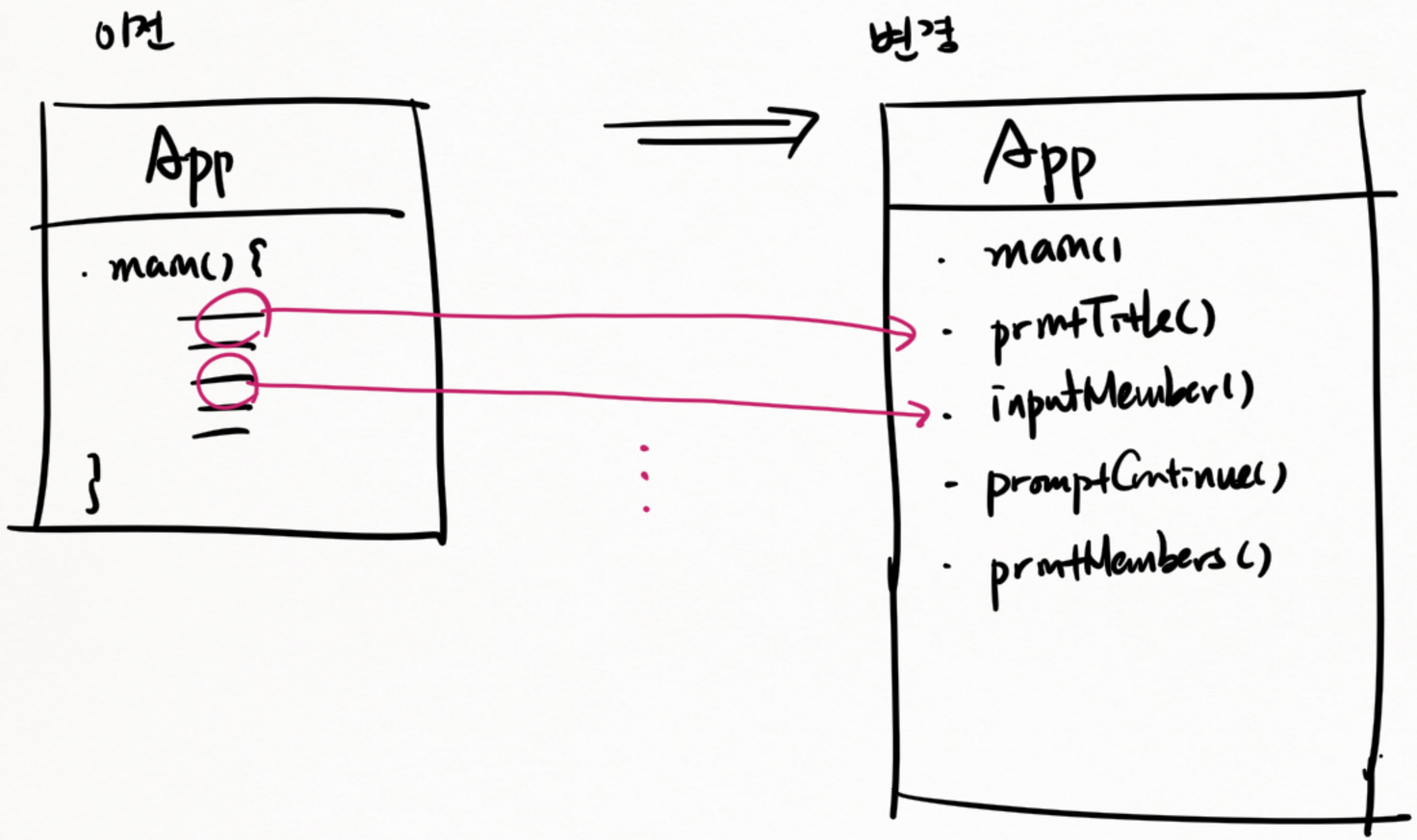
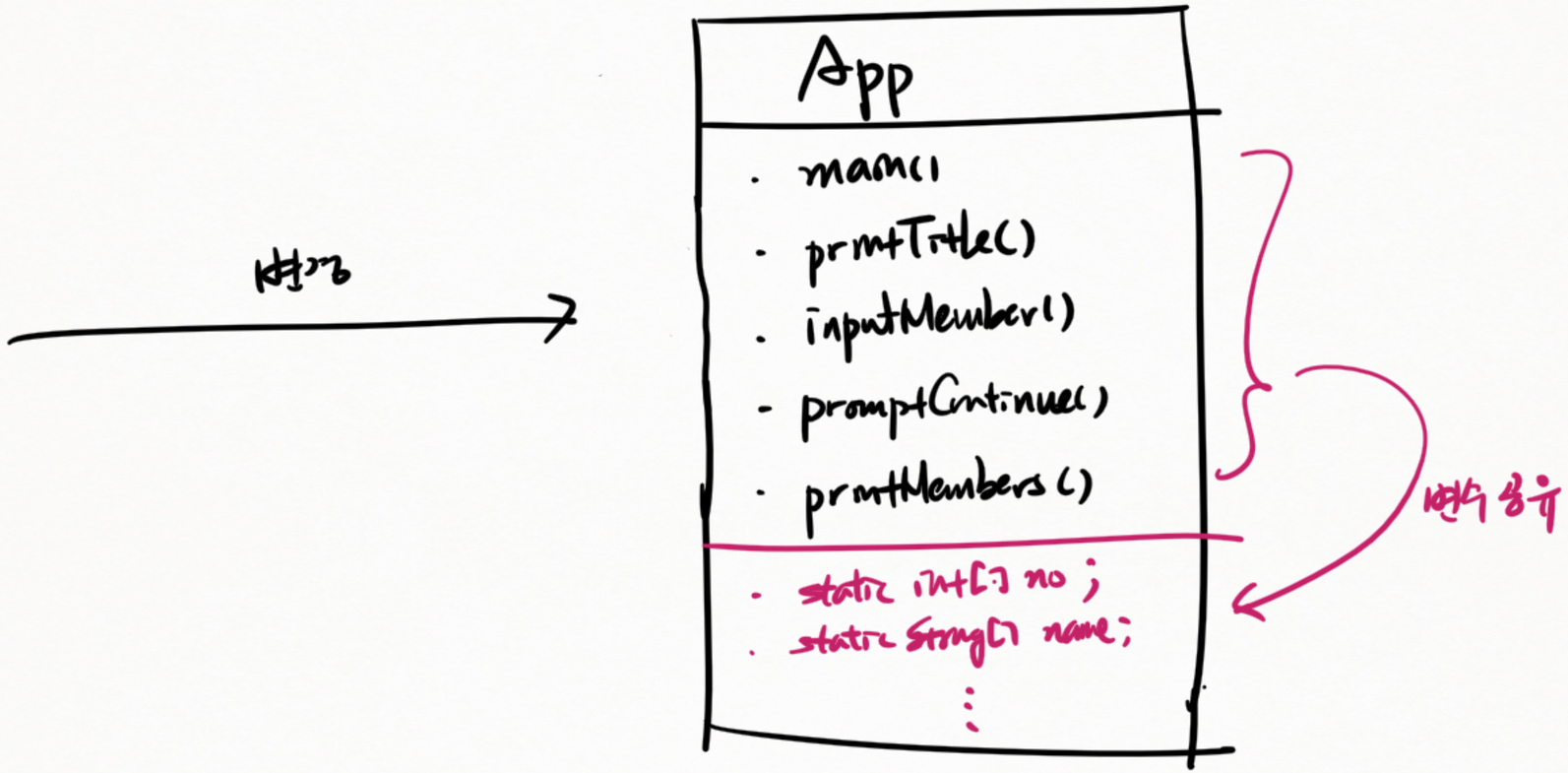
* 7. 메서드 사용법


이전
App
. main() {
}
변경
App
. main()
. printTitle()
. inputMember()
. promptContinue()
. printMembers()

* 응. 스태틱 변수 사용법


변경
App
. main()
. printTitle()
. inputMember()
. promptContinue()
. printMembers()
. static int[] no;
. static String[] name;
⋮
변수 공유

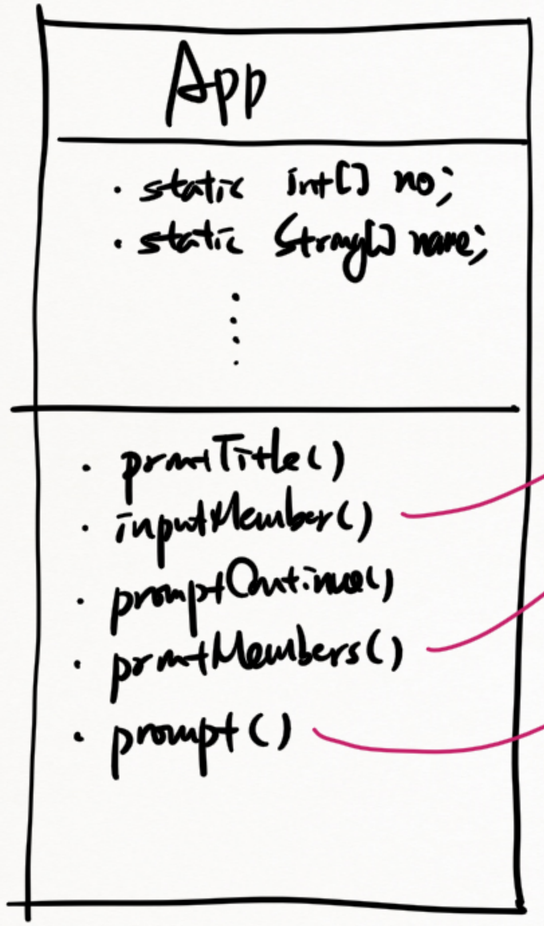
# * 9. 클래스 및 패키지 사용법

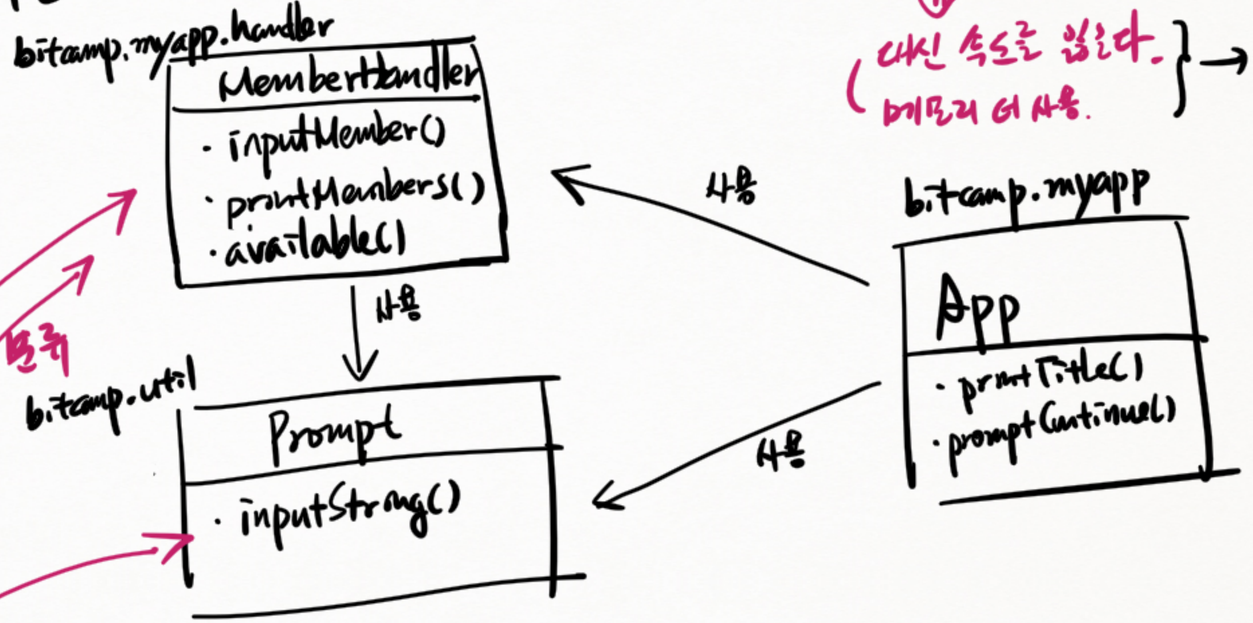
이전 구조 ⟹ 새 구조 : 메서드를 역할에 따라 분류 → 이유는 유지보수를 쉽게.
↳ 개발비용 절감

⇓

(대신 속도를 잃는다. 메모리 더 사용.) → H/W 학습량으로 처리

Architecture



# * 10. 메뉴 및 CRUD 구현

* 클래스
  ↳ 역할에 따라 메서드를 분류한 것

* 패키지
  ↳ 클래스를 분류한 것.

주기능을 분류한 것 →

**App**

- printMenu()
- printTitle()
- ~~promptContinue()~~
- main()

class 단위 → 메서드를 묶는 용도

**MemberHandler**

- inputMember()
- printMembers()
- available()
- viewMember()
- toGenderString()
- updateMember()
- inputGender()
- deleteMember()
- indexOf()

← 회원 데이터를 다루는 메서드를 묶어 놓은 것 분류한 것.

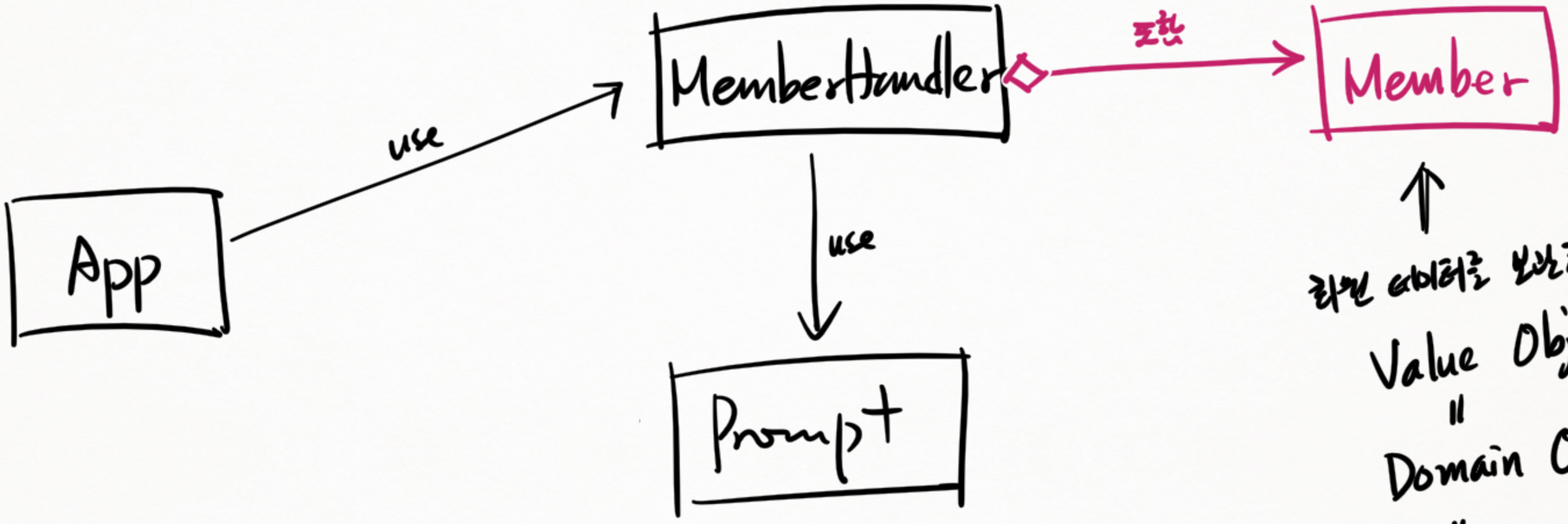
↓

**Prompt**

- inputString()
- close()
- inputInt()

