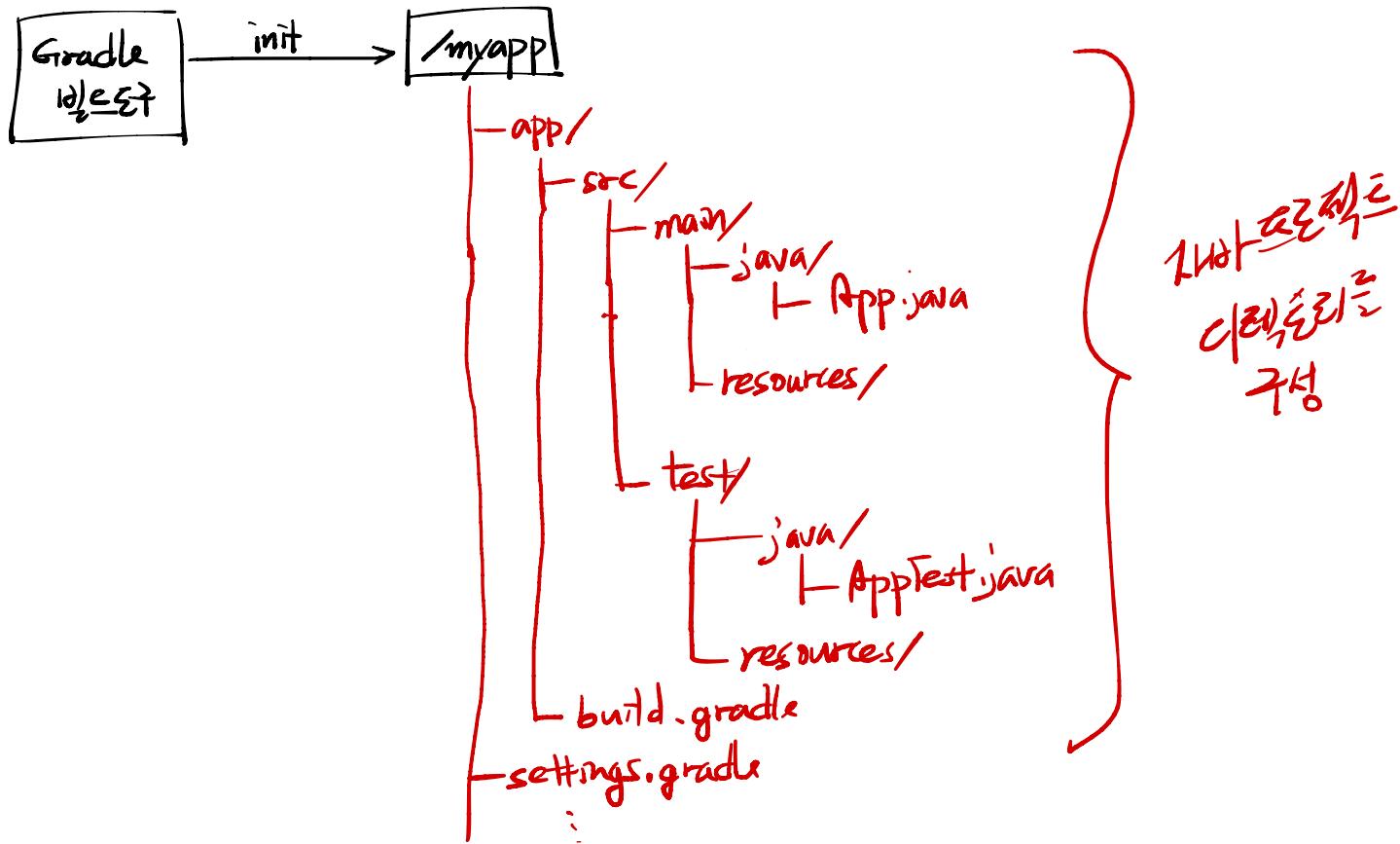
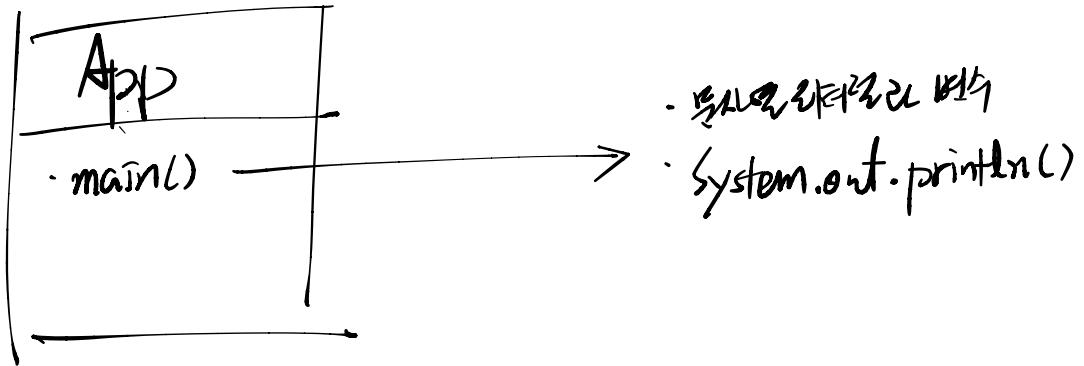


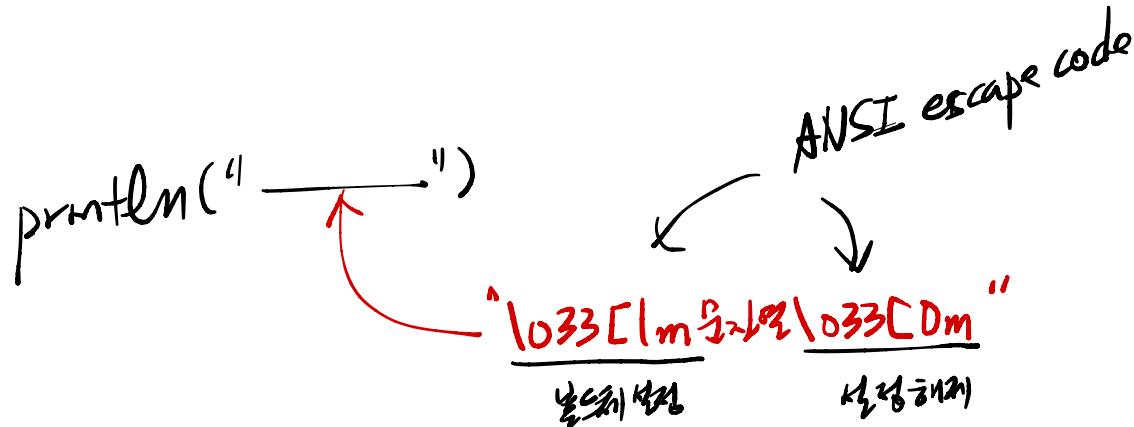
## 01. 자바 프로젝트 준비



02. 자바를 사용하는 방법으로 출력문을 출력하기



03. ANSI escape 코드를 사용하여 콘솔 출력을 조작하기



\* gradle run --quiet

↳ gradle 실행 과정을 제거  
gradle 실행 과정을 설명하는 문구

## 04. 키보드 입력 다루기

main() →

- Scanner keyboard = new Scanner();
- keyboard.nextInt()  
↳ next(), nextLine(), ...
- keyboard.close();

System.io

## 05. 배열을 활용하여 메뉴 출력하기

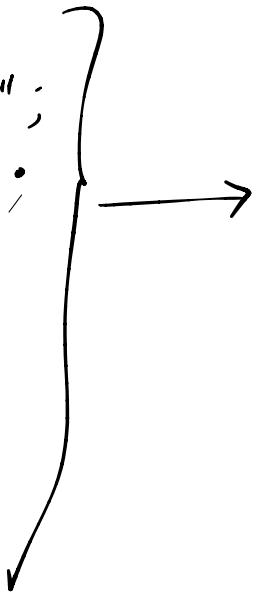
```
String menu1 = "1. 퇴원";
```

```
String menu2 = "2. 청진";
```

```
:
```

```
:
```

```
:
```



```
String[] menus = new String[] {
```

"퇴원", "청진", "재진료", ...

```
}
```

String  
퇴원 선택됨

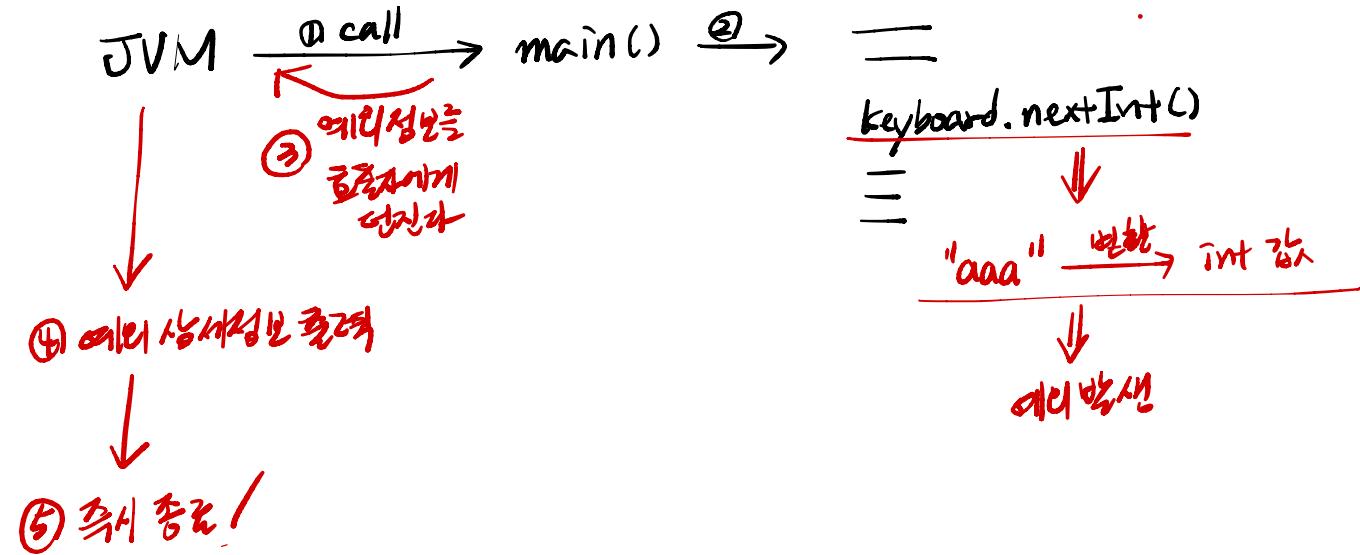
String  
재진료 선택됨

for 선택된 메뉴 출력

```
for ( ; ; ) {  
    ==  
}
```

## 06. 예외 처리

### 예외 처리 전



## 06. 예외처리

예외처리 후

JVM  $\xrightarrow{\text{① call}}$  main()  $\xrightarrow{\text{②}}$

try {

==

keyboard.nextInt()

==



"aaa"  $\xrightarrow{\text{변환}}$  int 값

↓

예외발생

→ 전달

} catch (InputMismatchException ex) {

자세한 조치 취함

}

예외처리 방법을 통해  
예외 상황을 통제하여  
JVM에게 알리지 않도록 하여  
실행을 계속 유지.

계속 실행

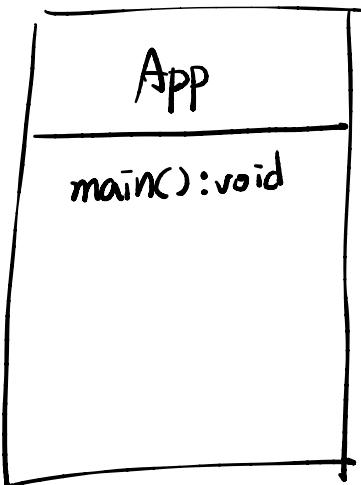
07. 문자열 바꾸기 쓰기위해 어떤 걸까요?

int menuNo = keyboard.nextInt() }  
} ⇒

String command =  
    keyboard.nextLine();  
  
if (command.equals("menu")) {  
    <sup>문자열 비교</sup>      ↑ 문자열 비교  
}  
  
int menuNo =  
    Integer.parseInt(command);  
  
String "2" → 2  
                ↓              ↑  
                INT 문자열

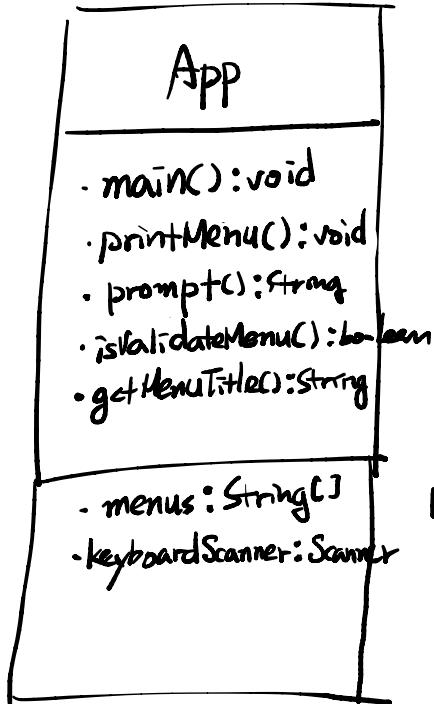
## 08. 1개의 메소드를 대량으로 쓰기 : 클래스 추상화

07.



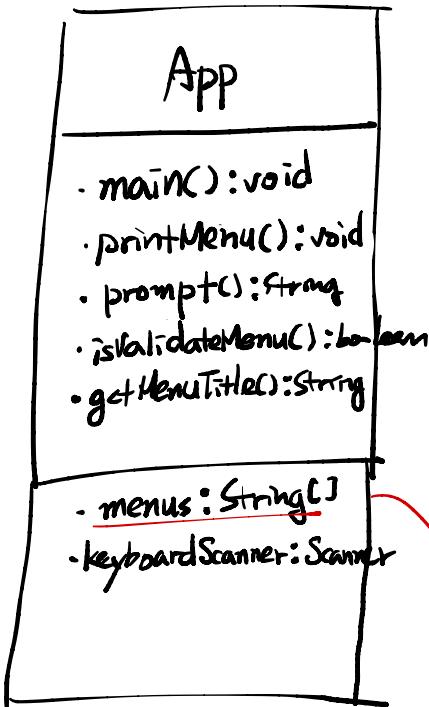
*refactoring*  
↓  
"extract method"

08.



