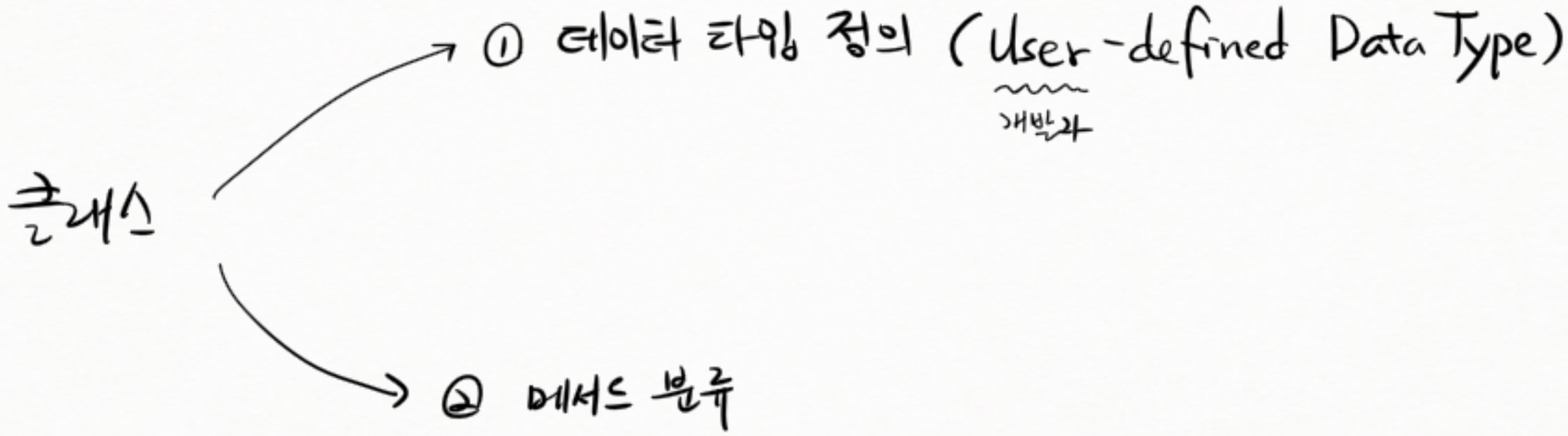
* 클래스 문법



* 클래스 문법 - 데이터 타입 정의

```
class 데이터타입명 {
    변수선언
    :
}
```

예)

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

← 메모리 설계도

new 명령을 실행하면 클래스에 정의된 대로 변수가 준비된다.

* 클래스를 이용해 내 데이터 타입의 메모리 만들기

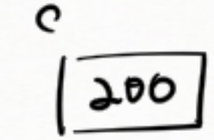
Contact 인스턴스의 주소를 저장한 변수 = 레퍼런스(reference)

데이터타입명 = 클래스명

```
Contact c = new Contact();
```

클래스 설계도에 따라 변수(메모리) 준비 => Heap 영역

인스턴스(instance) = 객체(object)



* 레퍼런스 배열

① 배열 사용 전

Contact c1, c2, c3; ← 인스턴스의 주소를 저장하는 변수 = 레퍼런스 = 포인터(pointer)

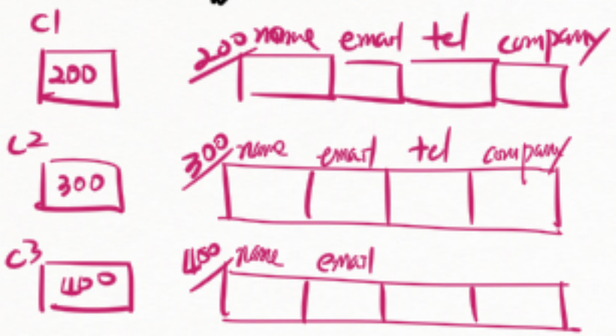
c1 = new Contact();

c2 = new Contact();

c3 = new Contact();

⇓

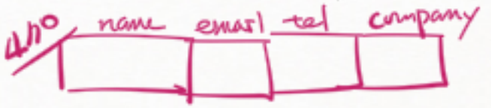
← Contact 설계도에 따라 만들어진 메모리 "Contact의 인스턴스" "Contact의 객체"



② 배열 사용 후

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

arr[0] = new Contact();



arr[1] = new Contact();

arr[2] = new Contact();

* 레퍼런스와 인스턴스 변수

```
Contact c = new Contact()
```

기존 인스턴스의 주소를 잃고 있는 레퍼런스가 있게 되면 "Garbage"가 된다.

① 인스턴스 변수에 값 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 = "인경정";
c.email = "leem@";
c.tel = "2222";
c.company = "캔트";
```

Method Area

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

JVM Stack

Score s;

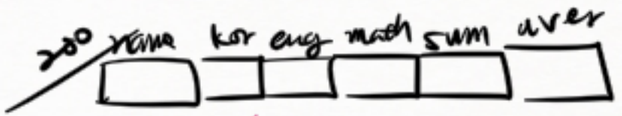
s [200]

↑ Score의 레퍼런스

Heap

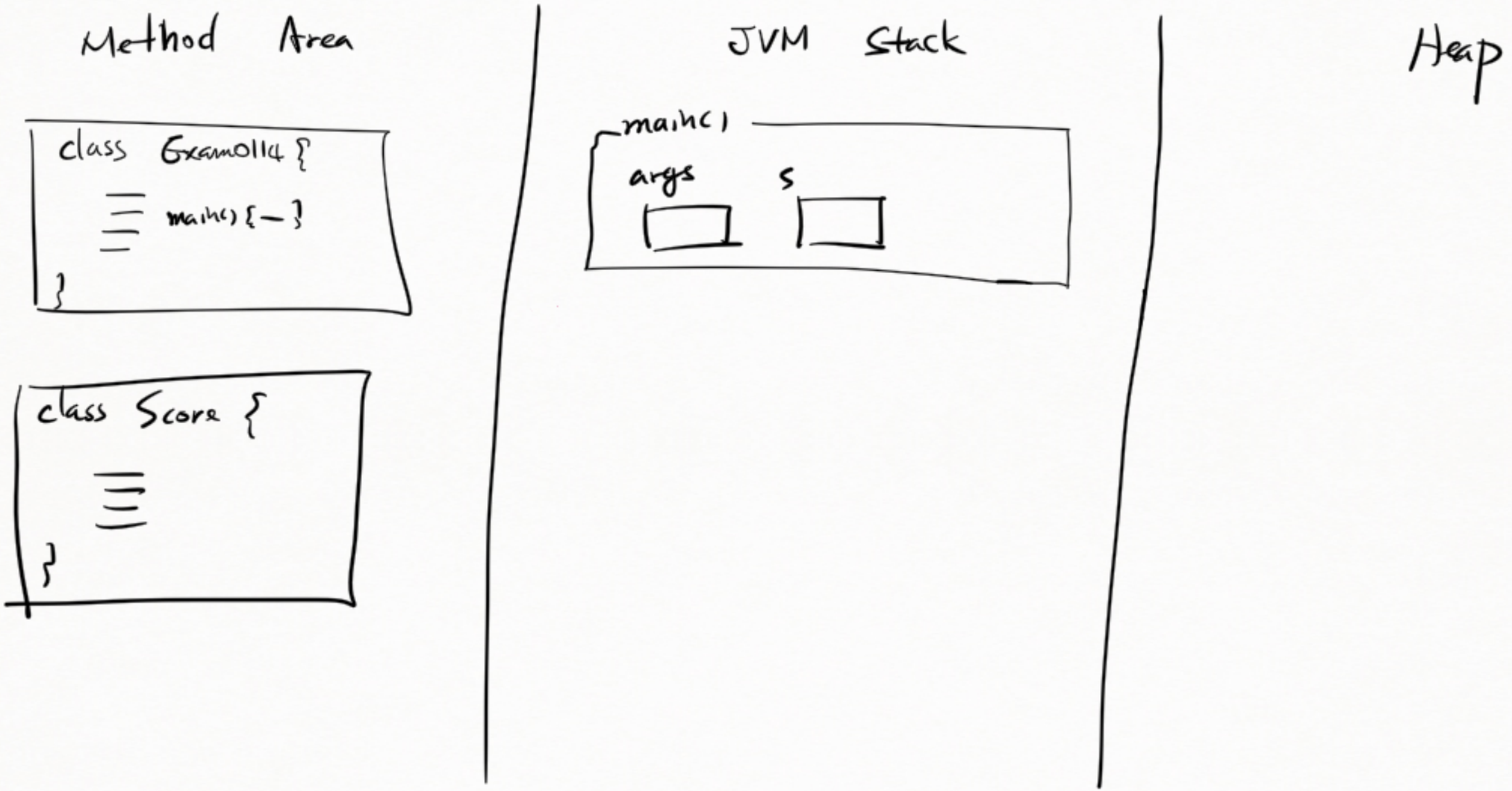
new Score()

↓

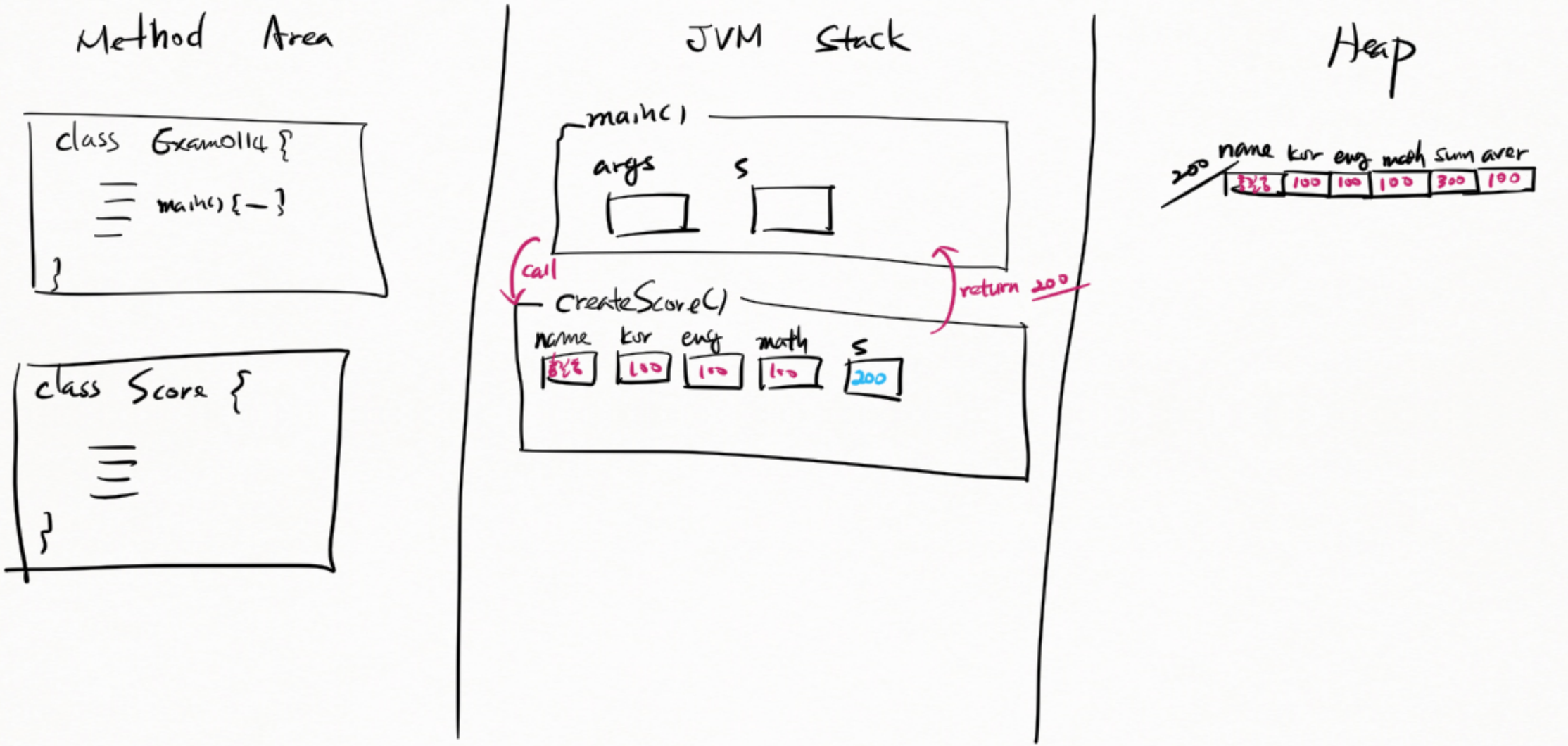


↑ Score의 인스턴스 (객체)

* com.eomcs.oop.ex01.Exam0114



* com.eomcs.oop.ex01.Exam0114



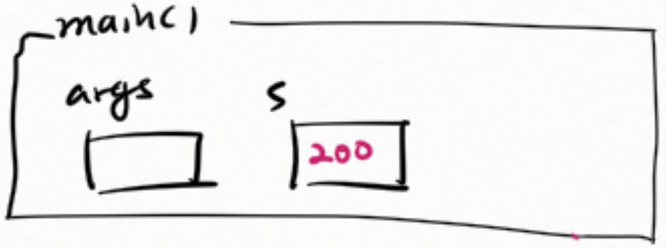
* com.eomcs.oop.ex01.Exam0114

Method Area