← 사용자 입력을 다루는 메서드를 분류한 것

## 11. 사용자 정의 데이터 타입 만들기

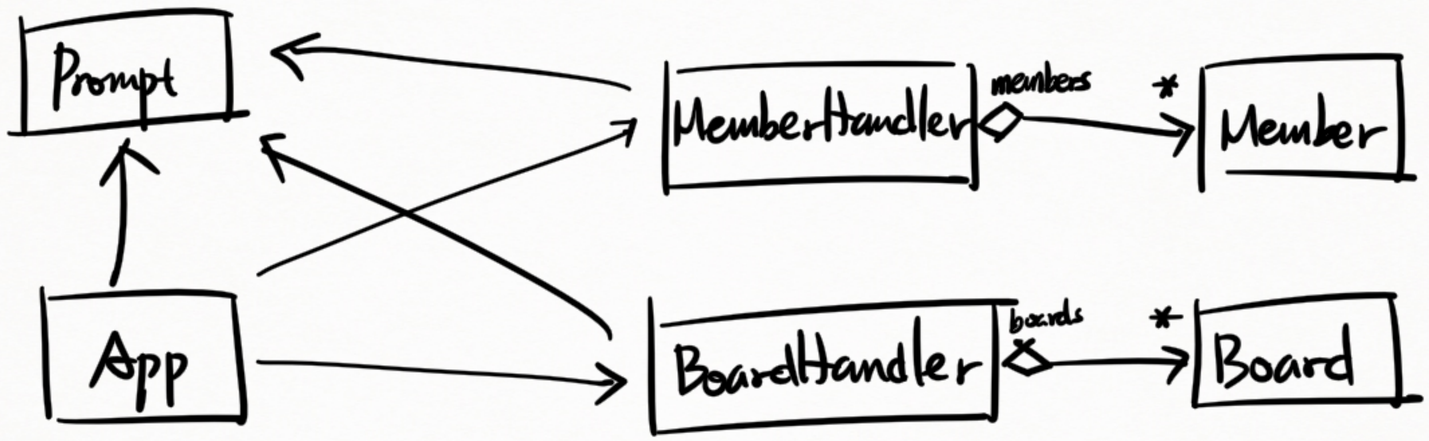



리턴 데이터를 보관하는 역할

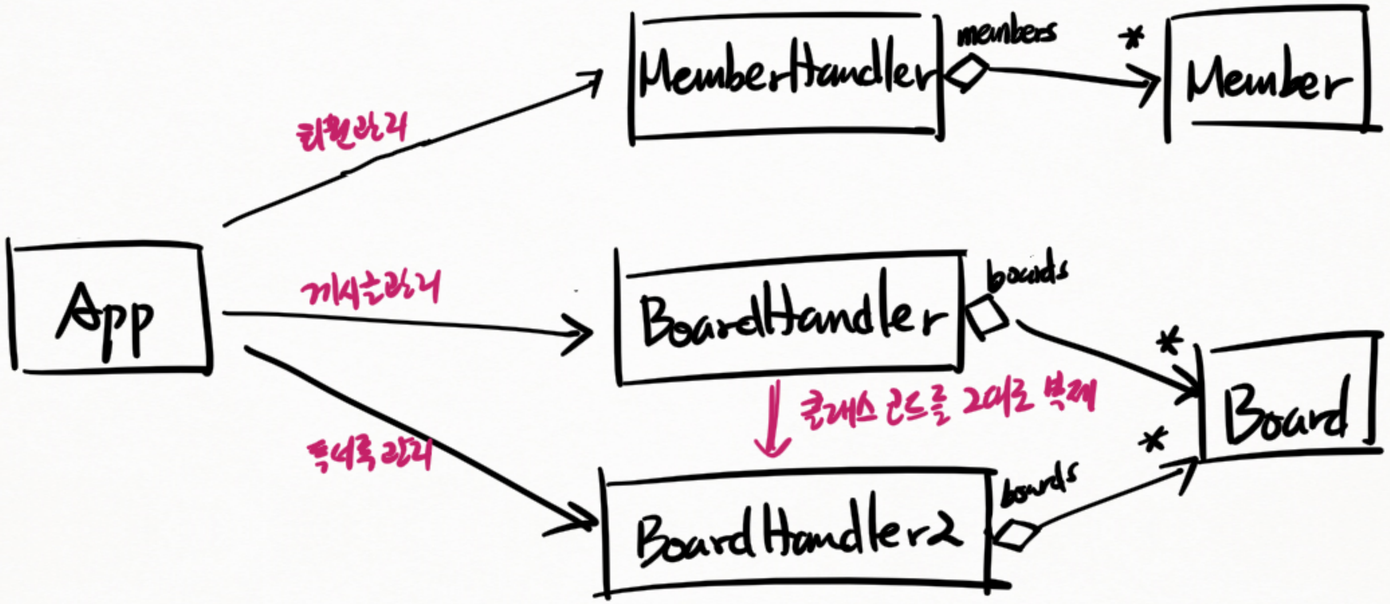
Value Object (VO)
=
Domain Object
=
Data Transfer Object
(DTO)

* class 등록 → 데이터 타입을 정의

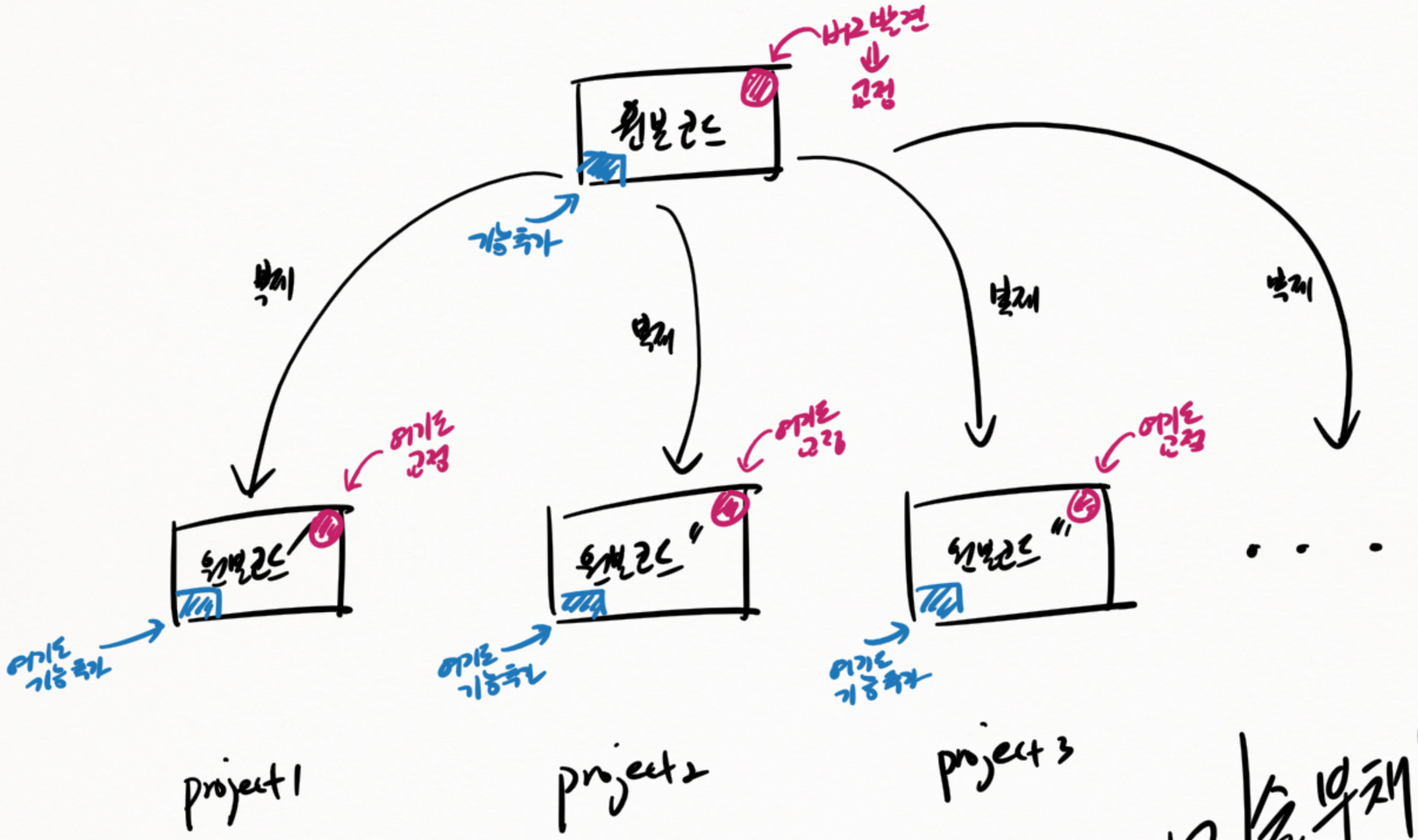
13. 게시판 CRUD 추가



## 14. 독서록 CRUD 추가



* 기존 코드 복제해서 새 기능을 구현할 때 문제점



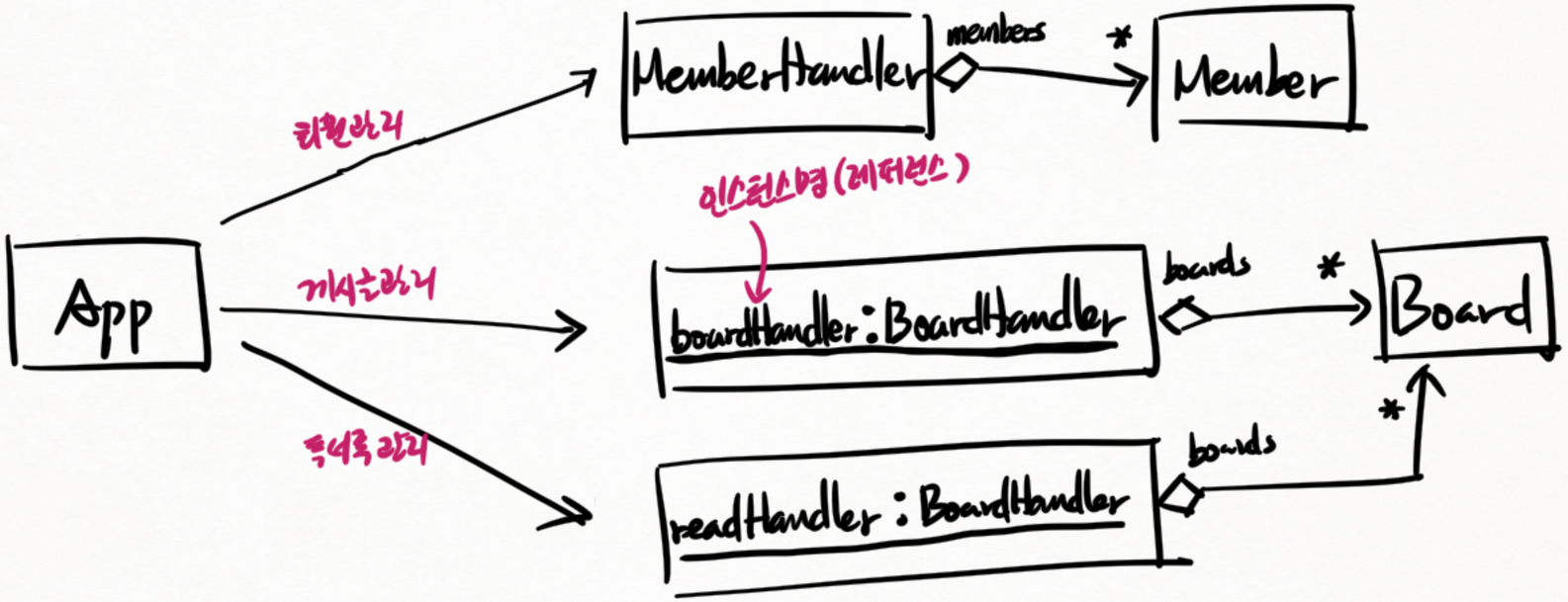

* 버그를 교정하거나 기능을 추가하게 되면 복제한 코드에 대해서도 똑같은 일을 수행해야 한다.

⇓

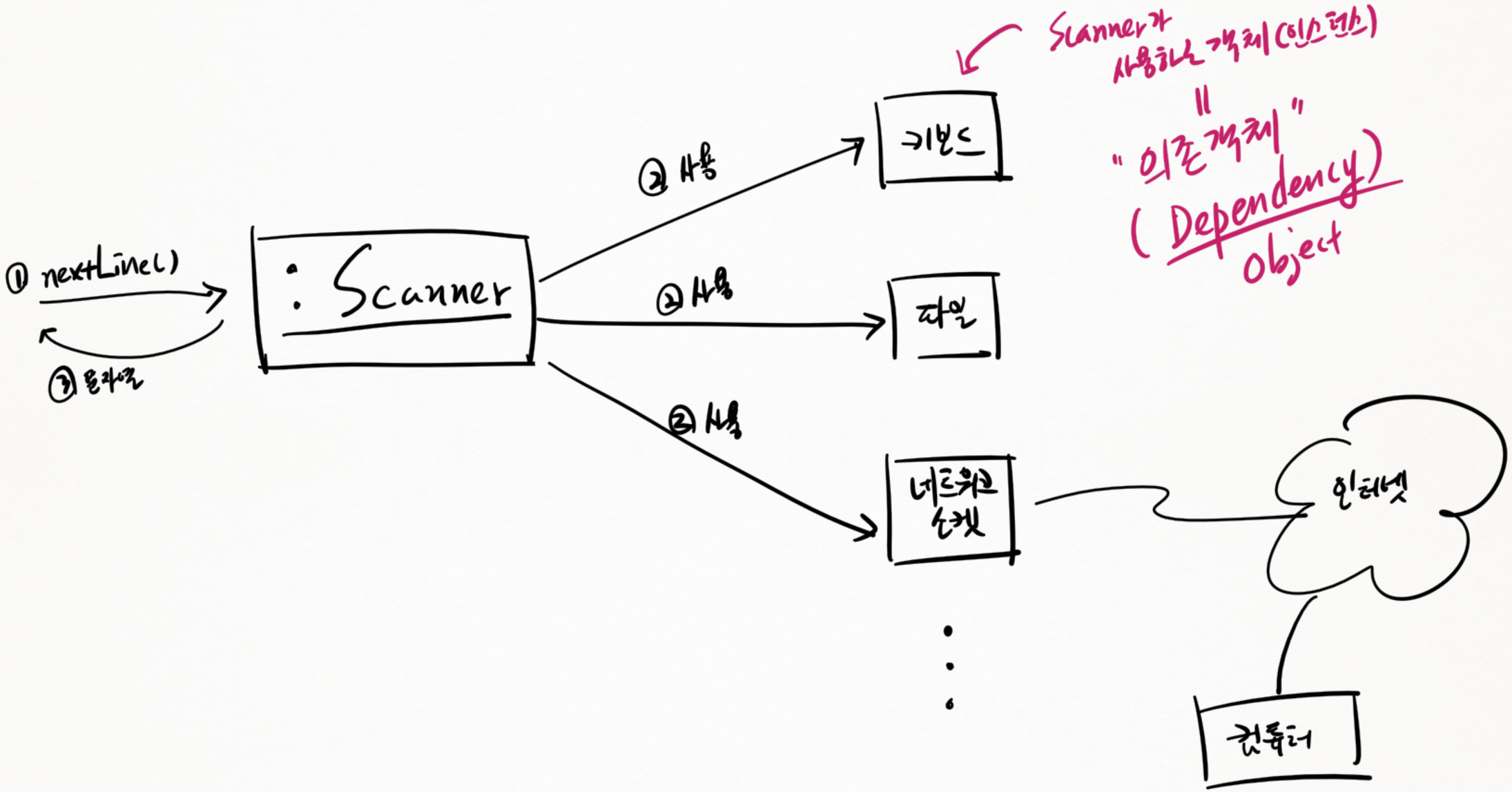
코드를 복제하면 할수록 유지보수가 점점 힘들어진다.