↑  
기능  
↓  
O/FN  
↓  
기능 시각화  
↓  
기능 추상화  
↓  
기능 선택

## 09. 자바 기본문법 활용 연습



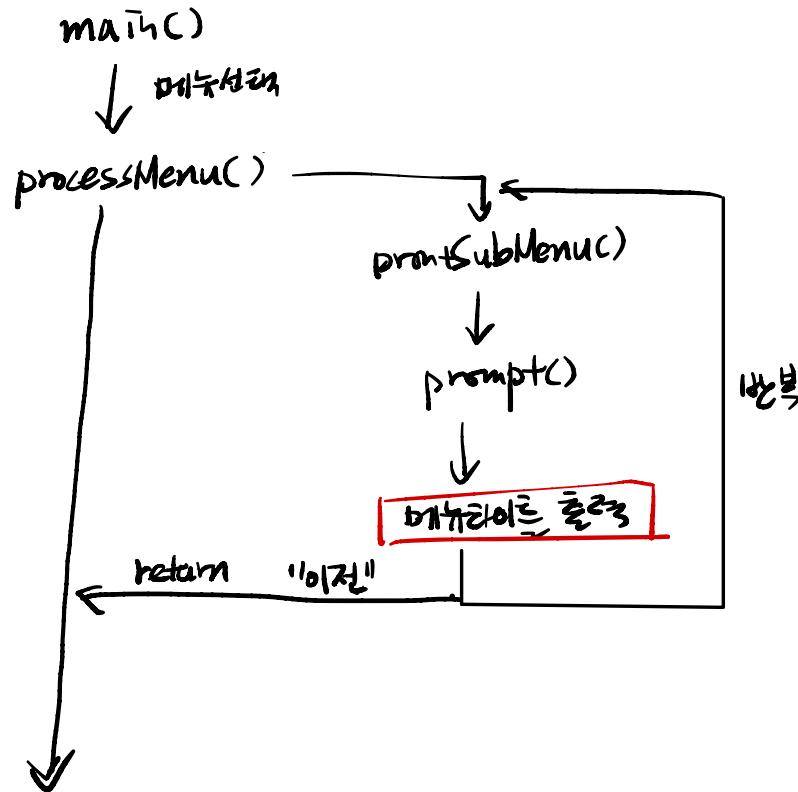
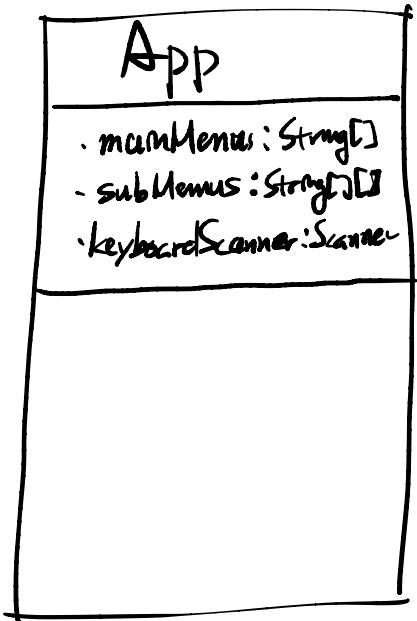
→ Handwritten code generated from the App class:

```
+ printSubMenu(){} -}
+ prompt(String title){} -}
+ validateMenu(int menuNo, String[] menus){} -}
+ getMenuTitle(int menuNo, String[] menus){} -}
+ processMenu(String menuTitle, String[] menus){} -}

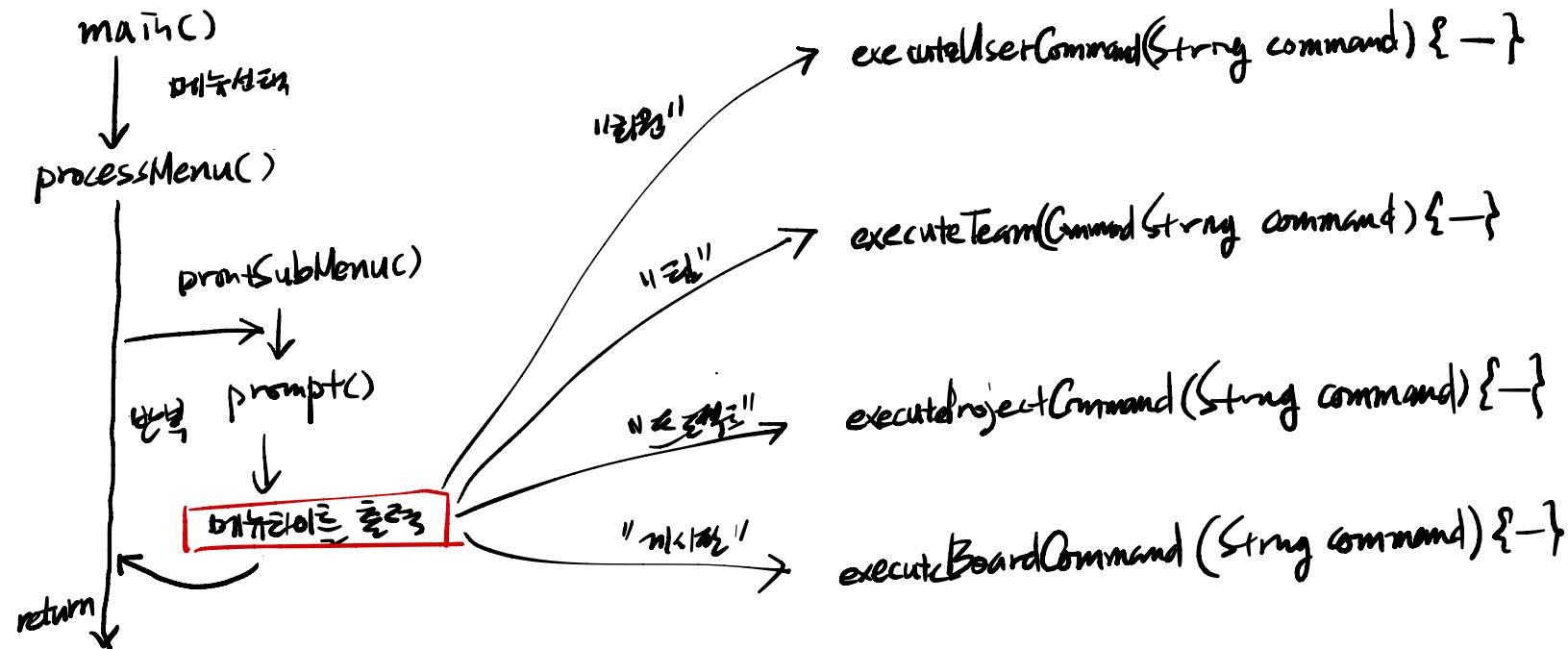
+ mainMenus: String[]
+ subMenus: String[][]
```

## 10. CRUD 퀴즈하기

설명문

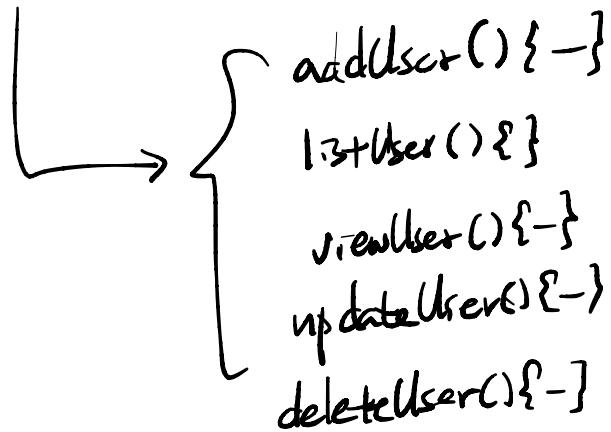


## 10. CRUD 툴 만들기 (2/3)

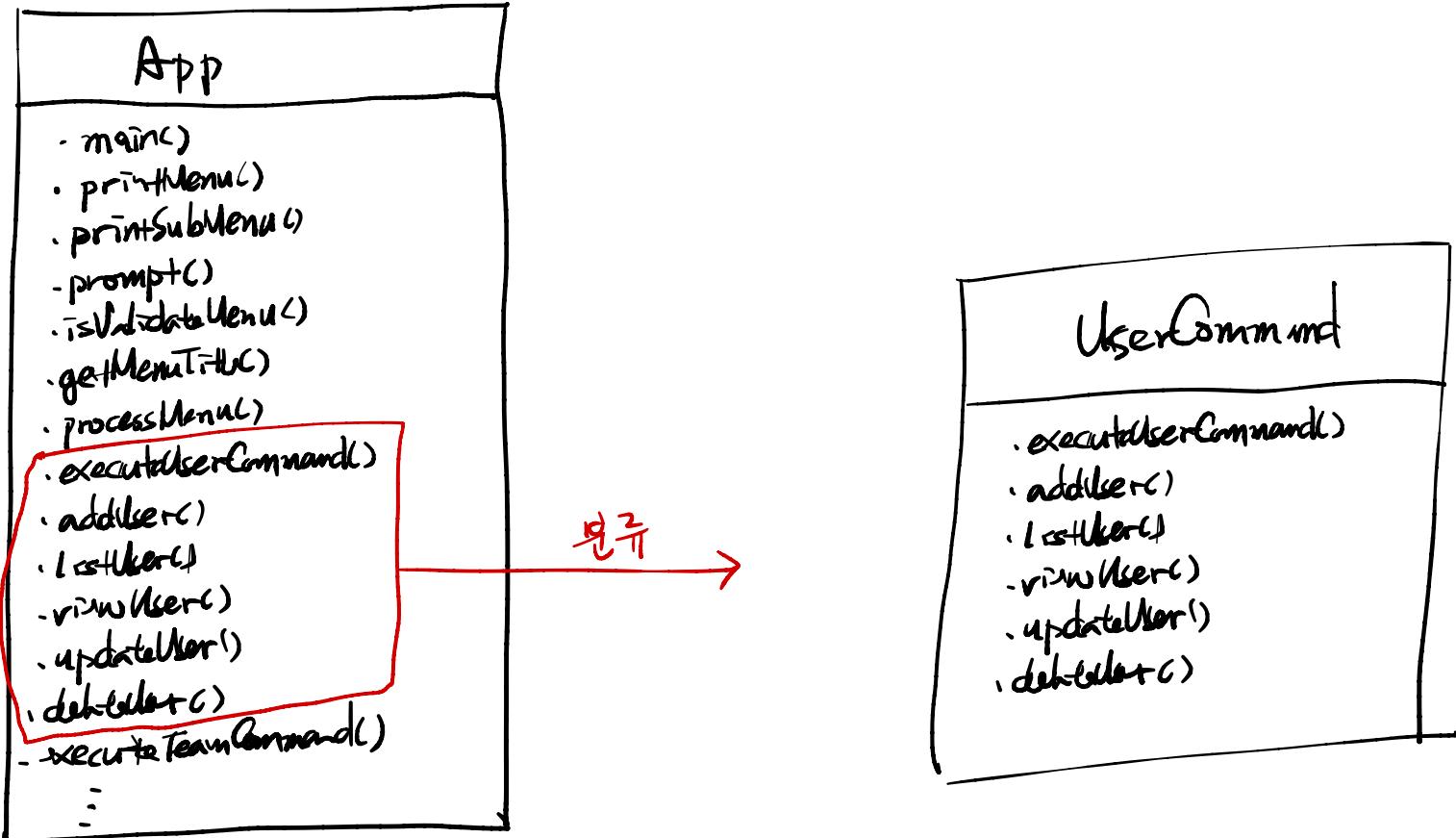


## 10. CRUD 주제(1) (2/3)

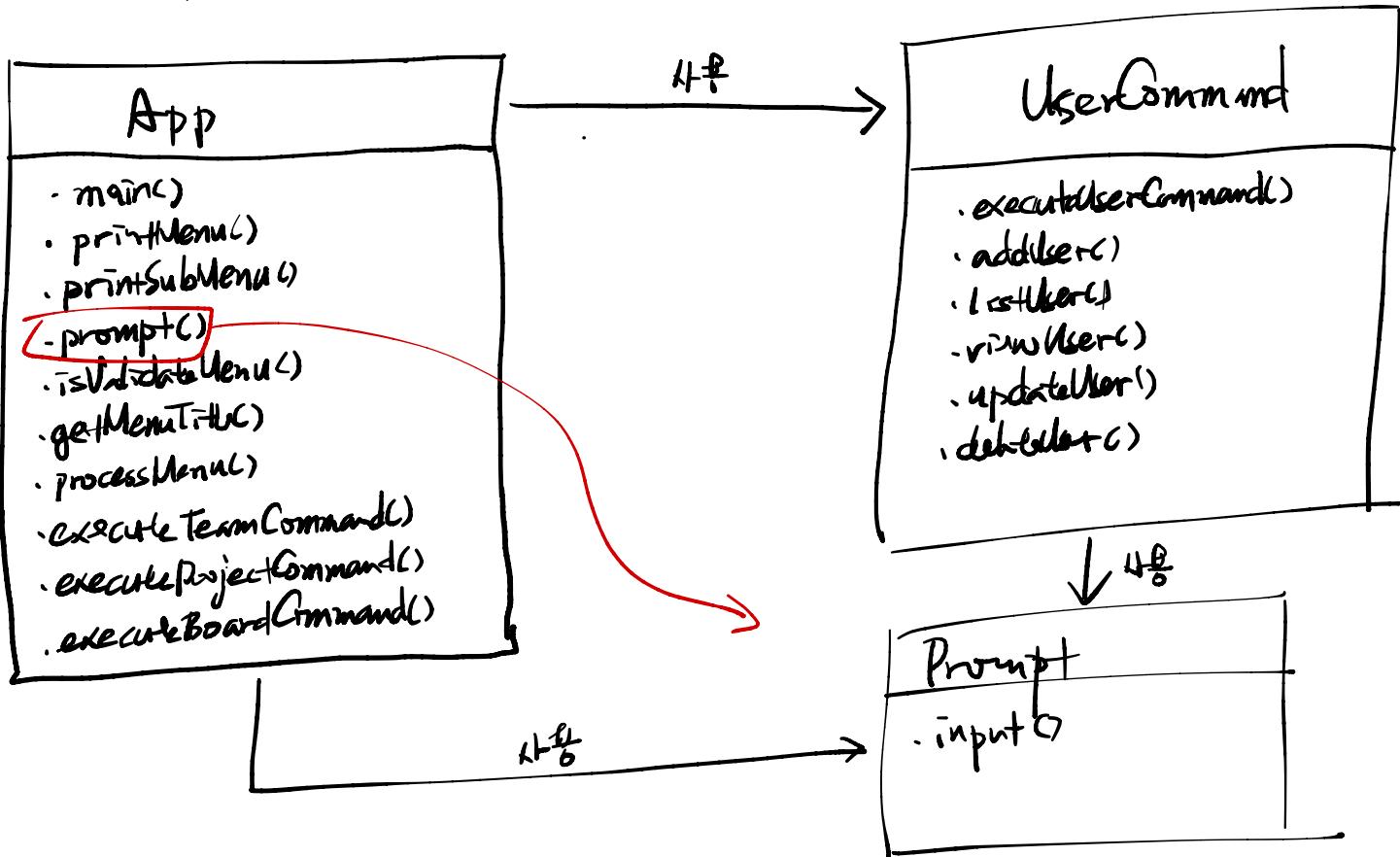
executeUserCommand(String command) { - }



## 10. CRUD 구현하기 (2/3)

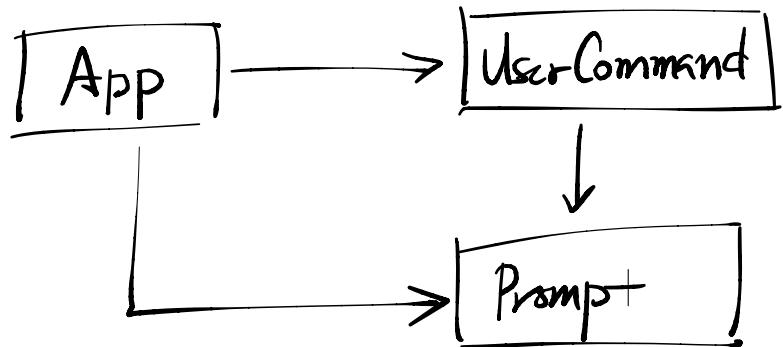


## 10. CRUD 구현하기 (2/3)



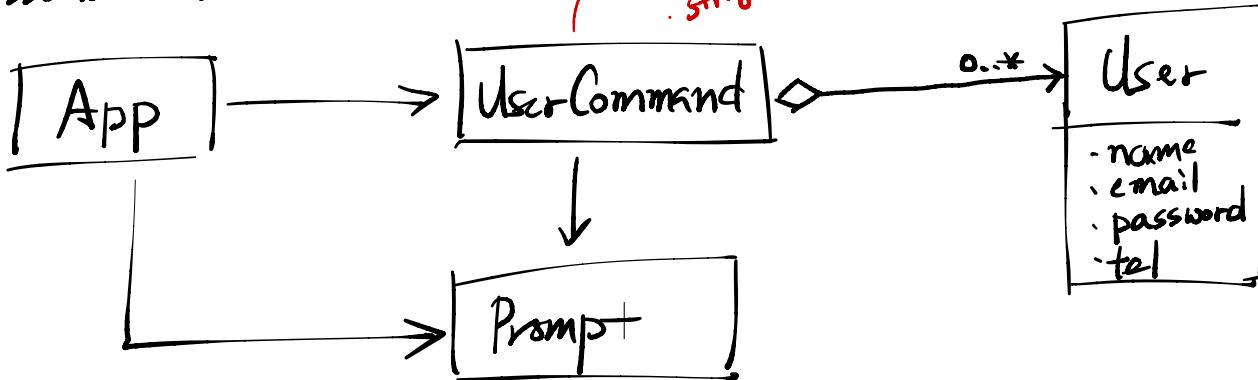
## 10. CRUD 구현하기 (2/3)