```
class Exam0114 {
    main() {-}
}
```

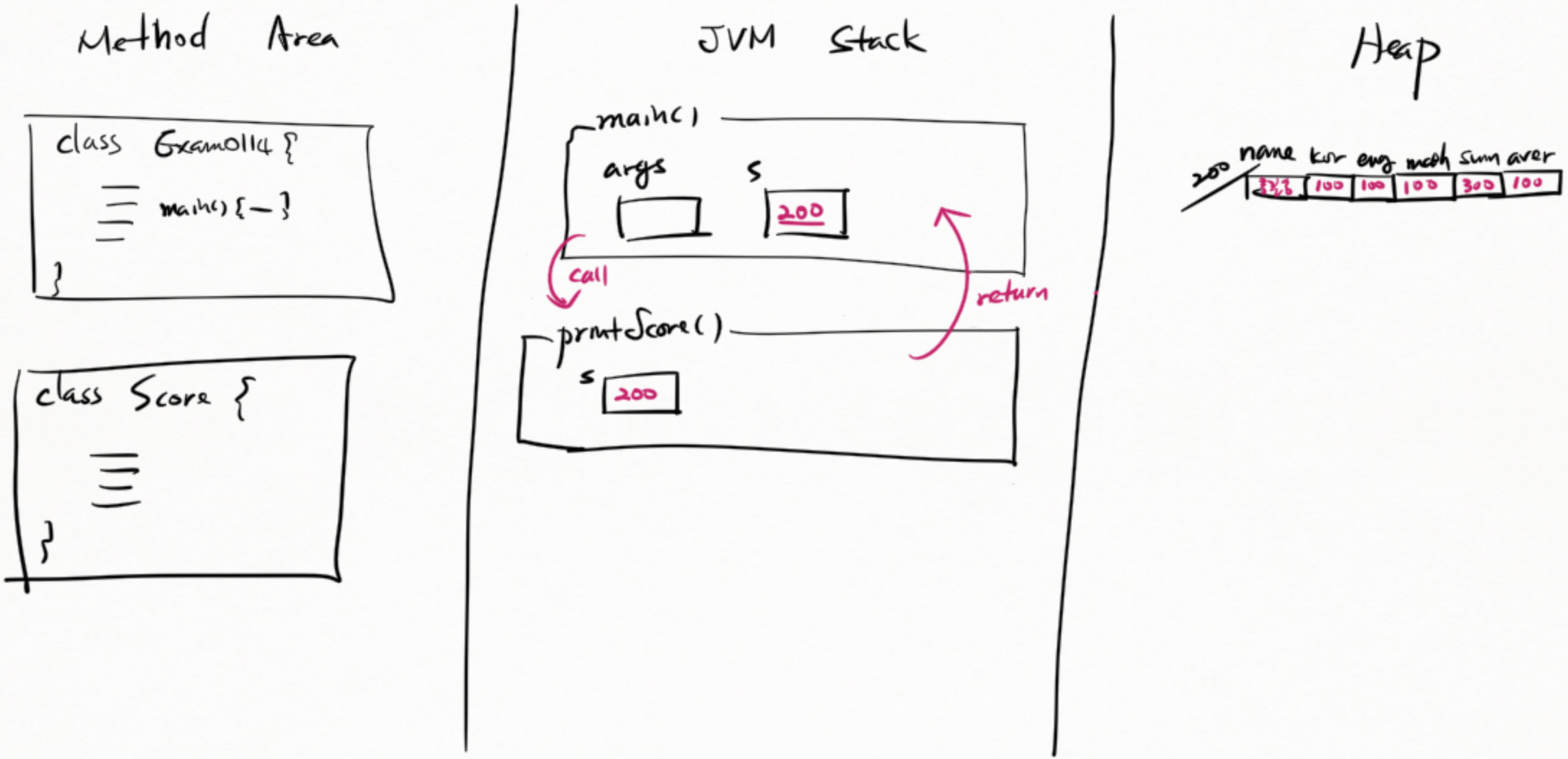
```
class Score {

}
```

JVM Stack



Heap

* com.eomcs.oop.ex01.Exam0114



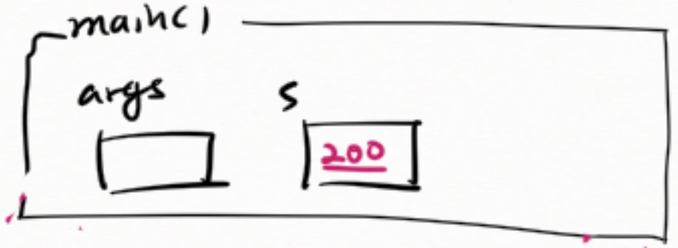
* com.eomcs.oop.ex01.Exam0114

Method Area

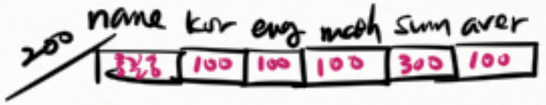
```
class Exam0114 {
    main() {_}
}
```

```
class Score {

}
```

JVM Stack

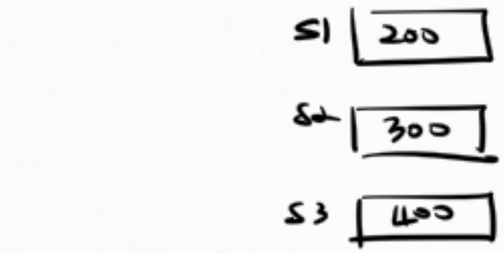


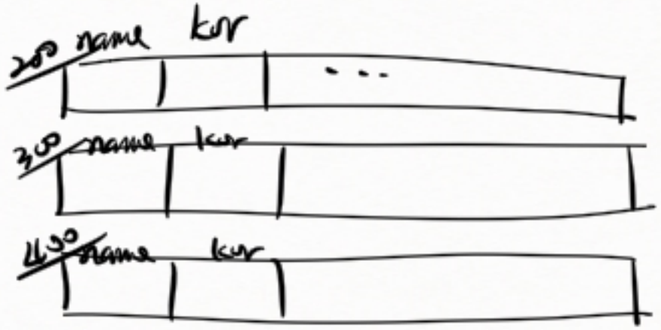
Heap



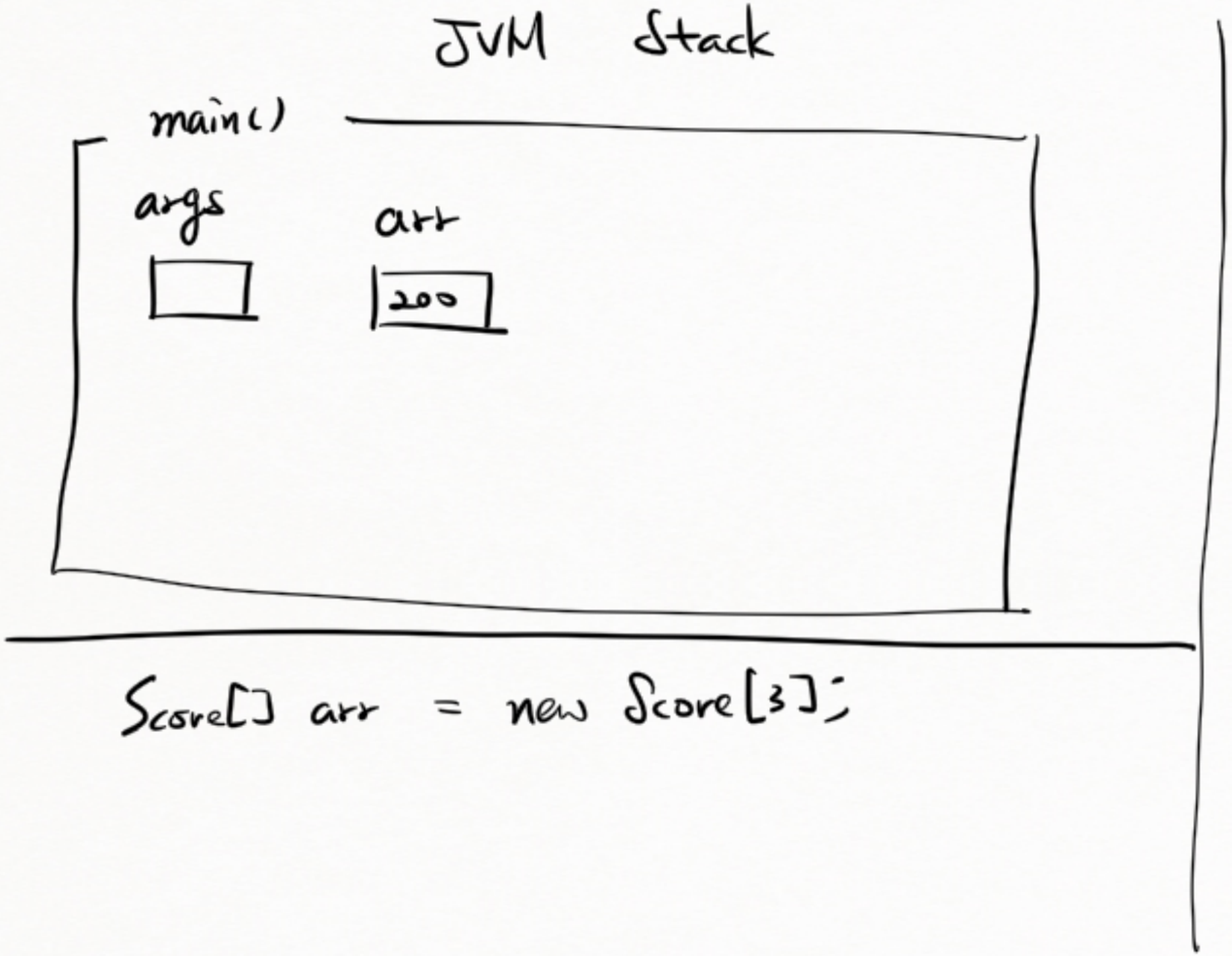
* com.eomcs.oop.ex01.Exam0210

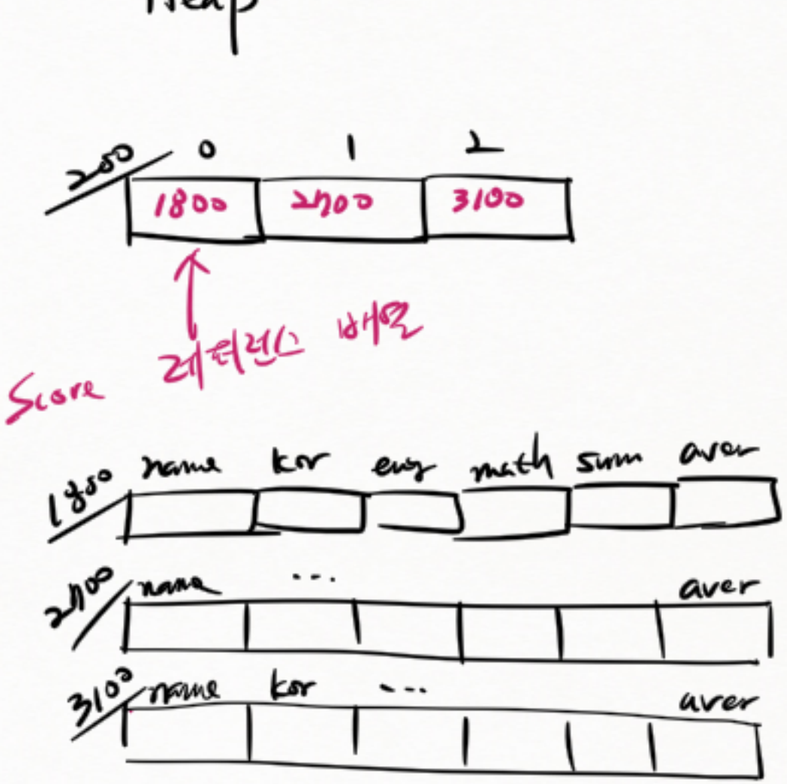
Score s1, s2, s3;





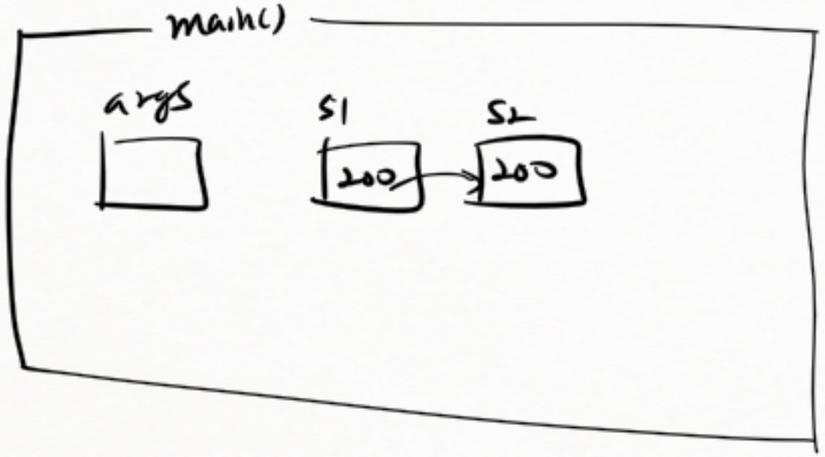
* com.eomcs.oop.ex01.Exam0220



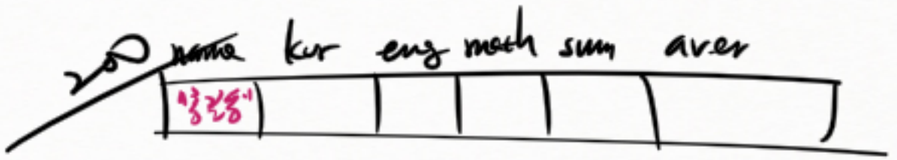


* com.eomcs.oop.ex01.Exam0310

JVM Stack

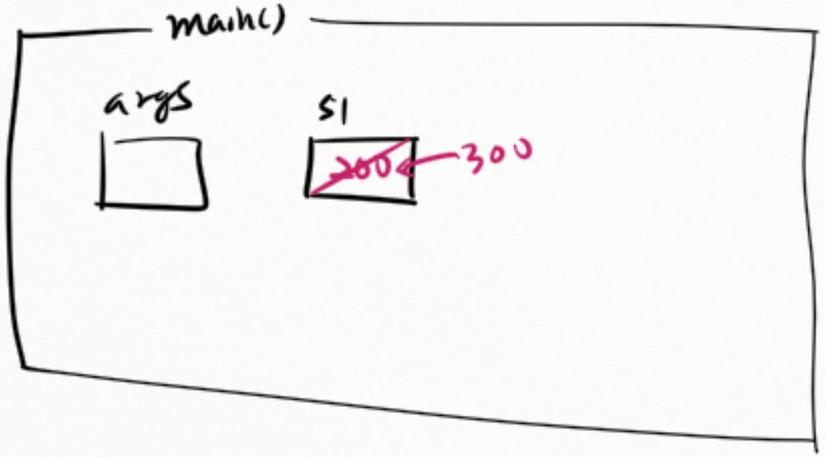


Heap



* com.eomcs.oop.ex01.Exam0320

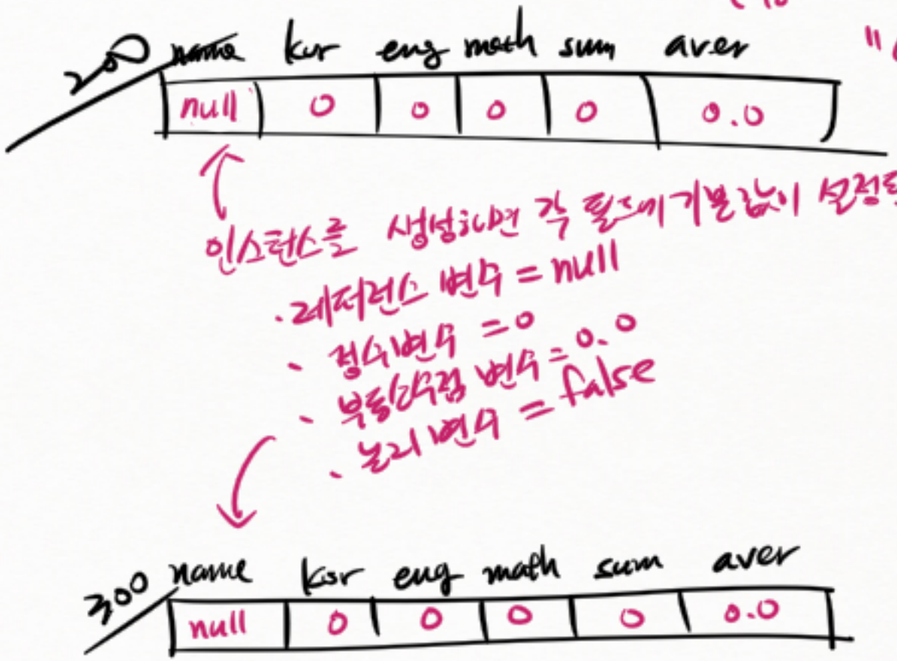
JVM Stack




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

Heap

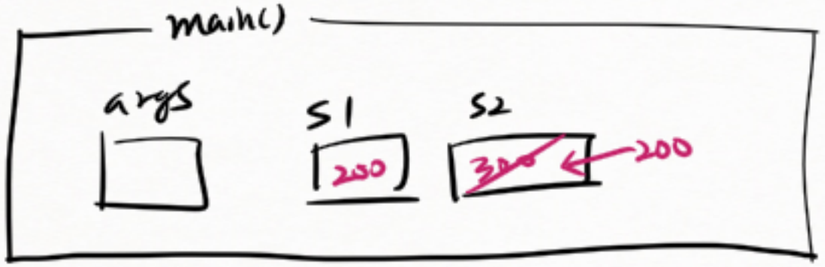
이 인스턴스의 주소를 가리키는 레퍼런스가 단 하나도 없으면 사용할 수 없다 (접근할 수 없다)
"Garbage"가 된다.



인스턴스를 생성하면 각 필드에 기본값이 설정된다.
- 레퍼런스 변수 = null
- 정수변수 = 0
- 부동소수점 변수 = 0.0
- 논리 변수 = false

* com.comcs.oop.ex01.Exam0330

JVM Stack




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

s2 = s1;
```