"기술부채"

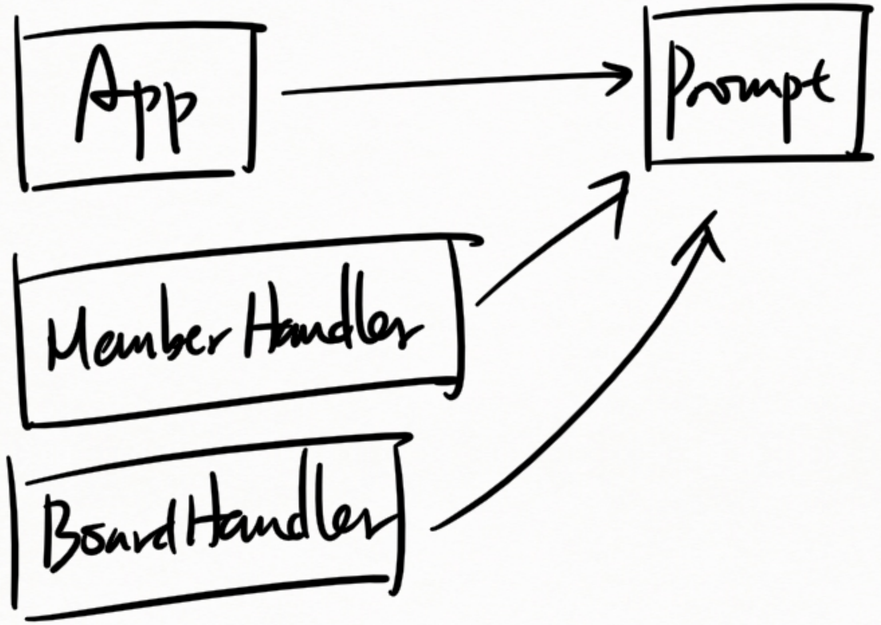
15. 인스턴스 필드 및 메서드 활용



Scanner와 의존객체



App, MemberHandler, BoardHandler 에 Prompt




의존객체

생성자를 통해 주입!

어떻게 콘스로 자동생성하는가?

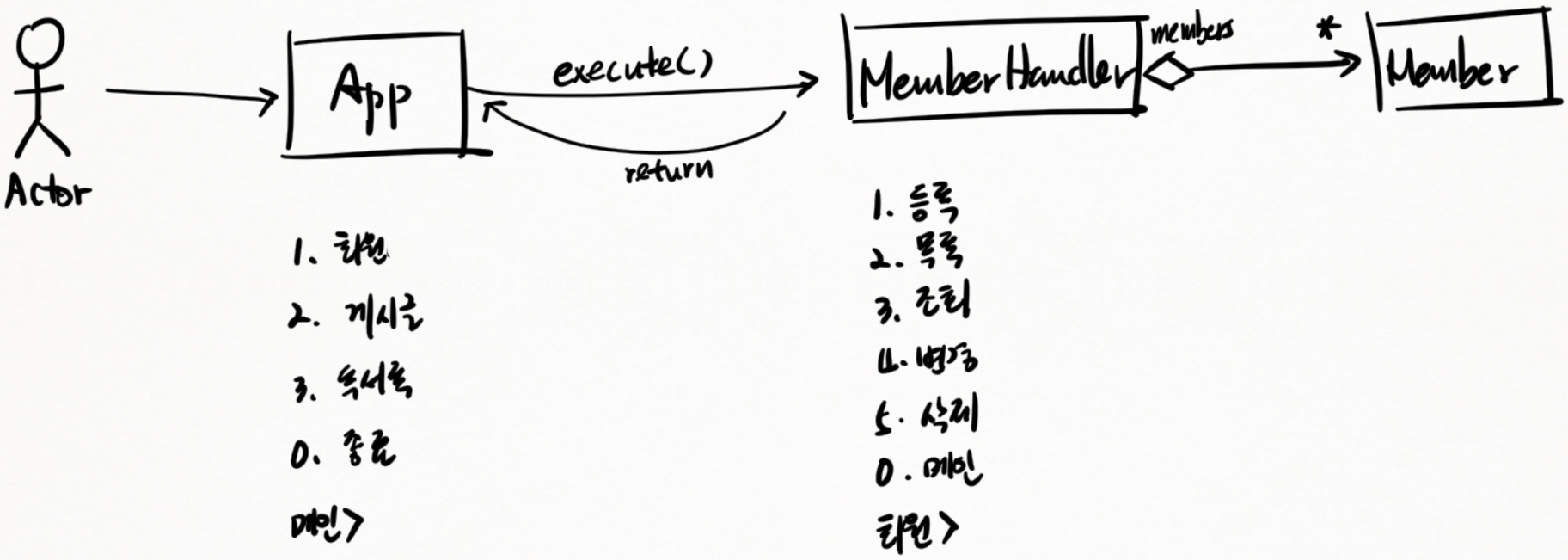
"의존객체 주입"
Dependency Injection
(DI)

* Prompt 가 인스턴스 방식으로 변경

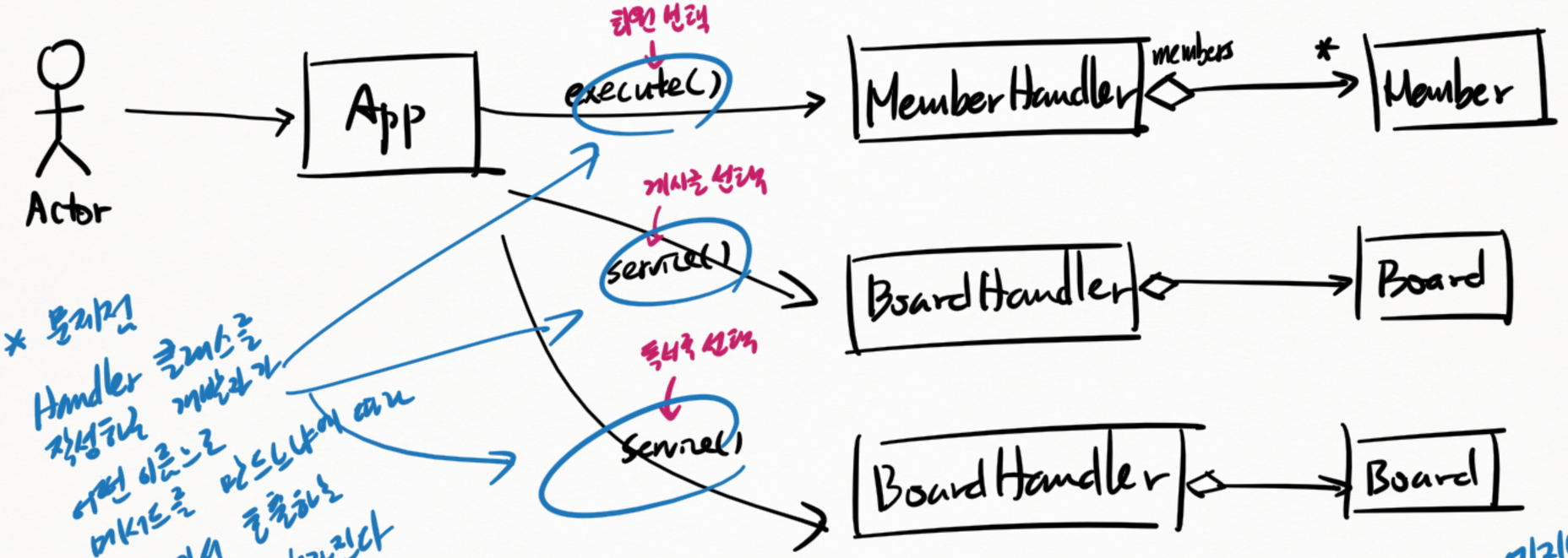
이 객체를 사용하는 클래스를 변경해야 한다

생성자를 통해 의존객체(Prompt)를 주입!
Injection

16. Handler에게 메뉴 기능을 위임



# 16. Handler에게 메뉴 기능을 위임




"인터페이스"
(Interface)

클래스 사용법을
특정 방식으로
강제할 수 있는
문법이 필요!

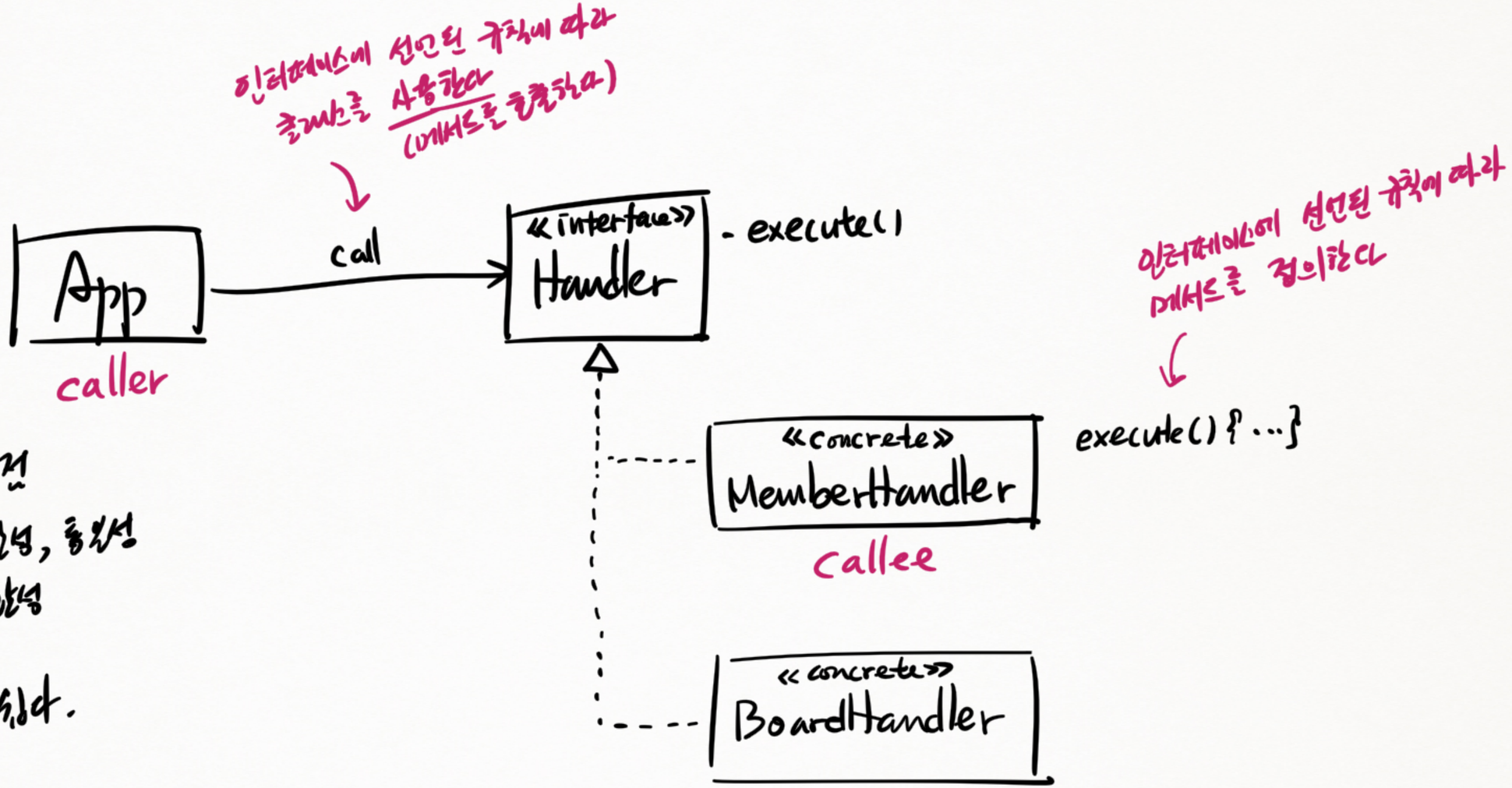
* 문제점
Handler 클래스를
작성하는 개발자가
어떤 이름으로
메서드를 만드느냐에 따라
App에서 호출하는
코드가 달라진다

⟹ 개발자가
같은 이름으로
메서드를 만들면 되는 것이
아닌가?

⟹ 강제성이 없어
개발자의
선의에 기댈 수 밖에 없다!

⟹ 인터페이스 세계에서
선의에 기대는 것은
의미 없다.

## 17. Handler 의 사용규칙을 인터페이스로 정하기



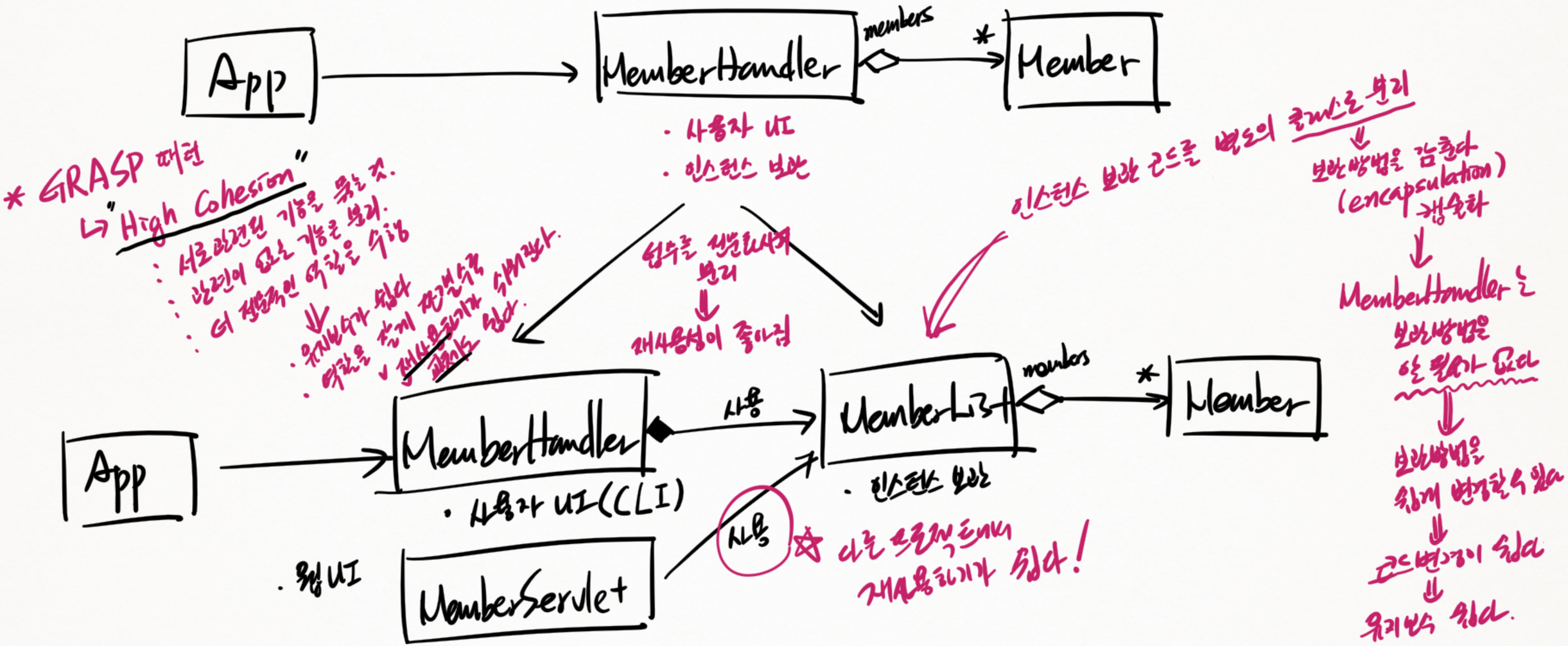

* 인터페이스 이점

- 구현의 일관성, 통일성
- 사용의 일관성

⇓

대체가 쉽다.

