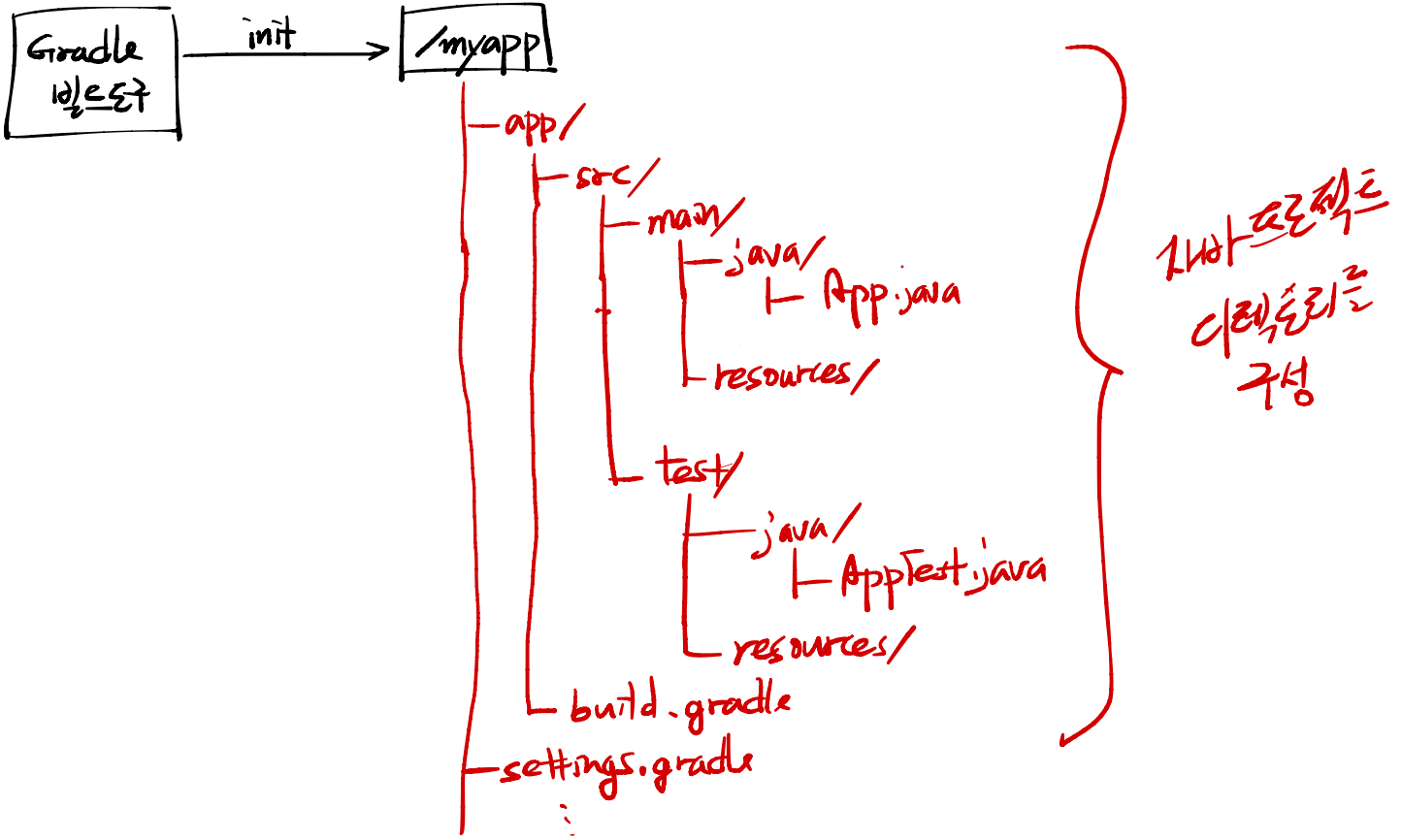
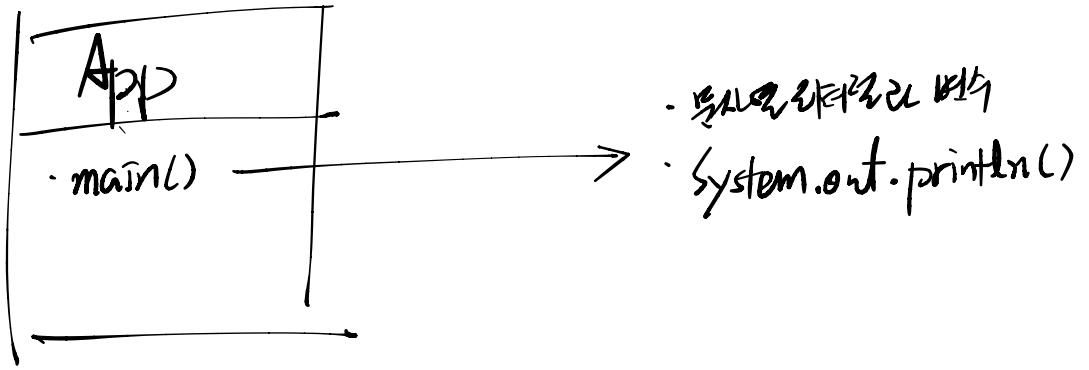
01. 자바프로젝트 준비


Gradle 빌드툴
init
/myapp
app/
src/
main/
java/
App.java
resources/
test/
java/
AppTest.java
resources/
build.gradle
settings.gradle
자바프로젝트 디렉토리 구성

02. 리터럴과 변수를 사용하여 문자열 출력하기



03. ANSI escape 코드를 사용하여 출력 문자열을 꾸미기

println("_____")

ANSI escape code

"\033[1m문자열\033[0m"

볼드체 설정 / 설정 해제

* gradle run --quiet

↑ 빌드 로그 출력을 지워 gradle 실행 과정을 간략하게 하는 명령

04. 키보드 입력 다루기

System.io

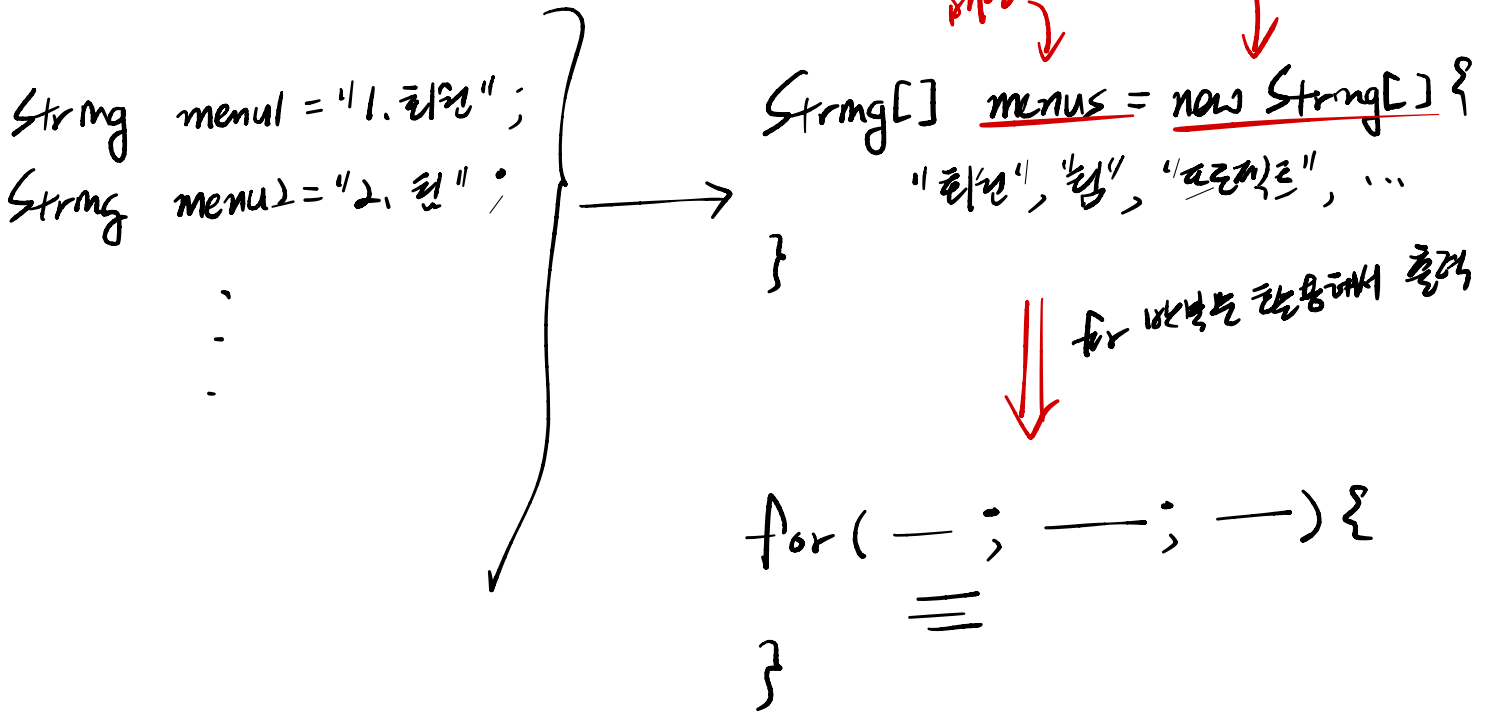
main() ⟶ · Scanner keyboard = new Scanner();

· keyboard.nextInt()

↳ next(), nextLine(), ...

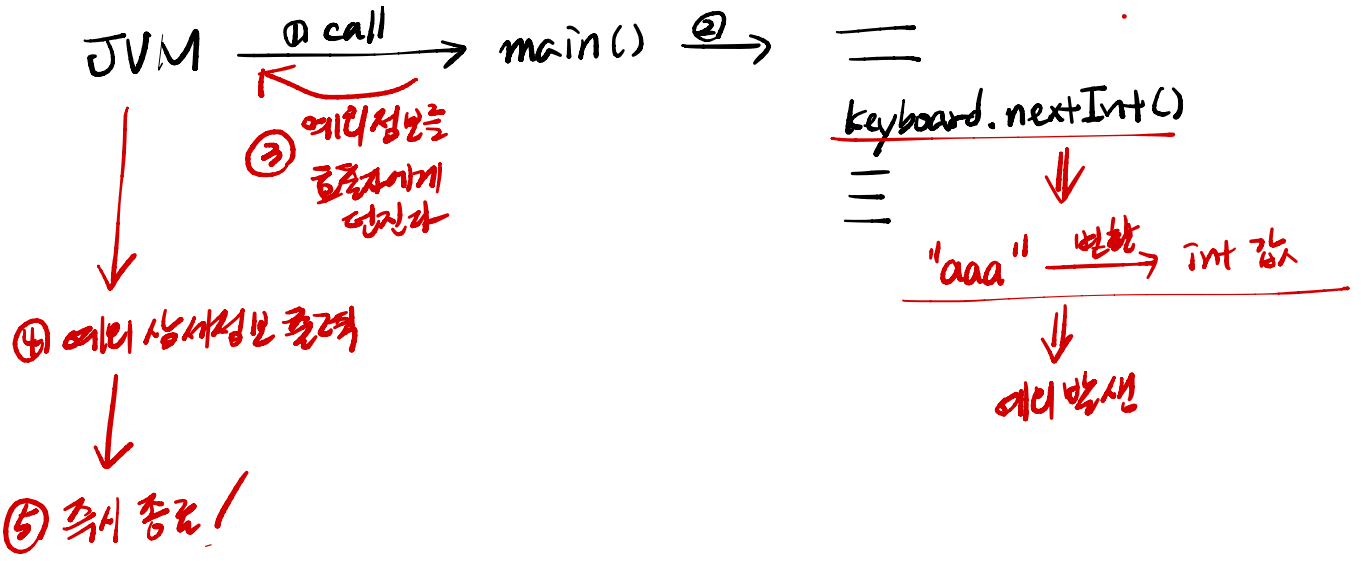
· keyboard.close();

05. 배열을 활용하여 메뉴목록 다루기



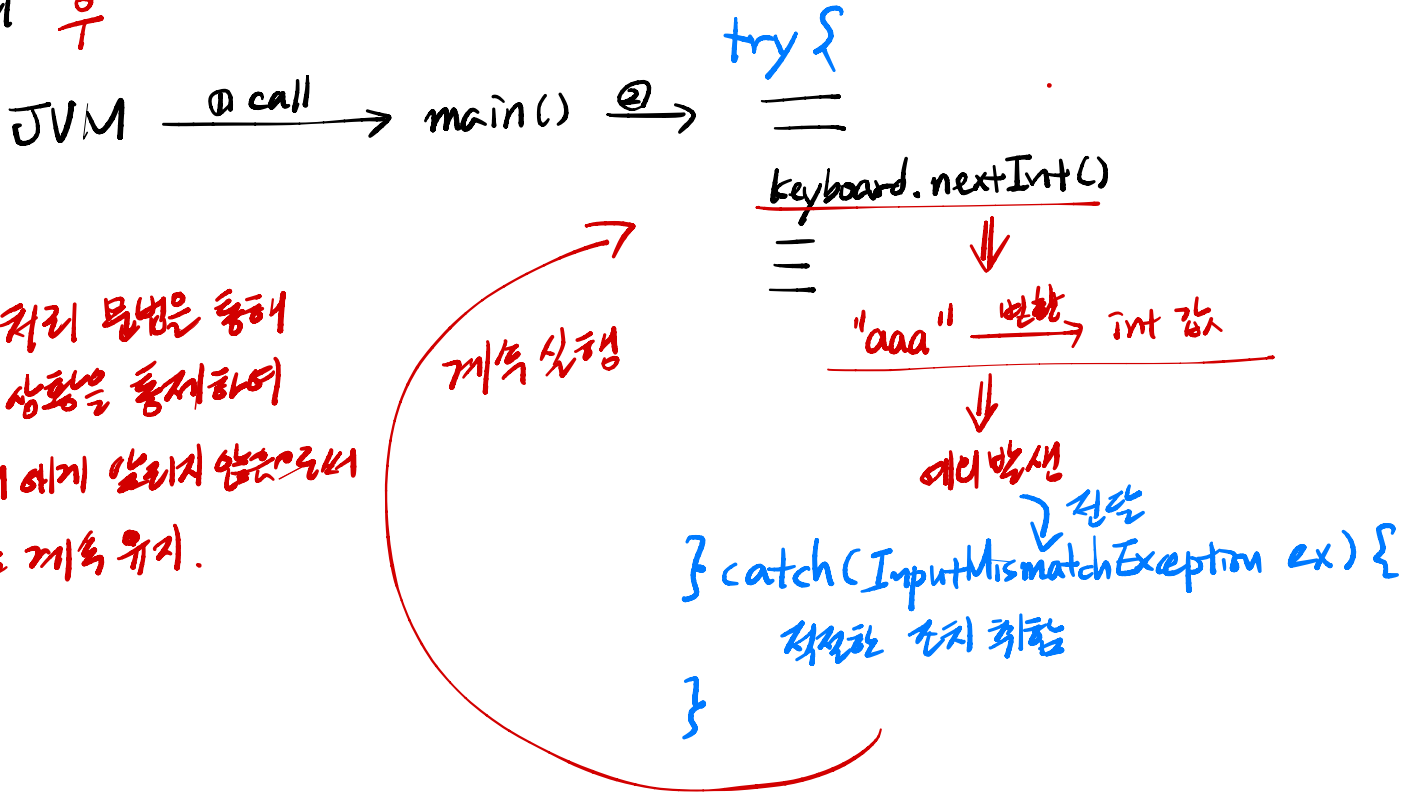
06. 예외처리

예외처리 전



06. 예외처리

예외처리 후



예외처리 문법은 통해
예외 상황을 통제하여
JVM 에게 알리지 않고서
실행을 계속 유지.

