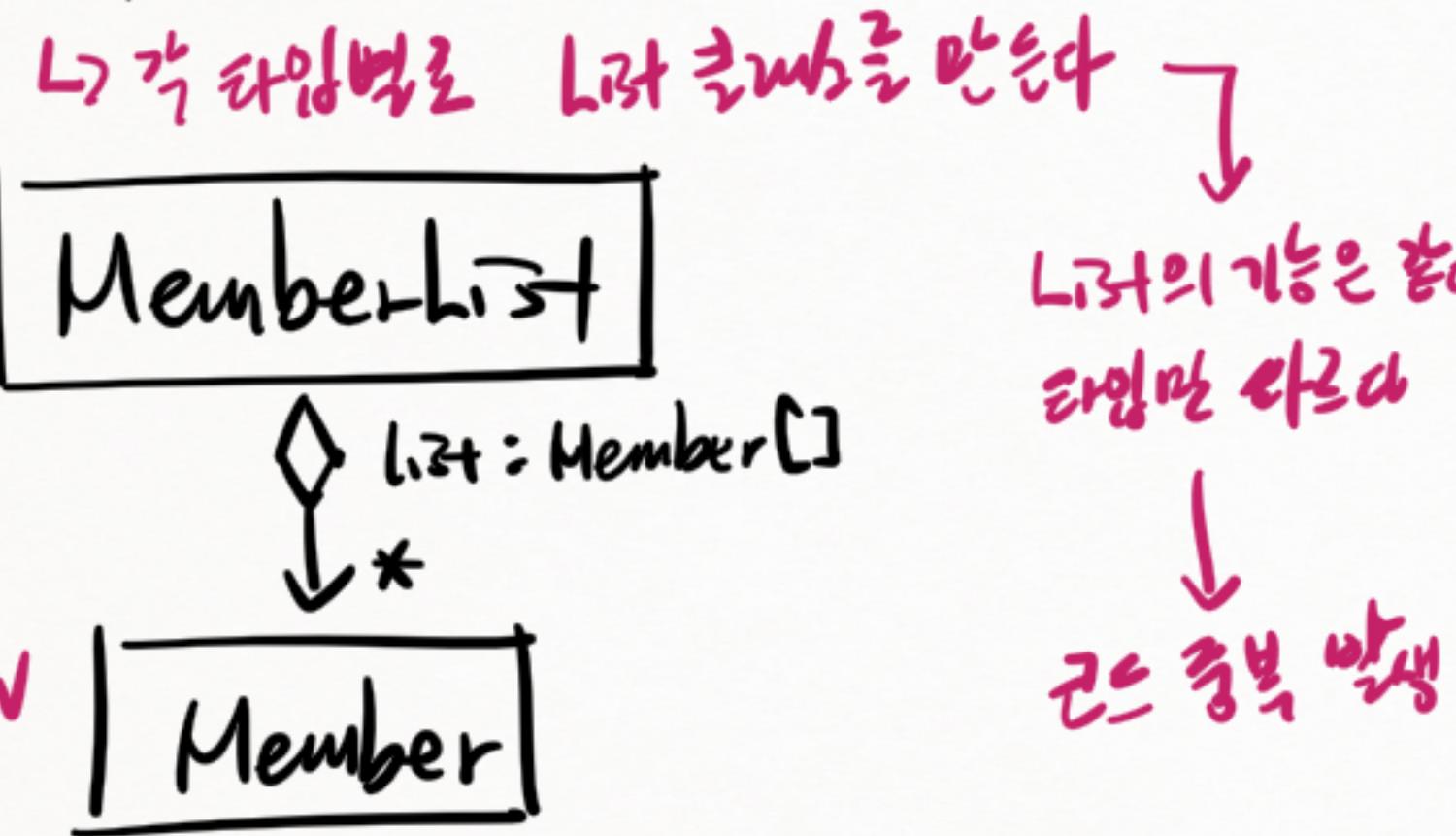


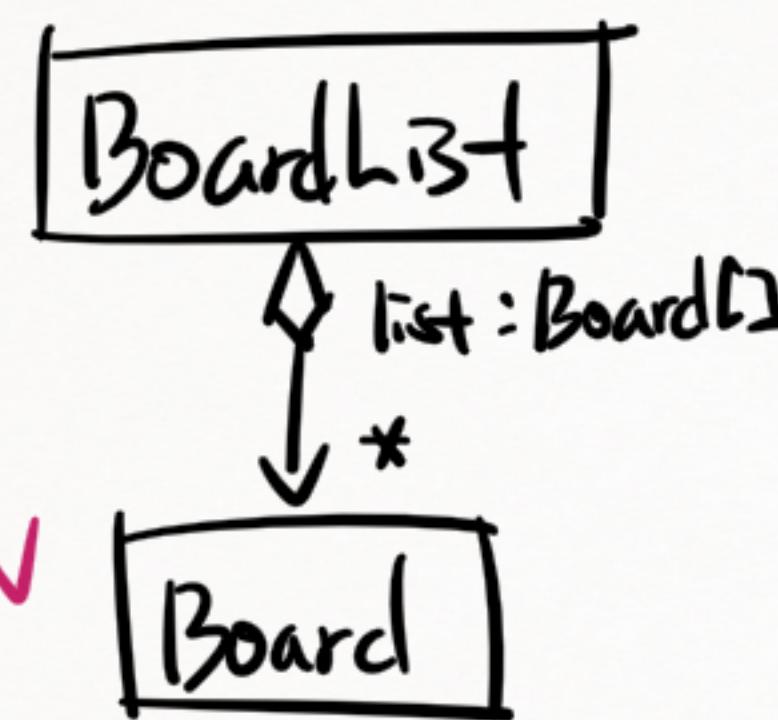
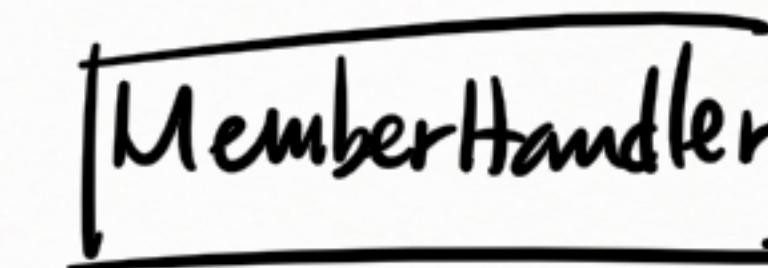
24. 제네리ك (Generic) 적용 : (object 타입처럼 다양한 타입에 대응할 수 있다.)
특정타입으로 제한할 수 있다.

① 다형적 변수 적용 전



② 다형적 변수 적용

→ 미처 각 타입별로 클래스를 정의한 듯한
효과를 볼 수 있다.



* 단점 :

- ① 인스턴스를 만들어야 하다
형변환 필요!
- ② 특정타입별로 다수로
제한할 수 있다.

← 타입별로
List를 만들 필요가
없다!
← 타입 안정성을 해친다.

. Member
. Board
:
: {
} 다양한 타입의
인스턴스(주)를
생성할 수 있다.

* Generic \Rightarrow Type Parameter

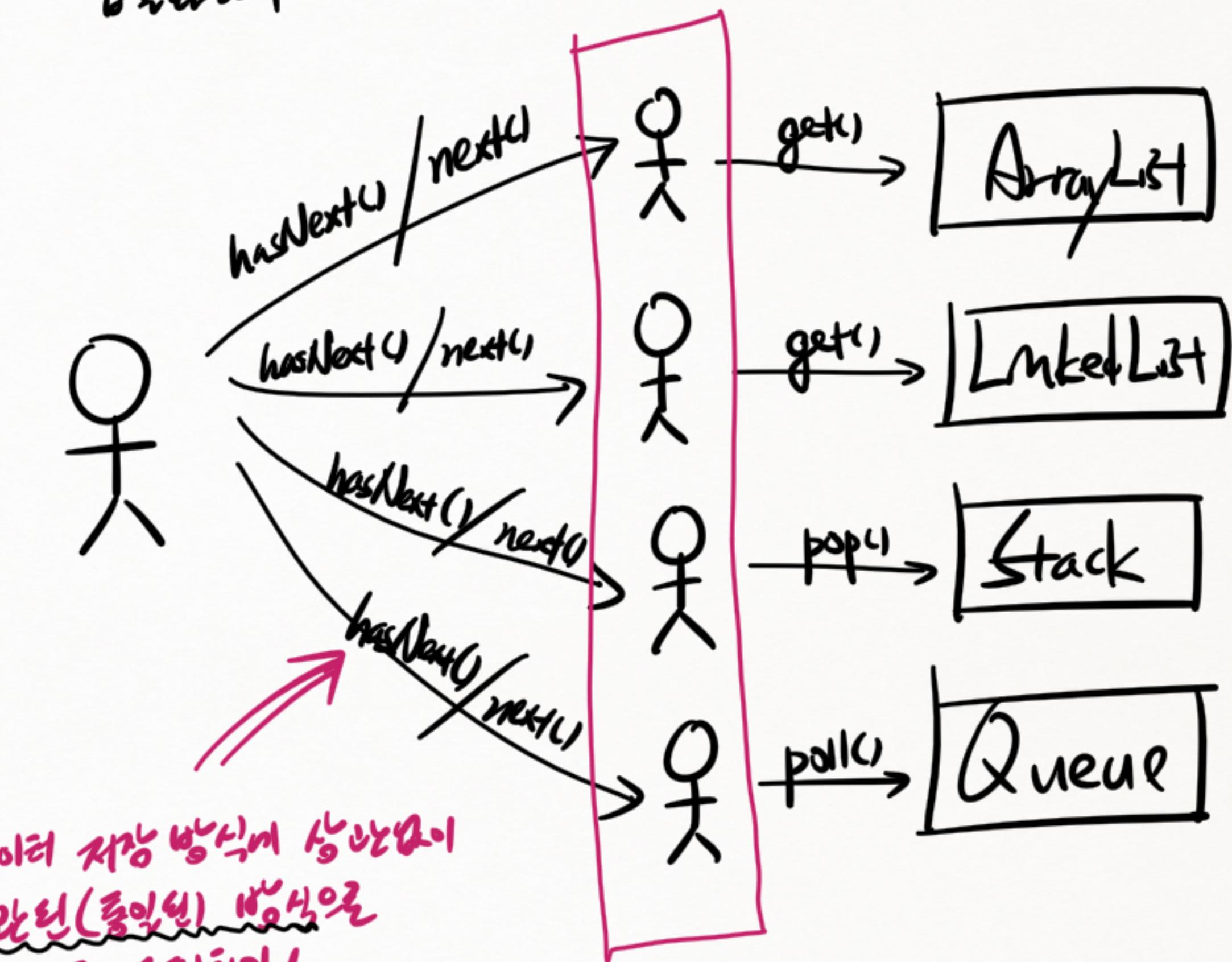
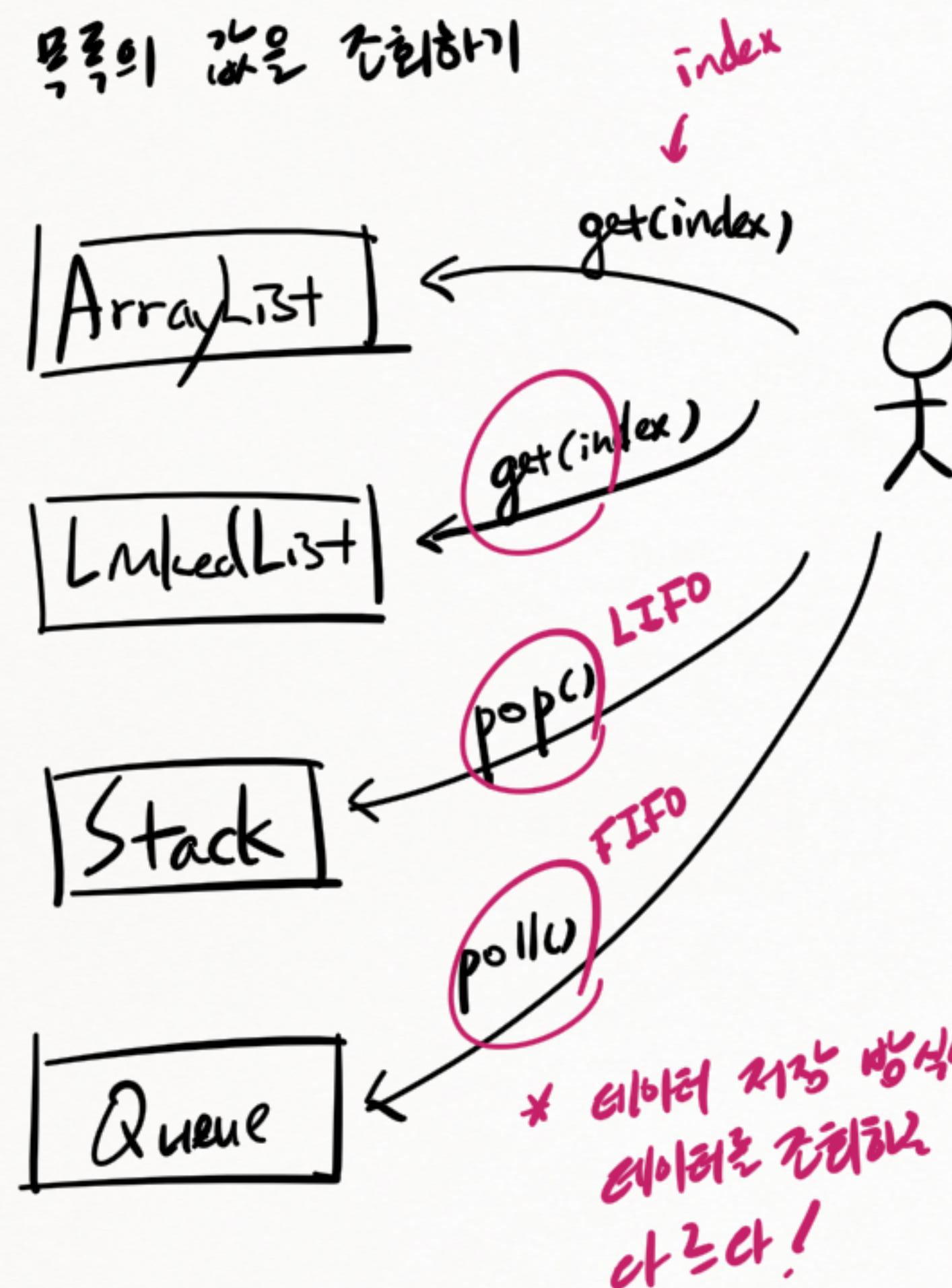
클래스의 타입 변수를 넣는다

```
class ArrayList<What> {  
    public void add(What obj) {  
        =  
    }  
    public What get(int index) {  
        =  
    }  
    :  
}
```

타입 이름을 넣을 변수 = "Type Parameter"

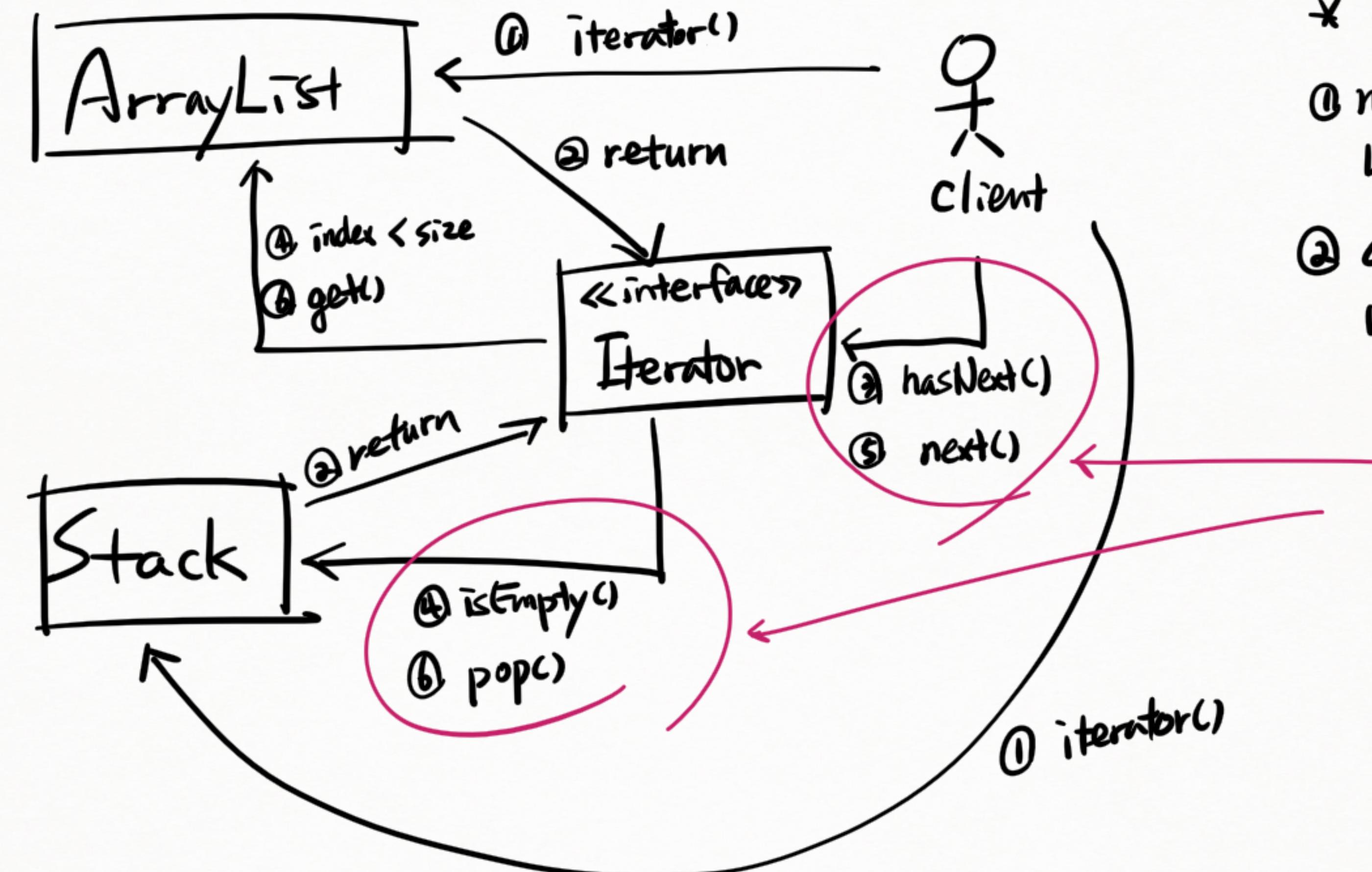
25. Iterator 대신을 활용하여 iterator 조작기능을 기존에 쓰울하기 →활용하기

① 목록의 값을 조회하기



Iterator
(데이터를 순회하는 일을 하는 객체)

* Iterator mechanism (구조원리)