\* Association (연관)



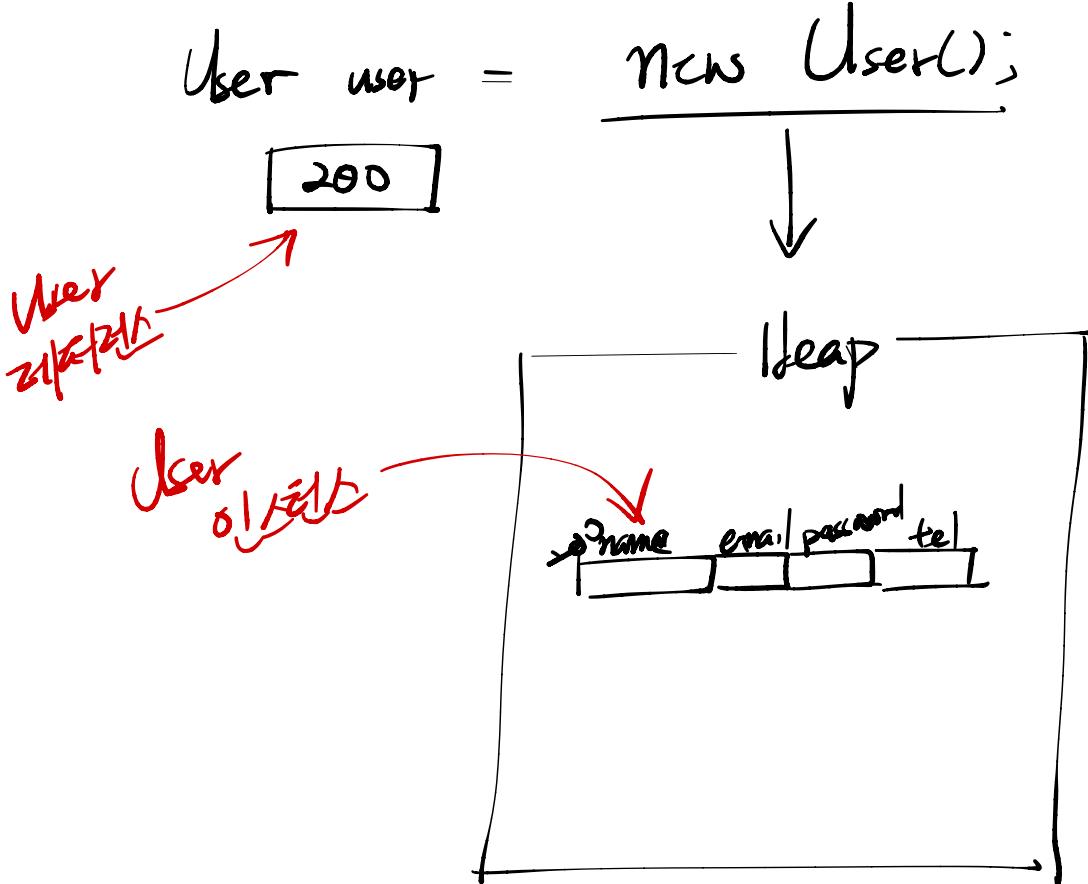
## 10. CRUD 구현하기 (2/3)

### \* Association (연관)



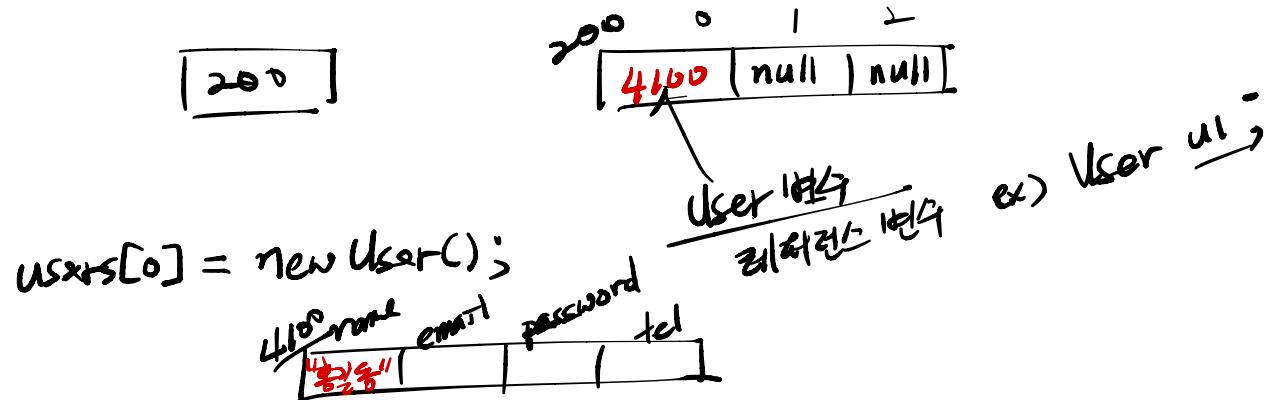
## \* 인스턴스 필드

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



\* 주의점

User[] users = new User[3];



user[0].name = "홍길동";

user[1].name = "임꺽정";

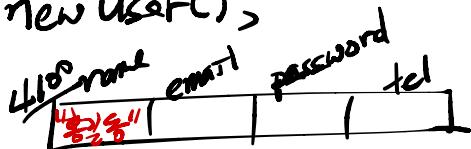
NullPointerException  $\xrightarrow{\text{null}} \text{ NullPointerException}$

\* 주소리스트 초기화

User[] users = new User[3];



users[0] = new User();

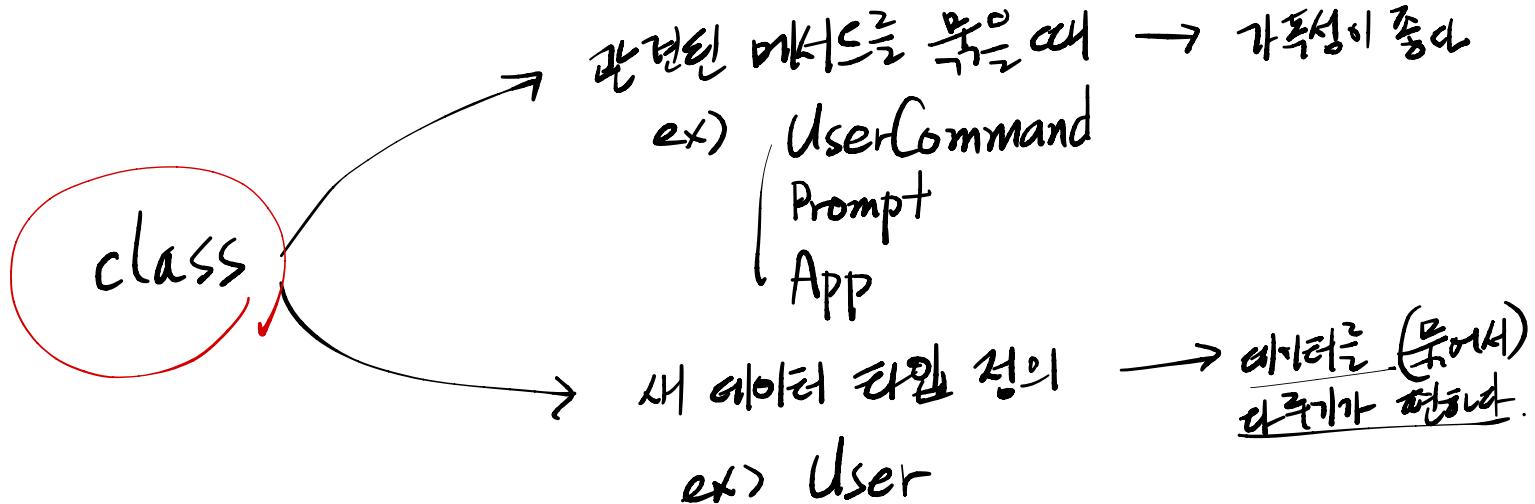


User user = users[0];

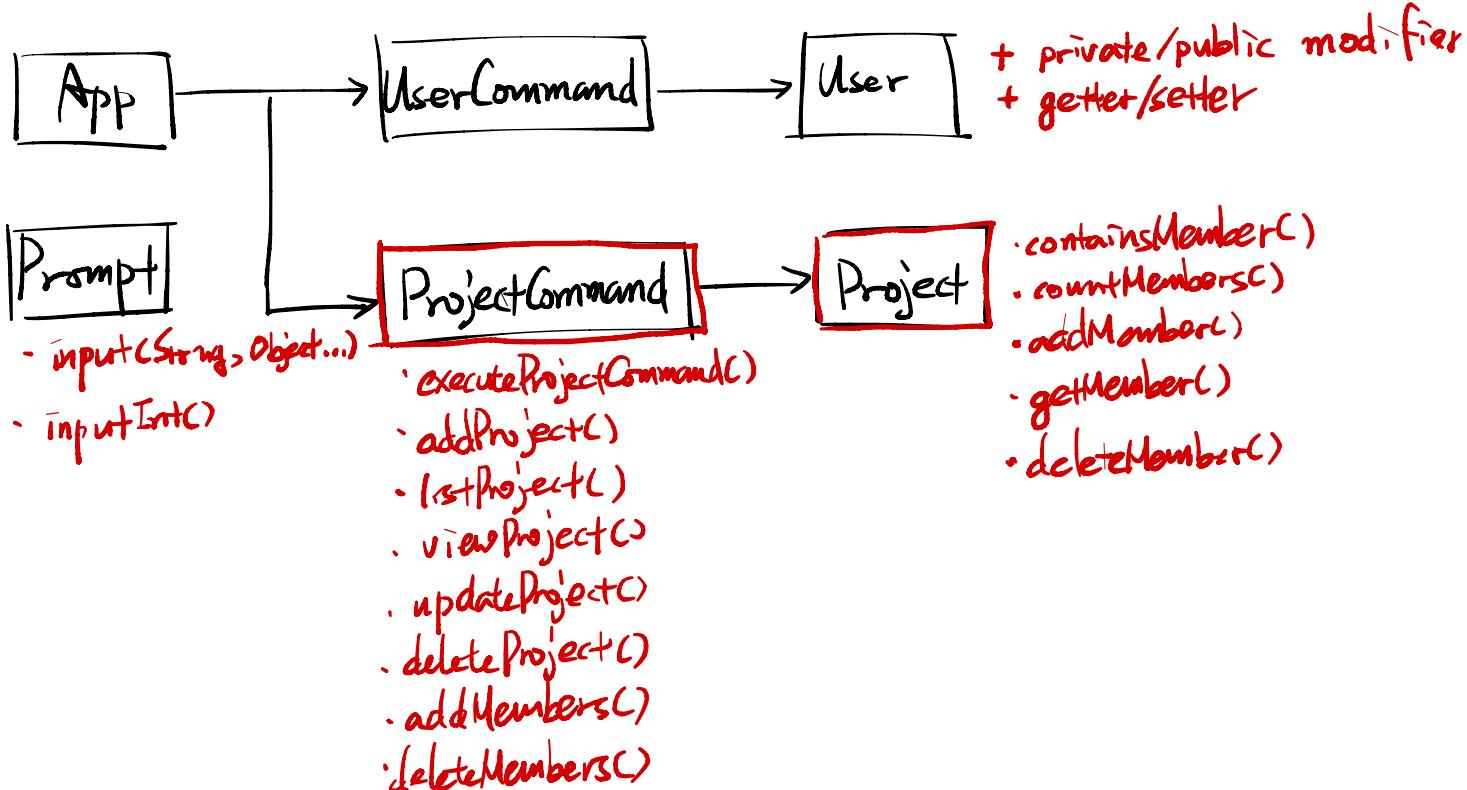


users[0].name = "aaa";  
user.name = "aaa";

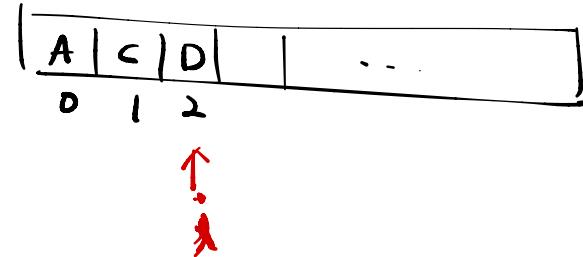
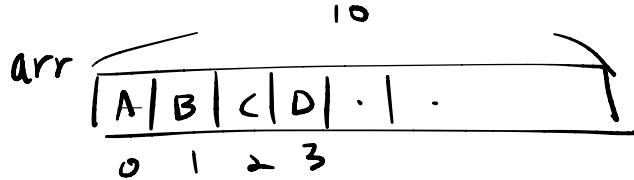
## \* 흔한 문법



## 10. CRUD 툴 - 캐스팅 CRUD



\* 배열의 항목을 연속으로 삭제할 때



B와 D를 삭제

```
for (int i = 0 ; i < arr.length ; i++) {
```

```
    if (arr[i] == 'B' || arr[i] == 'C') {
```