04. 문자열 비교와 데이터 변환 타입

```
int menuNo = keyboard.nextInt()
```

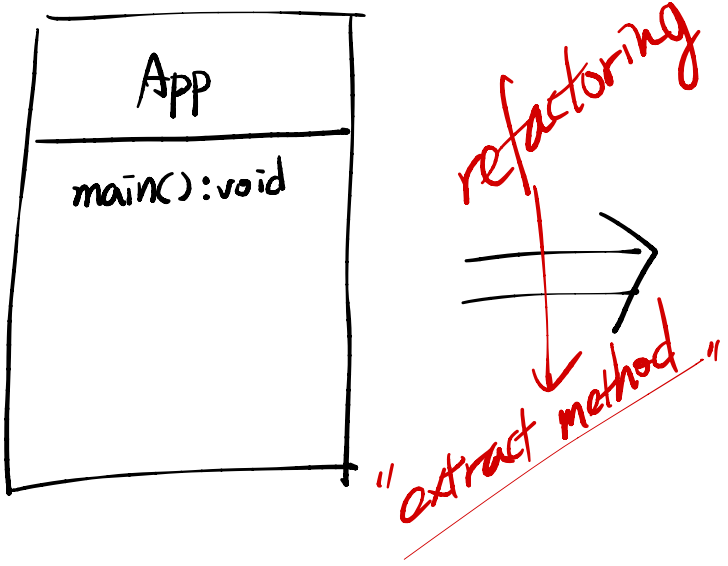
⇒

```
String command =
        keyboard.nextLine();
if(command.equals("menu")){
}
int menuNo =
    Integer.parseInt(command);
```

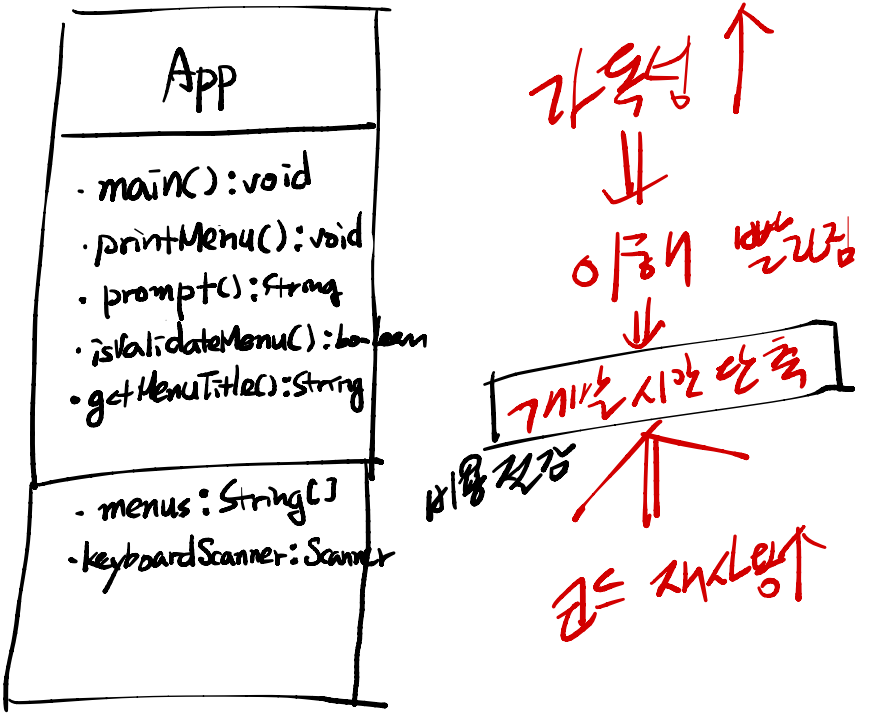
- `nextInt()` → 버퍼 확인
- `equals("menu")` ← 문자열 비교
- `"2"` (String 문자열) → 변환 → `2` (int 값)

08. 기능 단위로 메서드 묶기 : 메서드 사용법

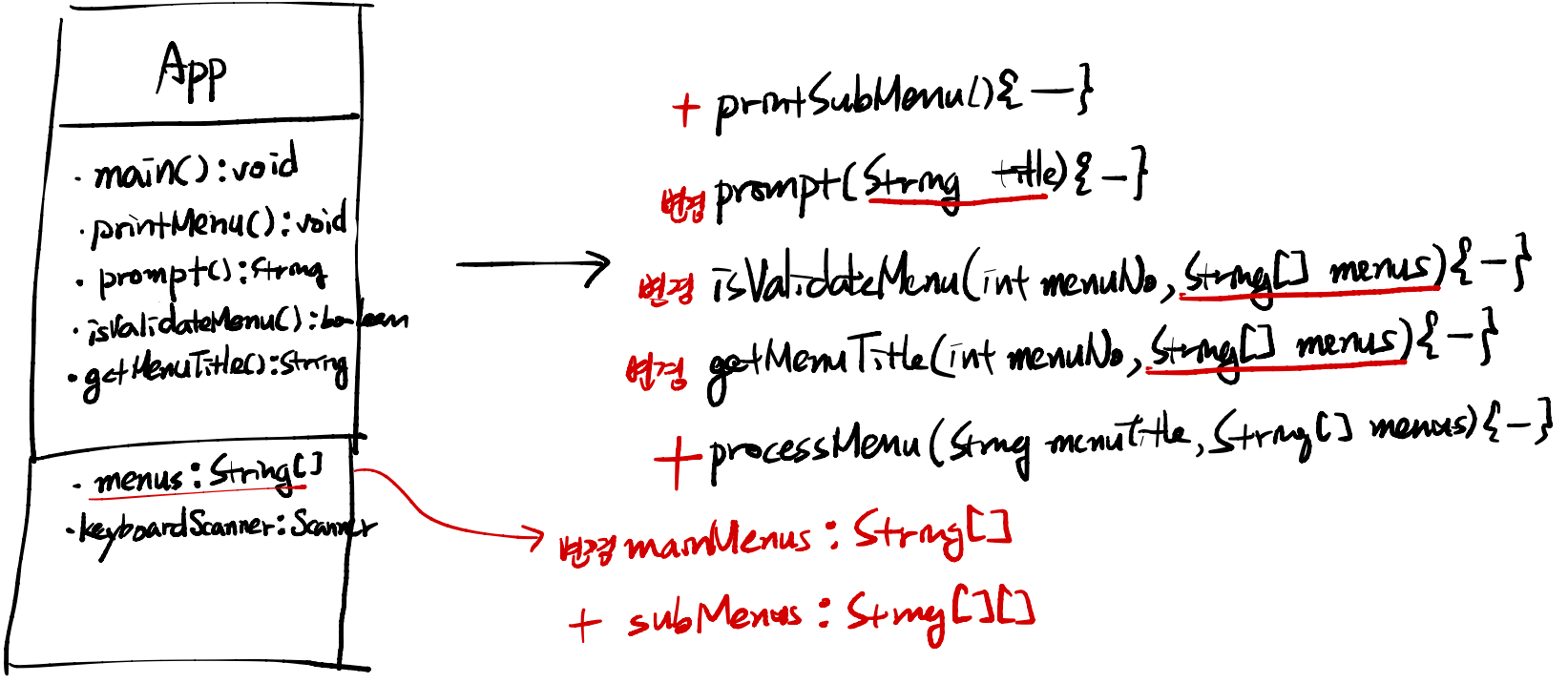
07.



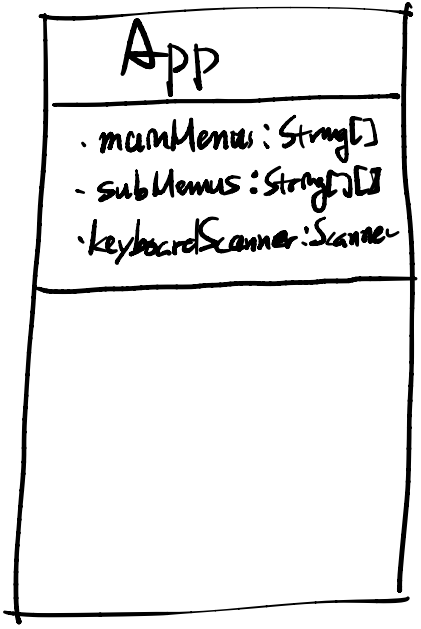
08.