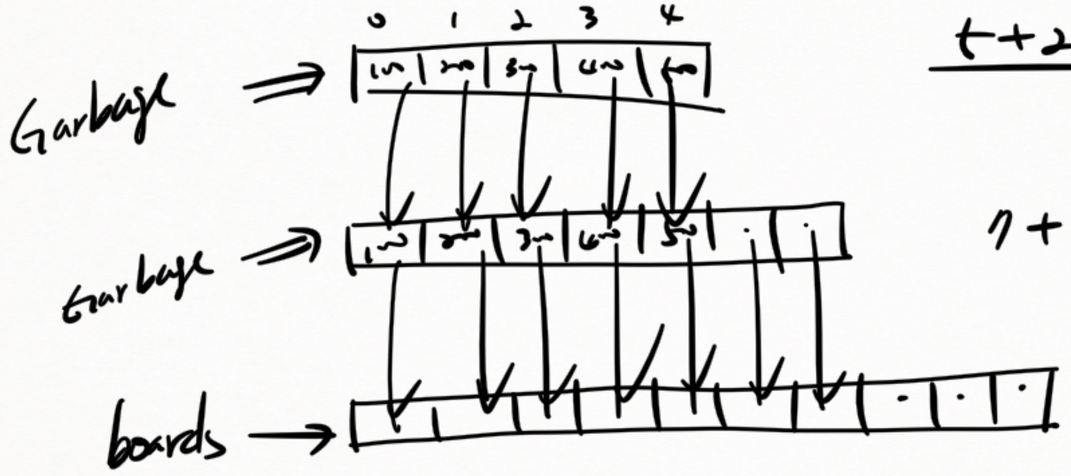
# 18. 인스턴스 목록을 다루는 코드를 별도의 클래스로 분리



* 배열 늘리기

$5/2 = 2.\cancel{5}$

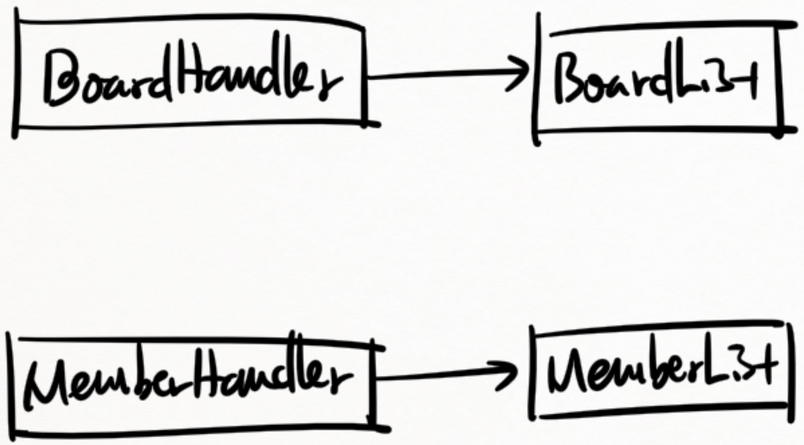
$5+2=7$



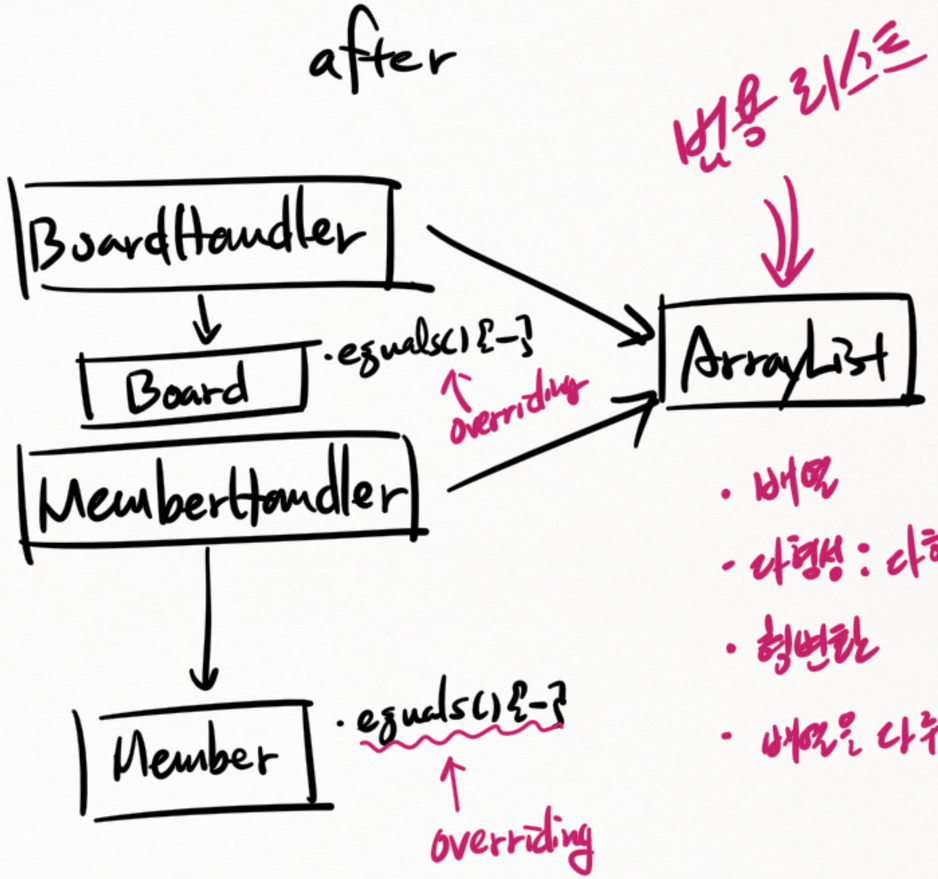

$7 + \frac{(7/2)}{3} = 10$

# 19. 범용 리스트 만들기

## before



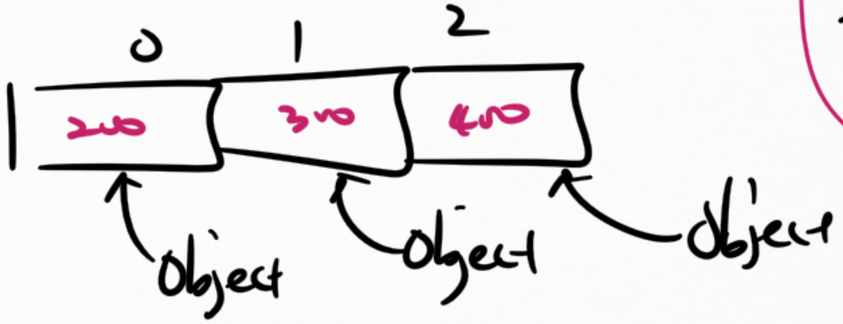

## after

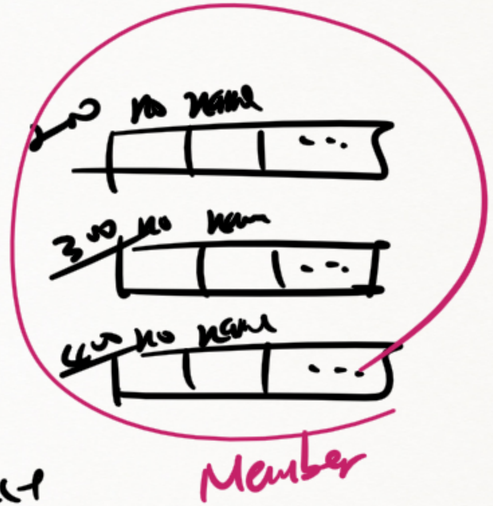



- 배열
- 다형성 : 다형적 변수
- 형변환
- 배열을 다루는 API 사용법

```
Object[] arr1 = new Object[3];

arr1[0] = new Member();
obj1[1] = new Member();
obj1[2] = new Member();
```

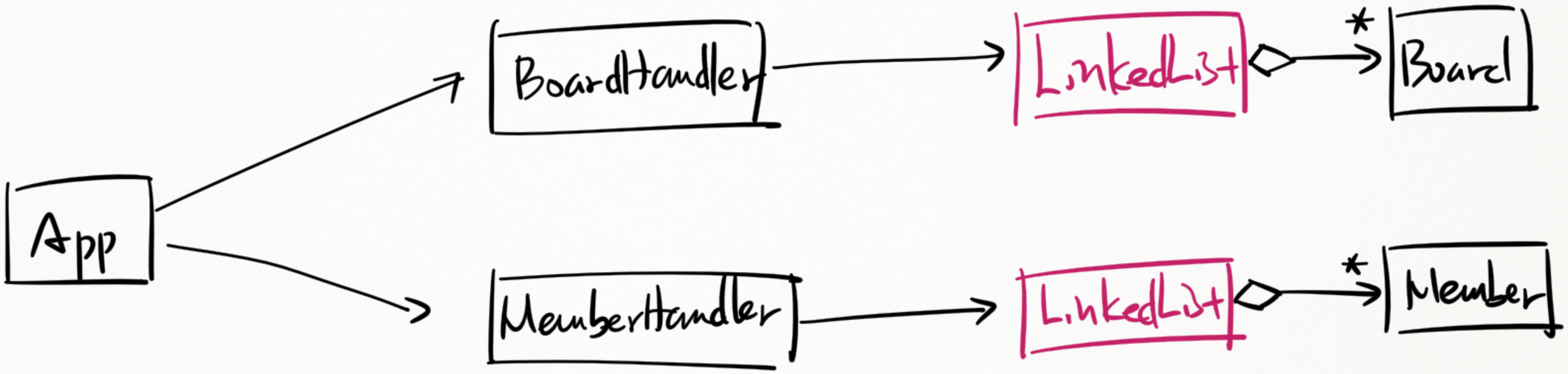



```
Member[] arr2 = (Member[]) arr1;
```

arr1 이 가리키는 것은 Member 레퍼런스 배열이 아니라 Object 레퍼런스 배열이다. 따라서 형변환 할수 없다.!

* 20. LinkedList 자료구조 구현하고 적용하기

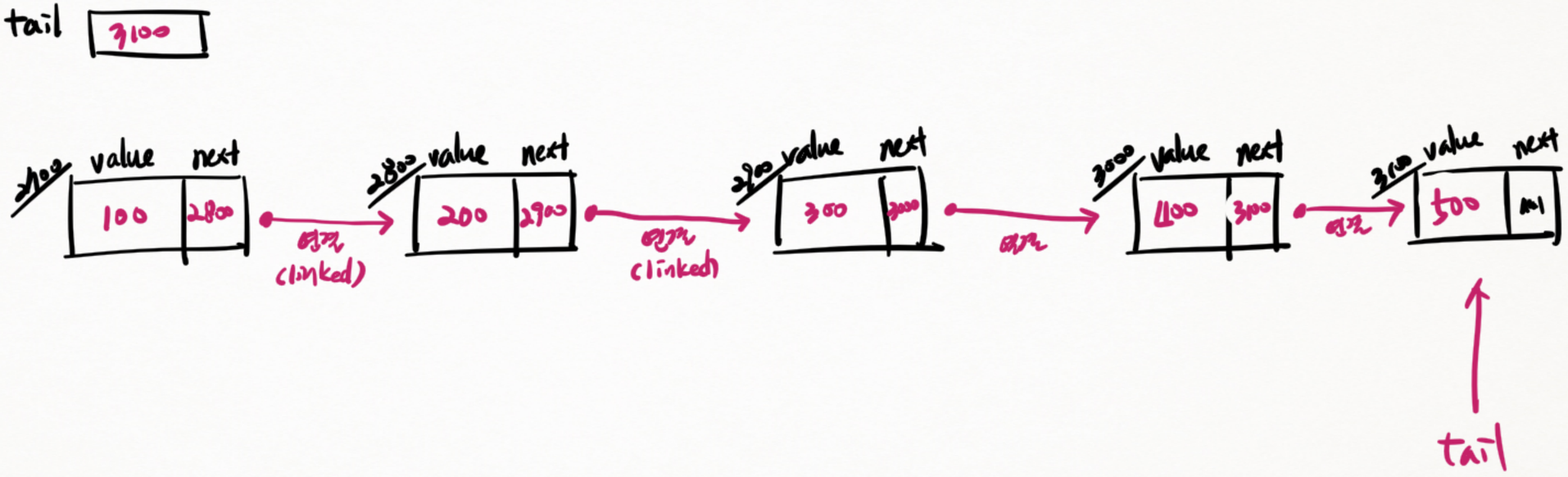


# ArrayList vs LinkedList

| | ArrayList | LinkedList |
|---|---|---|
| 초기크기 | 일정 메모리 확보 (10개) | 0 부터 시작 |
| 크기증가 | Yes → 기존배열이 가비지가 된다 | Yes → 가비지 없다. |
| 조회 | 인덱스로 조회 (LinkedList 보다 빠르다) | 링크를 따라가야 한다 (ArrayList 보다 느리다) |
| 삽입, 삭제 | 삽입 → 배열값 밀기<br>삭제 → 배열값 당기기<br>(LinkedList 보다 느리다) | 해당 값을 찾은 후 노드 추가/삭제 (list 보다 빠르다) |

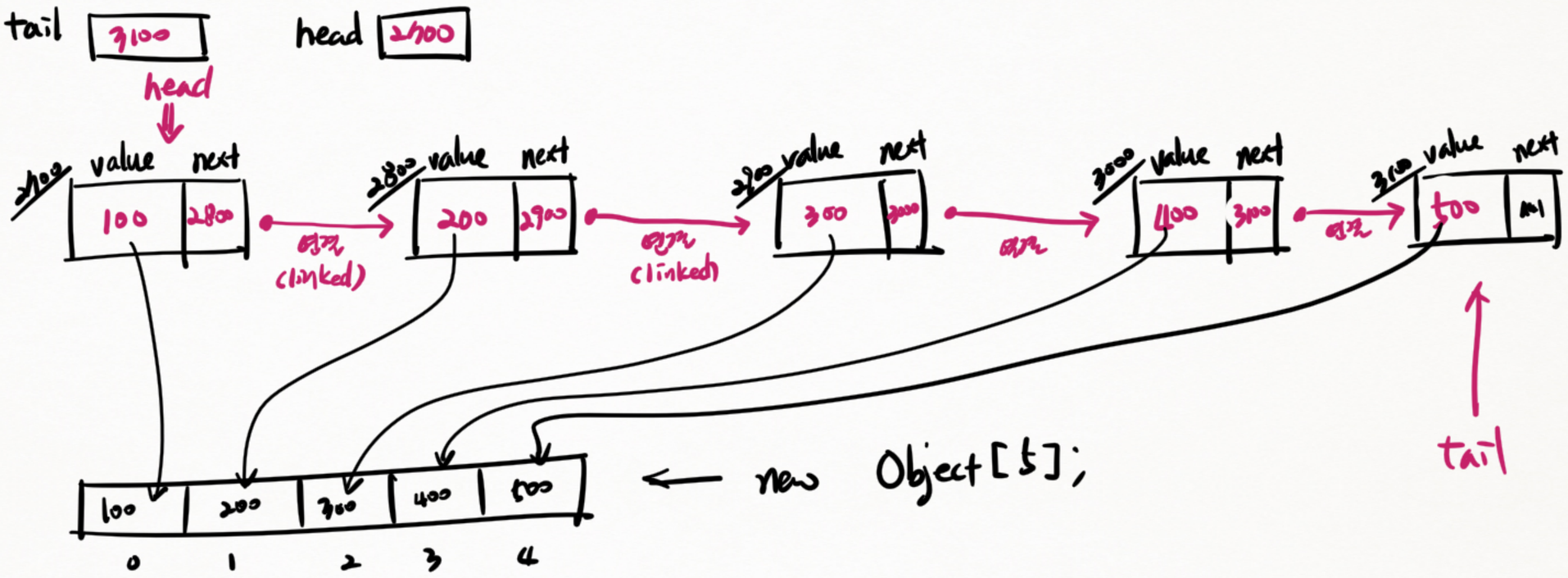
# LinkedList - add()

```
add(100);
add(200);
add(300);
add(400);
add(500);
```