// 해당 인덱스 삭제

}

}

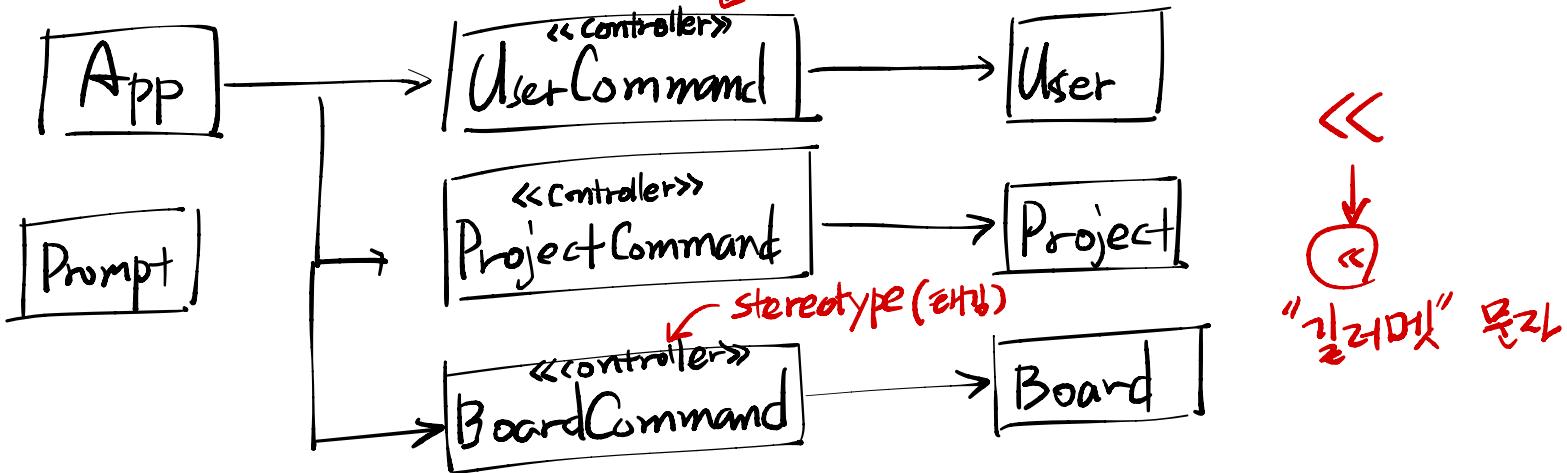
'C'가 앞에 올라오면서  
같은 인덱스에서 2개의 다른 원래 번호

↓  
해결책?  
마구로 반복

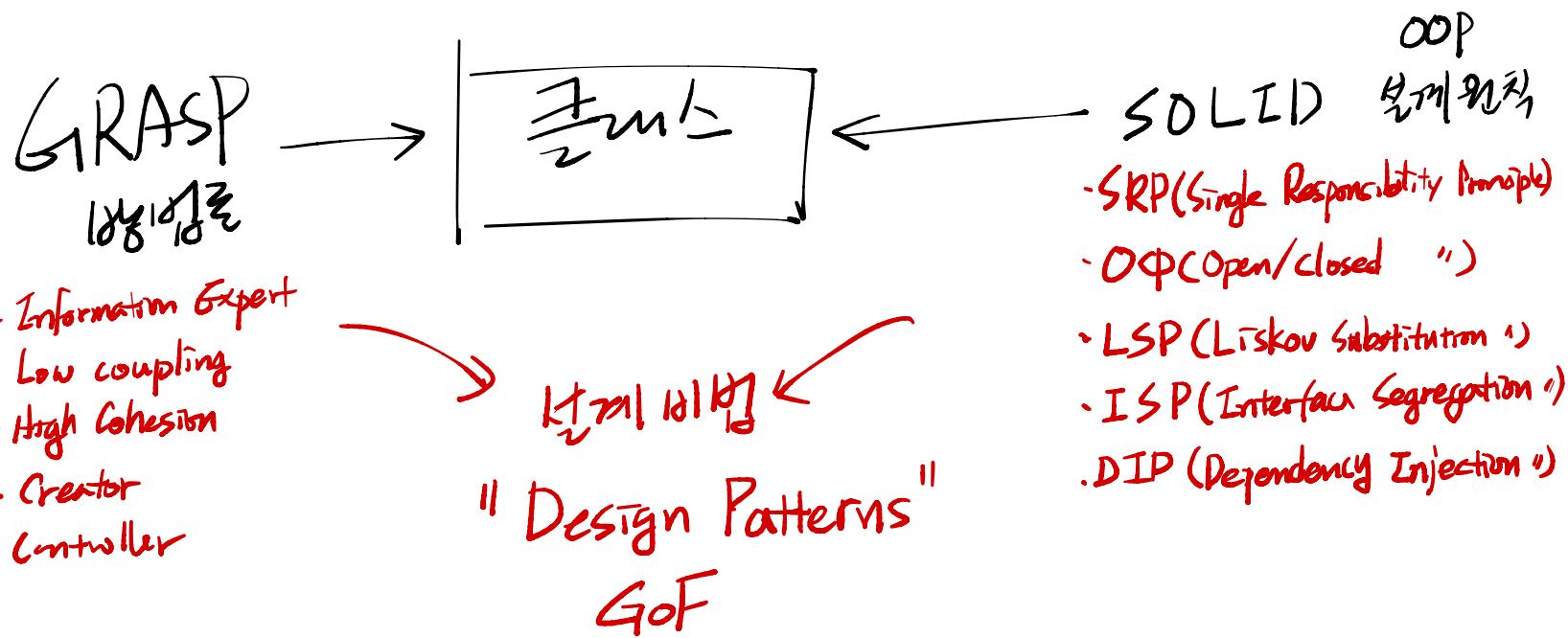
10. CRUD 투명성 - 거시적 CRUD

전통적인 흐름 / 관리 / 관리 / 관리 / 관리

role = 책임(responsibility)



\* 프로그램 설계 방법론  
→ 프로그램 설계 방법론에 대한 이해



\* 2023.01.21

ex) addProject()

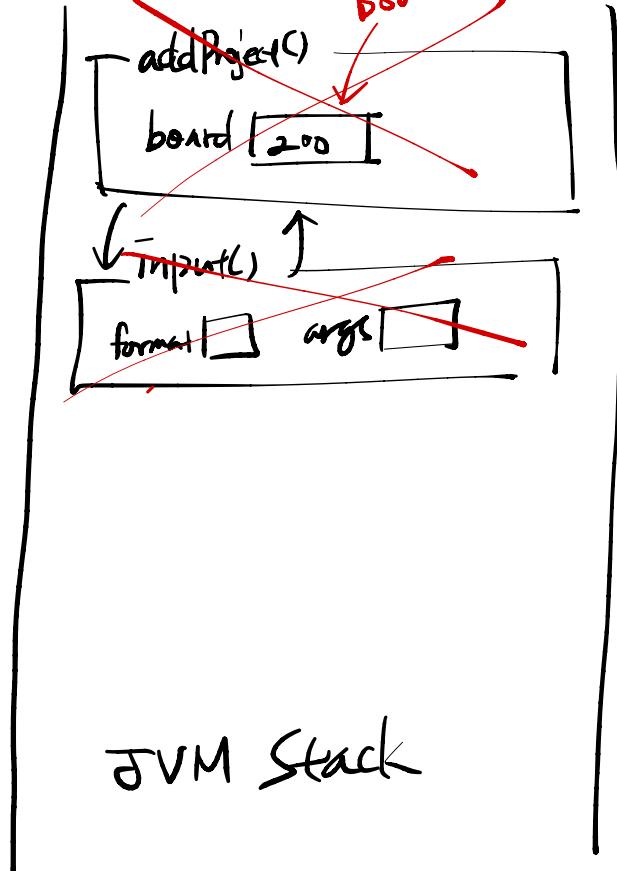
```
class BoardCommand {  
    addProject(){}  
    (addProject(){}  
}
```

```
}  
class Board {}  
class Prompt {}
```

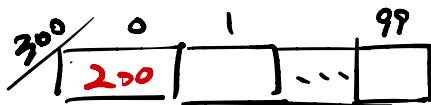
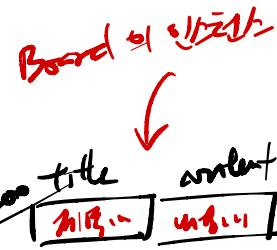
MAXSIZE boards boardlength  

100	300	0
-----	-----	---

Method Area



JVM Stack



Heap

\* 접근수준 초기화

class A {

static int v1;

void m1() {

Board b = new Board();

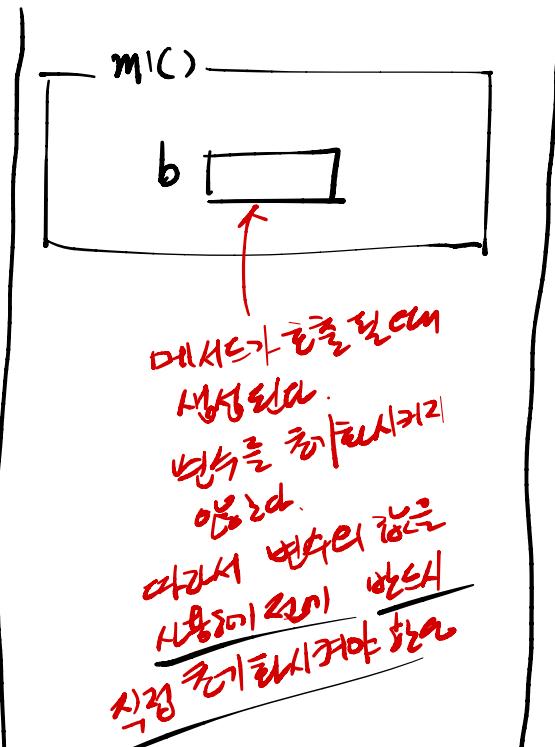
}

}

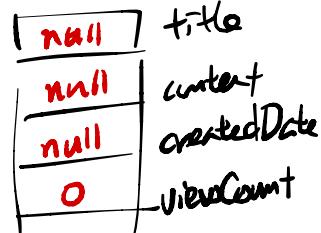


A는 static 멤버인  
v1은 static 멤버  
Board는 static 멤버  
기본 초기화

Method Area



JVM Stack



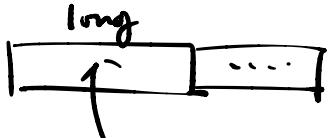
↑  
new Board  
null이 생성  
시작에 초기화  
가능하지만

Heap

\* new Date()

java.util

↓ field



1970년 1월 1일

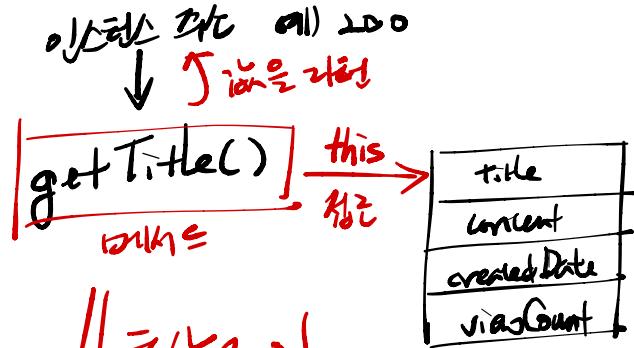
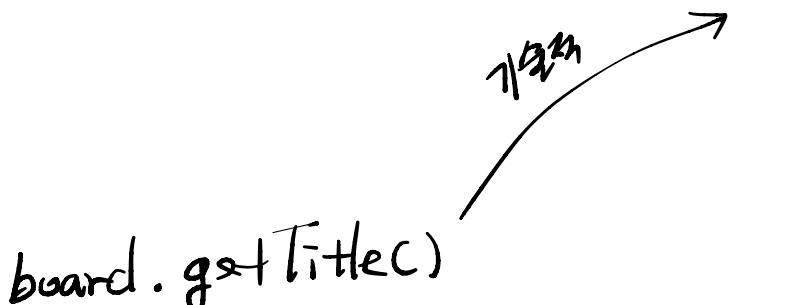
00:00:00 부터

현재까지 경과된

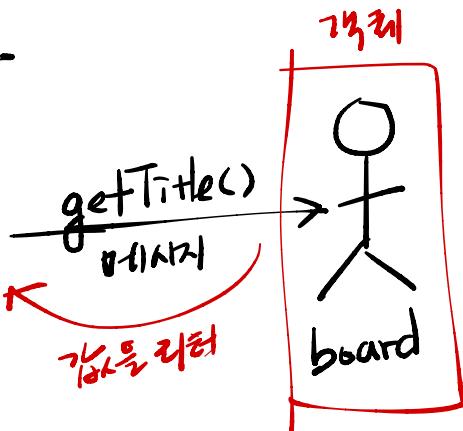
millisecond 수

값.

## \* 인스턴스 메서드 호출



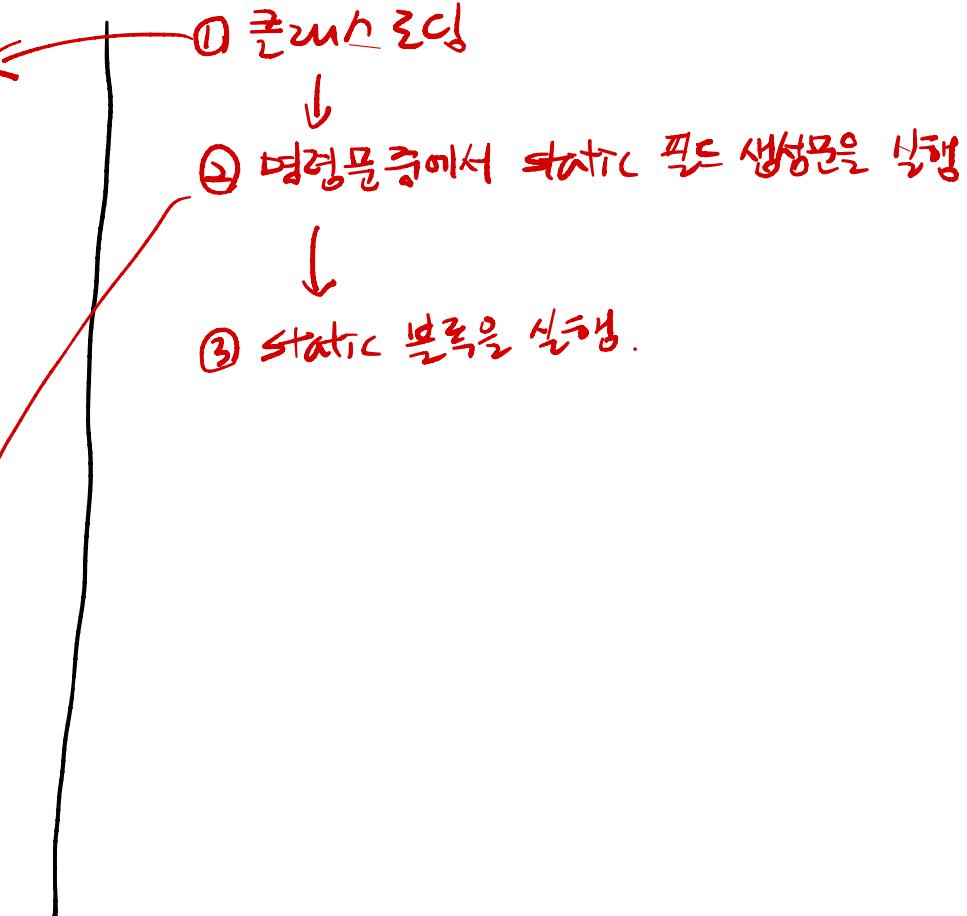
작성일  
등록일  
작성자



\* 변수 선언과 변수 → 메모리  
↳ 변수를 생성시키는 명령문

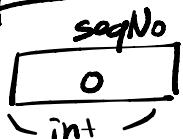
```
class User {  
    static int seqNo;  
  
    String name;  
  
    void m() {  
        int a = 100;  
    }  
}
```

seqNo  
0  
int



\* 변수 선언과 변수

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



Method Area

```
class Test {
```

```
void main() {
```

User obj = new User();

m();  
    ↑  
    생성시키는  
    메소드

이 클래스 변수 선언을  
    생성시키는  
    메소드

main()

args

obj

  ↓  
  |  
  |  
  |

  ↓  
  |  
  |  
  |

m()

a

  ↓  
  |  
  |  
  |

  ↓  
  |  
  |  
  |

  ↓  
  |  
  |  
  |

  ↓  
  |  
  |  
  |

  ↓  
  |  
  |  
  |

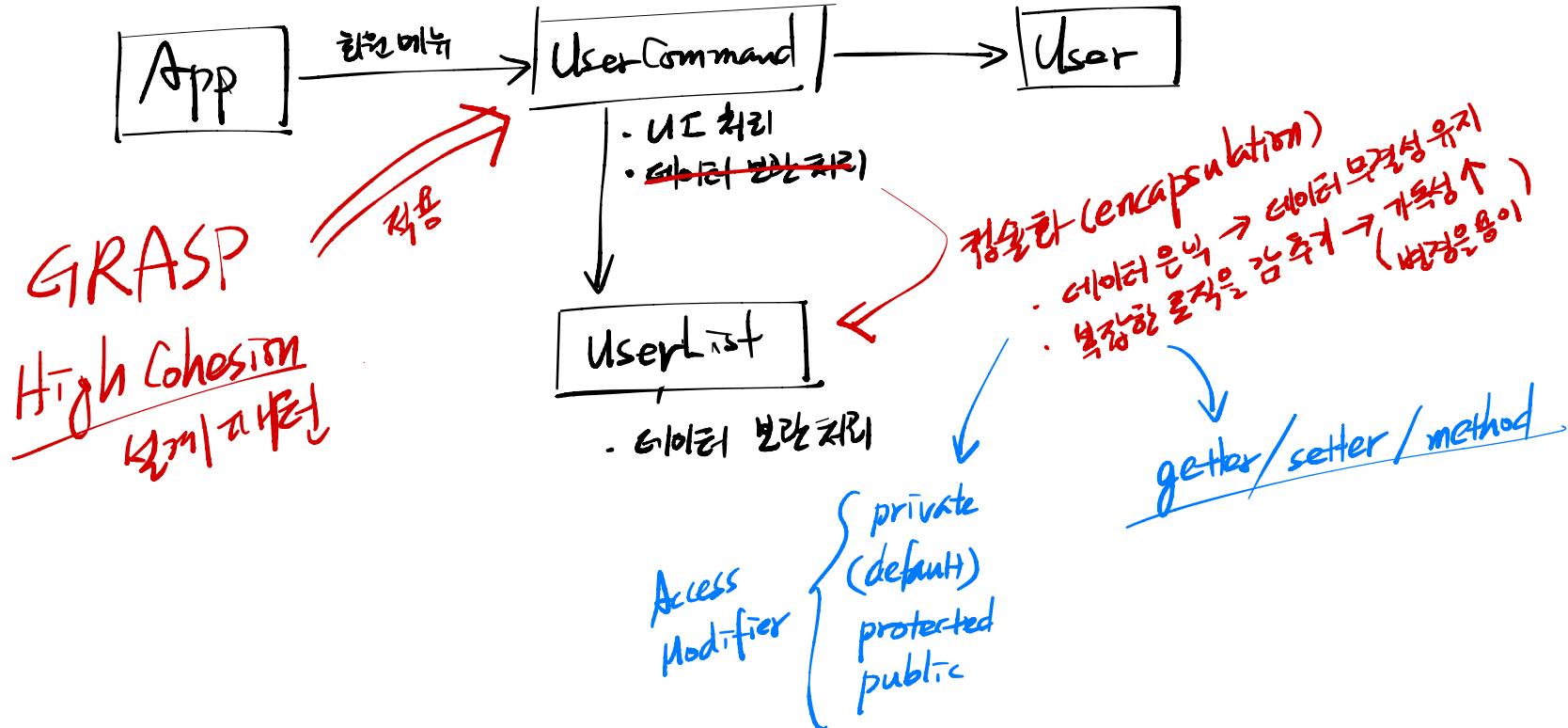
  ↓  
  |  
  |  
  |

JVM Stack



Heap

## 12. 인스턴스 속성을 다른곳에 분리하기



### 13. 클라우드의 핵심을 추적하기

1. 회원

2. 프로젝트

3. 개시판

4. 공지사항

5. 도와드릴

6. 풍선

Board Command  
Board List  
Board

증가할 내용은  
만들 수 있는가?