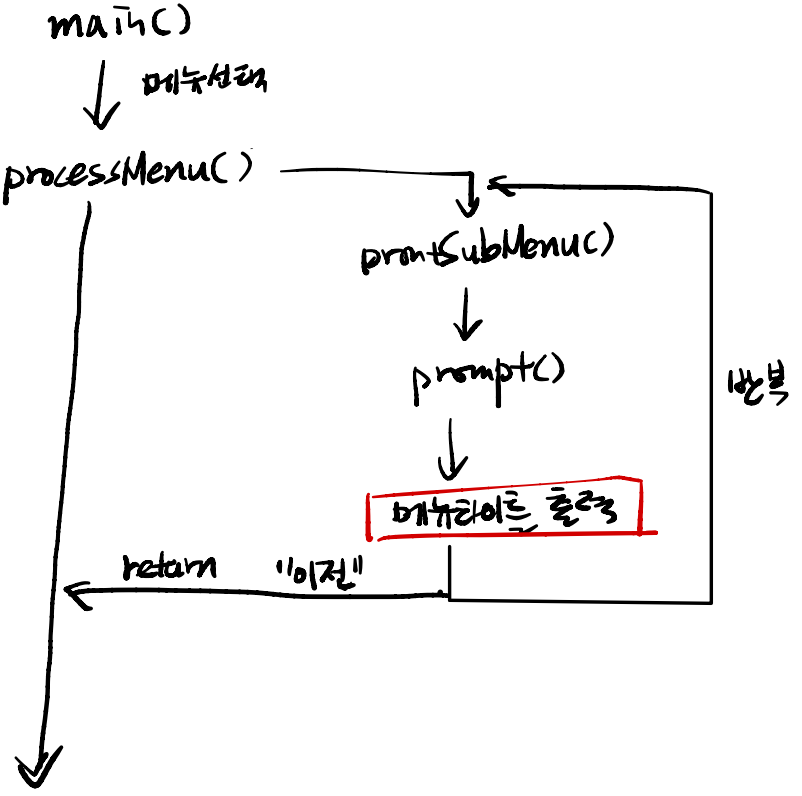
09. 자바 기본문법 활용 연습



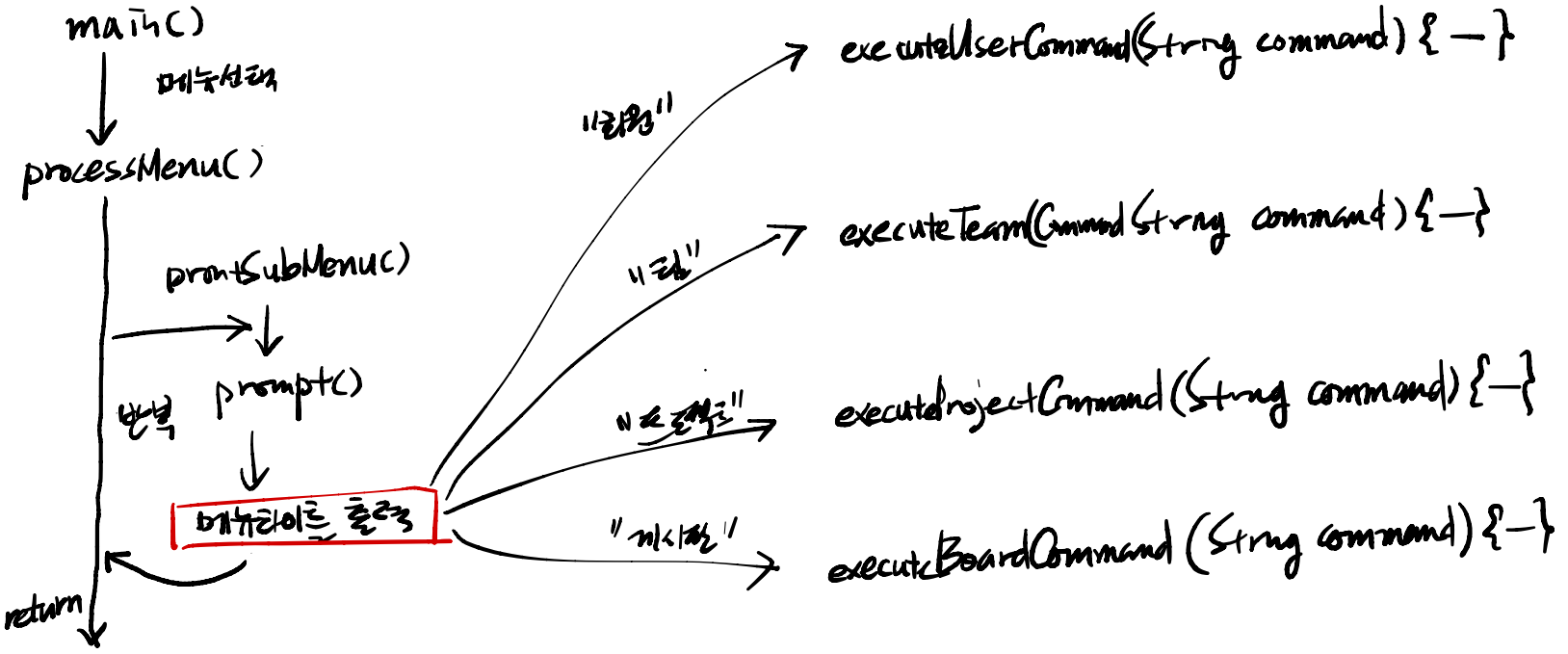
10. CRUD 구현하기



실행흐름

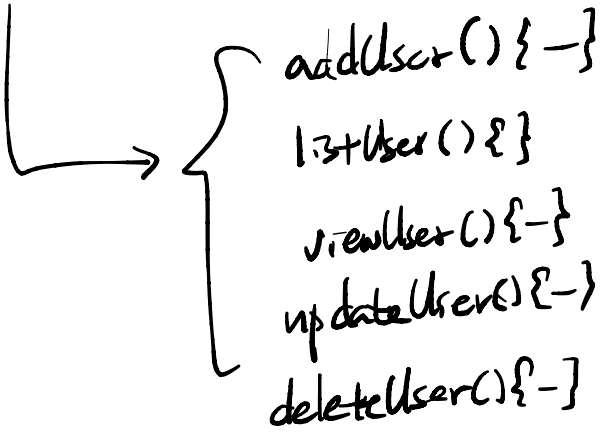


10. CRUD 구현하기 (계속)

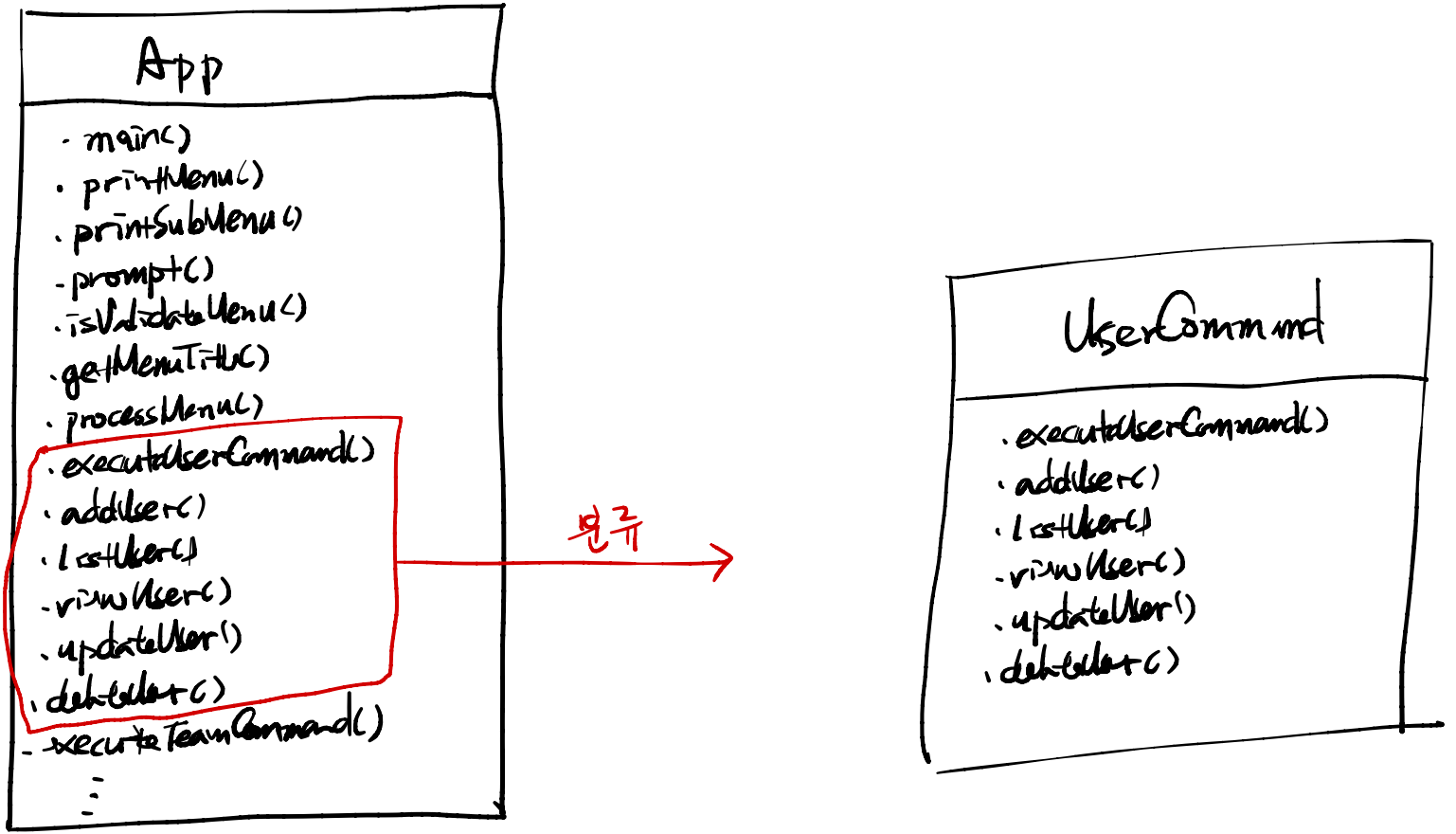


10. CRUD 구현하기 (계속)

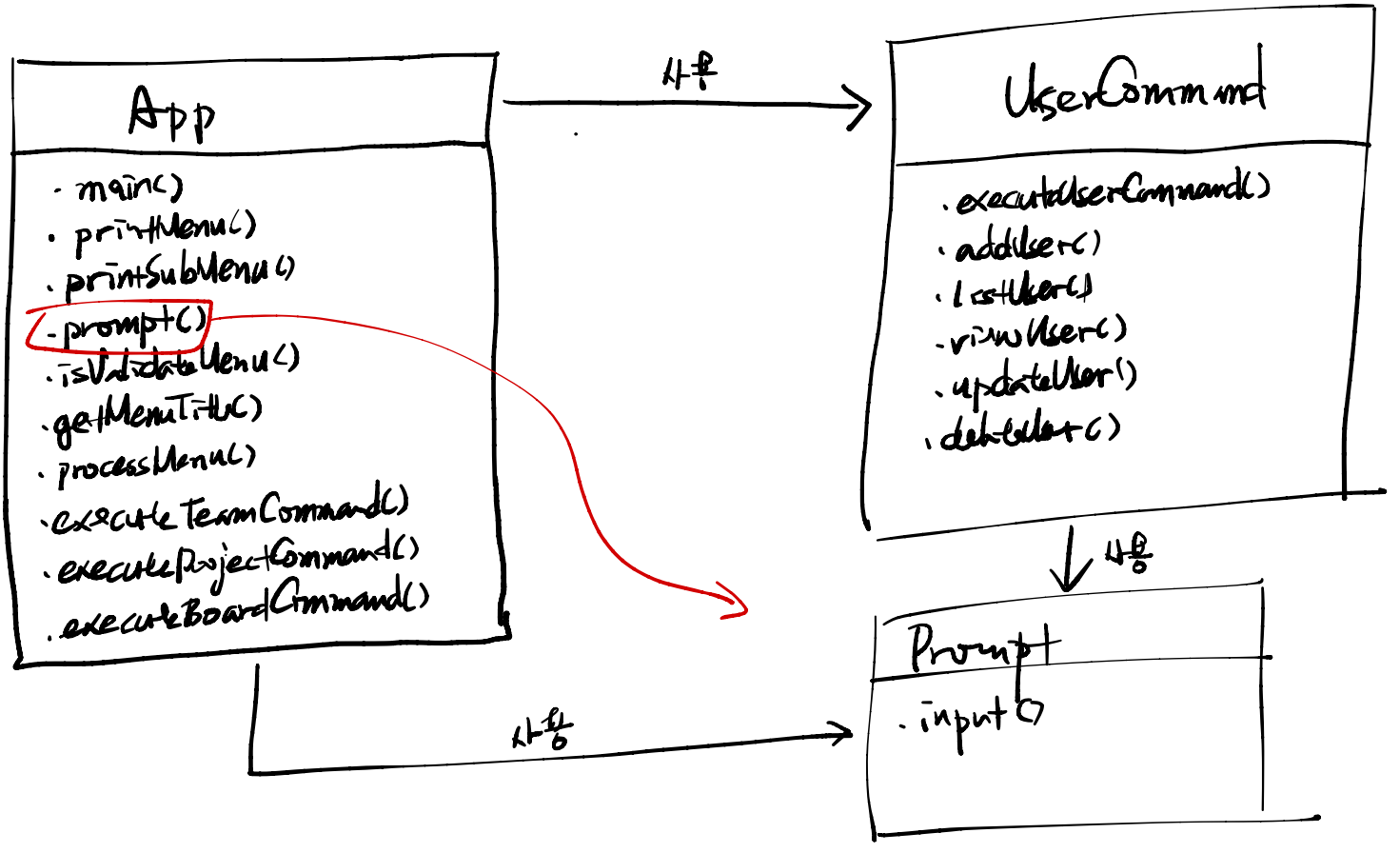
executeUserCommand(String command) { - }



10. CRUD 구현하기 (계속)

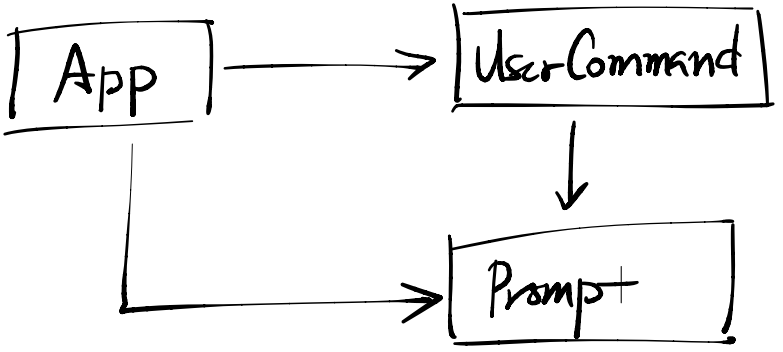


10. CRUD 구현하기 (계속)



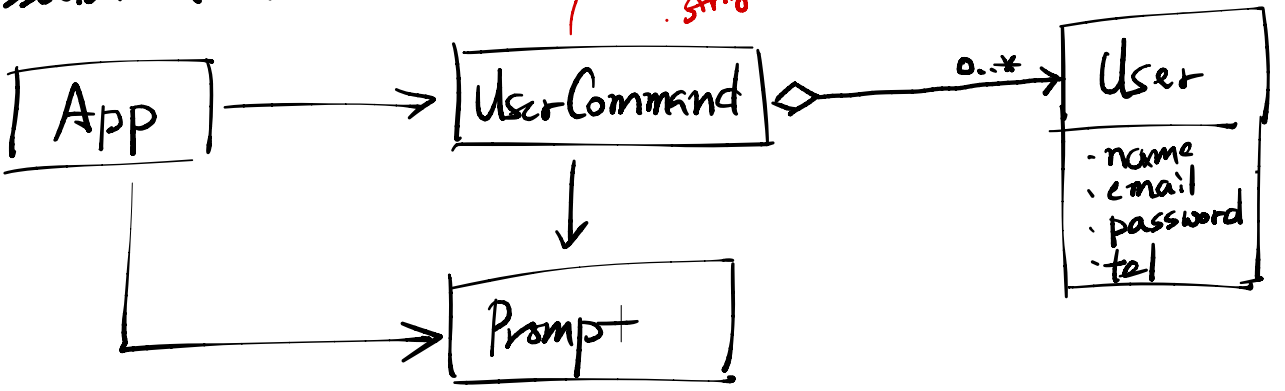
10. CRUD 구현하기 (계속)

* Association (연관)



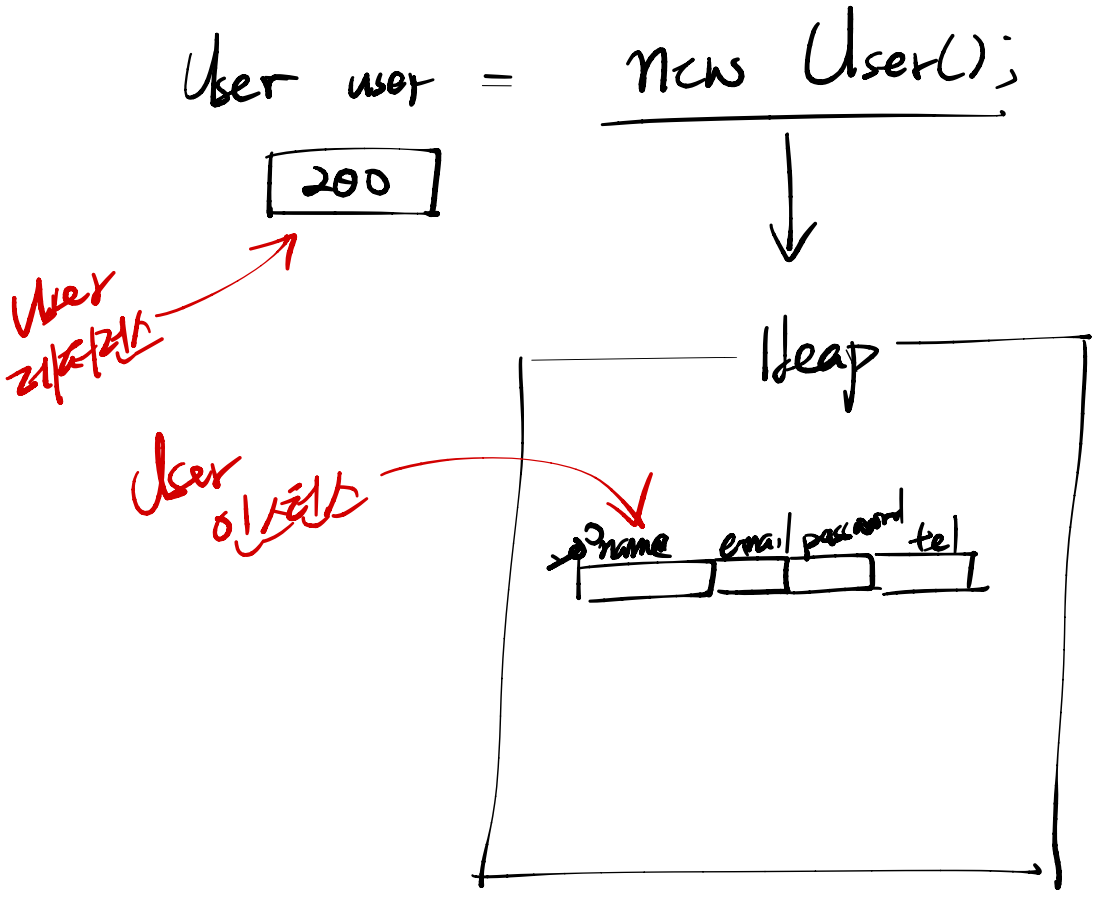
10. CRUD 구현하기 (계속)

* Association (연관)



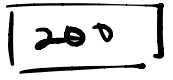
* 인스턴스 필드

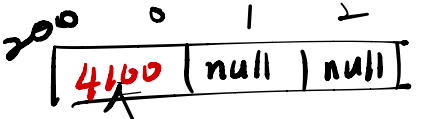
```
class User {
  String name;
  String email;
  String password;
  String tel;
}
```



* 레퍼런스 배열

```
User[] users = new User[3];
```

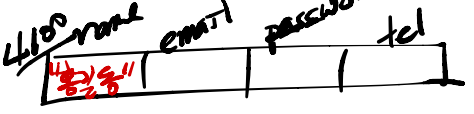




User 변수
레퍼런스 변수

ex) User u1;

```
users[0] = new User();
```



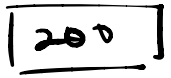

```
user[0].name = "홍길동";
```
4100

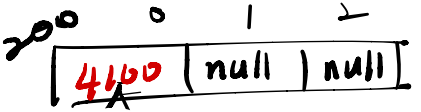
```
user[1].name = "임꺽정";
```
null

→ NullPointerException 에러 발생!

