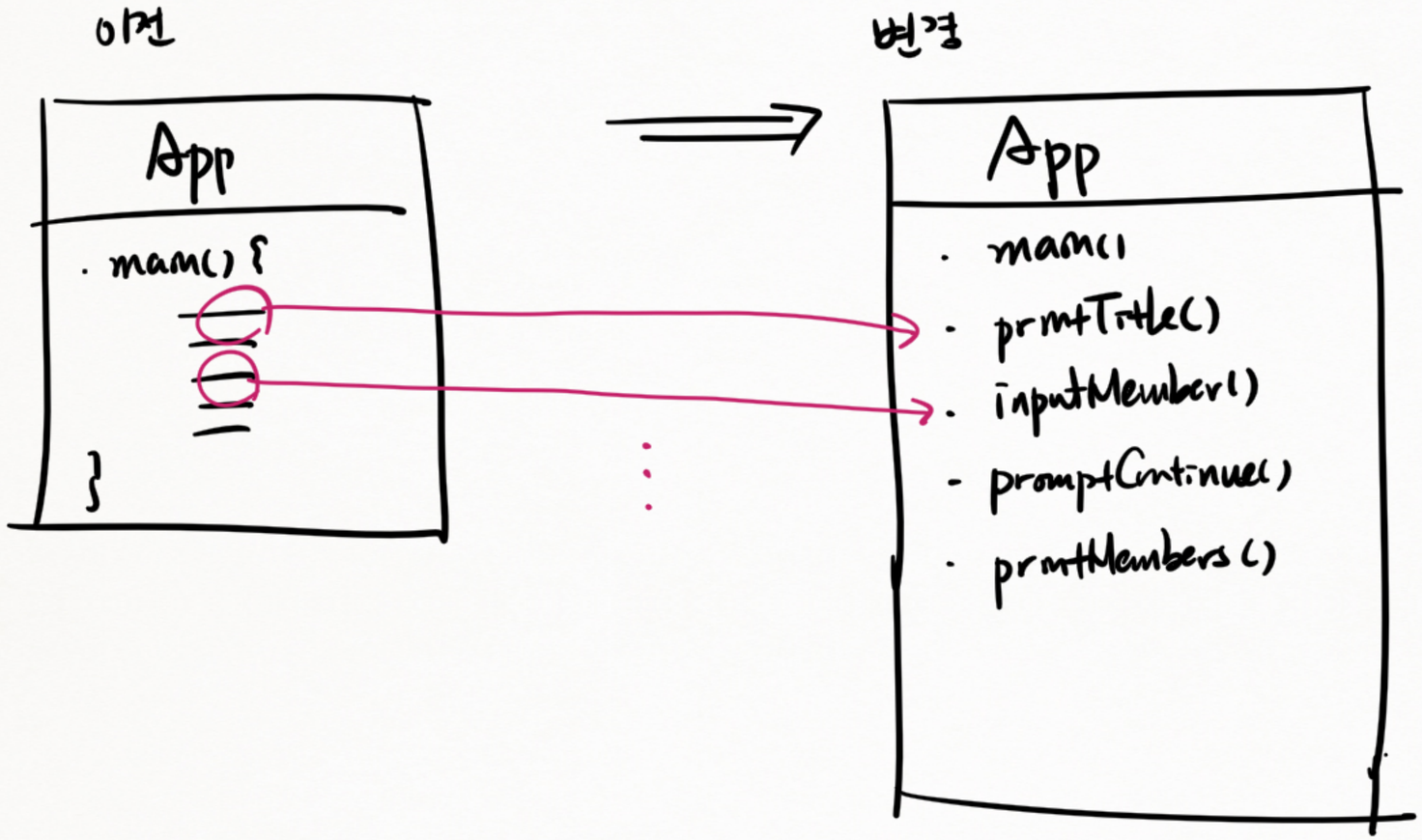
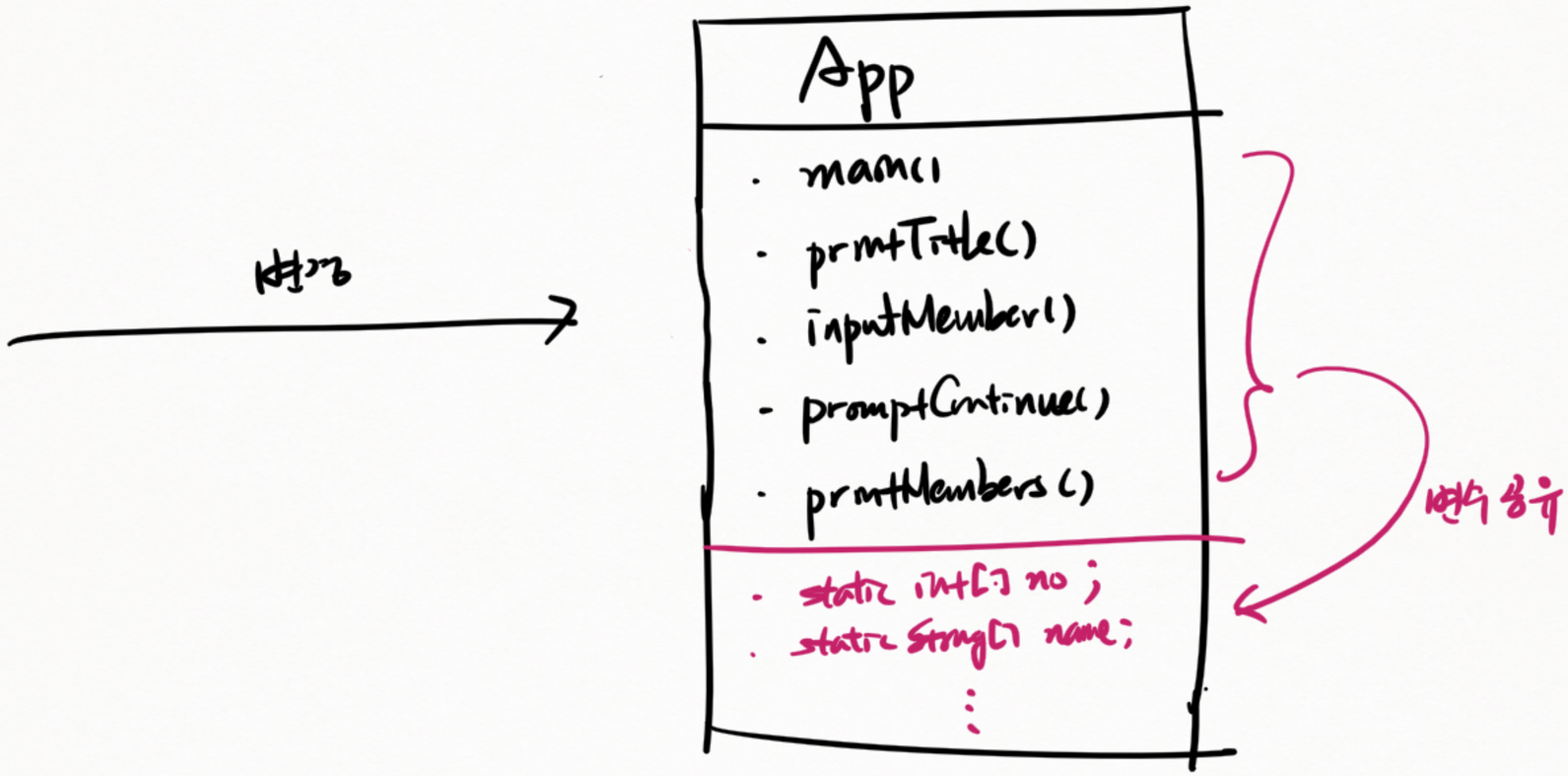
* 7. 메서드 사용법


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

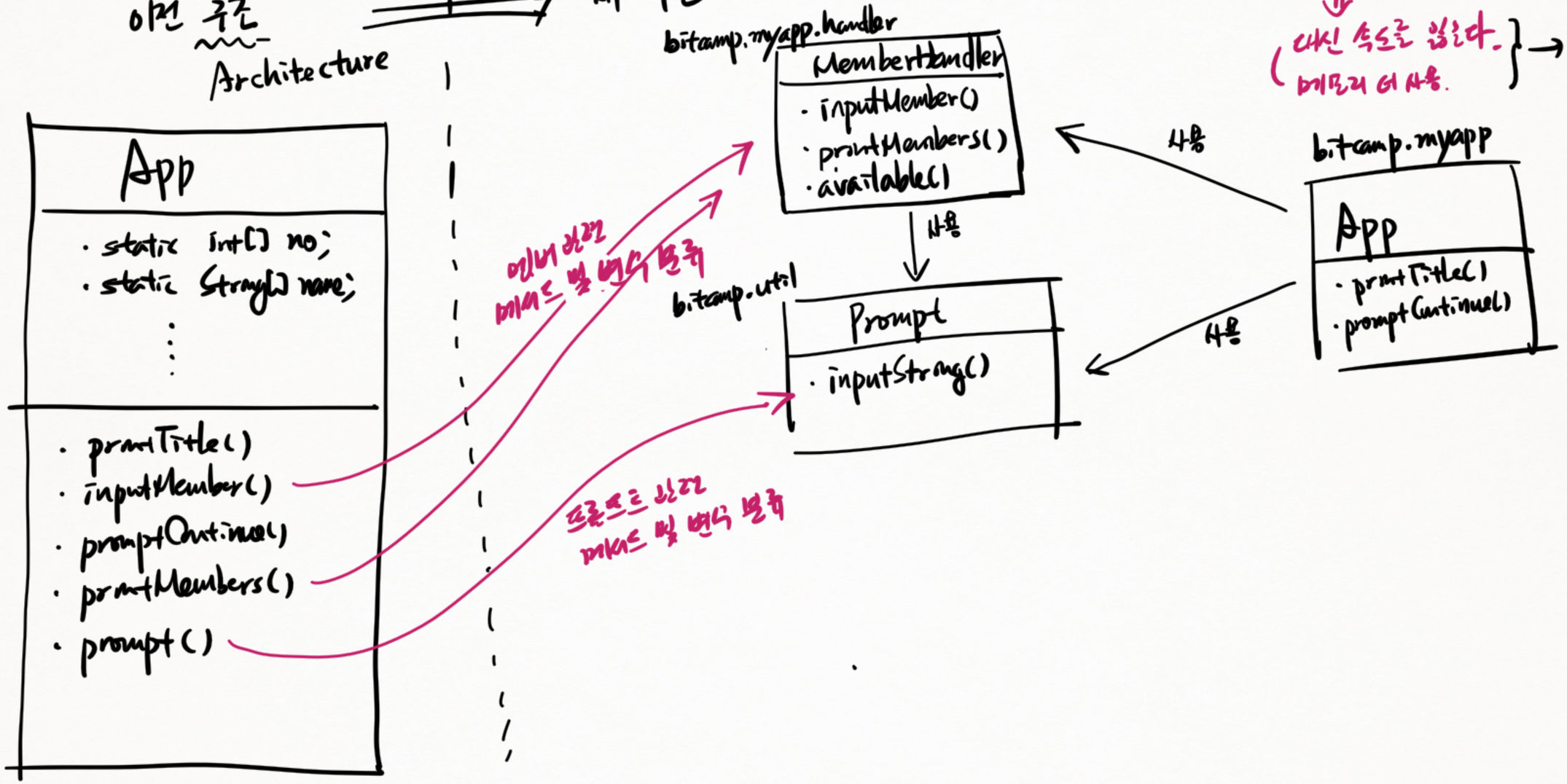
* 응. 스태틱 변수 사용법


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 → 이유? 유지보수를 쉽게.
↳ 재발비용 절감

Architecture



대신 속도를 잃는다. (메모리 더 사용.) → H/W 확장으로 처리

* 10. 메뉴별 CRUD 구현

* 클래스
  ↳ 역할에 따라 메서드를 분류한것
* 패키지
  ↳ 클래스를 분류한 것.

기능을 분류한 것 →

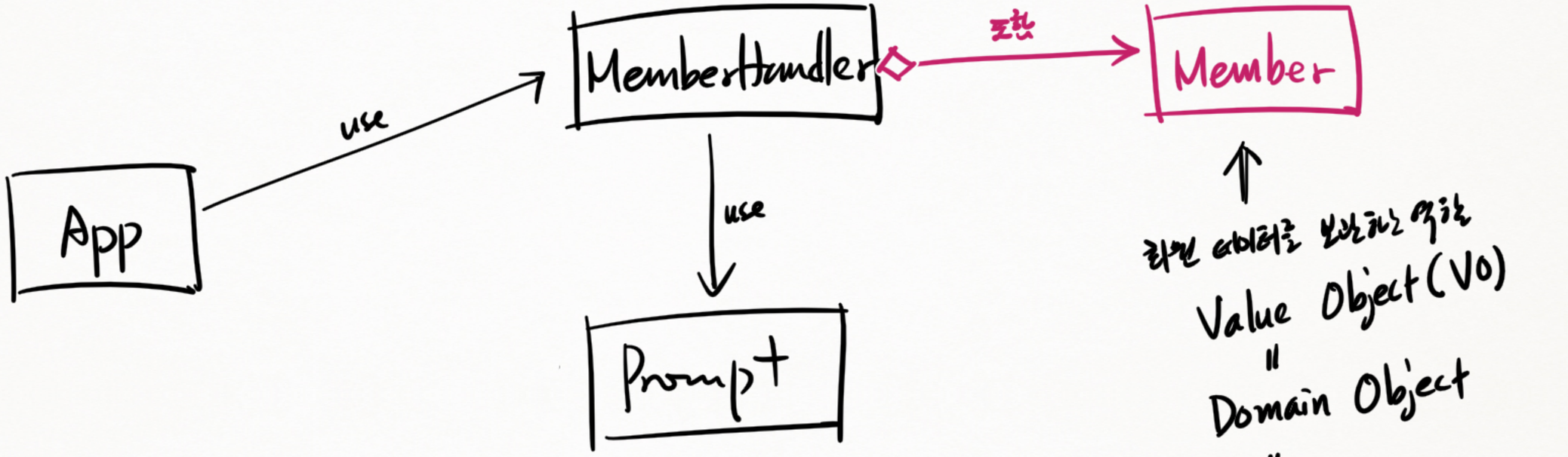
App
- ~~printMenu()~~
- printTitle()
- ~~promptContinue()~~
- main()

MemberHandler ← 회원 데이터를 다루는 메서드를 묶어 놓은것 분류한것.
- inputMember()
- printMembers()
- available()
- viewMember()
- toGenderString()
- updateMember()
- inputGender()
- deleteMember()
- indexOf()

Prompt ← 사용자 입력을 다루는 메서드를 분류한것
- inputString()
- close()
- inputInt()

