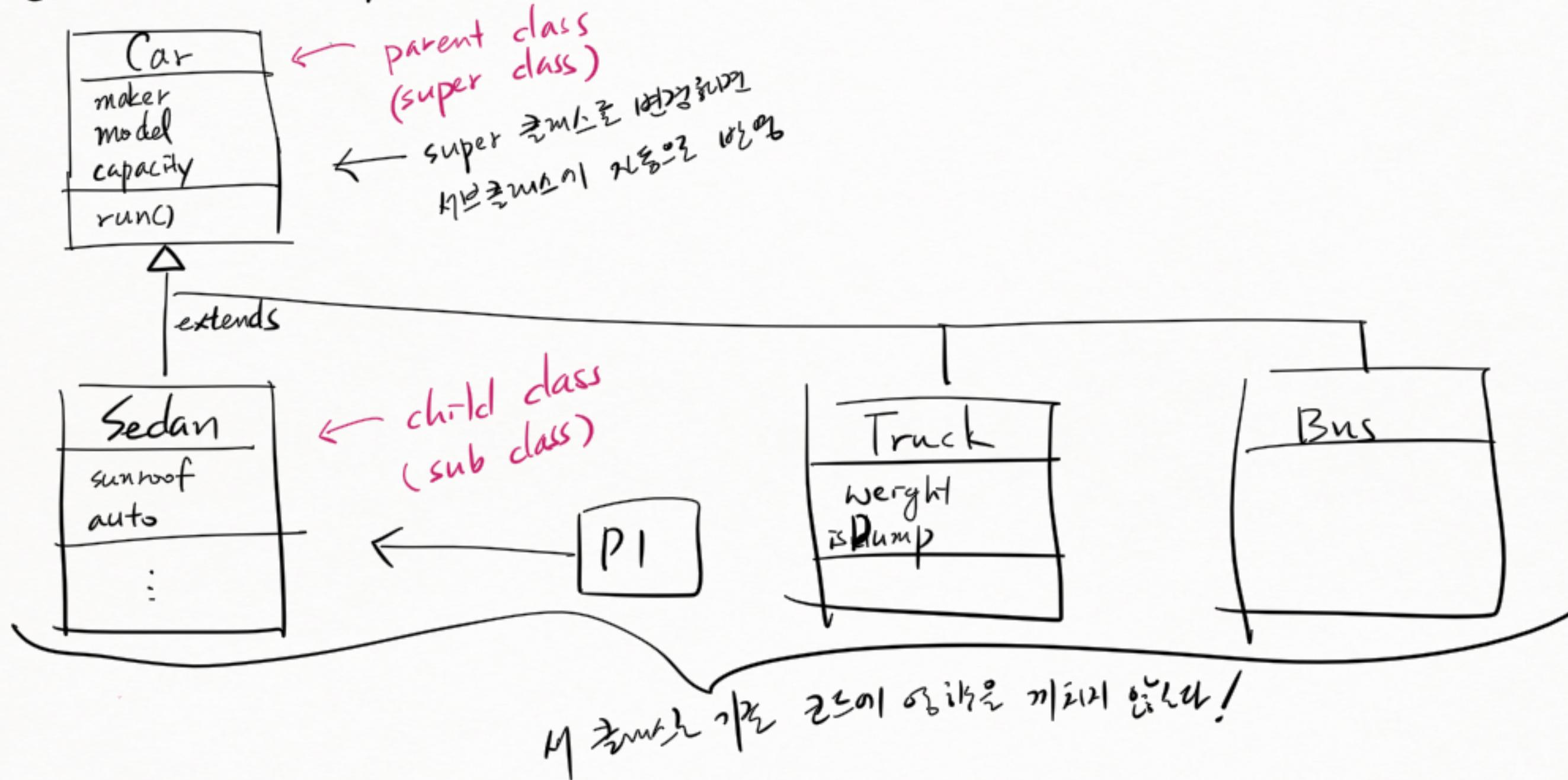
장점:
기존 코드를 손대지 않음
↓
기존 프로젝트에 영향을 미치지 않음

원본 버전 수정 → 모든 복사본 버전 수정
원본 기능변경 → "

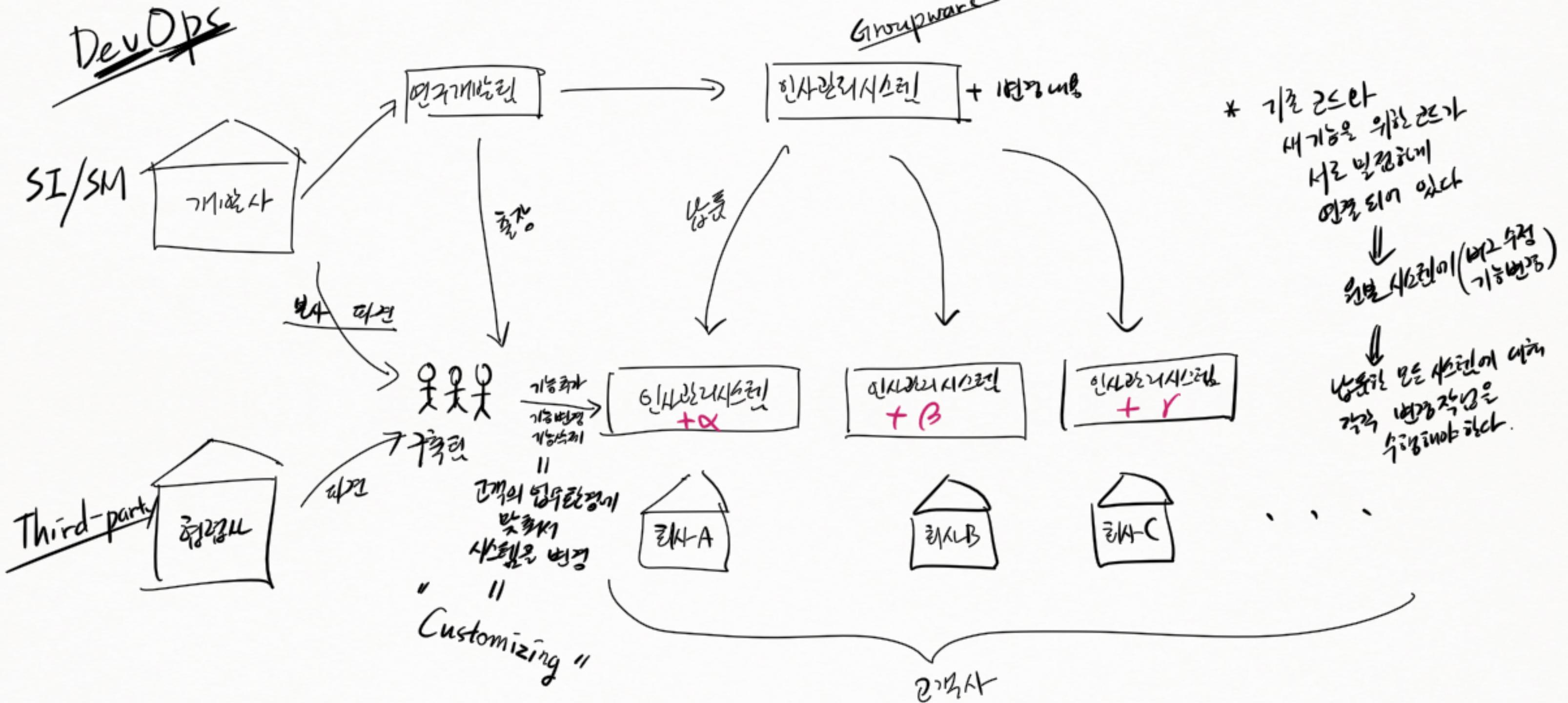
단점:

* 상속 - 기능 확장은 위한 기법 (oop.ex05)