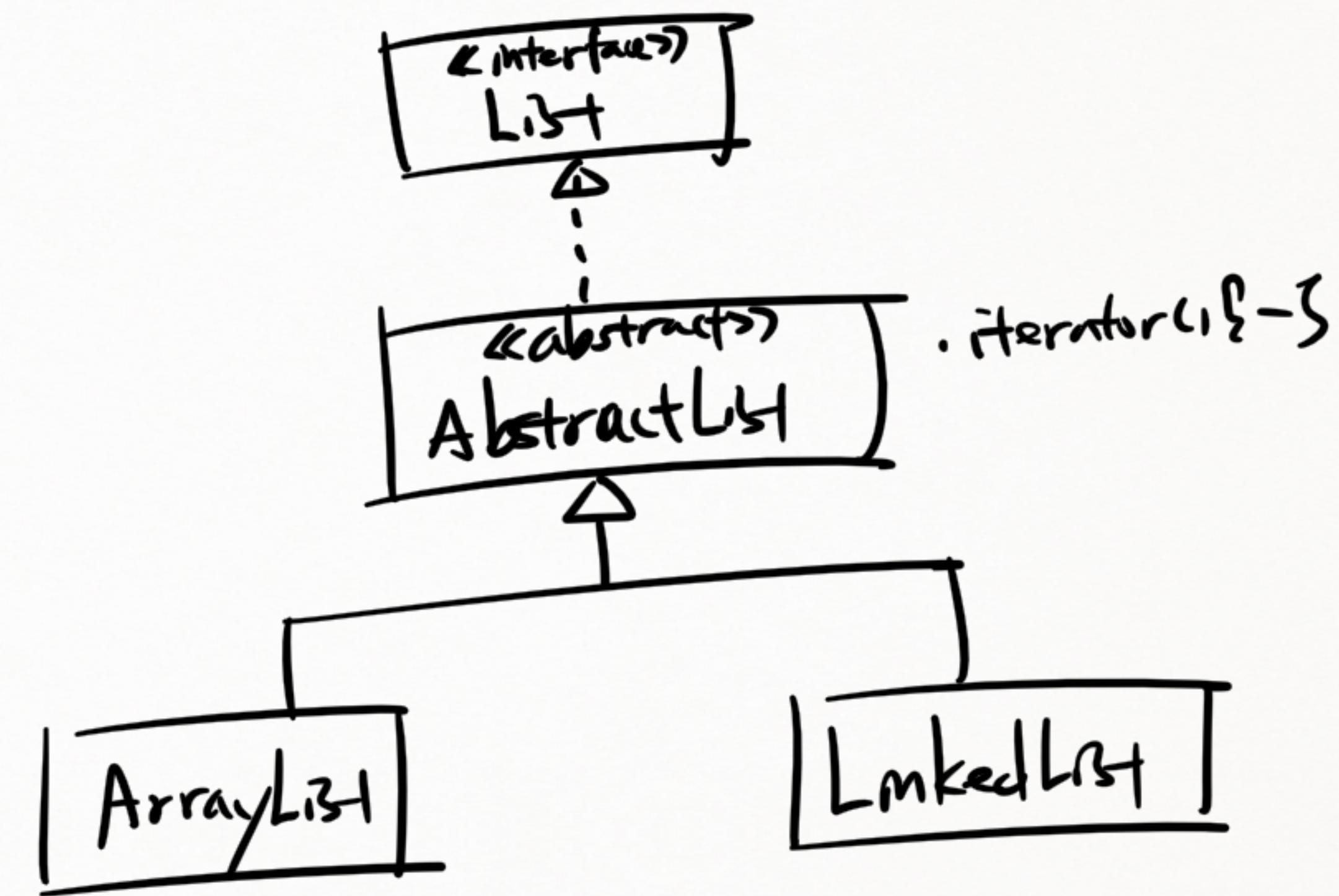
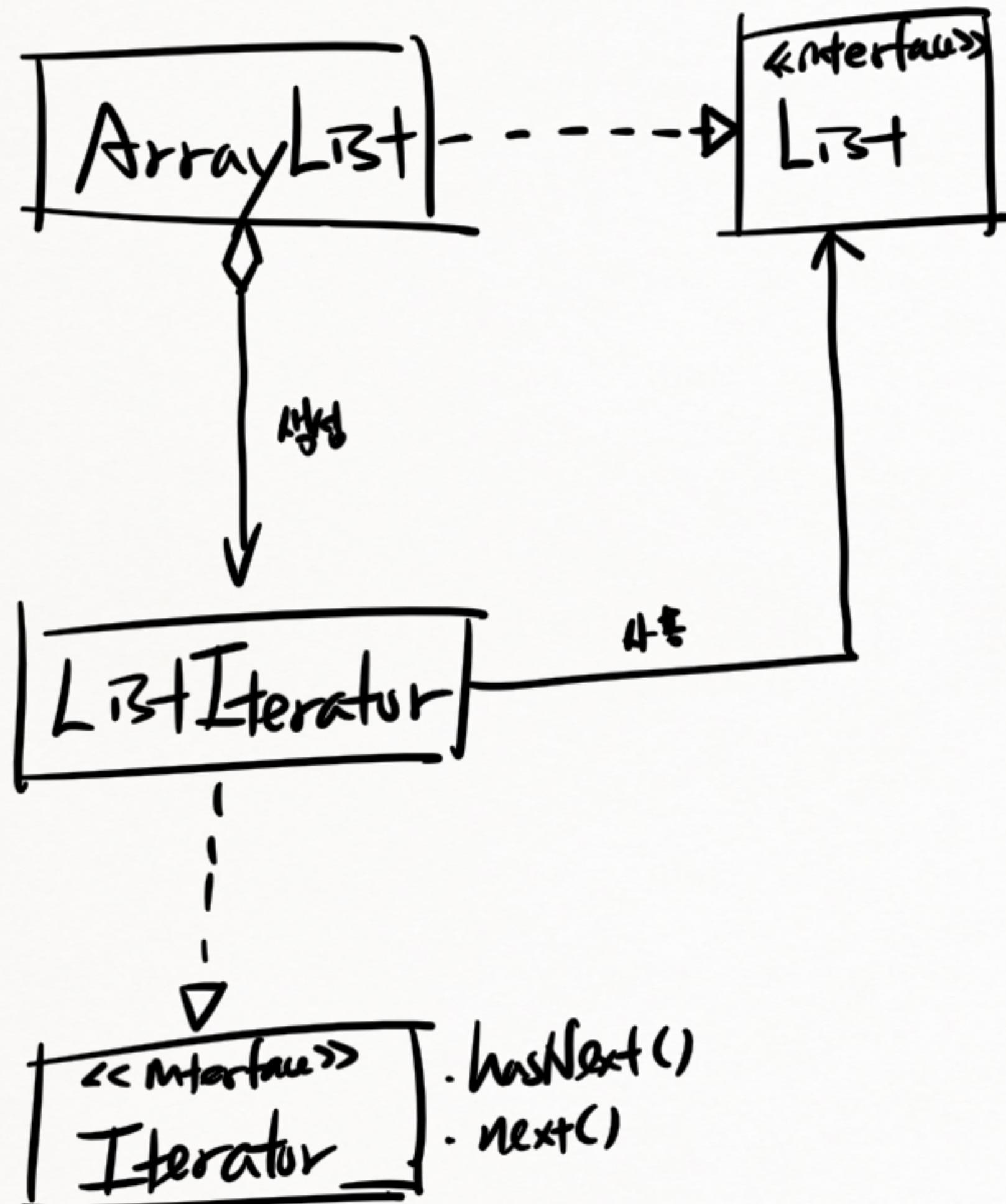


- * client
- ① network 분야
↳ 네트워크 요청으로 연결을 수행 S/W
- ② OOP 분야
↳ 다른 객체를 사용하는 개체.

제작업체가 상관없이
일관된 방식으로
작동을 가능케 한다!