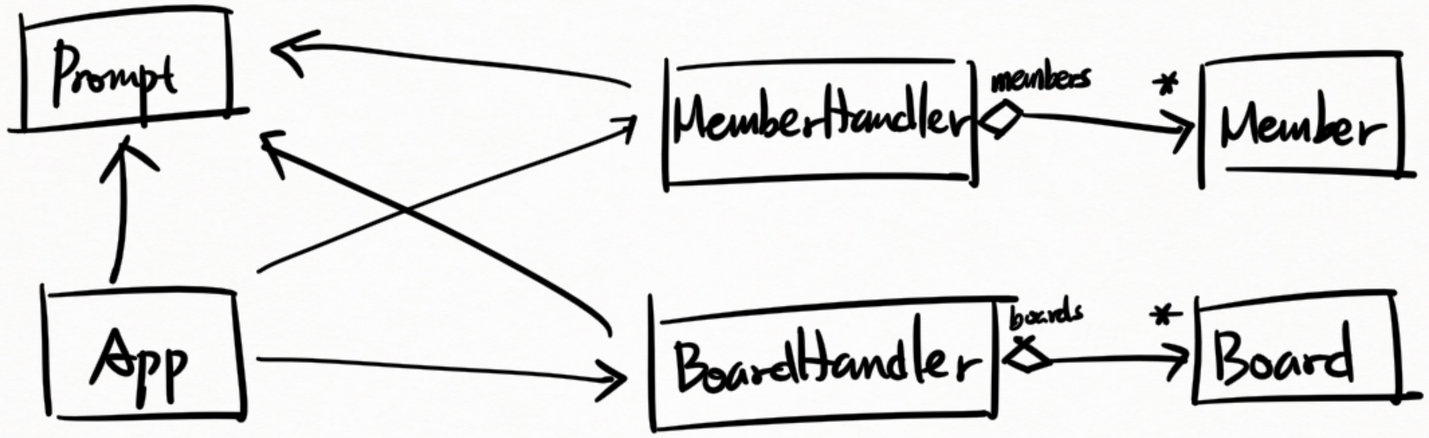
class 분류 → 메서드를 묶는 용도

11. 사용자 정의 데이터 타입 만들기

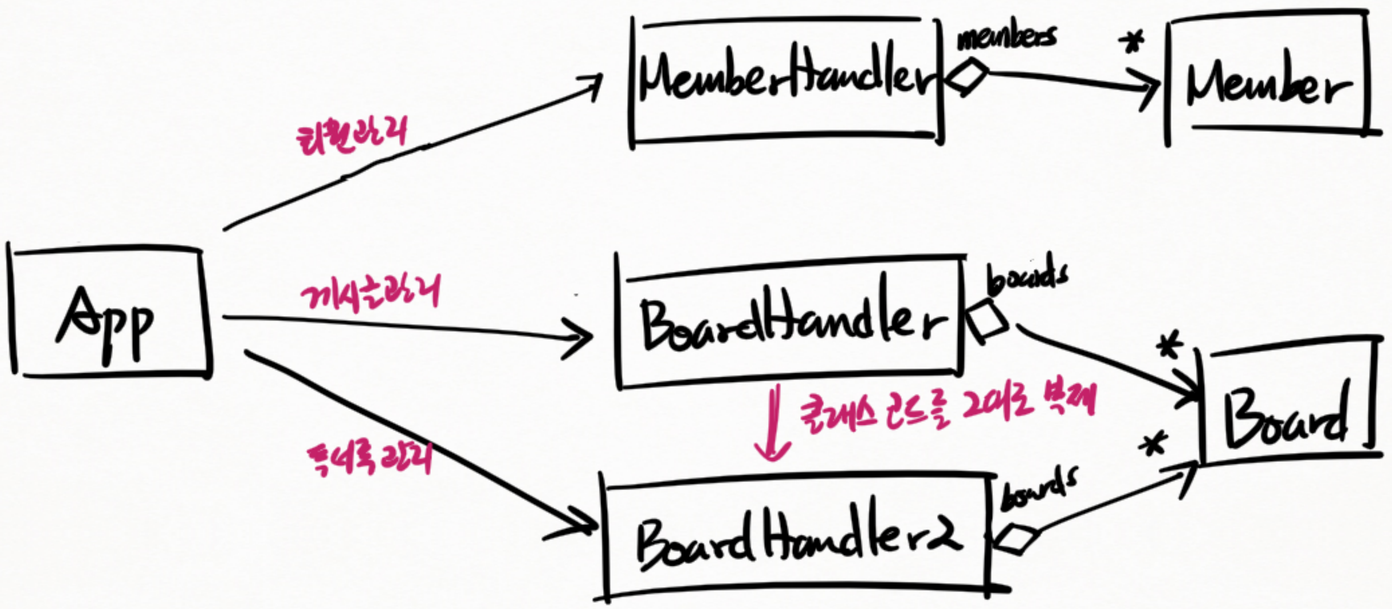


* class 등장 → 데이터 타입을 정의

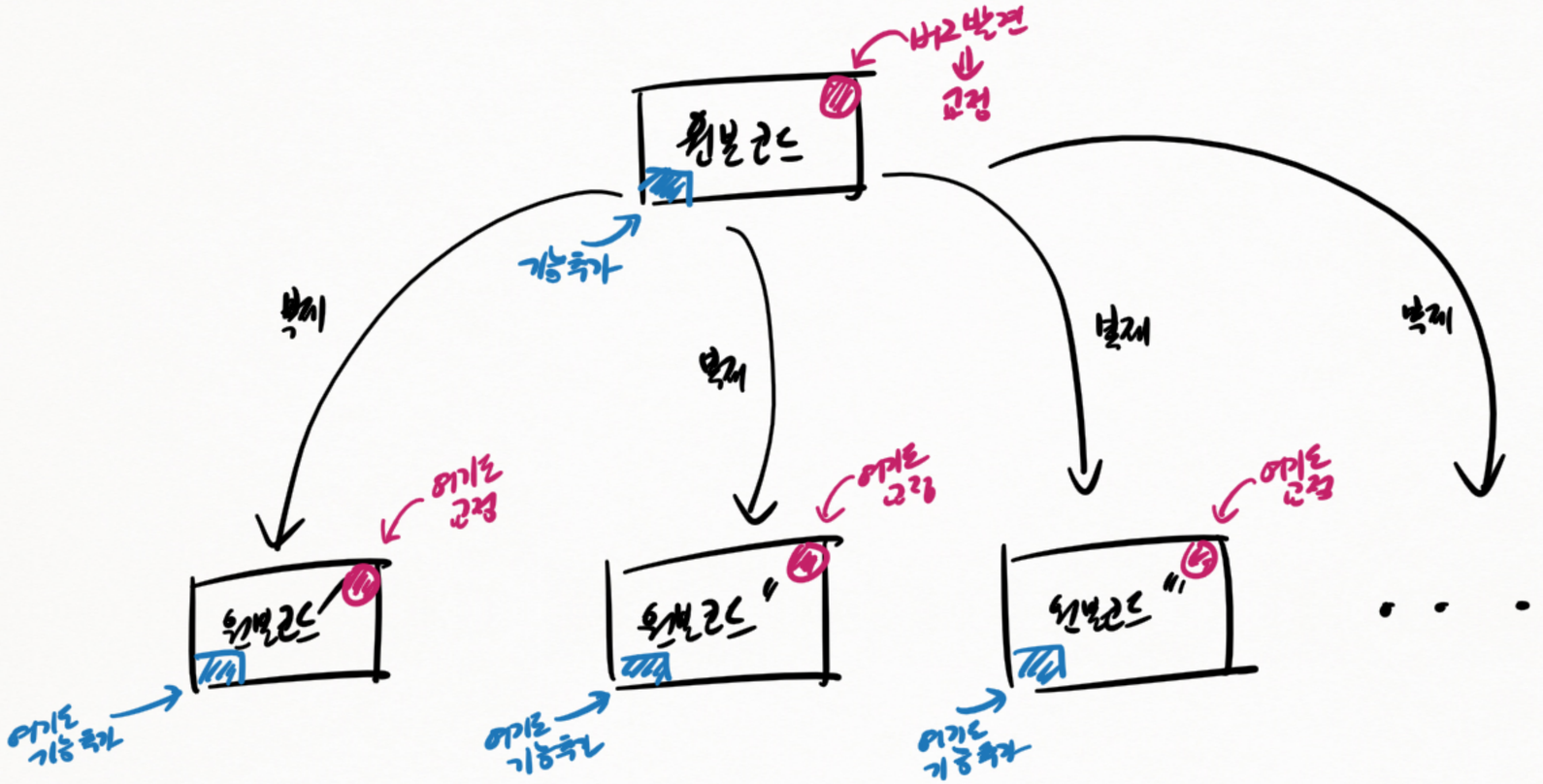
13. 게시글 CRUD 추가



14. 도서목 CRUD 추가



* 같은 코드 복제해서 새기능을 구현할 때 문제점



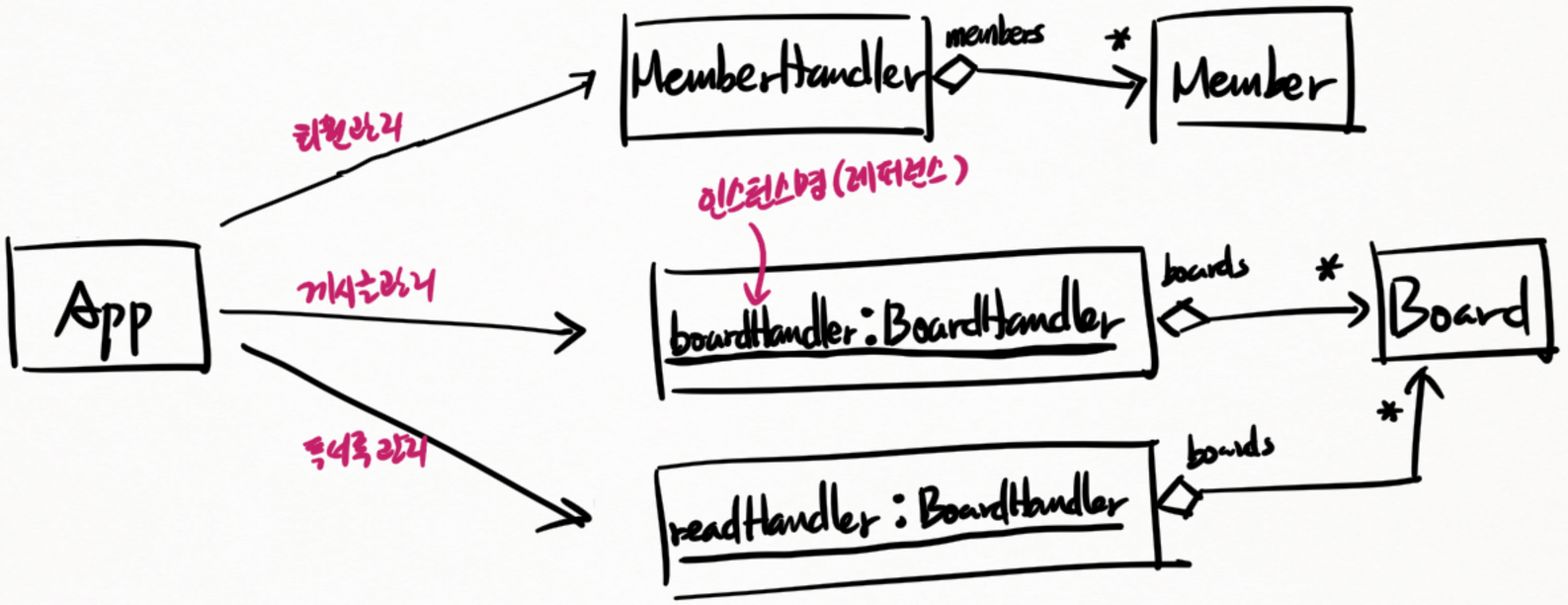
* 버그를 수정하거나 기능을 추가해야 하면 복제한 코드에 대해서도 똑같은 일을 수행해야 한다.

⇓

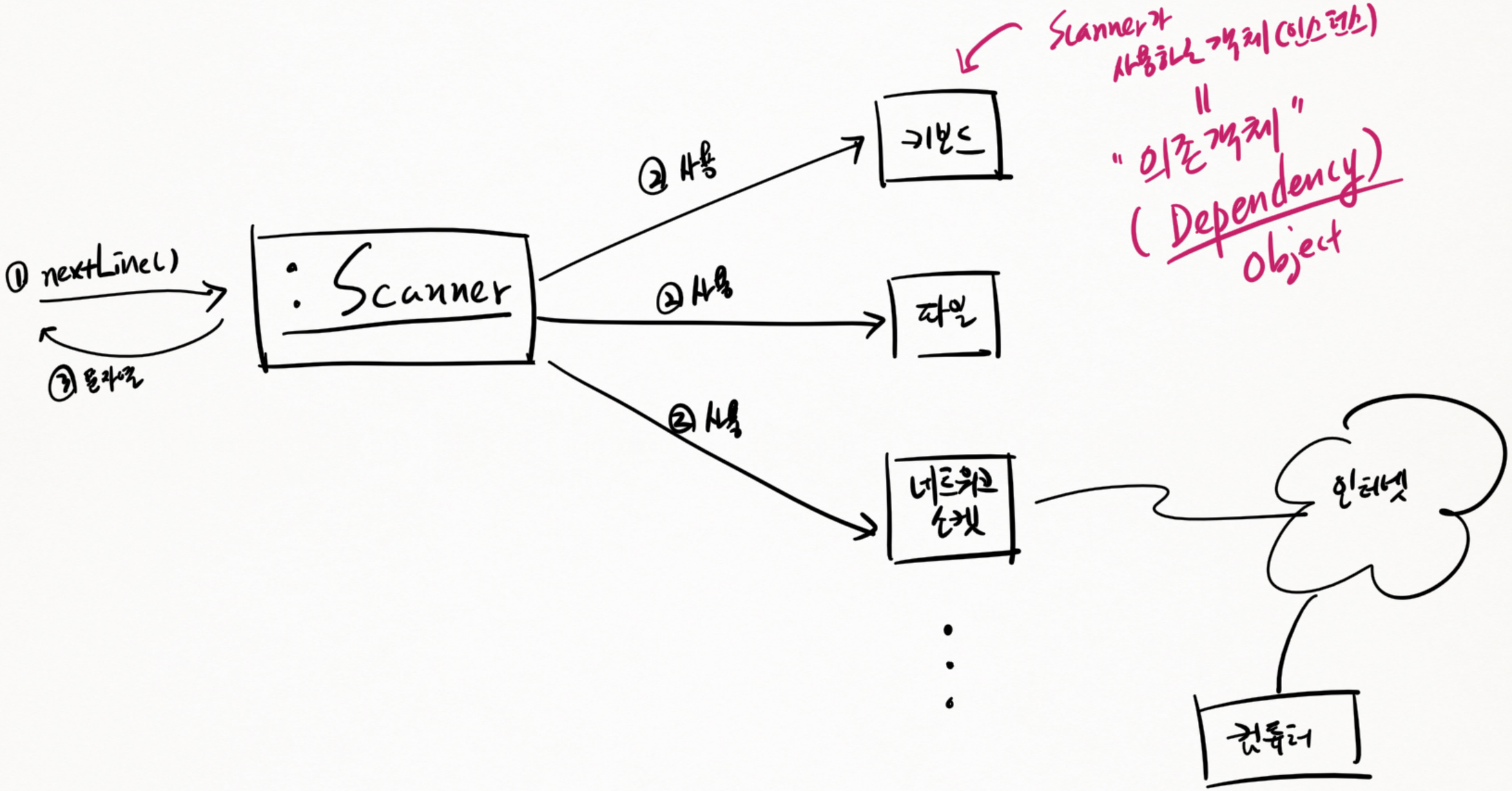
코드를 복제하면 할수록 유지보수가 점점 힘들어진다.