OK!



OK!

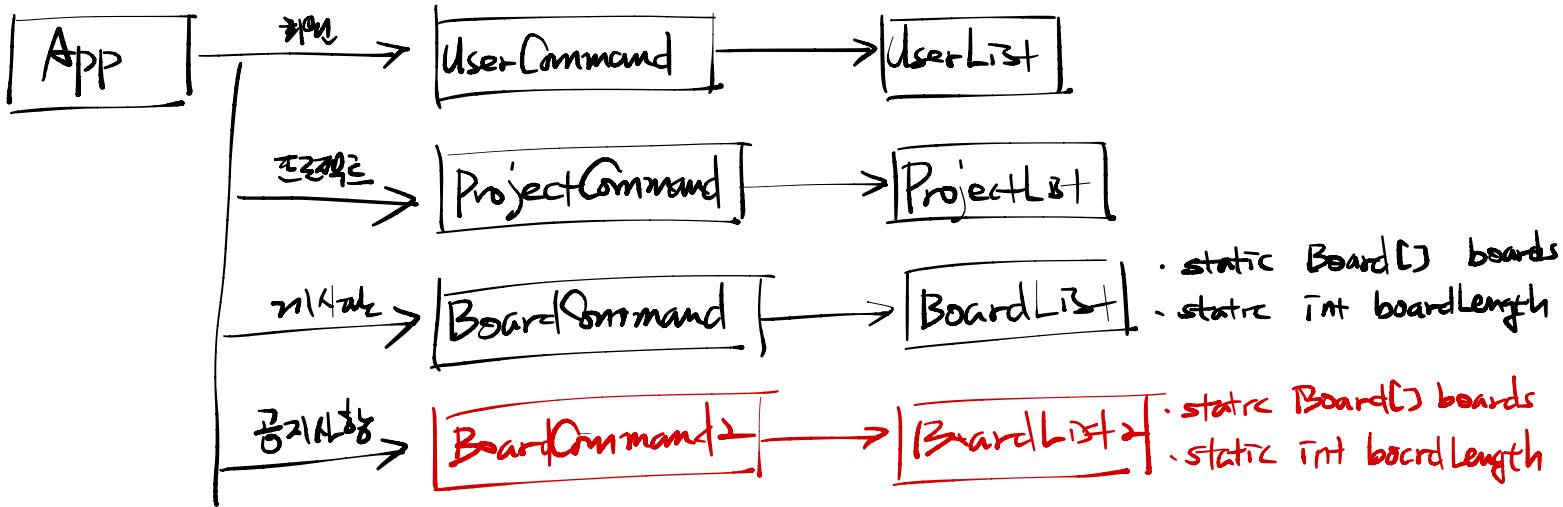
이전에 링크로  
연결!

어디로?

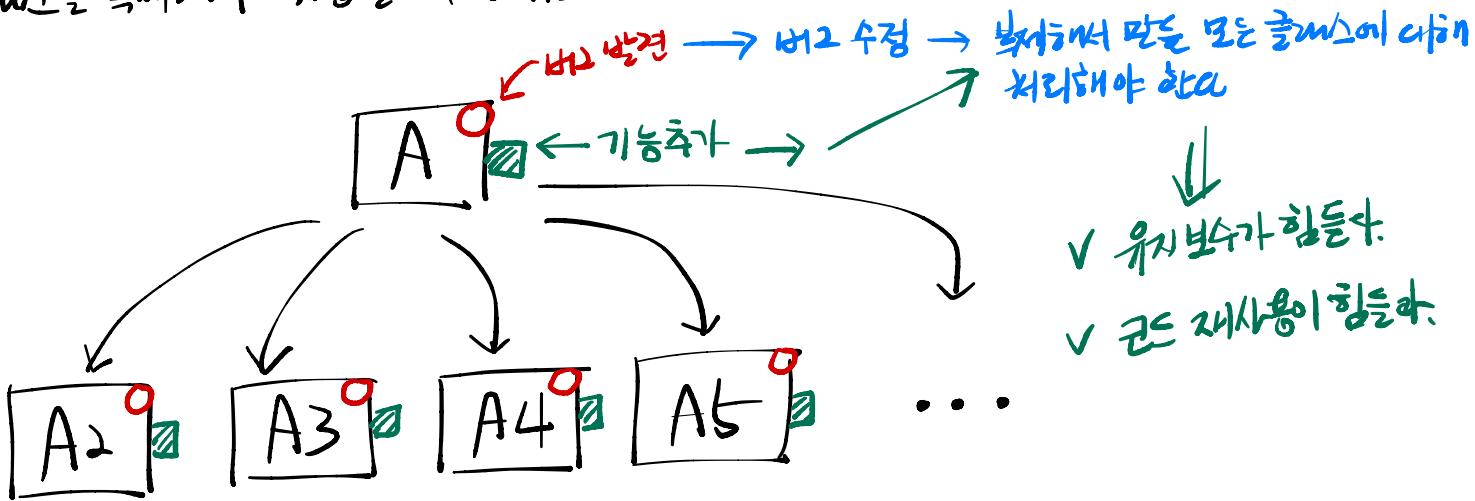


데이터를 저장하는 풋드가  
스티커이다!

## \* 디자인 차이



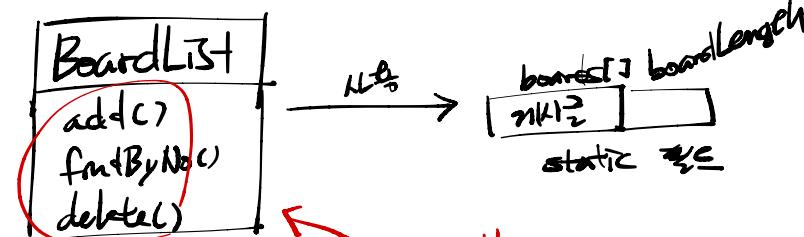
\* 클래스를 복제해서 사용할 때 문제점



\* 문제점

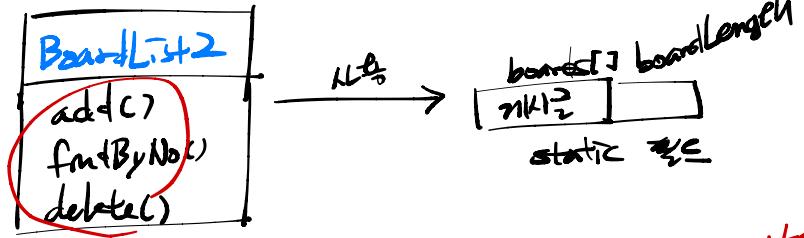
- 코드 중복 → 변경이 어렵다.

\* 해답이? 같은 변수를 사용, 그린데 이야리는 블록



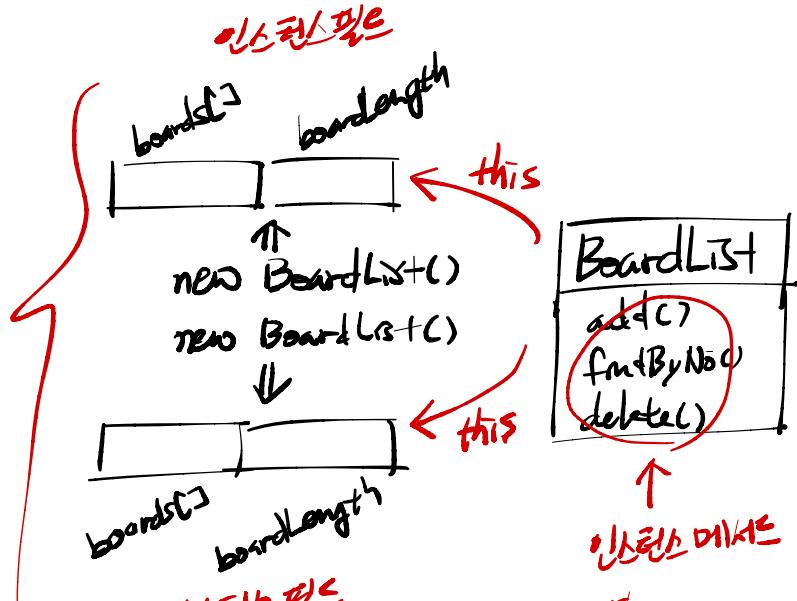
단락코드

코드가 중복



단락코드

자료구조

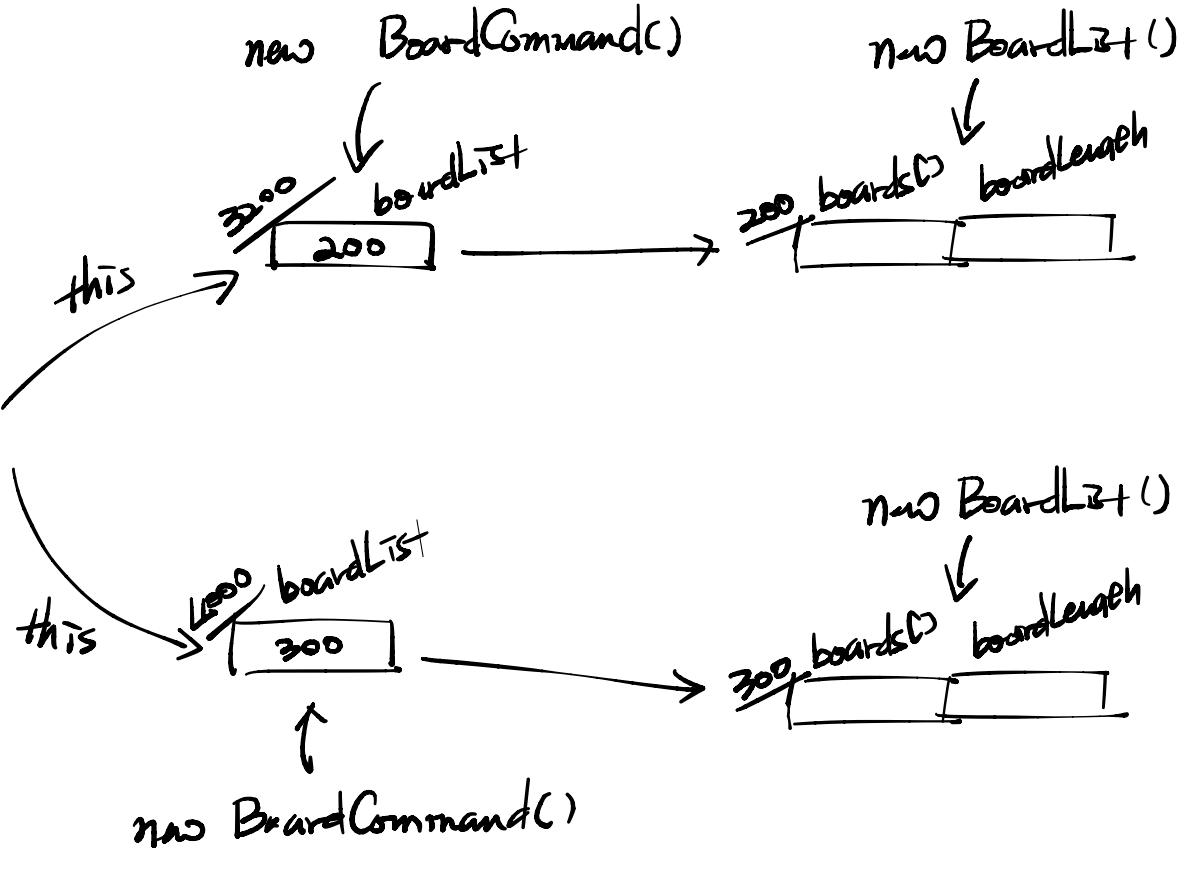
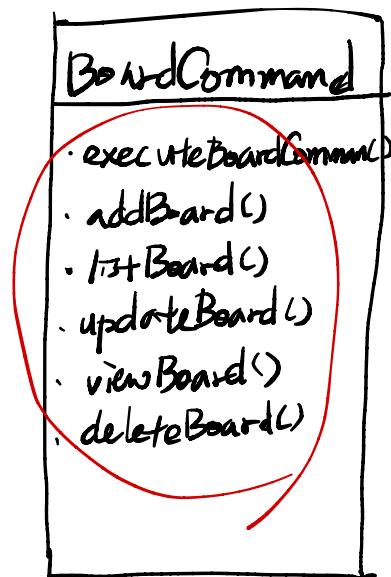


↑

인스턴스 메서드

↑

같은 코드



~~later~~  $\Rightarrow$  ~~already~~  
2nd

## \* 인스턴스와 메서드

A 클래스



A 클래스



레터런스 . 메서드();



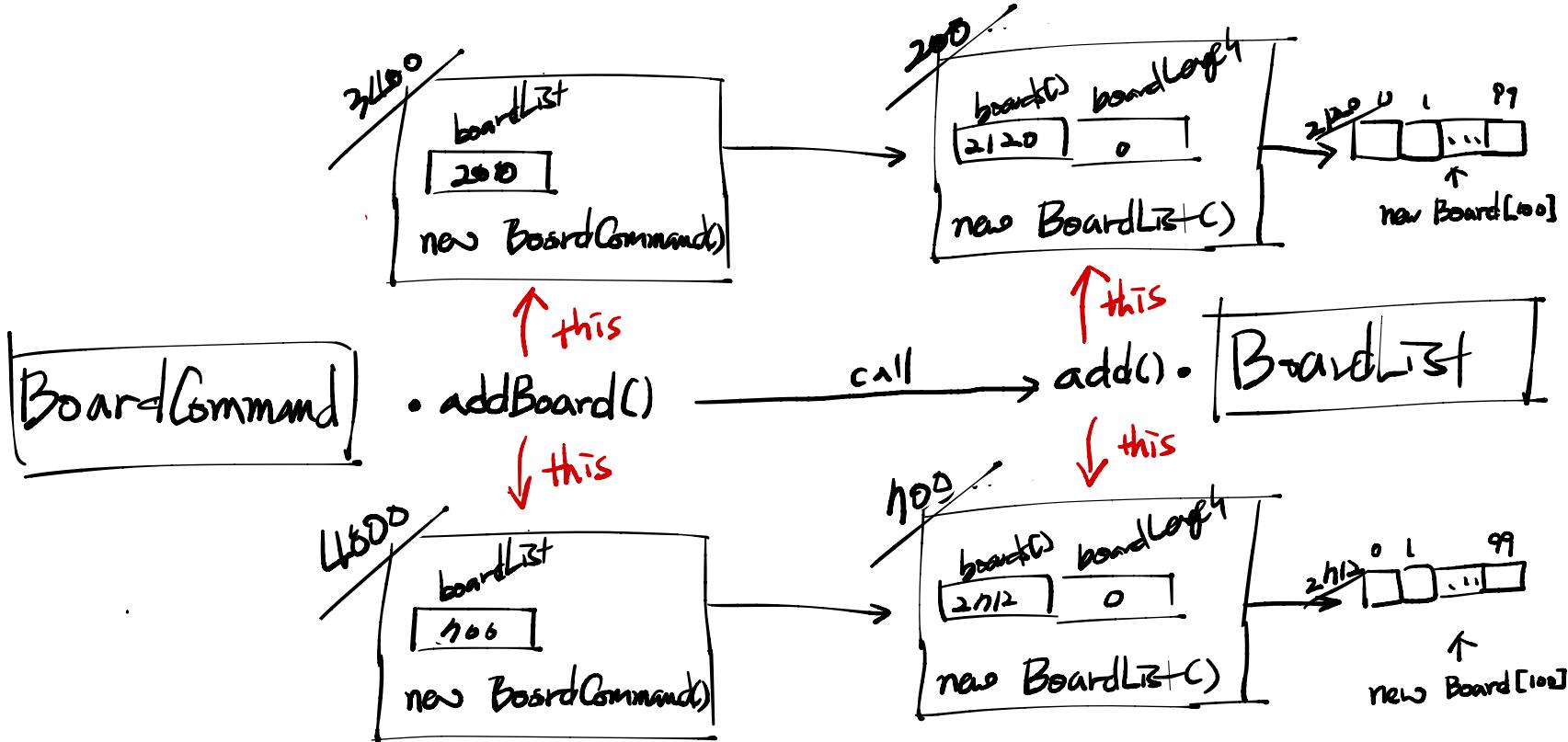
this 내장 변수는 메서드가 정의된  
클래스의 인스턴스 주소를 빙기 때문에