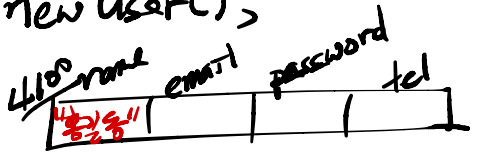
* 레퍼런스 배열

```
User[] users = new User[3];
```





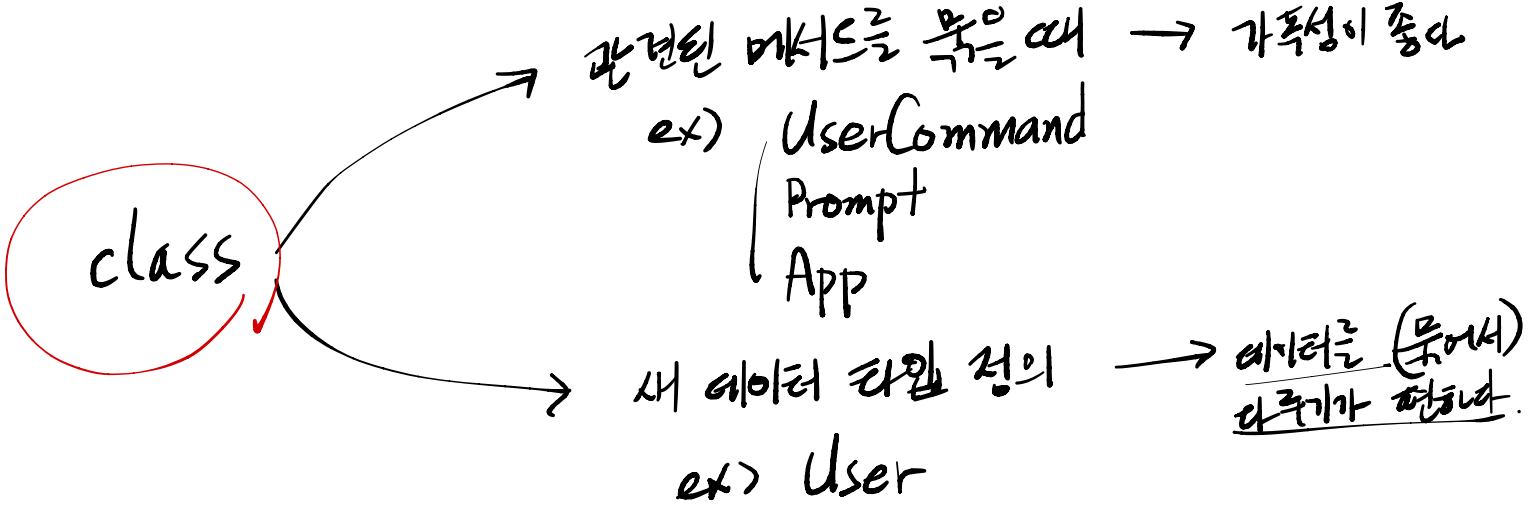

```
users[0] = new User();
```



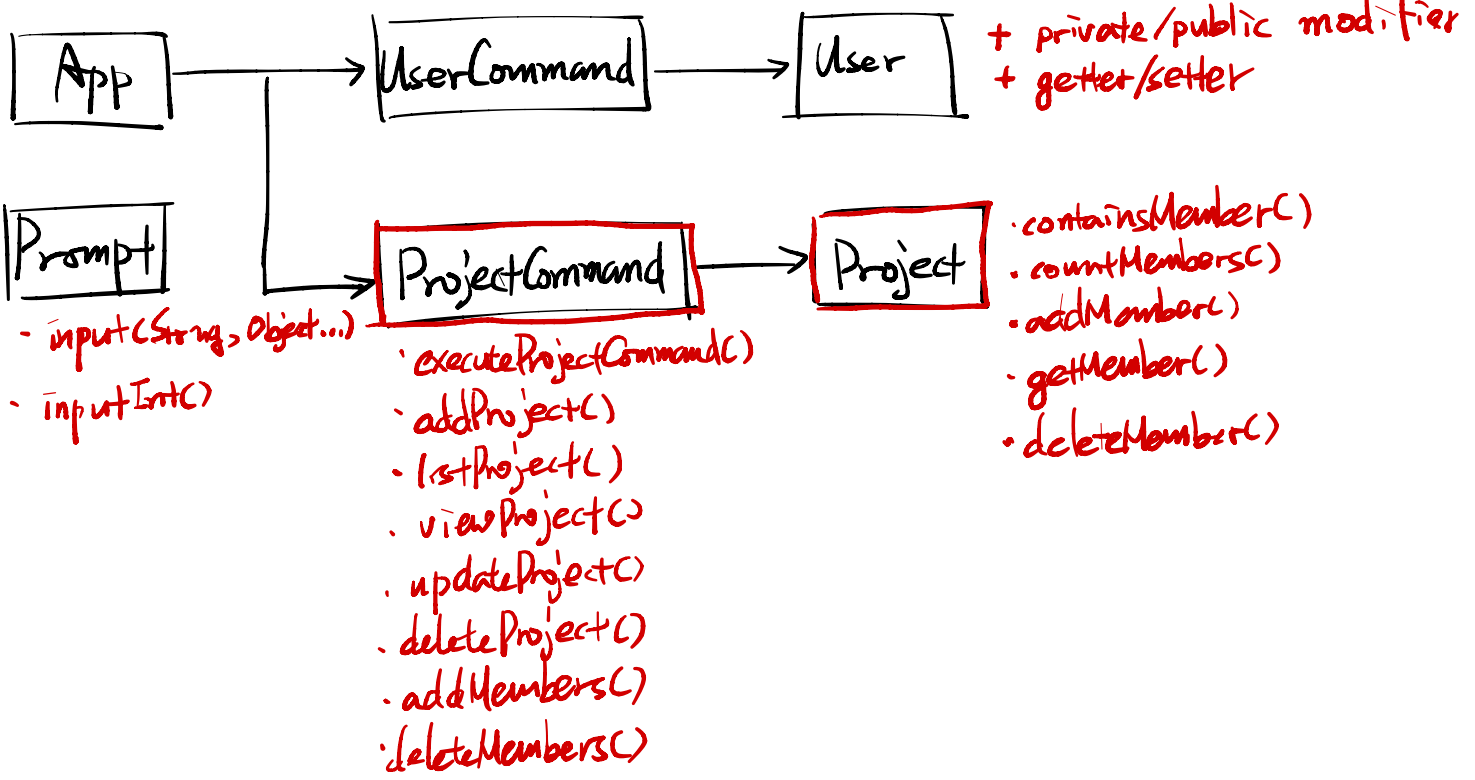

```
User user = users[0];
```
4400


```
users[0].name = "aaa";
user.name = "aaa";
```

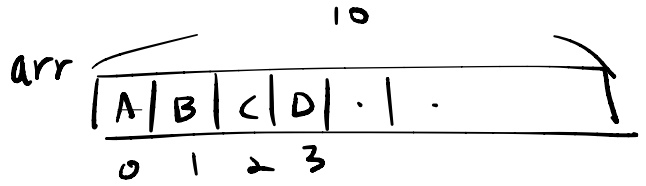
* 클래스 목적

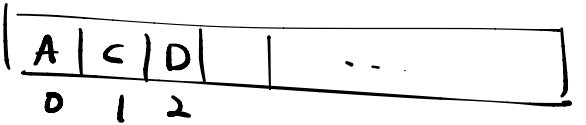

class
관련된 메서드를 묶을 때 → 가독성이 좋다
ex) UserCommand
Prompt
App
새 데이터 타입 정의 → 데이터를 (묶어서) 다루기가 편하다.
ex) User

10. CRUD 구현 - 프로젝트 CRUD



* 배열의 항목을 인덱스로 삭제할 때





B 랑 D를 삭제

```
for(int i = 0; i < arr.length; i++){
    if (arr[i] == 'B' || arr[i] == 'C'){
            내용까지 삭제 수행
    }
}
```

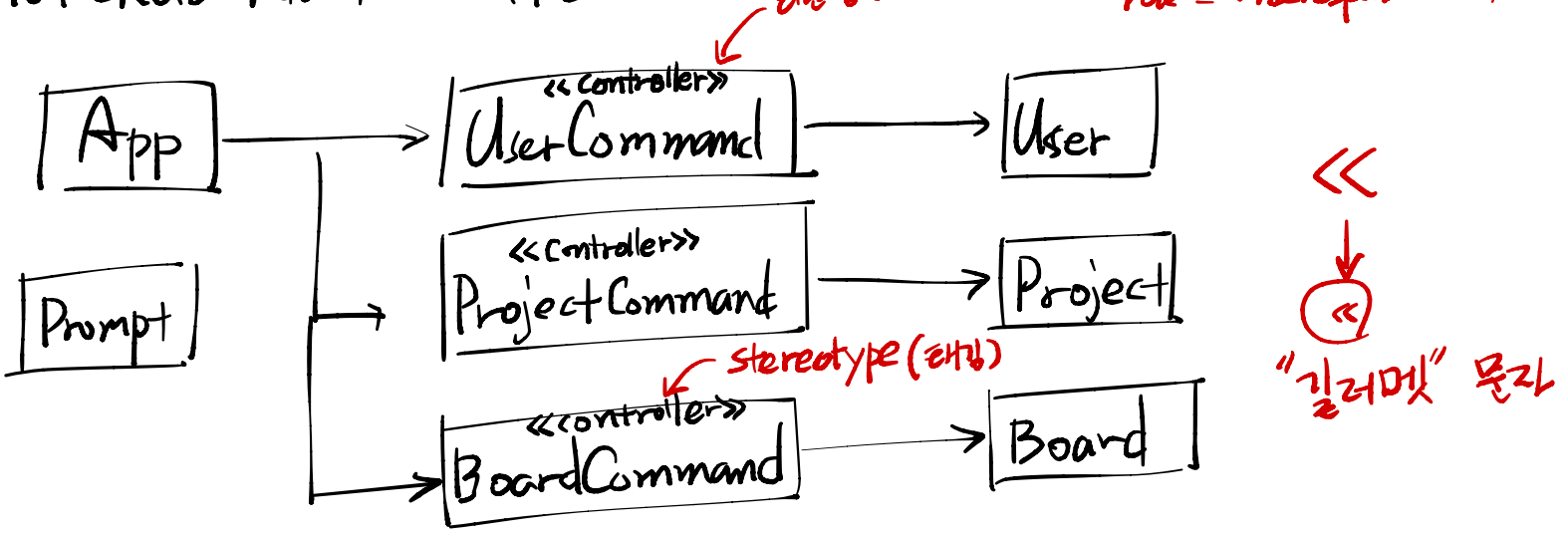
'C'가 앞으로 당겨지면서
검사 대상에서 제외되는 문제 발생

⇓

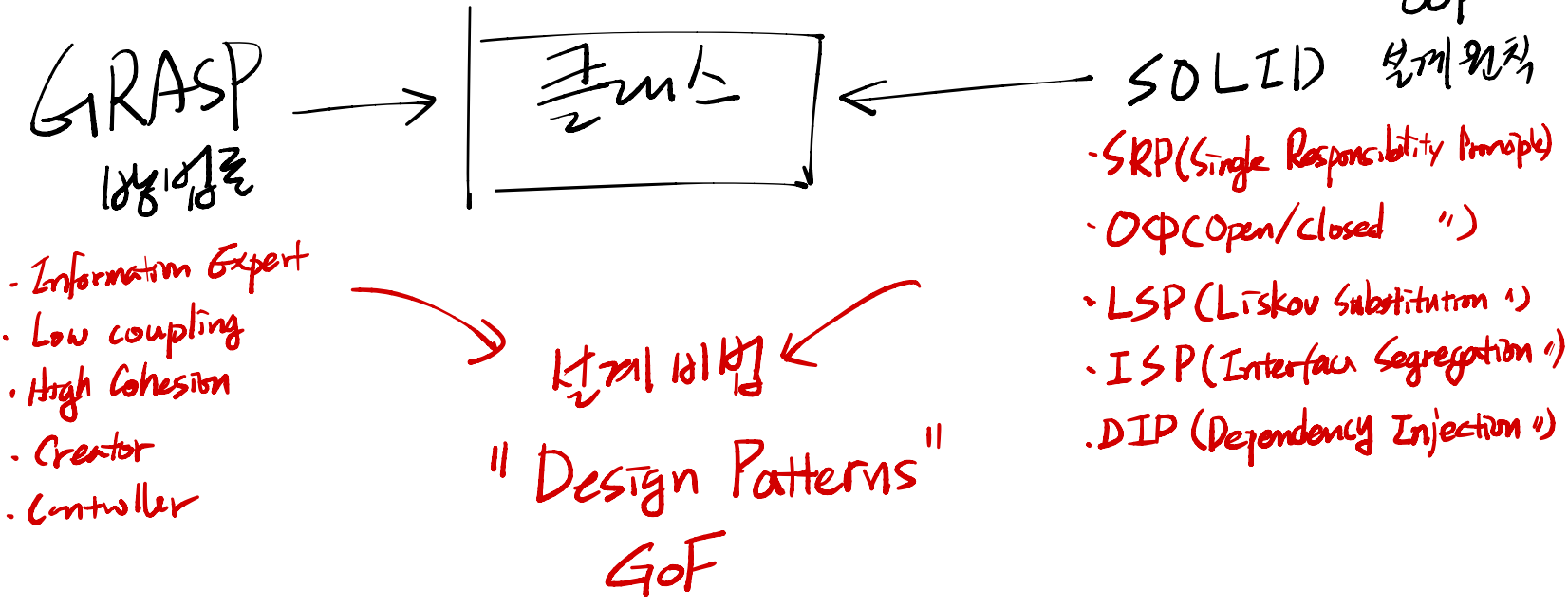
해결책은?