"기술부채"

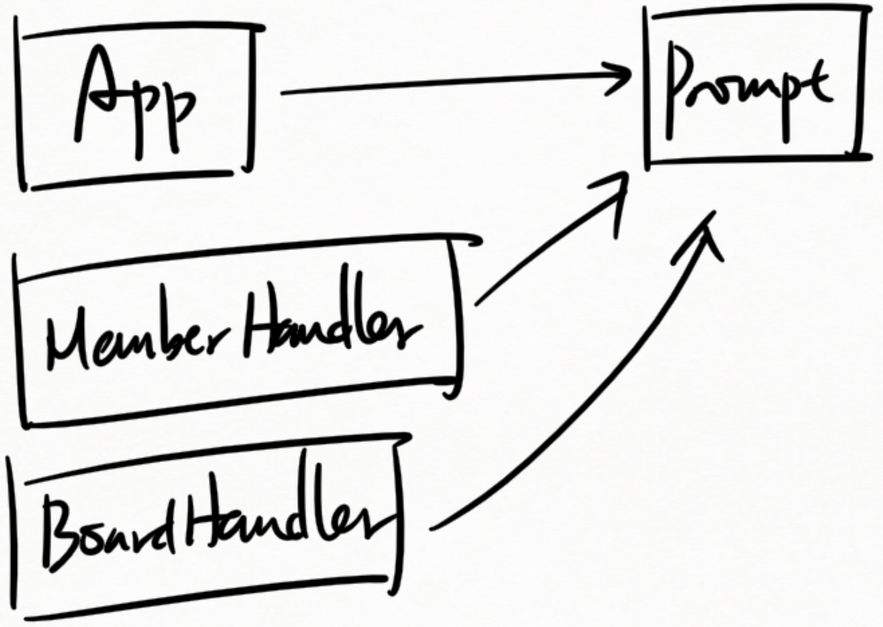
14. 인스턴스 변수 및 메서드 활용



Scanner 와 의존객체



App, MemberHandler, BoardHandler 에 Prompt



의존객체

생성자를 통해 주입!

어떻게 콘솔로 작동하나?

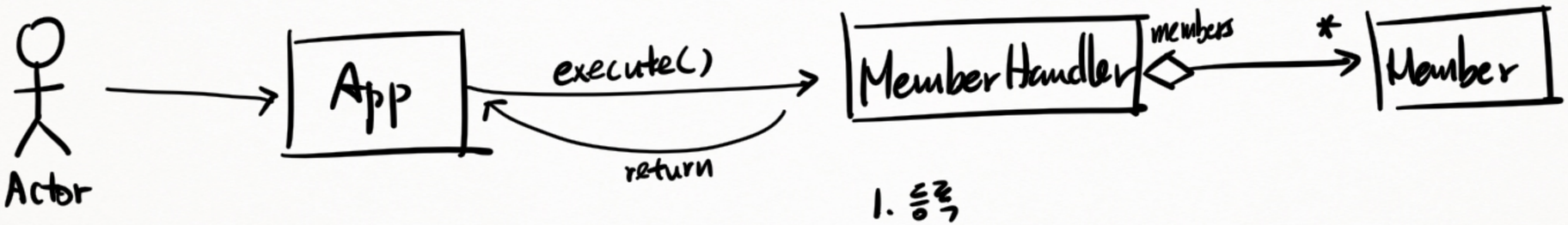
"의존객체 주입"
Dependency Injection
(DI)

* Prompt 가 인스턴스 방법을 변경

이 객체를 사용하는 클래스를 변경해야 한다.