# LinkedList — getList()

getList();




node

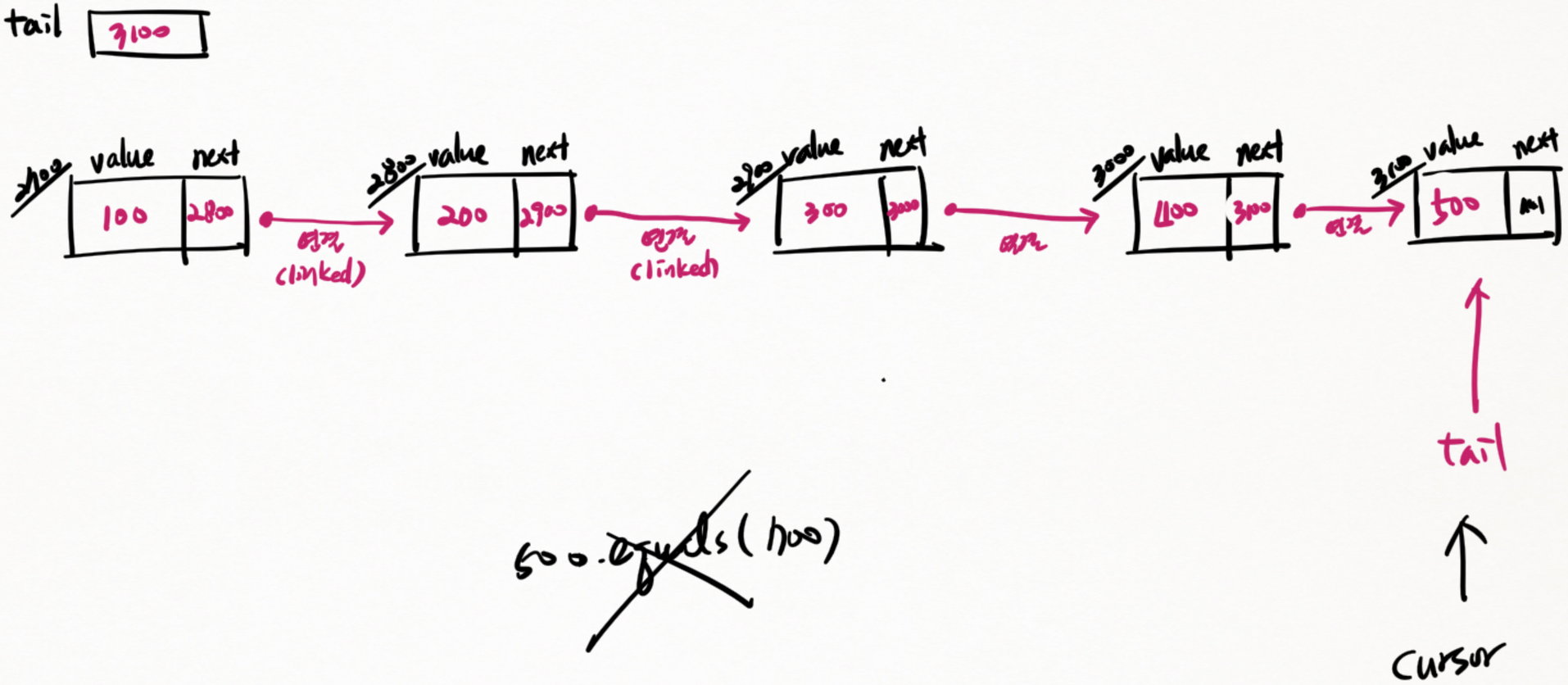
2800

node = node.next;

2800

# LinkedList — retrieve()

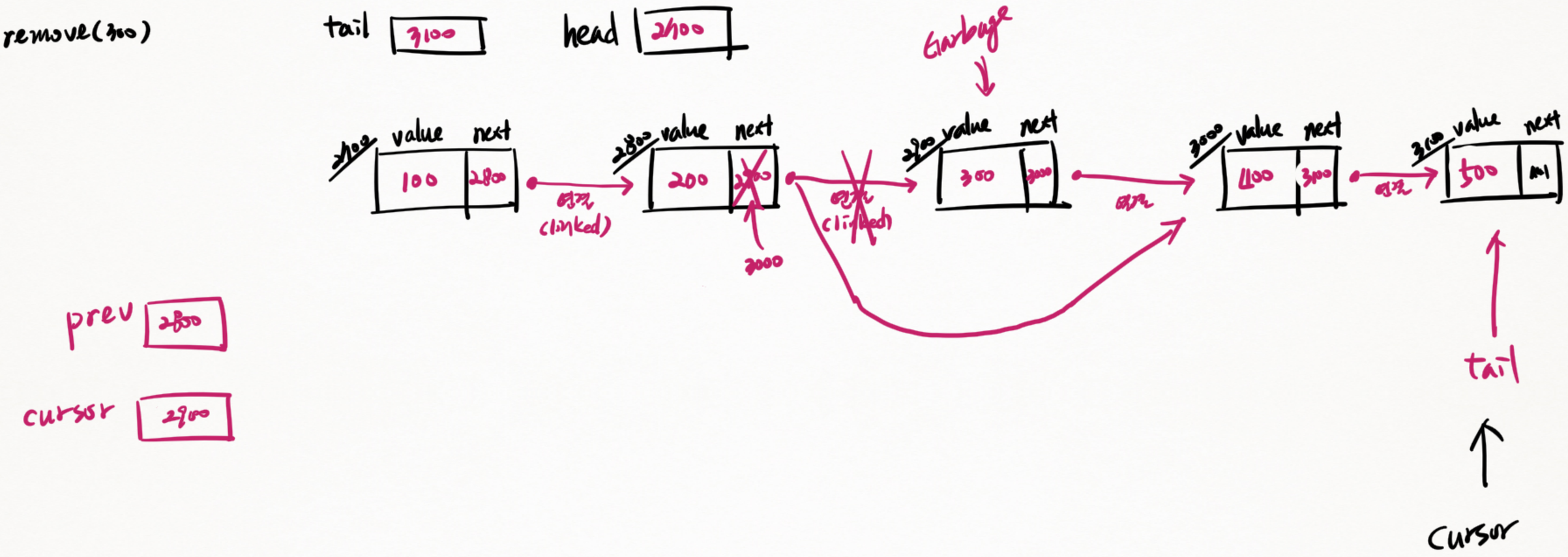
retrieve(400);




500.equals(400)

# LinkedList — remove() : 중간 항목 삭제

remove(300)

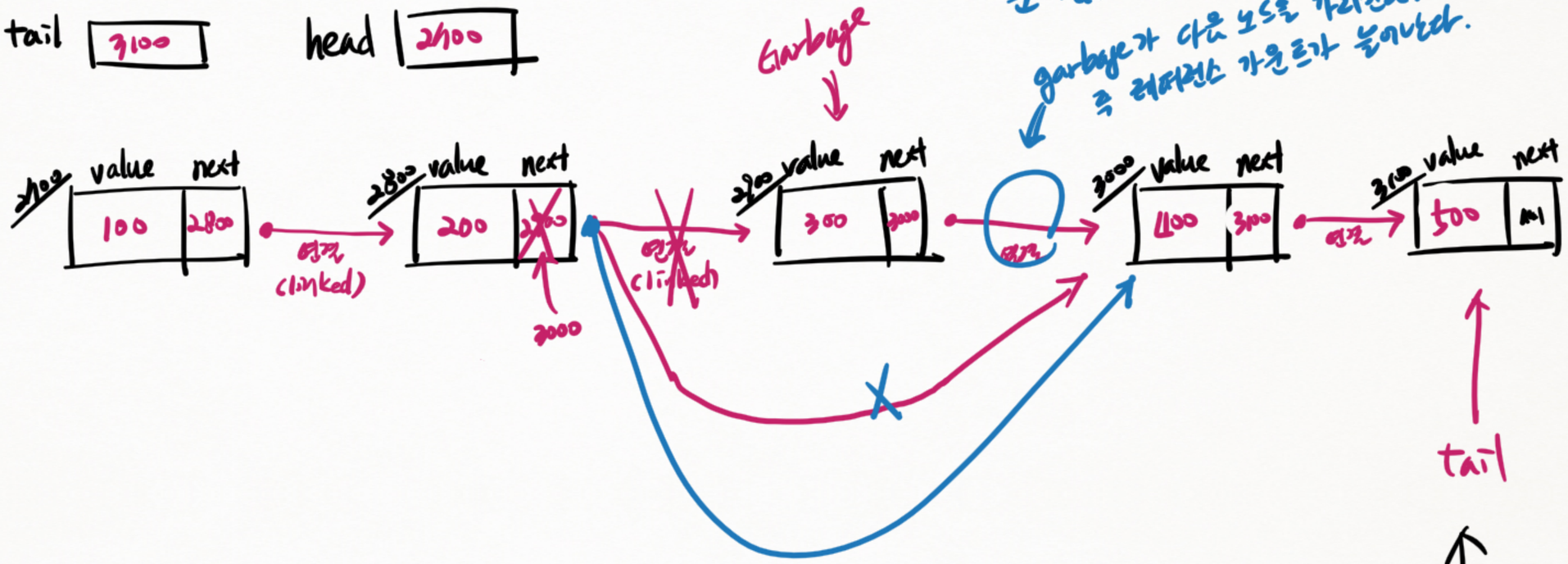


# LinkedList — remove() : 중간 항목 삭제 ⟹ 가비지가 인스턴스를 가리킬때 문제점.

remove(300)

tail [3100]  head [2100]

garbage가 다음 노드를 가리킨다.
즉 레퍼런스 카운트가 남아있다.

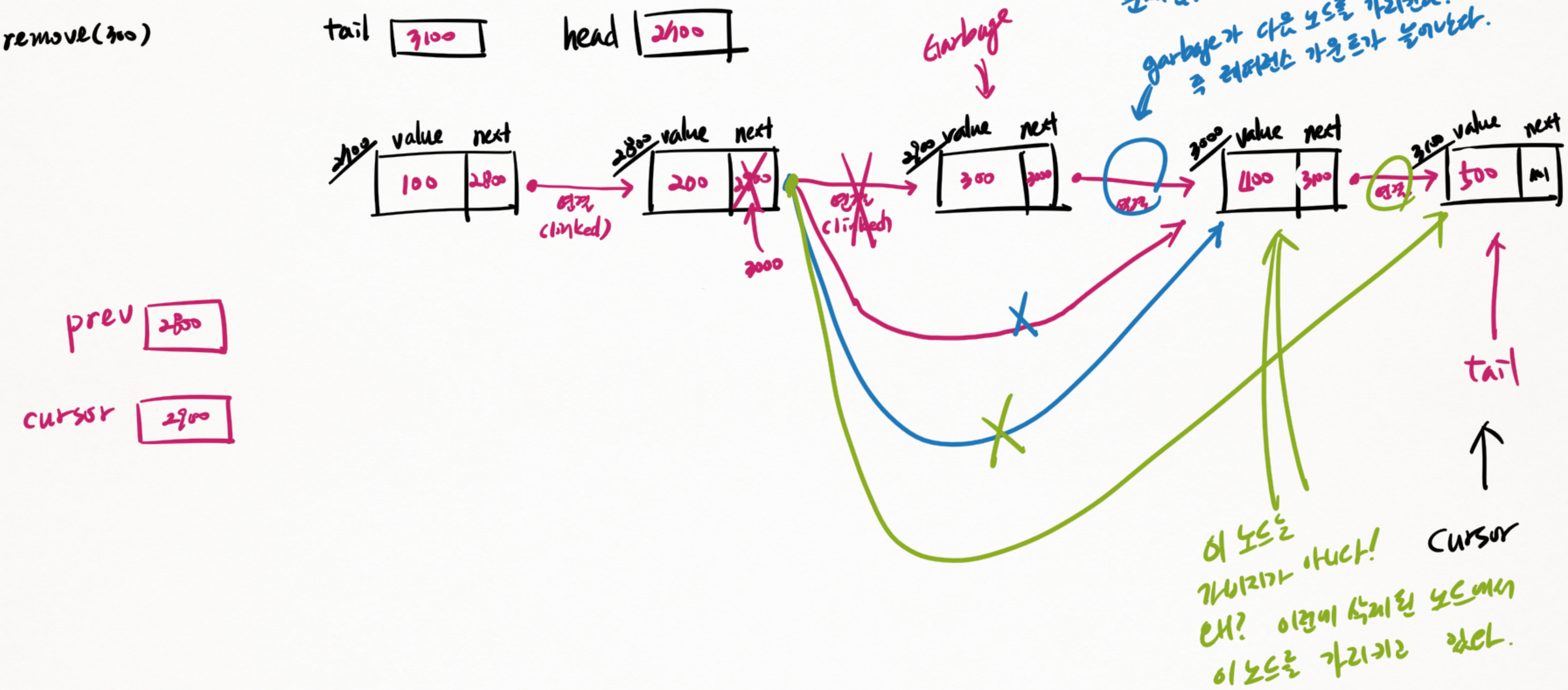



prev [2800]

cursor [2900]

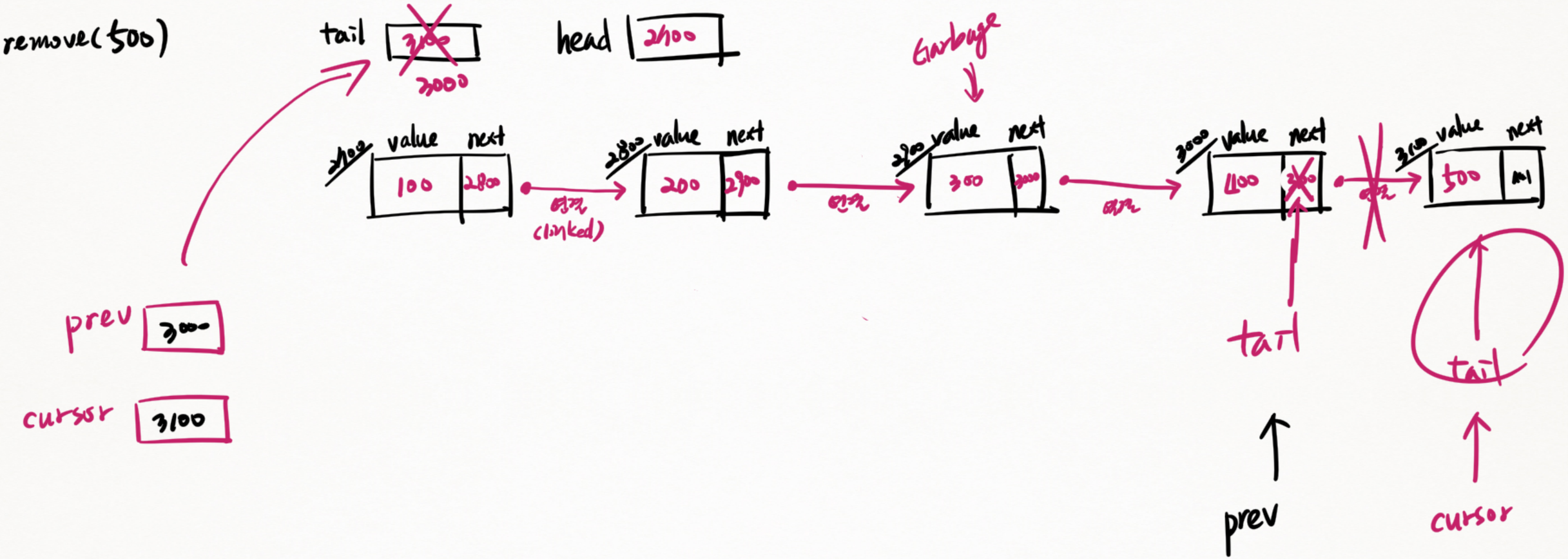
# LinkedList — remove() : 중간 항목 삭제 ⟹ 가비지가 인스턴스를 가리킬때 문제점.

remove(300)