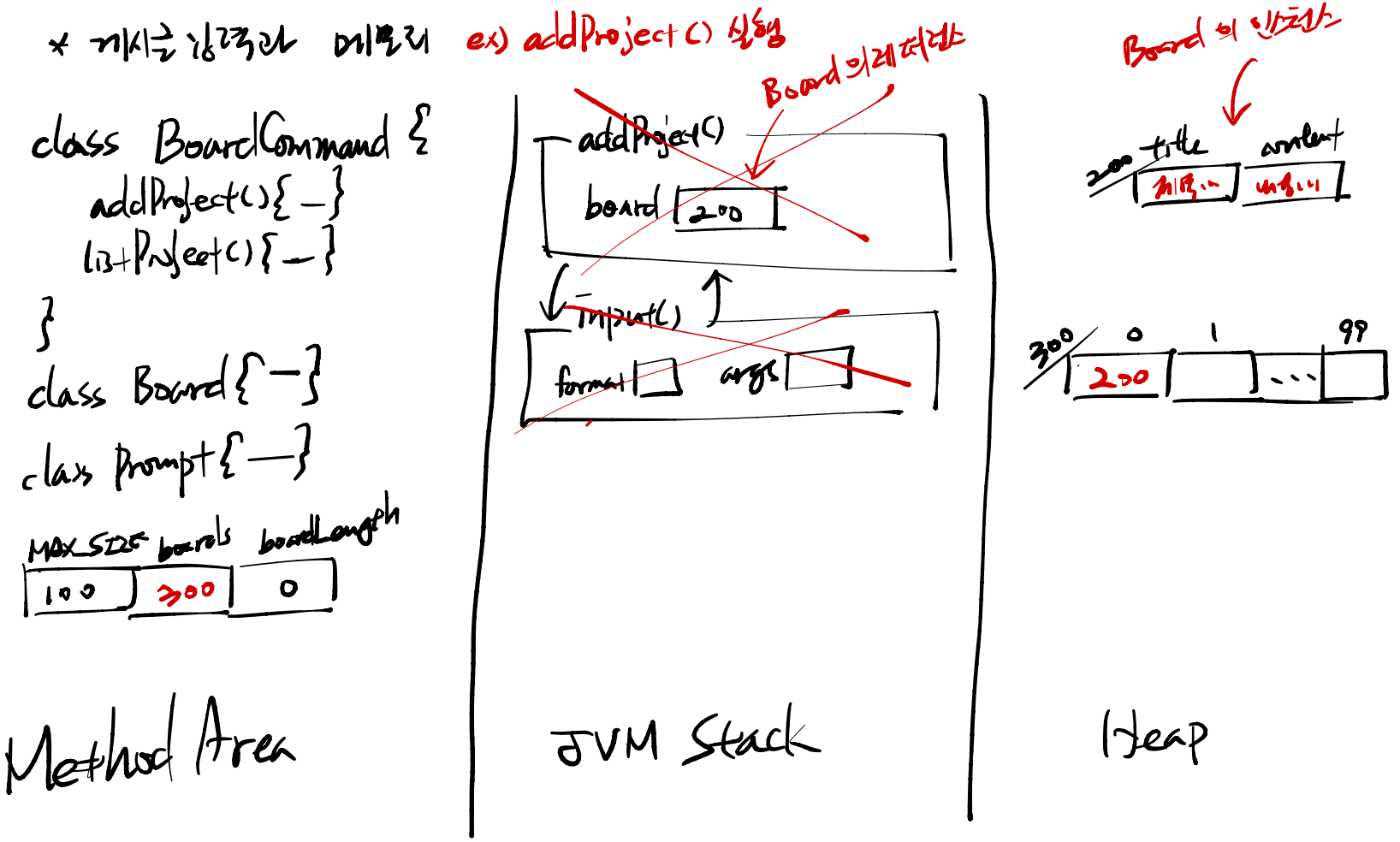
거꾸로 반복

10. CRUD 구현하기 – 게시판 CRUD



* 클래스 정의 : 방법론 → 클래스를 설계하는 개념들의 집합



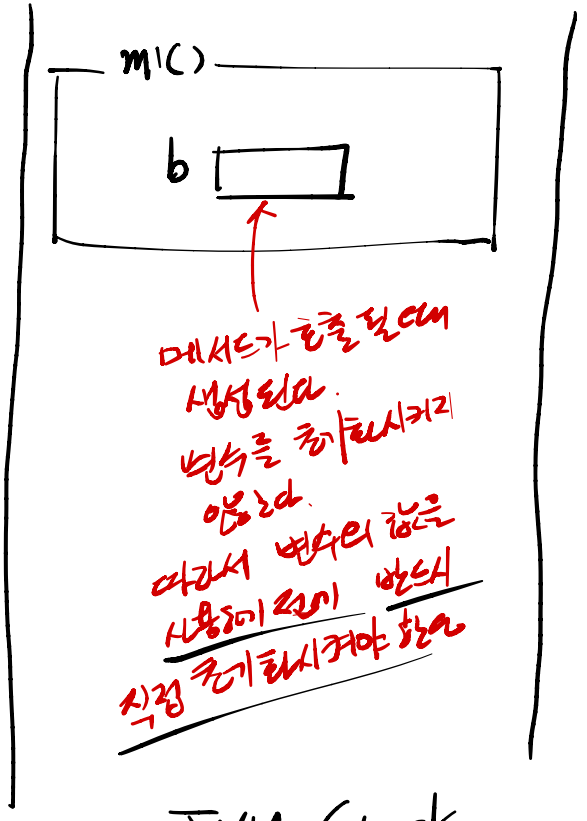

* 메서드 영역과 메모리
ex) addProject() 실행
Board 의 인스턴스
class BoardCommand {
addProject(){_}
listProject(){_}
}
class Board{—}
class Prompt{—}
MAX_SIZE
boards
boardLength
100
200
0
addProject()
Board 의 레퍼런스
board
200
input()
format
args
200
title
content
300
0
1
99
200
Method Area
JVM Stack
Heap

* 변수초기화

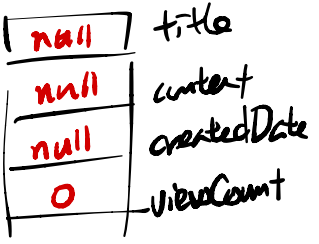
```
class A {
    static int v1;

    void m1() {
        Board b = new Board();
    }
}
```

A 클래스가 메모리에 로딩될때 자동생성 되어서 맞춰서 자동초기화



메서드가 호출될때 생성된다. 변수를 초기화시키기 위해서. 따라서 변수의 값은 사용되기 전에 반드시 초기화시켜야 한다.



new 명령 실행시 생성 되어서 맞춰서 자동초기화.

Method Area

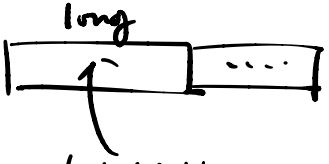
JVM Stack

Heap

* new Date()