생성자를 통해 의존객체(Prompt)를 주입!
Injection

16. Handler 에게 메뉴 기능을 위임



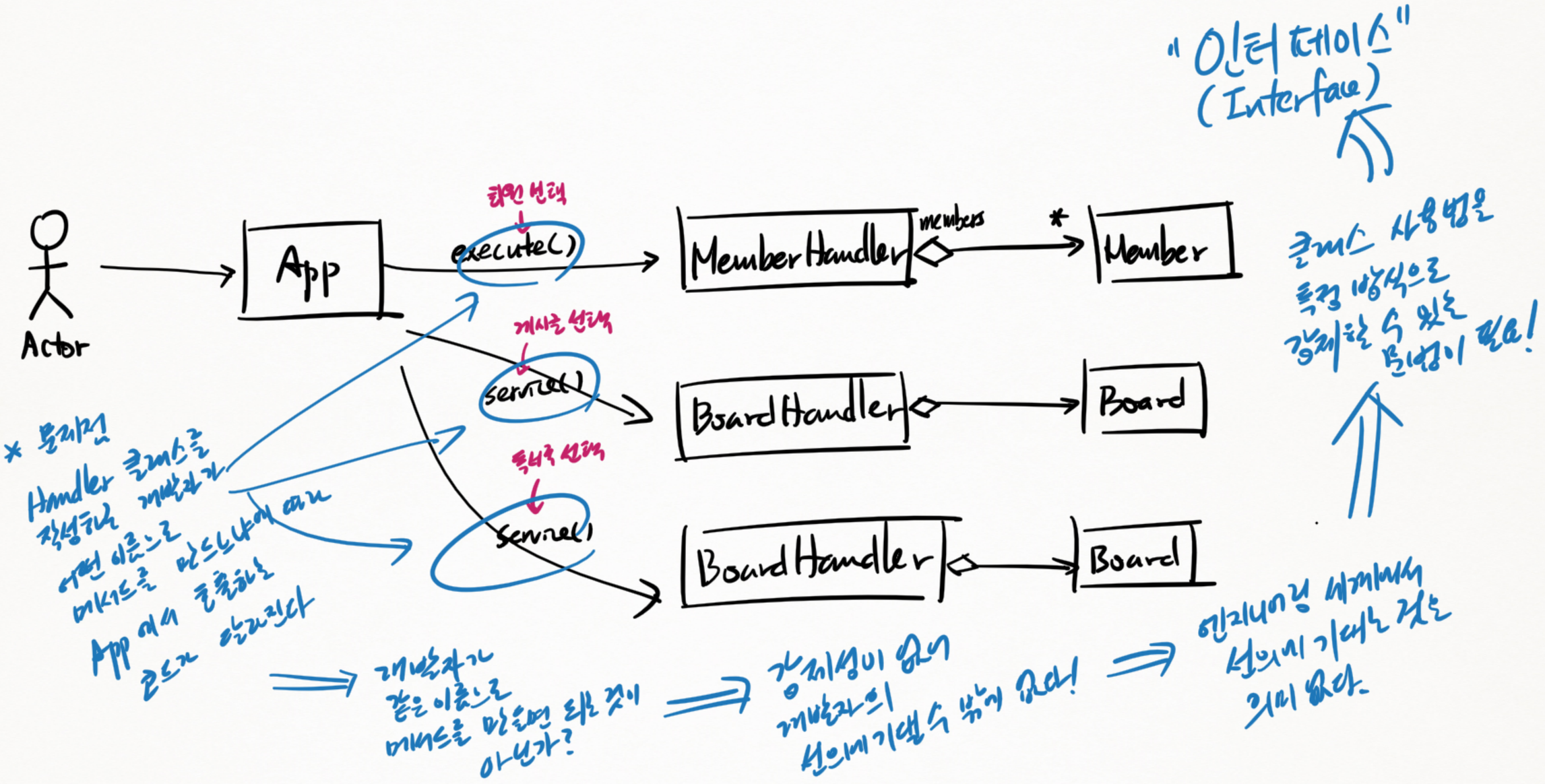
1. 회원
2. 게시글
3. 독서록
0. 종료

메인>

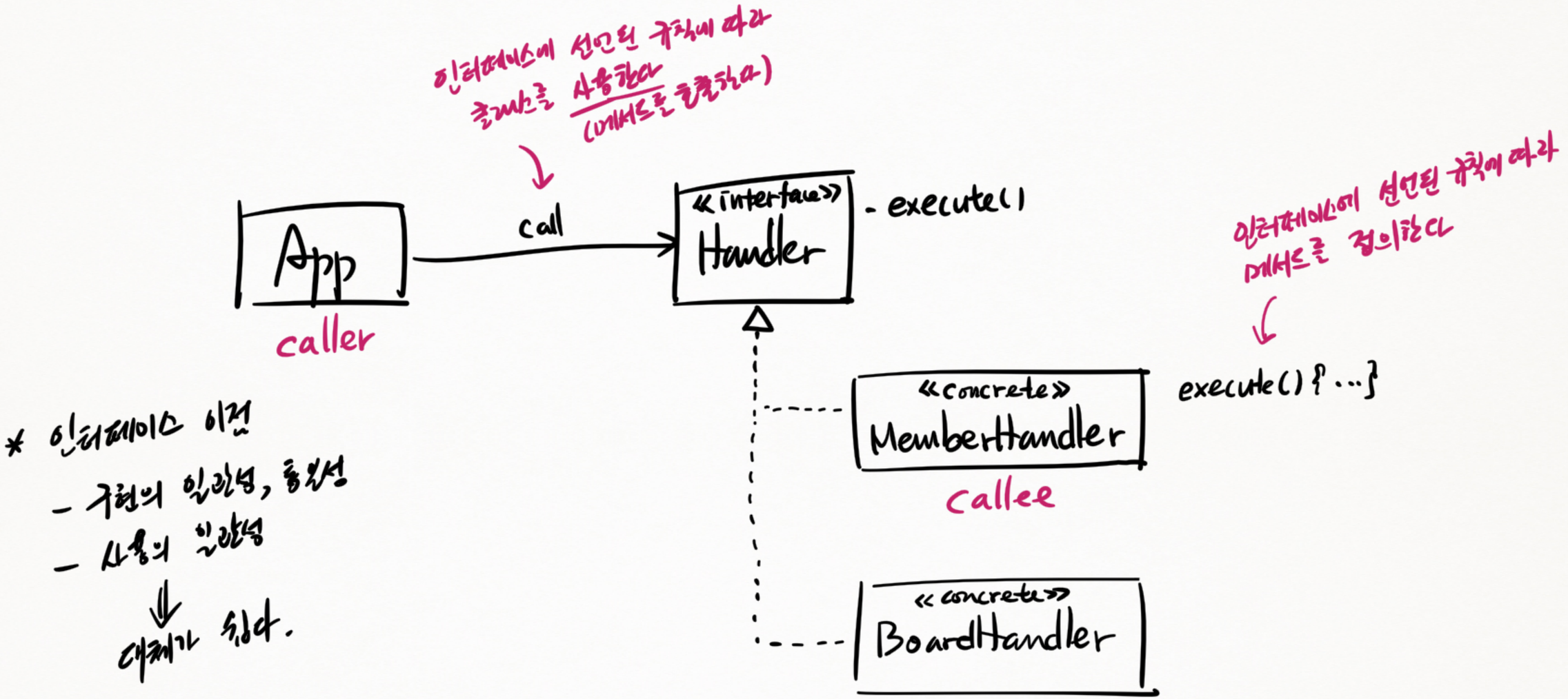
1. 등록
2. 목록
3. 조회
4. 변경
5. 삭제
0. 메인

회원>

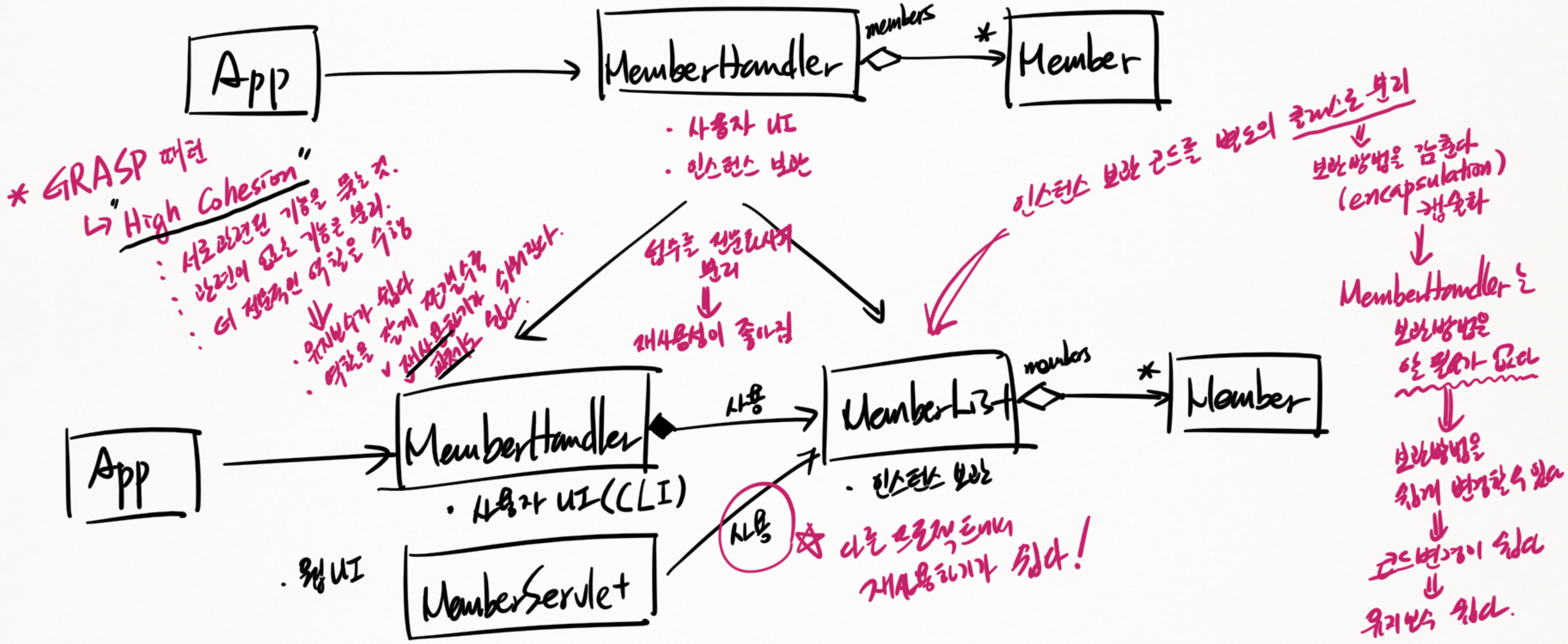
16. Handler 에게 메뉴 기능을 위임



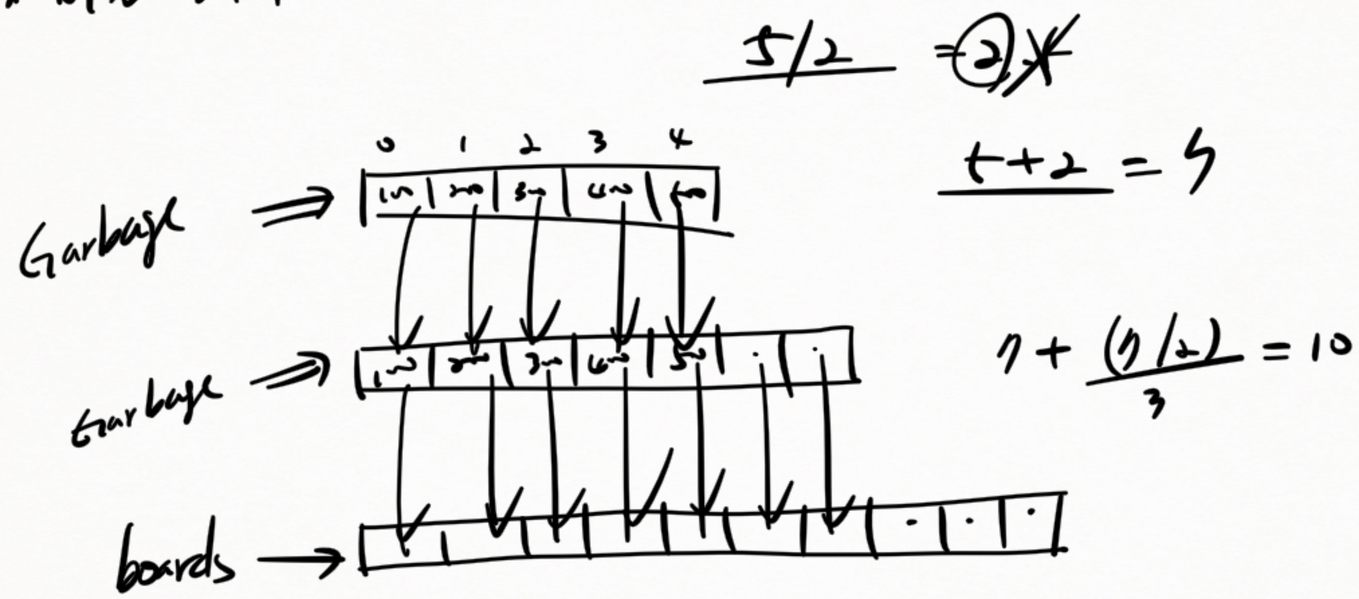
11. Handler 의 사용규칙을 인터페이스로 정하기



18. 인스턴스 목록을 다루는 코드를 별도의 클래스로 분리

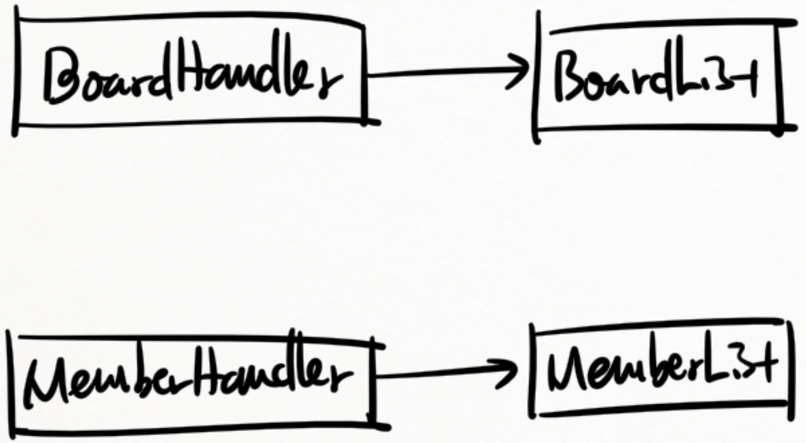


* 배열 늘리기

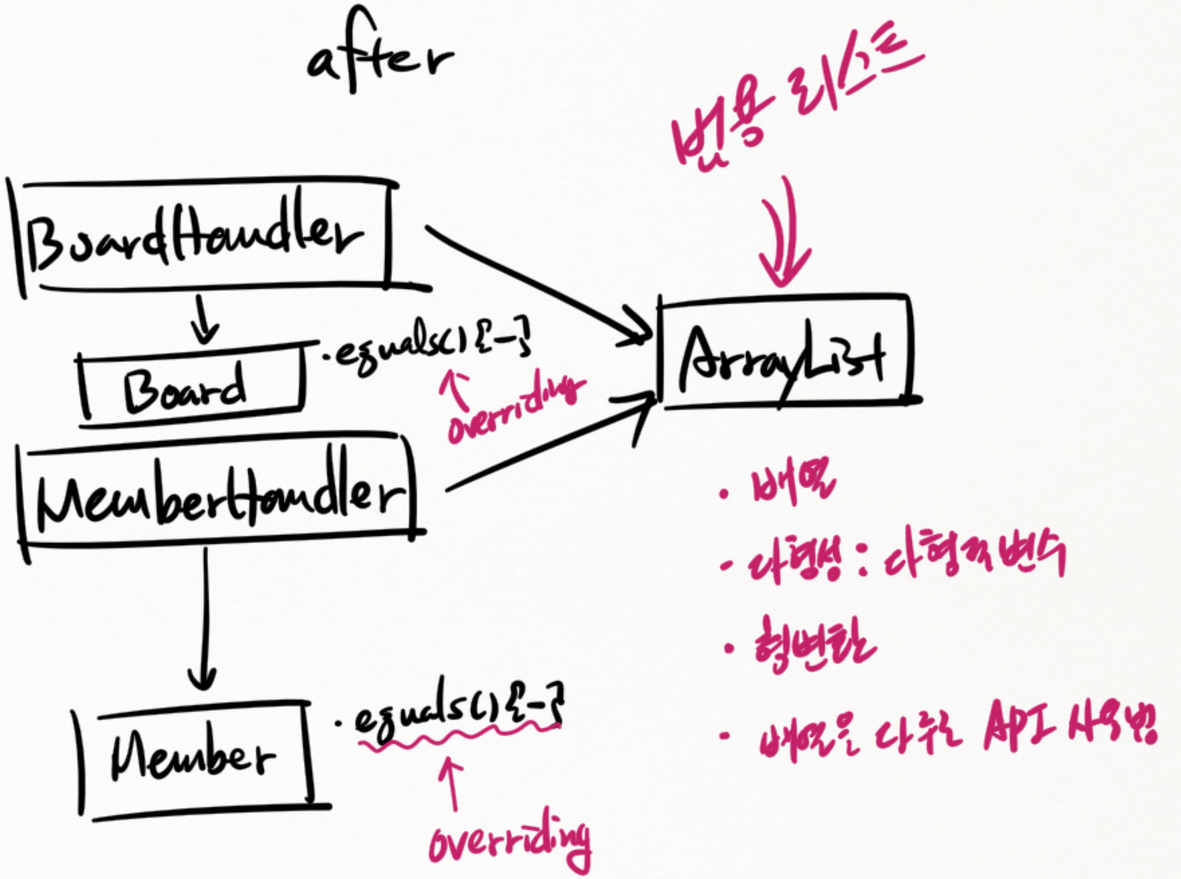


19. 범용 리스트 만들기

before

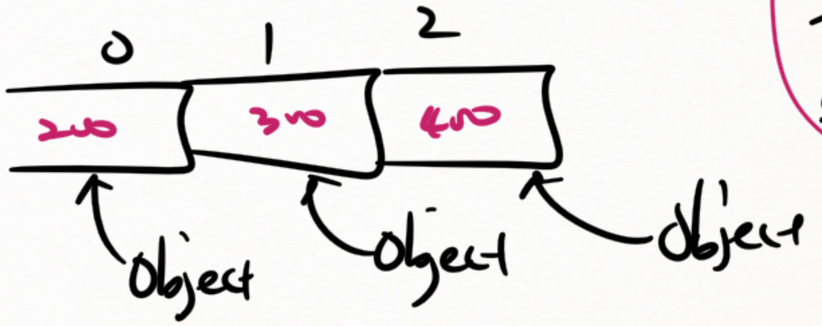


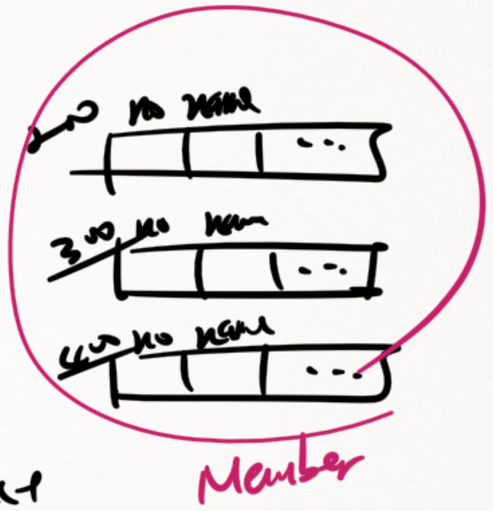
after



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

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

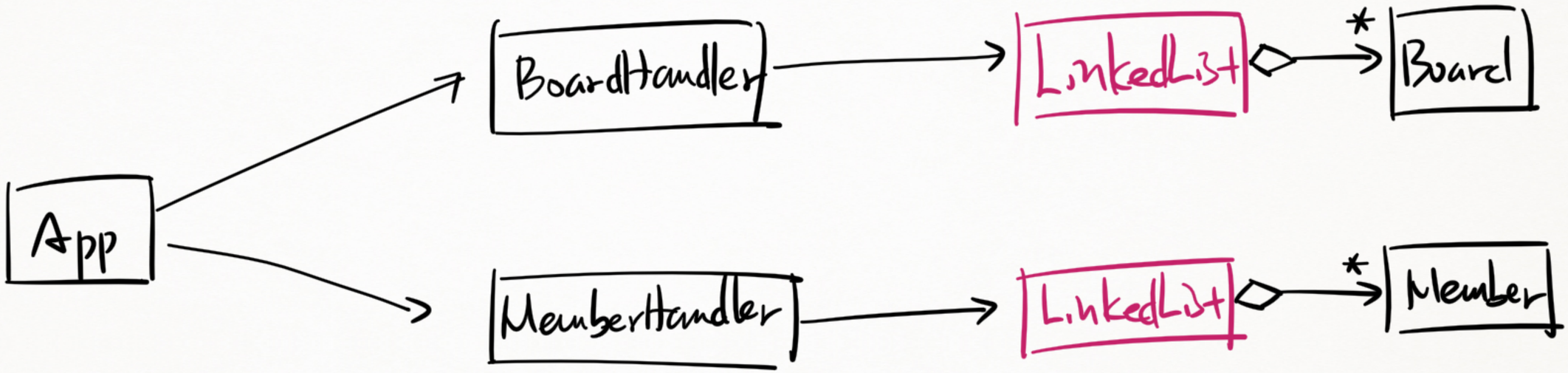





```
Member[] arr2 = (Member[]) arr1;
```

arr1 이 가리키고 있는 Member 레퍼런스 배열이 아니라 Object 레퍼런스 배열이다. 강제 형변환 할 수 없다!

* 20. LinkedList 자료구조 직접 구현해 사용하기



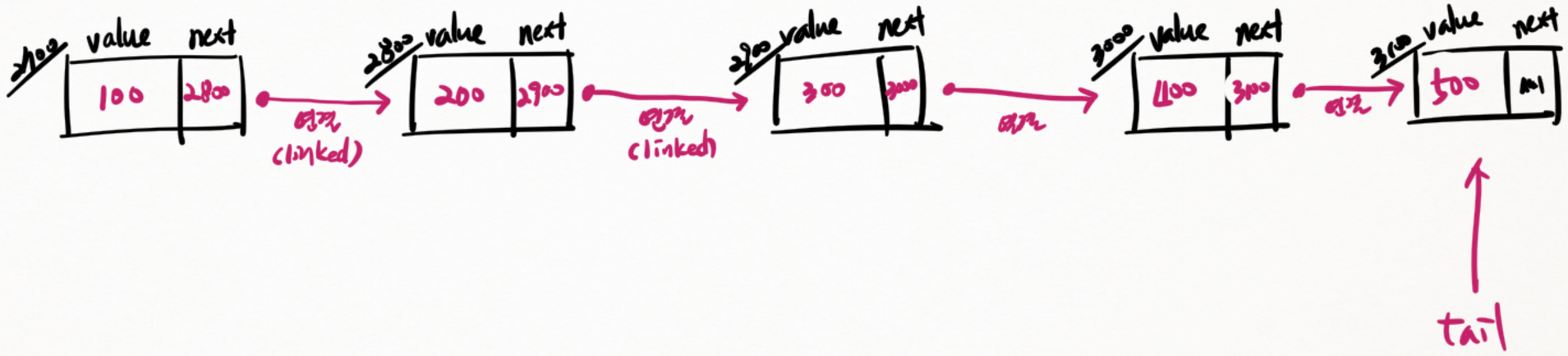
ArrayList vs LinkedList

	ArrayList	LinkedList
초기크기	일정 메모리 확보 (10개)	0 부터 시작
크기증가	Yes → 기존배열이 가비지가 된다	Yes → 가비지 없다.
조회	인덱스로 조회 (LinkedList보다 빠르다)	링크를 따라가야 한다 (ArrayList보다 느리다)
삽입, 삭제	삽입 → 해당값 밀기 삭제 → 해당값 앞기기 (LinkedList보다 느리다)	해당 값을 찾은후 노드 추가/삭제 (List보다 빠르다)

LinkedList - add()