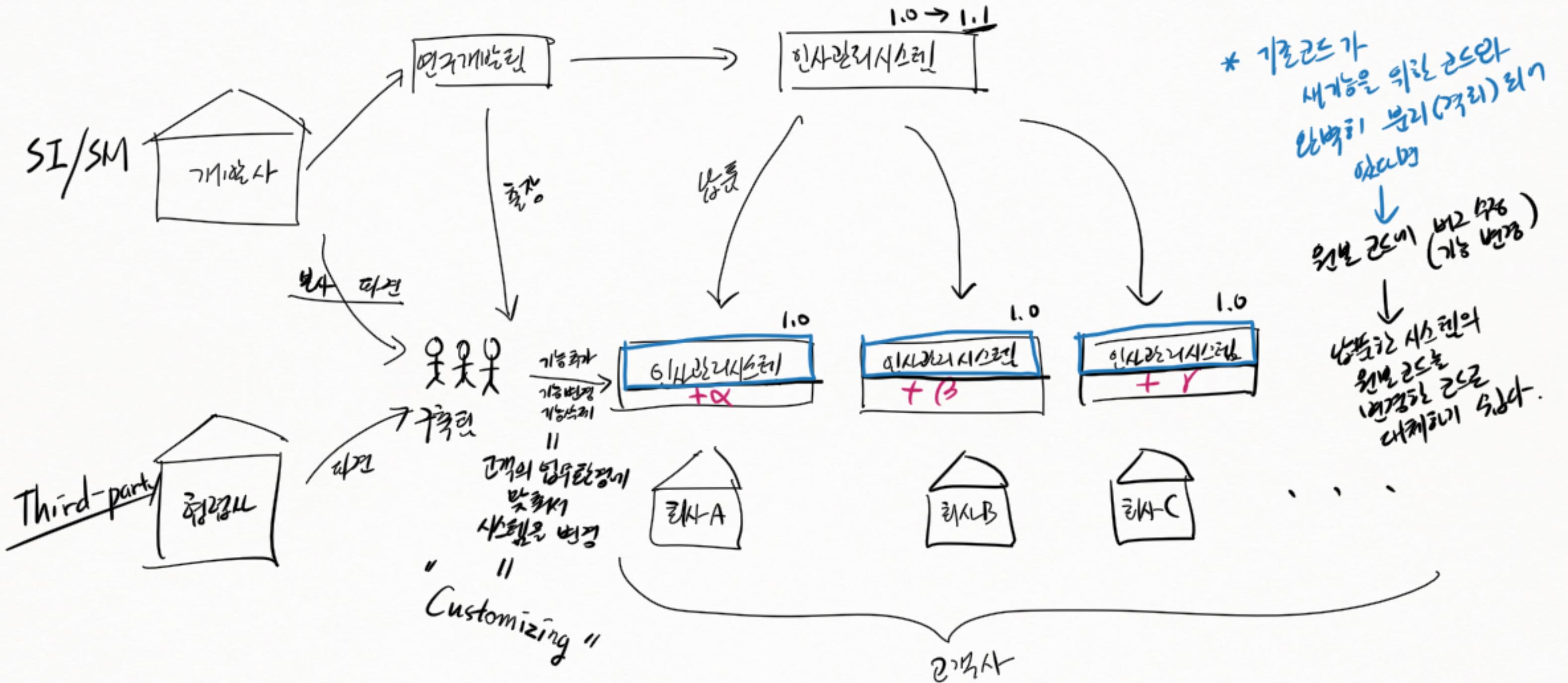
③ 상속을 이용한 기능 확장



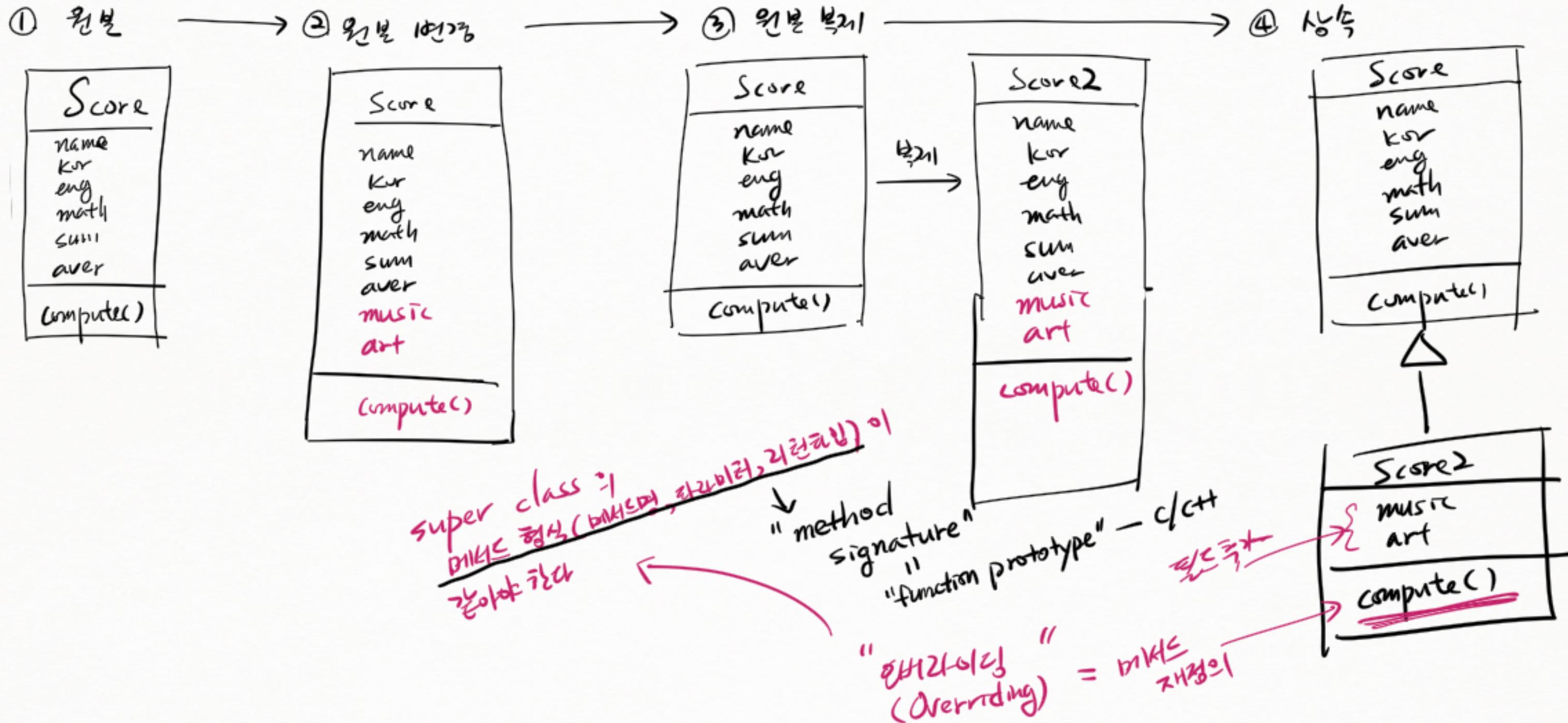
* 시스템화 - 고객사 설문을 토대로 커스터마이징 수준



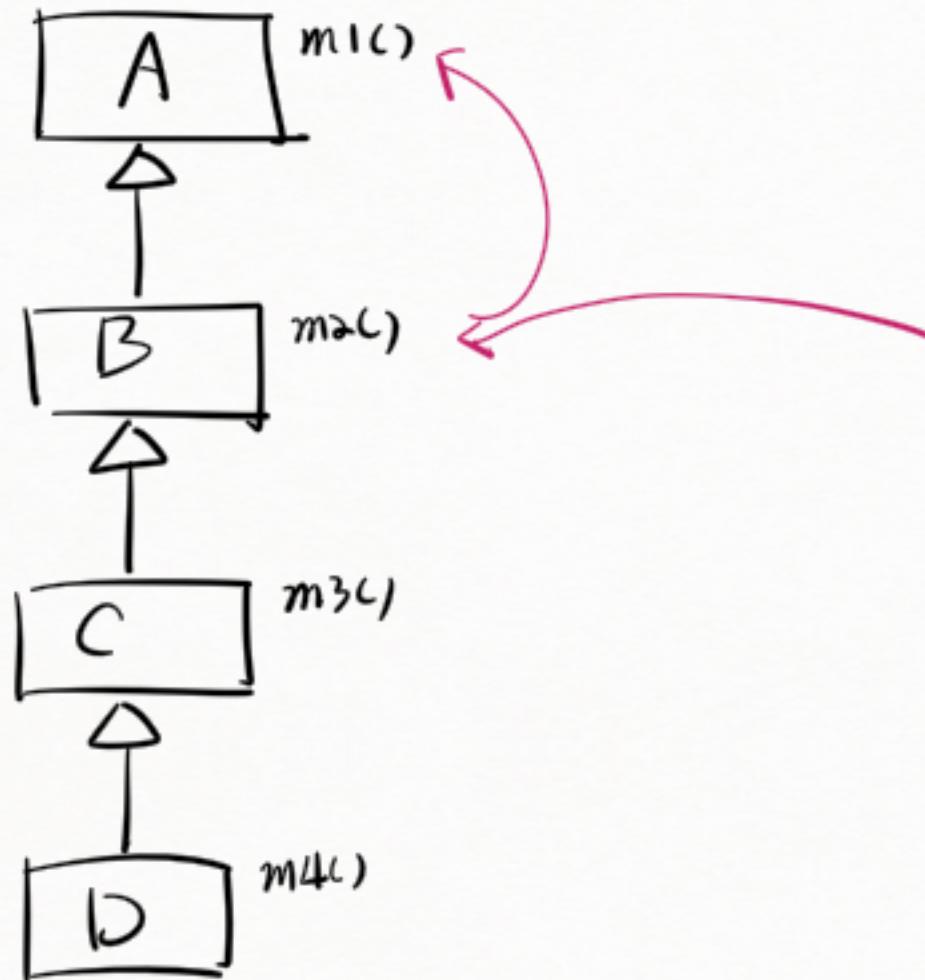
* 시스터즈 - 고객사별로 특화화 커스터마이징 가능



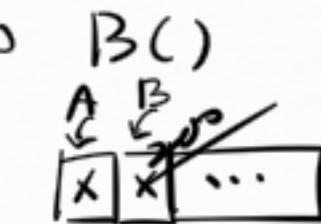
* 학습 예 - oop. ex05. a/b/c/d



* 상속 예 - oop. ex05. e

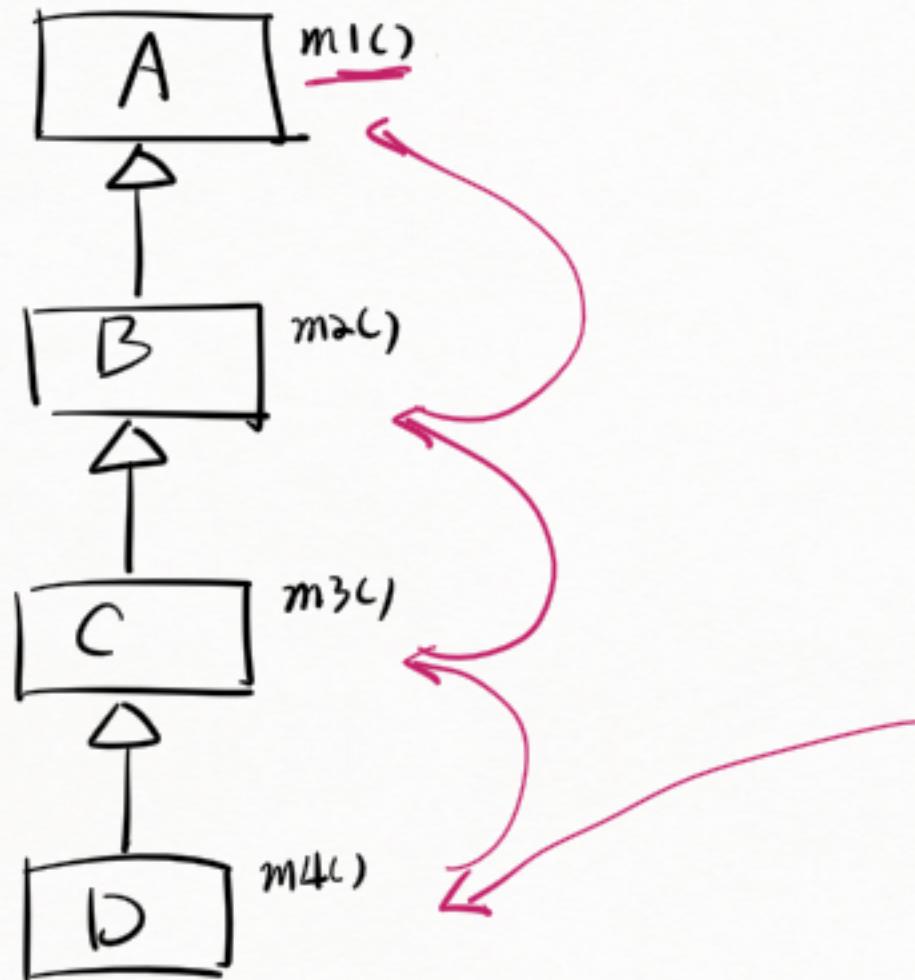


B obj = new B()
obj.m1();
obj.m2();



메서드 찾는 순서 \Rightarrow B \rightarrow A.m1() 훸
 ↓
 레퍼런스의 클래스부터 찾기 시작해서
 수퍼 클래스로 따라 올라가면서 찾는다.
 B.m2() 훸

* կենա - oop. exst. e



D obj = new D()

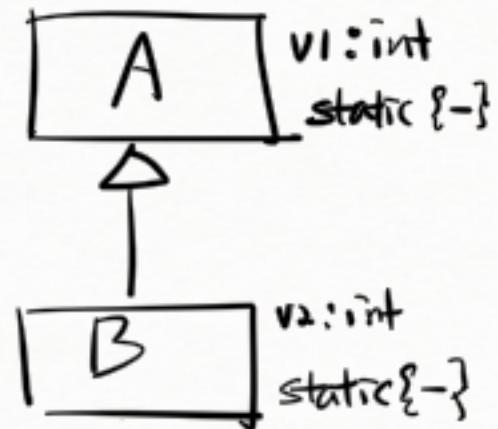
obj. m4(); D.m4() ↗

obj. m3(); D → C.m3() ↗

obj. m2(); D → C → B.m2() ↗

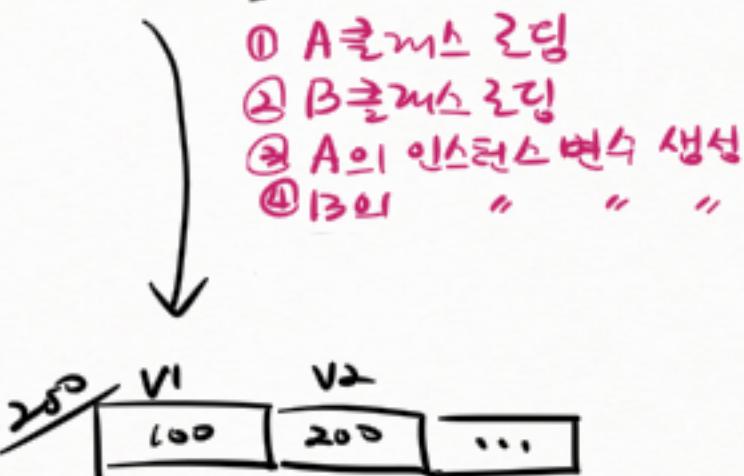
obj. m1(); D → C → B → A.m1() ↗

* 상속 예 - oop.ex05.f



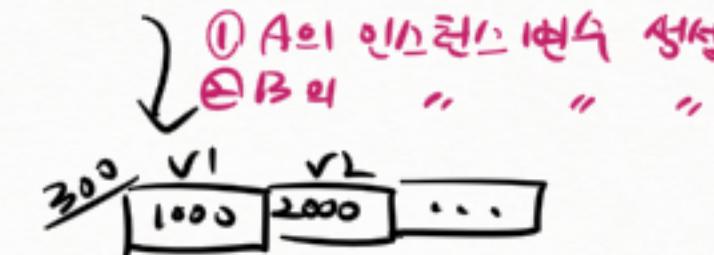
`B obj = new B();`

`obj`
[200]

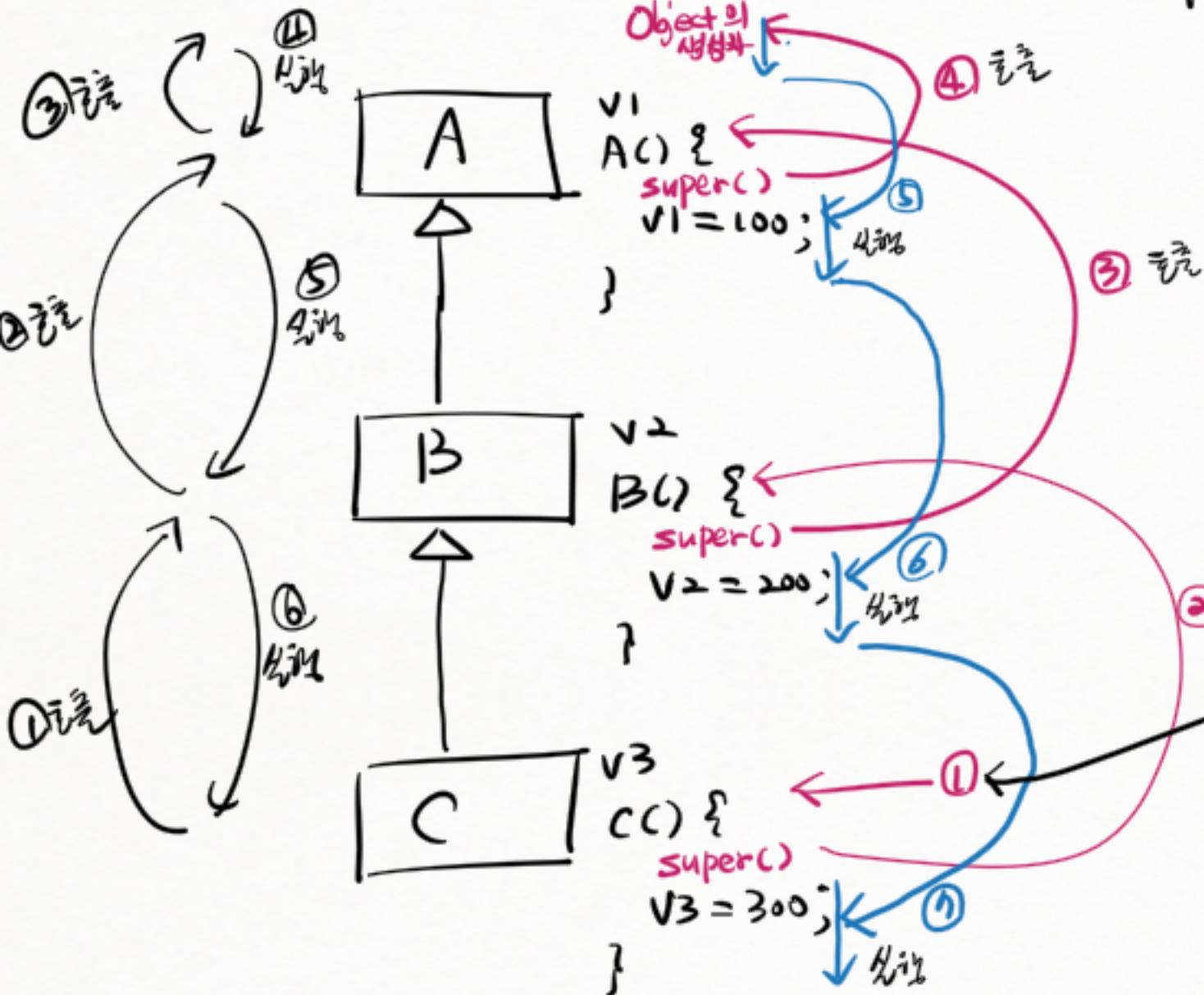


`B obj2 = new B();`

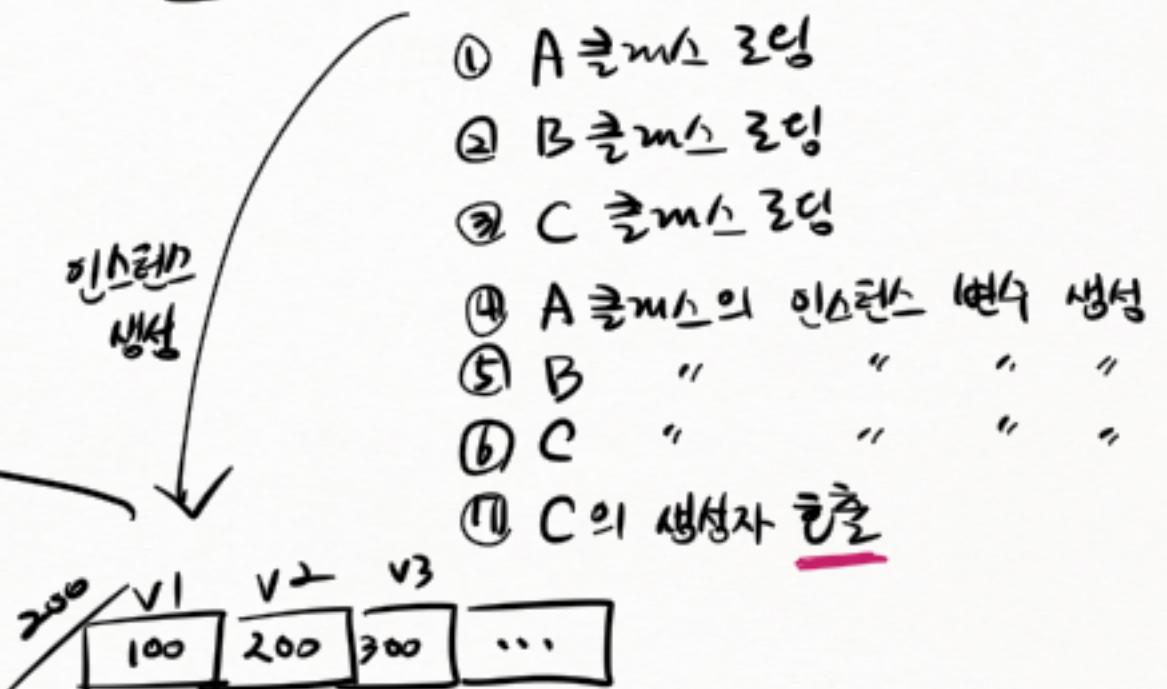
`obj2`
[300]