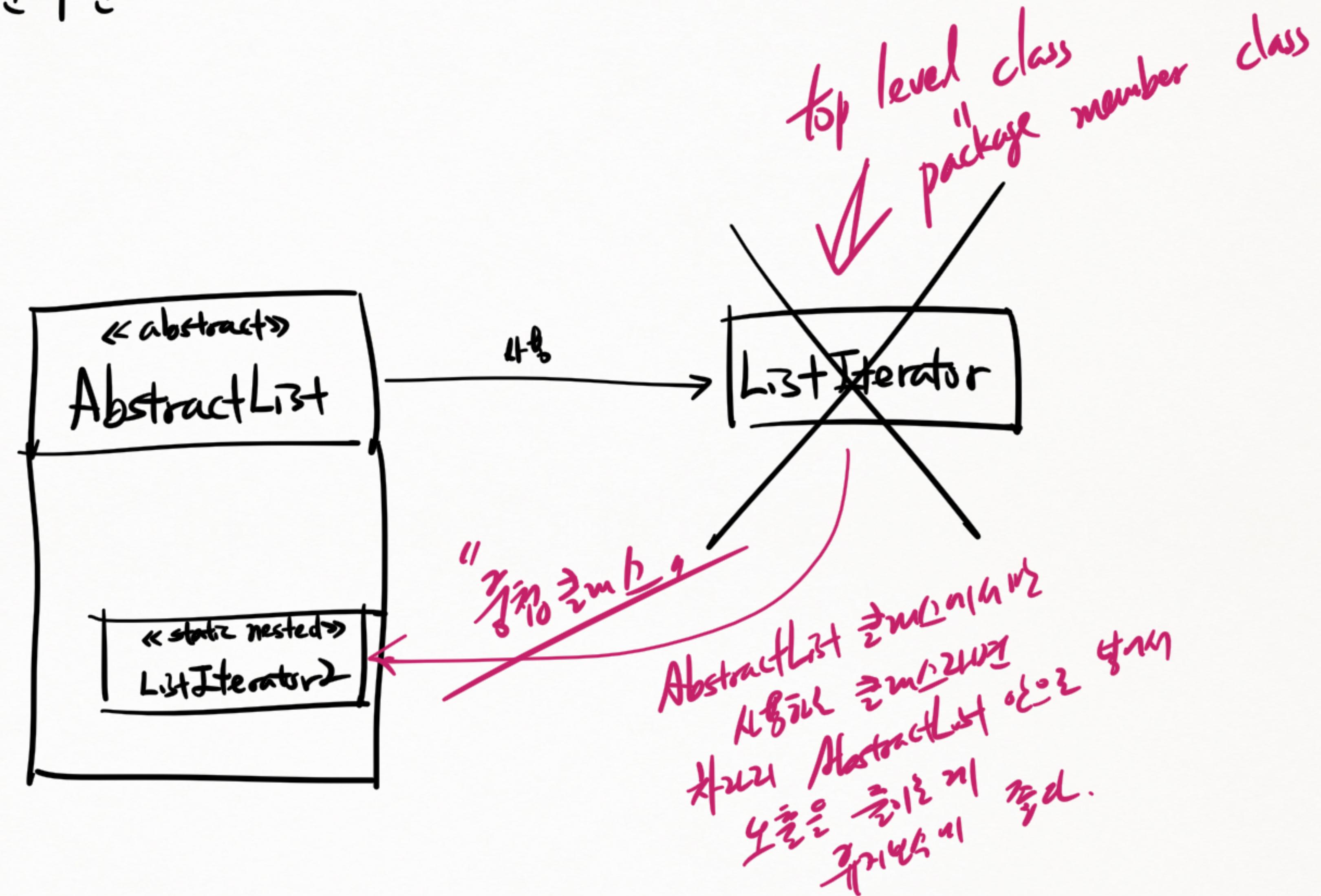
* Iterator 퀘션 구조

① 퀘션 - top level class



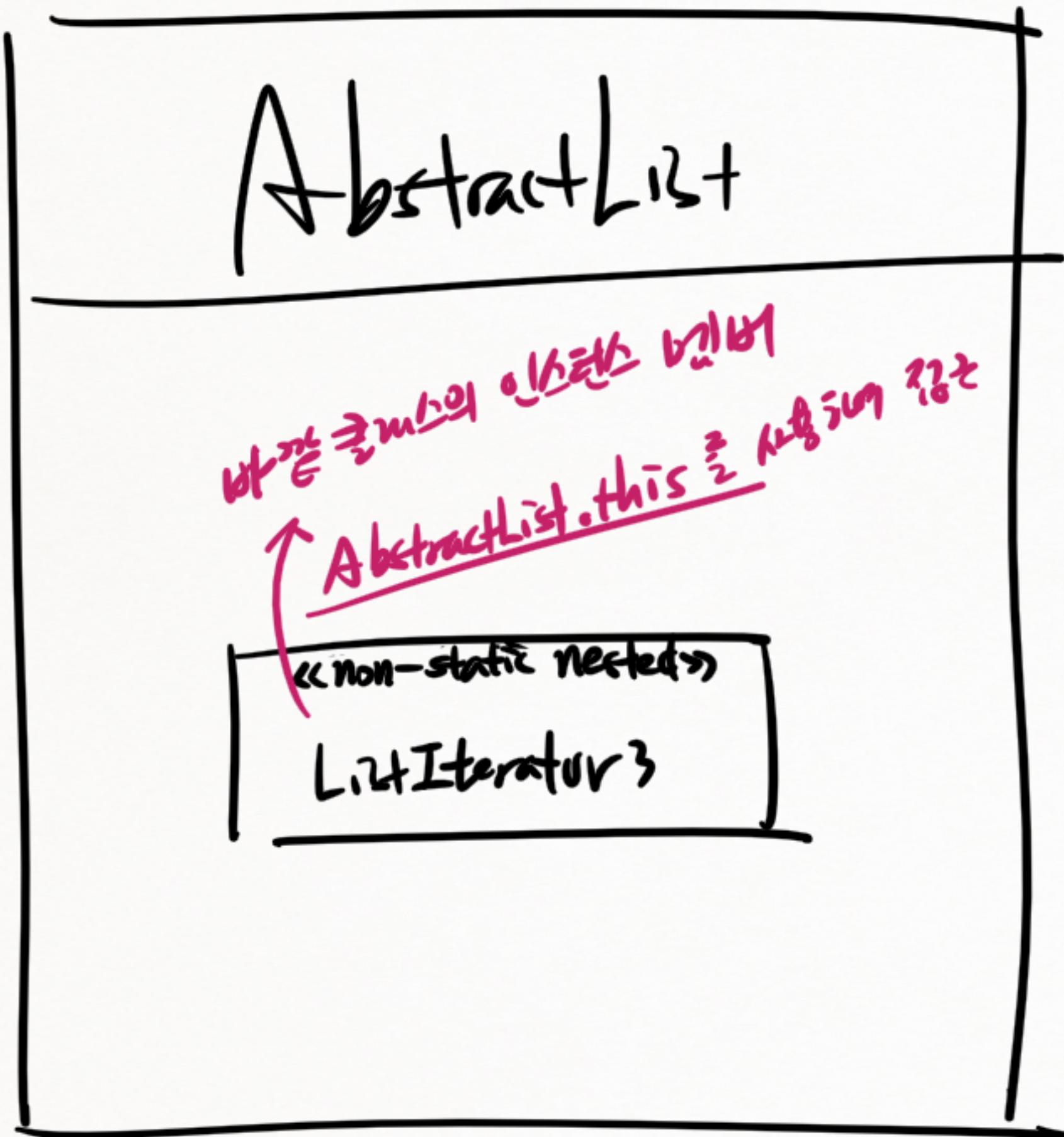
* Iterator 퀘션 구현

② 구현 — static nested class

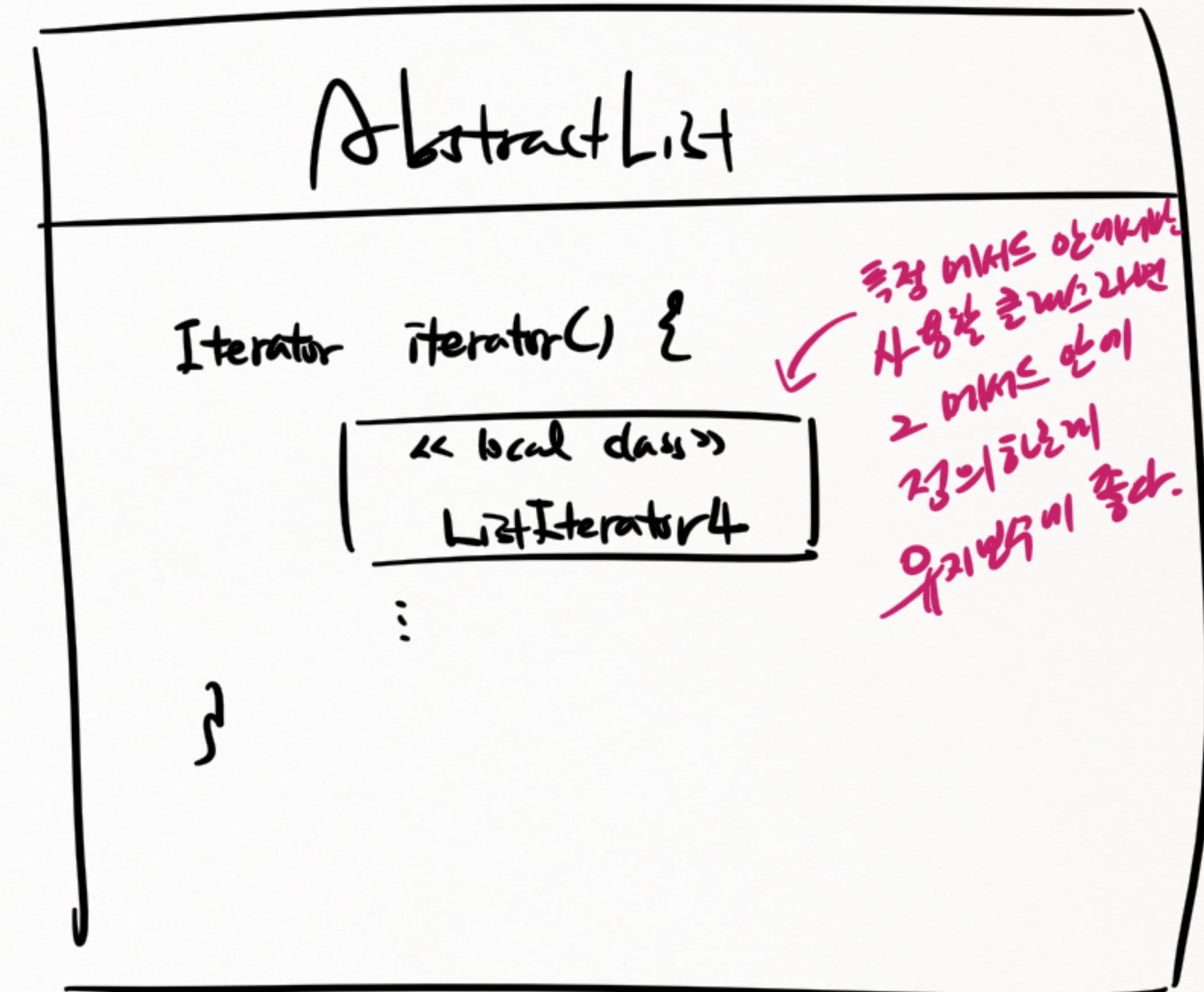


* Iterator 파편 구현

③ 구현 - non-static nested class

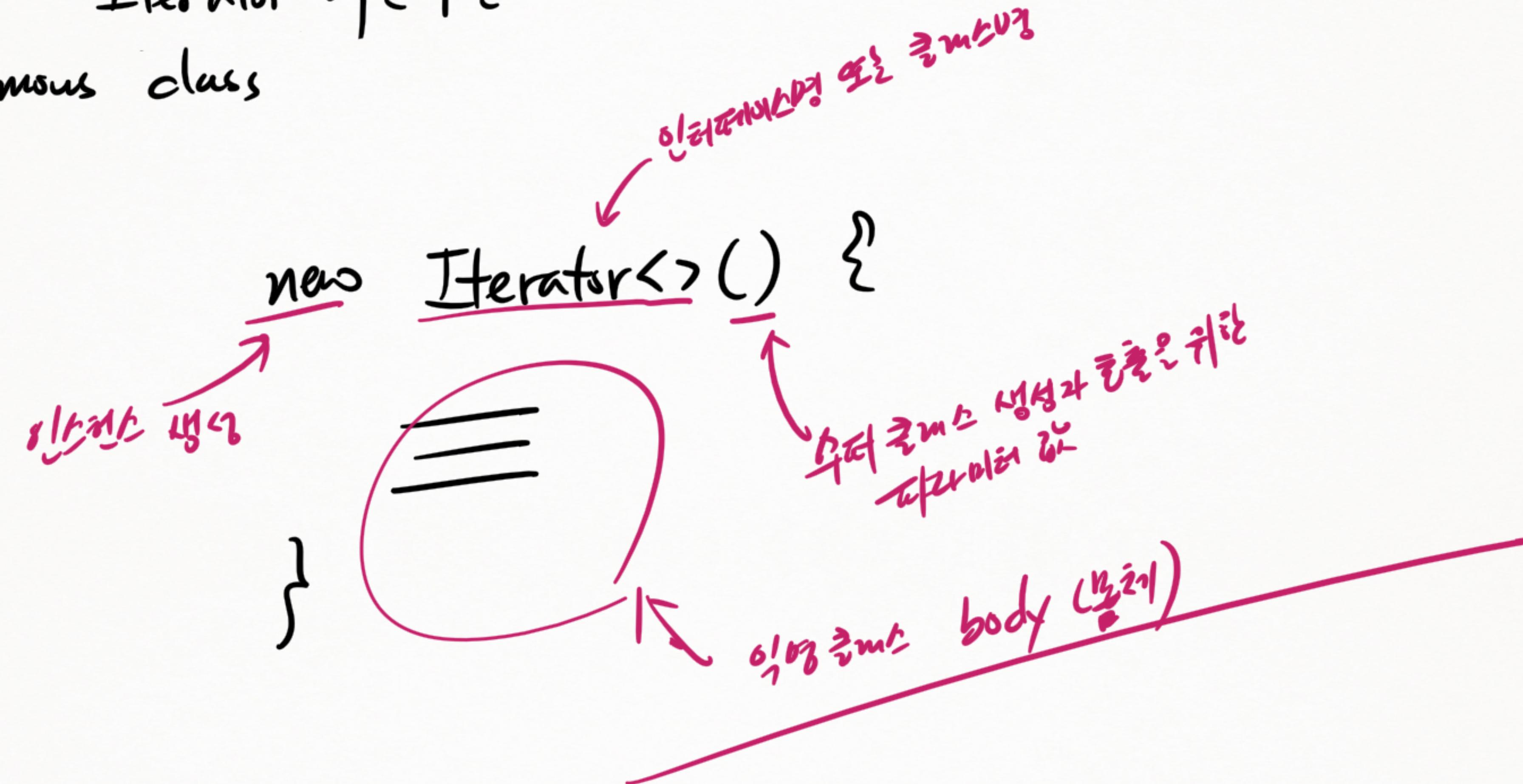


④ 구현 - local class



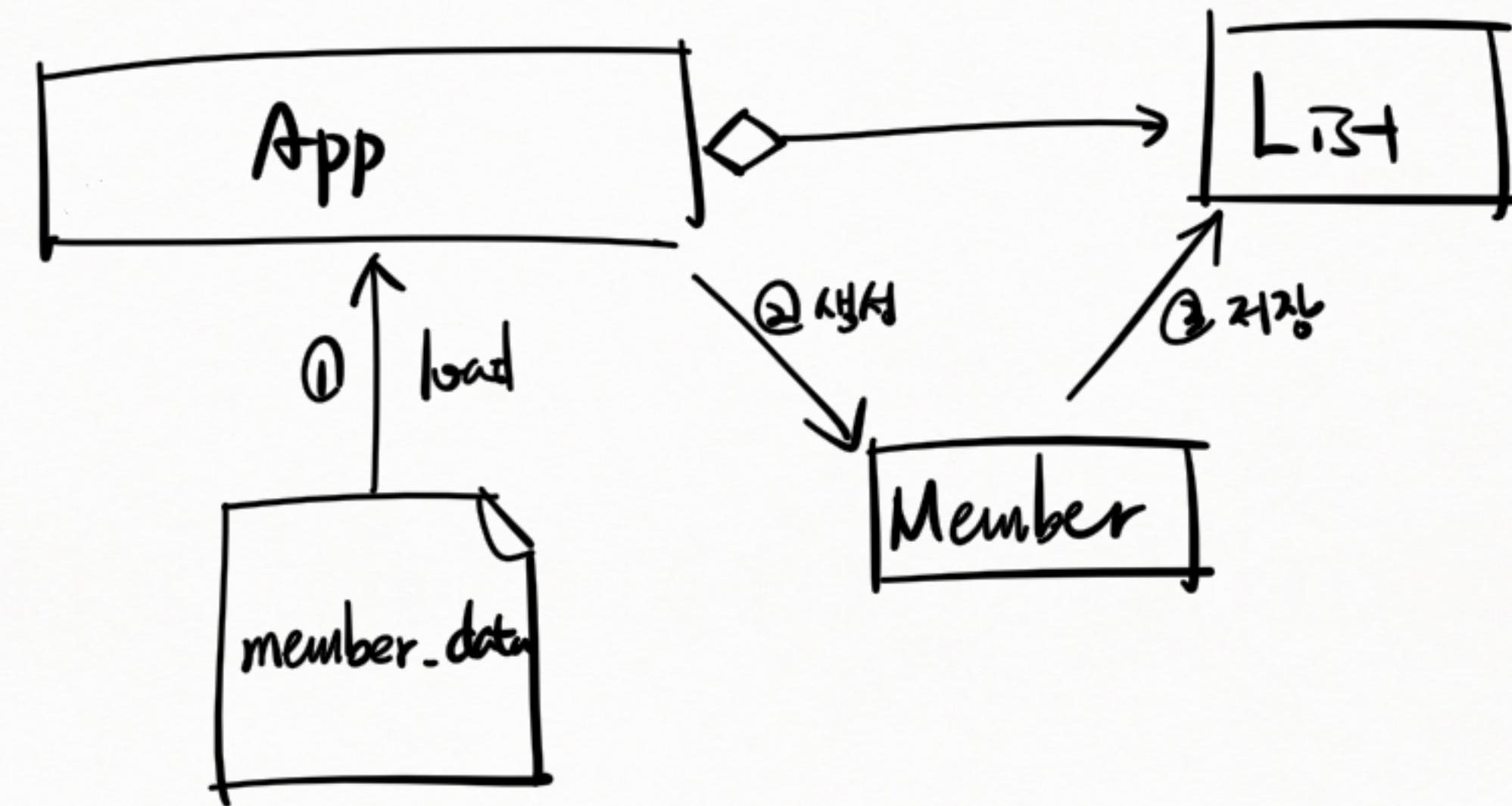
* Iterator 파편 구현

⑤ anonymous class



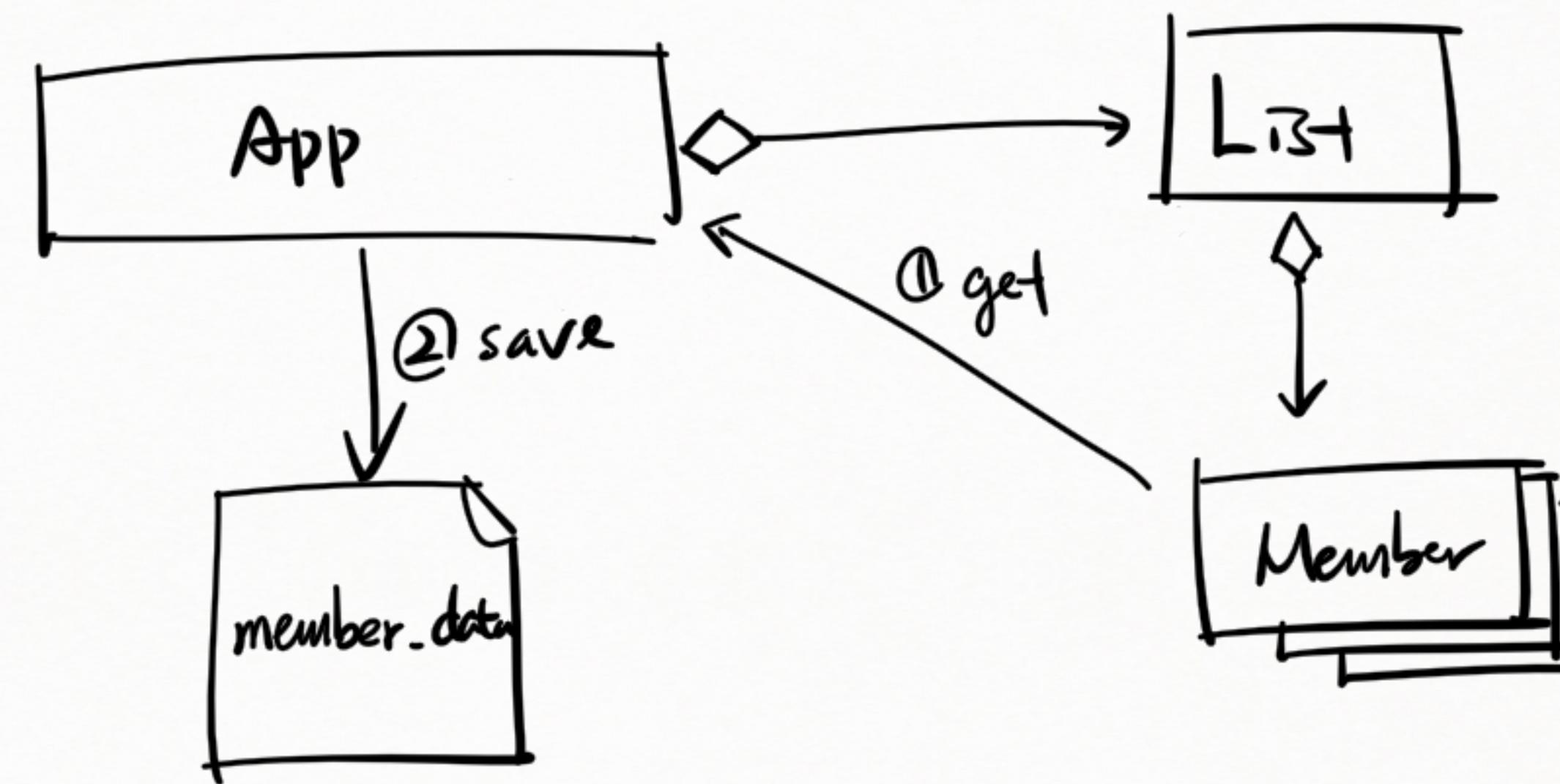
27. Data I/O Stream API - binary stream API 사용 ↳ Data 저장

① Data 읽기



27. Data I/O Stream API - binary stream API 사용 ↳ Data 저장

② Data 쓰기



```
int length;
```

100|00|00|03

00|03

in.read(4)

...|00|00|00| << 8

...|00|00|00|

in.read()

...|00|03|

: ...|00|00|00|
...|00|03|
00|00|00|03|

61|61|61|

in.read()

12345678910 length

↓
↓

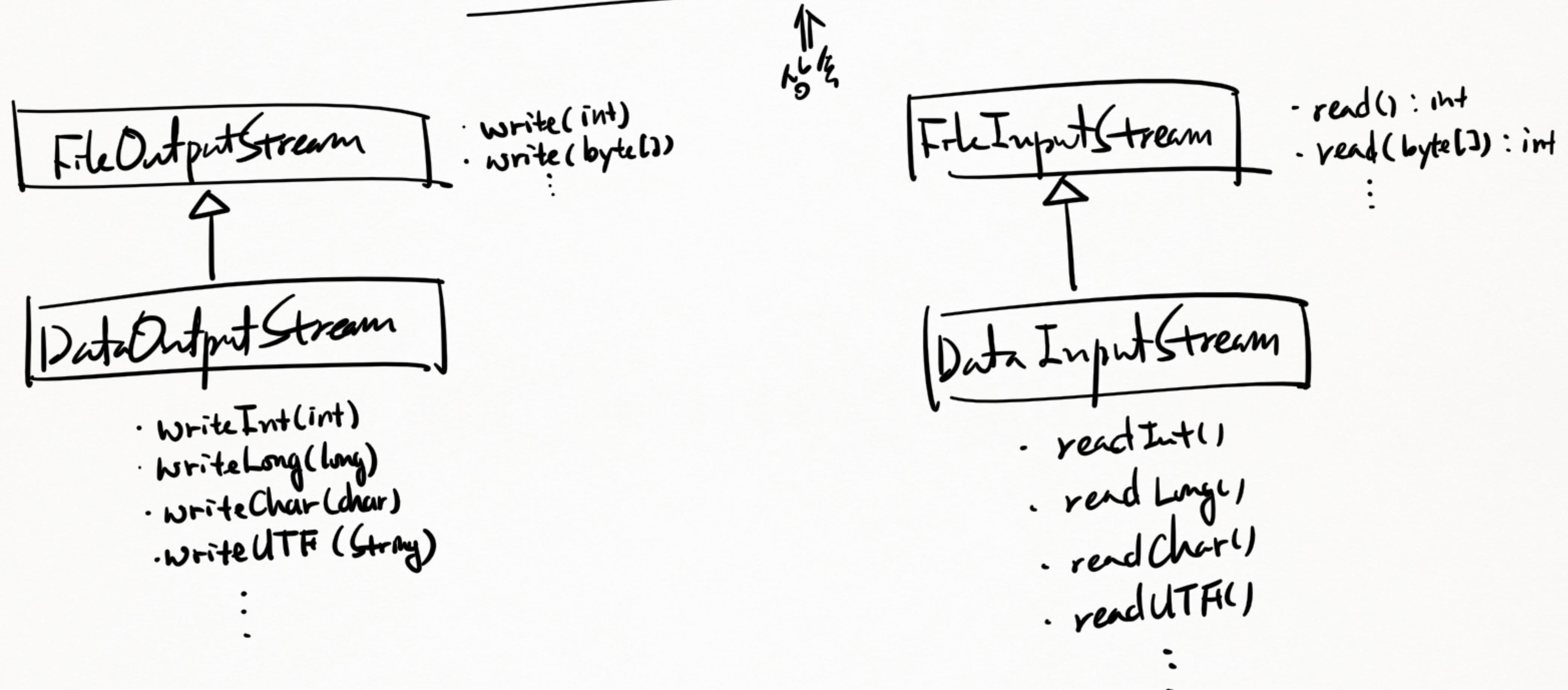
in.read(buf, 0, 3)

0|1|2|
61|61|61| ...
1000 bytes

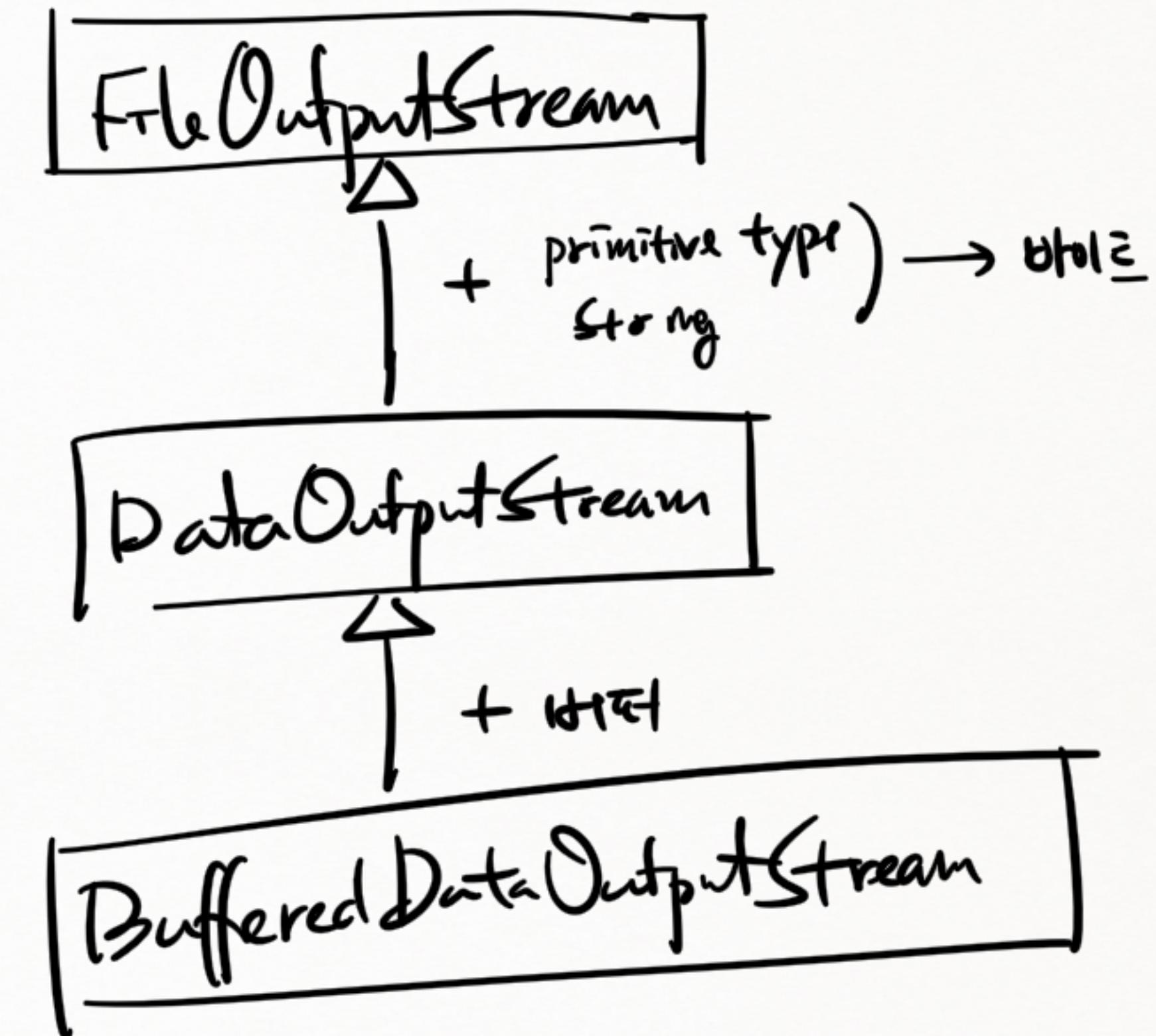
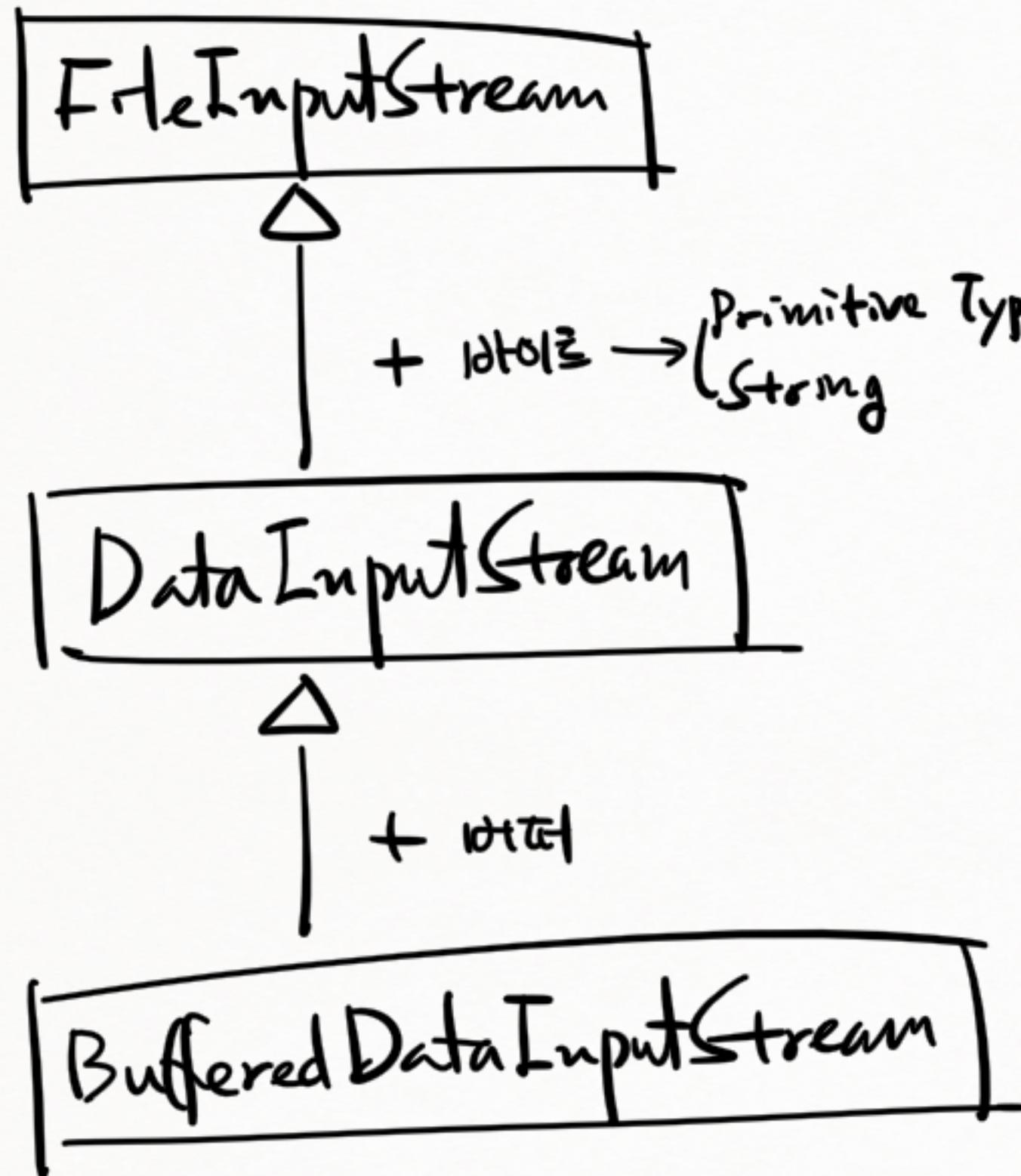
new String(, 0, count, "UTF-8")

member - setName()

28. FileInputStream / FileOutputStream + (primitive type) String 입출력 기법



29. 파일 입출력의 버퍼링 예제 : 파일 읽기 쓰기

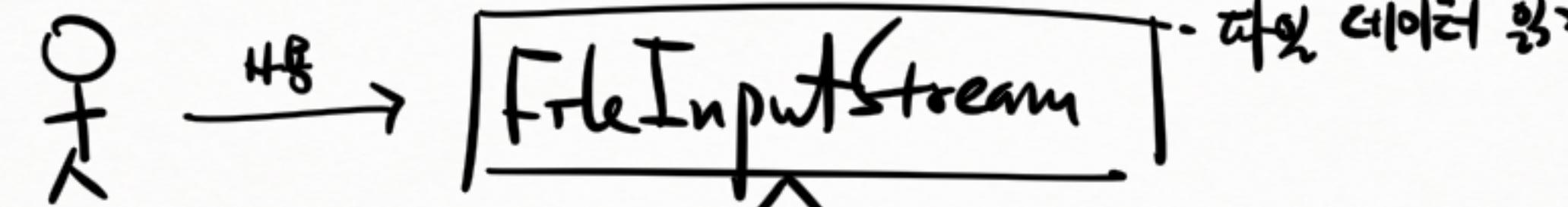


30. 기능을 확장할 때 상속과는 Decorator 패턴 적용

① 상속을 이용한 기능 확장의 문제점

- byte / byte[] 읽기

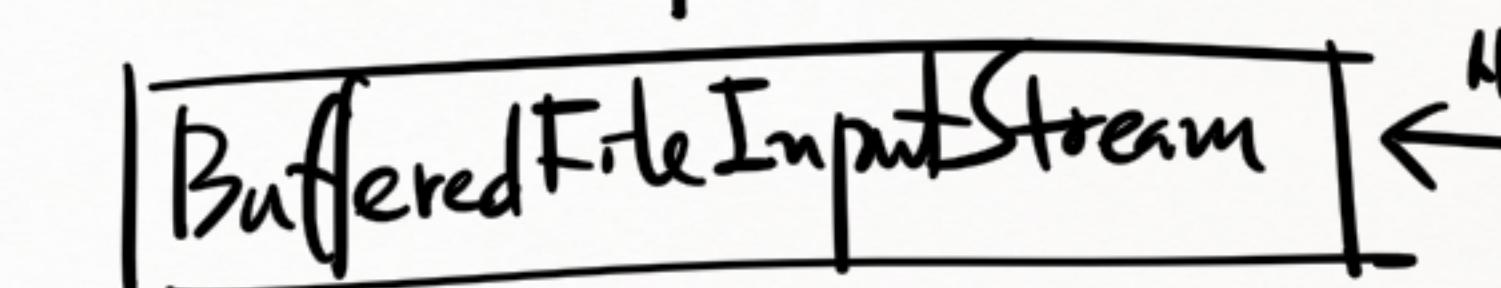
- * Primitive Type / String 읽기 불편



- (primitive type) 읽기 편함
String

- 바이트 단위로 읽기 대비해 대량의 데이터를 읽을 때 overhead 발생!
(Data Seek Time)