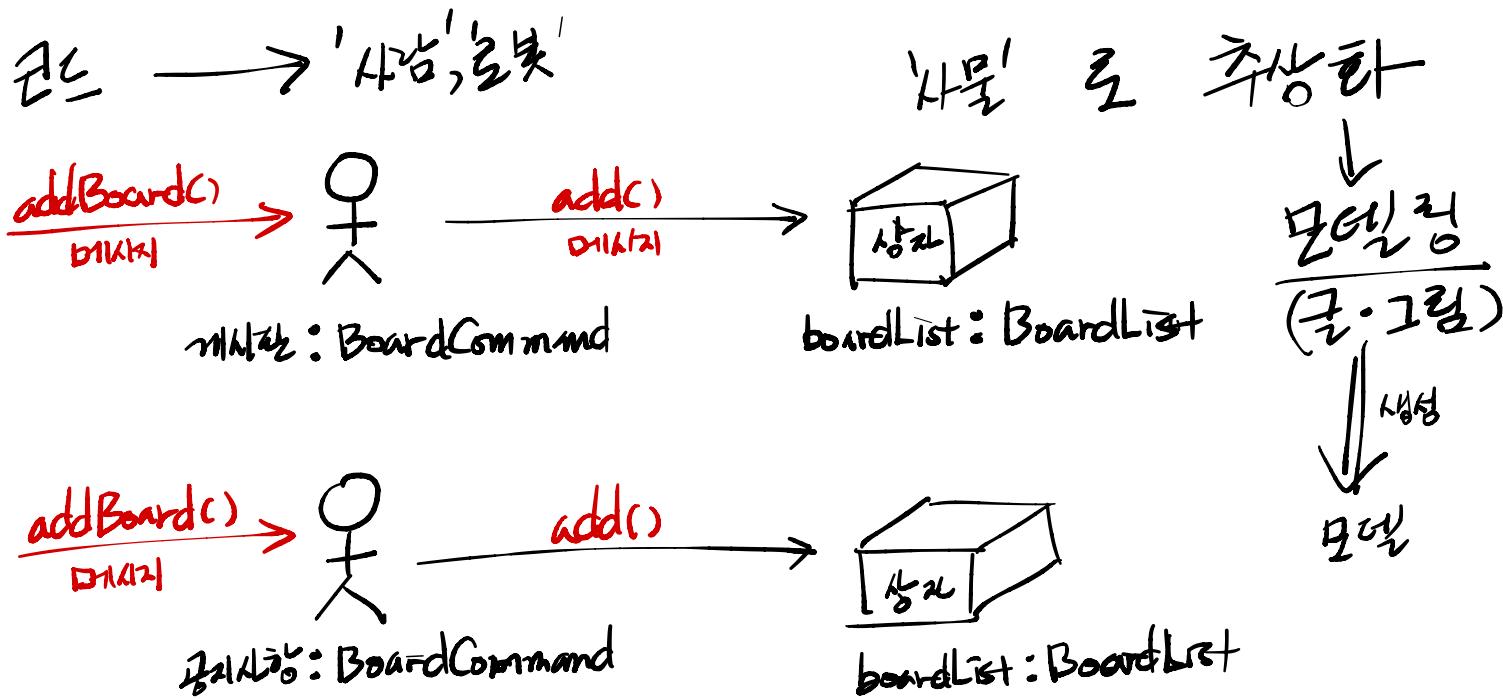
String s = "hello";

s~~++~~; < String 피연산자를 처리하는  
++ 연산자가 정의되어 있지 않다.

int i = 100;

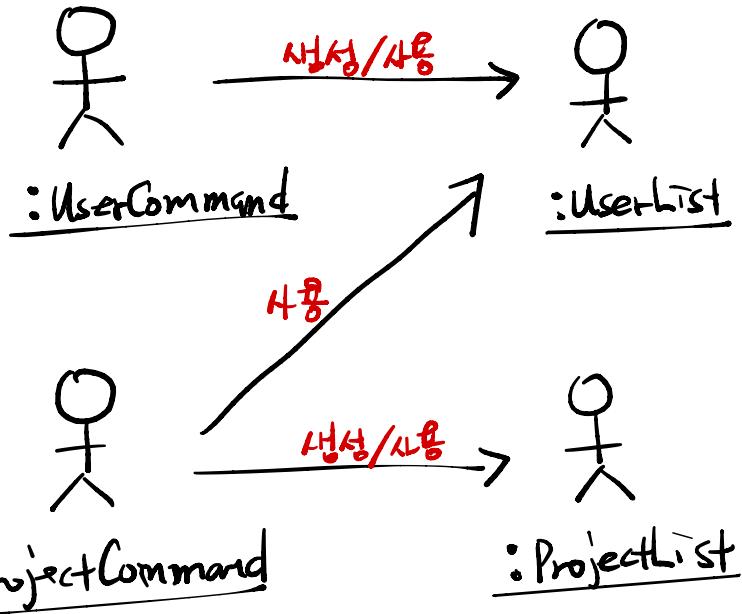
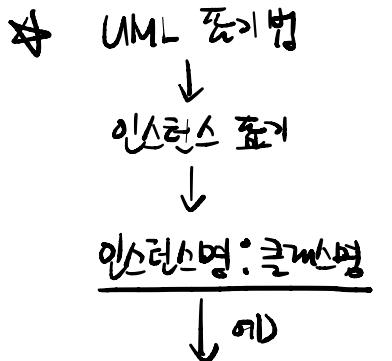
i++;





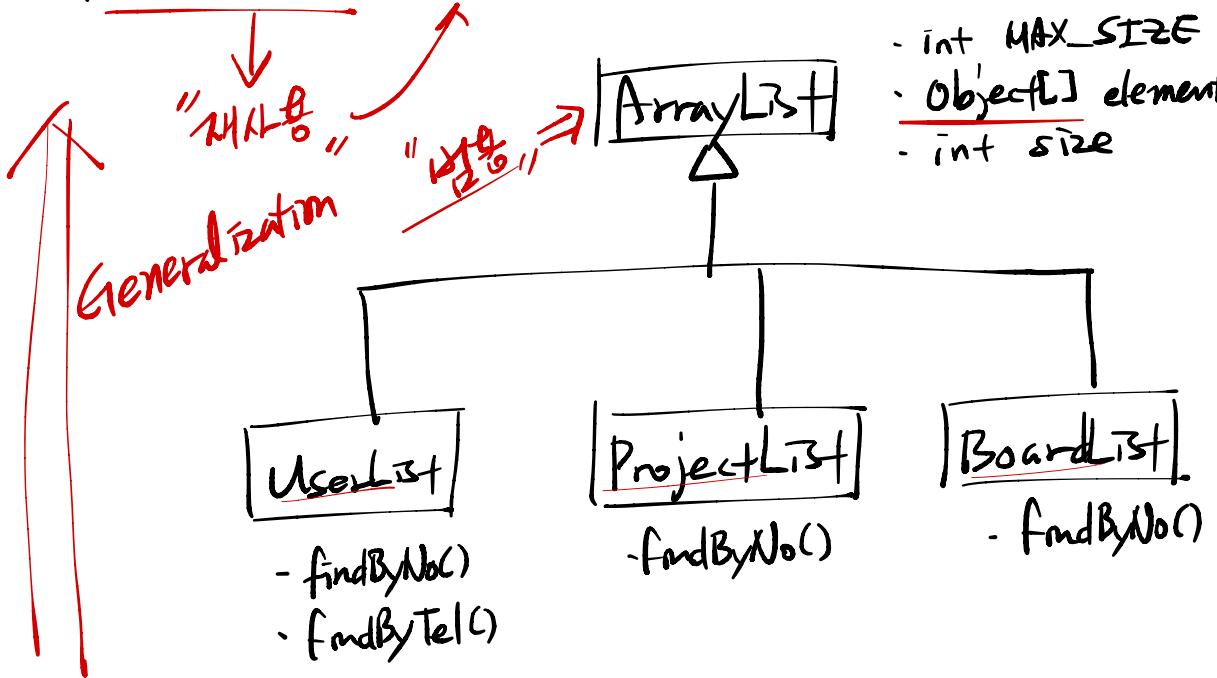
\* 인스턴스 고유

의존개체 (dependency)



인스턴스 이름을  
준기하지 않아도  
모델을 이해하는데  
문제가 없다면  
생략 가능

14. 상통근드 분리 및 사용하기 : 상속의 일반화(generalization) 기법



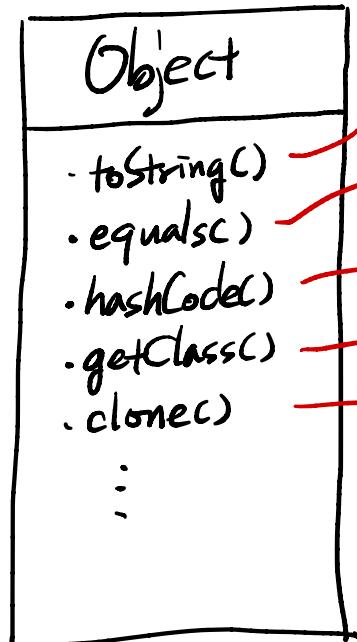
toArray()  
add()  
delete()  
indexOf()  
~~findByNo()~~  
↑  
일반화로 표기된  
클래스가 아니라  
데이터 구조로 표기  
변환을 사용하는  
방법이라는  
설명은 필요없다.  
그냥 사용하는  
수도 ArrayList이  
놓치고 싶다.

\* Object 클래스  $\Rightarrow$  자바의 최상위 클래스



✓ 자바의 모든 클래스는 Object의 하위 클래스이다.

✓ 클래스가 가지어야 할 기본 메서드를 정의하고 있다 제거지명 포함



→ 인스턴스의 상태를 문자열로 표현 : 리턴값 = "클래스명 @ 해시값"

→ 인스턴스가 같은지 비교 : 리턴값 = true / false

→ 인스턴스 식별번호 : 리턴값 = hash 알고리즘으로 생성한 int 값

→ 인스턴스의 클래스 정보 : 리턴값 = Class 객체

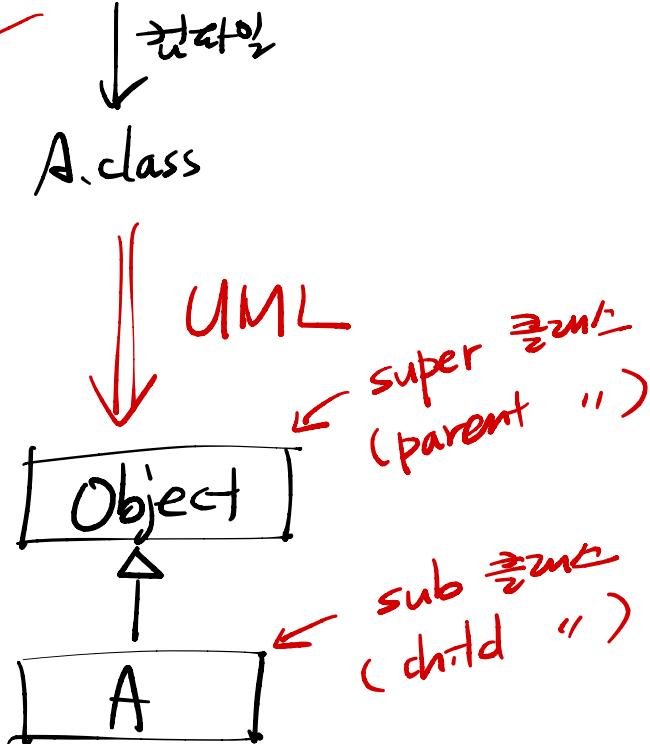
→ 인스턴스를 복제한 후 생성된 객체 : 리턴값 = 인스턴스 주소

\* Object을 상속할 때 상속관계

class A {}       $\xrightarrow[\text{부모}]{} \text{class A extends Object} {}$

내가 자식임  
"Object의 코드를 사용한다" 의미  
↓ 가상적인 용어

"Object를 상속 받는다"



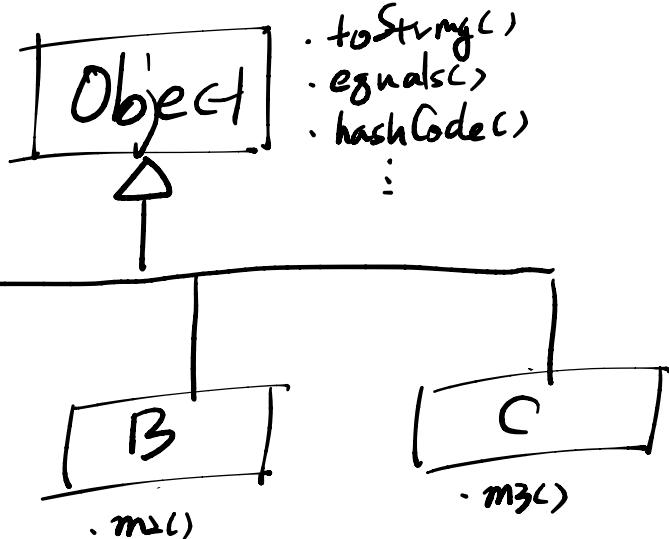
\* Object वर्ग

class A {  
 void m1() {}  
}

class B {  
 void m2() {}  
}

class C {  
 void m3() {}  
}

extends Object  
Object वर्ग से वृत्त

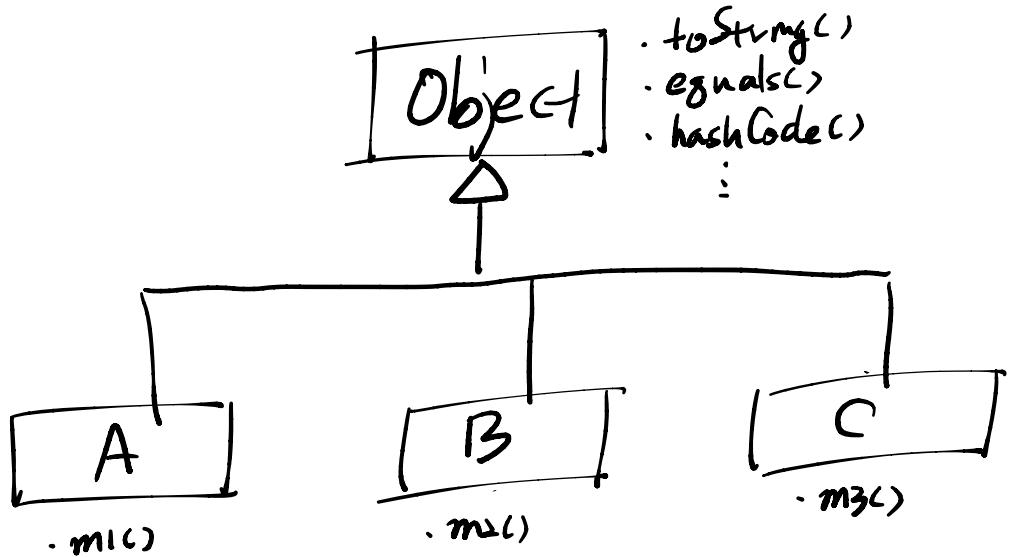


\* 히터 더그/ 골드 캐스팅

```
class A {  
    void m1();  
}
```

```
class B {  
    void m2();  
}
```

```
class C {  
    void m3();  
}
```