* 상속 예 - oop. ex05.g

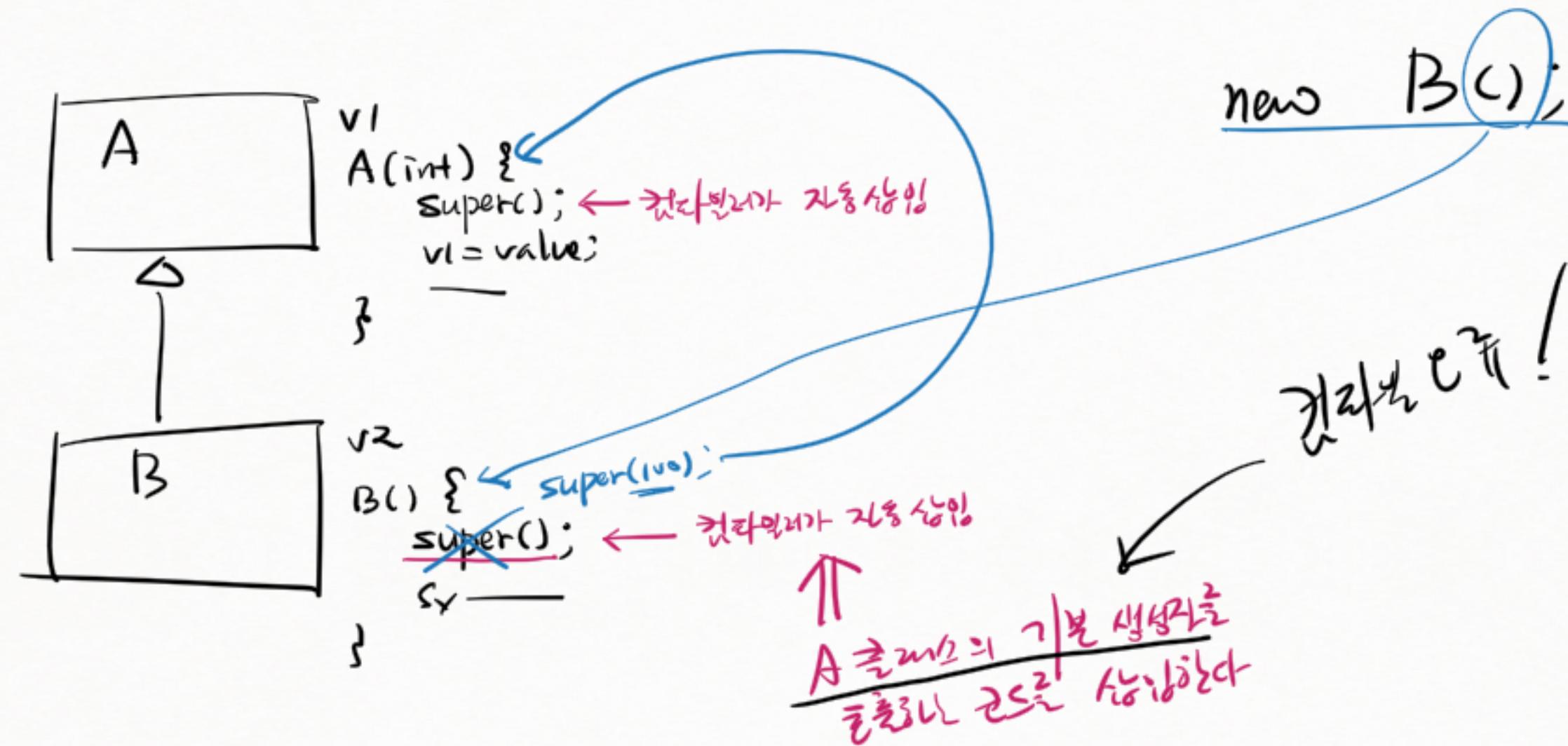


C obj = new C();



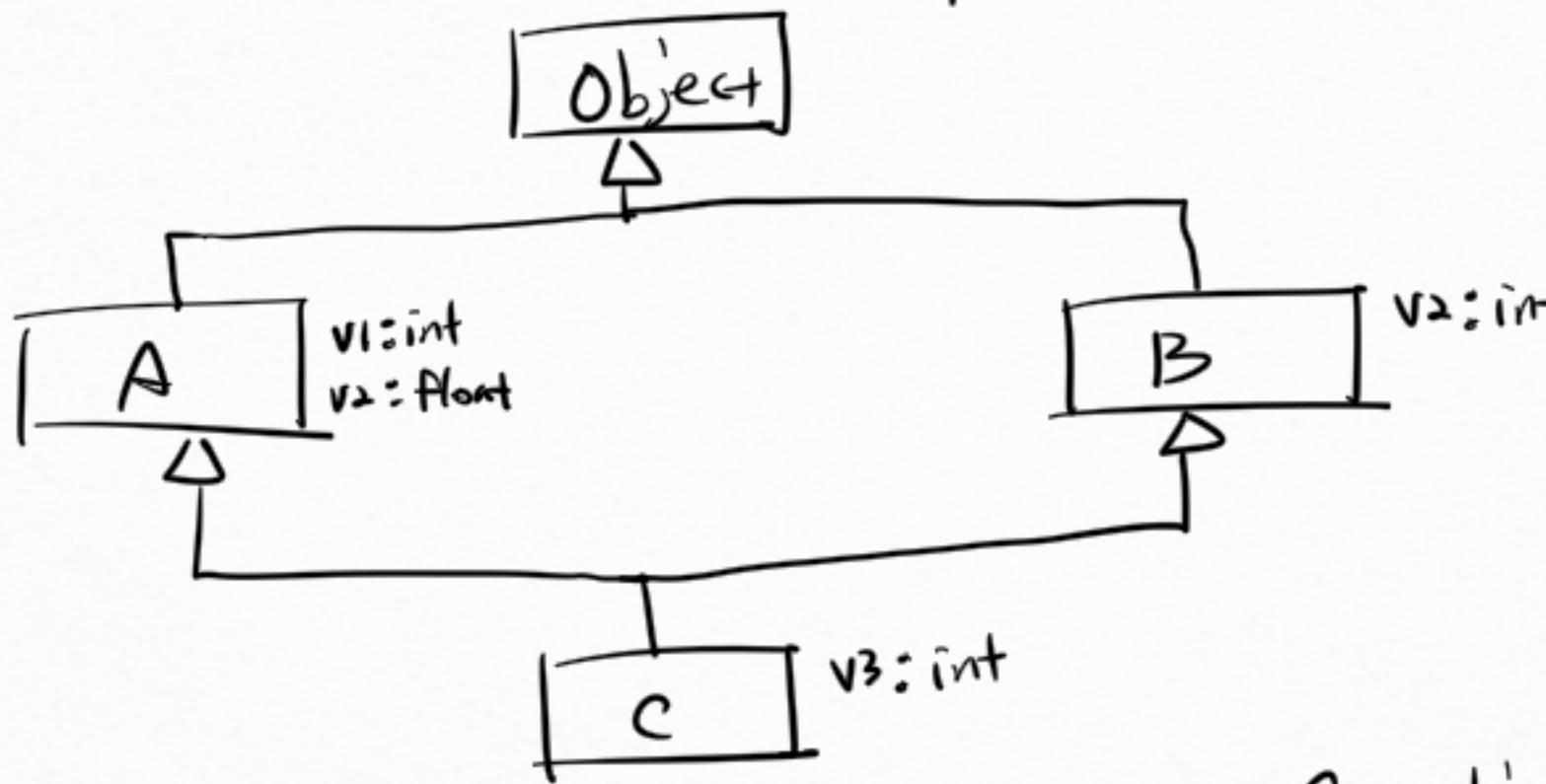
- ① A 클래스 로딩
- ② B 클래스 로딩
- ③ C 클래스 로딩
- ④ A 클래스의 인스턴스 변수 생성
- ⑤ B " "
- ⑥ C " "
- ⑦ C의 생성자 호출

* 상속과 슈퍼클래스 생성자 (oop-ex5.h)



* 다중 상속

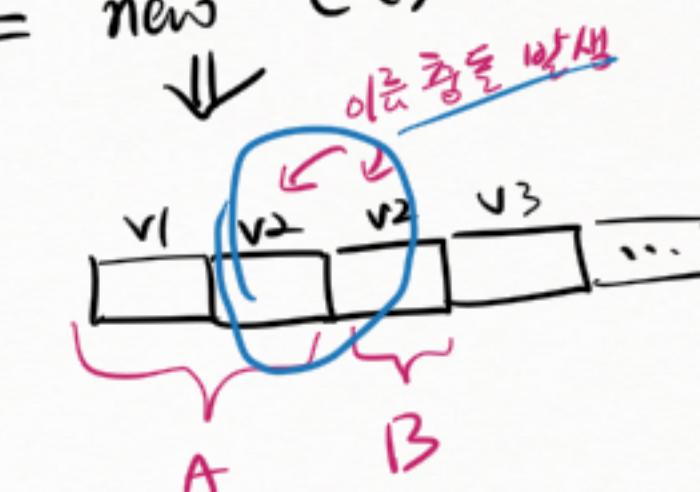
(oop. ex05. i)



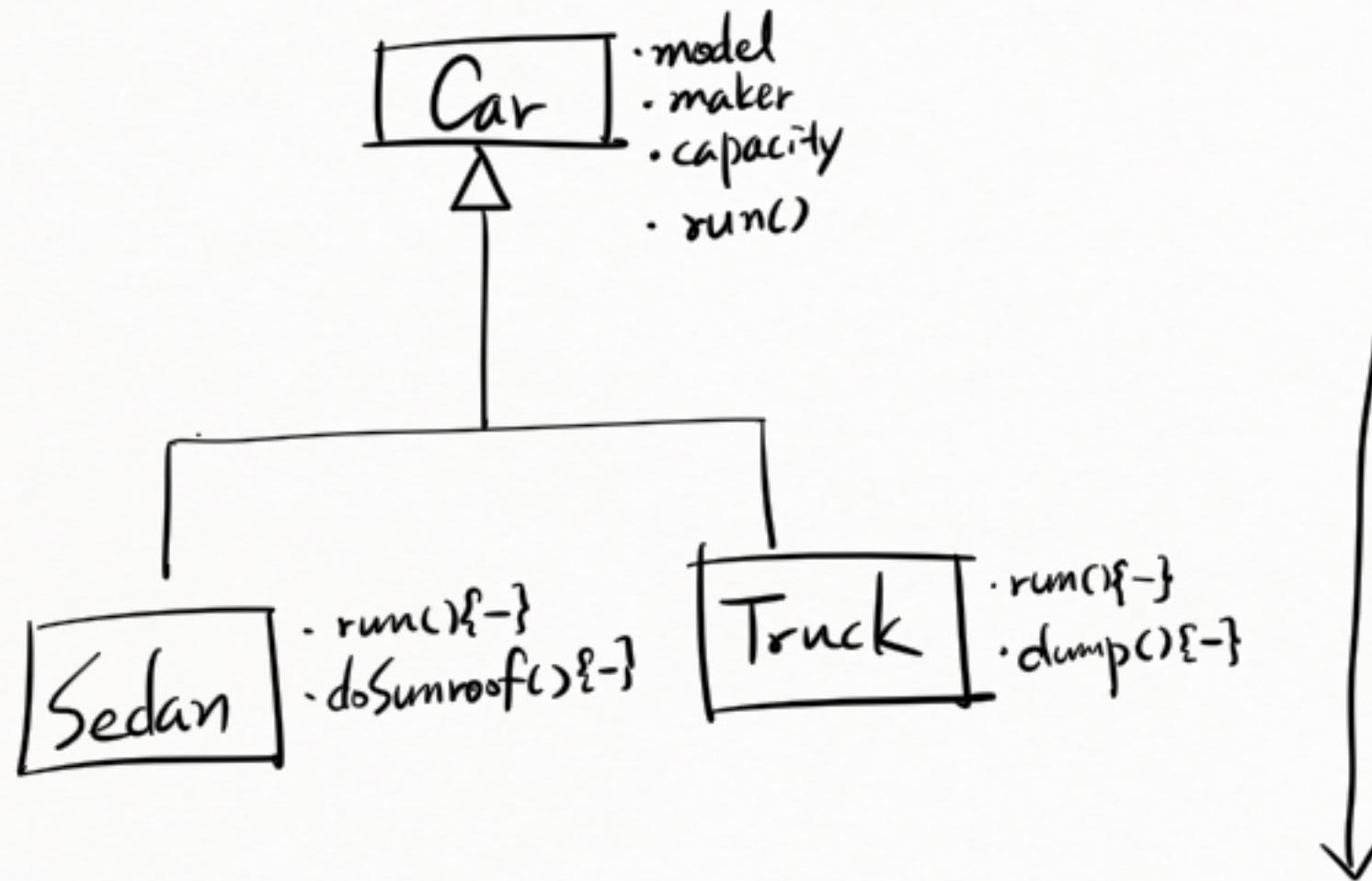
* 여러 클래스를 동시에 상속받아
변경시, 원래의 초기화를
이루어갈수록
어려워진다.