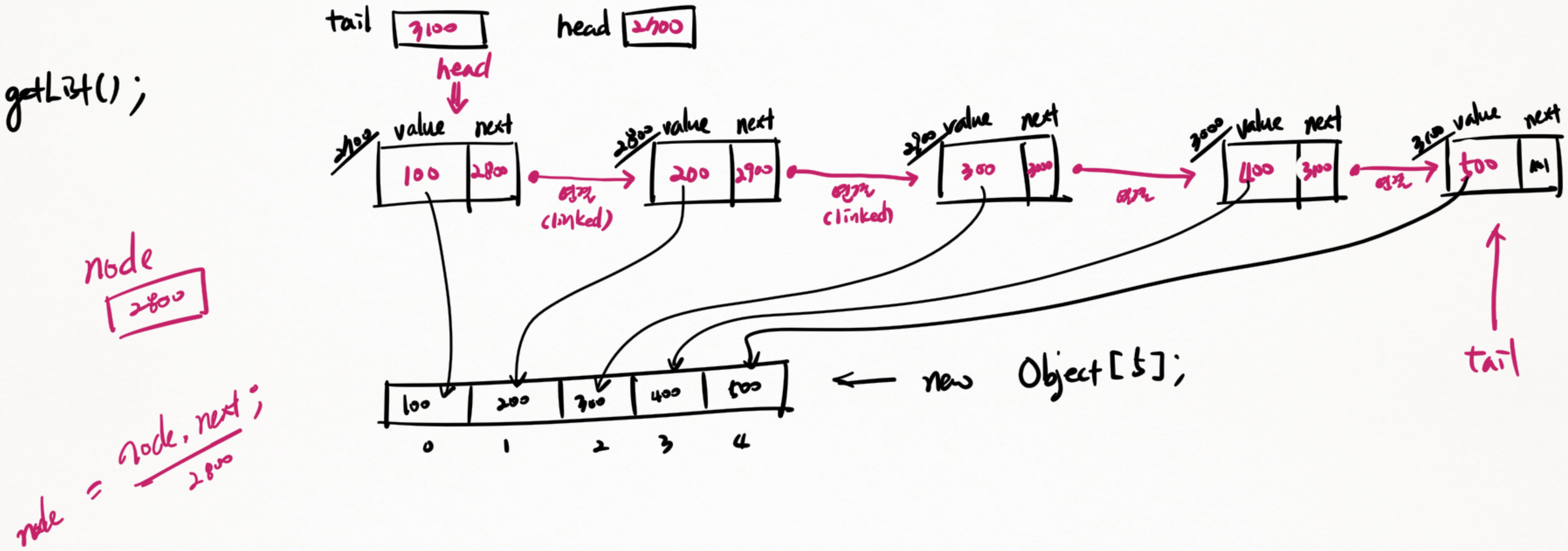
tail 3100

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



LinkedList — getList()

getList();



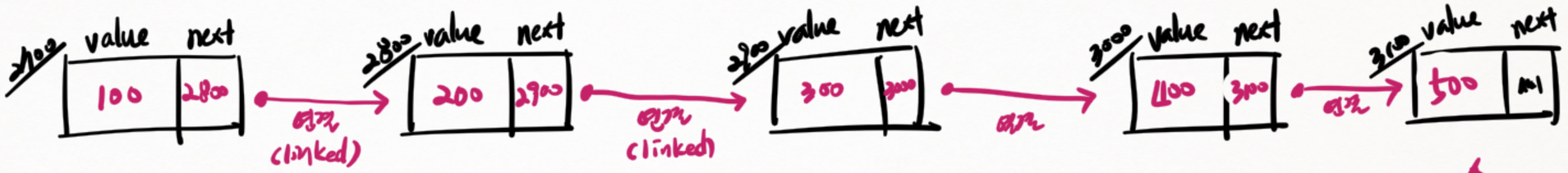
node
2800

node = node.next;
2800

LinkedList — retrieve()

tail 3100

retrieve(900);



cursor

null

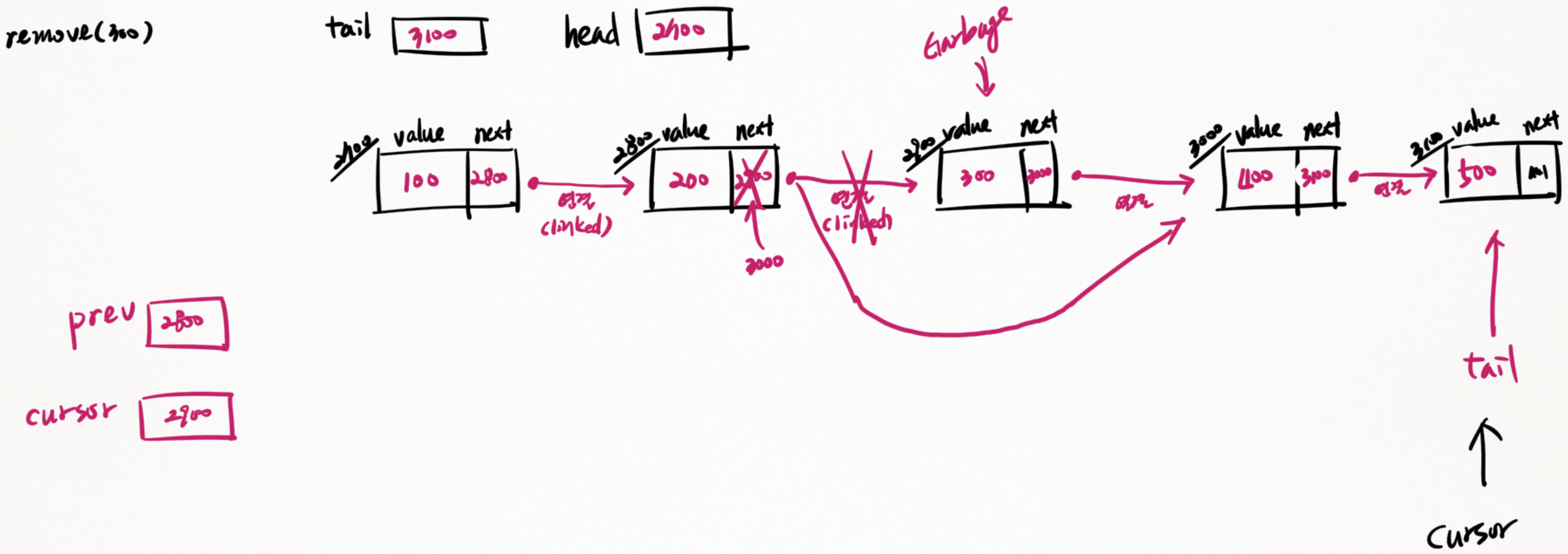
500.equals(900)

tail

cursor

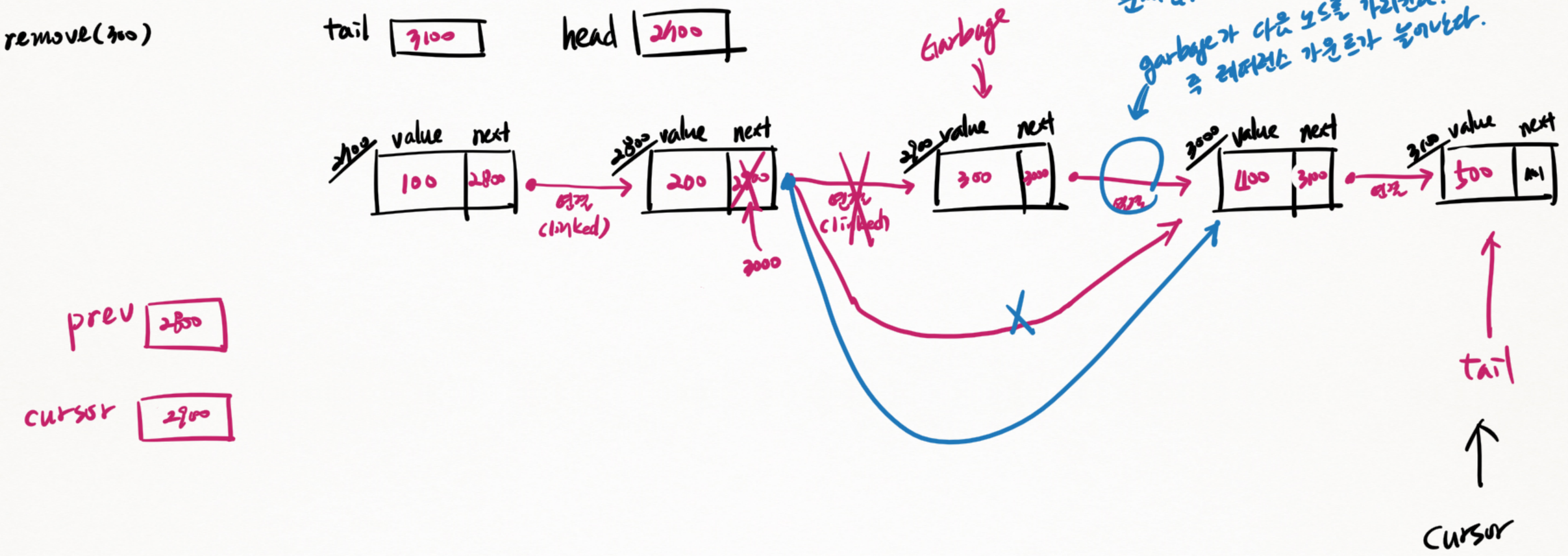
LinkedList — remove(1) : 중간 항목 삭제

remove(300)



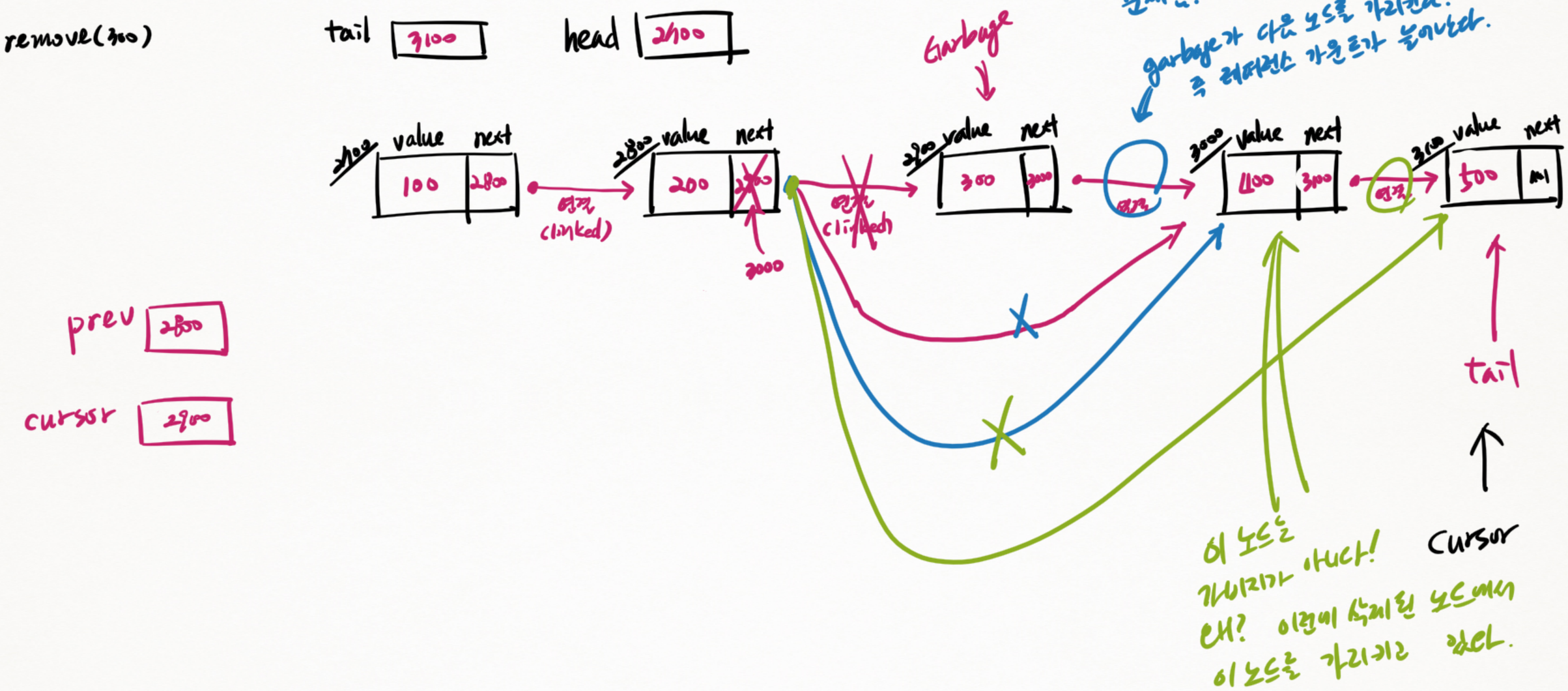
LinkedList — removeCl : 중간 항목 삭제 ⇒ 가비지가 인스턴스를 가리킬때 문제점.

remove(300)

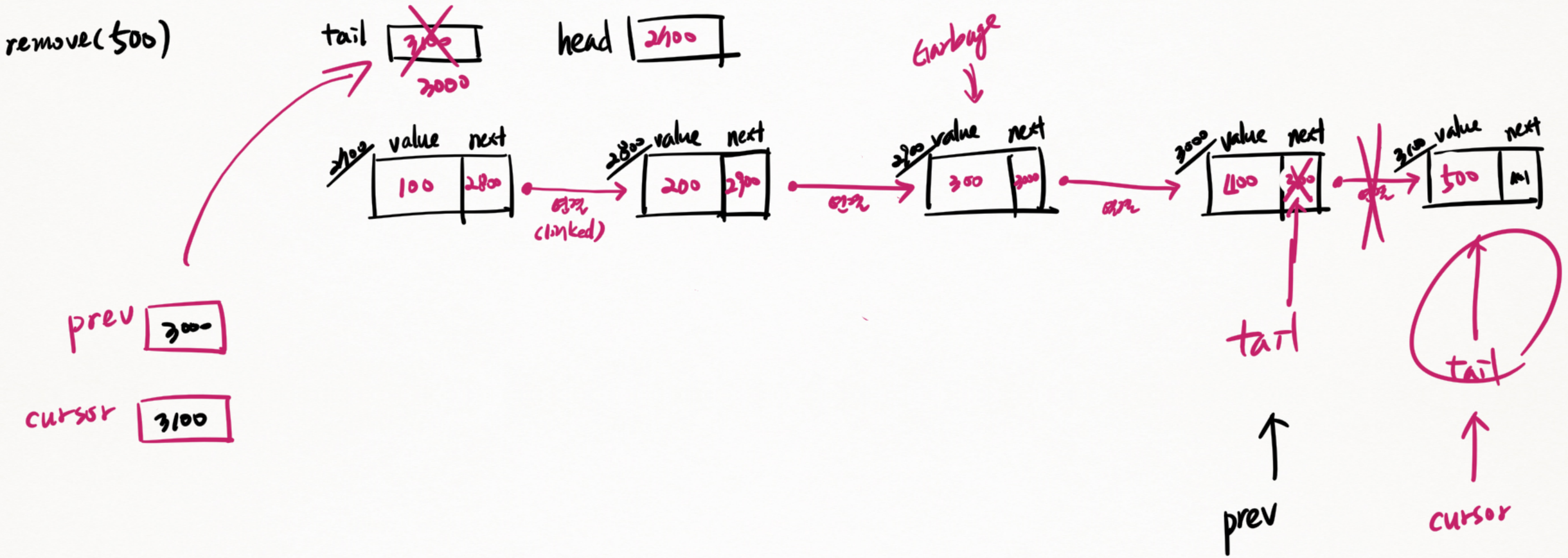


LinkedList — remove() : 중간 항목 삭제 ⇒ 가비지가 인스턴스를 가리킬때 문제점.