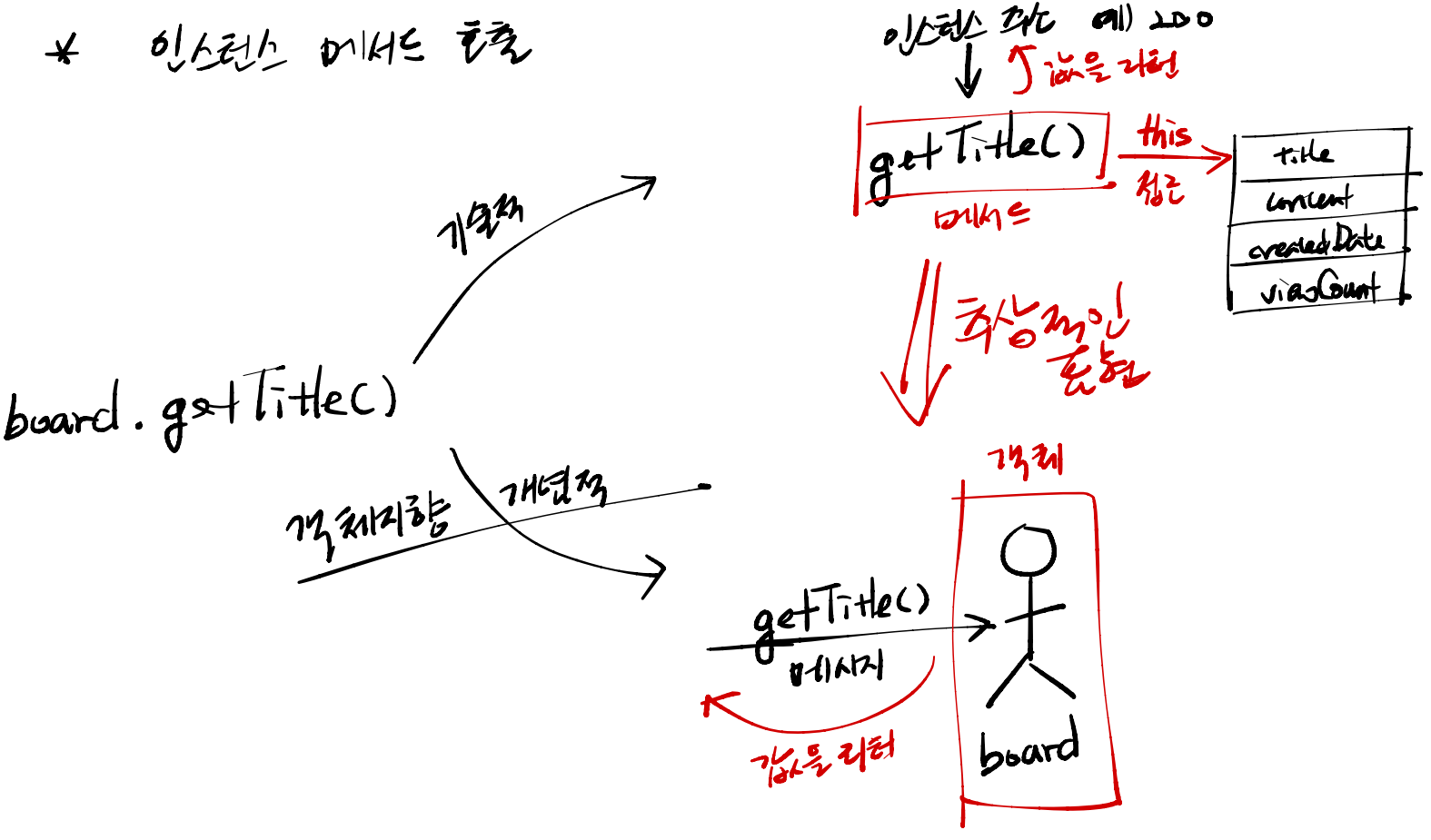
java.util

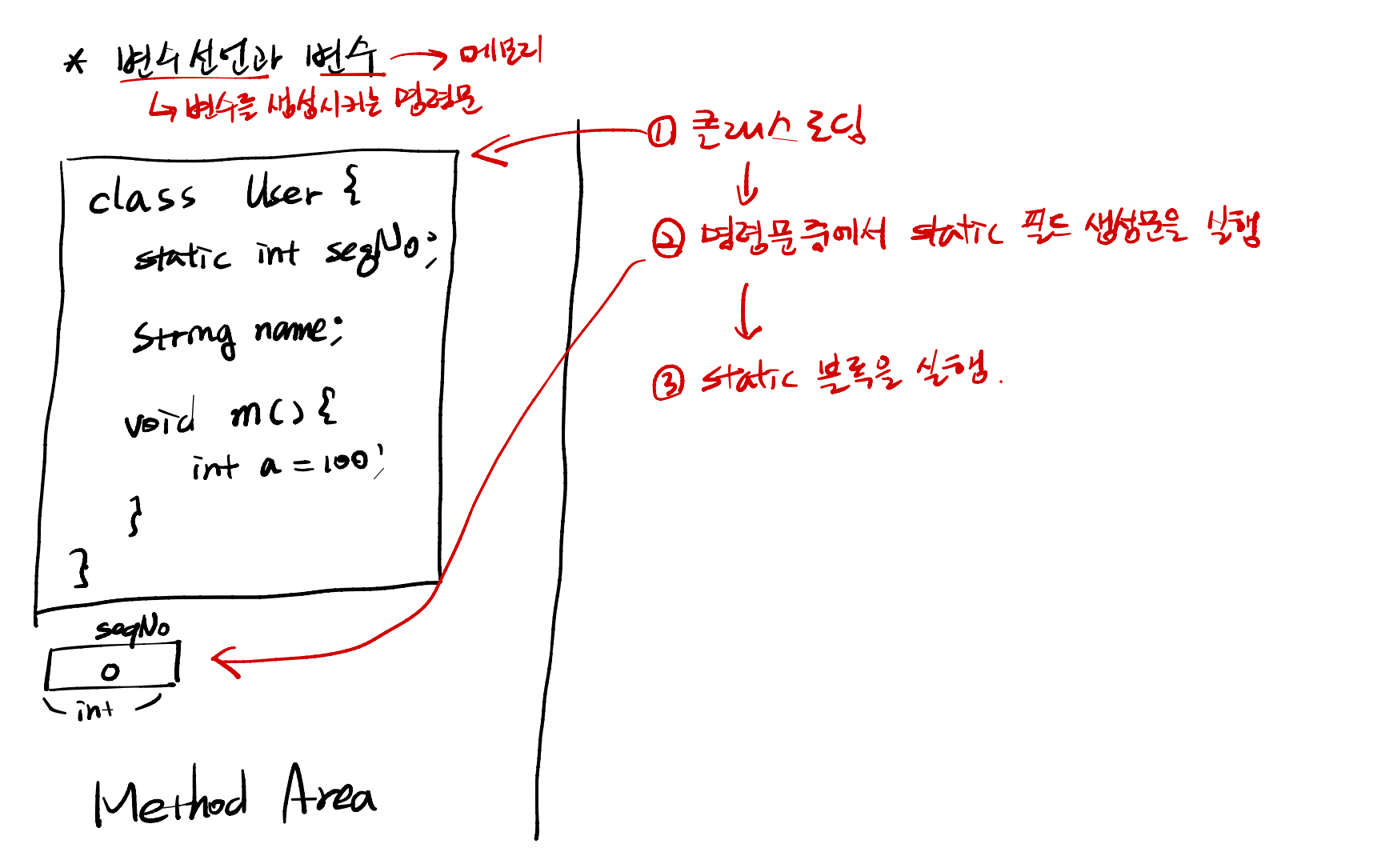
↓ 인스턴스



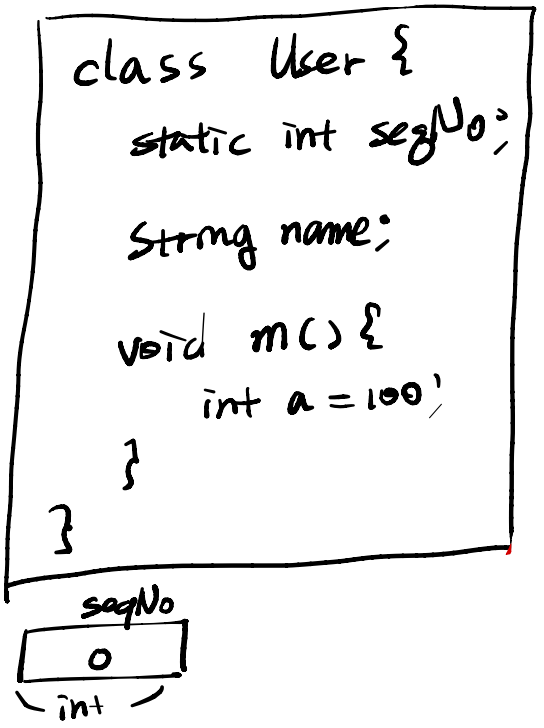
1970년 1월 1일
00:00:00 부터
현재까지 경과된
milisec 값을
반환.

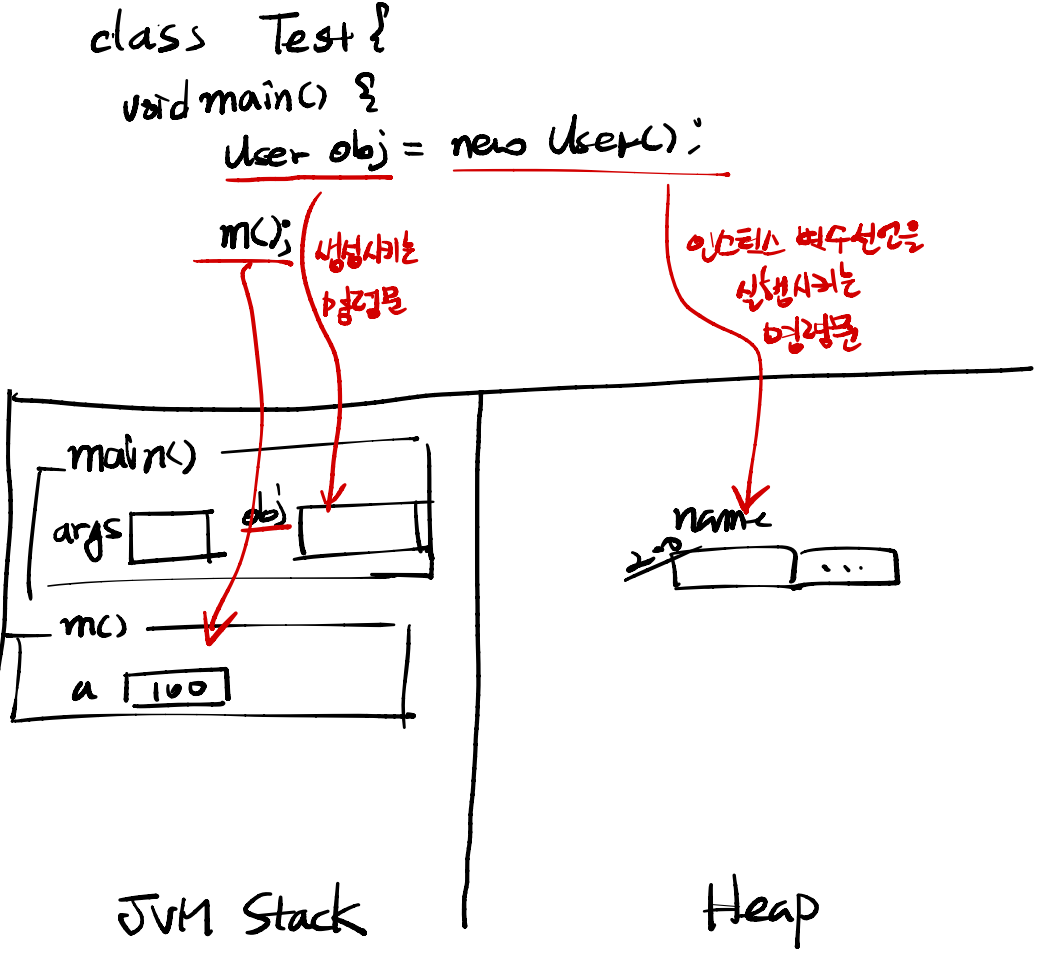
* 인스턴스 메서드 호출


인스턴스 관련 메서드
값을 리턴
getTitle()
this
참조
메서드
title
content
createdDate
viewCount
프로그램의 흐름
board.getTitle()
개발자
객체지향
개념적
객체
getTitle()
메시지
값을 리턴
board

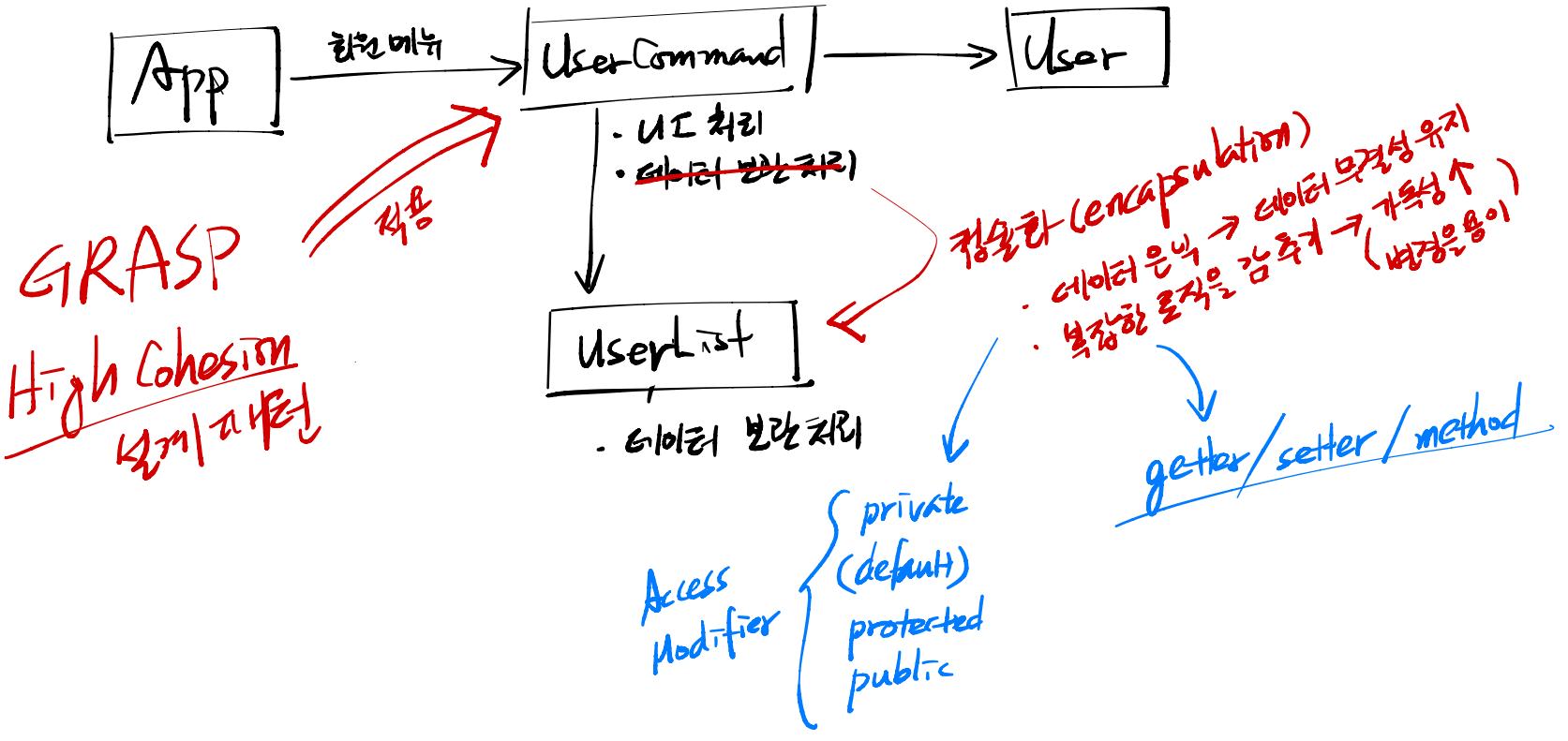

* 변수선언과 변수 → 메모리
↳ 변수를 생성시키는 명령문
class User {
static int seqNo;
String name;
void m() {
int a = 100;
}
}
① 클래스 로딩
② 명령문중에서 static 필드 생성문을 실행
③ static 블록을 실행.
seqNo
0
int
Method Area

* 변수선언과 변수


class User {
static int seqNo;
String name;
void m() {
int a = 100;
}
}
seqNo
0
int
Method Area


class Test {
void main() {
User obj = new User();
m();
생성시키는 명령문
인스턴스 변수선언을 실행시키는 명령문
main()
args
obj
m()
a
100
name
...
JVM Stack
Heap

12. 인스턴스 목록을 다루는 코드를 분리하기



13. 스태틱 필드의 한계를 극복하기

1. 회원
2. 프로젝트
3. 게시판
4. 공지사항 ← 기능 추가 ⇒ 기존의 (BoardCommand, BoardList, Board) 클래스를 사용해서 만들수 있었다!
5. 로그인
6. 홈

인스턴스 필드로 변경!

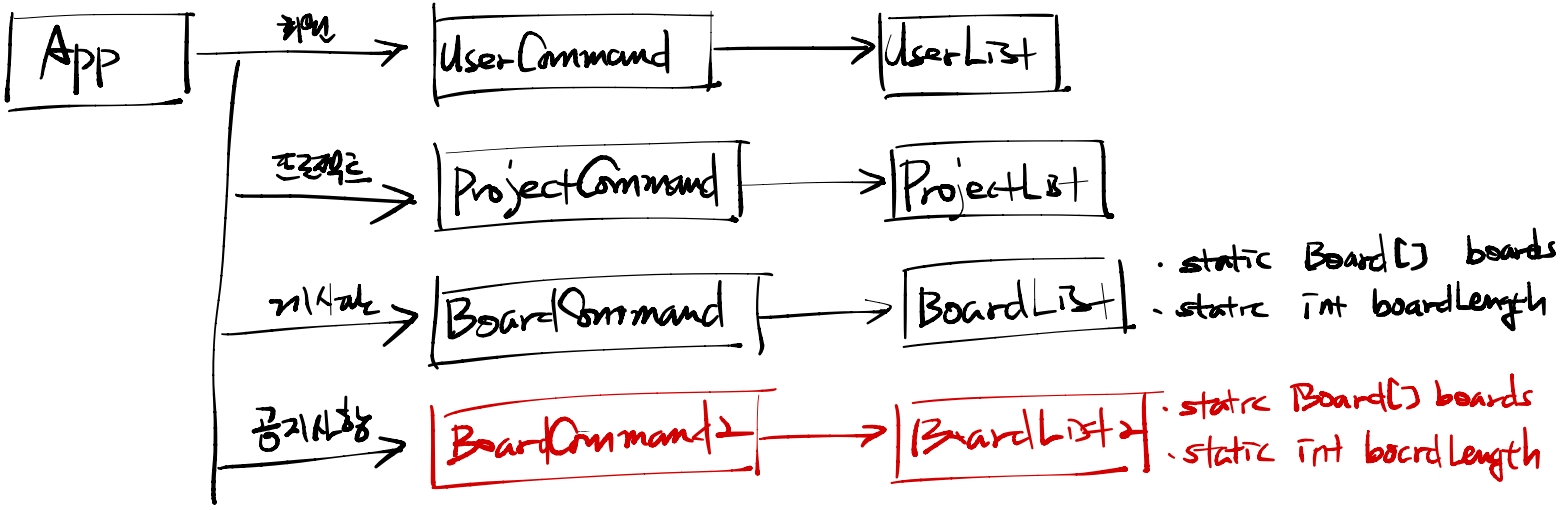
있다!