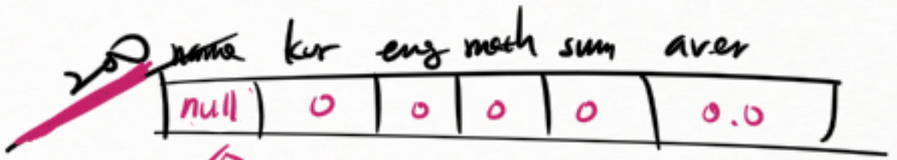
인스턴스 주소를 레퍼런스에 저장할 때

JVM은 내부적으로 인스턴스의 레퍼런스 카운트를 관리한다

인스턴스	레퍼런스개수	
200	~~1~~ 2	+1
300	~~1~~ 0	-1

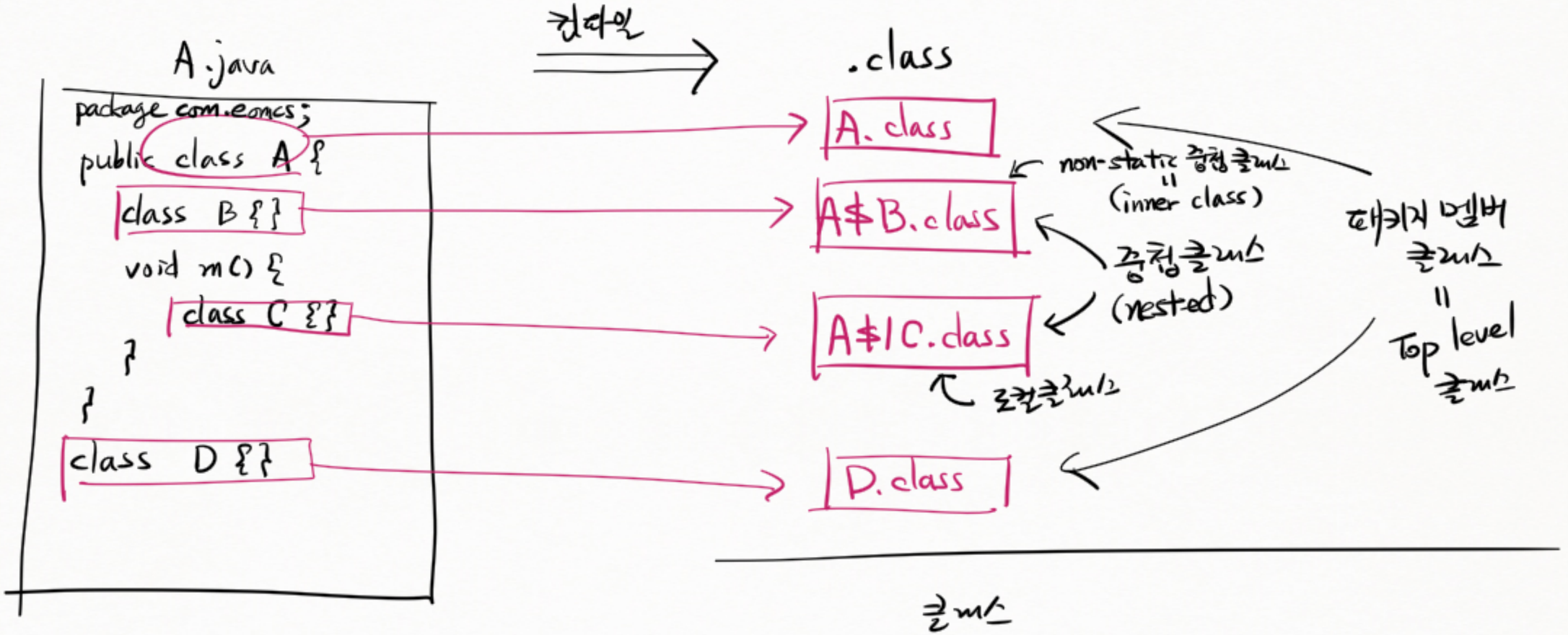
레퍼런스 카운트 개수가 0 이면 "가비지(Garbage)"가 된다.

Heap

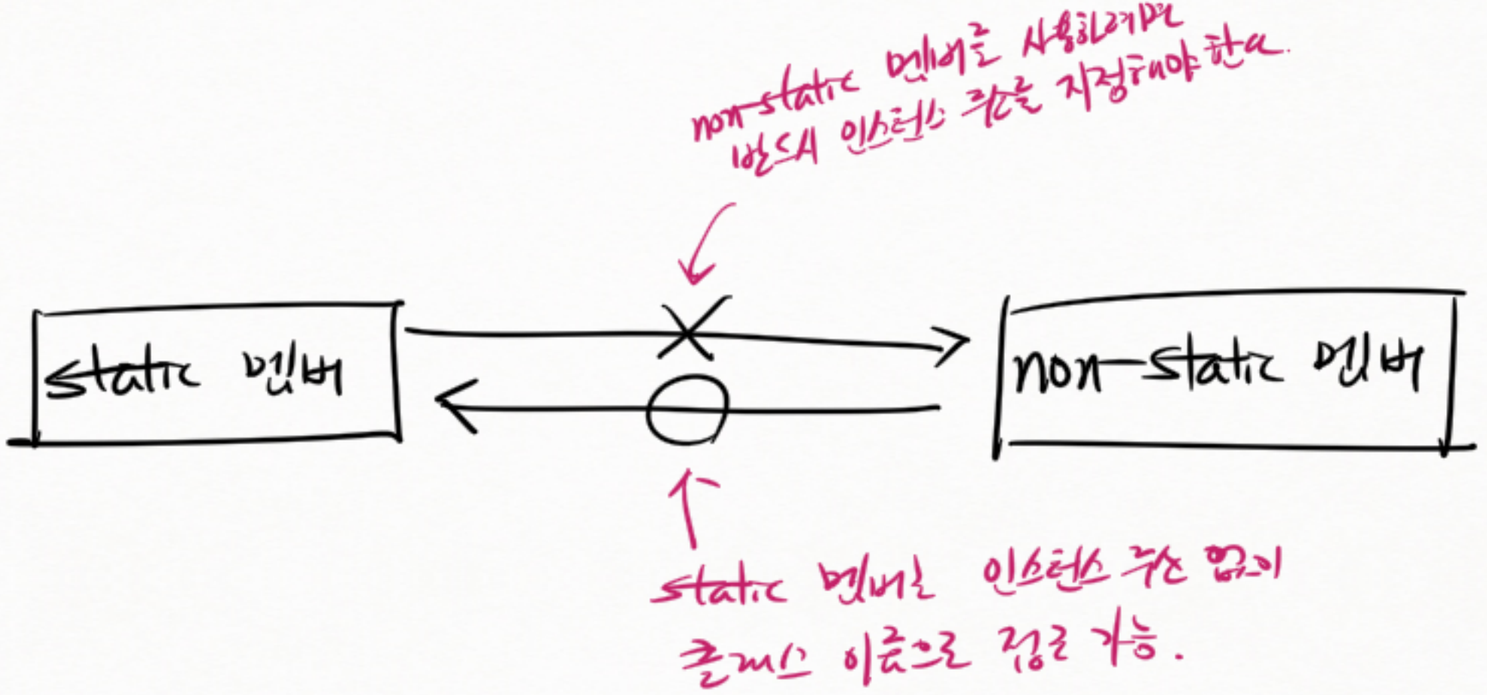


인스턴스를 생성하면 각 필드에 기본값이 설정된다.
- 레퍼런스 변수 = null
- 정수변수 = 0
- 부동소수점 변수 = 0.0
- 논리변수 = false

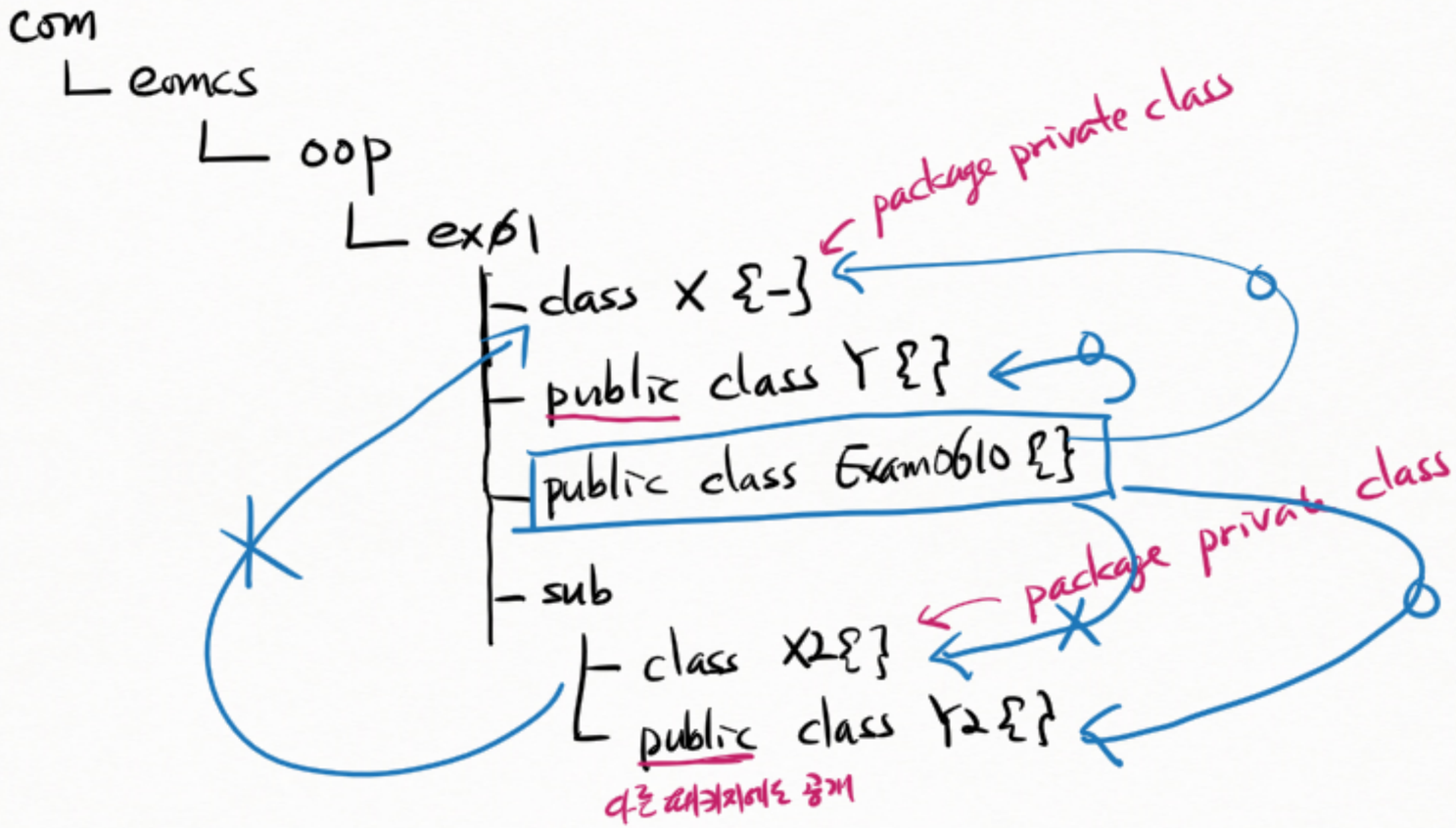
* 클래스 블록과 .class 파일



* static 멤버와 non-static 멤버 간의 접근 규칙



* 패키지 멤버 클래스 : (default) vs public



* 메서드 호출과 로컬 클래스 정의

```
class A {
    void main() {
        class Contact {
            String name;
            String tel;
        }
        Contact c = new Contact();
        c.name = "홍길동";
        c.tel = "010-1111-2222";
    }
}
```