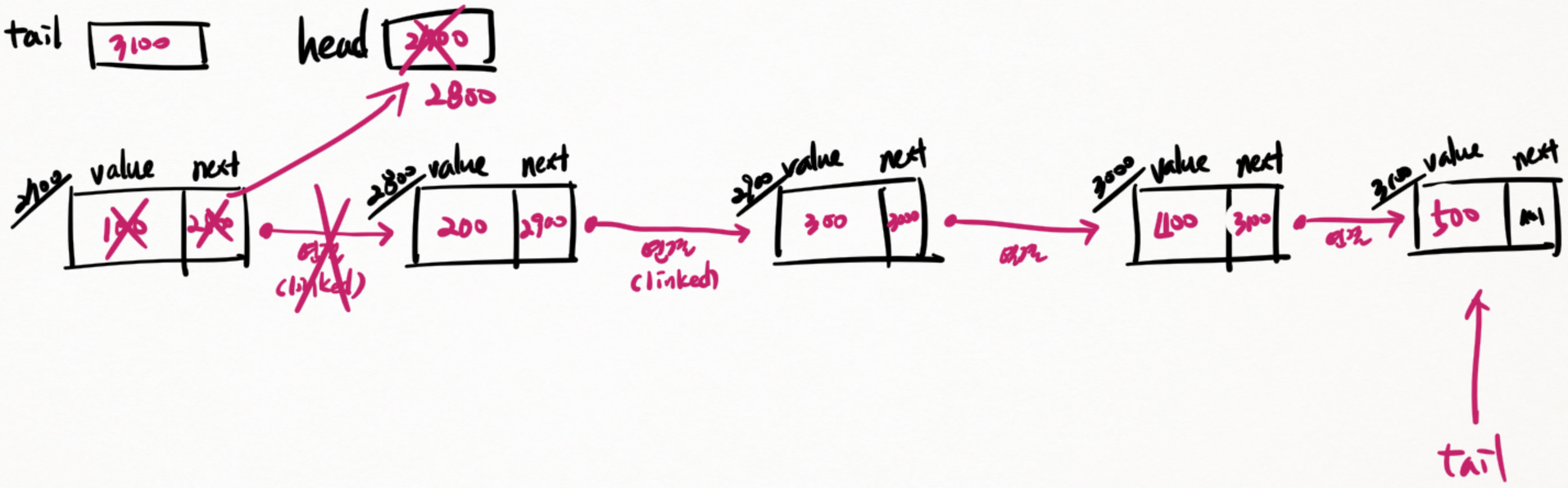
# LinkedList — remove() : 끝 항목 삭제

remove(500)



# LinkedList - remove() : 시작 노드

remove(100)




prev

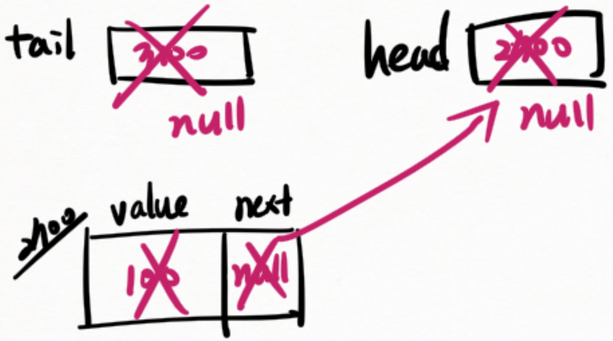
null

cursor

2700

LinkedList - remove() : 시작노드 + 끝노드

remove(100)



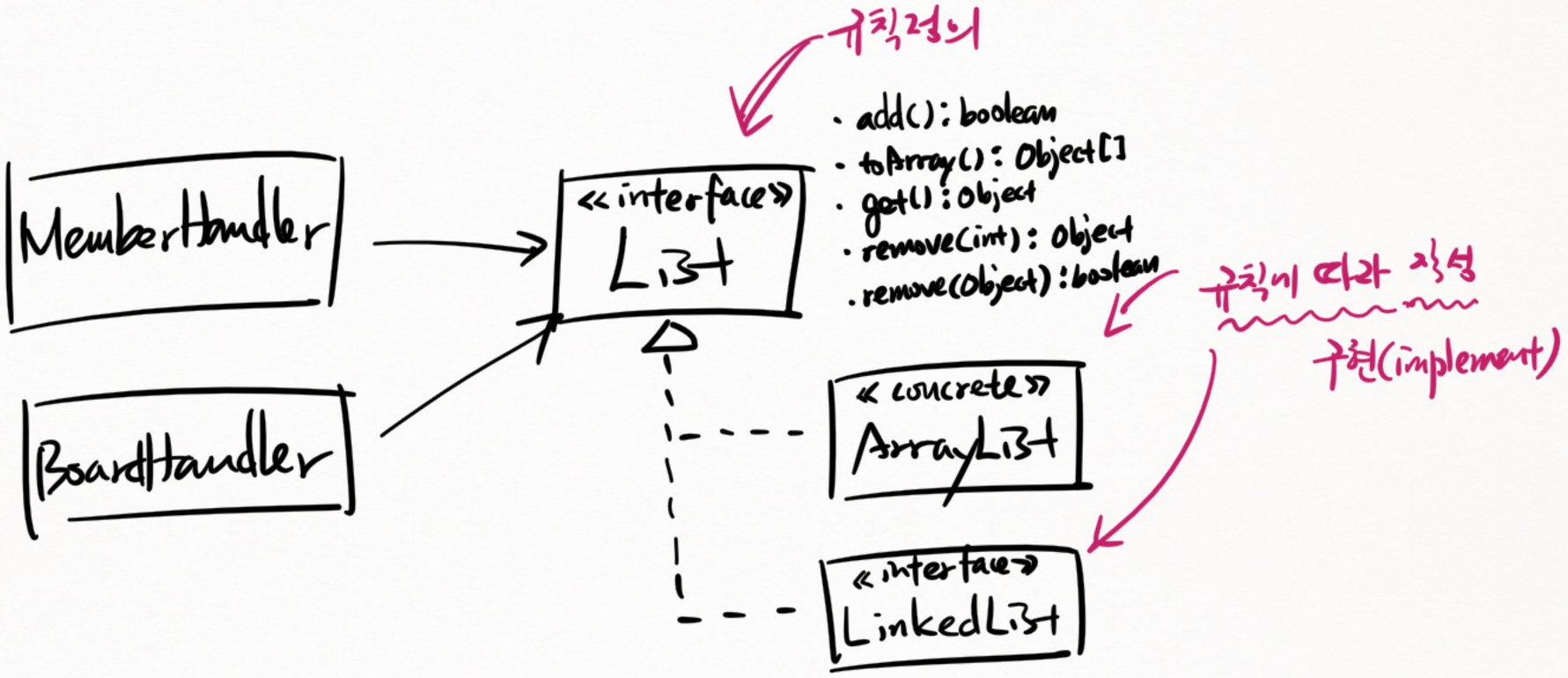

prev

null

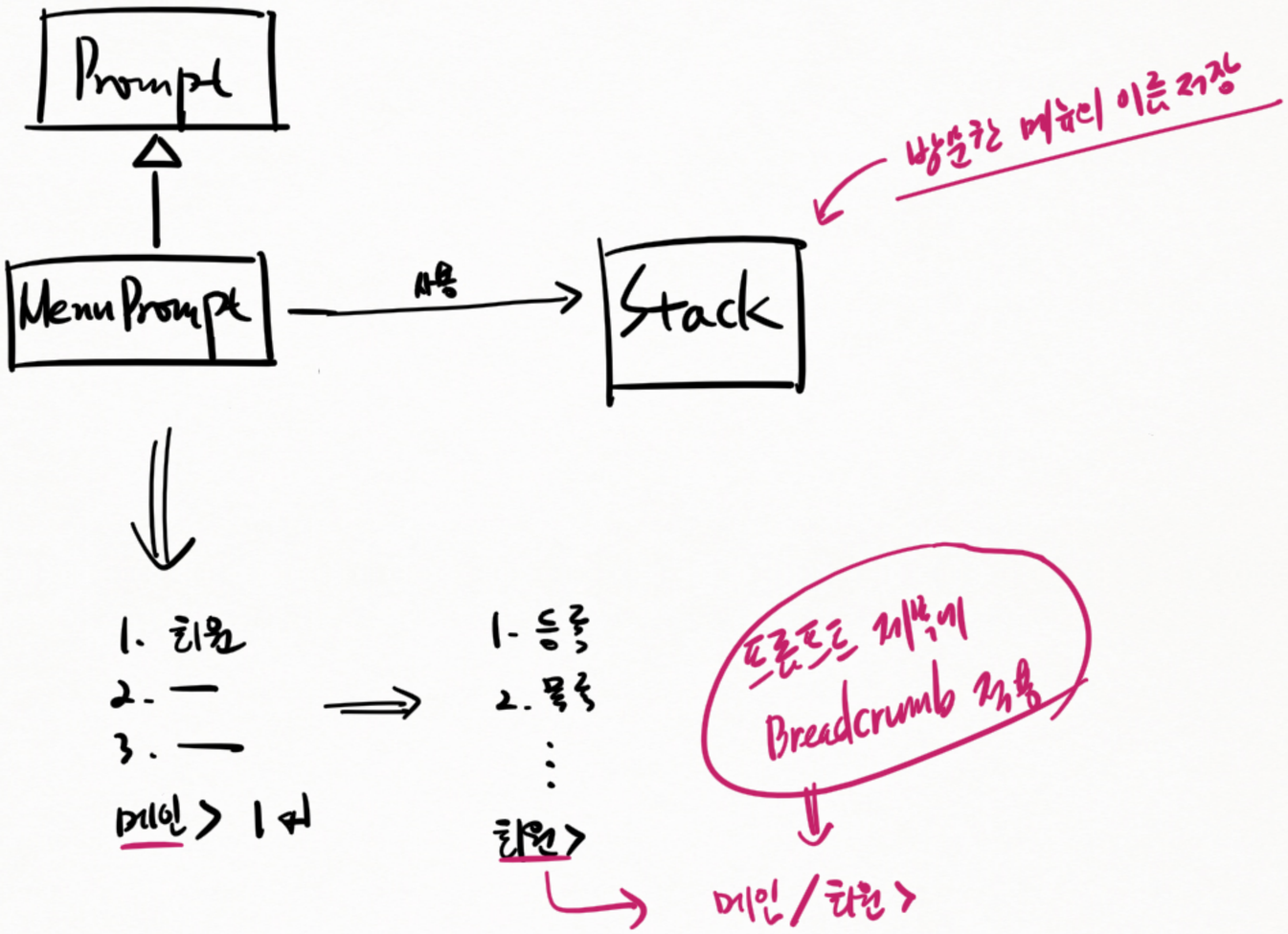
cursor

2100

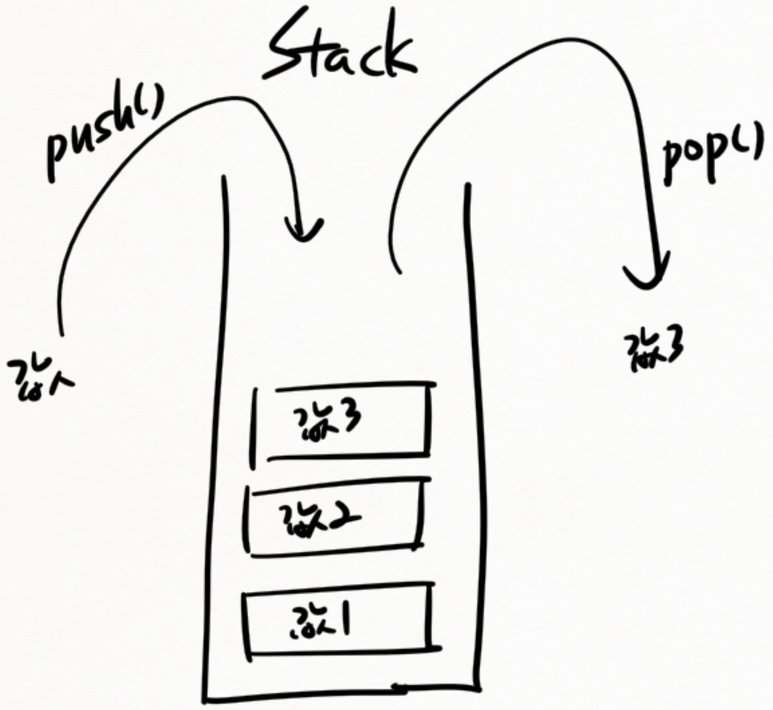
21. 인터페이스로 목록관리 객체의 사용법을 규칙으로 정의.



## 22. Stack, Queue 자료구조 구현



# * Stack 구동원리




Last In First Out
(LIFO)

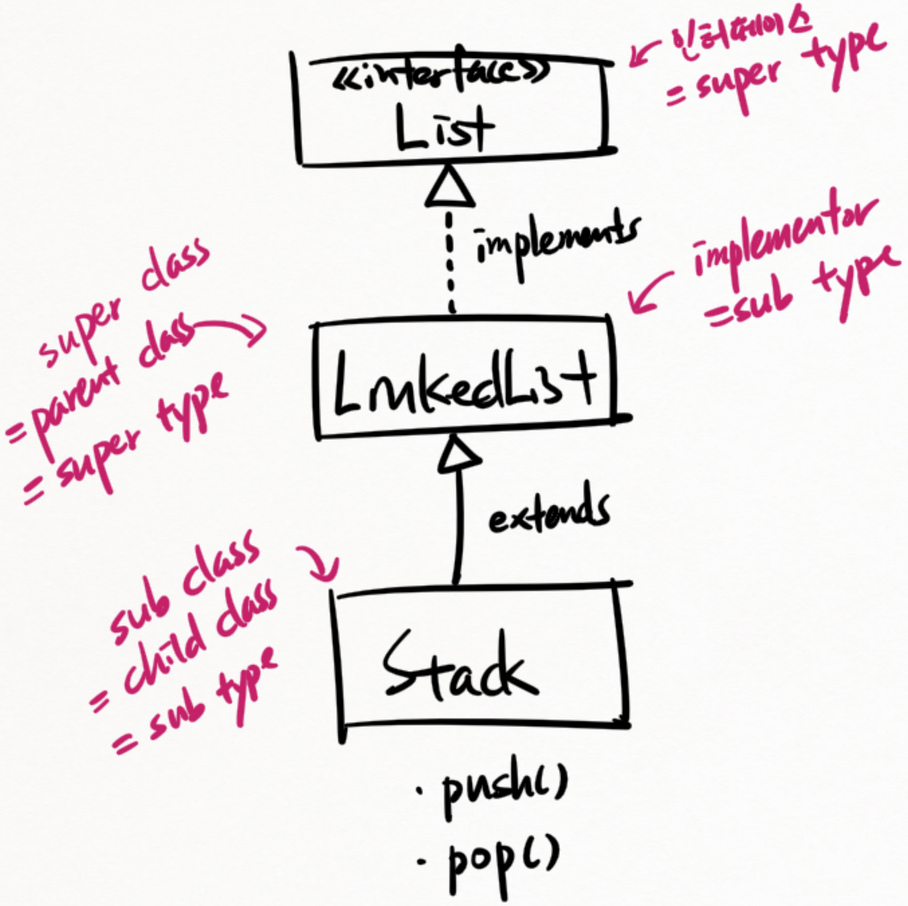
breadcrumb 구현에 적용할 수 있다

예) c:/ > 사용자 > bitcamp > git > bitcamp-study

뒤로 뒤로 뒤로 뒤로

- 방문한 메뉴이름 저장
- 〃 폴더이름 저장
  왕테이블 저장

# * Stack 구현




```
List obj;
obj = new ArrayList();
obj = new LinkedList();
obj = new Stack();
```

직접적으로 List 인터페이스 구현

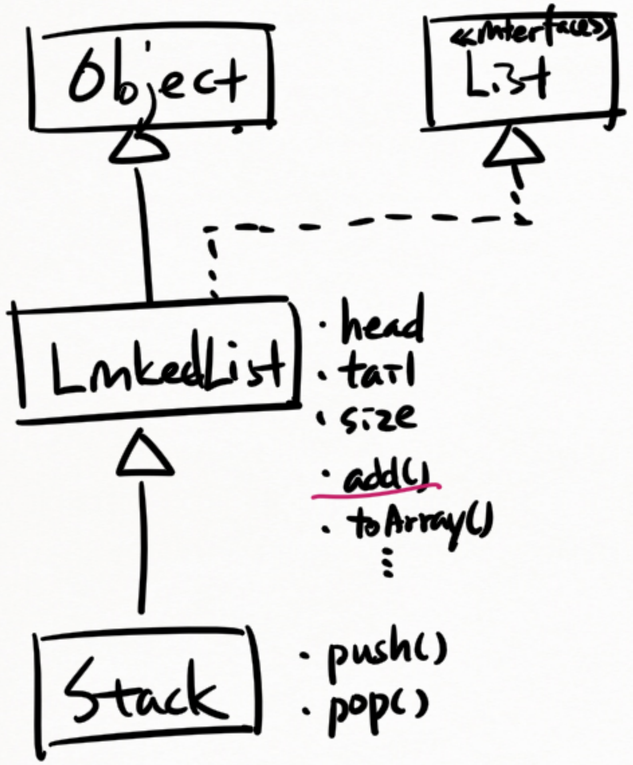
List 인터페이스를 간접적으로 구현.

sub 클래스는 super 클래스가 구현한 메서드를 사용할 수 있다

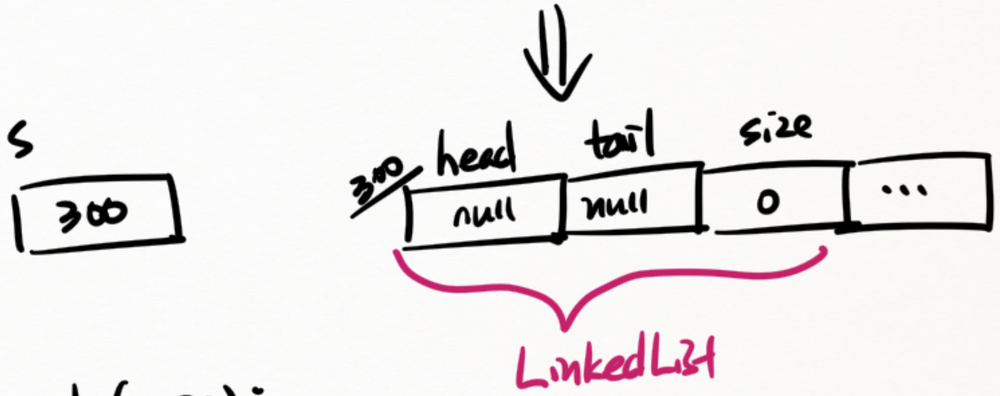
||

부터클래스가 인터페이스를 구현했으면 서브클래스는 구현한 것으로 간주한다.