A obj = new AC();

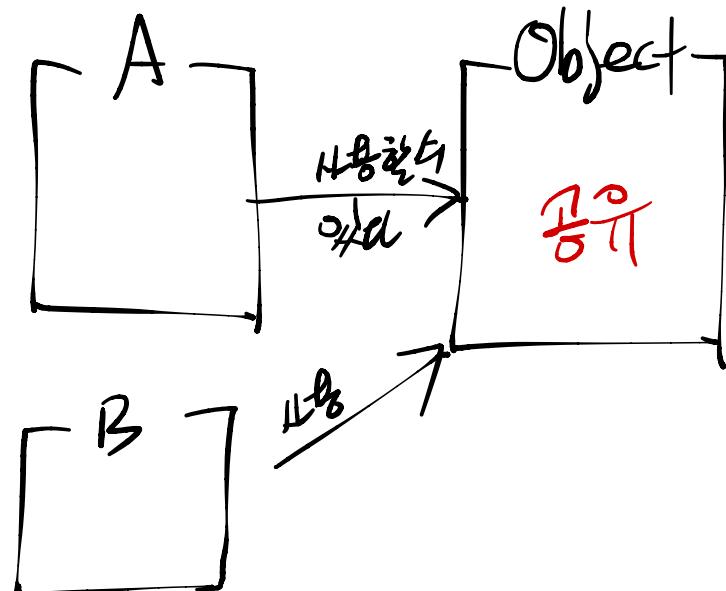
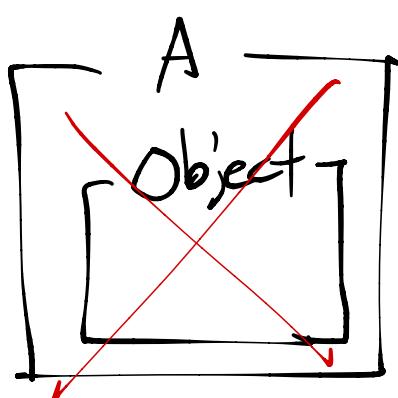
obj.m1(); → A.m1()

obj.toString(); → A.toString() → Object.toString()

\* 상속의 의미  $\Rightarrow$  코드를 가지 않는다  
공유한다

class A extends Object {}

class B extends Object {}



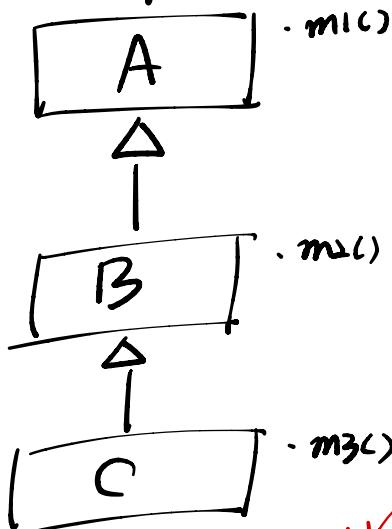
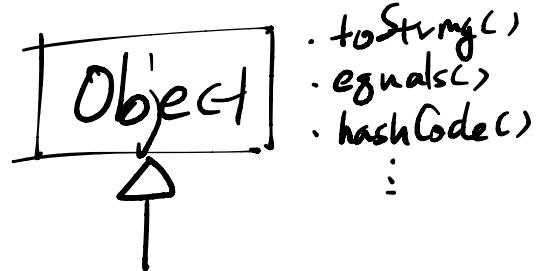
## \* 特別な例

```
class A {  
    void m1(){}  
}
```

```
class B extends A {  
    void m2(){}  
}
```

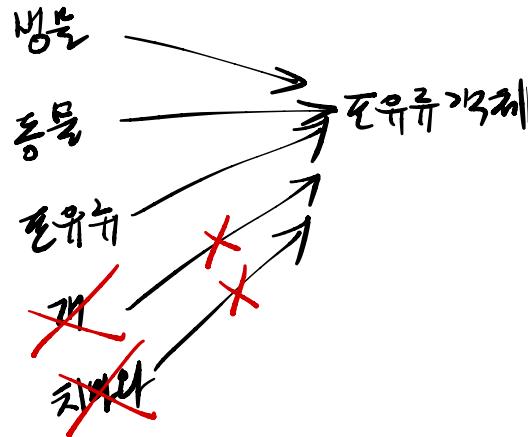
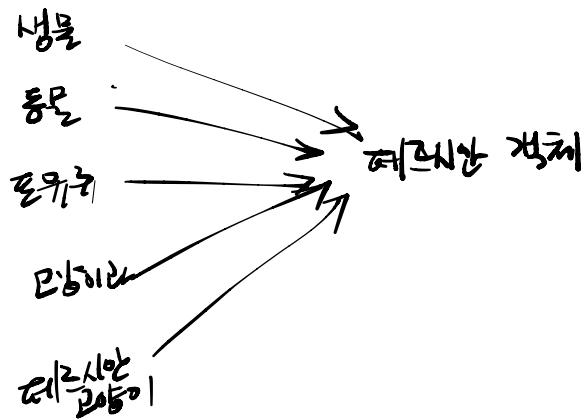
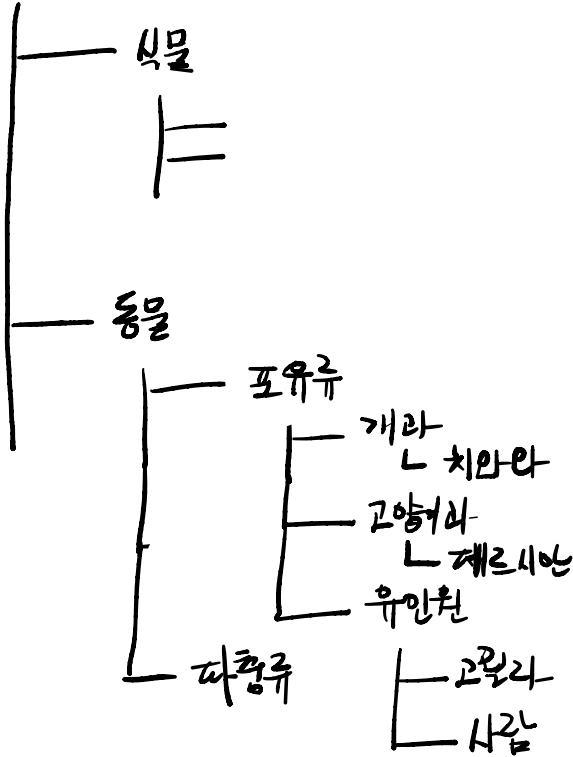
```
class C extends B {  
    void m3(){}  
}  
  
C obj = new C();  
obj.toString();
```

$\hookrightarrow C.\cancel{\text{toString}}() \rightarrow B.\cancel{\text{toString}}() \rightarrow A.\cancel{\text{toString}}() \rightarrow \text{Object}.\text{toString}$

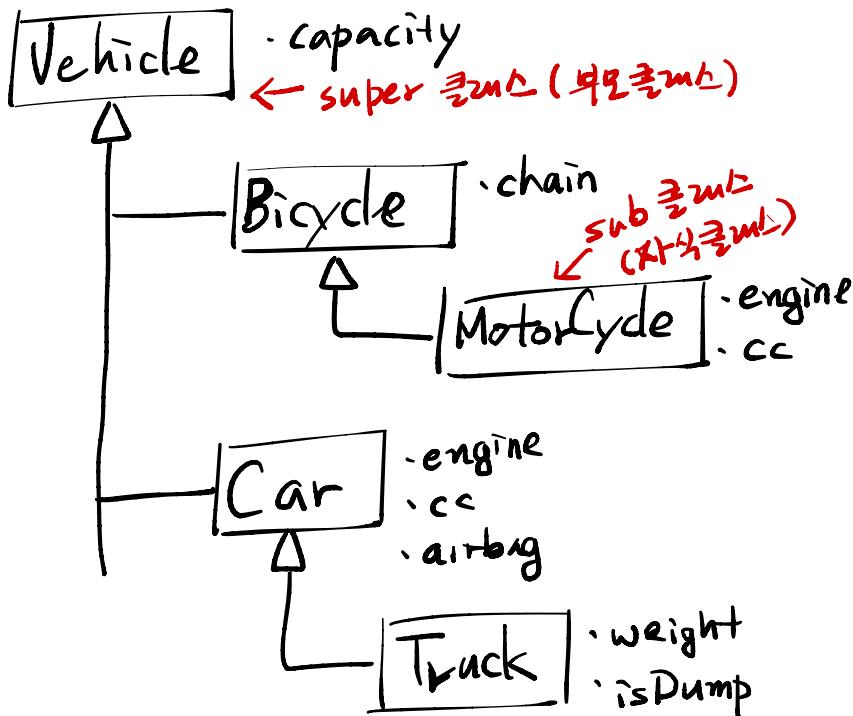


## \* 봉류 계통도와 흐름

생물



\* 자신의 클래스 계층도와 관련됨



`MotorCycle m = new MotorCycle();`

`Bicycle bi = new MotorCycle();`

`Vehicle ve = new MotorCycle();`

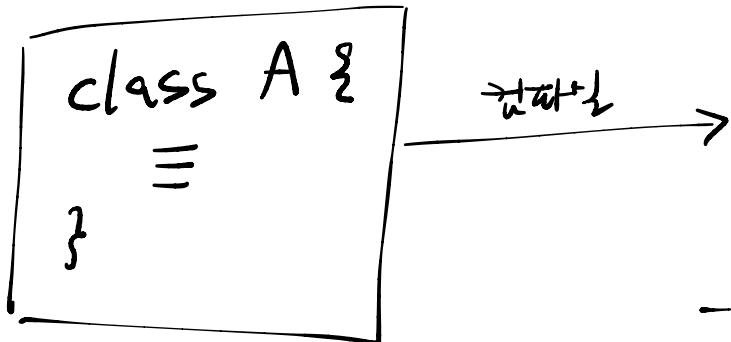


상위 클래스의 데이터 멤버는  
하위 클래스의 인스턴스를 쓸 수 있다

"상위 클래스의 하위 분류 개체를  
가지칠 수 있다"

*java.lang*

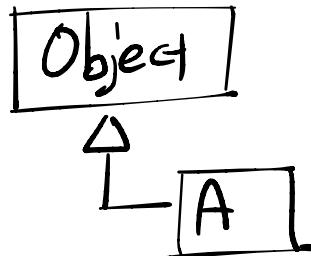
\* Object 클래스



super class

\* 결론

✓ Java의 모든 클래스는  
Object의 자손이다.



Object obj = all 인스턴스;  
↑ 제이터

\* 개발자의 코드 퀴즈 풀기

void m1( Object obj ) { }

↑ m1() 메서드를 만든 개발자의 의도는  
자리에 차지해 놓은 어떤 클래스의 인스턴스로는 올바르지 않다!

Object m2() { }

↑ 이 메서드는 어떤 대상의 인스턴스 주소를 리턴할 것이다.  
메서드를 호출하는 상황이 아니라 어떤 클래스의  
인스턴스를 리턴하는지 반드시 리턴값을 말하라!

\* 1. 품목구조화된 클래스의 상속

[ArrayList]

- Object[] list
- int size
- add()
- remove()
- indexOf()
- get()
- toArray()

[UserList]

- findByNo()

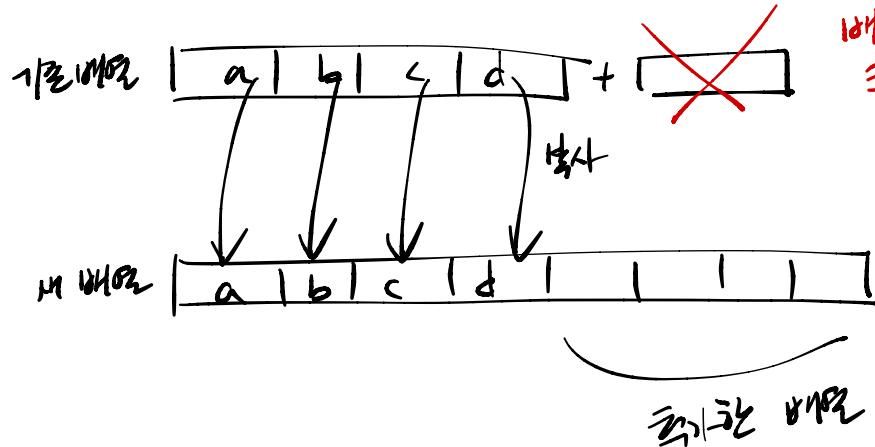
UserList obj = new UserList();

super constructor  
size method.



obj.add(); → ArrayList.add()  
*this*

## 15. 배포 크기를 자동으로 증가시키기



배포로 다른 사용할 때 크기로 결정된다.  
크기를 변경할 수 있다

기존 배포보다 더 큰 배포 사용  
기존 배포의 경우는 확장이 어렵다

## 16. Linked List 자료구조 주제

① 메모리 관리

장점: 메모리 크기 고정

↓  
인덱스를 이용한 접근이 용이

단점: 메모리 크기를 늘리기 힘들다

↓  
내 배열을 만든 후  
각 배열을 뒤에 붙여

↓  
가변적(가변)이 성능된다

수행, 브레이크 타임  
함수의 값을 빼고, 넣기  
제작하기 때문에  
실행 속도가慢(느리다)



② Linked List 특성

데이터 구조에서 사용이 많아 소요

↑ 단점

Linked List 특성으로

데이터를 저장

✓ 메모리를 늘리기 용이

✓ 수행 속도가 빠르다

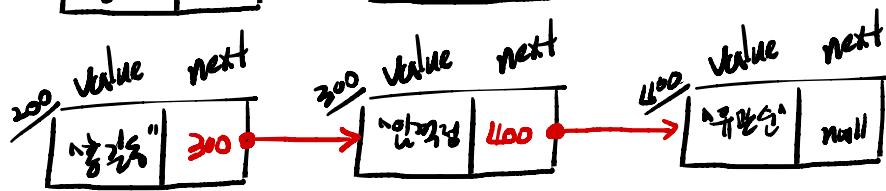
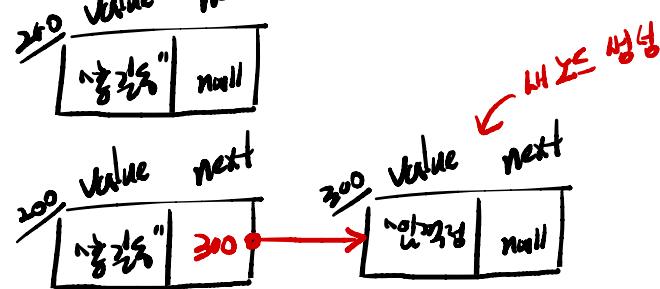
\* Linked List의 구조 관리

① 0123

list.append("김민기")

list.append("김민경")

list.append("유관순")



\* Linked List의 주동 원리

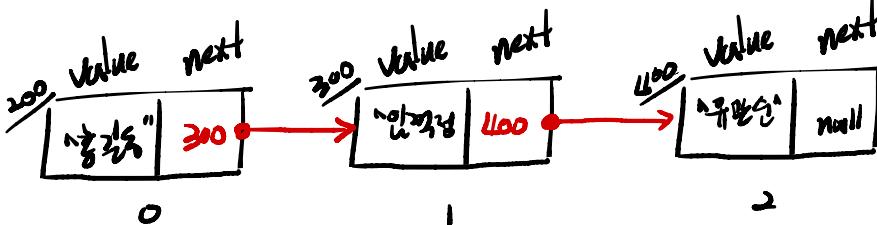


② 조회

list.getValue(0)  $\Rightarrow$  "200"

cursor [200]

currentIndex [0]



list.getValue(1)  $\Rightarrow$  "300"

cursor [300]

currentIndex [1]

$$\frac{\text{cursor} = \text{cursor.next}}{400}$$

list.getValue(2)  $\Rightarrow$  "400"