- 버퍼를 이용해서 읽기 성능 개선

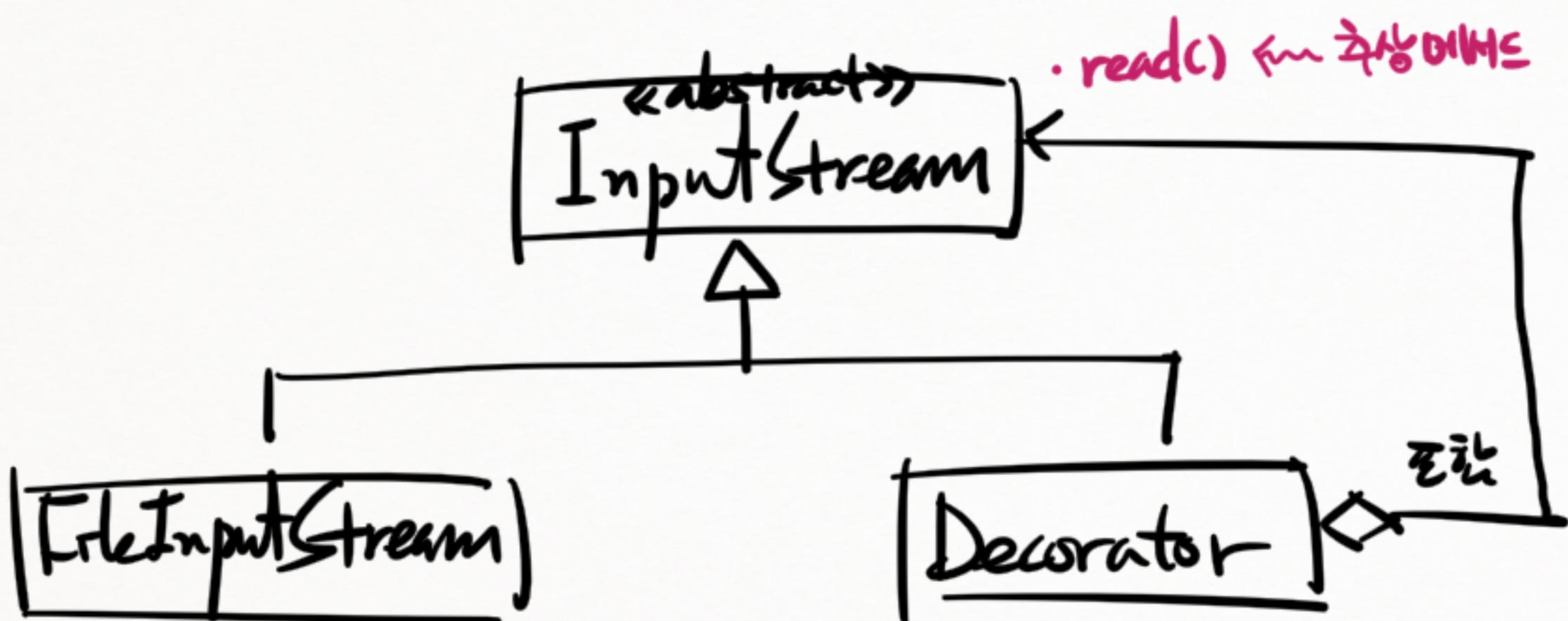


* 상속은 좋은 유도의 기능은
제한된 한계.
그러나 상속의 sub 클래스는 그 상위 클래스를
상속하는 한계의 한계!

(primitive type)의 데이터는
읽을 때마다 많은 성능이 사용할 때
바이트 단위로 데이터를 읽을 때
입기 성능은 개선되는 경우.
하지만.

30. 기능을 확장할 때 쌍축 with Decorator 티켓 적용

② 장식적(Decorator) 패턴 → 기능을 덧붙이고 예상하지 않을 수 있는 문제를

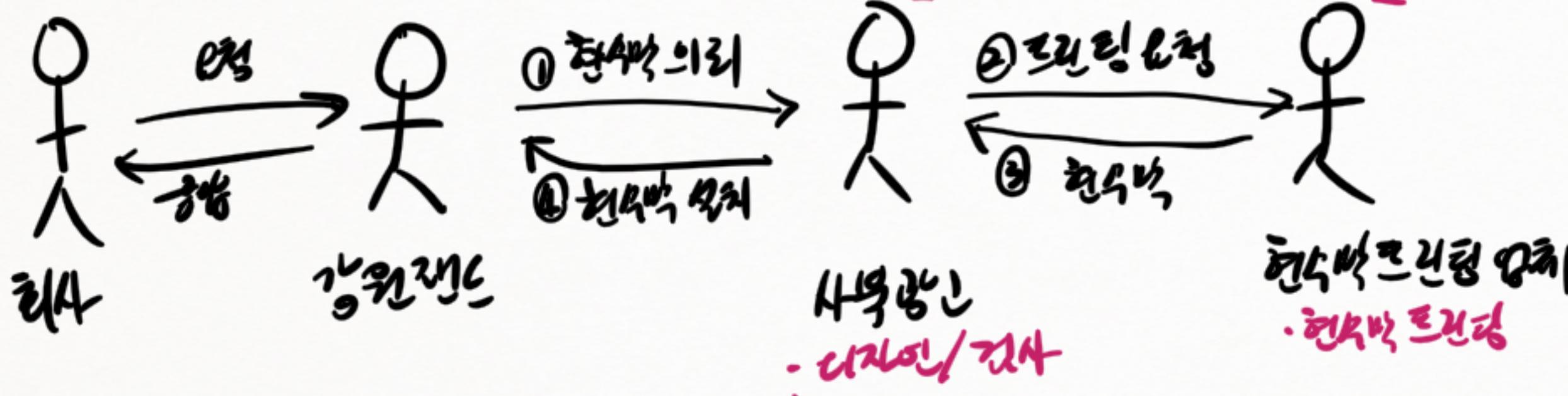


a) DataInputStream

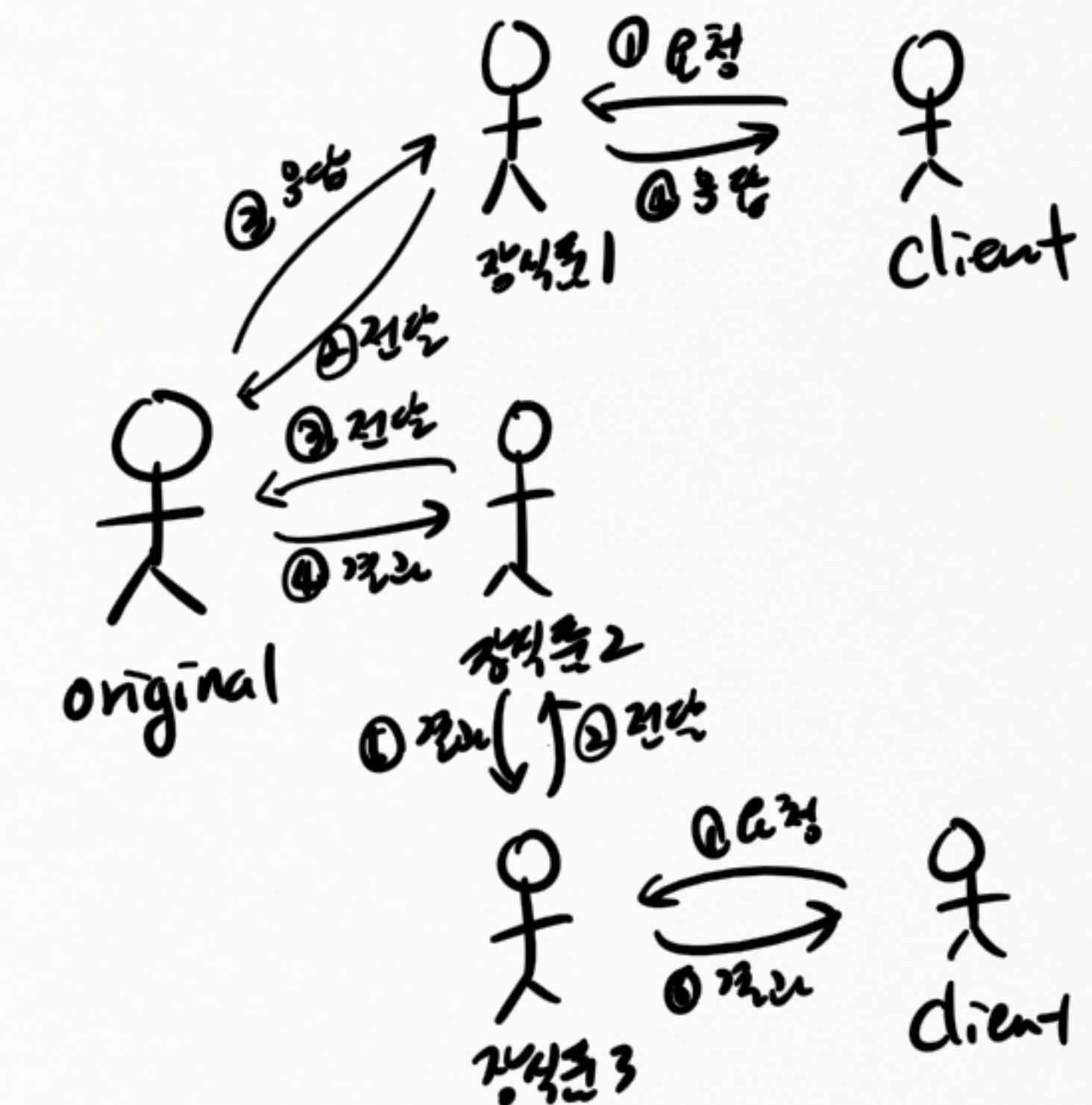
BufferedInputStream

✓ Decorator

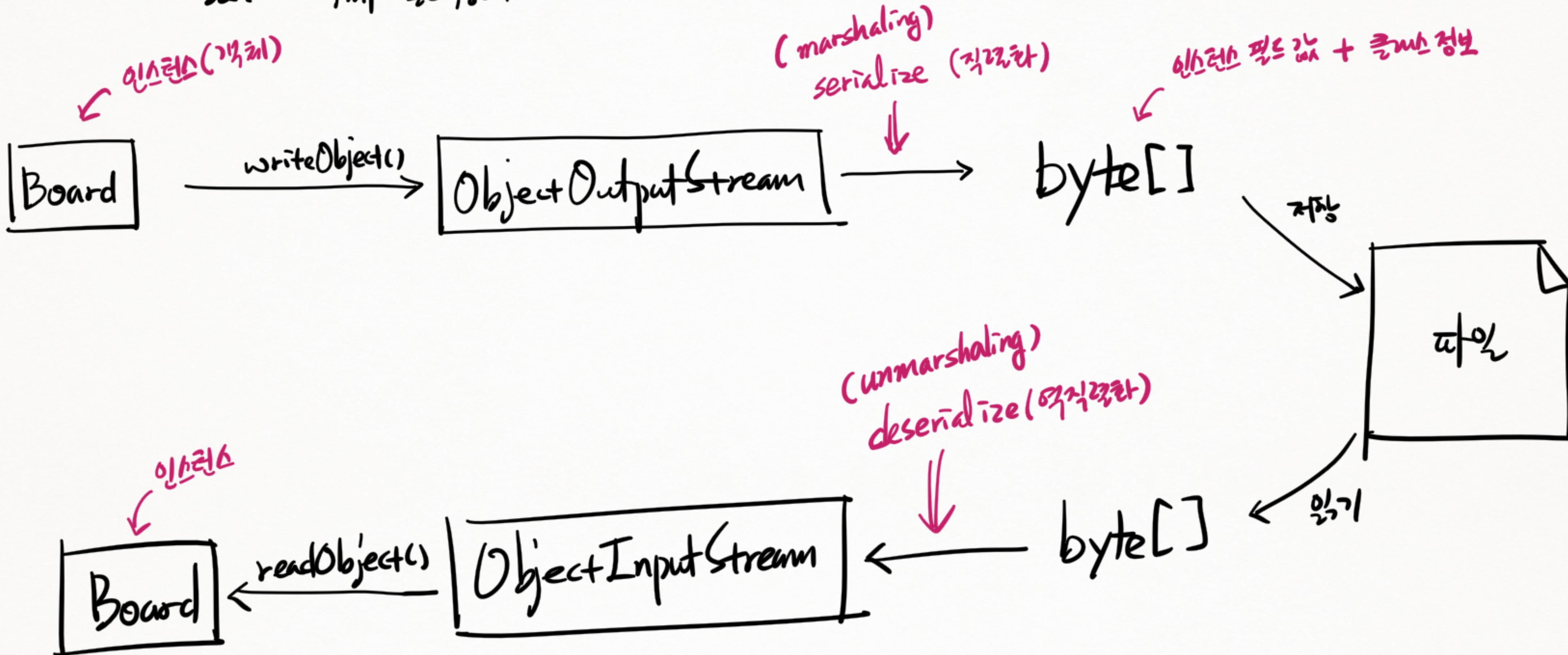
✓ Original



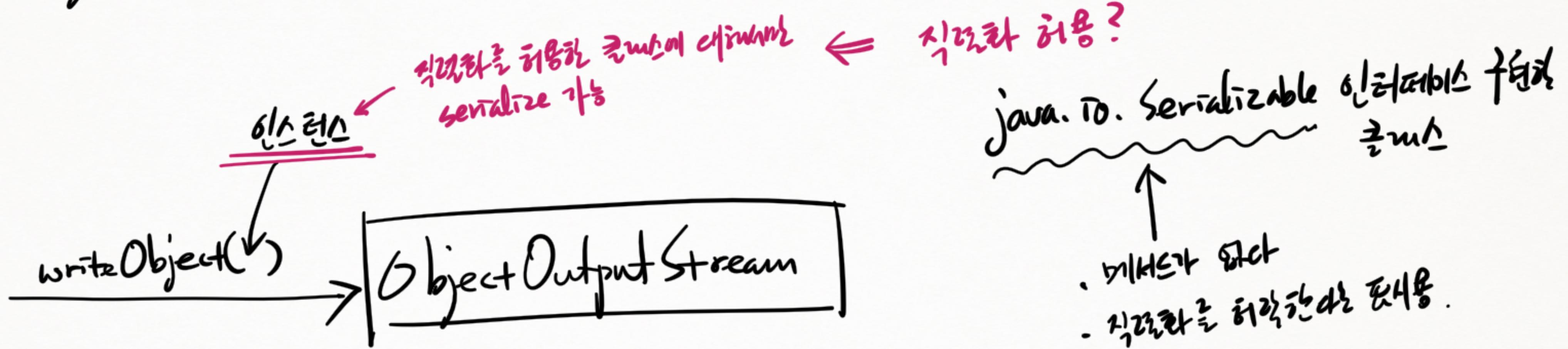
↳ Decorator 패턴 (GOF)
(Composite 패턴과 유사)



32. 객체 흘러보기



* java.io.Serializable 인터페이스

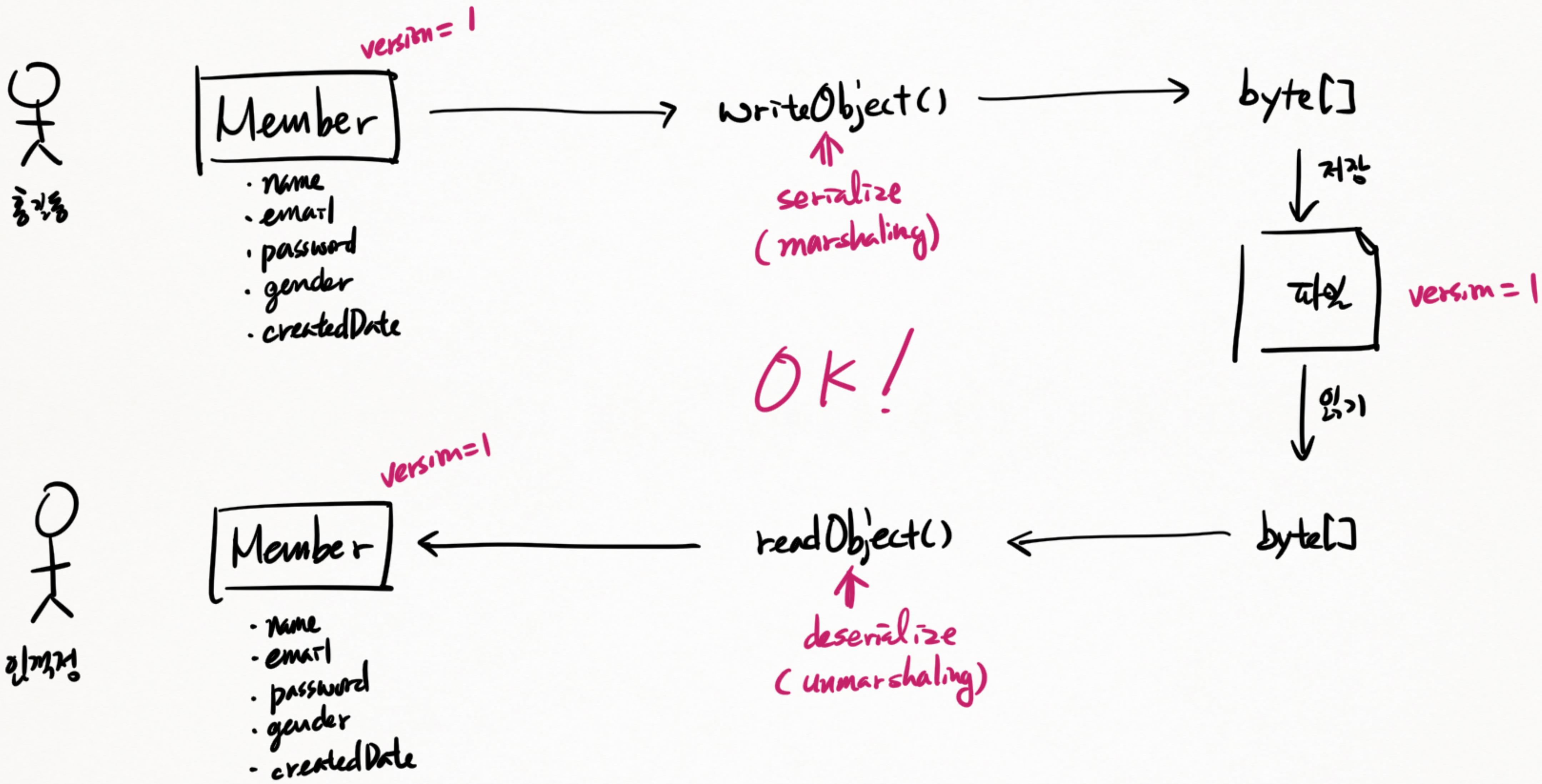


* 직렬화는 허락 받았던가?

↓
Yes

↑
"직렬화 사용을 위해 일부로 수정한 코드"
문장을 유롭게 표현!