remove(300)

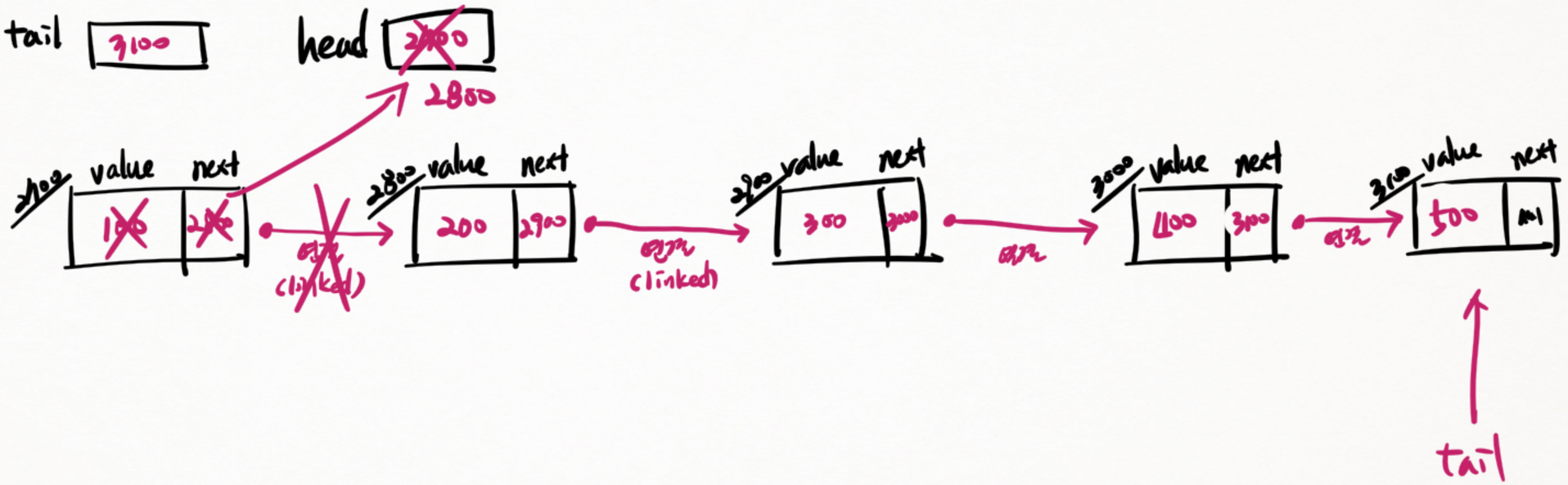


LinkedList — remove() : 끝 항목 삭제



LinkedList - remove() : 시작 노드

remove(100)



prev

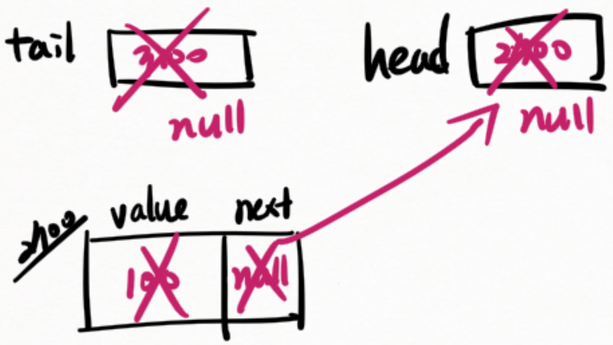
null

cursor

2700

LinkedList - remove() : 시작 노드 + 끝 노드

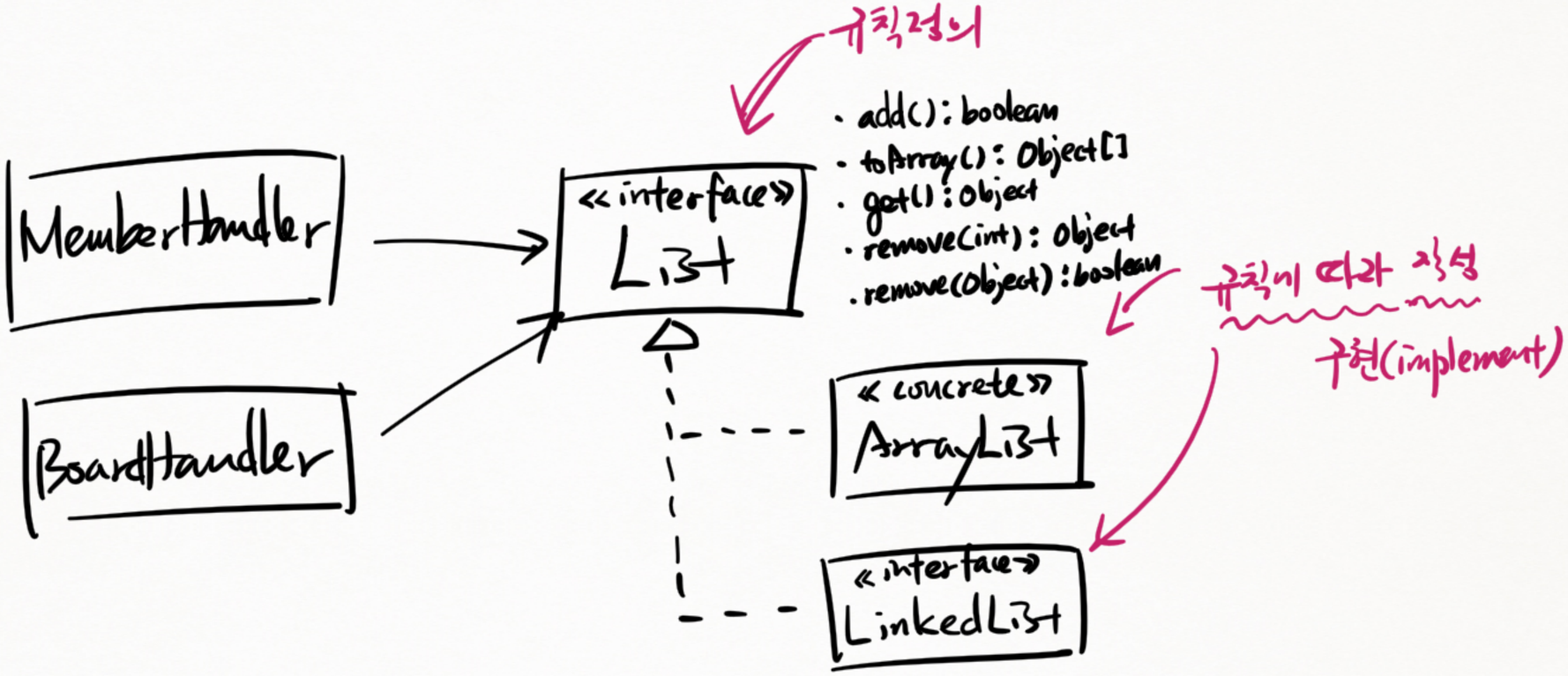
remove(100)



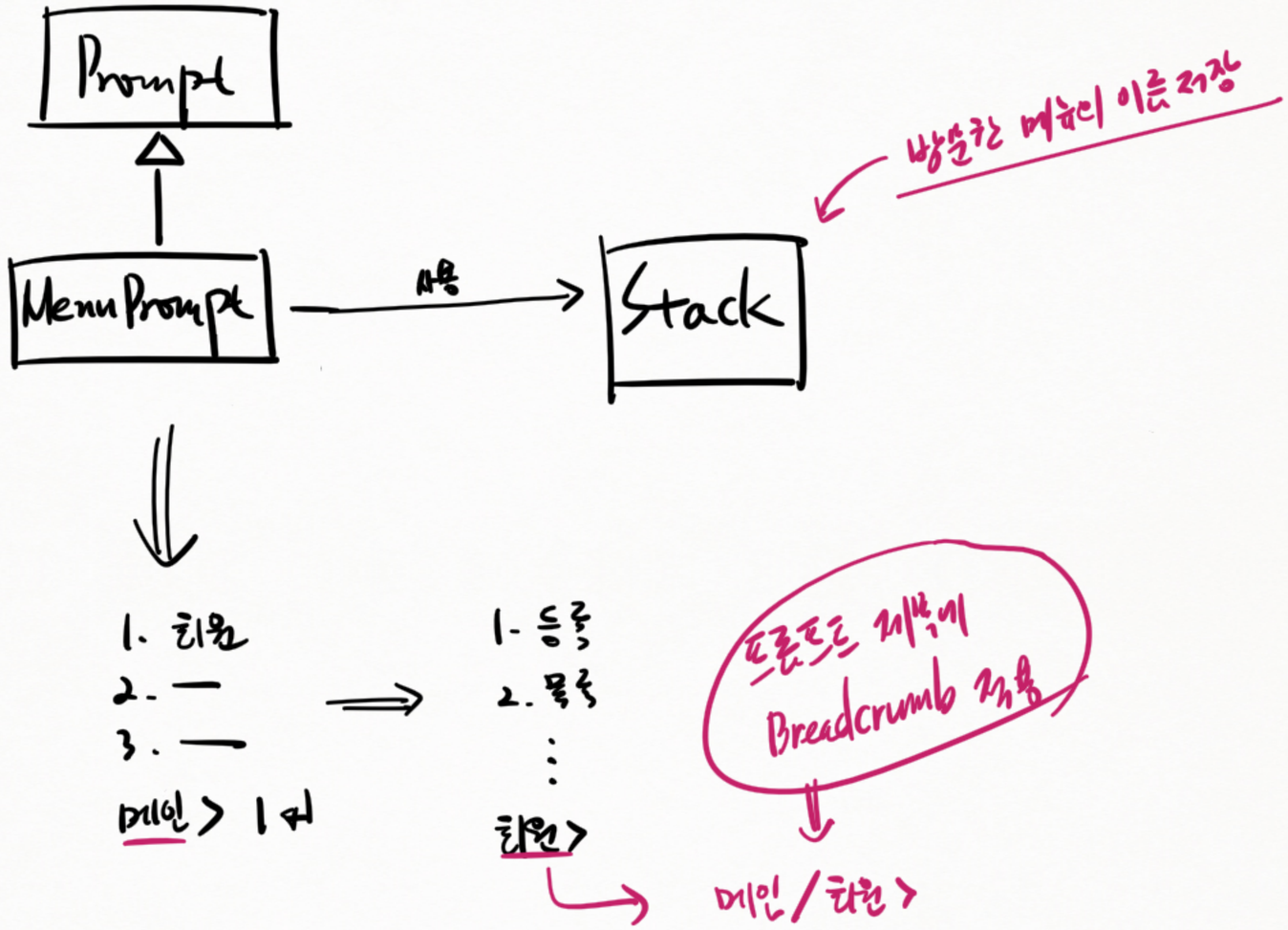
prev

cursor

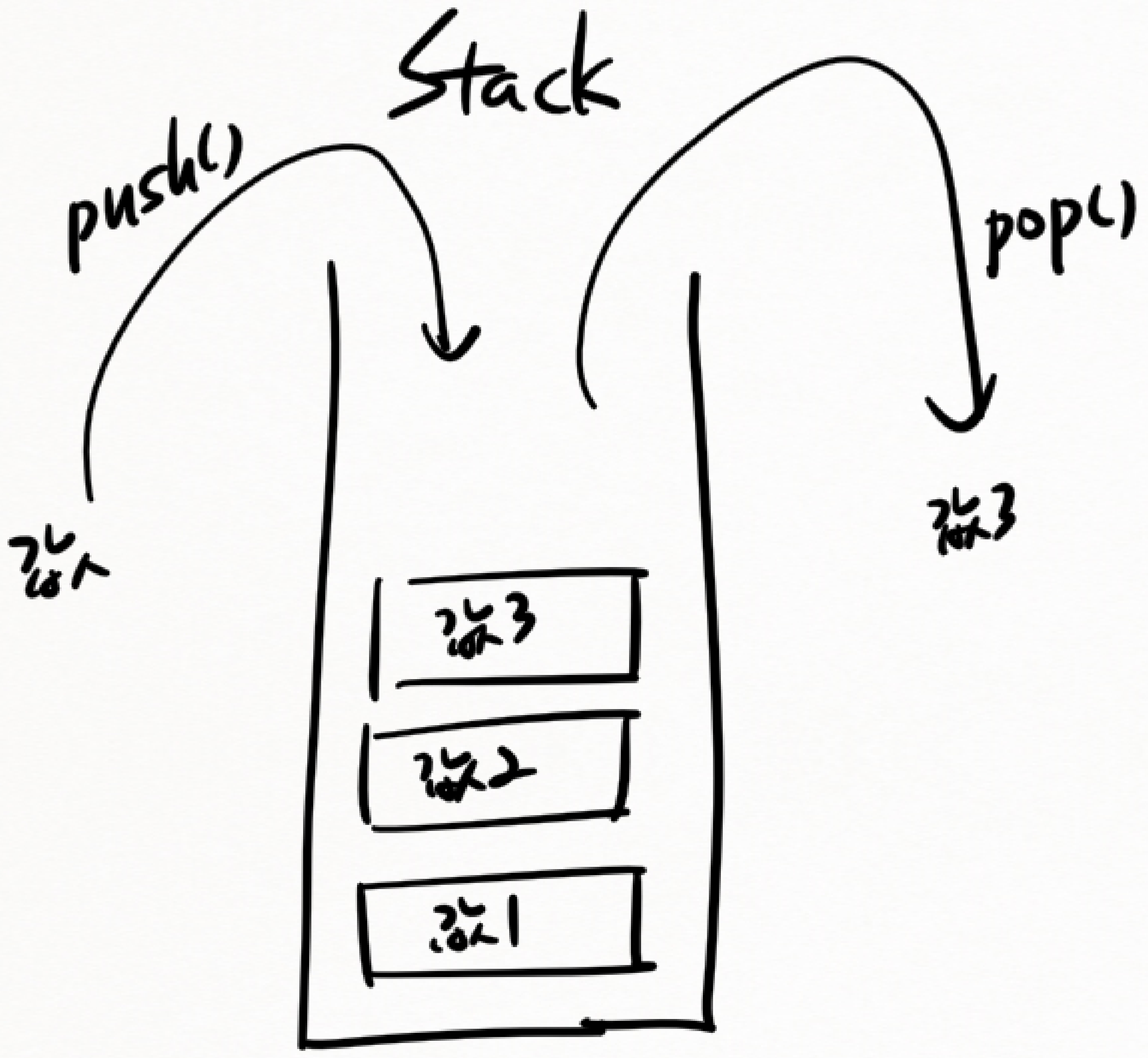
21. 인터페이스로 목록관리 객체의 사용법을 규칙으로 정의.