빼기

컴파일

Contact.class

```
class Contact {
    String name;
    String tel;
}
```

A.class

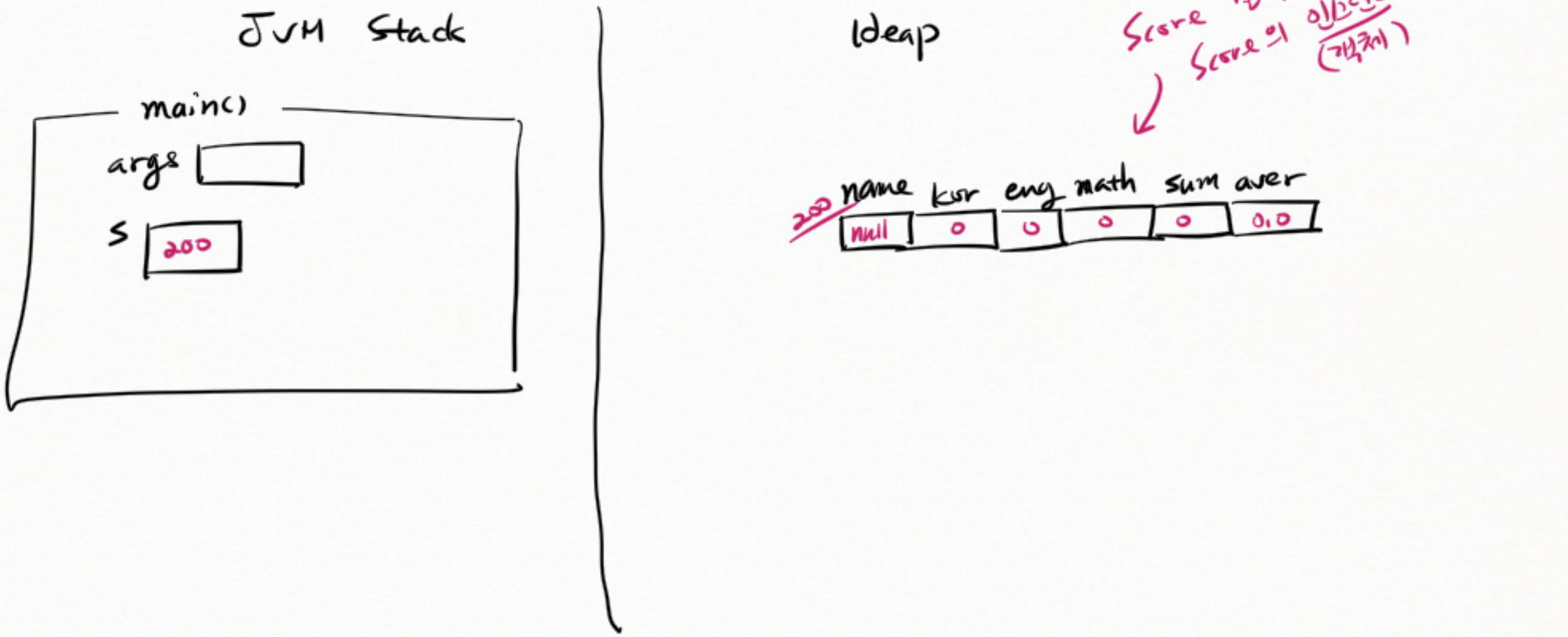
```
class A {
    void main() {
        Contact c = new Contact();
        c.name = "홍길동";
        c.tel = "010-1111-2222";
    }
}
```

* 클래스 문법 - 사용자 정의 데이터 타입

① 레퍼런스 유형 선언

```
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 정의하고 그 데이터를 다루는 연산자를 정의
스태틱 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;
    }
}
```

필드(변수) = Data 구조

메서드 = 다루는 연산자

```
int a;
a = -100;

a++;
```

a: 데이터 / ++: 연산자

⟺

```
Score s = new Score();
s.name = "홍길동";
s.kor = 100;
s.eng = 90;
s.math = 80;

Score.calculate(s);
```

Score: 설계/분류 → 연산자(operator) = 메서드(method) = 함수(function) = 메세지(message) ← 분류/설계

s: 데이터

calculate: 연산자

* 클래스 문법 — 사용자 정의 데이터 타입

③ 데이터 구조를 정의하고 그 데이터를 다루는 연산자를 정의

non-static 메서드 (인스턴스)

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

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

필드(변수) = Data 구조

메서드 = 다루는 연산자

```
int a;
a = -100;
a++;
```

a : 데이터, ++ : 연산자

```
Score s = new Score();
s.name = "홍길동";
s.kor = 100;
s.eng = 90;
s.math = 80;
~~Score~~.calculate(s);
```

기존의 연산자를 사용하는 문법과 비슷하다

```
s.calculate();
```

s : 데이터, calculate() : 연산자

메서드의 내장 변수 this에 저장됨.

인스턴스 주소를 메서드에 전송

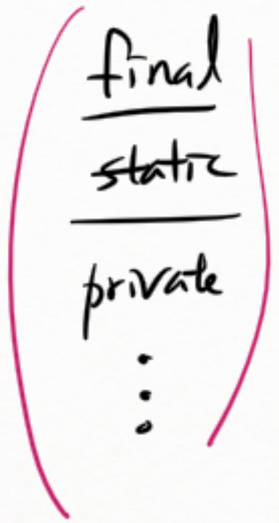
* 클래스 정의

데이터를 저장할
메모리 구조 정의 ⇐ 인스턴스 필드 선언

+

새 데이터 유형을 다룰
연산자 정의 ⇐ 메서드 정의

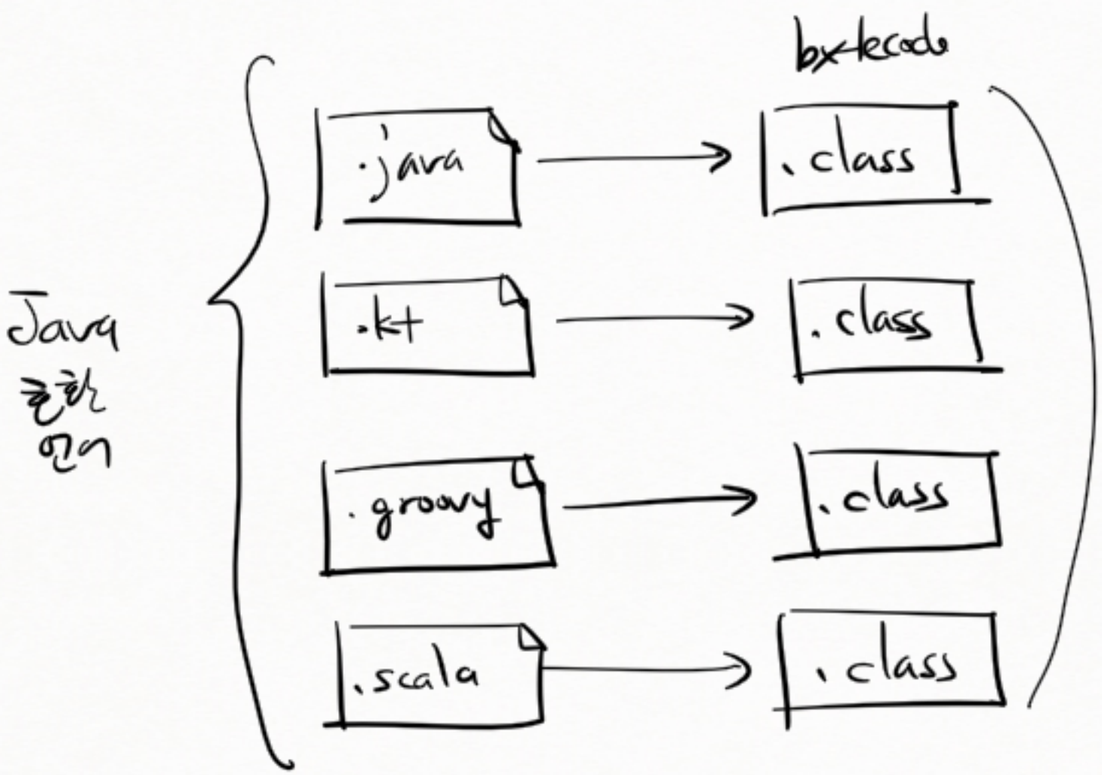
* modifier

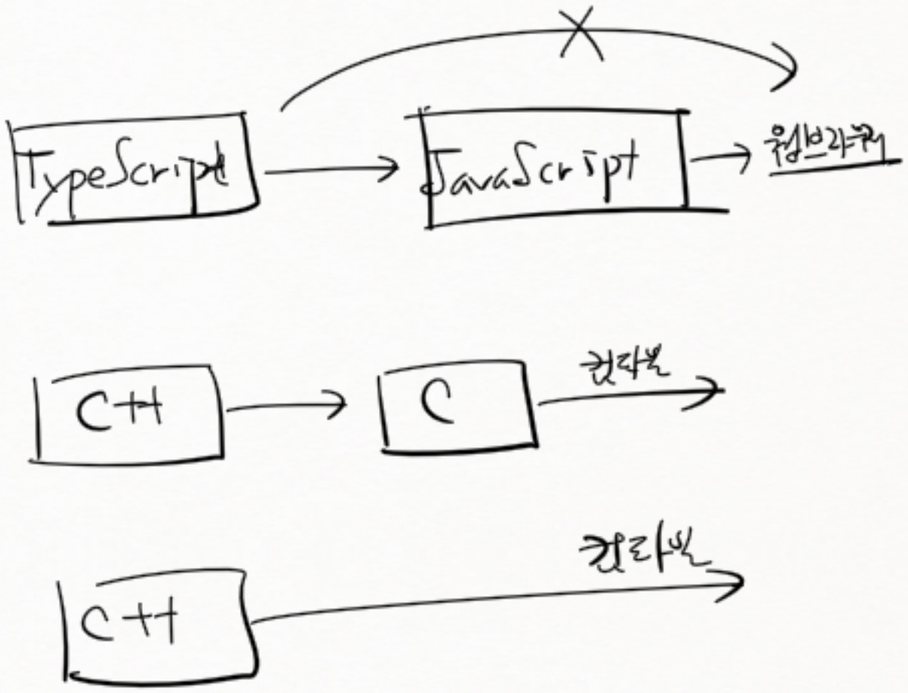


int i = 100;

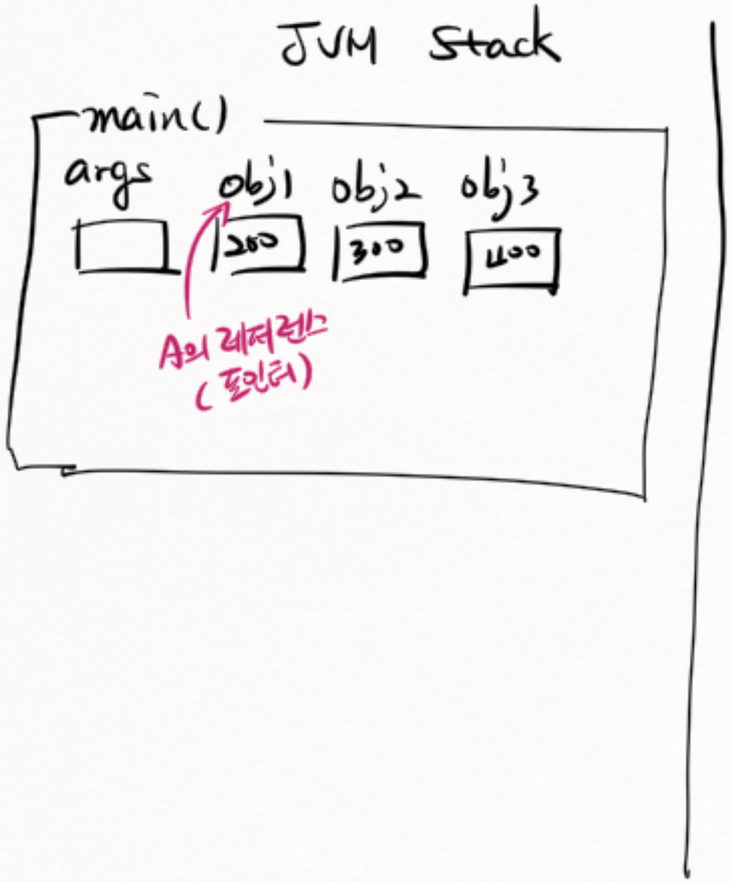
← 앞에 누것은 참조하느냐에 따라-
뒤에 선언한 변수(또는 메서드, 클래스)의
성질(특성)을 바꿀 CL
||
변경을 가하는 명령 (modifier)
||
변경자/한정자/제한자-