* serialVersionUID 스태틱 필드



* serialVersionUID 스태틱 필드



등록

version = 1



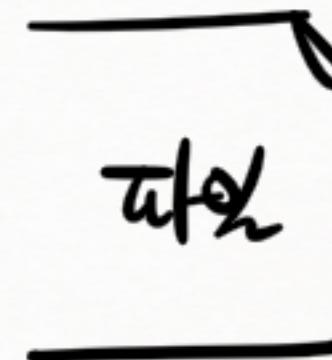
* 이런 버전은 serialize 한 데이터를
내부의 인스턴스로 맞애들이기 쉬워 ⇒ 버전 번호는 알리하라!

writeObject()

serialize
(marshaling)

byte[]

저장



version = 1

Error!

version = 2

구조체의
변경이
发生了



수정



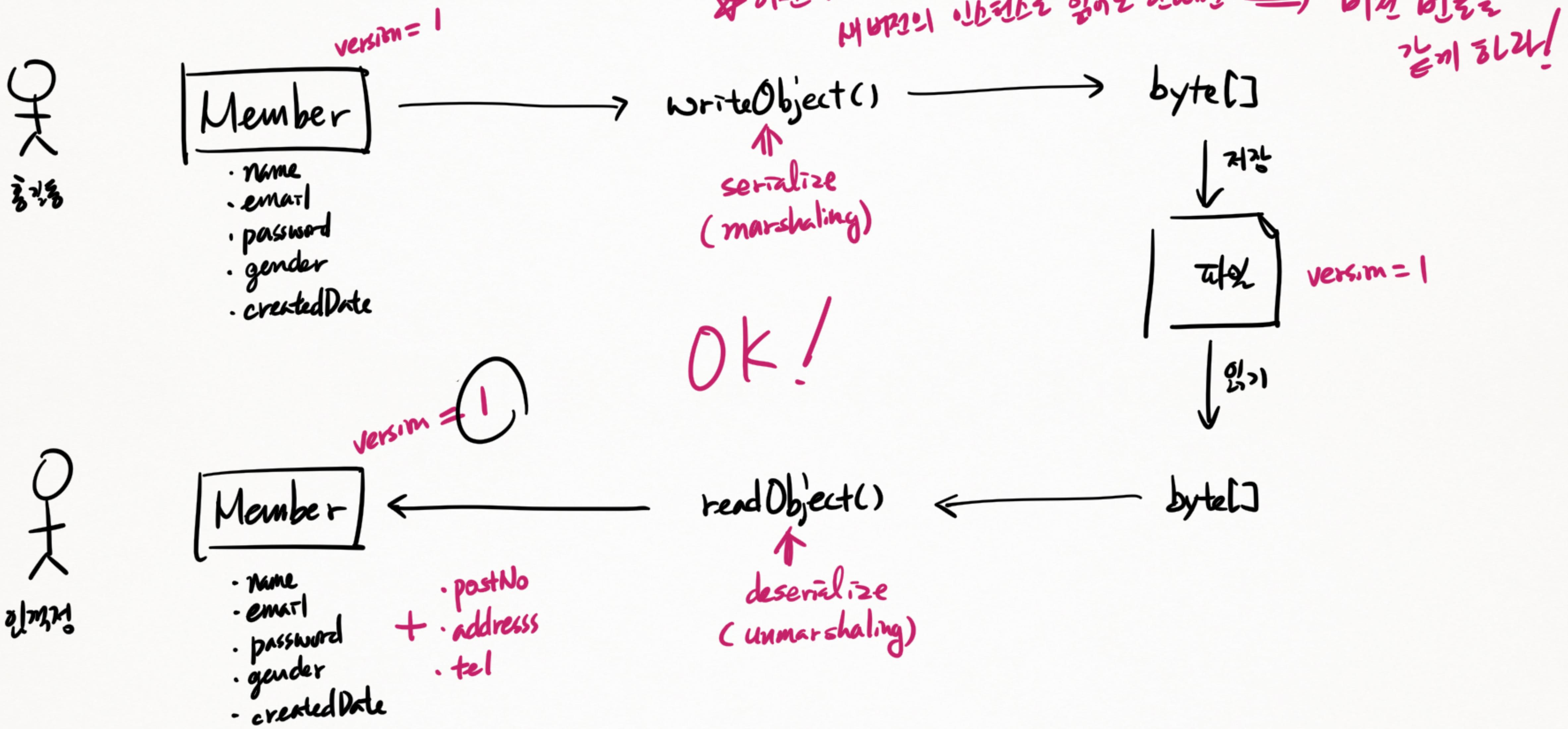
- name
- email
- password
- gender
- createdDate
+ postNo
+ addresss
+ tel

readObject()

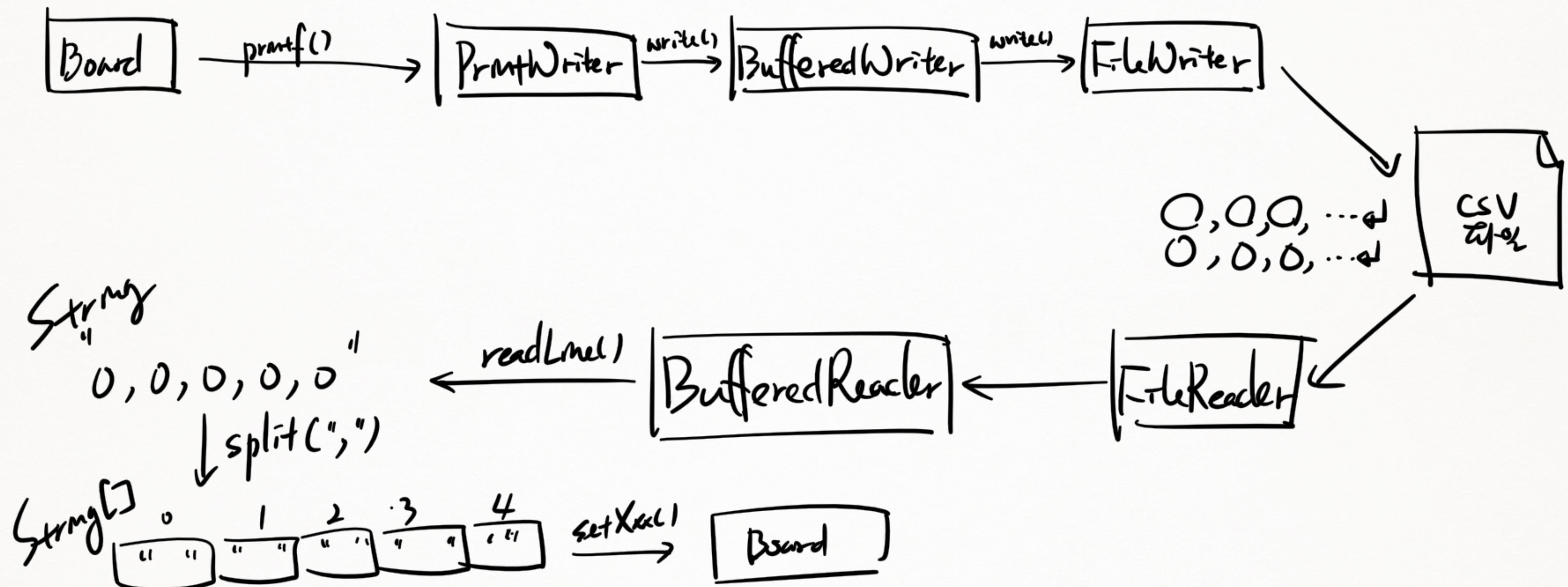
deserialize
(unmarshaling)

byte[]

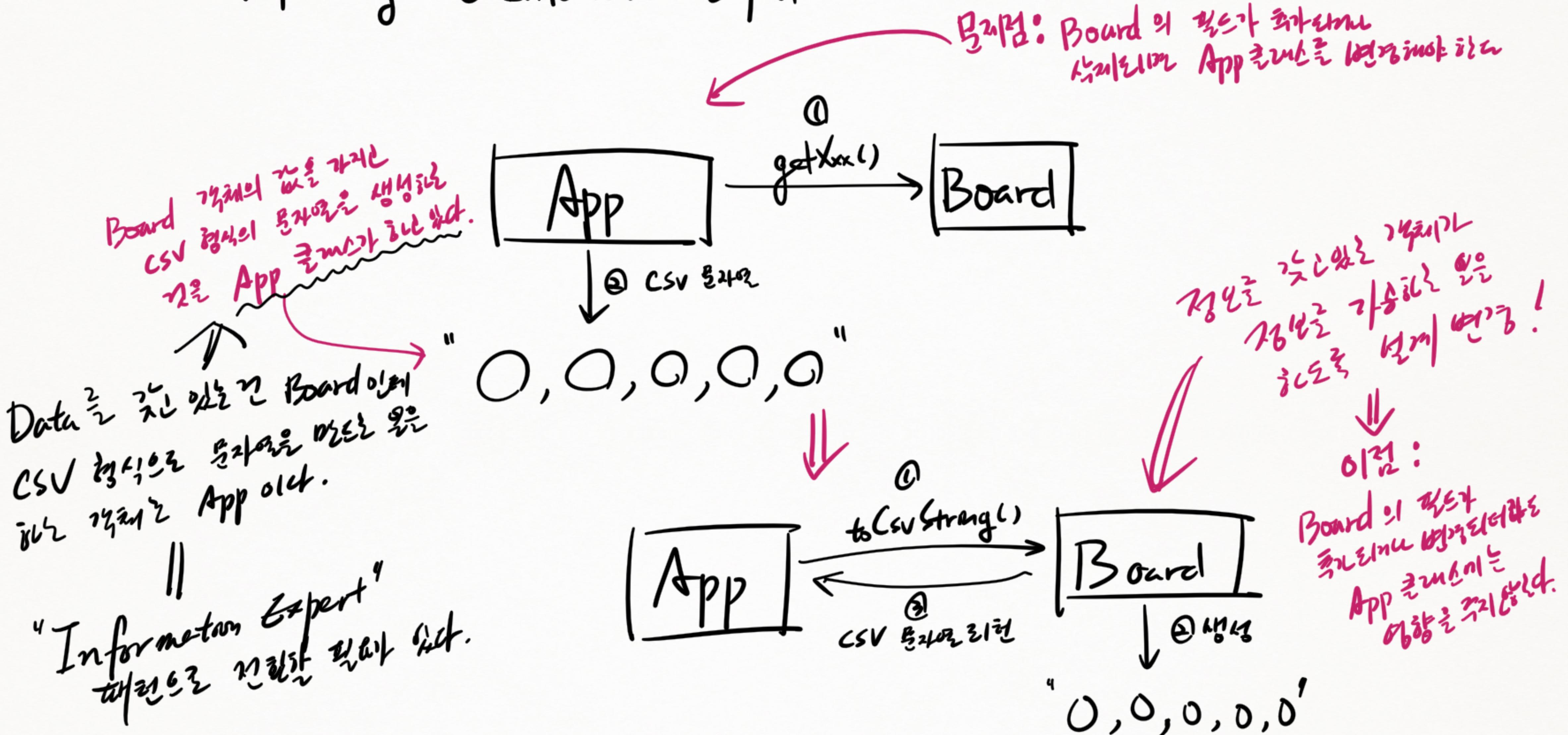
* serialVersionUID 스태틱 필드



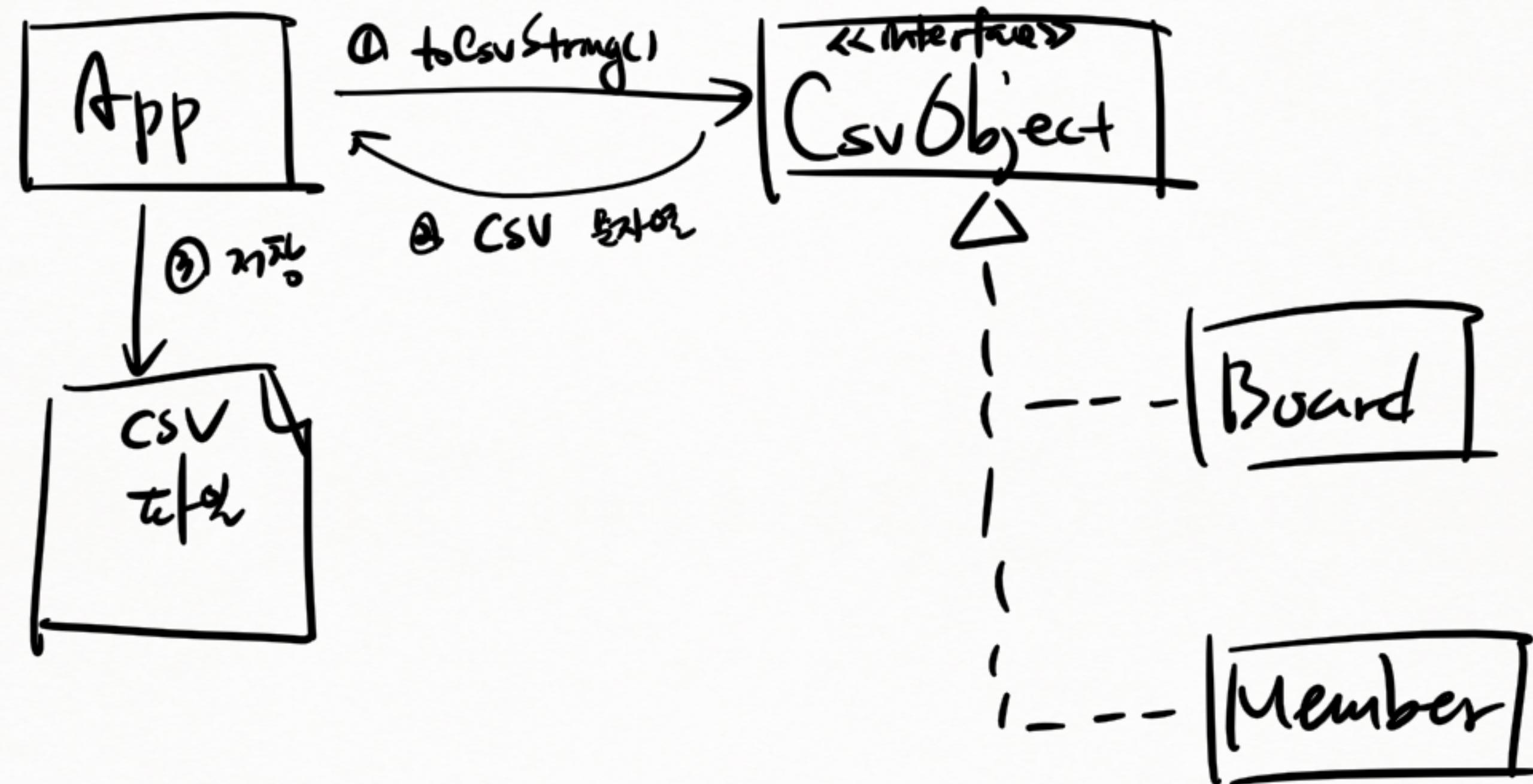
33. 텍스트 파일 (CSV) 을 파일로 읽기



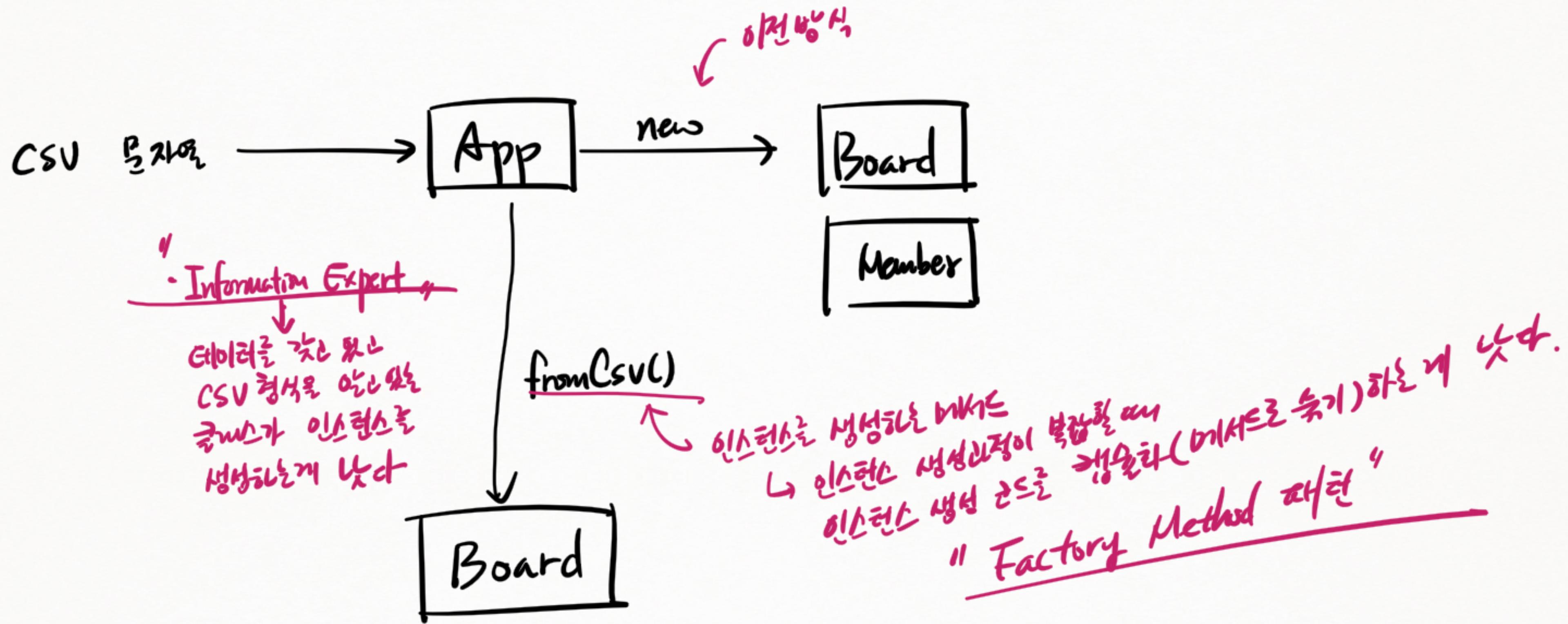
34. Refactoring : ① Information Expert



* Interface 분리의 원칙



② Factory Method (GOF) 패턴 적용



② Factory Method (GOF) 패턴 적용 → 개선

T가 타입 파라미터임을 선언!

```
<T> void loadCsv(String filename, List<T> list, Class<T> clazz) {
```

* Reflect.m API를 사용하면
매번 코드를 찾고 훔을 드립니다! + Generic을 사용하면
다양한 타입에 대응할 수 있고
매번 코드를 찾지 않아!

Method factoryMethod = clazz.getDeclaredMethod("fromCsv", String.class);

↑
매번 코드를 찾고 훔을 드립니다.
직접 코드를 찾는다.

↑
현재 클래스에서
정의된 메소드를
찾는다.
매번 코드를 찾는다.

↑
매번 코드를 찾는다.
파라미터 타입
파라미터 타입

(T) factoryMethod.invoke(null, line);
↑
매번 코드를 찾는다.
인스턴스 주고
(스레蚀 인스턴스의 경우, null을 넣기다.)
매번 코드를 찾는다.
매번 코드를 찾는다.
매번 코드를 찾는다.

35. JSON 형식으로 입출력하기

↳ JavaScript Object Notation : 자바스크립트 객체 리터럴 문법을 모방하여 만든 텍스트 파일 형식

① 자바스크립트 객체 리터럴

문자열 - { "문자열",
 '문자열'

숫자 - 3.14
 3.14

논리 - true/false

문서 - { *property name*: *property value*,
 name: "홍길동",
 "age": 20,
 'working': true
 }

JSON 흔히

{

"name": "홍길동",
"age": 20,
"working": true

}

반드시 큰따옴표(")를 사용해서
프로그래밍 이름을 써야 한다.

문자열은 반드시 큰따옴표 사용

XML이나
C언어하고 차이가 있다.

JSON

또는

XML

deserialize

"

decomposing

encoding

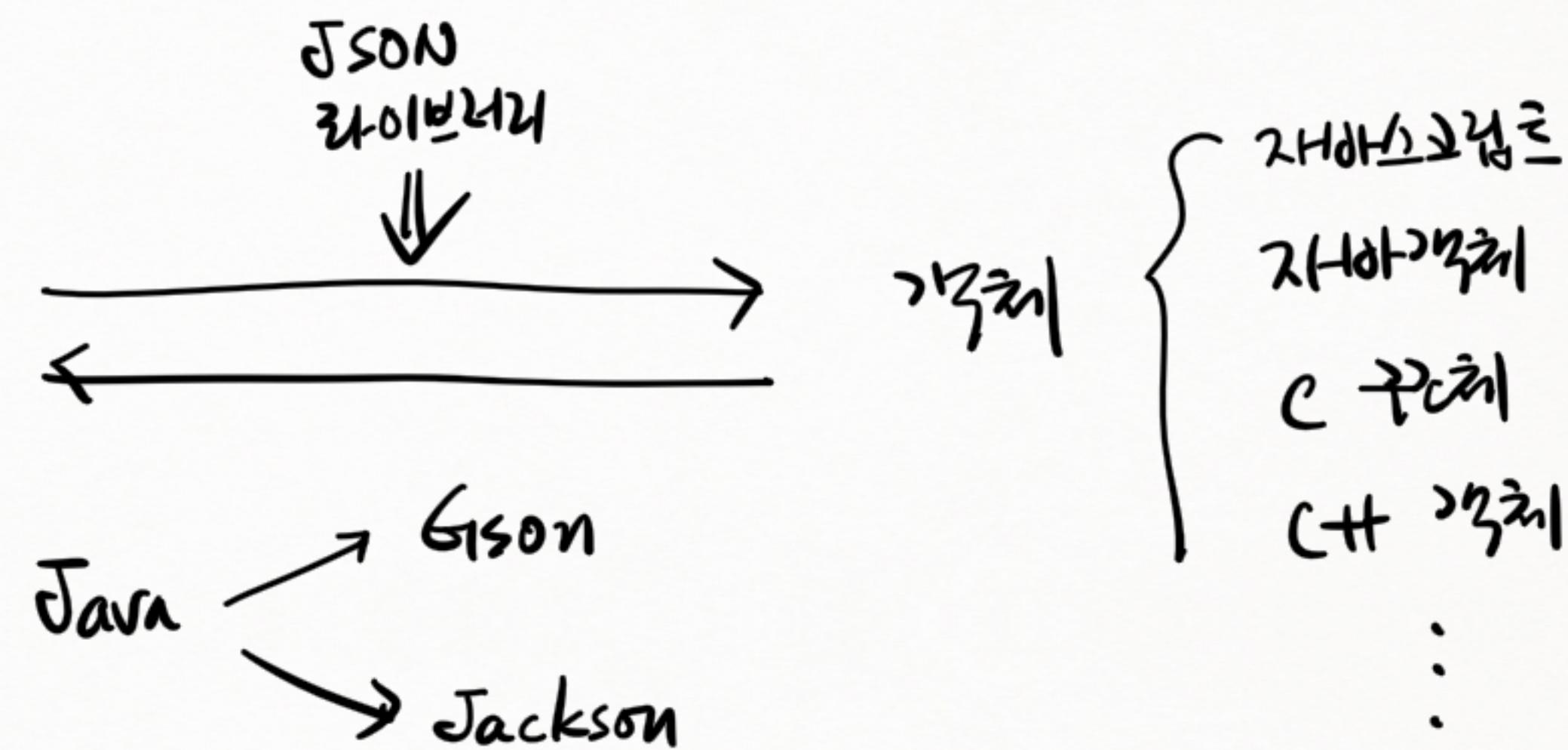
"

serialize

자바스크립트 객체

* JSON 인코딩 / 디코딩

JSON 형식의
문자열



* JSON 훔기시 필드명은 끄로처리링?