# * Stack 인스턴스




```
Stack s = new Stack();
```




```
s.push(100);
300
  ↳ this.add(100);
    300
      ↳ this.head
        this.tail
        this.size
        300
```

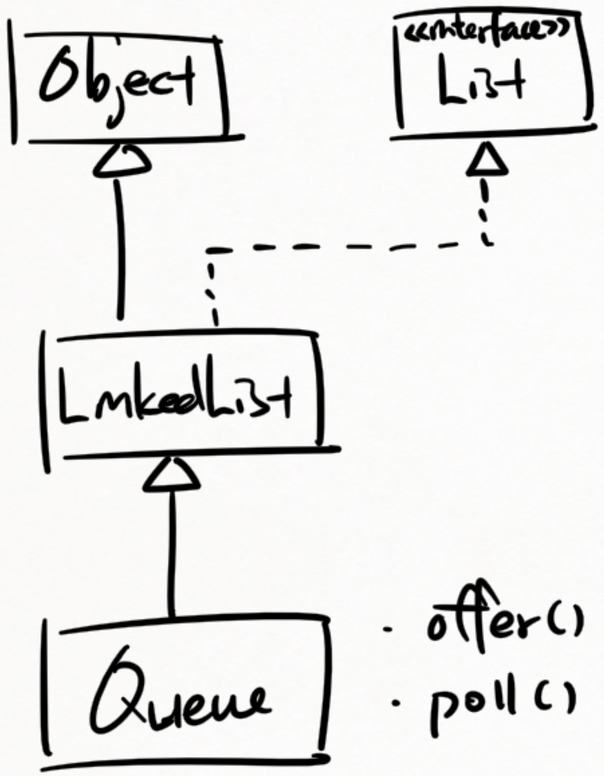
* Queue 구동 원리

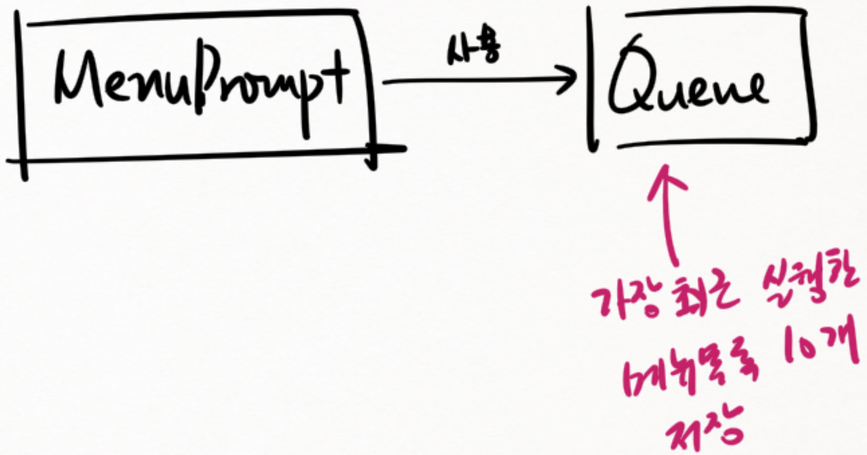
First In First Out
(FIFO)

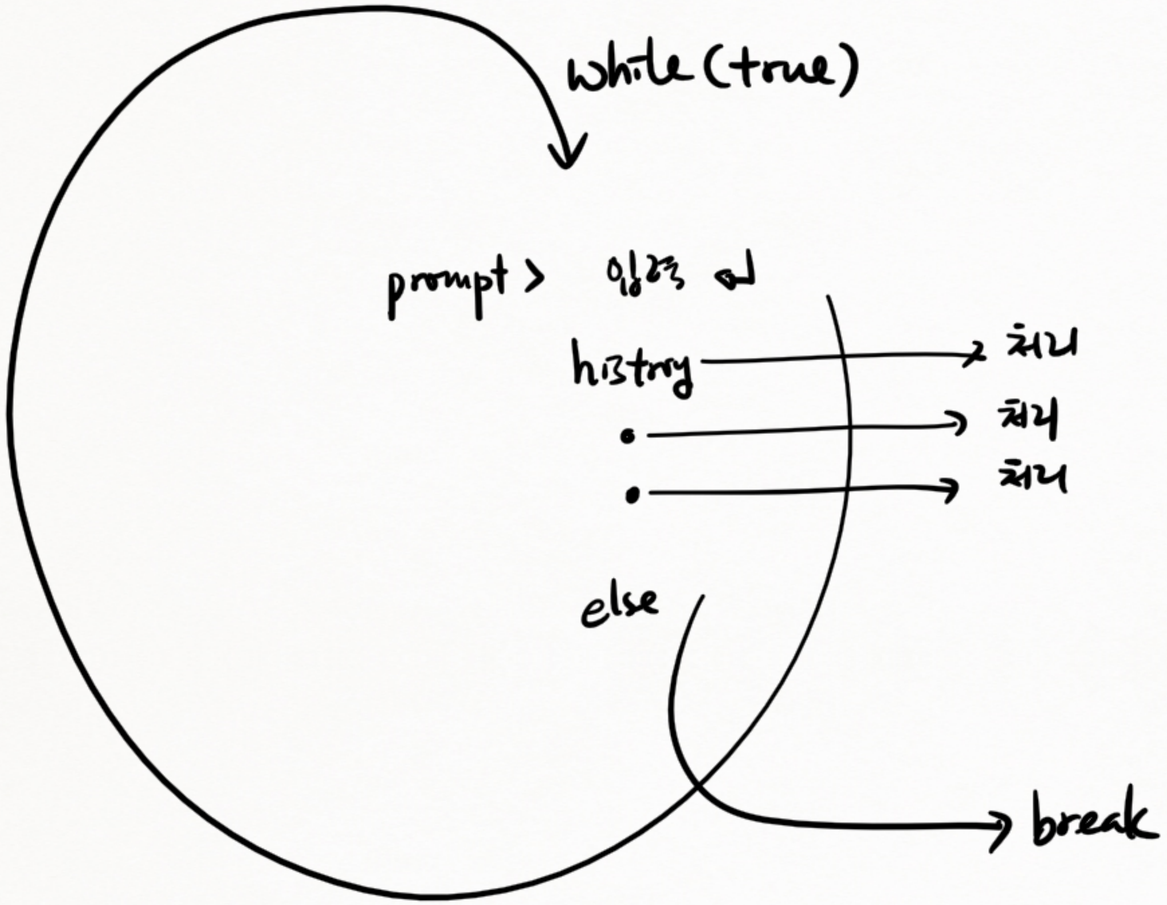
⟹ · 명령어 입력 history 기능에 적용

· (예약 주문) 받을 때

# * Queue 구현과 적용


Object
«interface»
List
LinkedList
Queue
· offer()
· poll()

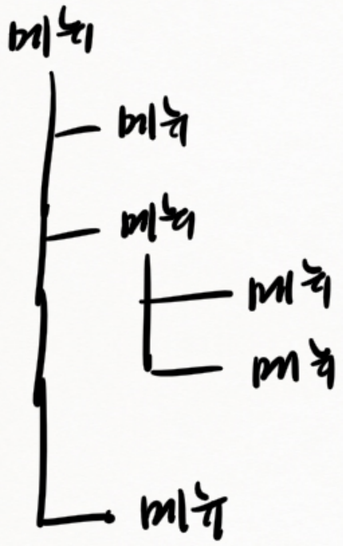

MenuPrompt
사용
Queue
가장 최근 실행한
메뉴목록 10개
저장

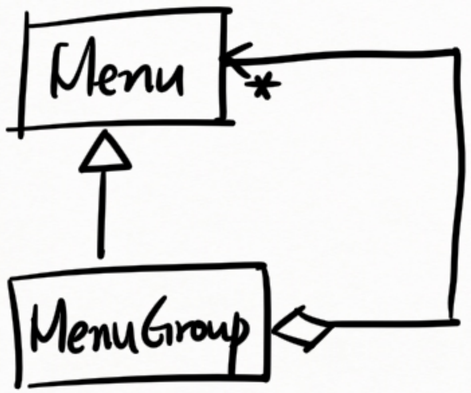

while (true)
prompt > 입력 ↵
history
처리
처리
처리
else
break

## 23. Composite, Command, Observer 패턴 활용

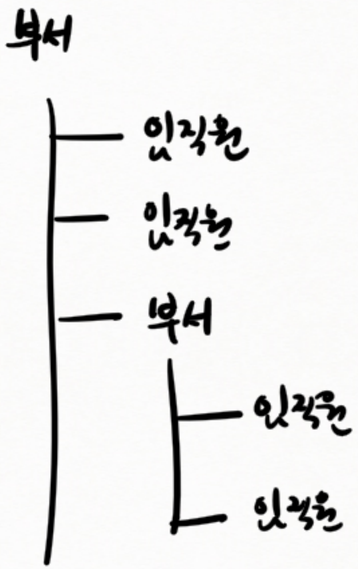
### ① Composite 패턴

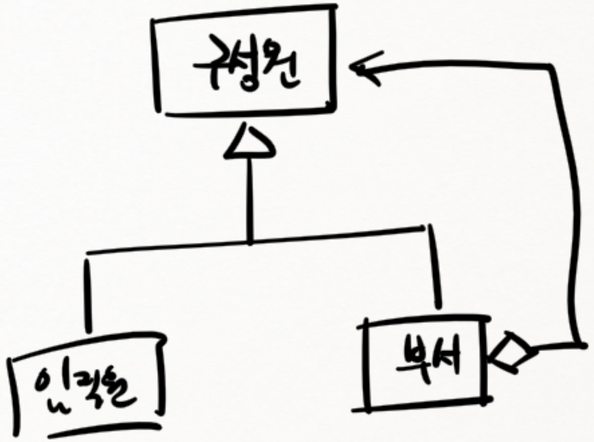
활용 예)



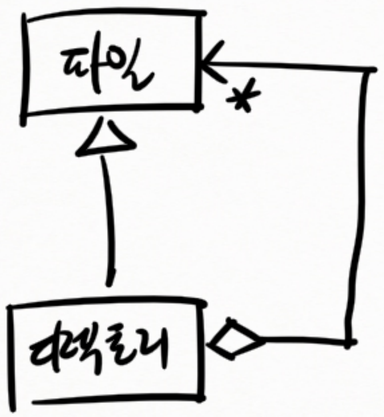




활용 예)

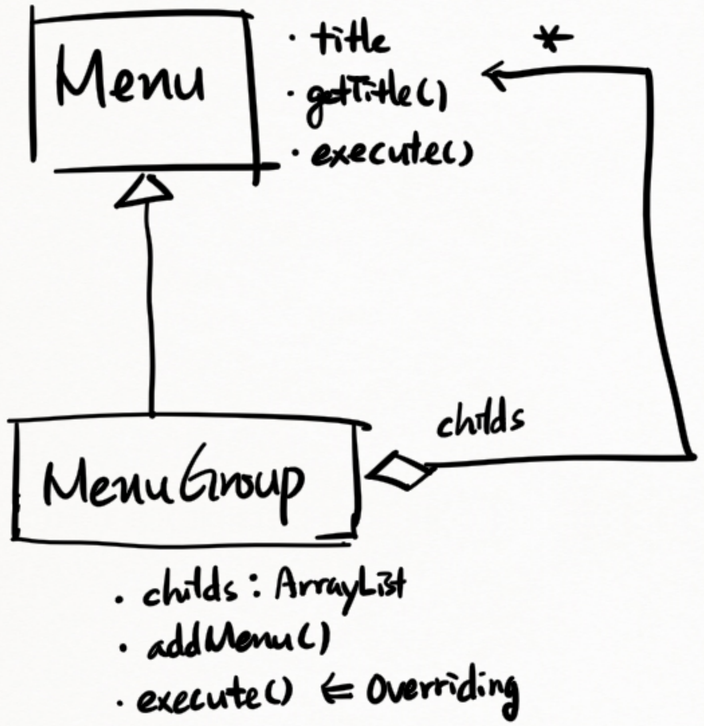


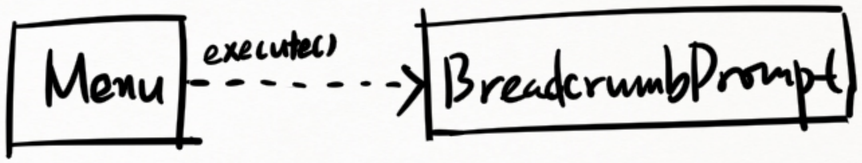



활용예)