그러니 多重상속을 피하기 한다

`C obj = new C()`

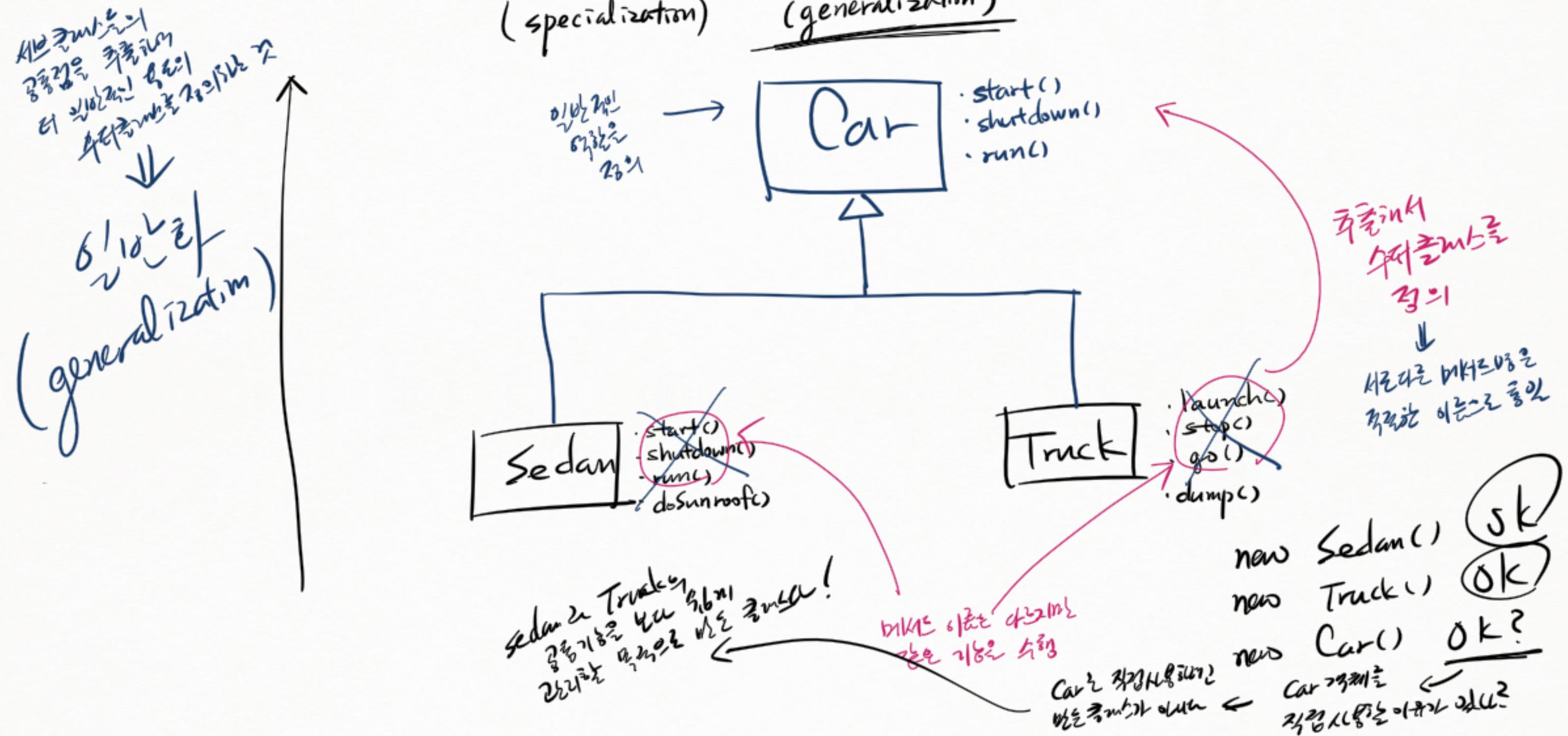


* 상속 : 전문화 와 일반화 (oop. ex05. j)
(specialization) (generalization)

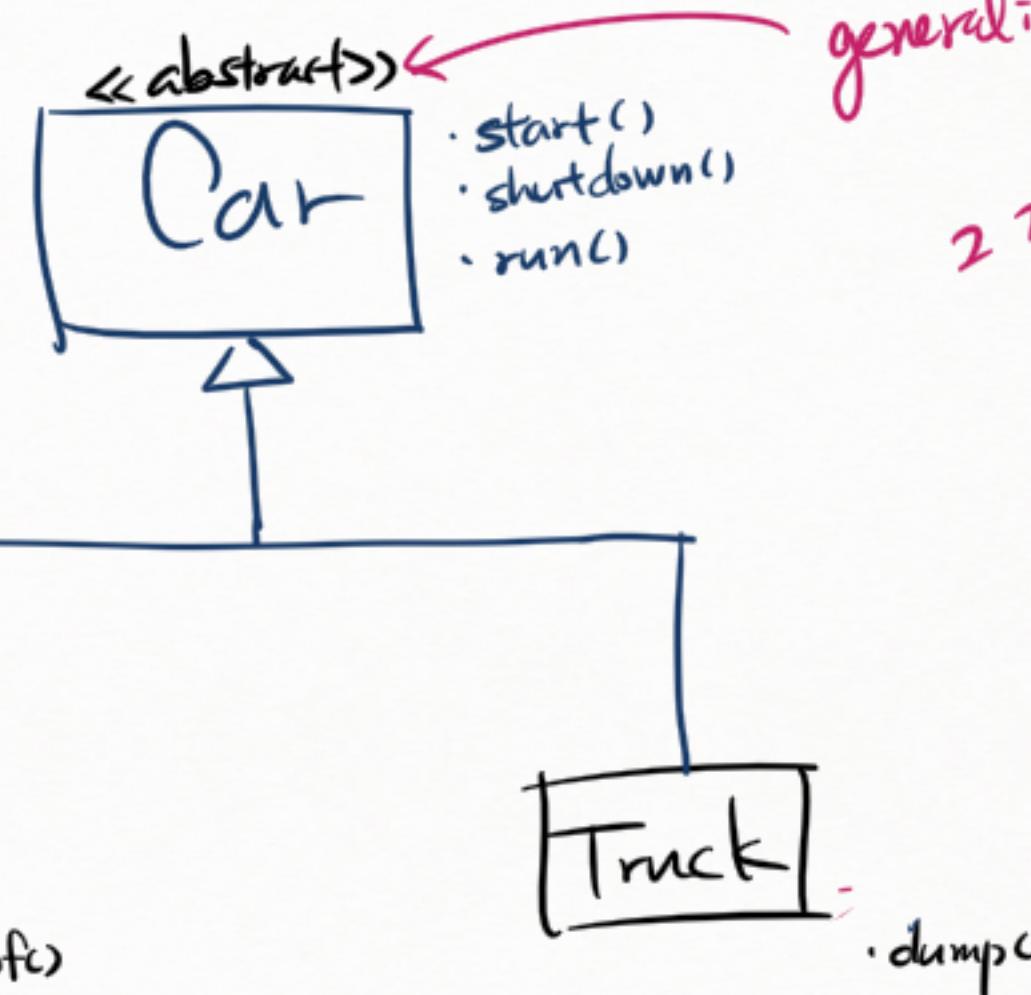
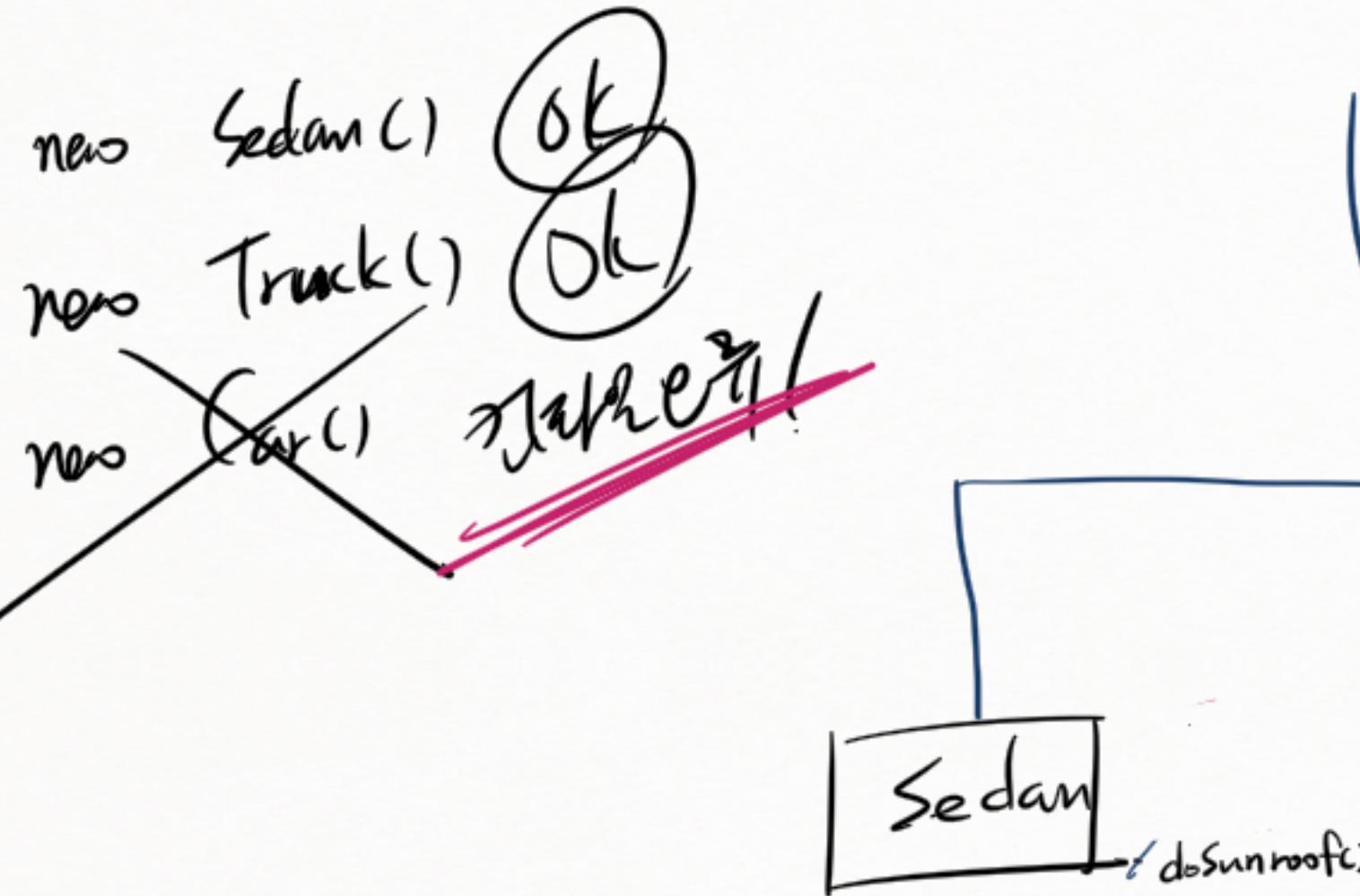


수퍼클래스를 상속 받을
기능은 상속된다
하지만 특수화된 예외는 가능!
아이언링크는 반드시
상속을 받을
"Specialization
(전문화)"

* 핵심 : 전문화와 일반화 (coop. ex05. k/l)
(specialization) (generalization)



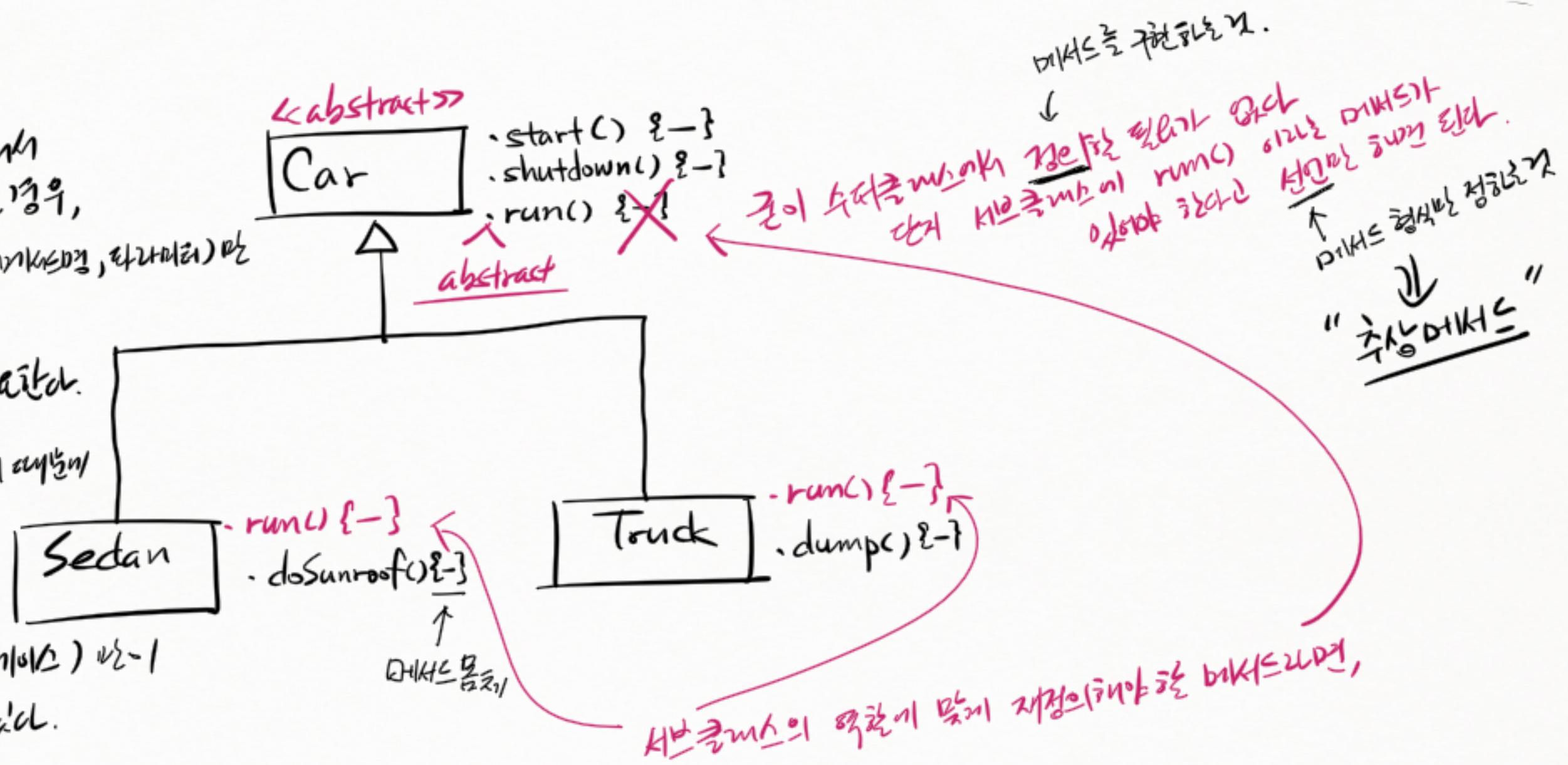
* 상속 : 일상화된 추상 클래스 (ex. ex05.m)