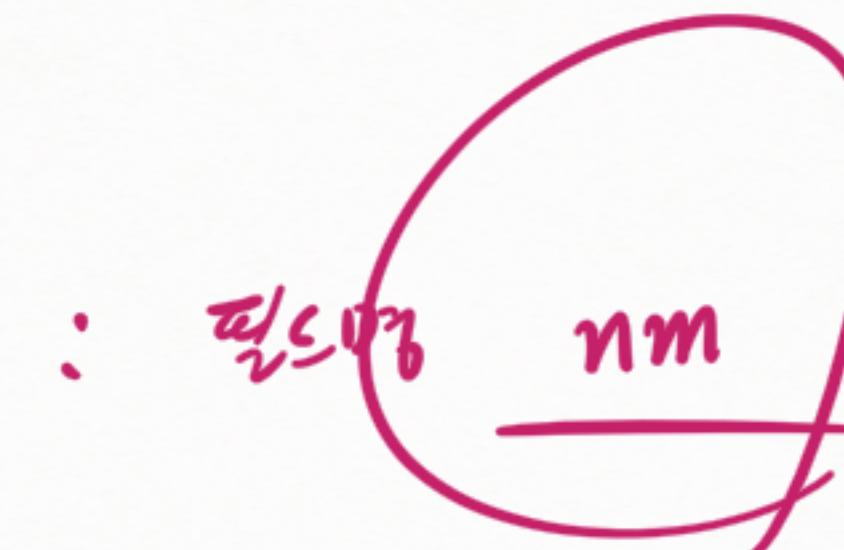
class Member {

String nm; ← field

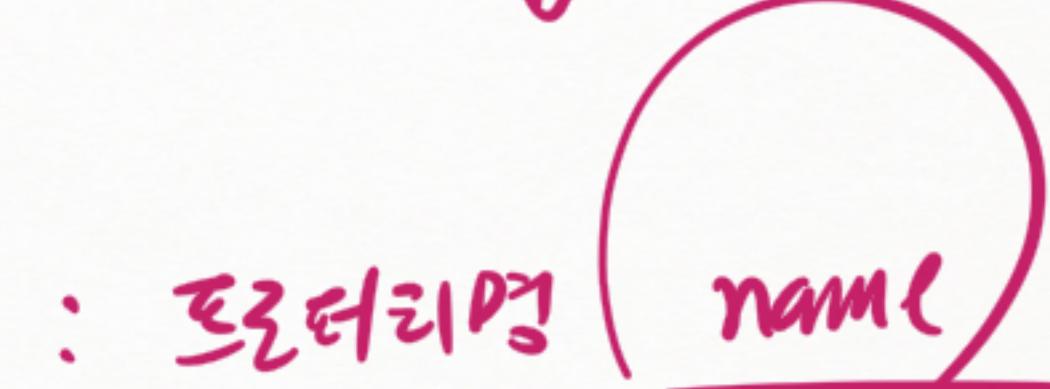
void setName(...){...}

String getName(){...}

}



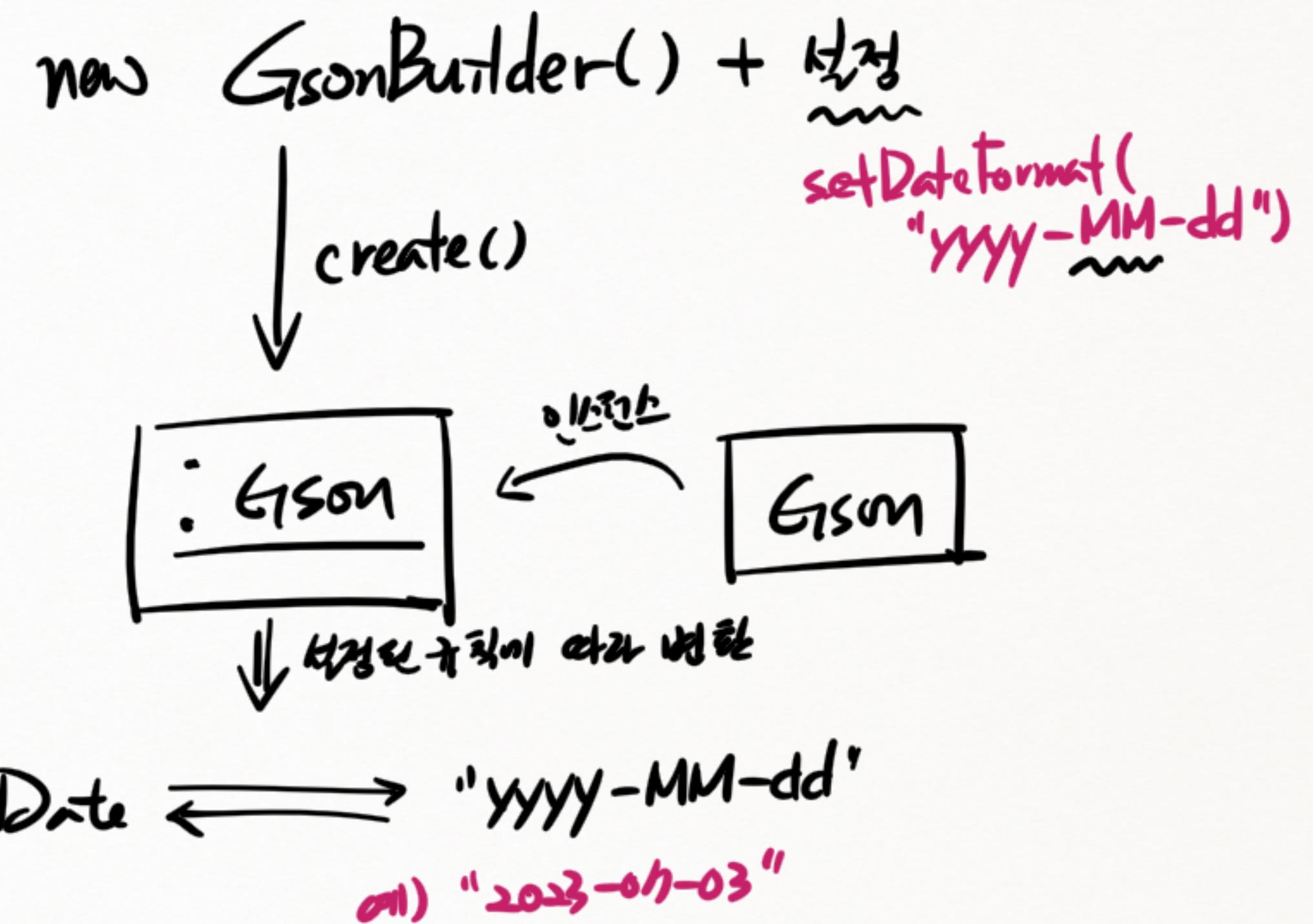
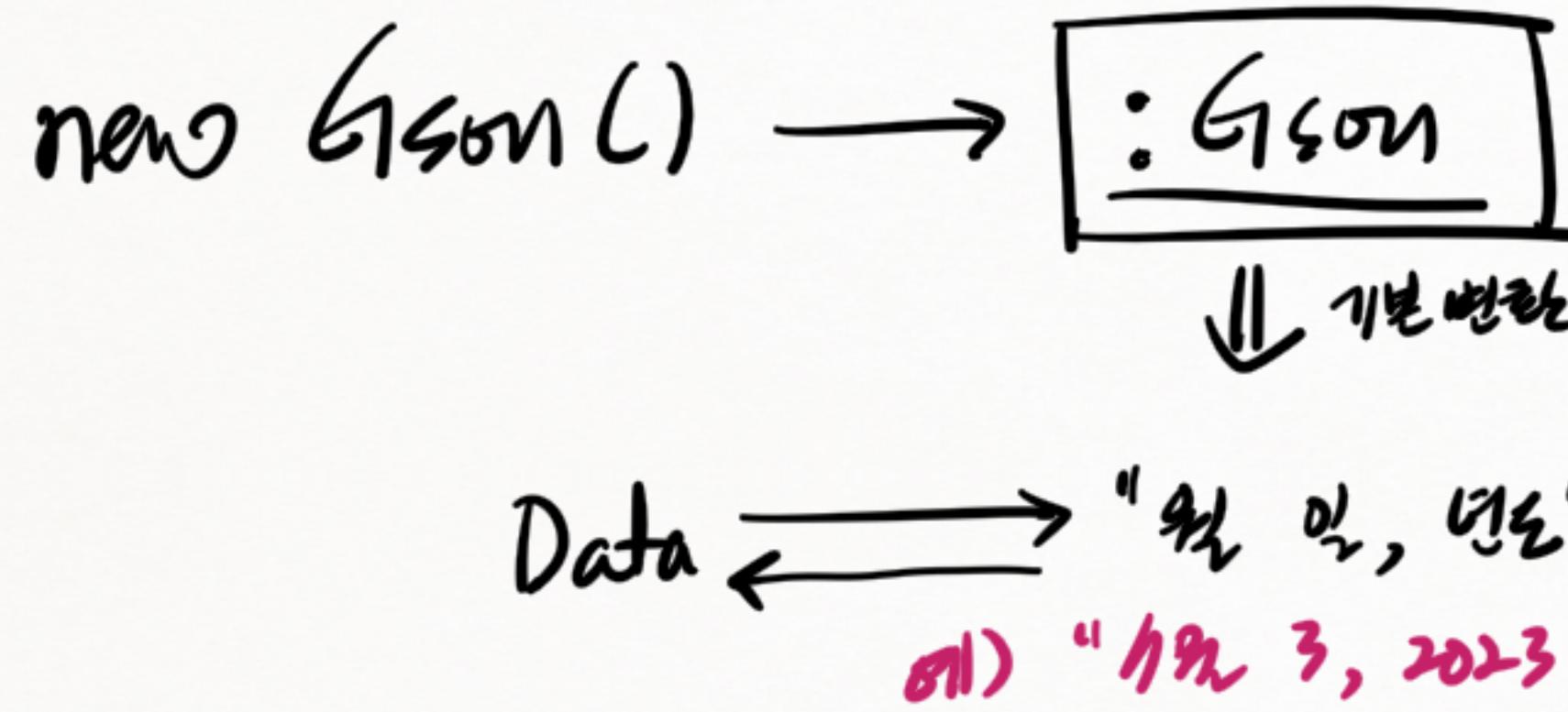
} ← property
" " setter/getter



- set/get 메서드
- 나머지 필드명이나
첫번째 알파벳을
소문자로 할 이는

프로퍼티명과 필드명이 다른 경우
• Gson → 필드명 사용
• Jackson → 프로퍼티명 사용
↓
결론!

* Gson with 날짜 다루기

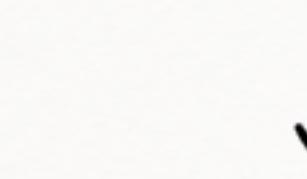


* yyyy : 2023년
month MM : 2023년
dd : 2023년 01월
HH : 2023년 01월
mm : 2023년 01월
ss : " 초 "

* Gof의 Builder 패턴

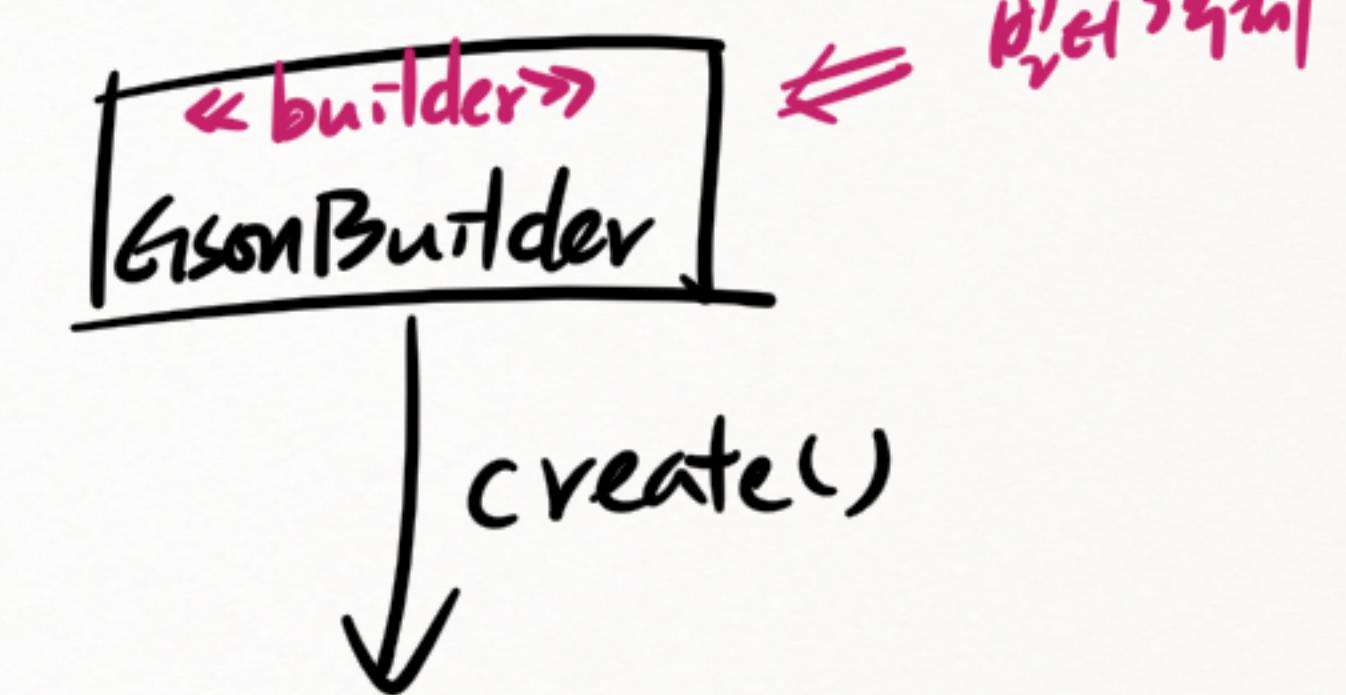
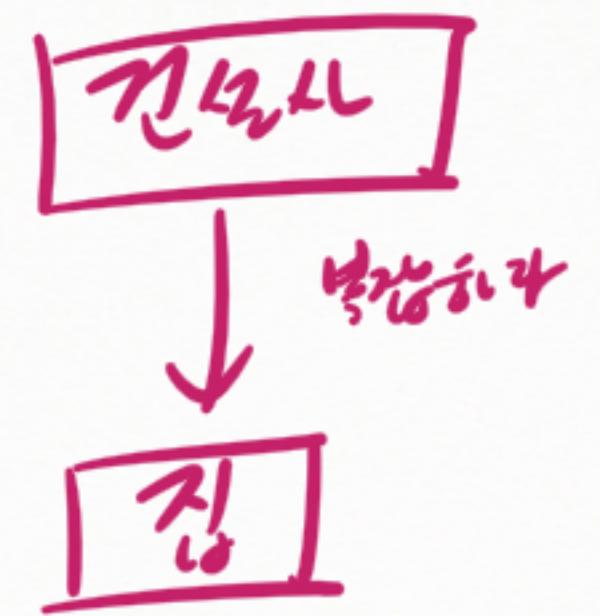
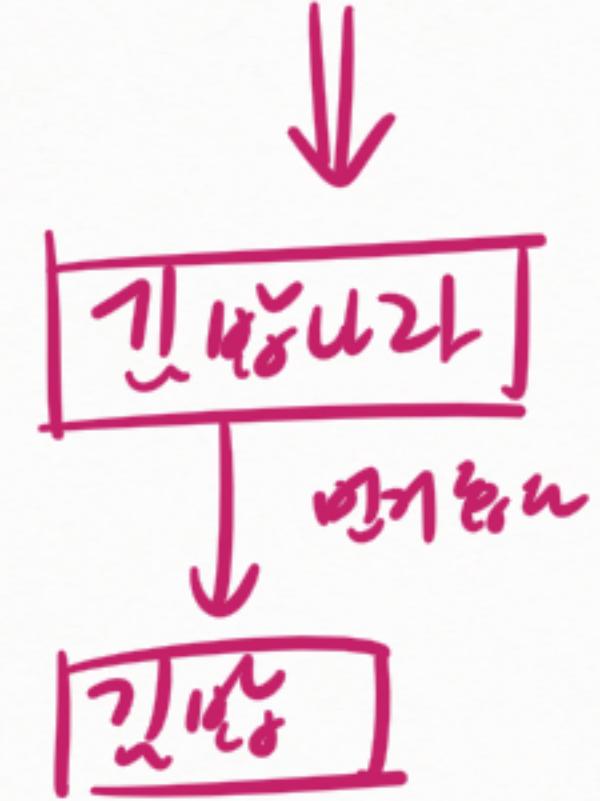
① new 뒤에
자기만 인스턴스 만들기

new Eison()