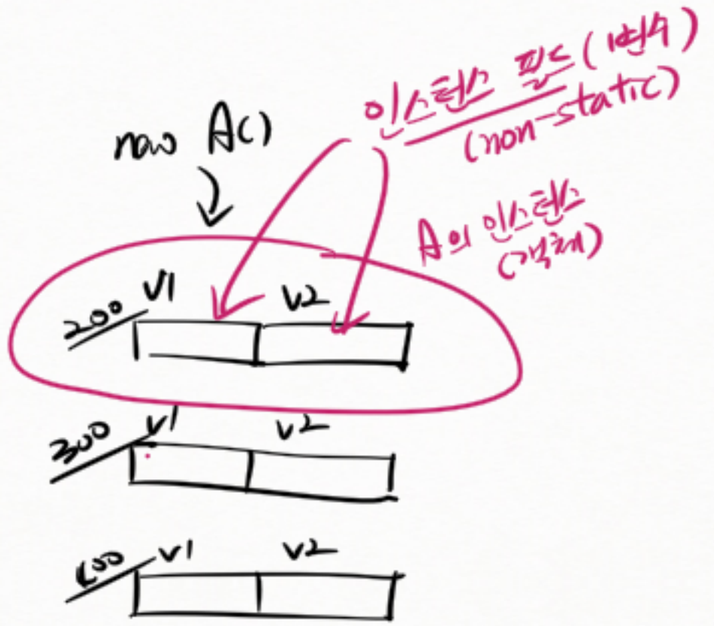
* 자바 호환 언어


bytecode
Java
호환
언어
.java
.class
.kt
.class
.groovy
.class
.scala
.class


TypeScript
JavaScript
컴파일러
C++
C
컴파일러
C++
컴파일러

* 인스턴스 변수(필드) : -oop.ex03.Exam0110
non-static

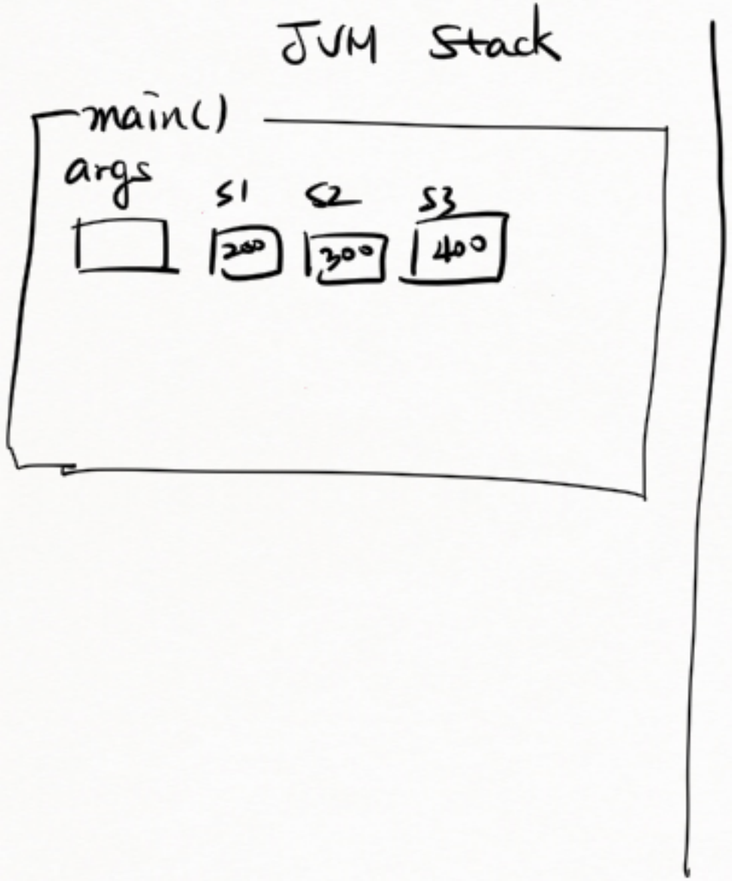


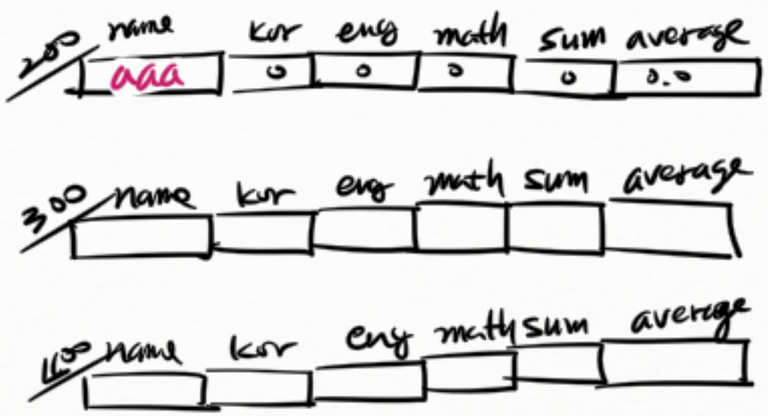


* 인스턴스 변수(필드) : -oop.ex03.Exam0120
non-static

non-static = 인스턴스 변수

new Score()
↳ Score 설계도의 선언된 변수들을 Heap에 준비한다





s1.name = "aaa"
200
↑
인스턴스 주소

* 클래스 변수(필드) : -oop.ex03.Exam0130

(static)

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

① JVM으로 Exam0130 클래스 실행

Method Area

② Exam0130 클래스 파일을 메모리에 올린다 (loading)

Exam0130.class

③ main() 호출

④ 로컬변수 준비

JVM Stack

main()

args

Heap

⑤ 클래스 로딩

A.class

⑥ A의 스태틱변수 준비

⑦ A.v1 = 100;

⑧ A.v2 = true;

v1 v2

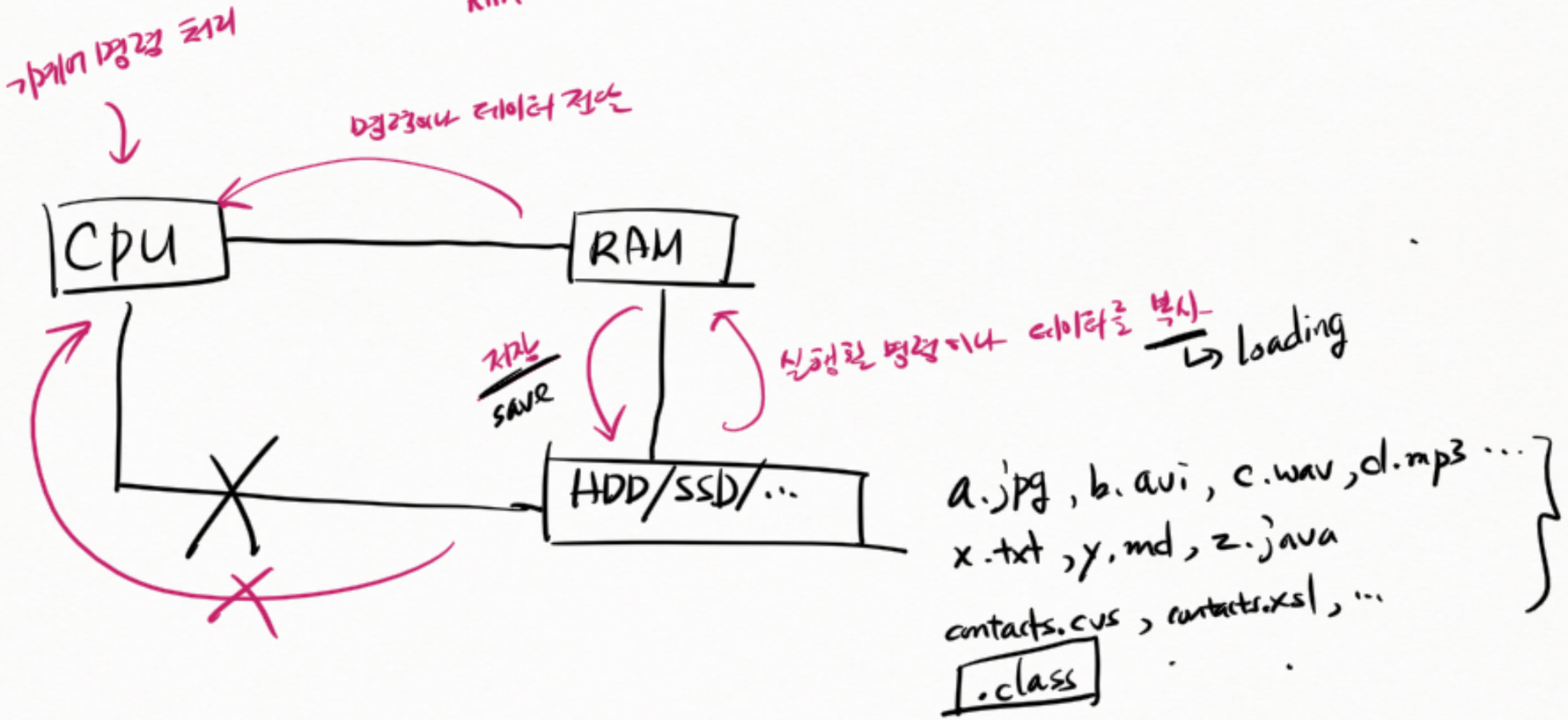
100 false true

* 클래스 로딩(loading)

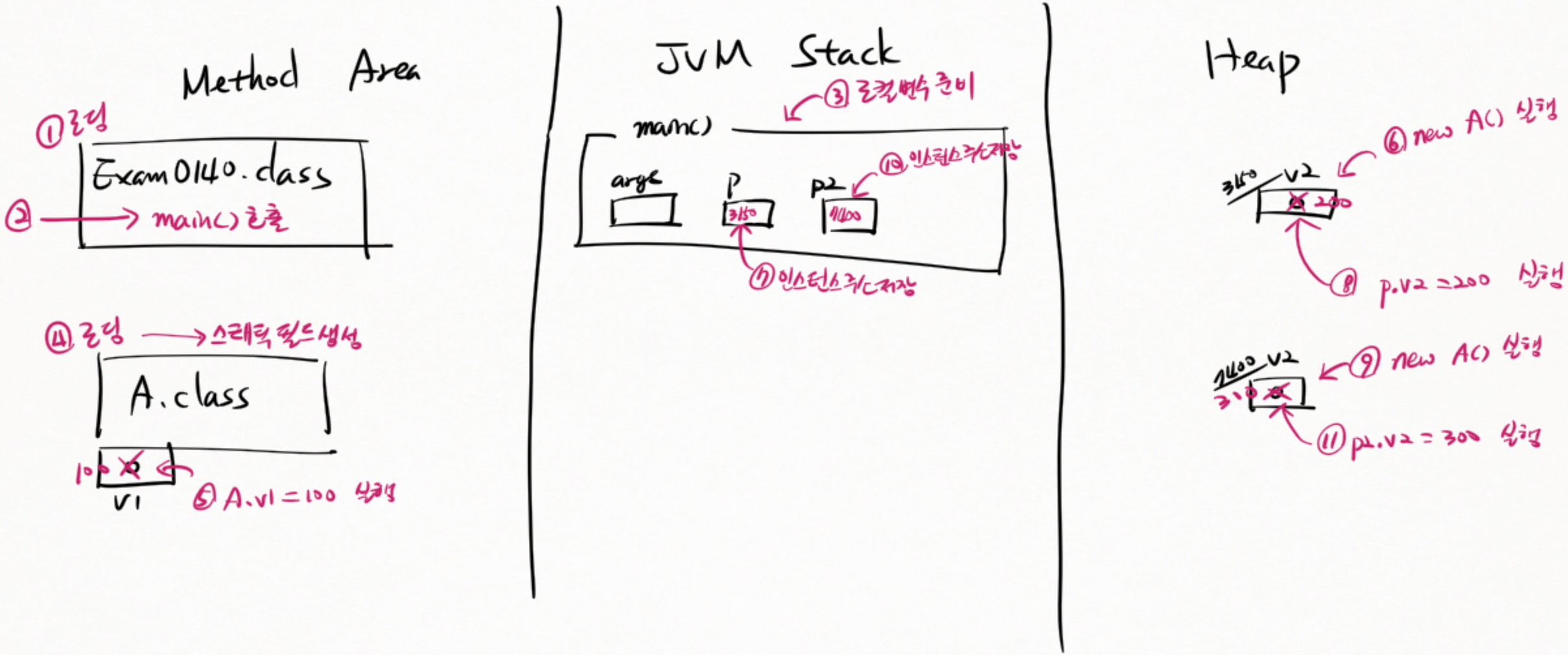
↳ 비휘발성장치(HDD, SSD, CD-ROM, USB등)에 보관된 .class 파일 읽어서