22. Stack, Queue 자료구조 구현



* Stack 구동원리



Last In First Out
(LIFO)

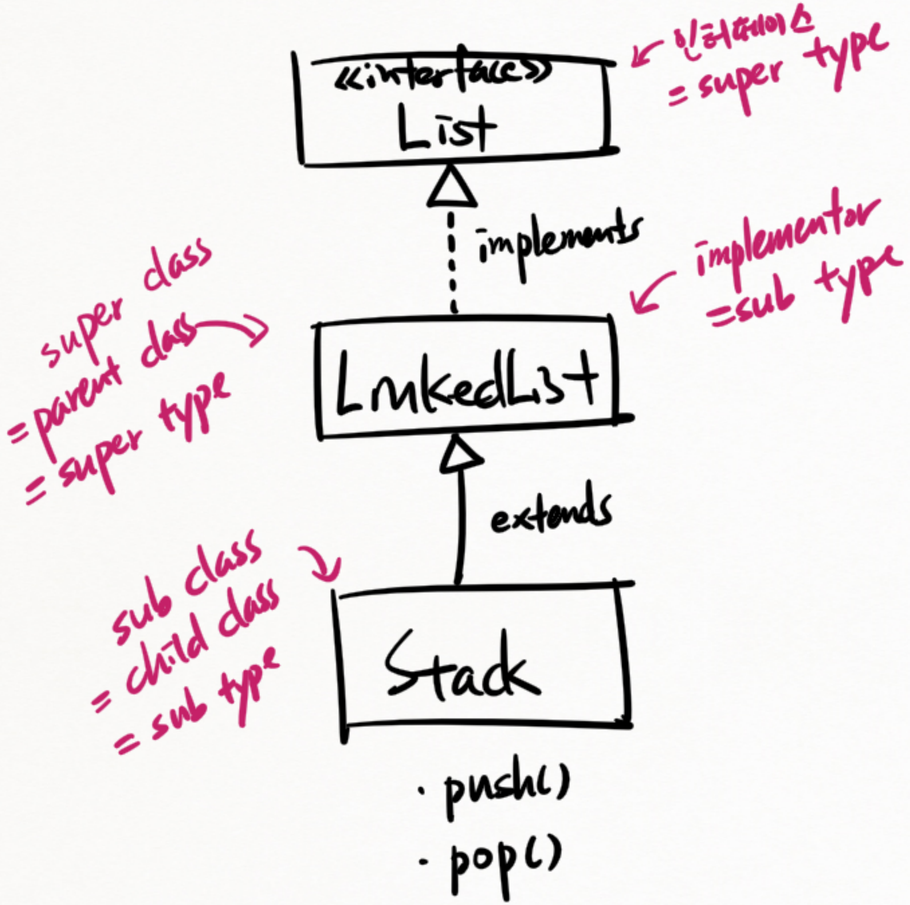
breadcrumb 구현에 적용할 수 있다

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

뒤로 뒤로 뒤로 뒤로

- 방문한 메뉴이름 저장
- 〃 폴더이름 저장
- 웹페이지 저장

* Stack 구현




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

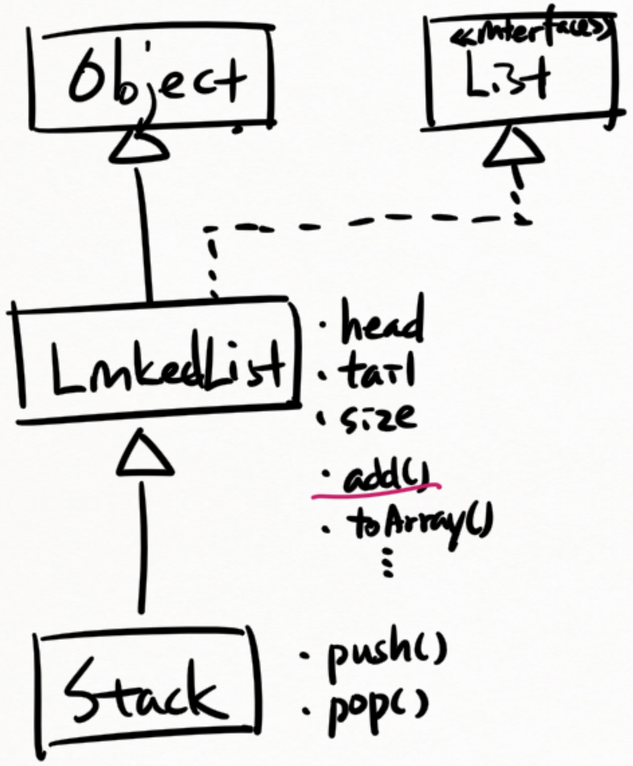
직접적으로 List 인터페이스 구현

List 인터페이스를 간접적으로 구현.

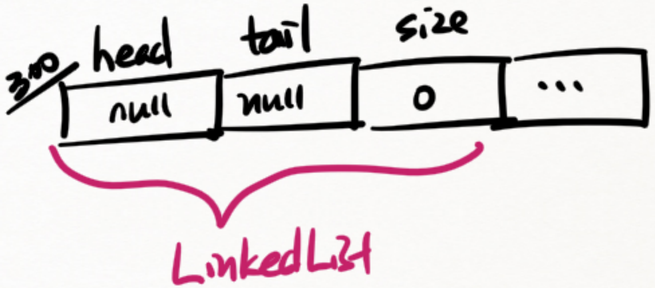
sub 클래스로 super 클래스가 구현한 메서드를 사용할 수 있다 //

부모클래스가 인터페이스를 구현했으면 서브클래스도 구현한 것으로 간주한다.

* Stack 인스턴스

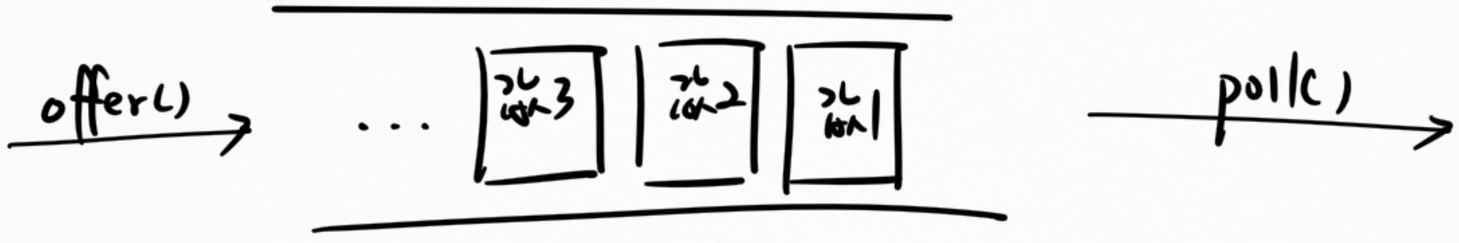



```
Stack s = new Stack();
```




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

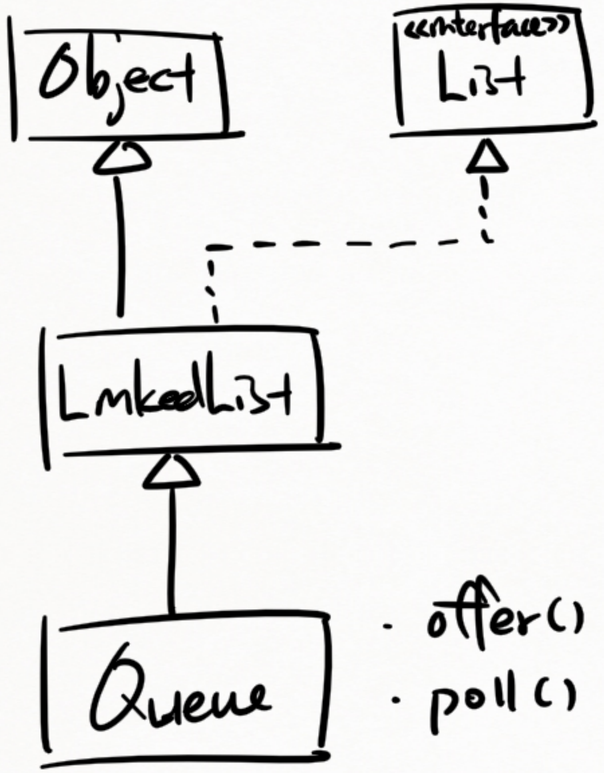
* Queue 구동 원리

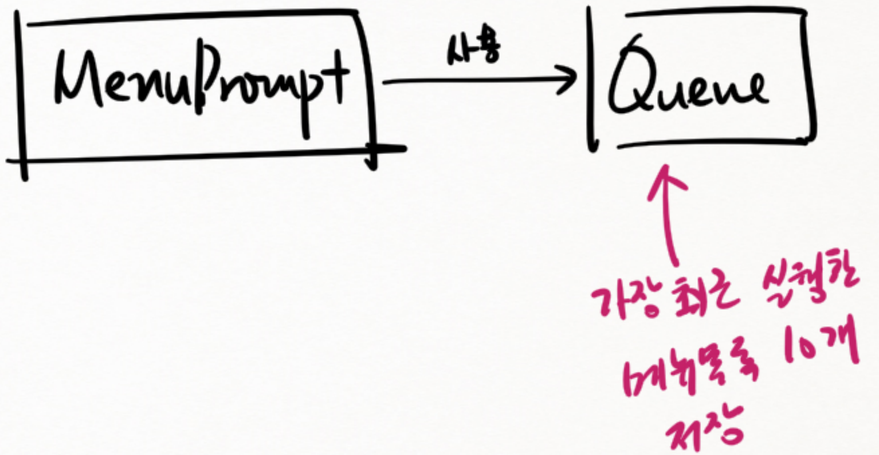


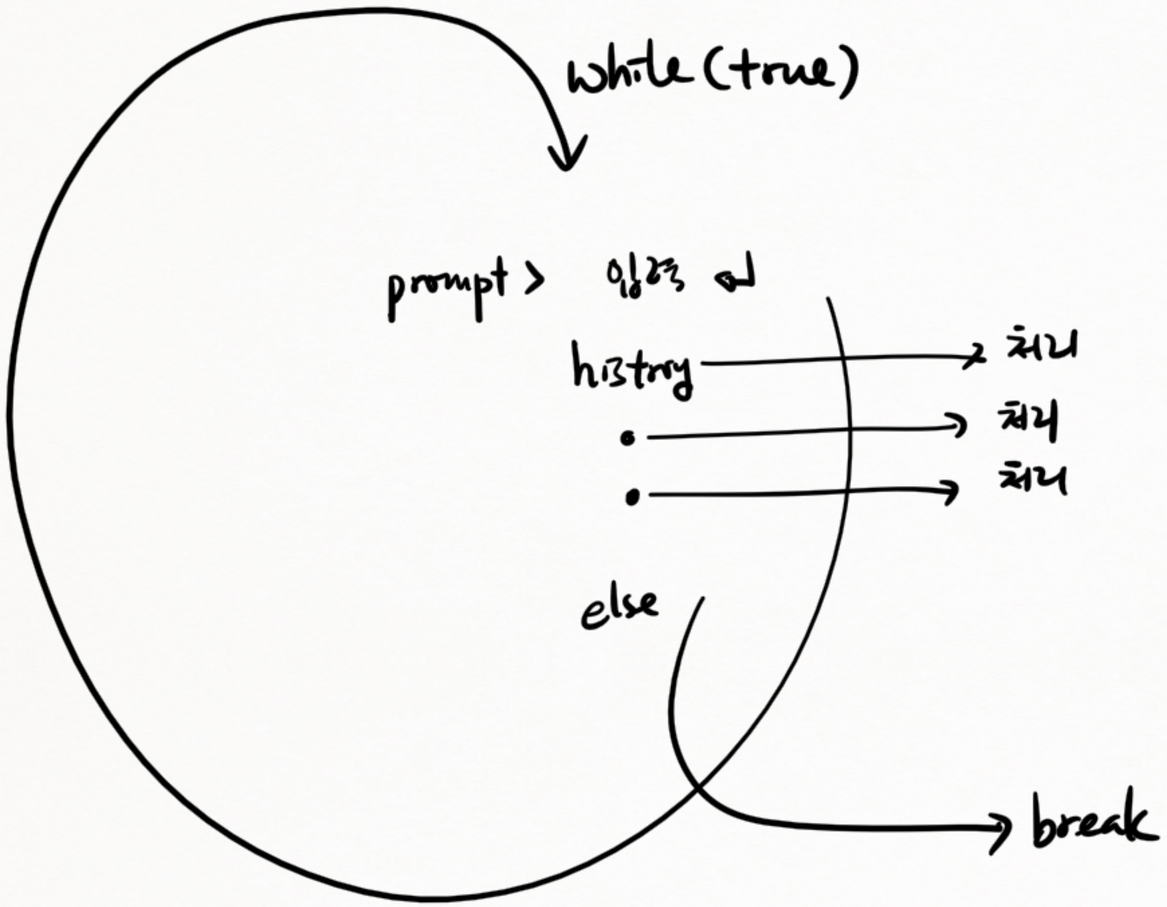
First In First Out
(FIFO)

⟹ · 명령어 입력 history 기능에 사용
· (예약어) 받을 때

* Queue 구현과 활용


Object
<<interface>>
List
LinkedList
Queue
· offer()
· poll()