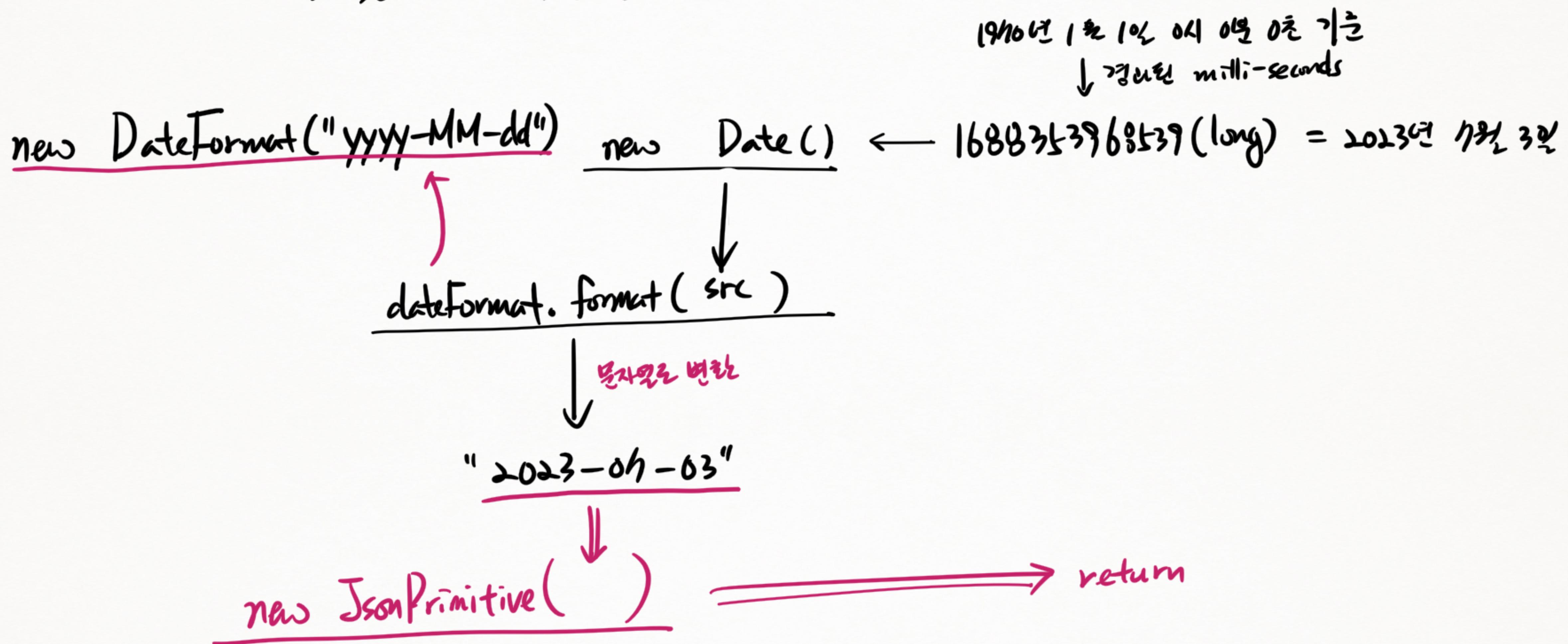
인스턴스 생성과정이 (복잡한 경우)
반복문을 경우

② 토끼의 낚지를 통해 인스턴스 생성

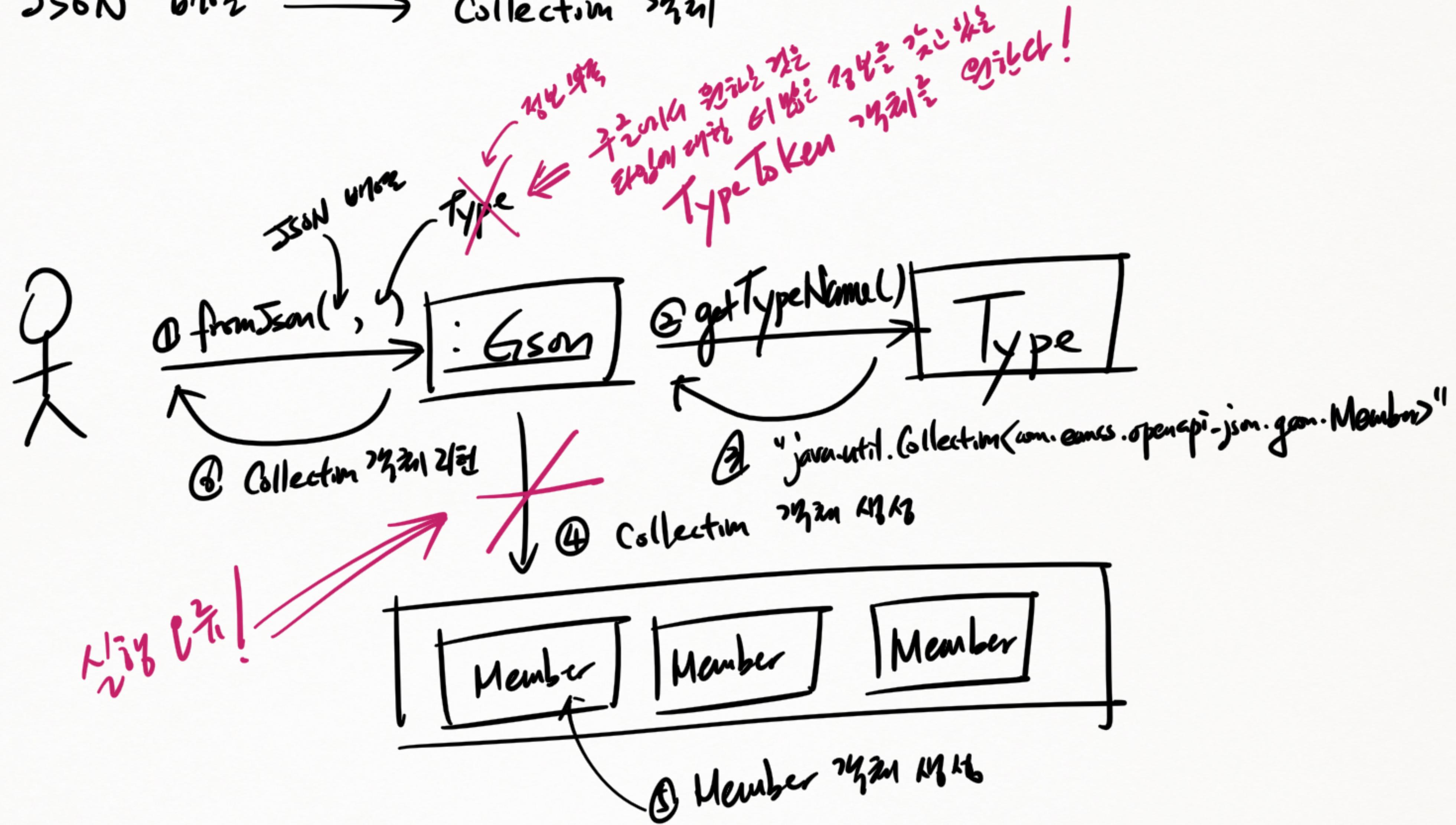


_builder

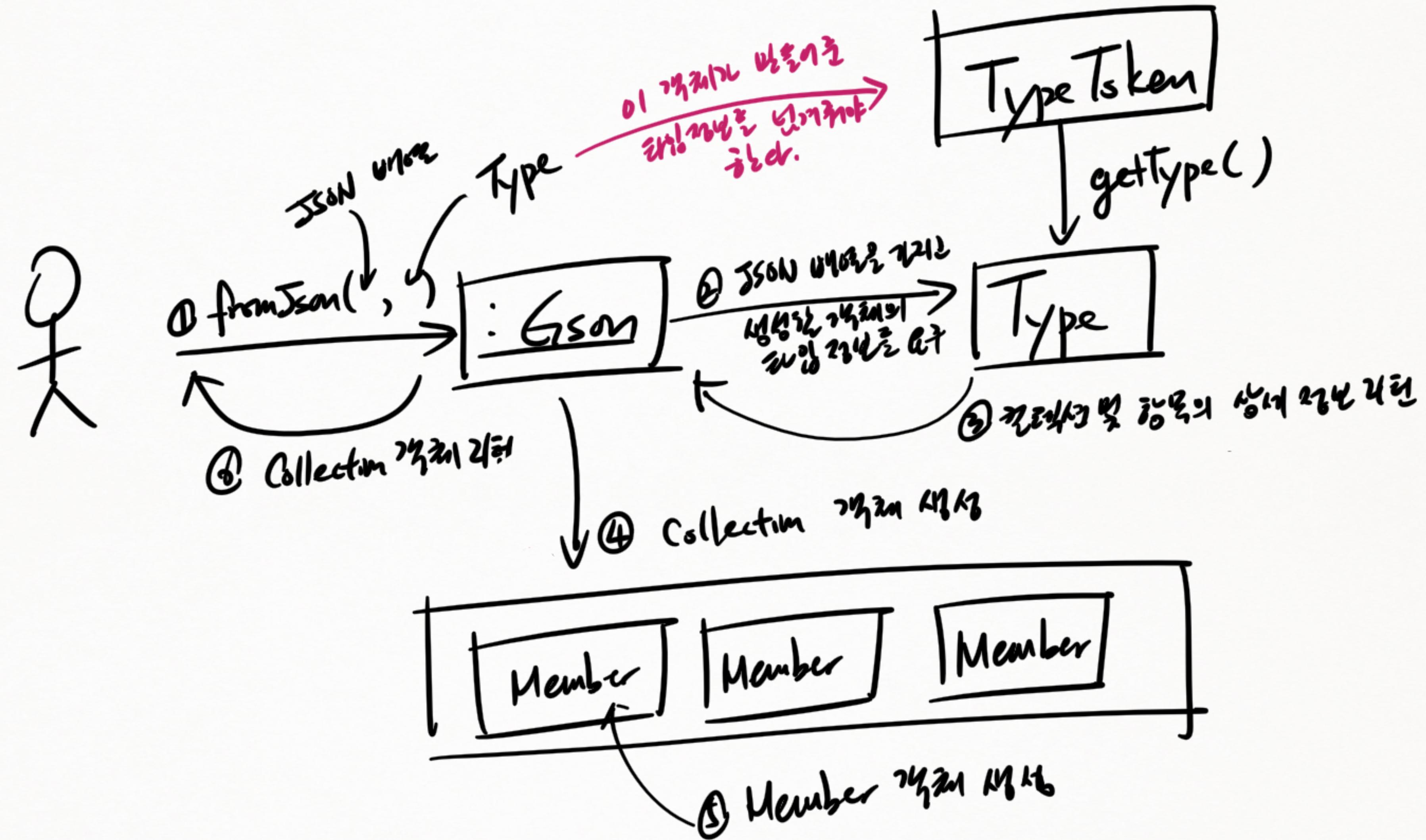
* JsonSerializer.serialize()



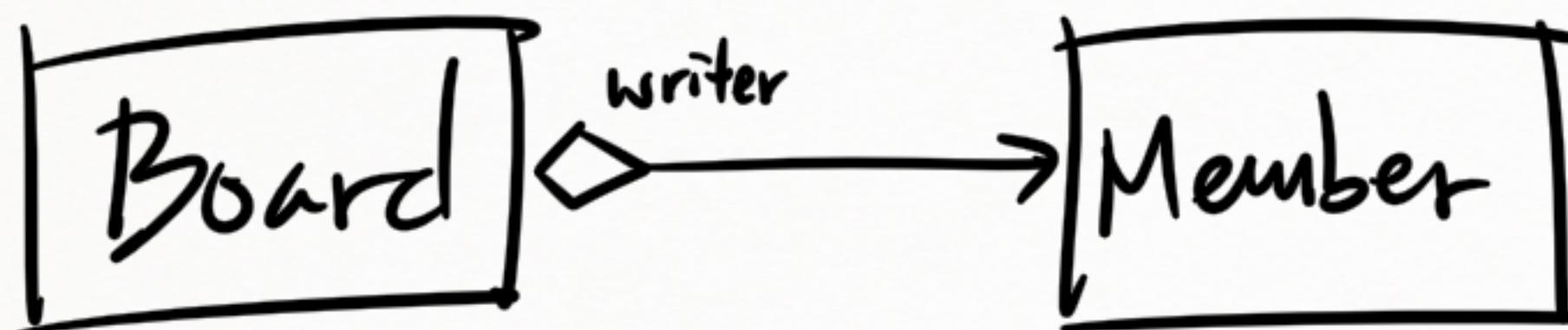
* JSON 풀기 → Collection 풀기



* JSON 풀기 → Collection 풀기



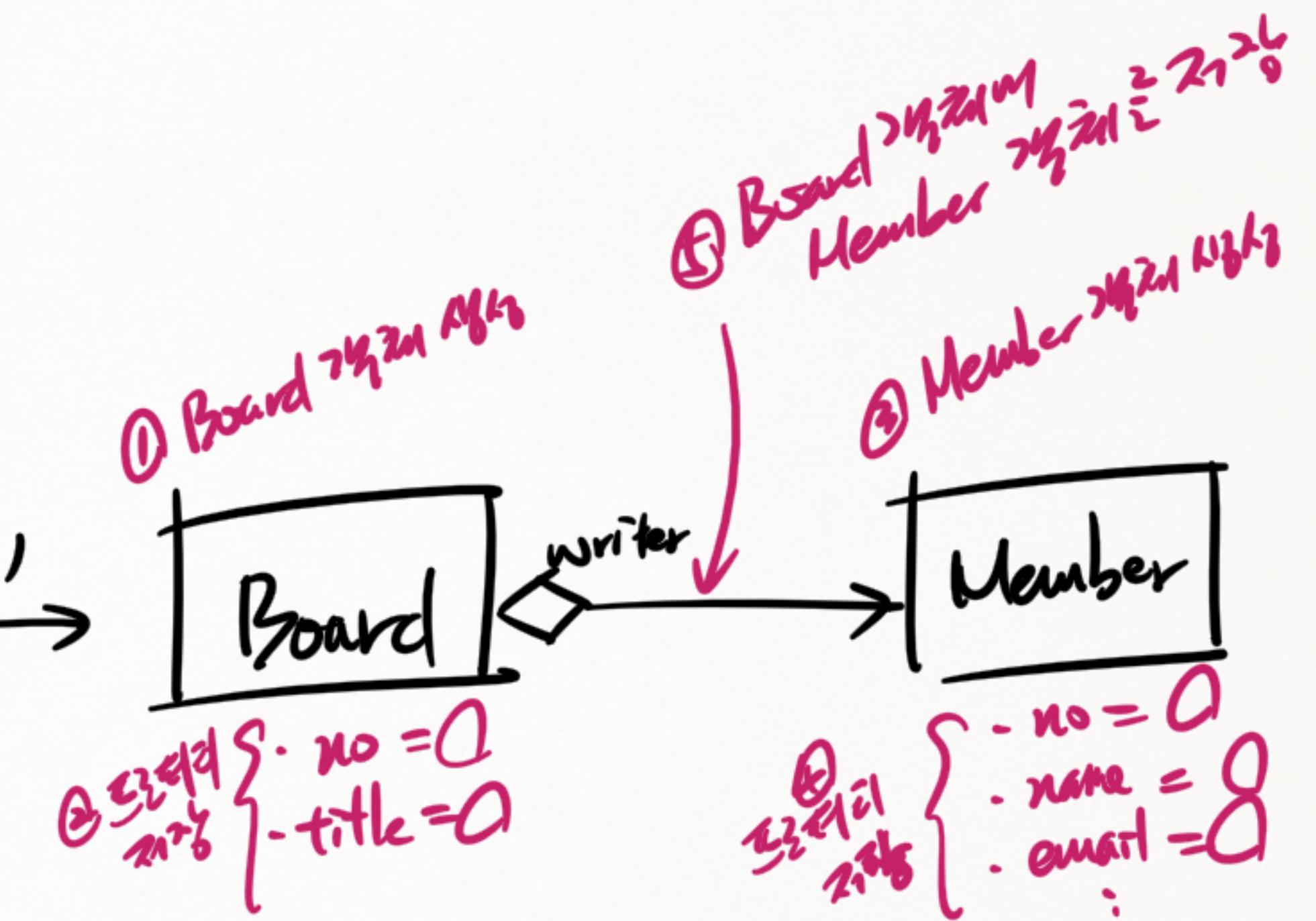
* 다른 객체를 찾고 있는 경우



↓
toJSON()

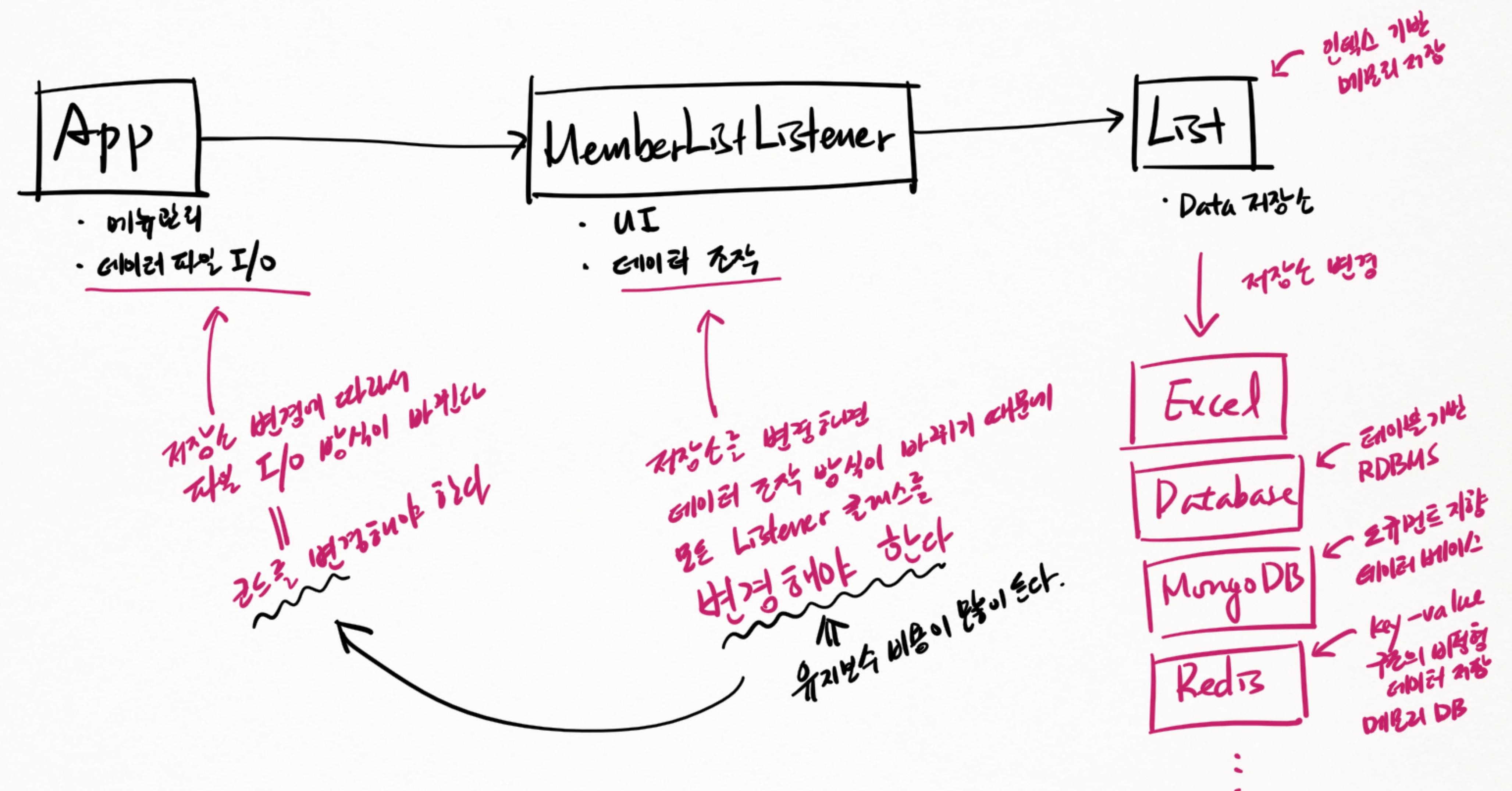
{
 "no": 1,
 "title": "—",
 :
 "writer": {
 "no": 100,
 "name": "김민경",
 :
 }
}

fromJson()



36. DAO (Data Access Object) එක

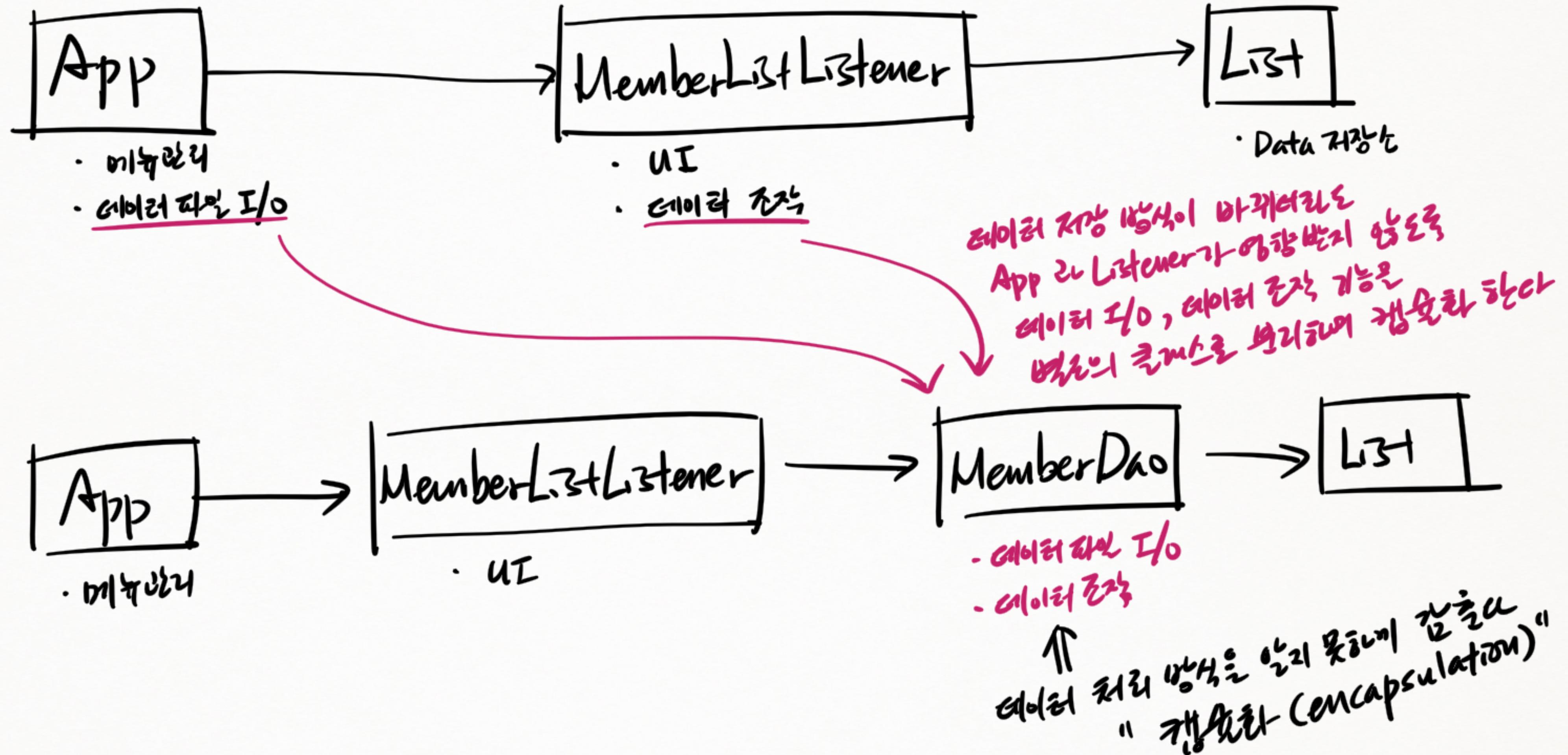
* 천장



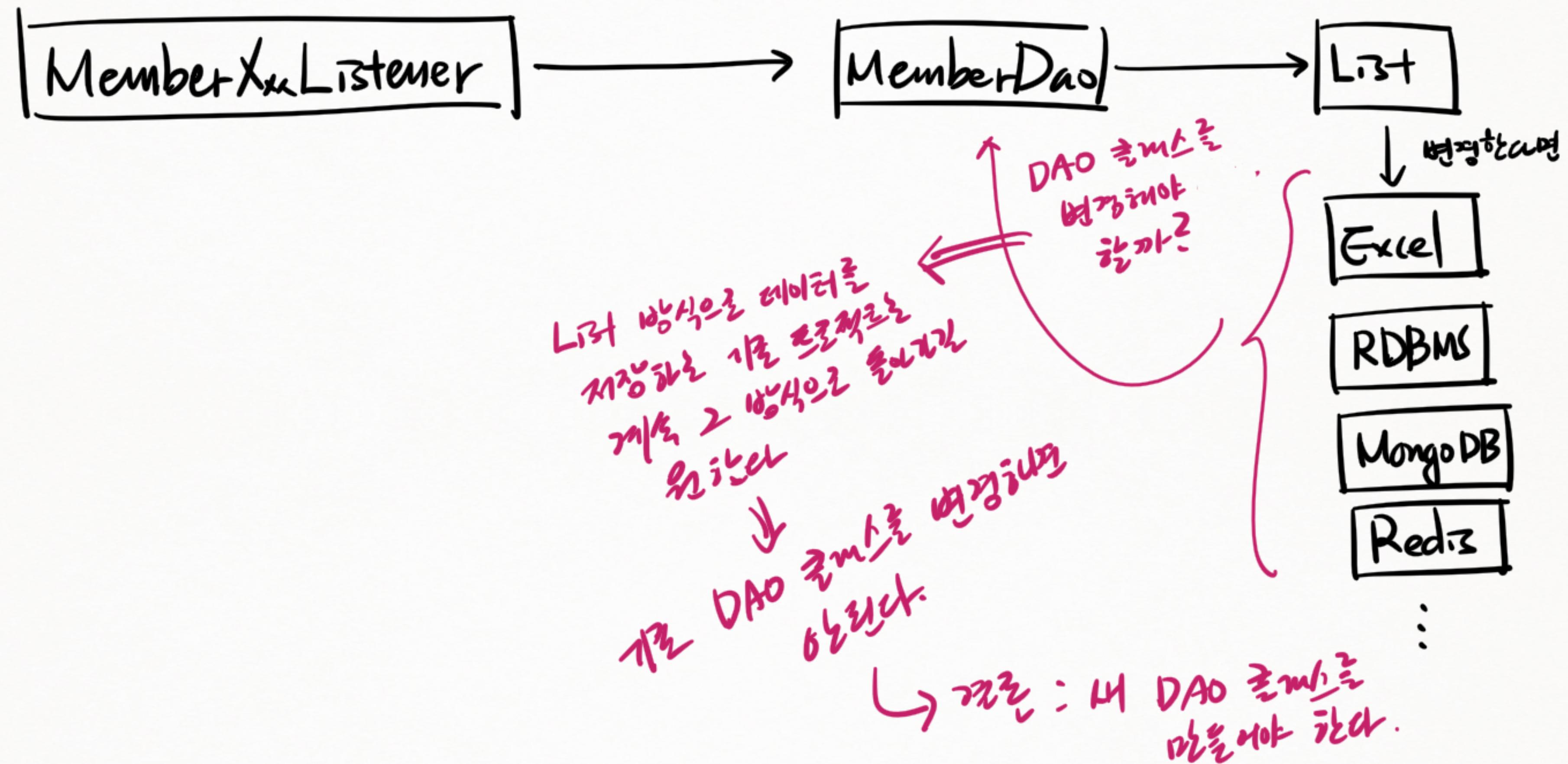
36. DAO (Data Access Object) 은?