# Composite 패턴 구현

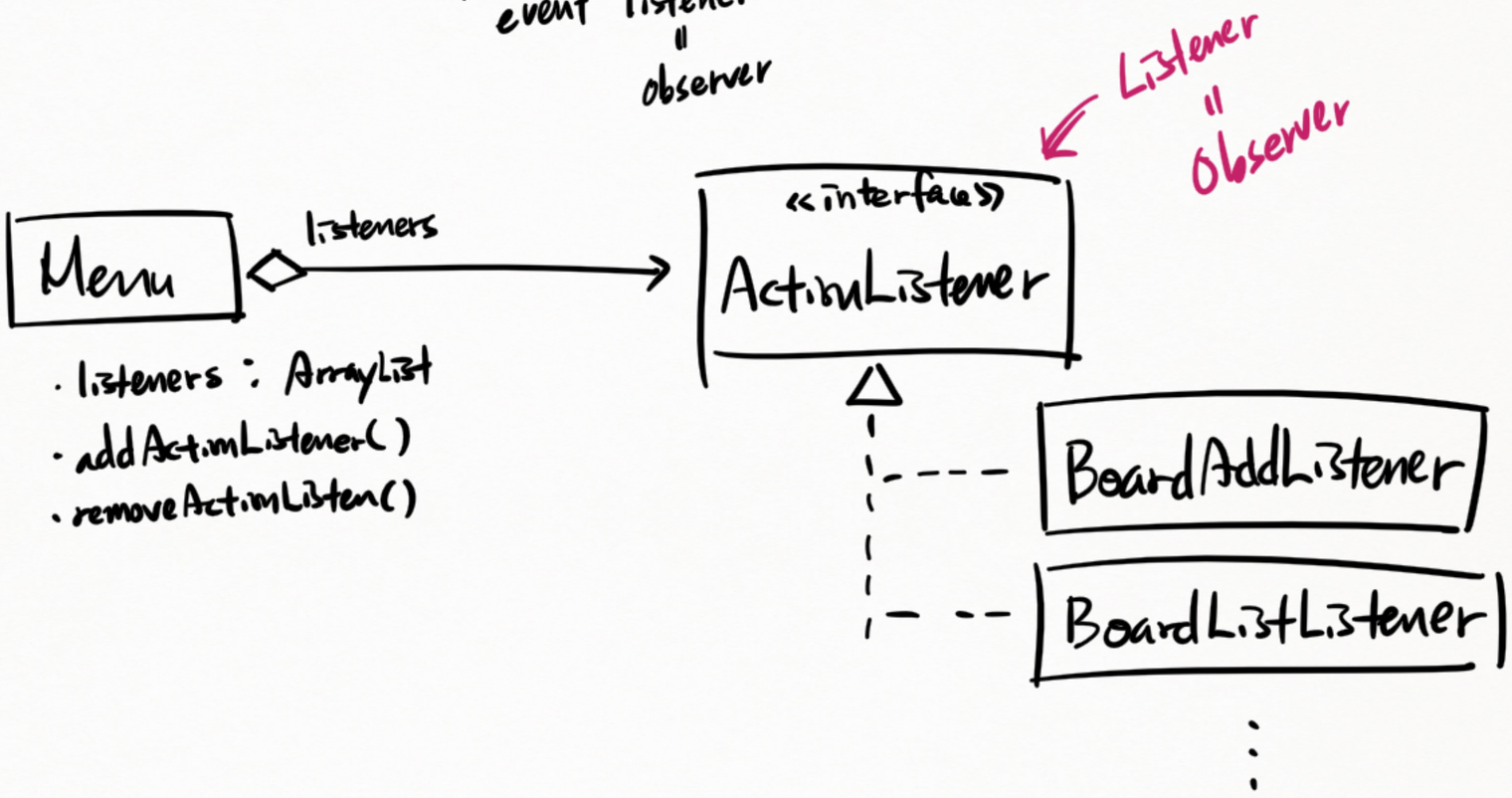


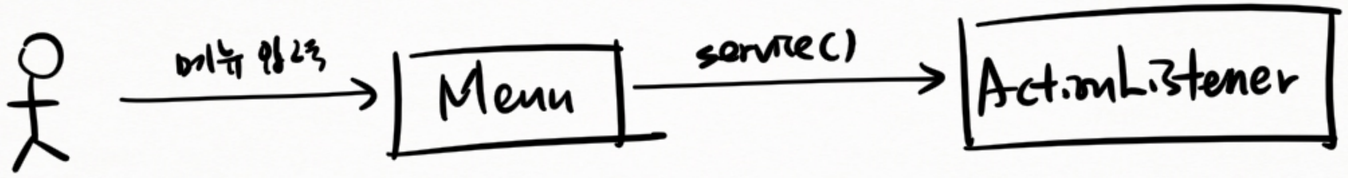
# * Observer 패턴

↳ 객체의 상태가 변경 됐을 때 보내받아서 처리하는 것

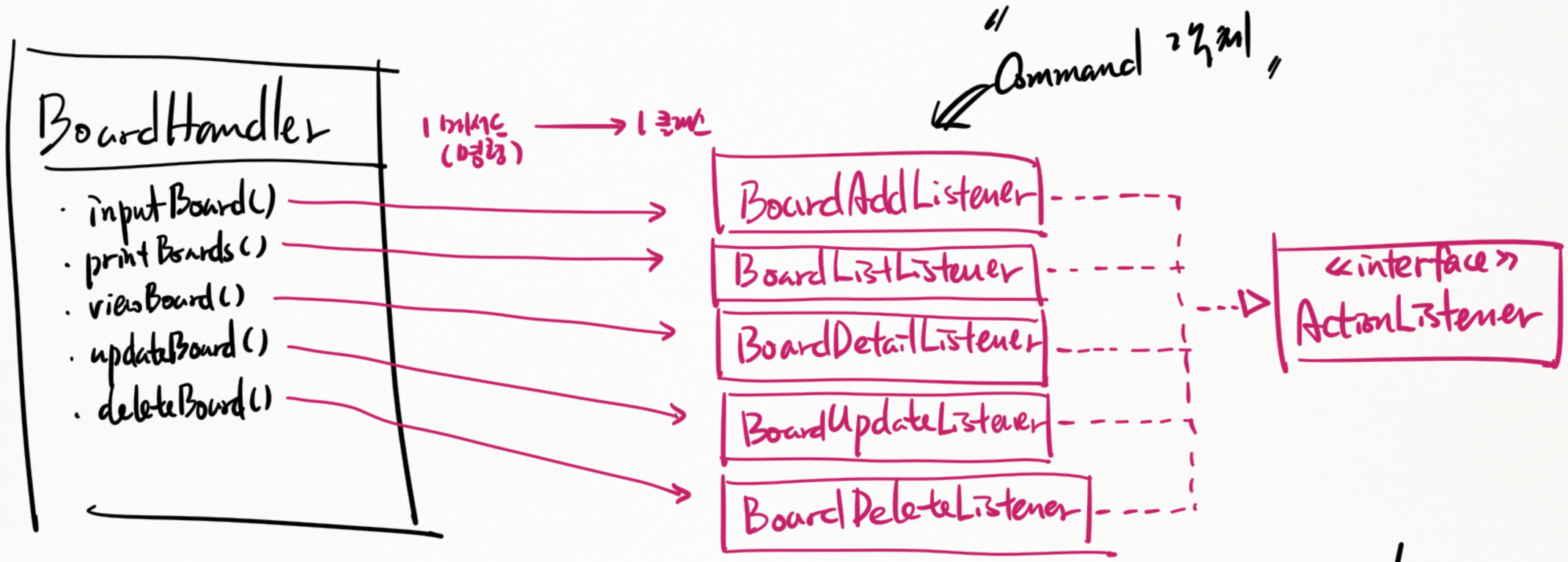
" event listener "
‖
observer



* Observer 패턴의 동작 원리
  ||
  Listener

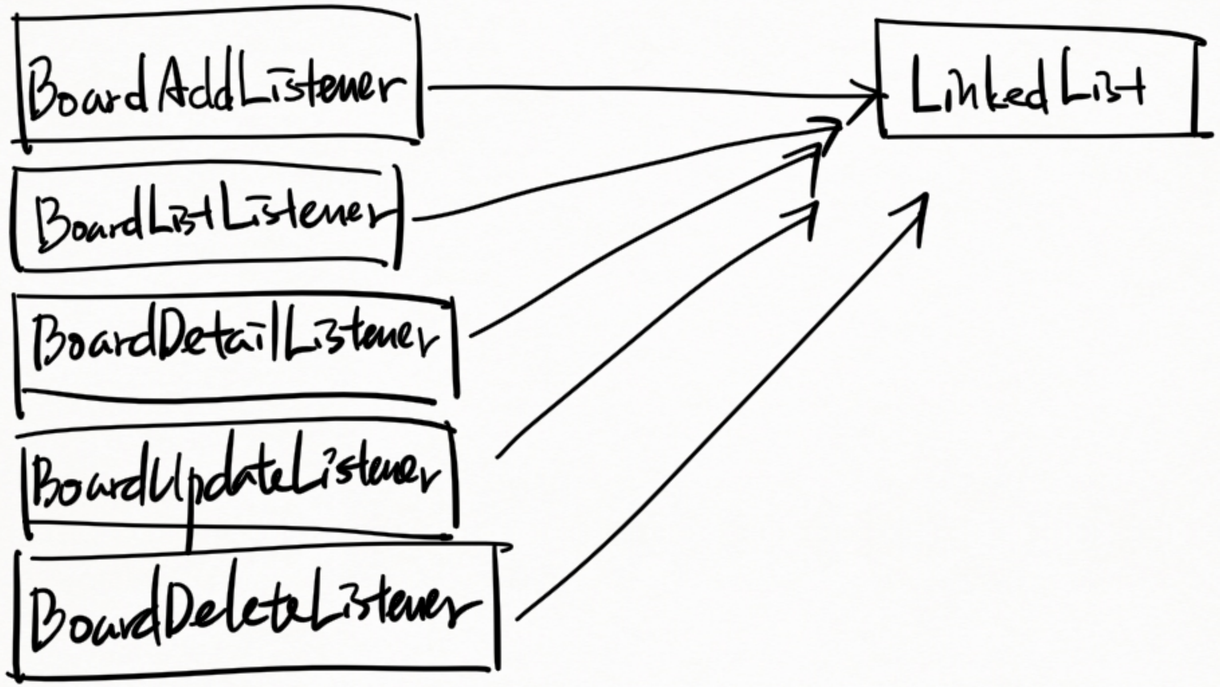
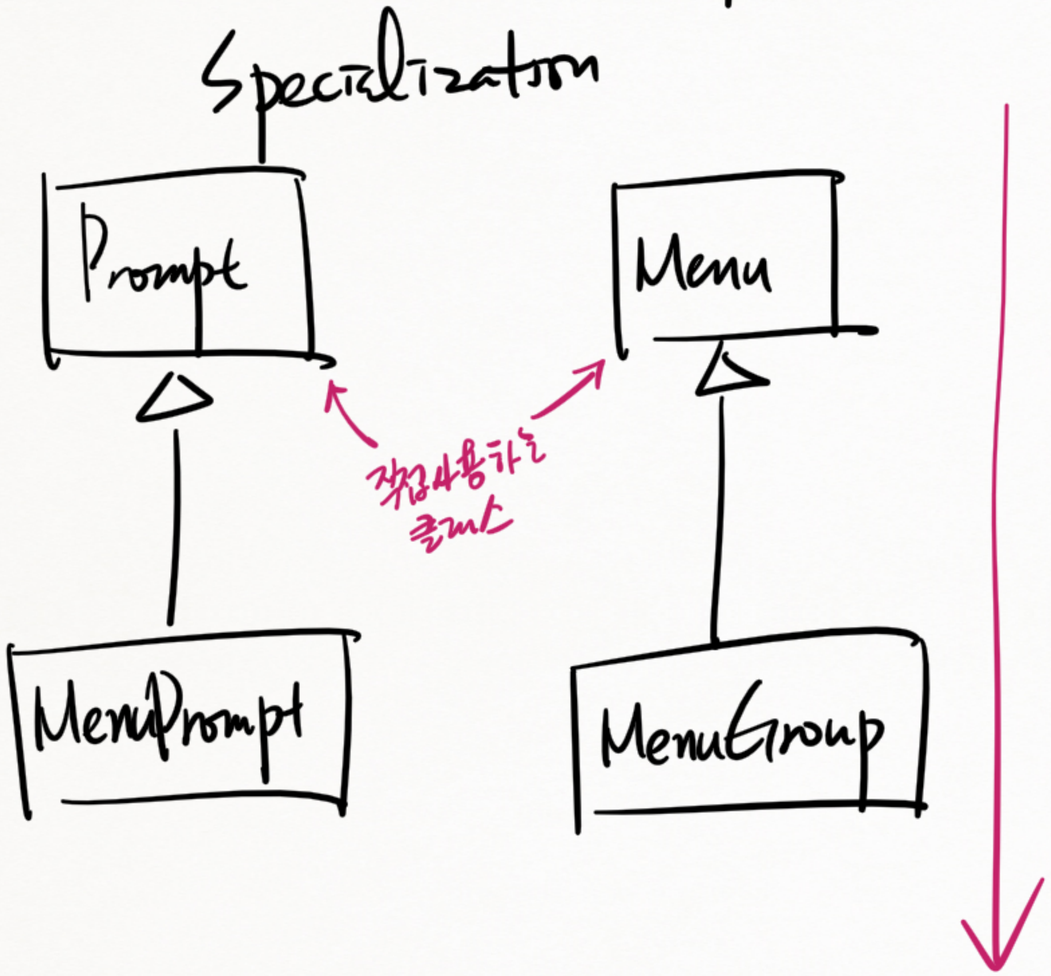
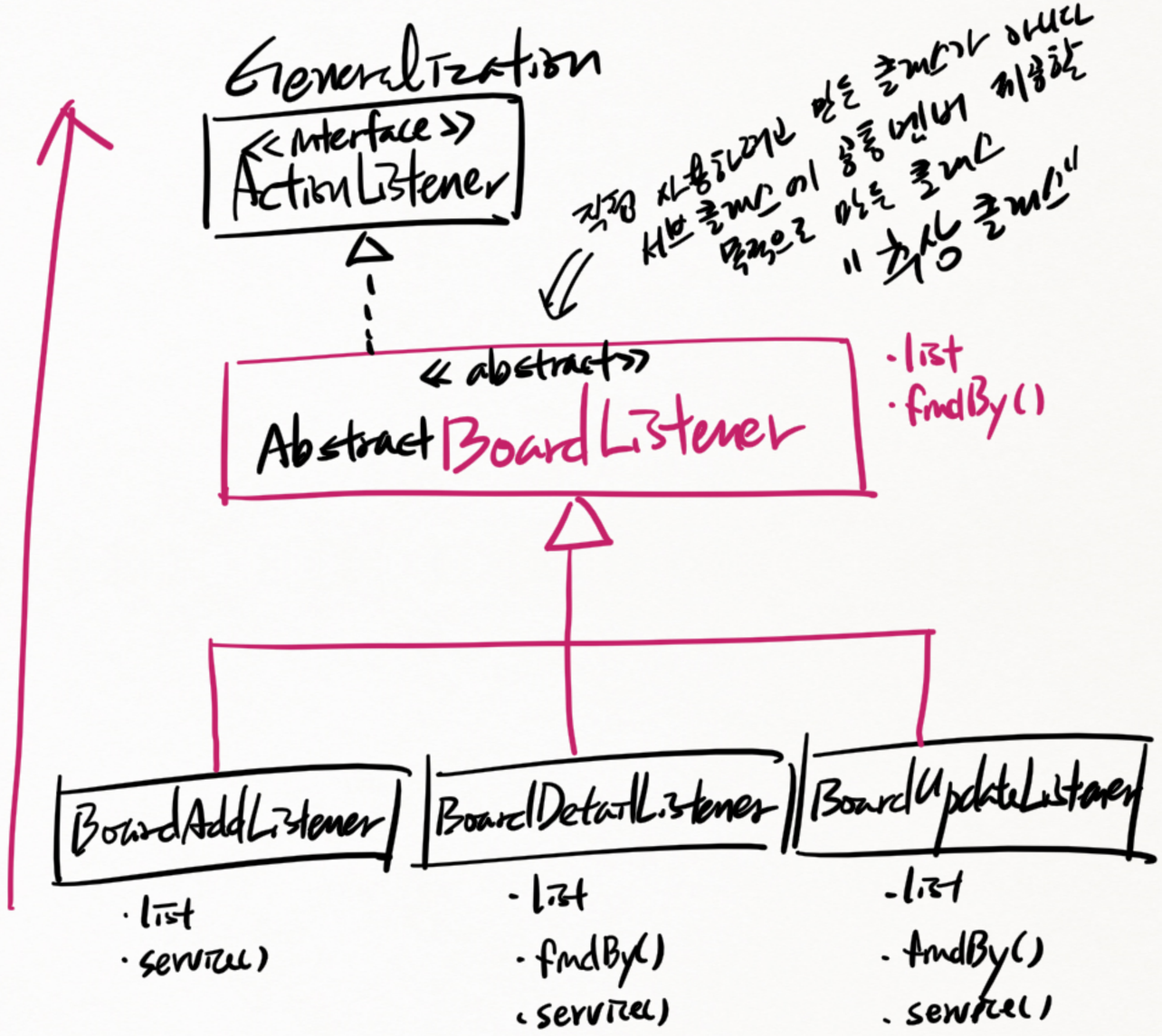


# * Command 패턴




* 이점
↳ 명령 추가 → 클래스 추가
기존 코드를 손댈 필요가 없다!

* 의존객체 공유


BoardAddListener
BoardListListener
BoardDetailListener
BoardUpdateListener
BoardDeleteListener
LinkedList

* 상속 : Specialization vs Generalization