메모리에 복사하는 것

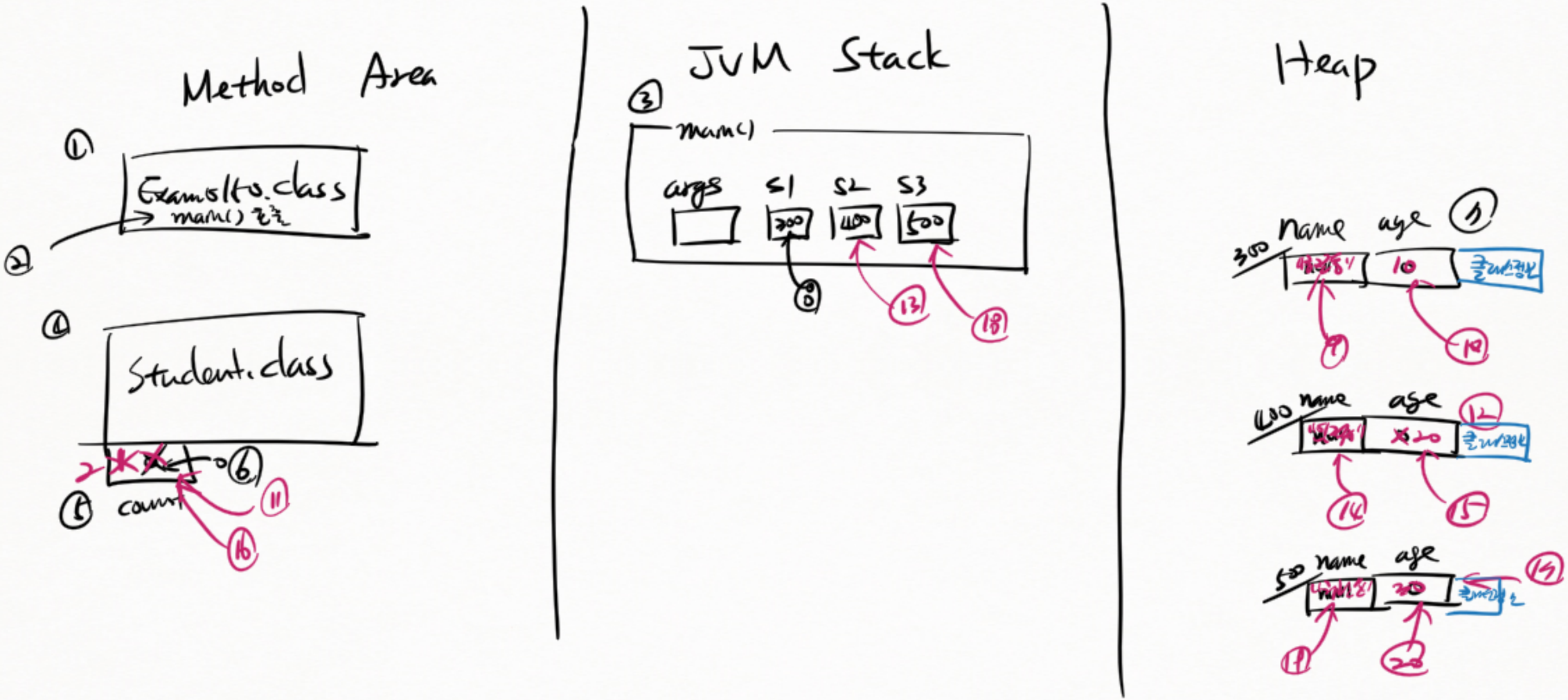
↑ RAM



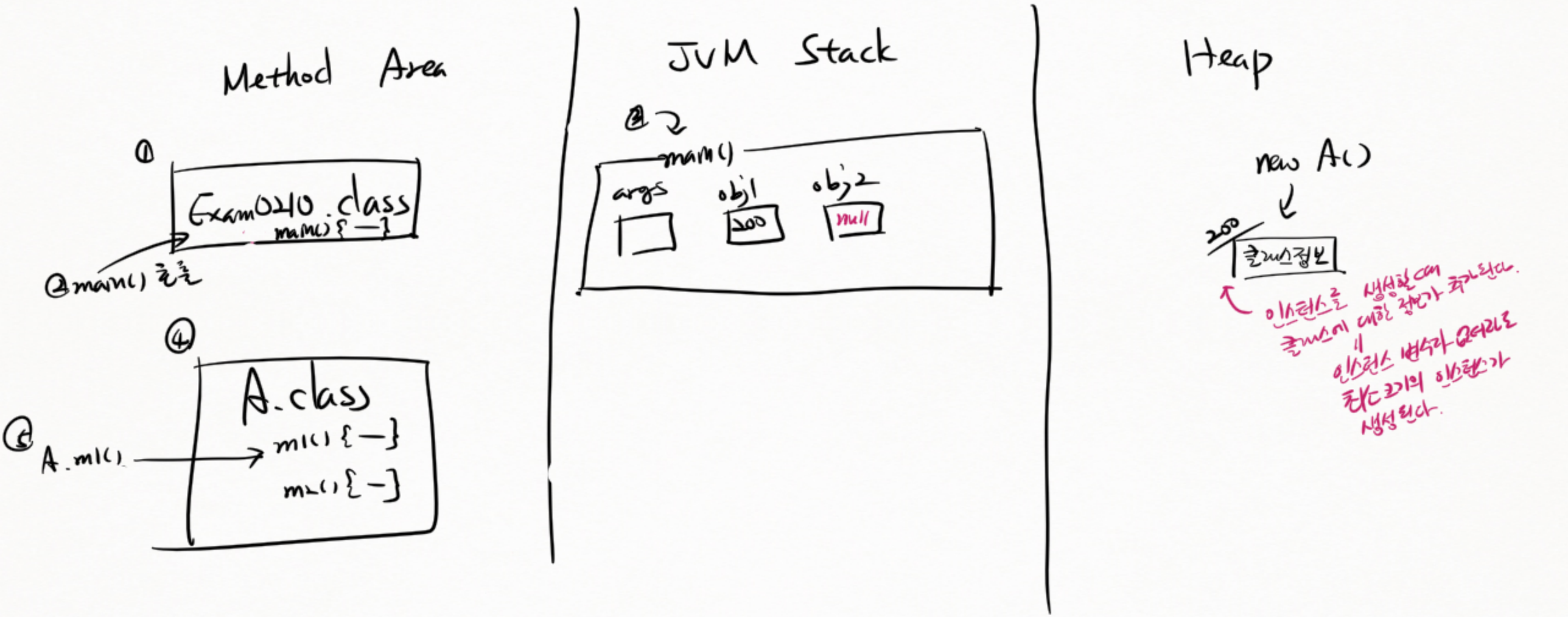
* 클래스 변수 와 인스턴스 변수 : oop.ex03.Exam0140



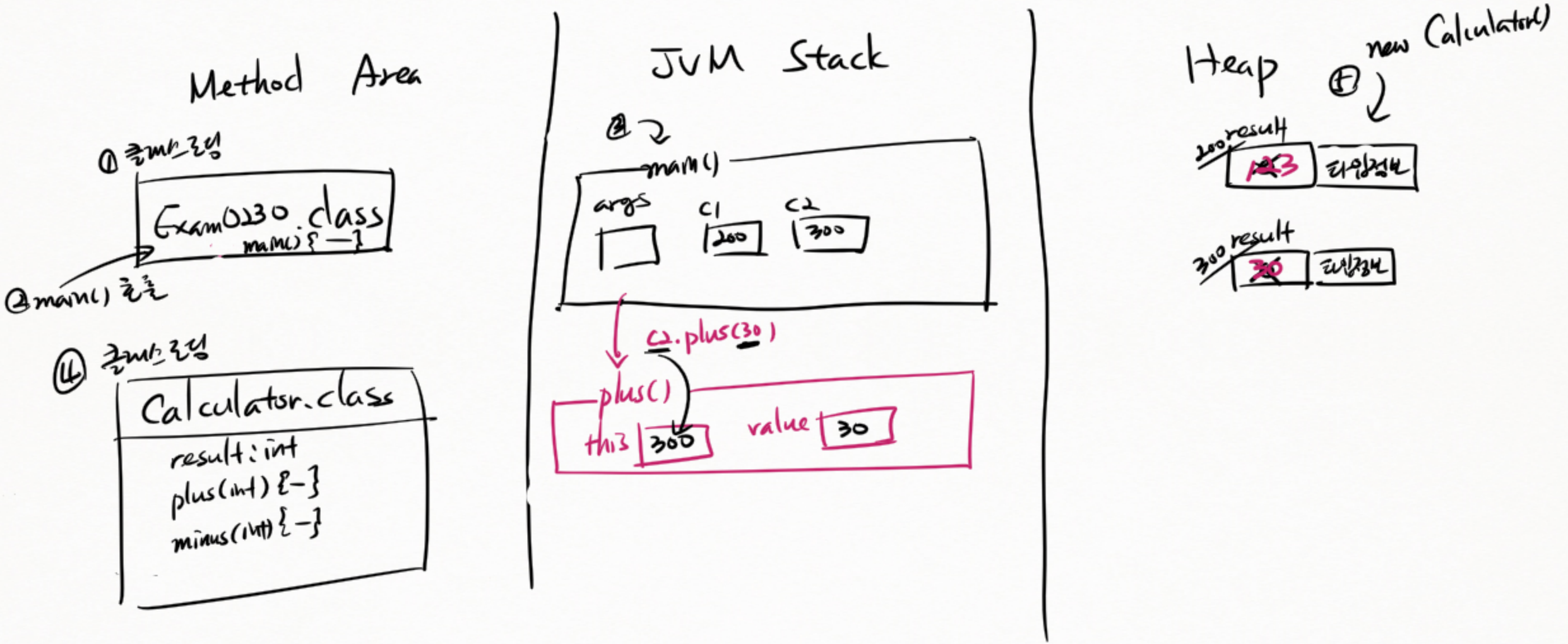
* 클래스 변수 와 인스턴스 변수 : oop.ex03.Exam01to



* 클래스 메서드와 인스턴스 메서드 : oop.ex03.Exam0210



* 인스턴스 멤버(변수,메서드) : oop.ex03.Exam0230



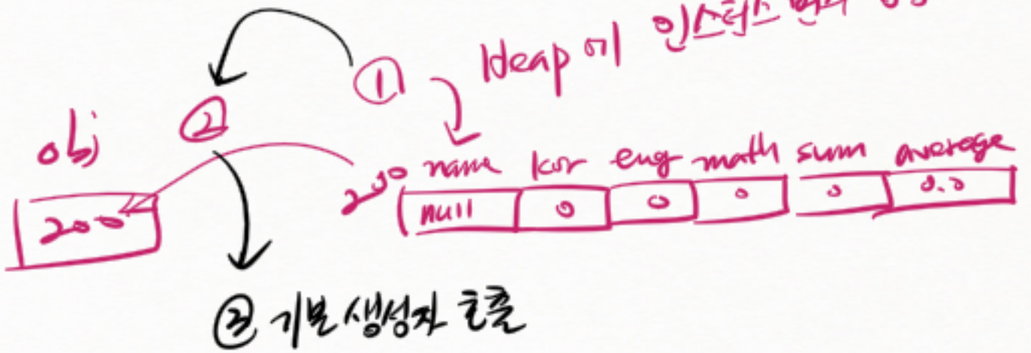
* 생성자

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

타입 ← primitive type / class / interface / enum

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

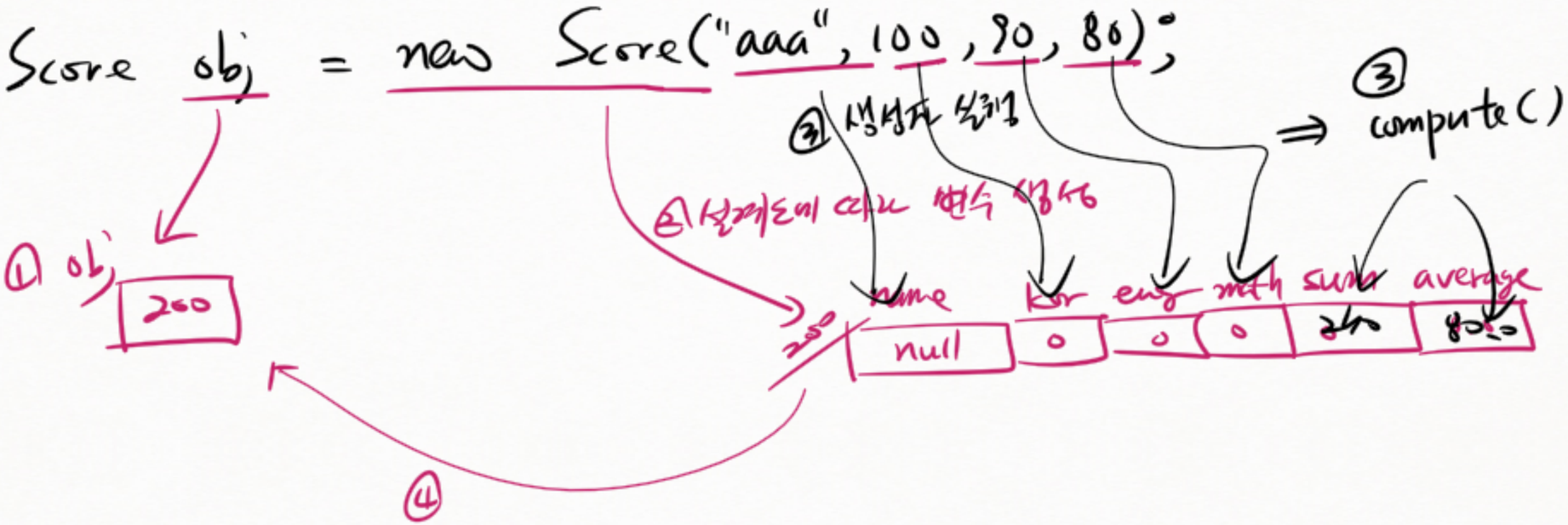
- Score obj : 레퍼런스 (= 포인터)
- Score() : 클래스명 / 자동으로 호출될 생성자를 지정.