이유

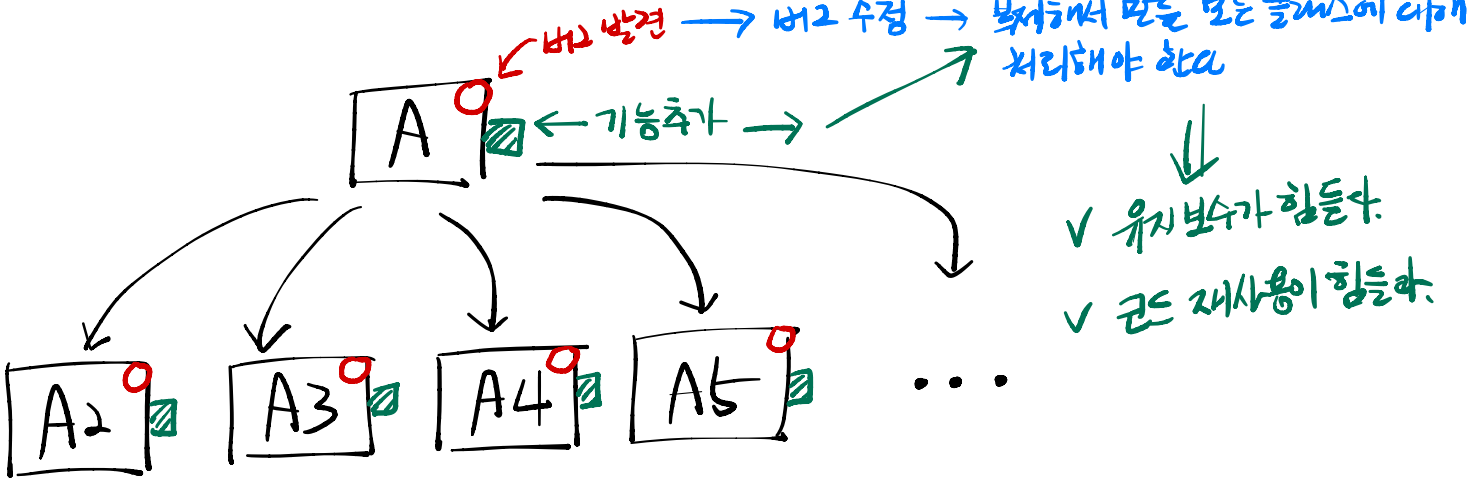
데이터를 저장하는 필드가 스태틱이다!

하면된다?

* 메뉴처리


App
회원
UserCommand
UserList
프로젝트
ProjectCommand
ProjectList
게시판
BoardCommand
BoardList
· static Board[] boards
· static int boardLength
공지사항
BoardCommand2
BoardList2
· static Board[] boards
· static int boardLength

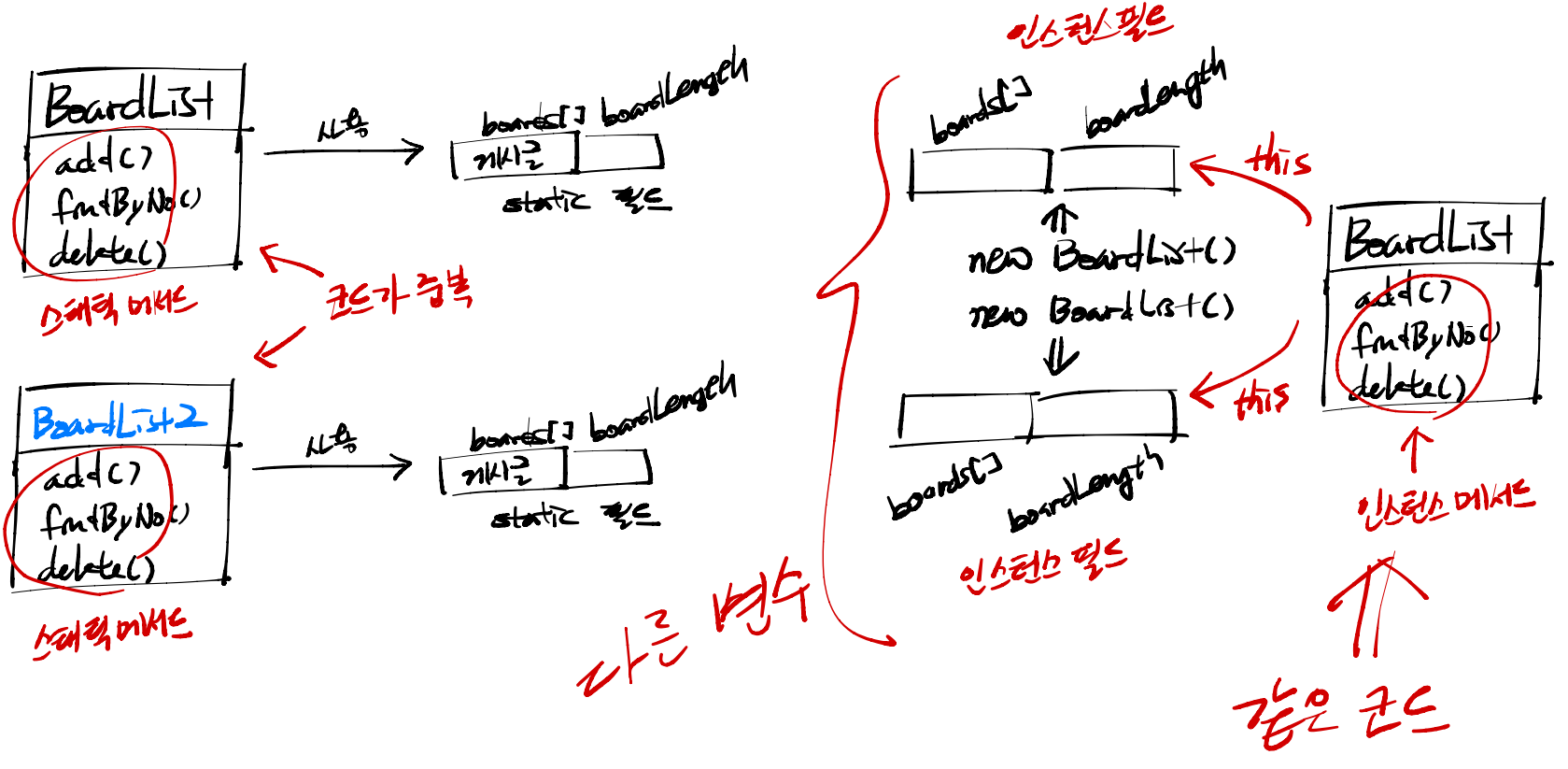
* 클래스를 복제하여 사용할 때 문제점

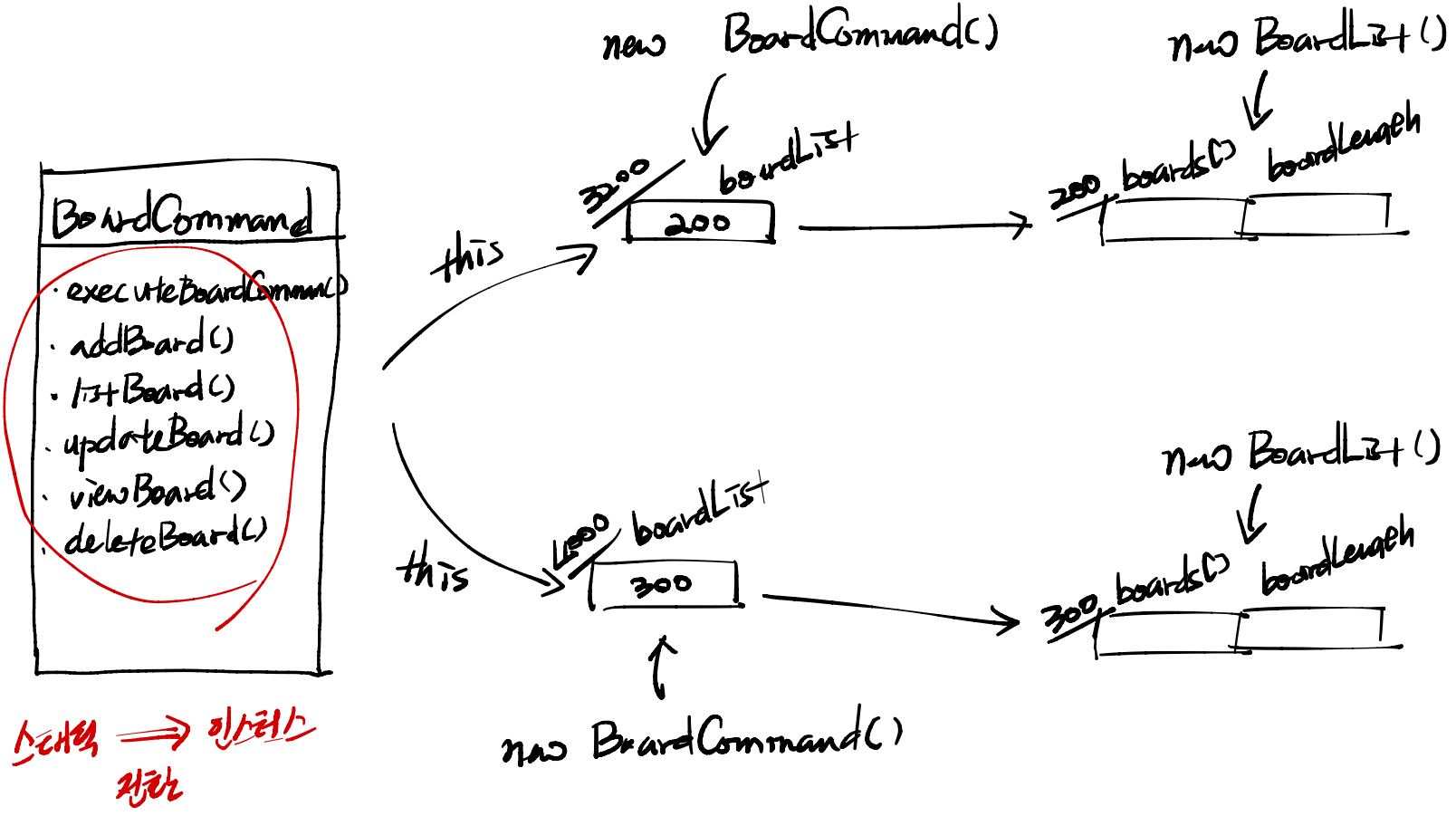


* 문제점

- 코드 중복 → 변경이 어렵다.

* 해결책? 같은 코드를 사용, 그러나 데이터는 분리




new BoardCommand()
new BoardList()
BoardCommand
executeBoardCommand()
addBoard()
listBoard()
updateBoard()
viewBoard()
deleteBoard()
this
3200
boardList
200
200
boards[]
boardLength
this
1600
boardList
300
300
boards[]
boardLength
new BoardCommand()
스태틱 ⟹ 인스턴스
전환