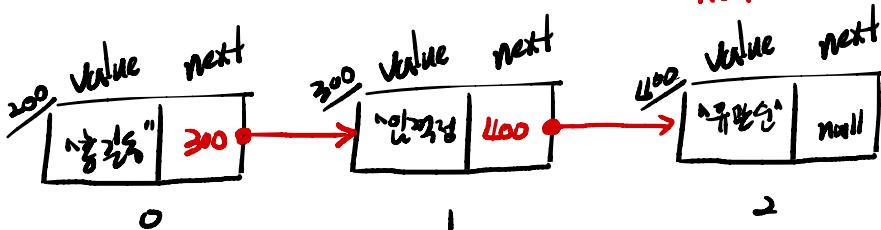
cursor [300]  $\xrightarrow{300 \rightarrow 400}$  400

currentIndex [1] + 2

\* Linked List의 주요 원리

### ③ 삽입 - 첫번째 노드

list.delete(0)



list.delete(0)

first = first.next;

list.delete(0)

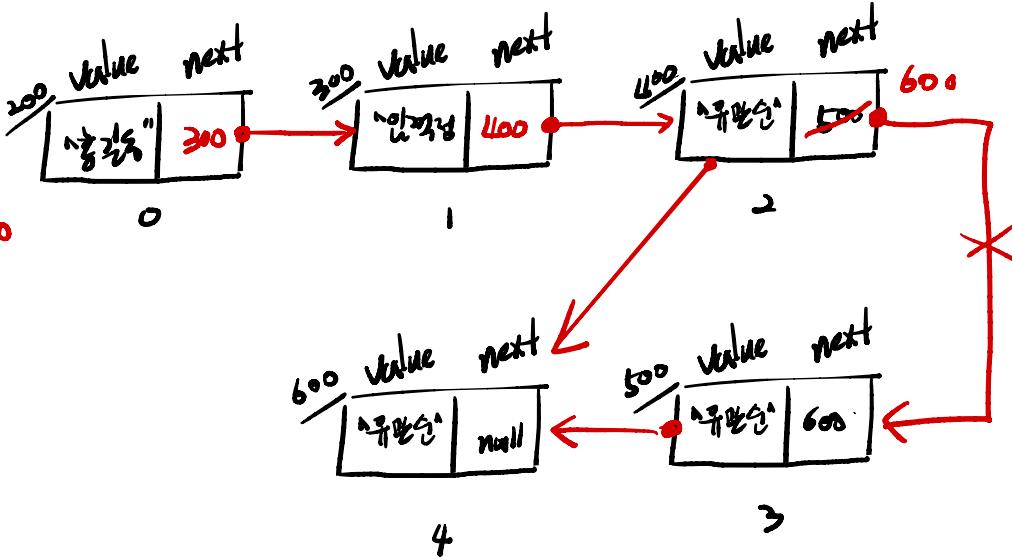
\* Linked List의 주요 원리

### ③ 삭제 - 중간 노드

list.delete(3)

cursor | ~~300~~ 300 400

currentIndex | ~~0~~ + 2



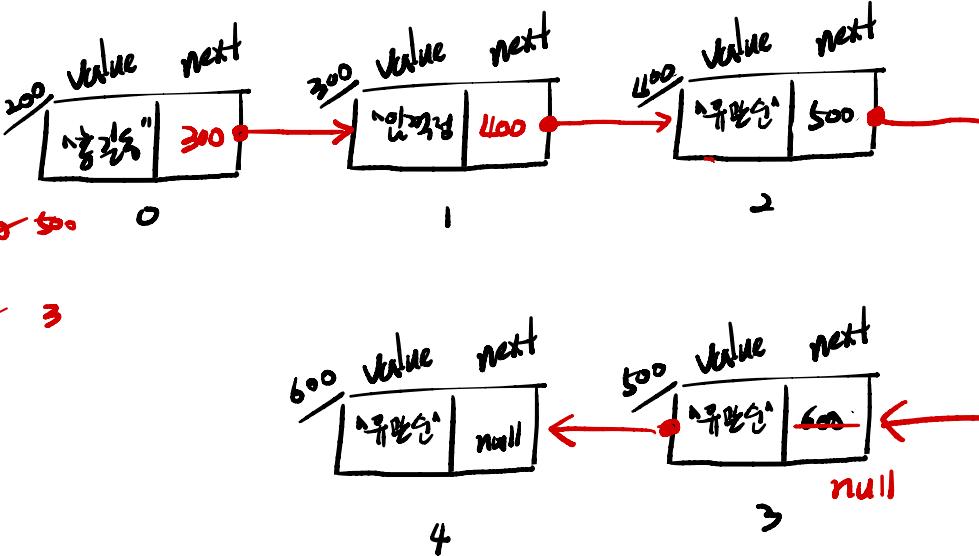
\* Linked List의 주동 원리

③ 삭제 - 0지역코드

list.delete(4)

cursor | ~~300~~ 300 400 500 0

currentIndex | ~~0~~ + 2 3

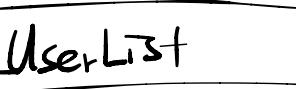
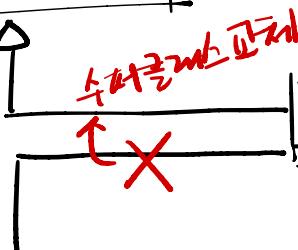


cursor.next = cursor.next.next;

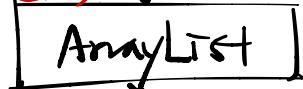




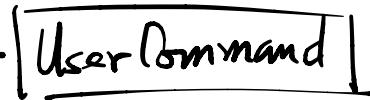
- append()
- delete()
- getValues()
- index()
- toArray()



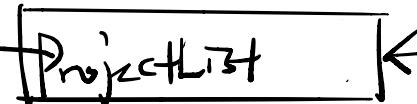
- findByName()



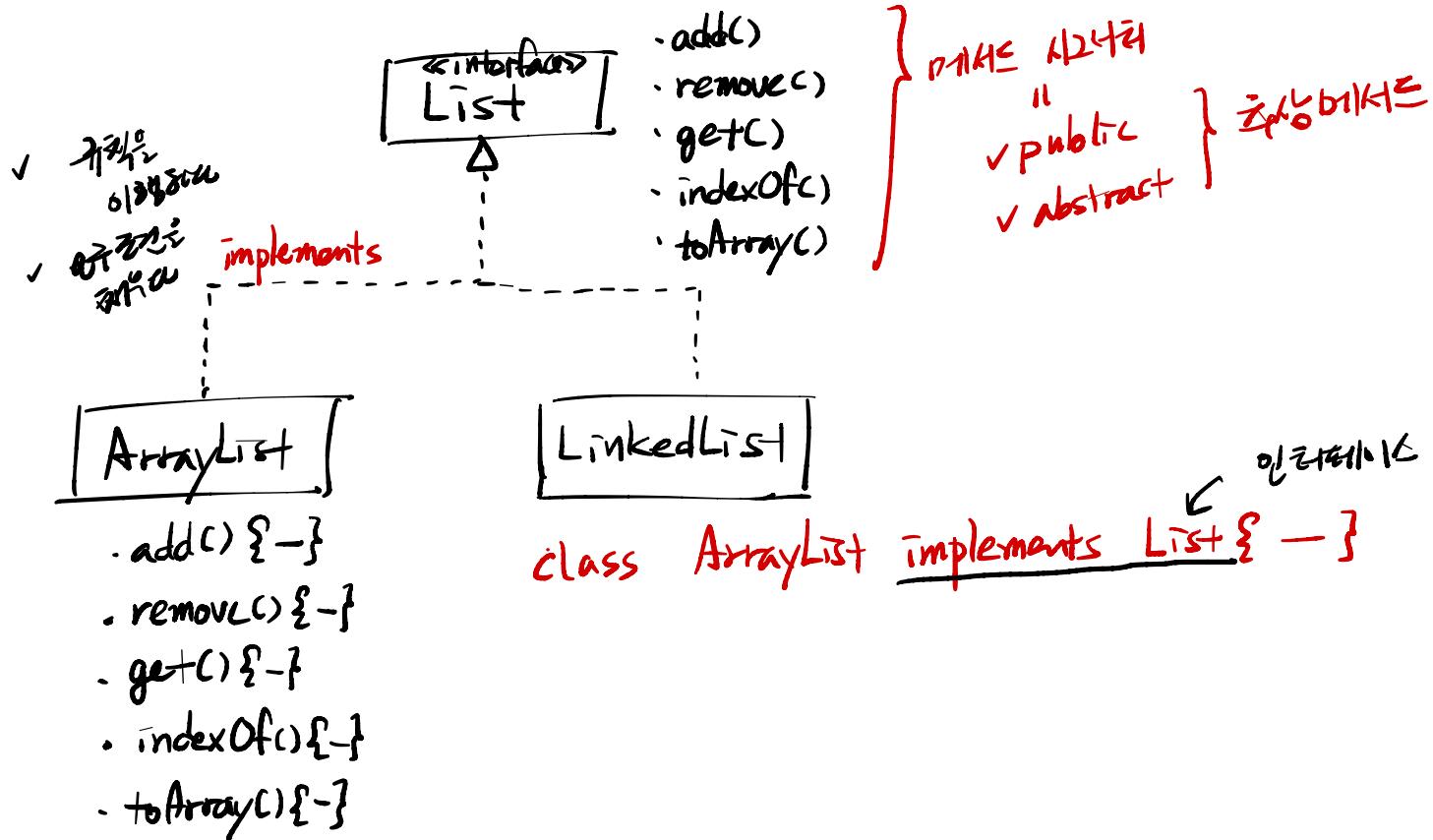
- add()
- remove()
- get()
- indexOf()
- toArray()



UserList의 수퍼클래스 표지에  
자주 사용됨(마스터드래그)이  
알라딘기 때문에 군드변경 허용!

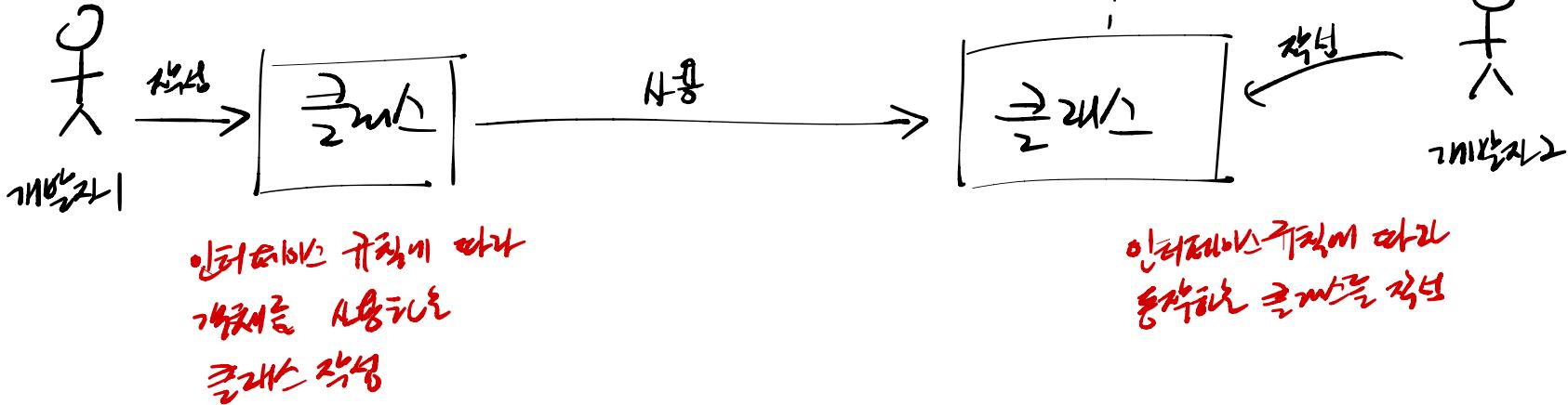
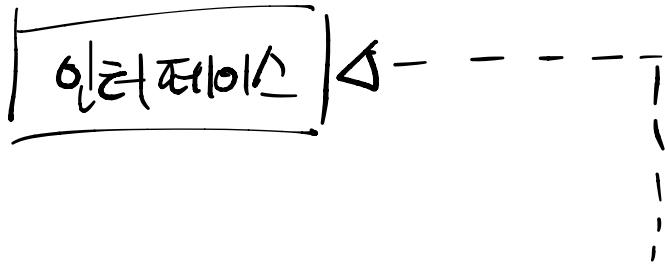


## 17. 인터페이스를 이용 목록을 다루는 구조의 사용 예제 보기



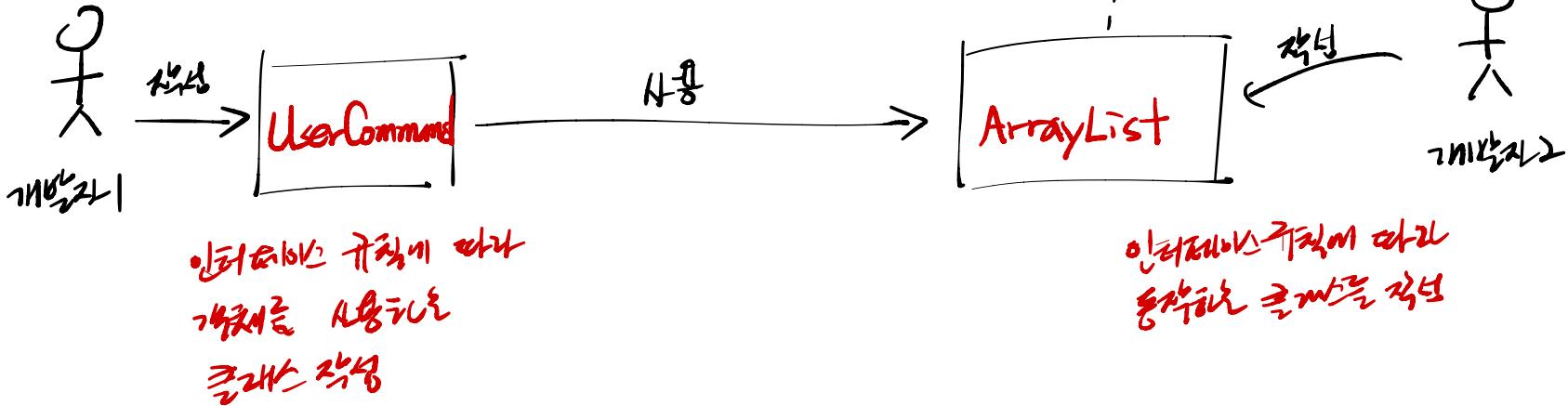
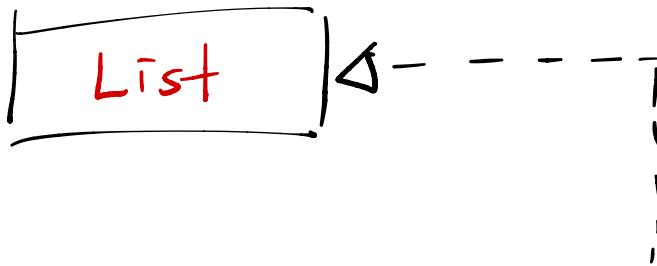
## \* 인터랙이스

기본 사용자인 경우에 보면  
 ① 프로그램의 일반성이 만족할 수 있다.  
 ② 교체가 가능하다.

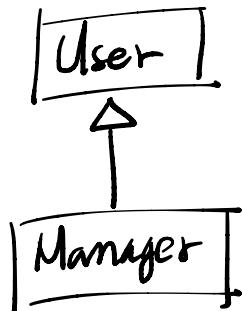


## \* 인터페이스

인터페이스는 구현체 없이  
제공하는 기능의 만족도가 높다.



## \* instanceof vs getClass()



`equals() {`

```

equals(Object obj) {
    if (this.getClass() != obj.getClass())
        return false;
}
  
```

Line ①

Line ②

```

if (!obj instanceof User)
    return false;
}
  
```

```

User u1 = new User();
Strong str = new Strong();
User u2 = new User();
Manager m = new Manager();
  
```

	obj 1	obj 2
<code>u1.equals(str)</code>	F	F
<code>u1.equals(u2)</code>	T	T
<code>u1.equals(m)</code>	F	T