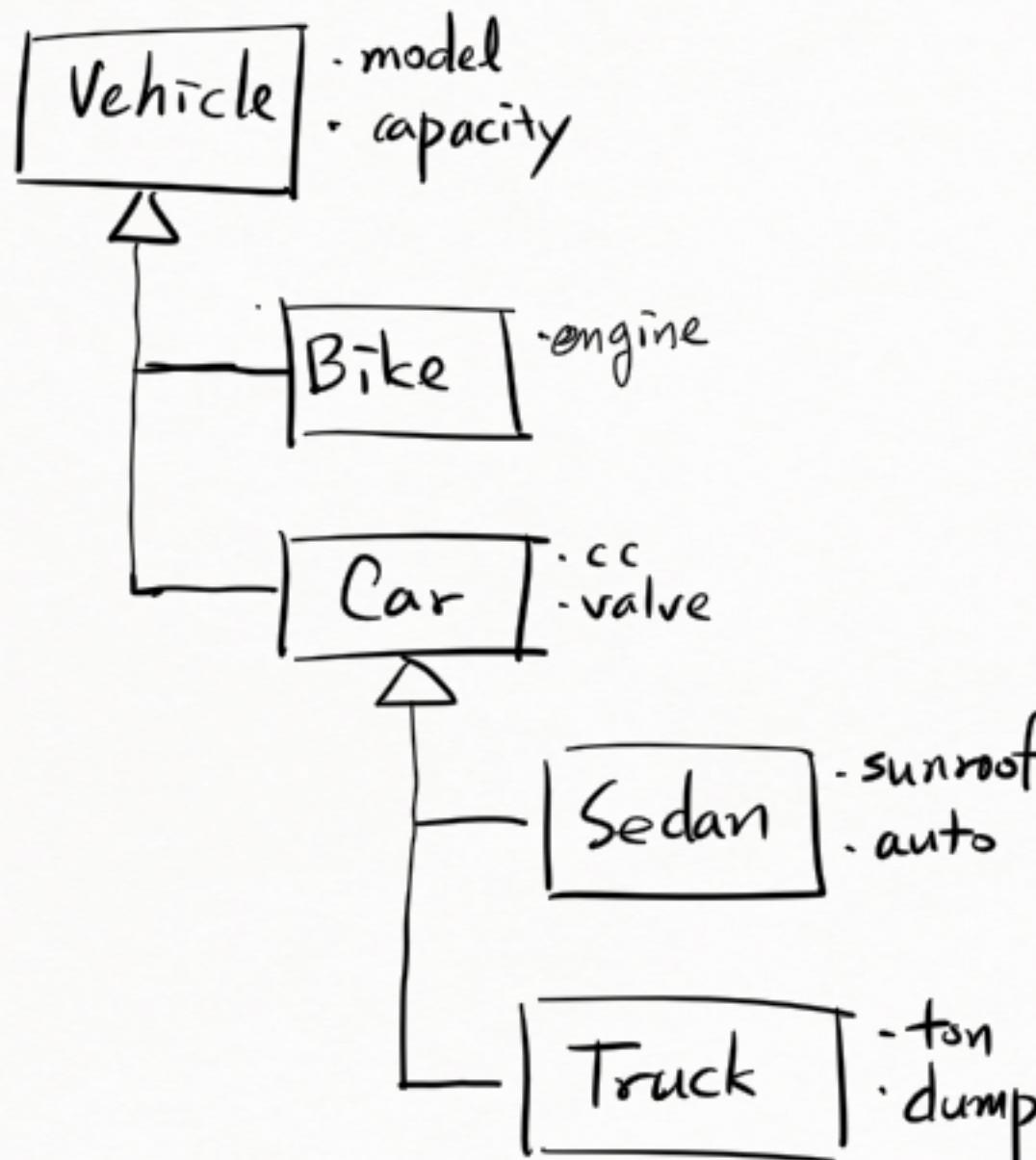
* 상속 : 추상클래스와 추상메소드 (oop.exst.n)

* 치수에서

- 서로 다른 클래스의 예제에 맞춰서
구현해야 하는 메서드인 경우,
 - 메서드 H그리치(리턴타입, 매개변수, 파라미터)를
작성한다.
 - 서로 다른 템플릿에게 구현을 강제한다.
 - 메서드의 문제를 만들지 않기 때문에
일반클래스 대로 추상메서드를
쓸 수 있다
 - 예전 추상클래스(또는 인터페이스) 대신
추상메서드를 가질 수 있다.



* 다형성 (polymorphism) : oop. exob. a



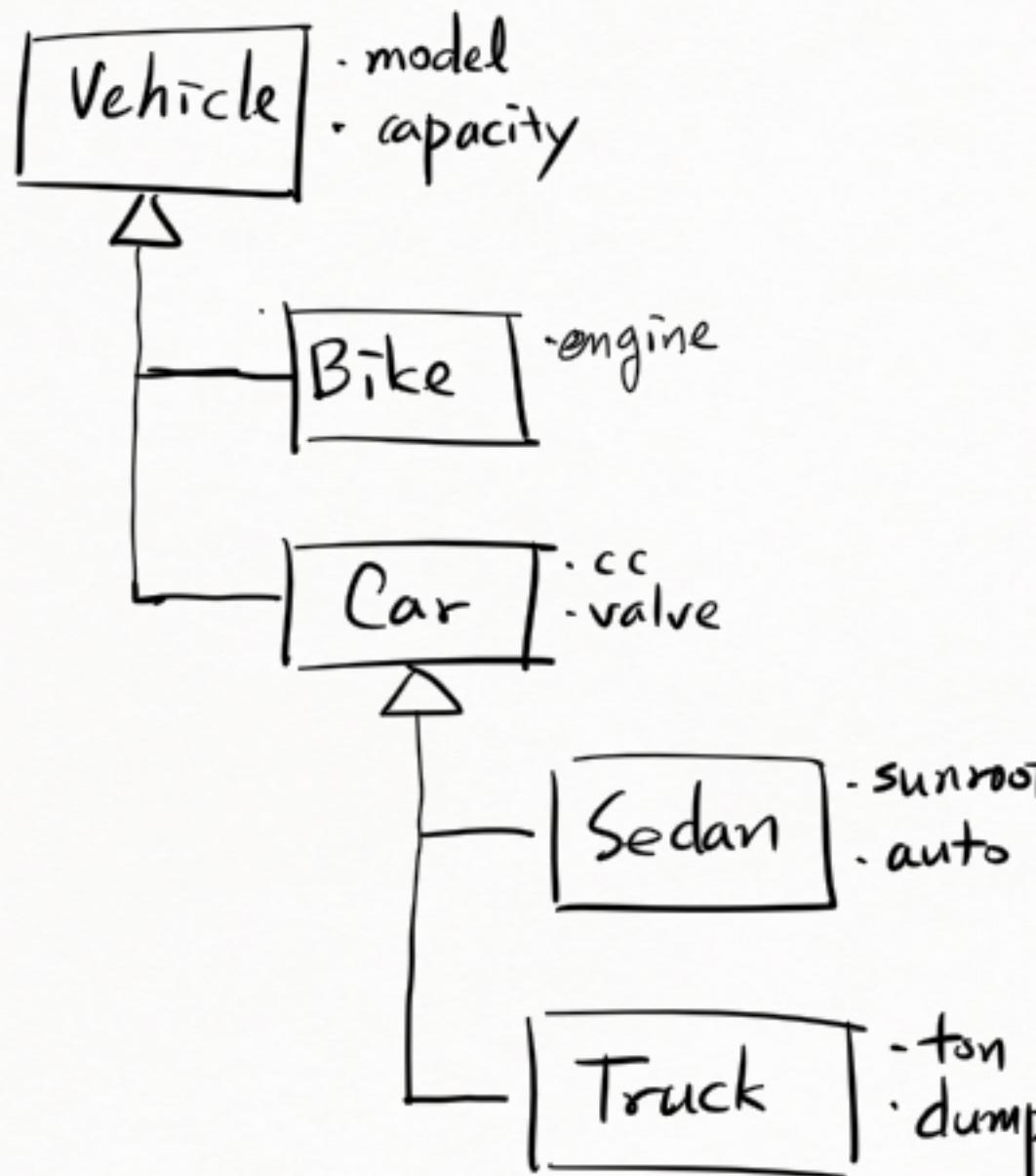
Vehicle c1;

c1 = new Vehicle(); → 

c1 = new Bike(); → 

- ① 상위 클래스의 레퍼런스는 하위 클래스의 인스턴스를 갖을 수 있다.
 ② " "
 ③ 하위 클래스의 인스턴스를 가지칠 수 있다.
 하위 클래스의 인스턴스를 가지칠 수 있다.

* 다형성 (polymorphism) : oop. exob. a



Car c1;
~~c1 = new Vehicle();~~ → **기타일 가능!**

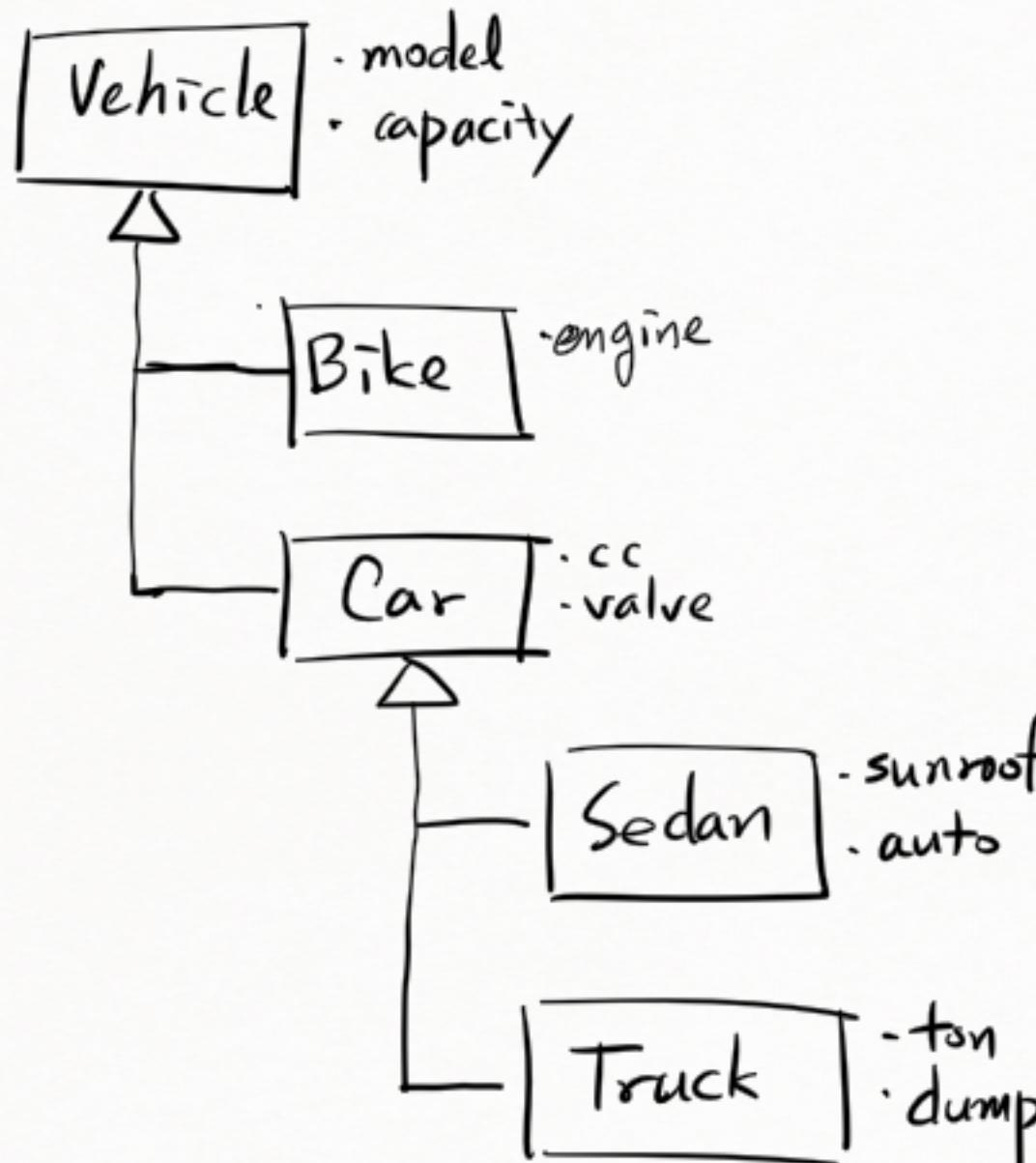
Car의 인스턴스 멤버를 초기화하는 경우,
Car의 인스턴스 멤버를 초기화하는 경우,
~~c1 = new Vehicle();~~ → **기타일 가능!**

Car의 인스턴스 멤버
{
 c1.cc = 2000;
 c1.valve = 16;
}

Vehicle의 인스턴스 멤버
{
 c1.model = "tta";
 c1.capacity = 5;
}

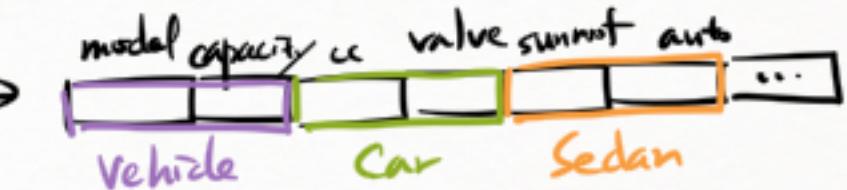
↑
존재하지 않는 멤버를 초기화하는 경우를 사용할 수 있기 때문이
아예 원천적으로 전파를 막을 것인
프로그램을 만드는 경우에 문제가 된다