* 인스턴스 와 메서드

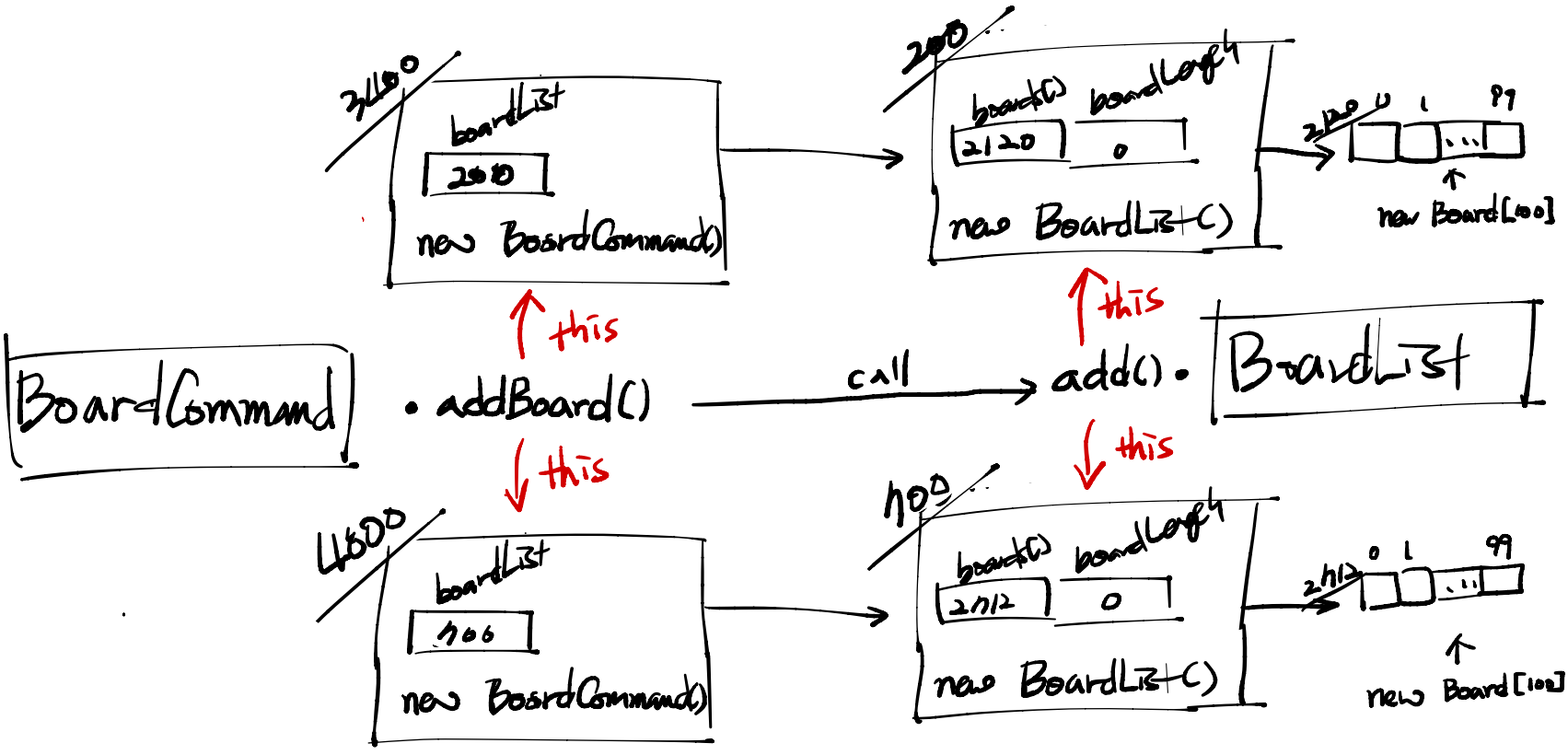
A 클래스 A 클래스

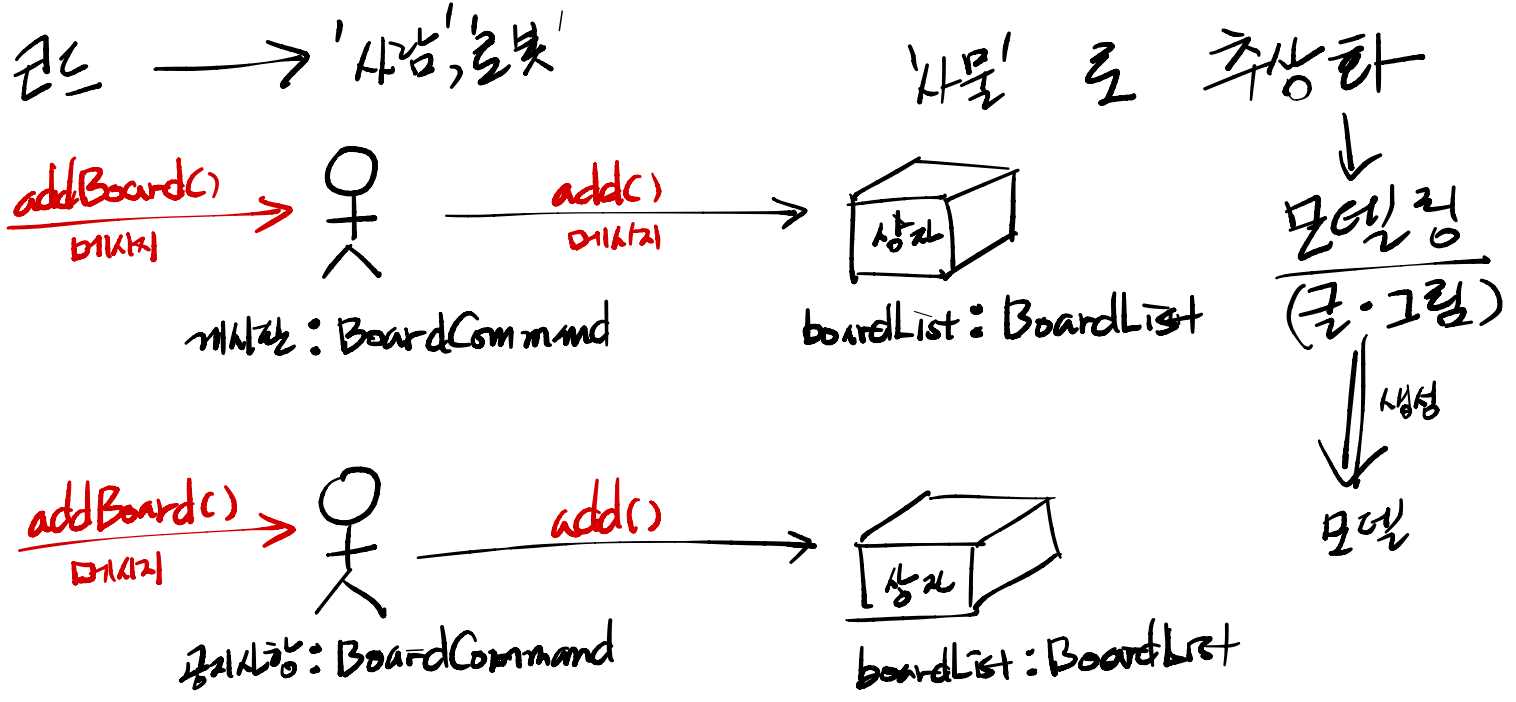
레퍼런스 . 메서드();

this 내장 변수는 메서드가 정의된 클래스의 인스턴스 주소를 받기때문

String s = "hello"; s++; ← String 피연산자를 처리하는 ++ 연산자가 정의되어 있지 않다.

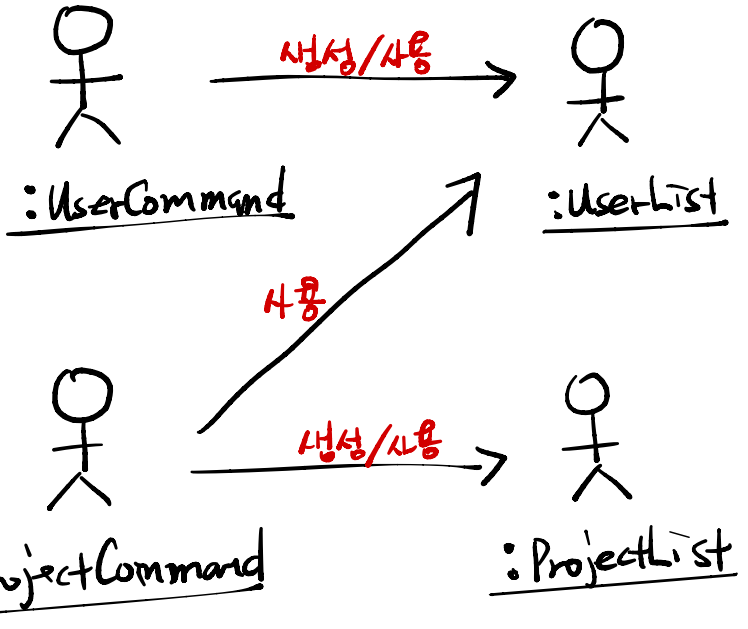
int i = 100; i++;


BoardCommand
• addBoard()
this
this
call
add() •
BoardList
this
this
boardList
2000
new BoardCommand()
boardList
1600
new BoardCommand()
board[]
boardLength
2120
0
new BoardList()
board[]
boardLength
2112
0
new BoardList()
new Board[100]
new Board[100]


관점 → '사람' 로봇
'사물' 로 추상화
addBoard()
메시지
add()
메시지
게시판 : BoardCommand
boardList : BoardList
상자
모델링
(클래스)
생성
모델
addBoard()
메시지
add()
공지사항 : BoardCommand
boardList : BoardList
상자

* 인스턴스 공유

의존객체 (dependency)



★ UML 표기법

↓

인스턴스 표기

↓

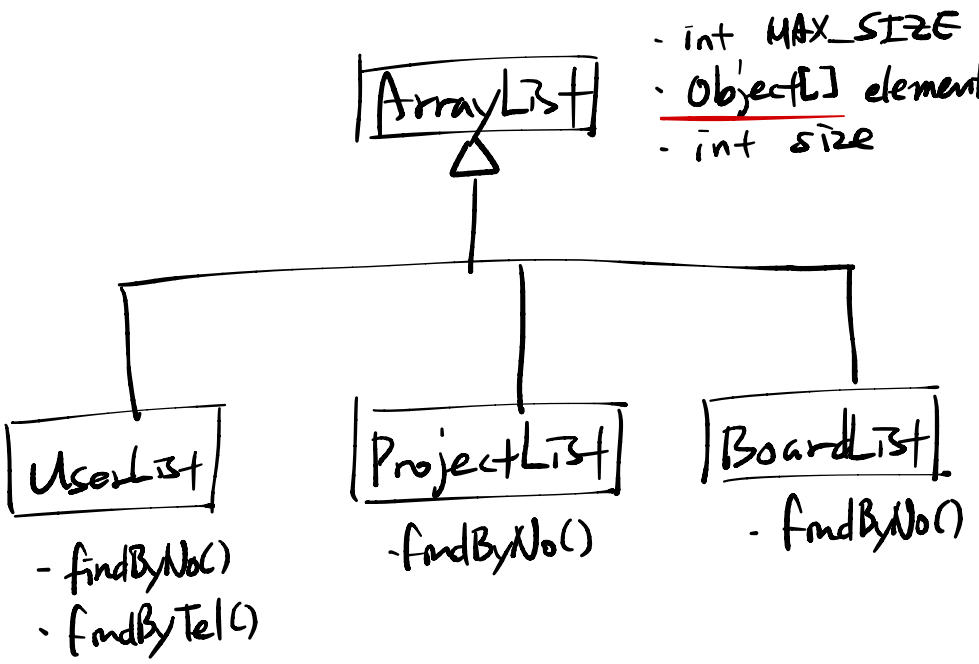
인스턴스명 : 클래스명

↓ 예)

list : UserList

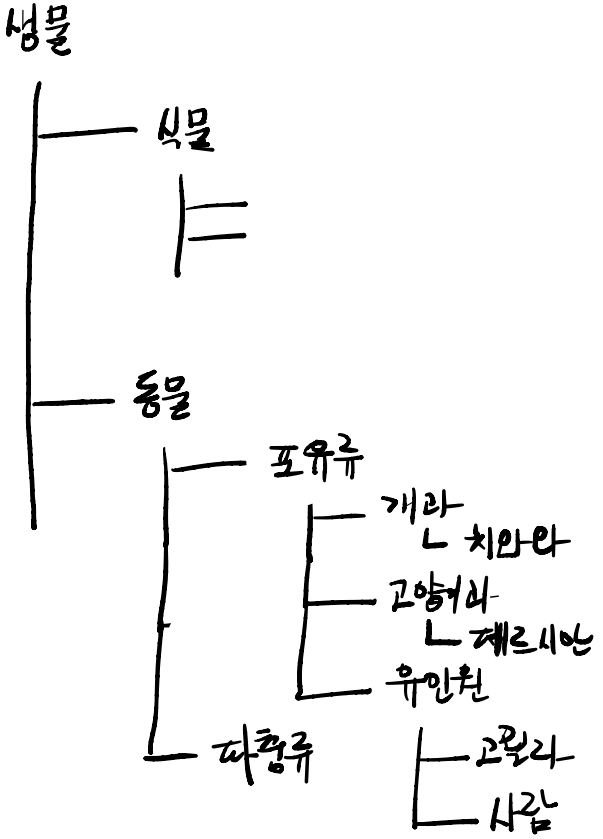
인스턴스 이름을 표기하지 않아도 모델을 이해하는데 문제가 없다면 생략가능

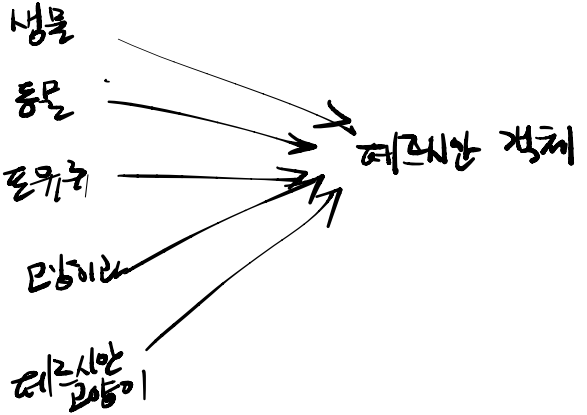
14. 공통코드 분리 및 사용하기 : 상속의 일반화(generalization) 개념

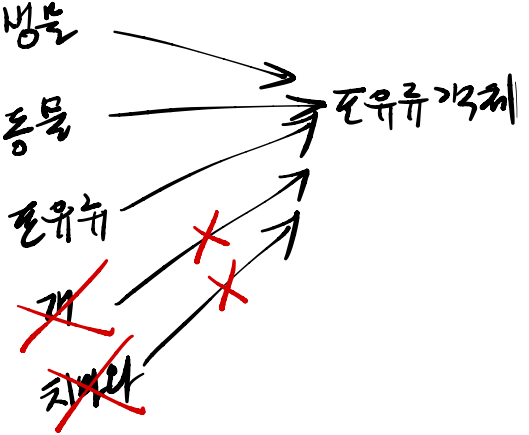


일반적으로 필요한 메서드가 아니라 데이터 식별할 때 한번은 사용하는 경우에만 필요한 메서드. 그래서 공통 코드로 두는 ArrayList에 넣지 않았다.

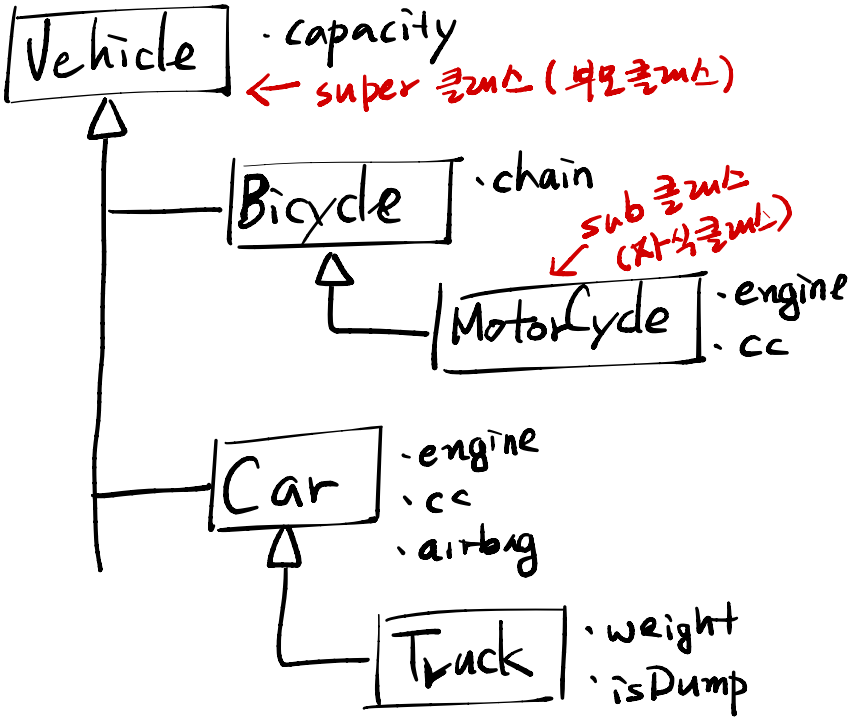
* 분류 계층도 표현


생물
식물
동물
포유류
개과
치와와
고양이과
래브라도
유인원
파충류
고릴라
사람


생물
동물
포유류
고양이
래브라도 고양이
래브라도 객체


생물
동물
포유류
개
치와와
포유류 객체

* 자바의 클래스 계층도와 레퍼런스



MotorCycle m = new MotorCycle();

Bicycle bi = new MotorCycle();

Vehicle ve = new MotorCycle();

⇓

상위 클래스의 레퍼런스로
하위 클래스의 인스턴스를 참조할 수 있다

"상위분류명으로 하위분류객체를
가리킬 수 있다"

java.lang

* Object 클래스

```
class A {
    ≡
}
```

컴파일 →

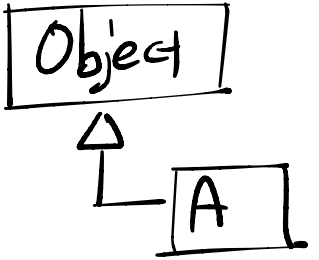
```
class A extends Object {
    ≡
}
```

super class

extends Object : 추가

* 결론

✓ 자바의 모든 클래스는 Object의 자손이다.



⇓

```
Object obj = all 인스턴스;
```

obj ↖ 레퍼런스