↳ 데이터 저장 및 조작 단위

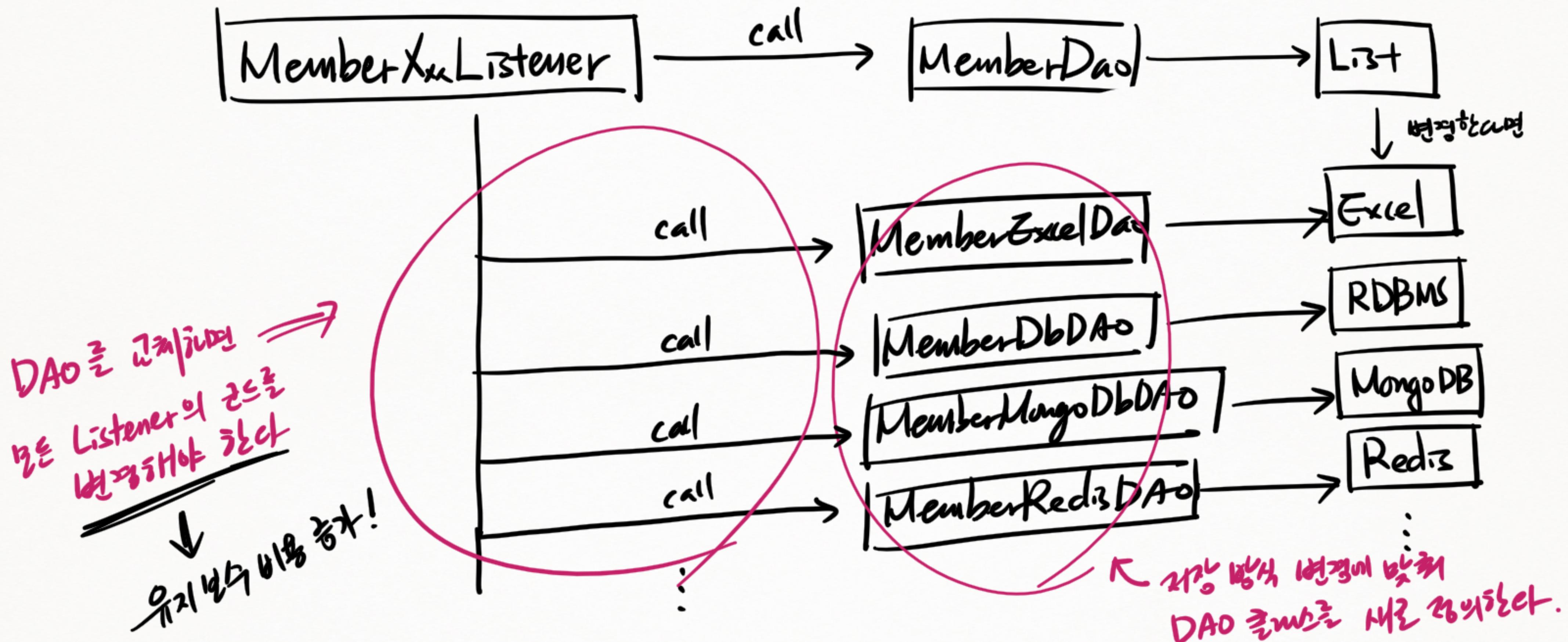
* 개선



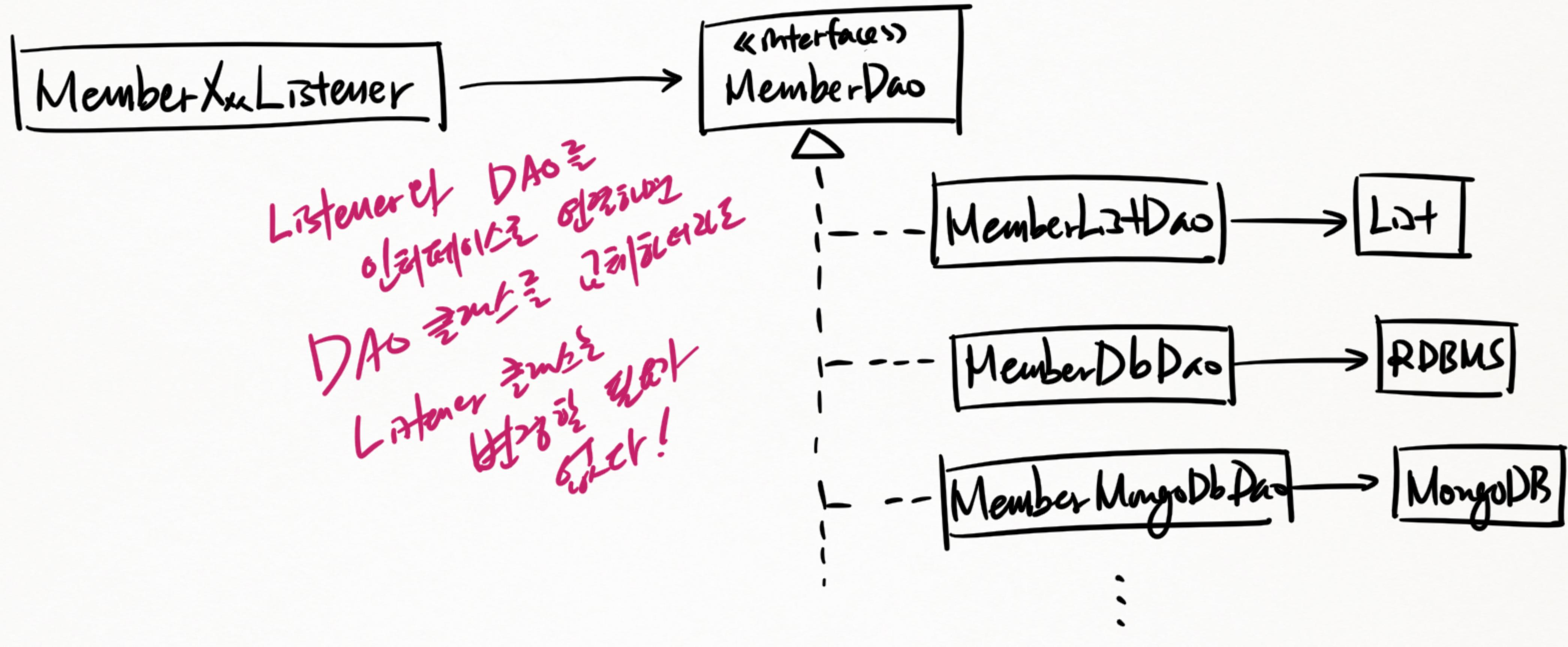
DAO 만들기



DAO 만들기

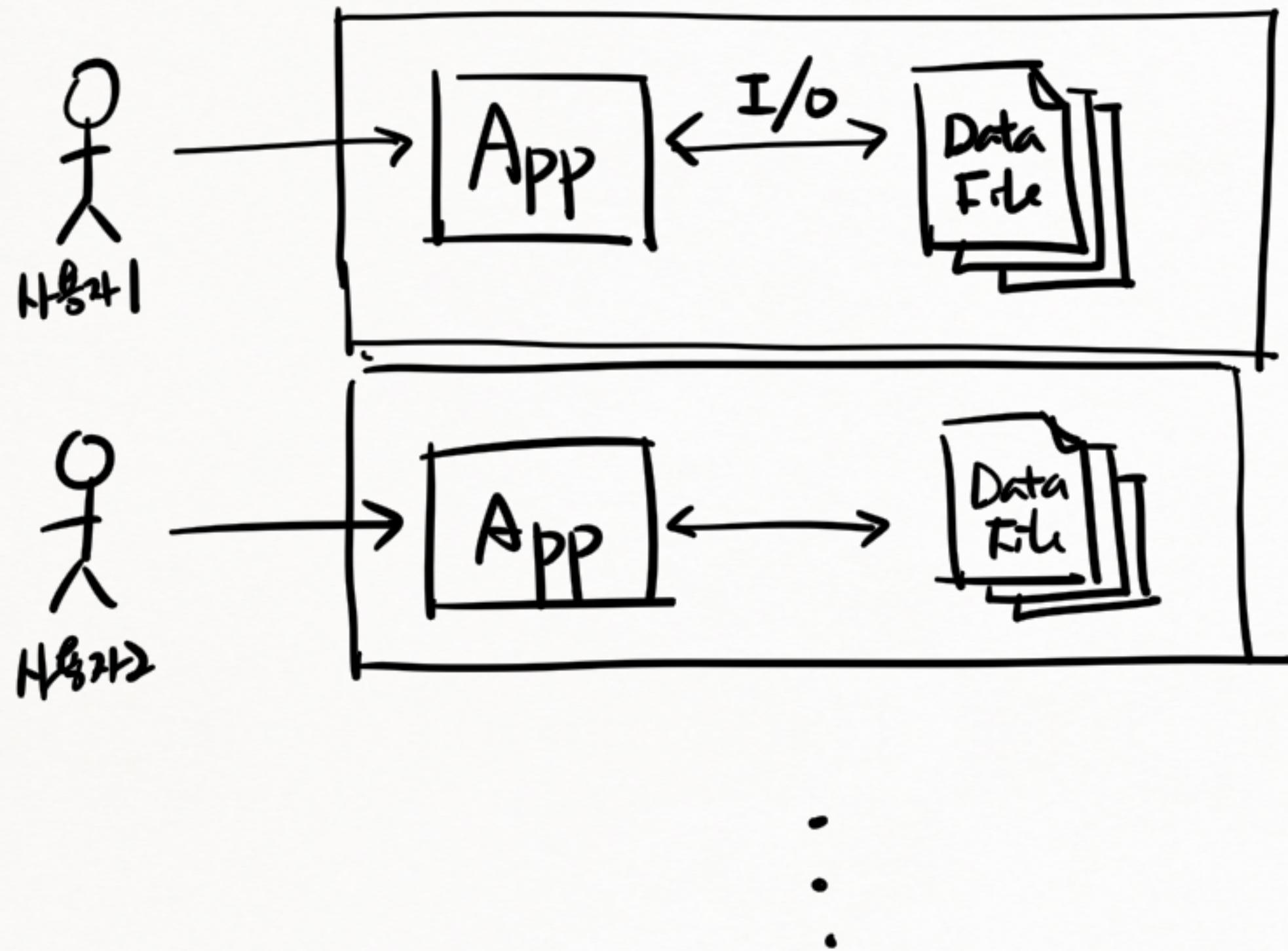


DAO 만들기



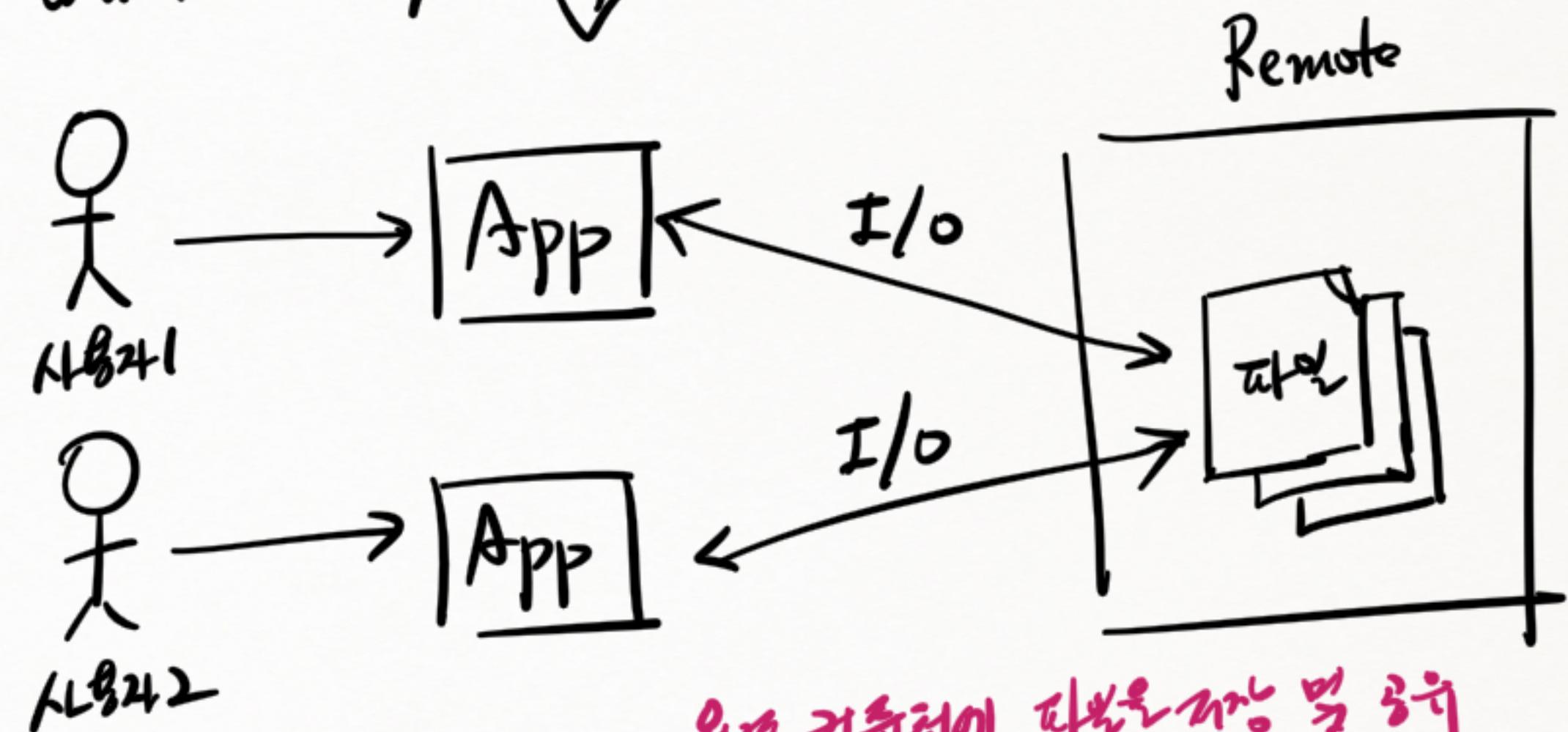
37. 바이오킹을 통한 데이터 공유

① 현황



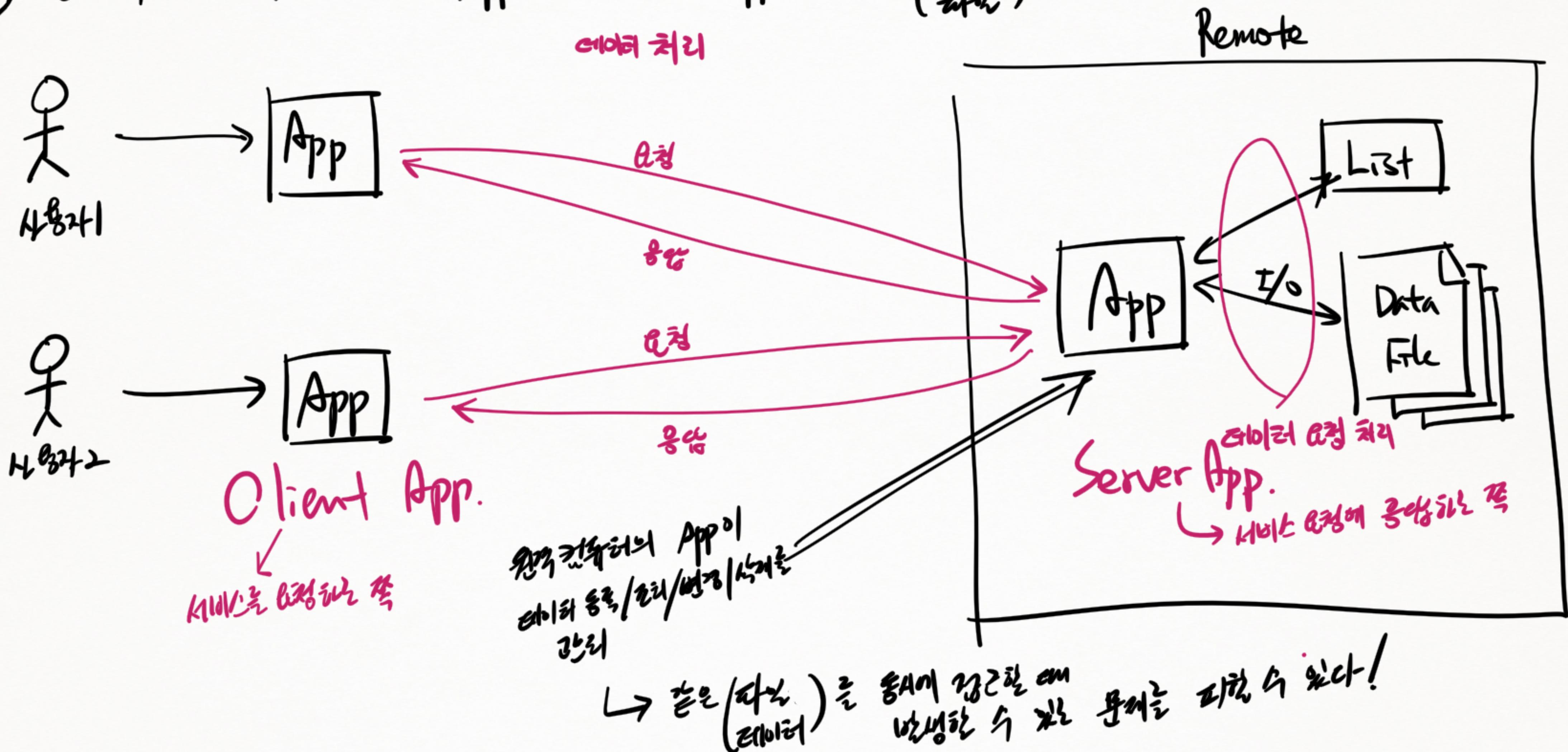
App을 실행하는 컴퓨터
데이터 파일이 로컬에 저장된다.
↓
App 간에 데이터를 공유할 수 없다!

② 데이터 파일은
별도의 컴퓨터에 분리/공유
가능할까?



- 원격 컴퓨터에 파일을 저장 및 공유
- 동시에 여러 App이 같은 파일을 편집하다 보면
파일의 데이터가 깨질수 있다.

③ 데이터 관리 기능을 별도의 App. 으로 분리 - App이 직제 (데이터)
를 접근하는 것을 막는다.



* System Architecture : Client / Server Architecture
(C/S Architecture)