* 다형성 (polymorphism) : oop. exob. a



Car c1;

c1 = new Sedan(); →



c1. model = "aaa"; }
c1. capacity = 5; } Vehicle

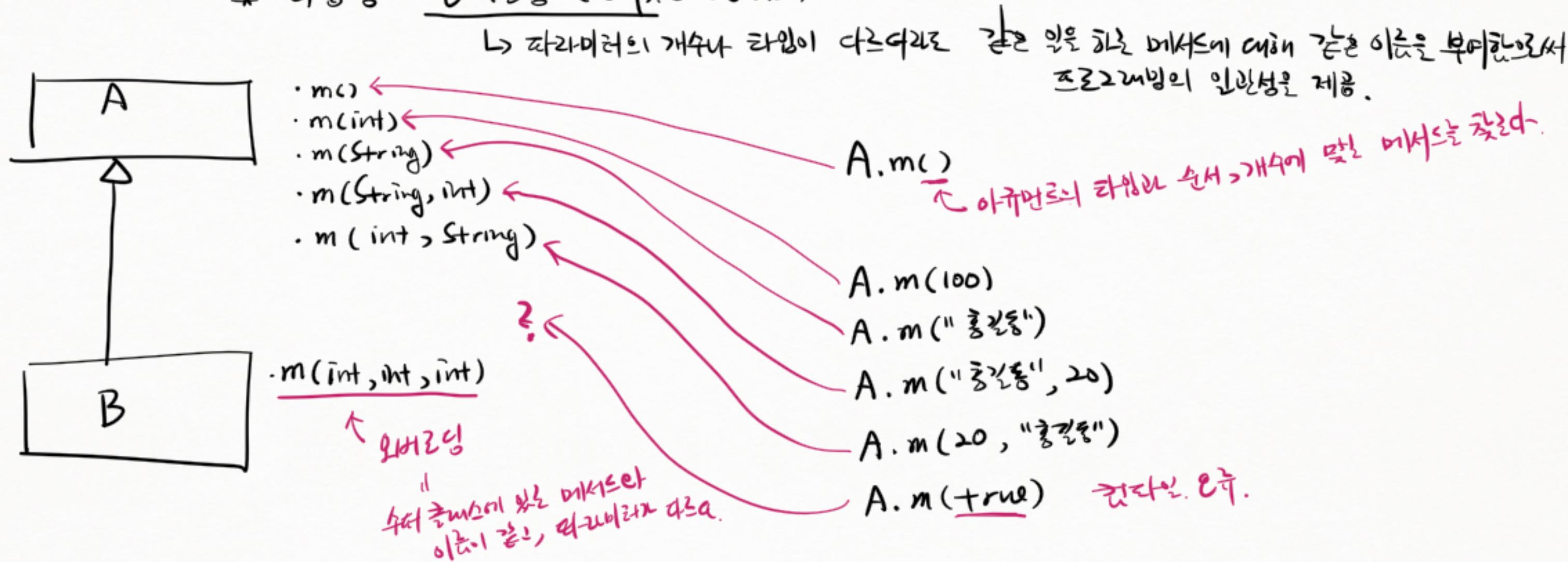
c1. cc = 2500; }
c1. valve = 16; } Car

c1. sunroof = true;
c1. auto = true;

값대로 초기화

- c1은 Car 클래스의 인스턴스이므로
 ↳ 자식인 Car 클래스의 멤버에 접근해서 해당 값을 초기화
- c1이 바로 Sedan 객체를 가지므로 초기화

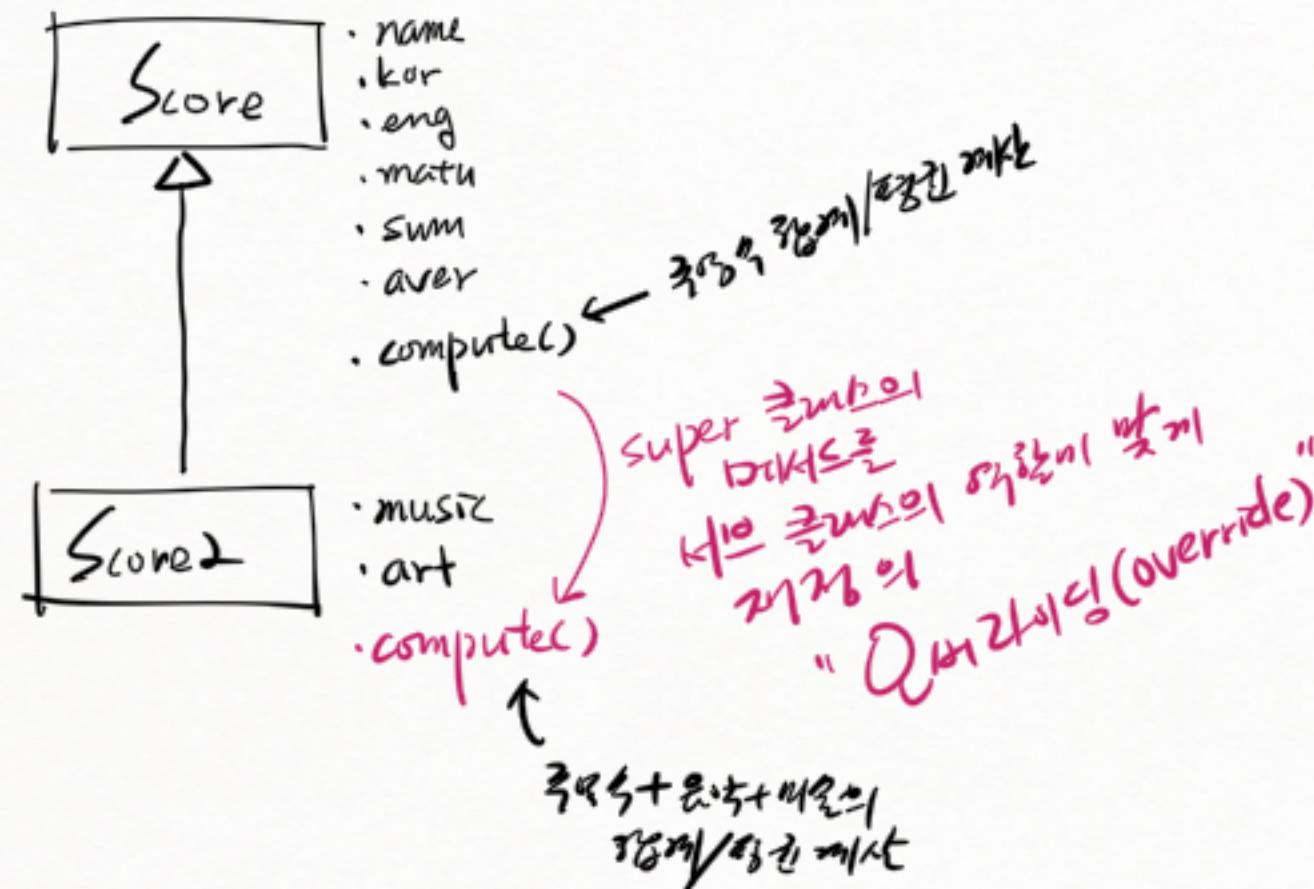
* 다형성 - 오버로딩 (Chap. ex06.b)



* 다형성 - 오버라이딩 (overriding)

com.econcs.oop.ex6.c

↳ 상속 받은 메서드를 서브 클래스의 예외에 맞춰 재정하는 것.



new Score2()

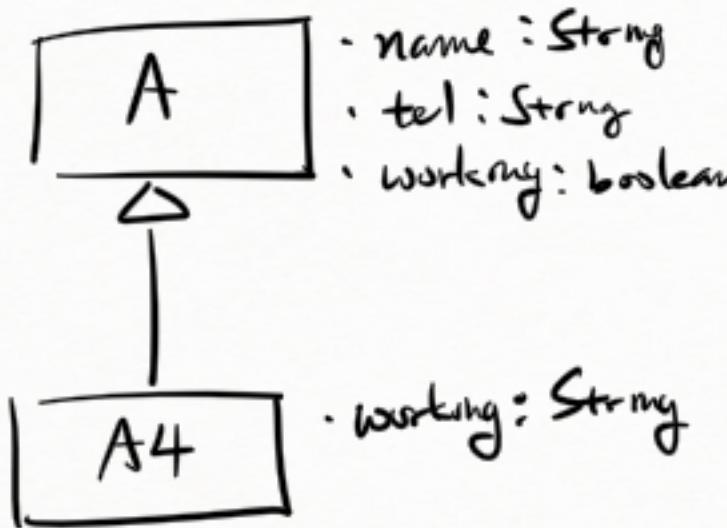


name	kor	eng	math	sum	aver	music	art	...
null	100	100	100	300	100.0	50	50	...

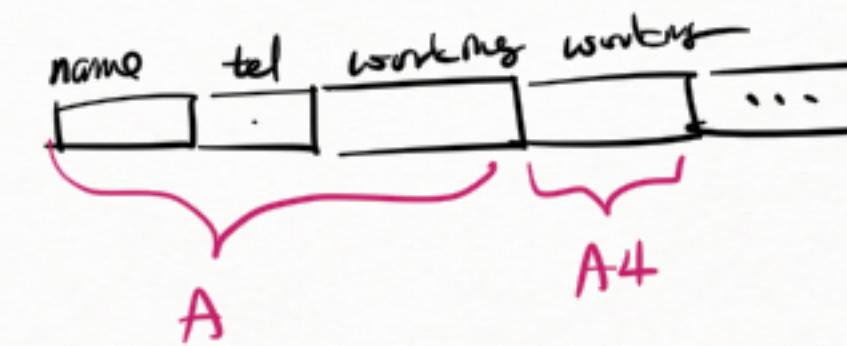
Score

Score2

* 초기화 - 힐드 초기화



A4 obj = new A4()

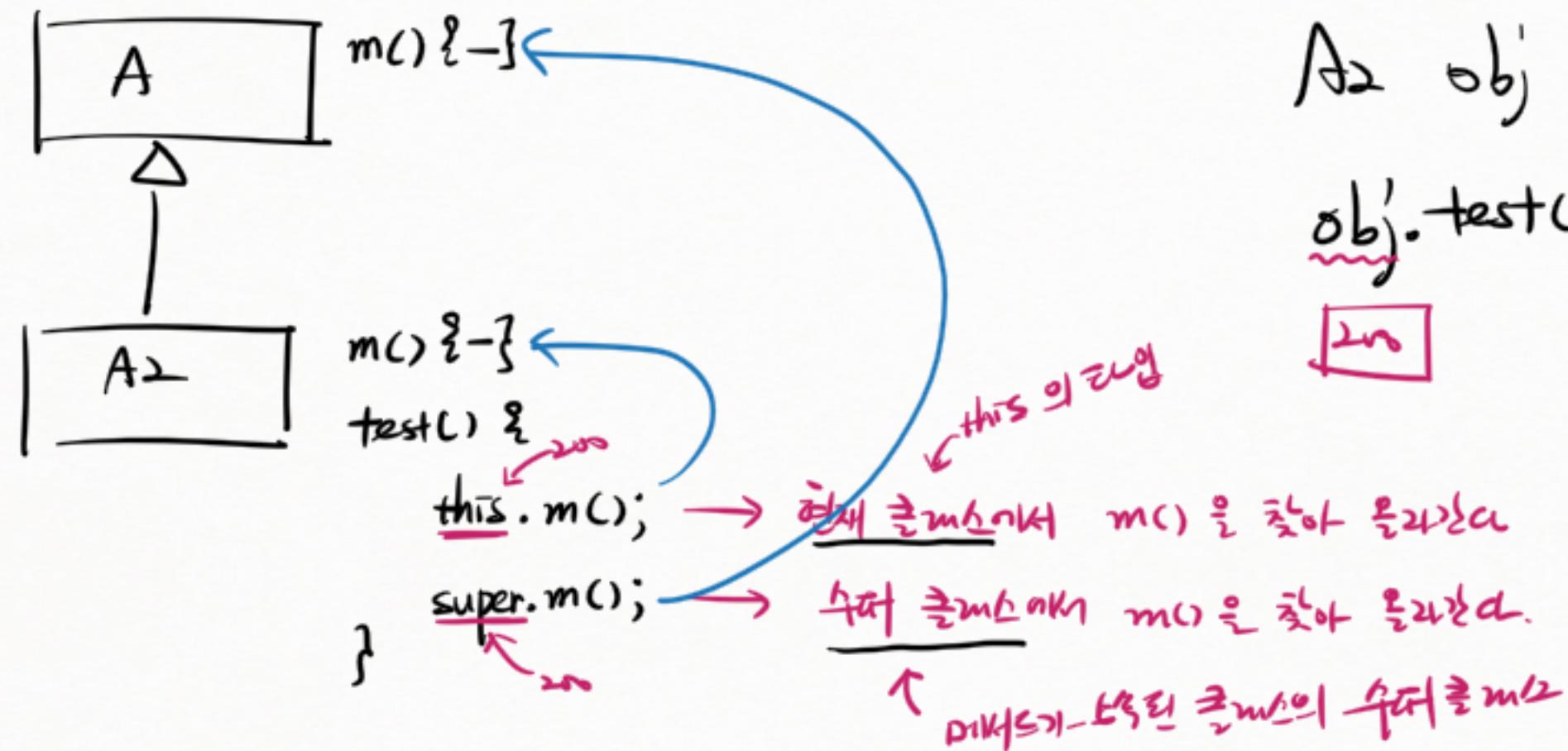


obj.name = "—";

obj.tel = "—";

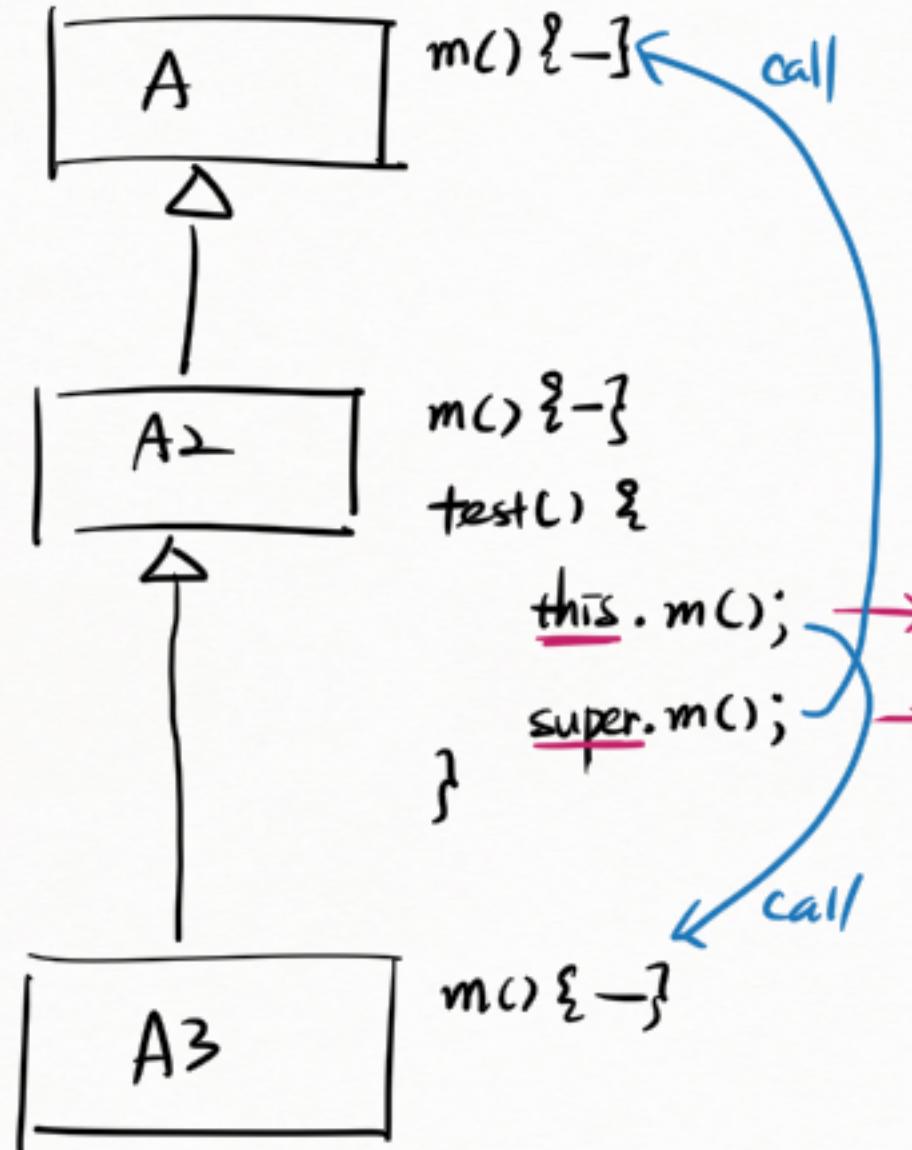
obj.working = ?