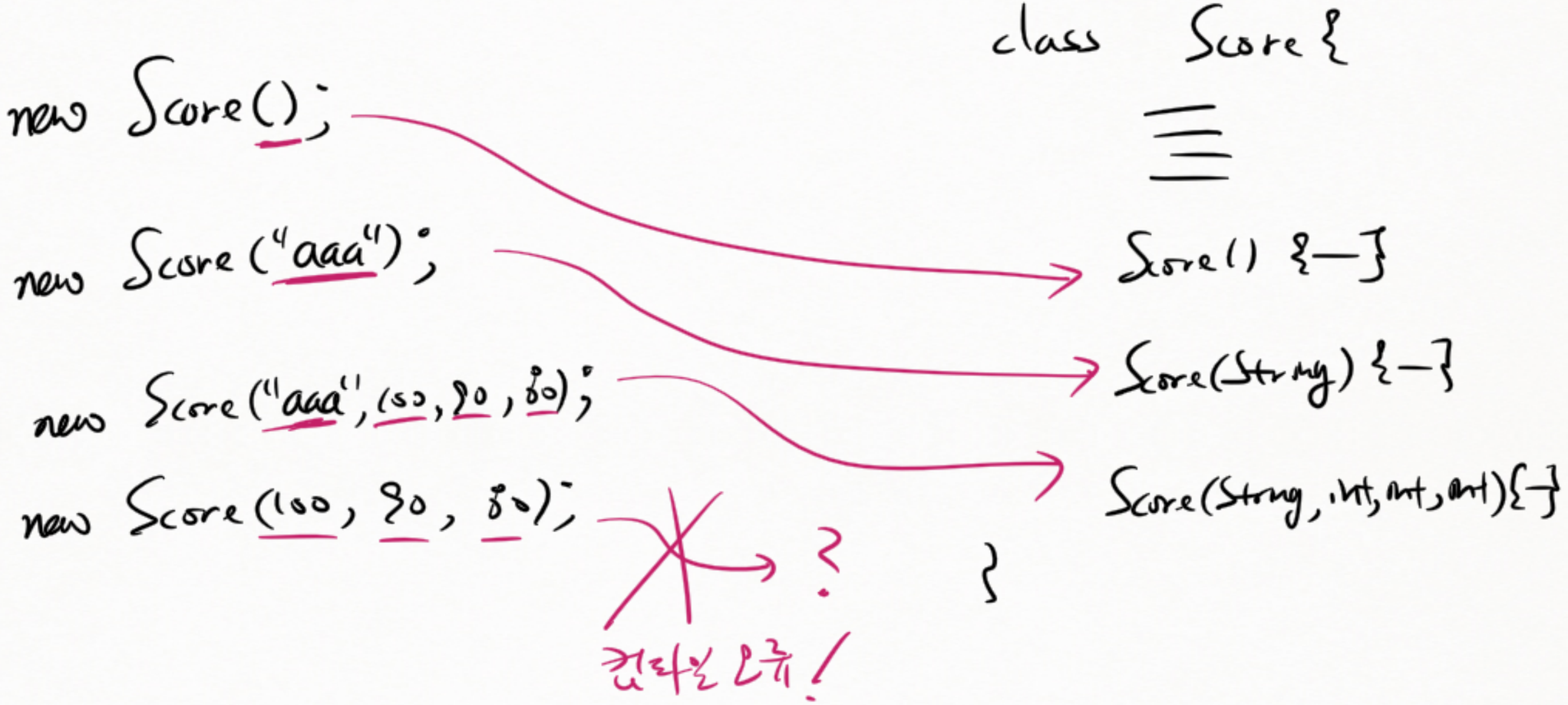
× Score() { }

생성자(constructor) ↓ 파라미터가 없는 생성자 "default constructor"

* 생성자 활용 ⇒ 인스턴스변수를 초기화시킨다.



* 호출될 생성자를 결정하는 방법



oop.ex03.Exam0910

Method Area

① 로딩

Exam0691.class

② main()호출

④ 로딩

A.class

⑥ a=4 실행

⑤ 스태틱변수 생성

a

⑬ a = a + B.b

1 2

30

⑧ 로딩

B.class

⑩ b=22 실행

⑨ 스태틱변수 생성

b

⑫ b = b + A.a

22 1

21

③ 프레임준비

JVM stack

main

args

console 창

⑦ A.static{}

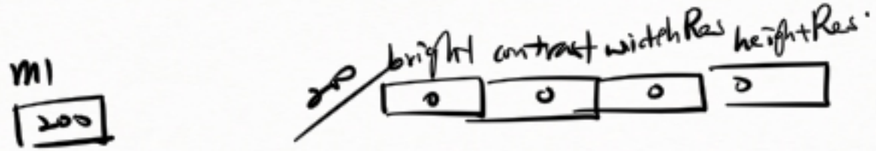
출력

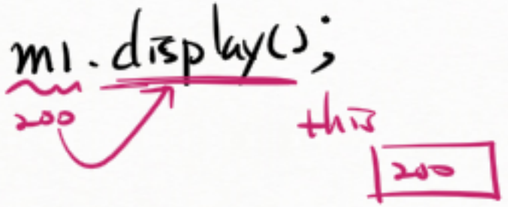
static 블럭실행

⑪ B.static{}

출력

new Monitor()


m1
200
200
bright
contrast
widthRes
heightRes
0
0
0
0


m1.display();
200
this
200

* String 클래스의 생성자

```
String s0 = new String();
```

기본 생성자 (default constructor)


```
System.out.println(s0);
```

```
String s1 = new String("Hello");
```

JVM이 사용하는 (charset) Unicode 계열 (UTF-16)

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

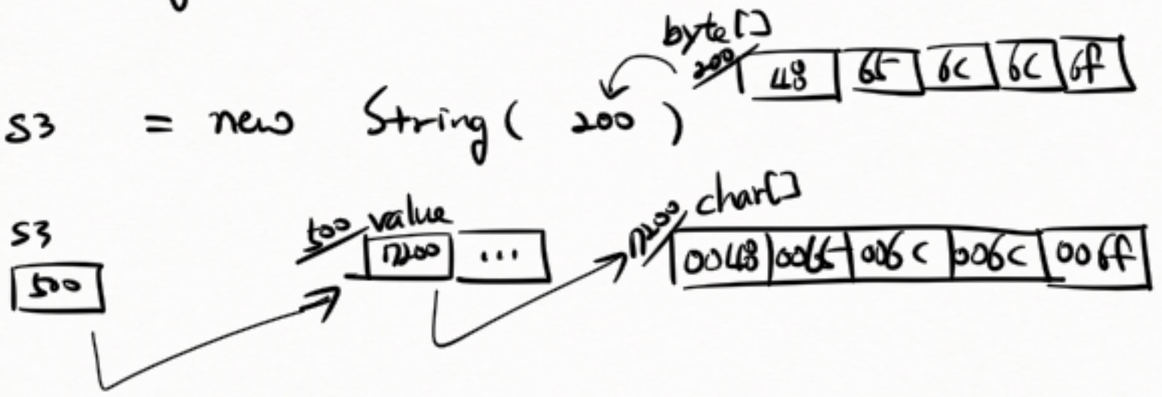
Java 9 부터는 byte[] 배열에 문자코드를 저장한다 (Unicode (UTF-16))

⋹

Java 8 까지는 char[] 에 문자코드를 저장했다 (Unicode (UTF-16))

* String 클래스의 내부구조

String s3 = new String(200)



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

* JVM 의 default characterset
UTF16 ↳ JVM 이 외부에 문자코드를 내보낼때 사용하는 charset
// 외부에서 문자를 가져올때 가정하는 charset (문자변환 규칙)

* String 클래스의 생성자 : oop.ex01.Exam0112

실행 환경 : Eclipse에서 실행

↓

JVM이 대부 문자코드를 읽을때 UTF-8이라 가정한다

"가족똘똘"의 EUC-KR 코드 실제 문자코드

byte[]

b0	a1	b0	a2	b6	ca	bc	cb

문자		EUC-KR
가	→	B0A1
갘	→	B0A2
똘	→	B6CA
똥	→	B6CB

```
String s = new String( ⇓ );
```

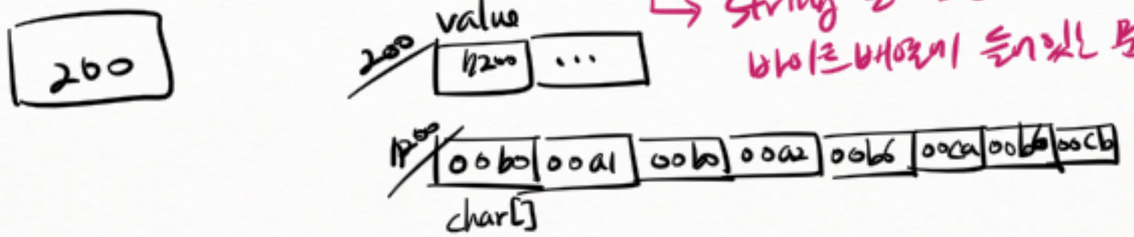
↳ String 클래스가 바이트배열이 들어있는 문자코드가 UTF-8 이라고 착각한다.

⇓ 문제발생!

잘못된 Unicode로 변환

⇓

한글이 깨진다.




```
System.out.println(s);
```

'가' → B0A1 (EUC-KR)
'가' → AC00 (UTF-16)
'가' → EAB080 (UTF-8)

* String 클래스의 생성자 : oop.ex01.Exam0112

실행 환경 : Eclipse에서 실행

JVM이 디폴트 문자코드를 읽을때 UTF-8이라 가정한다

"가운데줄" 의 EUC-KR 코드 실제 문자코드

byte[]

b0	a1	b0	a2	b6	ca	bc	cb

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

byte[] 배열에 들어있는 값이 어떤 문자 집합의 규칙을 따르는지 정확하게 지정하면 UTF16으로 정상적으로 변환한다

s → 200

200 → value → h200 ...

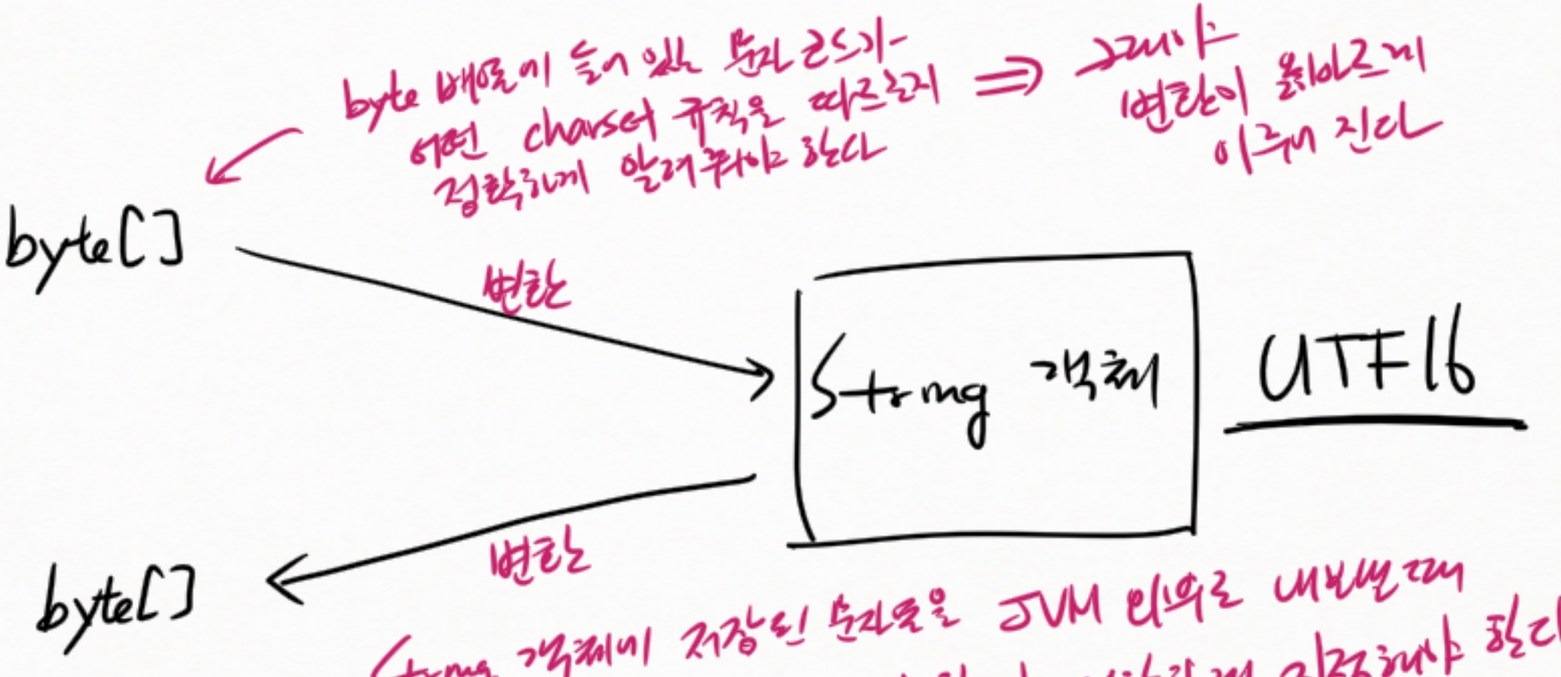
h200 → AC00 AC01 B618 B62C

char[]

EUC-KR ⇒ b0a1 b0a2 b6ca bccb

UTF16 ⇒ AC00 AC01 B618 B62C

* Java 한글 처리 규칙!



String 객체에 저장된 문자열을 JVM 밖으로 내보낼 때 어떤 charset으로 변환할지 정확하게 지정해야 한다.
보통 웬만하면 UTF-8 을 권장한다.