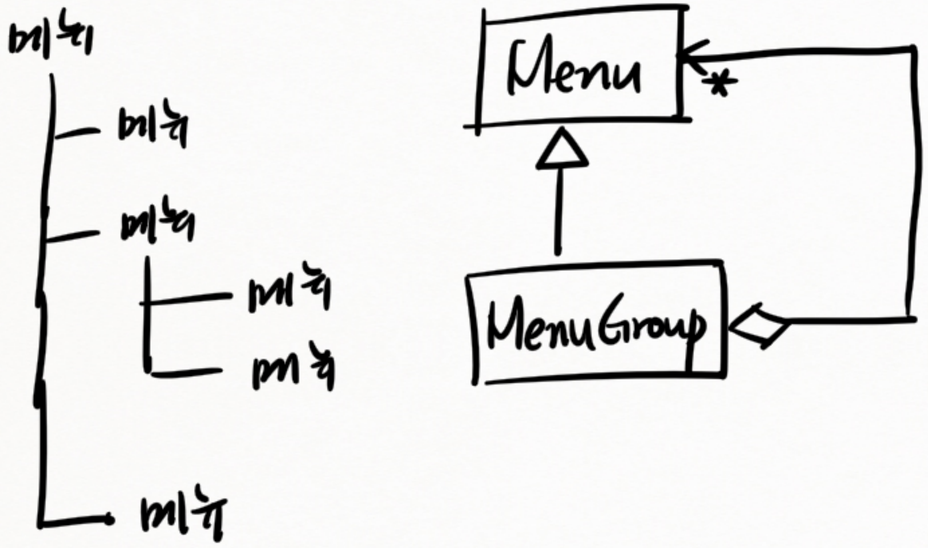

MenuPrompt
사용
Queue
가장 최근 실행한
10개
저장


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

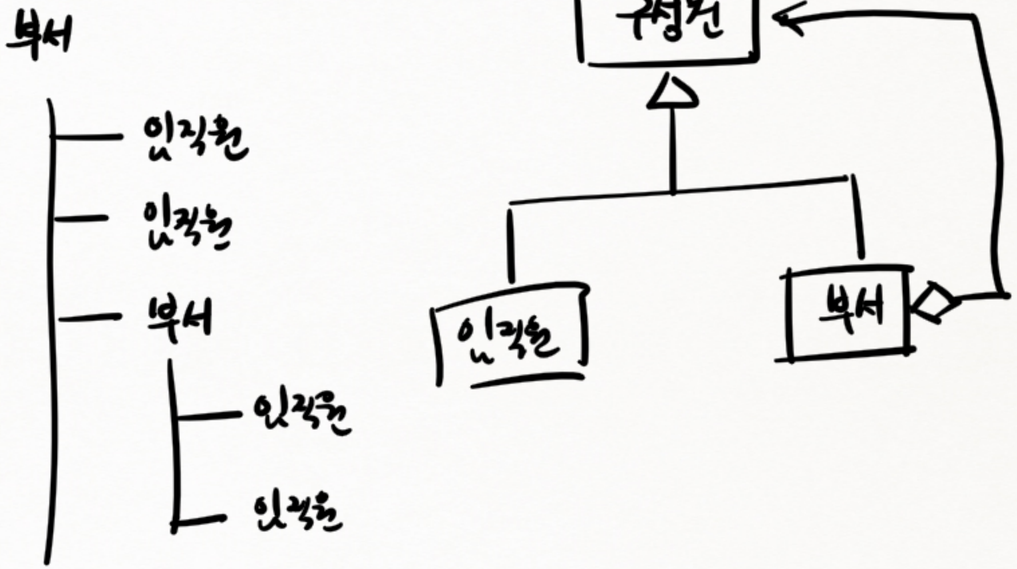
23. Composite, Command, Observer 패턴 활용

① Composite 패턴

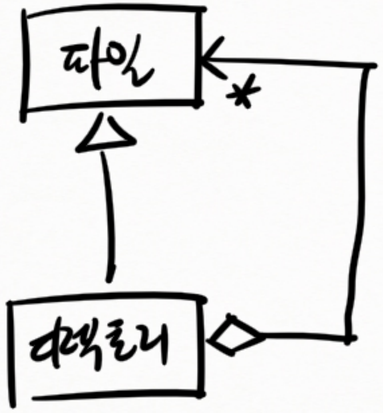
활용예)



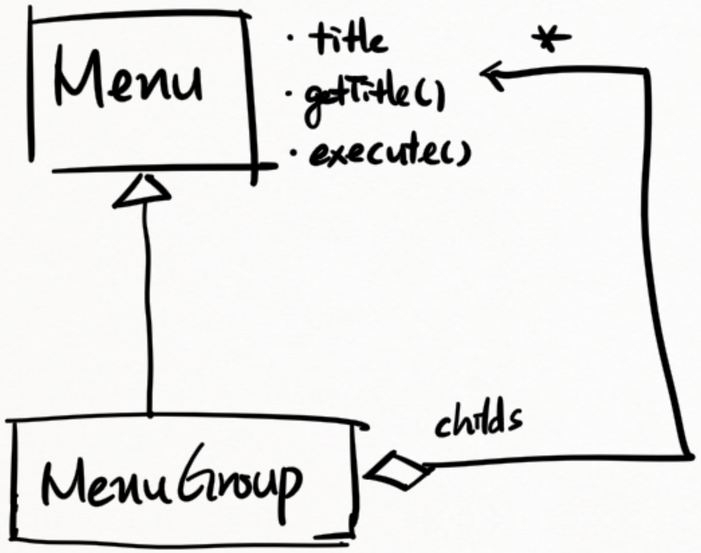
활용예)



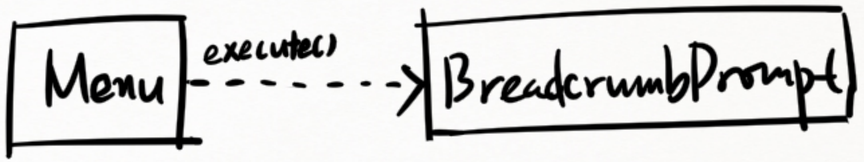
활용예)



Composite 패턴 구현



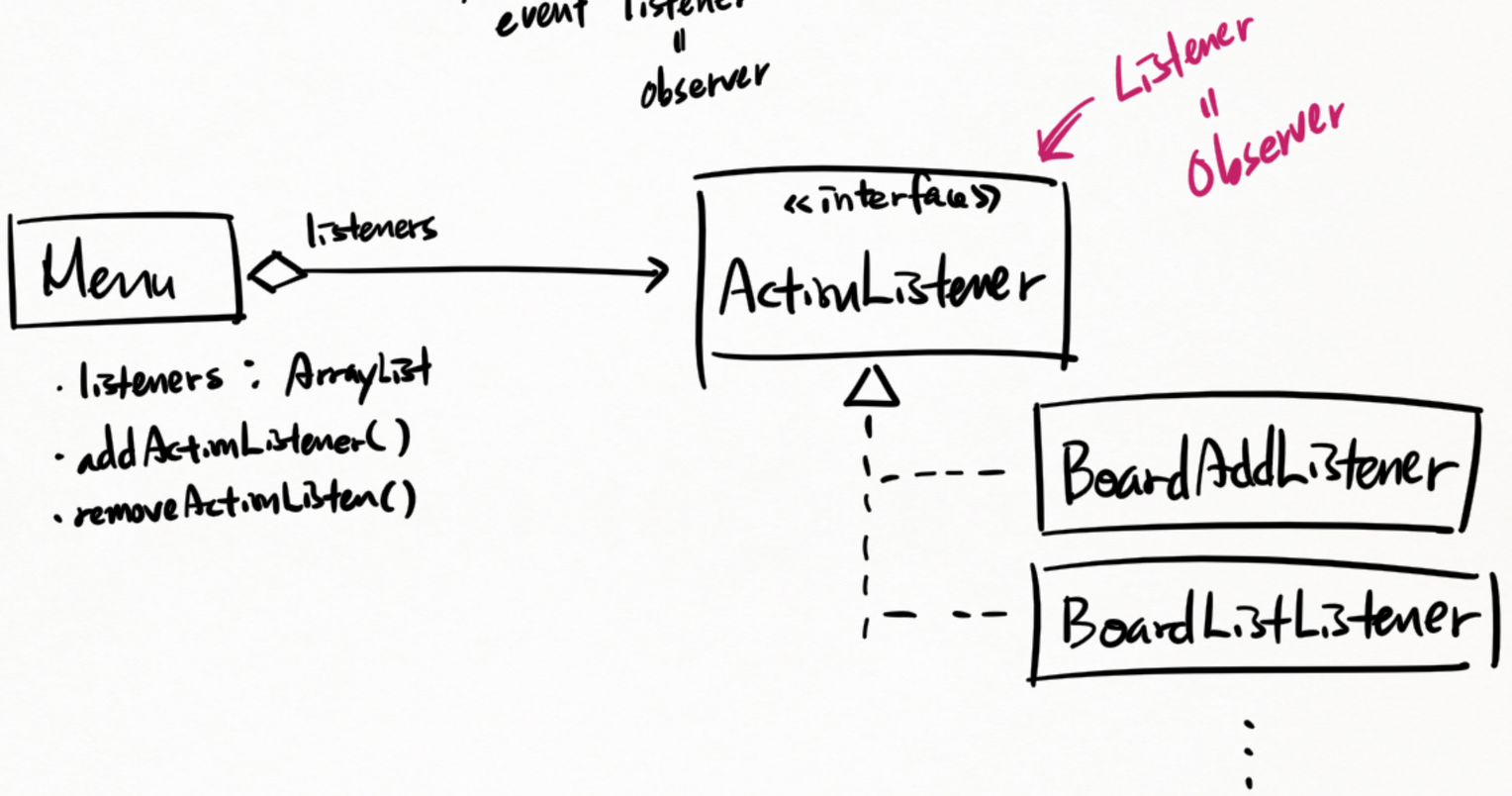
- childs : ArrayList
- addMenu()
- execute() ⇐ Overriding



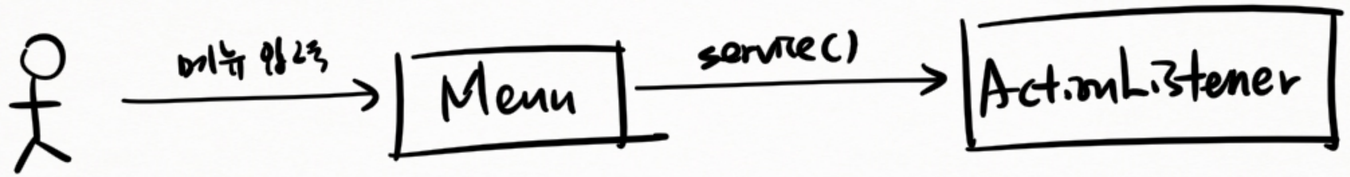
* Observer 패턴

↳ 객체의 상태가 변경 됐을 때 보내받아서 처리하는 것

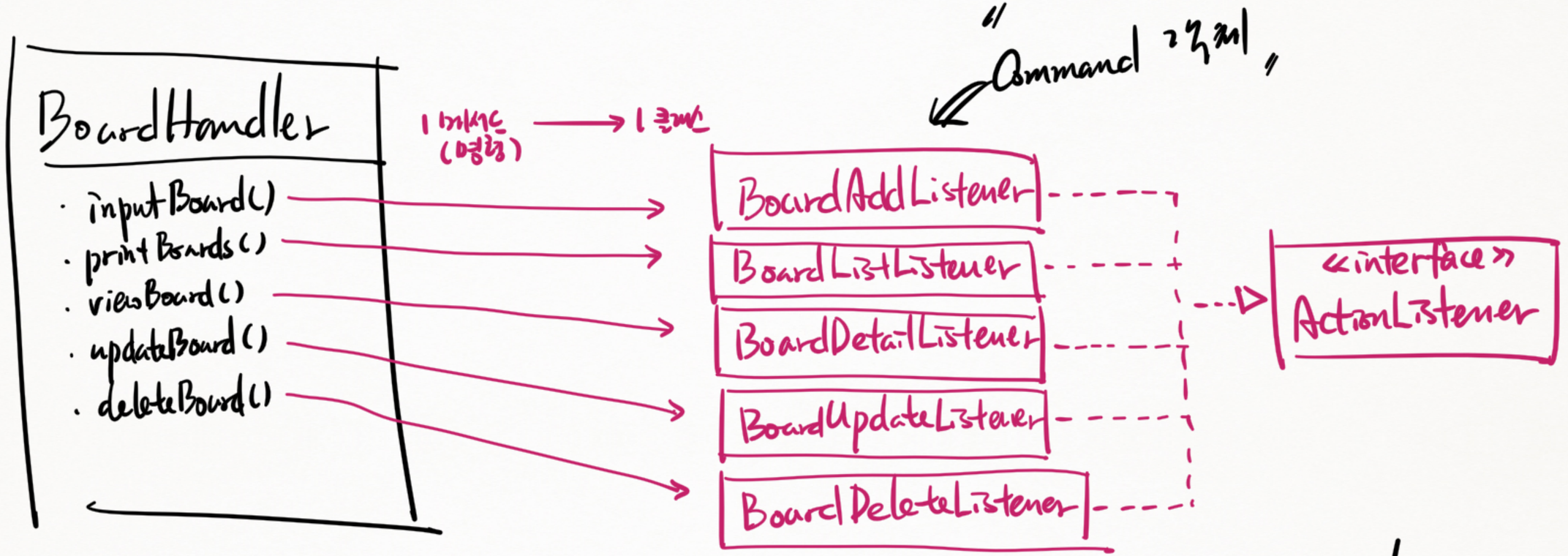
" event listener "
=
observer



* Observer 패턴의 동작 원리
  =
  Listener

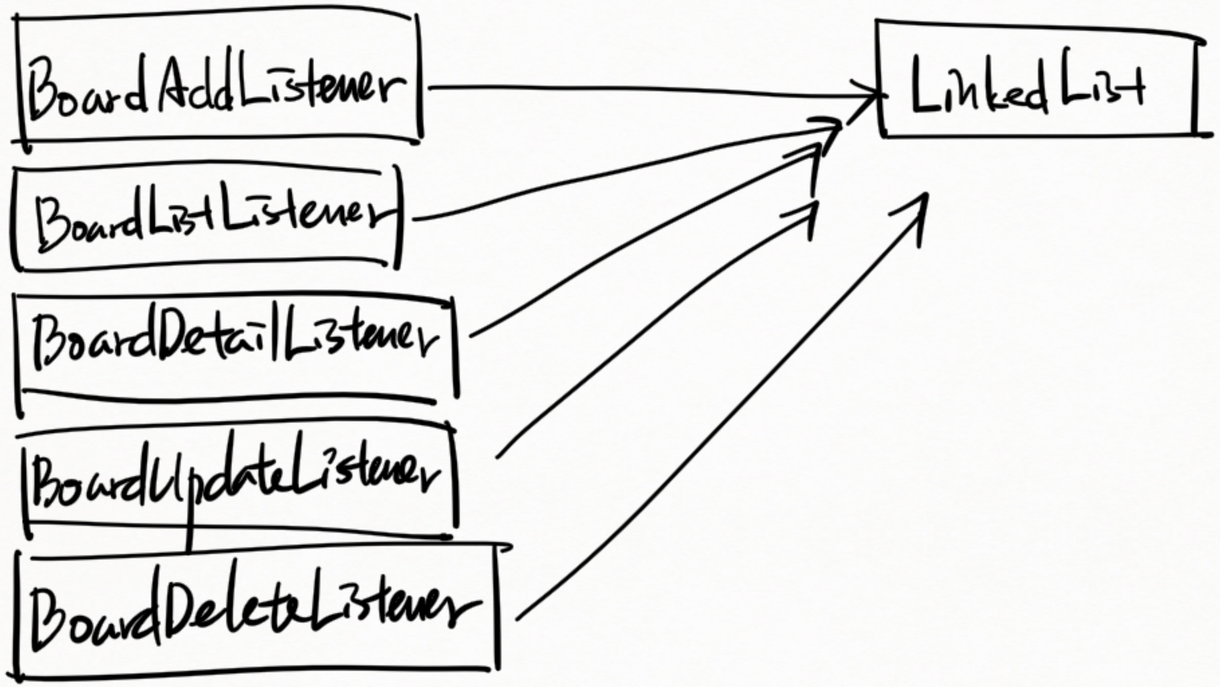


* Command 디자인



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

* 의존객체 방식


BoardAddListener
BoardListListener
BoardDetailListener
BoardUpdateListener
BoardDeleteListener
LinkedList

* 상속 : Specialization vs Generalization