* 상속이념 - super 키워드 (99. ex06.c. Exam0410)



`A2 obj = new A2();`
`obj.test();`
200

* 오버라이딩 - super 키워드 (opp. ex6. c.Exam0411)



A3 obj = new A3()

obj.test();

↑ 상속받은 메서드를 호출할 경우

this의 실제 태입

↳ test()는 호출될 때 현재로 개체의 태입 (A3)

현재 클래스에서 m()을 찾아 올라간다

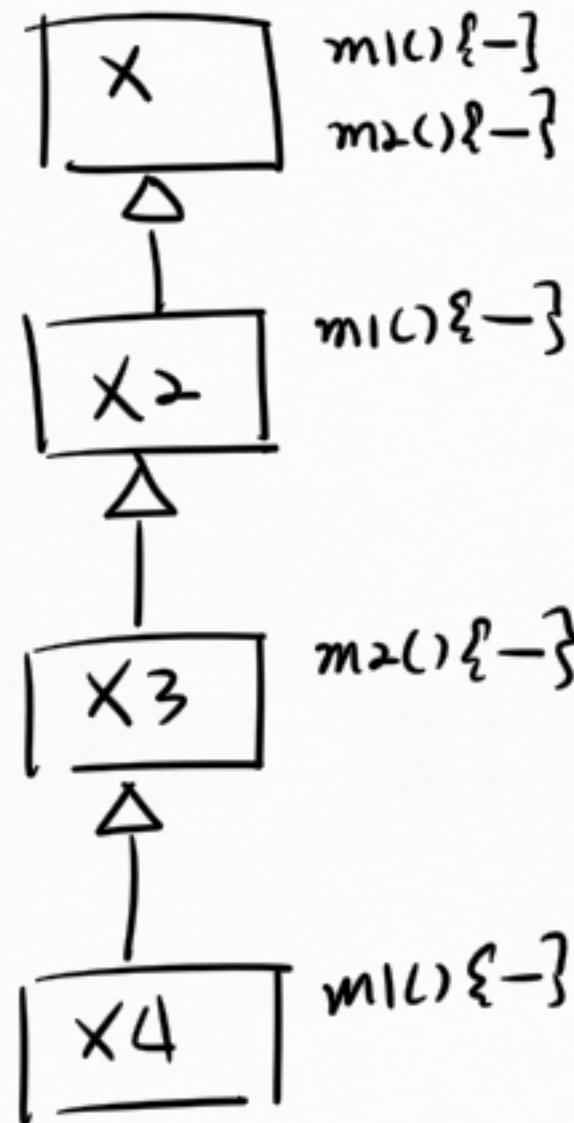
부모 클래스에서 m()을 찾아 올라간다.

메서드가 소속된 클래스의 부모 클래스

(A)

* this et super 例題

Exam 420



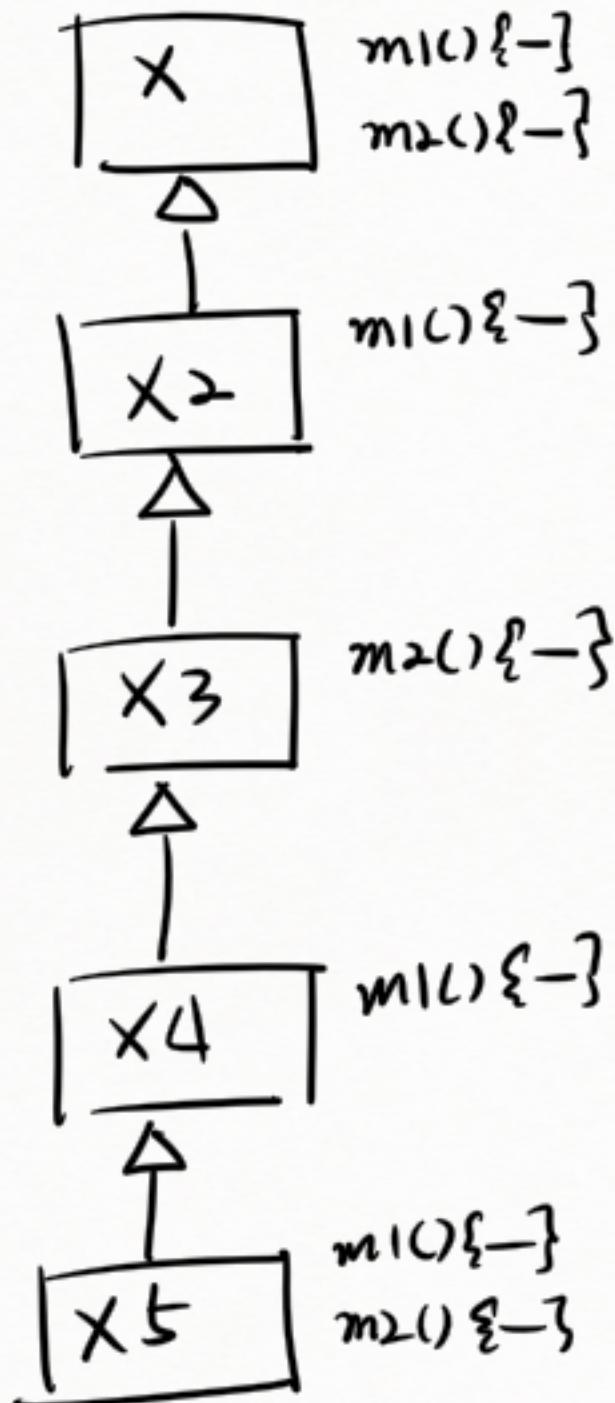
test()
 this.m1(); → X4.m1()
 super.m1(); → X2.m1()
 this.m2(); → X3.m2()
 super.m2(); → X3.m2()

X4 obj = new X4();
obj.test();
 ↑
 this

Internationalization
Localization

* this 와 super 차이

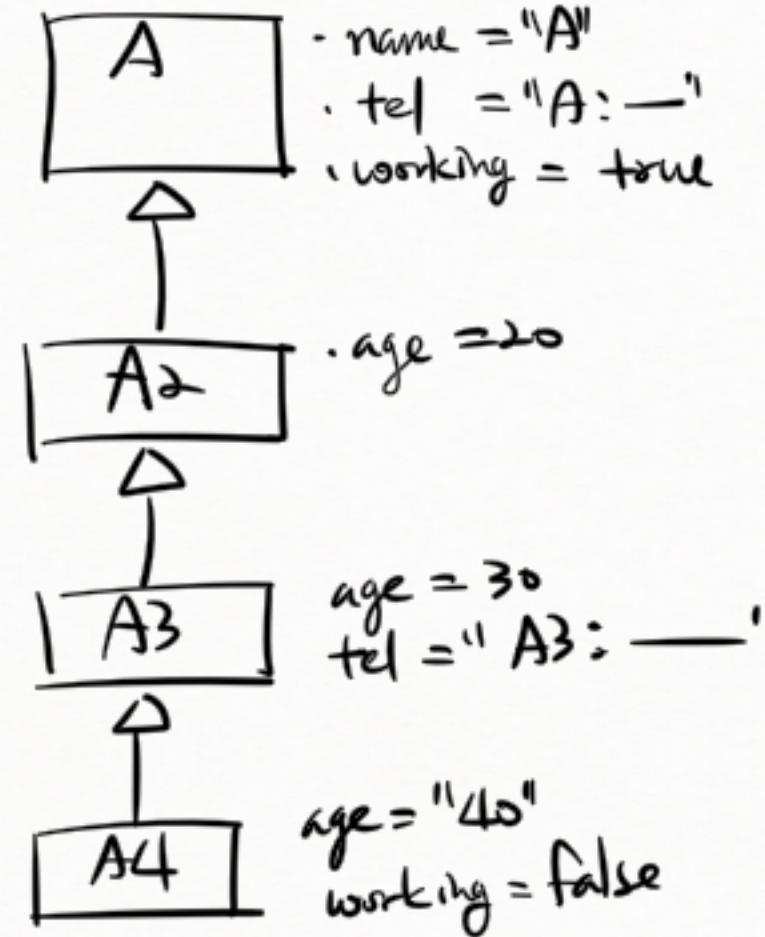
Exam 0421



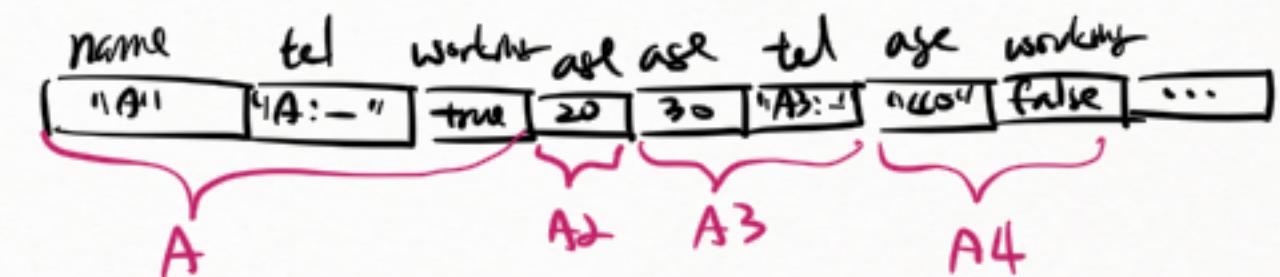
X5 obj = new X5();
obj.test();
↑
X4의 test()는?

test() {
 this.m1(); → X5.m1()
 super.m1(); → X2.m1()
 this.m2(); → X5.m2()
 super.m2(); → X3.m2()
}

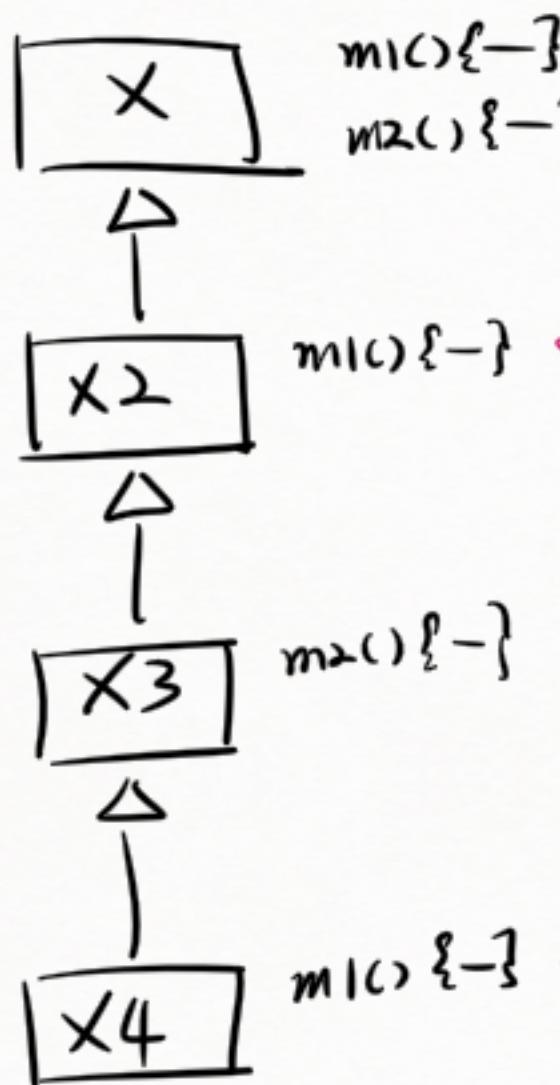
* 2) 디렉스와 퍽스



A4 a4 = new A4();