① -Dfile.encoding 옵션이 없는 경우

↳ Windows : MS-949 ✓

Unix/Linux : UTF-8

"ABC가각" (한글 1글자+α)

EUC-KR, UTF-8, UTF-16, ISO-8859-1

(한글 2글자) (한글 X)

byte[]

EUC-KR

b0	a1	b0	a2	b6	ca	b6	cb

(1)

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

MS-949 (EUC-KR+α)

String s = new String( );

UTF-16 ⇒ 스트링 객체

char[]

(2)

JVM이 디폴트에서 문자데이터를 읽거나 디폴트로 문자데이터 보낼때 사용하는 문자변환규칙

character set (문자집합)

UTF-16

'A' → 0041

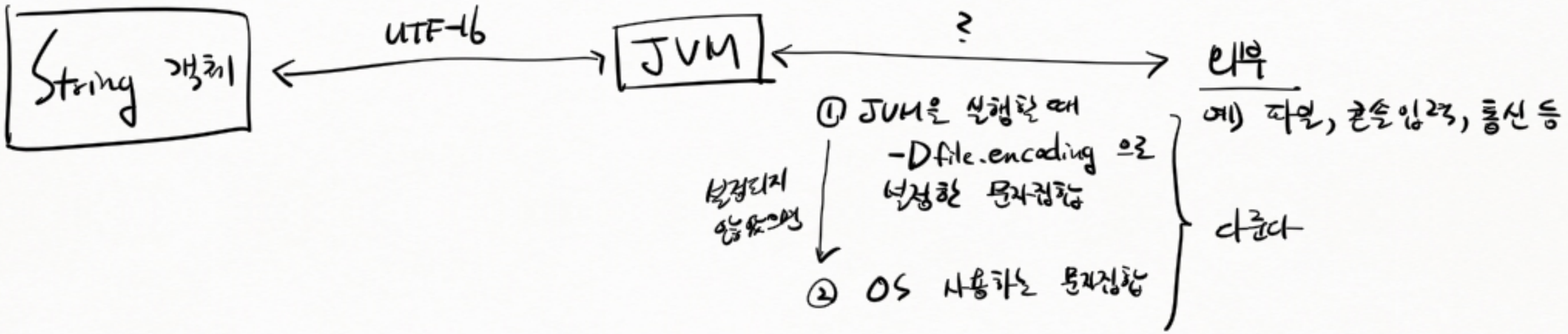
B → 0042

C → 0043

가 → AC00

각 → AC01

* JVM과 문자 집합


String 객체
UTF-16
JVM
?
외부
예) 파일, 콘솔입력, 통신 등
① JVM을 실행할 때
-Dfile.encoding 으로
변경한 문자집합
설정하지
않을경우
② OS 사용하는 문자집합
다르다

-Dfile.encoding=UTF-8 설정 안함

$ java -cp bin/main com

Windows (MS949)

MS949 → UTF16 (권장사용)

영어 한국

byte[] → new String()

영어 EUC-KR → O

트랜스 UTF-8 → X

Unix/Linux (UTF-8)

UTF-8 → UTF-16

트랜스 한국

byte[] → new String()

EUC-KR → X

UTF-8 → O

-Dfile.encoding=UTF-8 설정 함

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

Windows (MS949)

byte[] → new String()

EUC-KR → X

UTF-8 → O

Unix/Linux (UTF-8)

byte[] → new String()

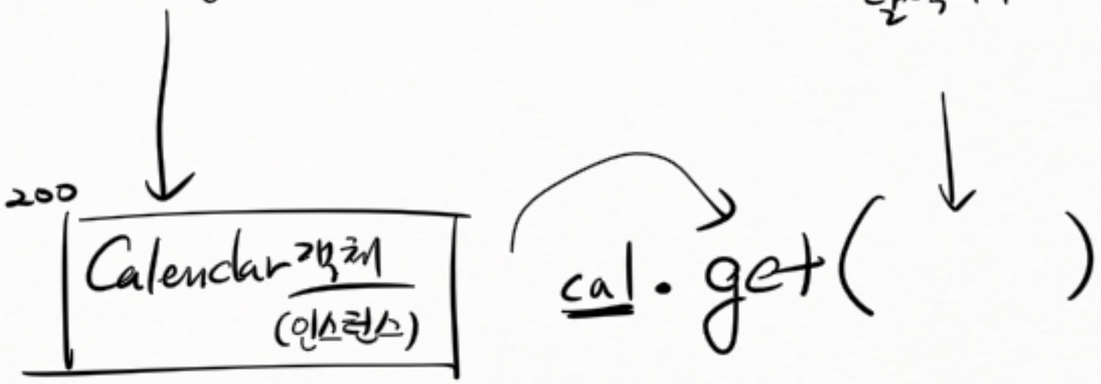
EUC-KR → X

UTF-8 → O

* Calendar 사용법

Calendar cal = Calendar.getInstance()

200



cal.get(1) → 년을 알 수 있는 값을 반환

cal.get(Calendar.YEAR) → 년도

값의 아이디를 앞으로 한다

그리고 스태틱 상수의 미리 그 번호를 저장해 두었다.

상수를 먼저이기 때문에 이해하기 쉽다.

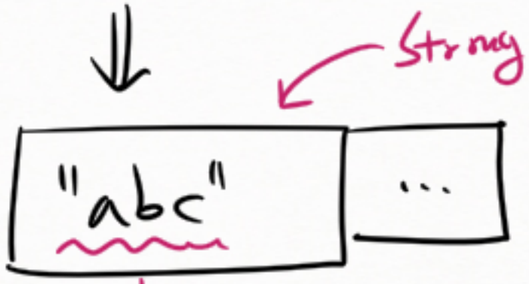
* String : compareTo()

s1.compareTo(s2)
this

Helli
Hello ← s1
Hellg

* Immutable 객체 of String

new String("abc")



불변의 객체이다 = 처음 객체를 만들면 그 객체의 값을 변경할 수 없다!
(immutable)

* primitive type 과 wrapper 클래스

int ⟶ java.lang.Integer

byte ⟶ 〃 .Byte

short ⟶ 〃 .Short

long ⟶ 〃 .Long

float ⟶ 〃 .Float

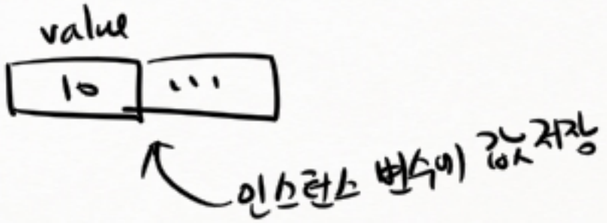
double ⟶ 〃 .Double

boolean ⟶ 〃 .Boolean

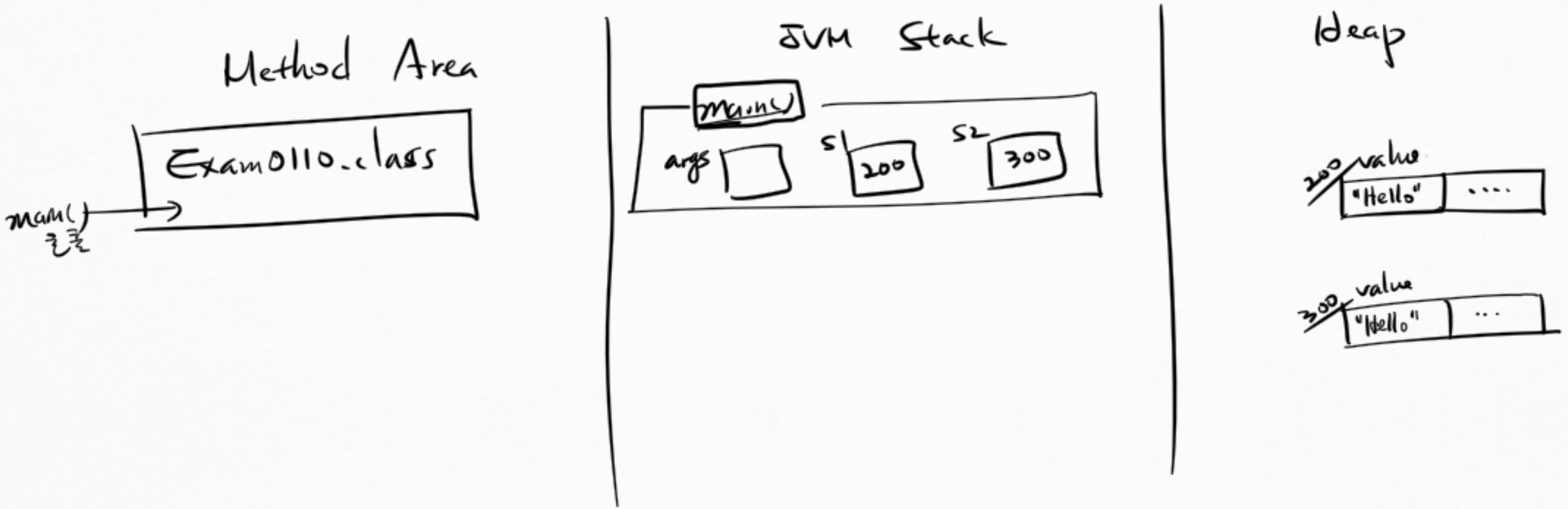
char ⟶ 〃 .Character

(값을 다룰수 있는 다양한 메서드 제공) ⟶ Wrapper

int ⟶ Integer

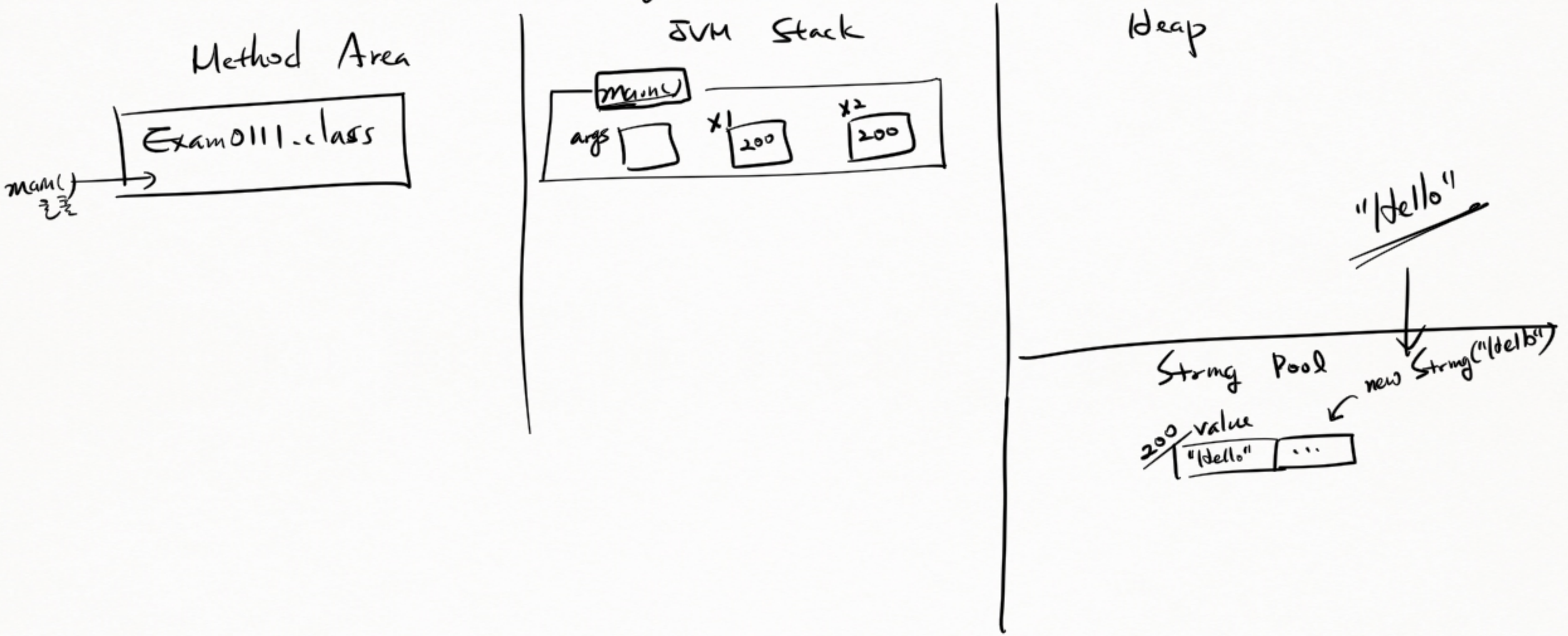


* lang.ex02.Exam0110 - String 클래스 사용하기



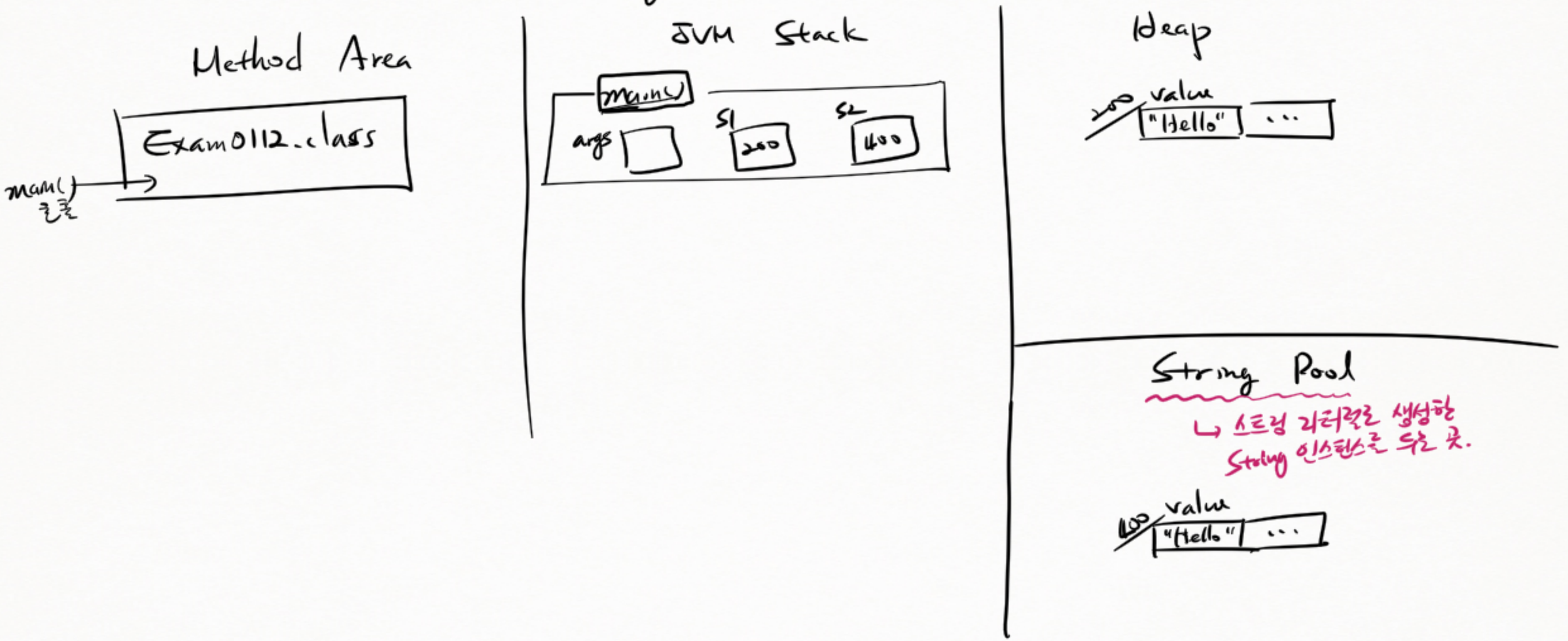
* String 클래스 사용법 – 스트링 리터럴로 인스턴스 생성하기

lang.ex02.Exam0111



* String 클래스 사용법 — new vs "—"

lang.ex02.Exam0112



* String 클래스 사용법 — intern()

lang.ex02.Exam0113