* 레퍼런스와 메서드



$x4 \quad x4 = \text{new } X4();$

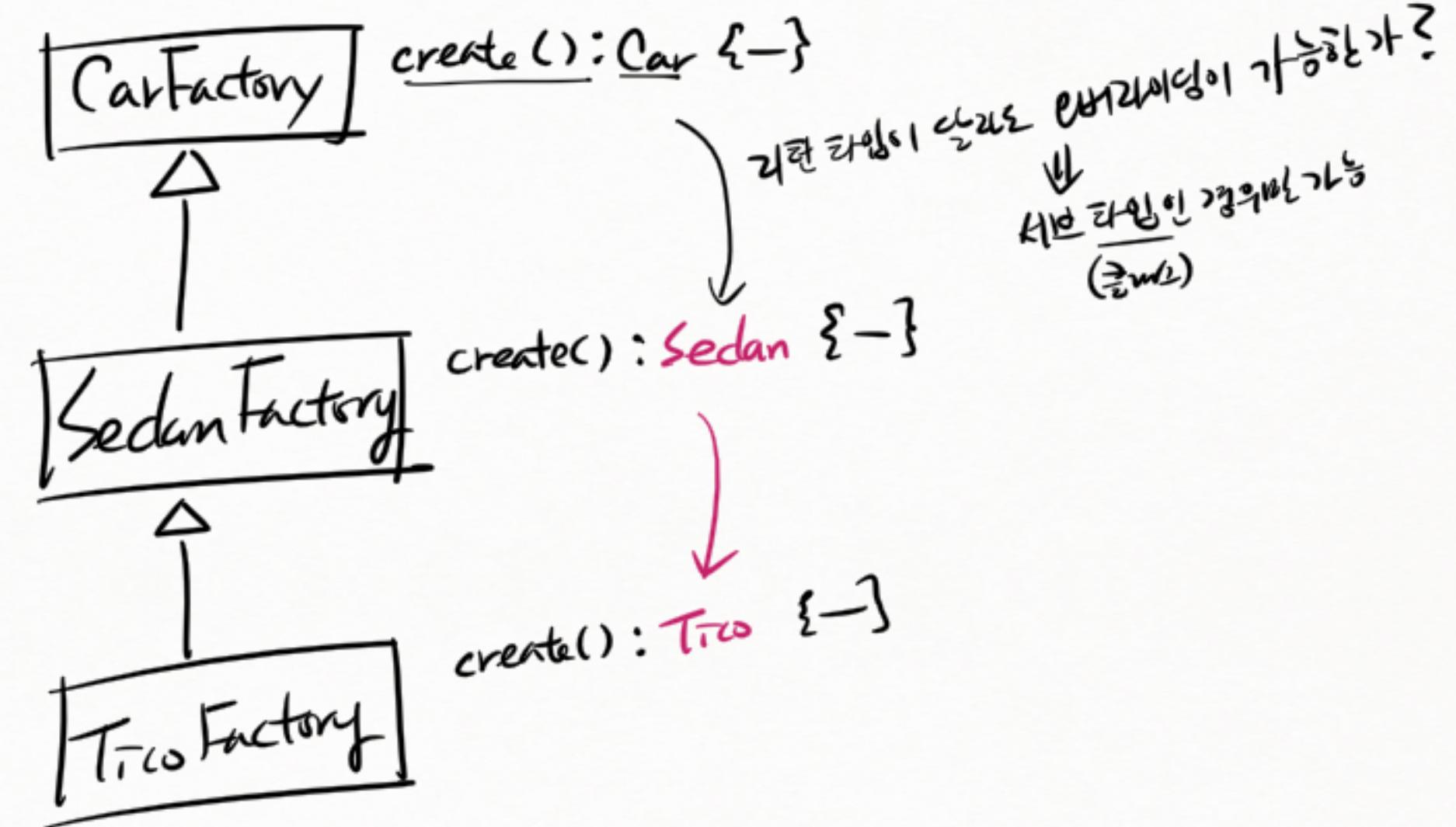
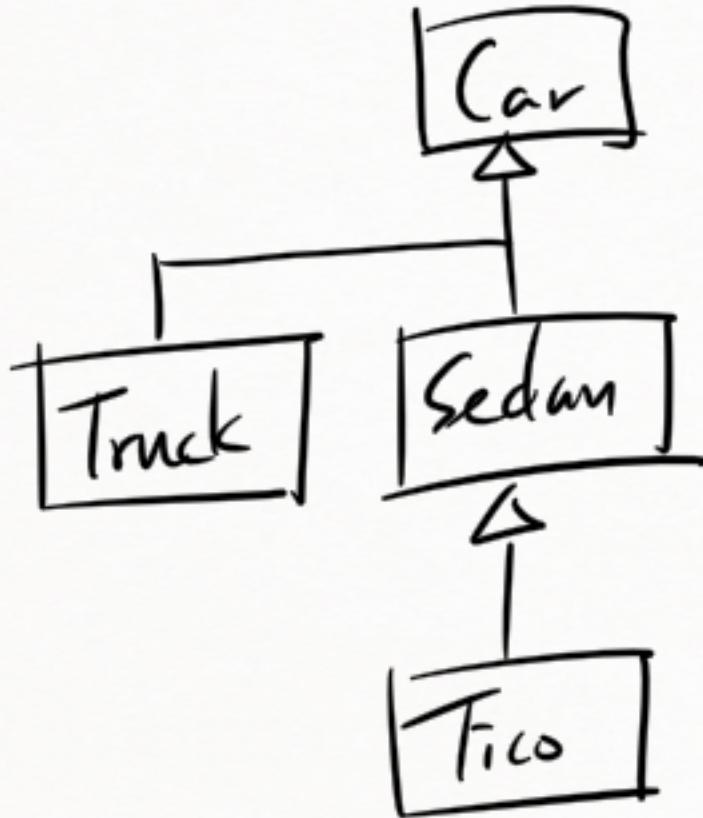
$x4.m1();$

$\underline{x2} \quad \underline{x2} = \underline{x4};$

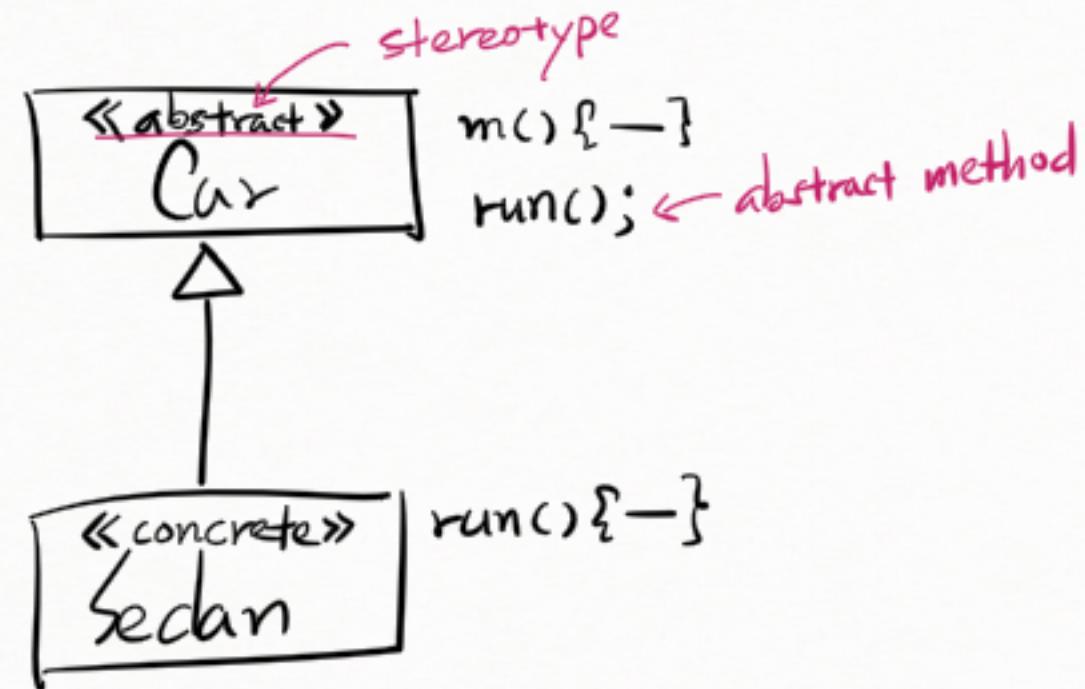
$\underline{x2.m1();}$

실제 레퍼런스가 가지는 자신의 태입이나 ~~타입이나~~ ~~타입이나~~

* 구조화된 구조화된



* 추상클래스의 추상메소드를 상속받아쓰고 레퍼런스



Car c = new ~~Car()~~;

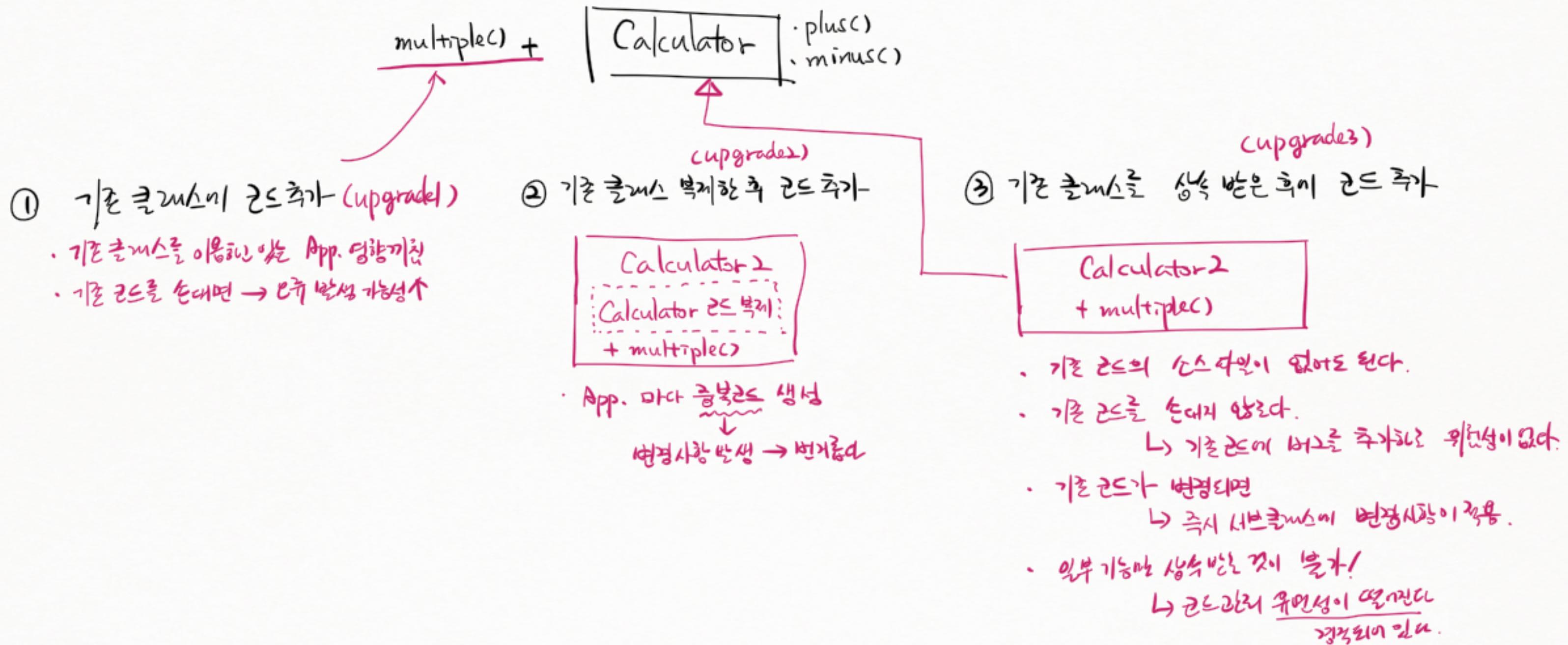
↑
추상클래스로
인스턴스 생성 불가!

Car c = new Sedan();

c.run();

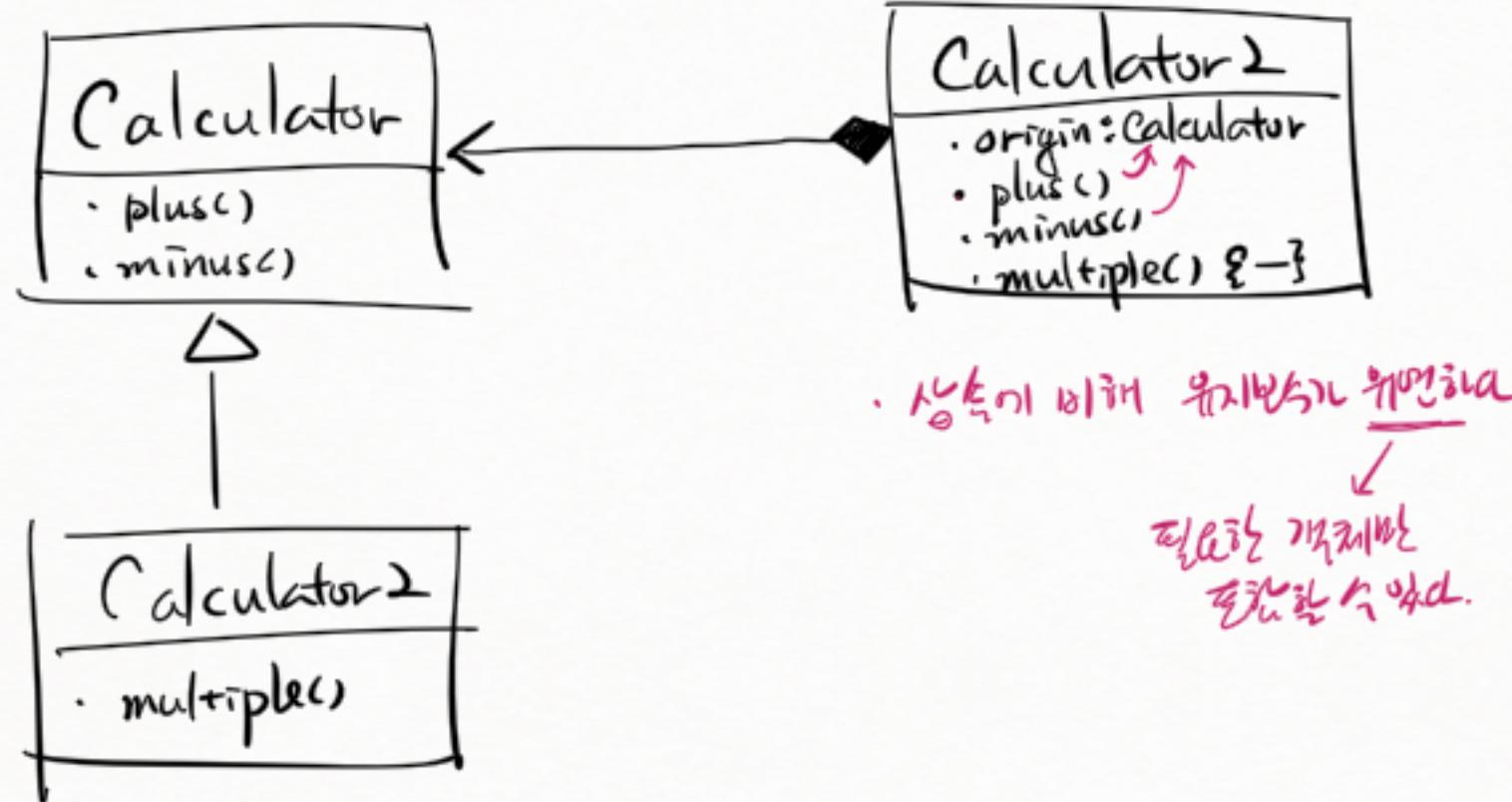
↑
하지만 c가 가리키는 객체의 m()이 실행된다
(Sedan ~ m())

* 가능한 확장을 위한 다양한 방법 (oop, ext, xl, *)



* 가능한 학점을 위한 다양한 방법 (oop, ext, xl, *)

④ 단계별로를 통해 가능한 학점



* Composition 관계
(자식)

Calculator2 == Calculator
(사람) ↑ (신장)

Lifecycle 이 같다

* 가능한 학점을 얻기 위한 다양한 기법 (oop, exat, xl, *)

