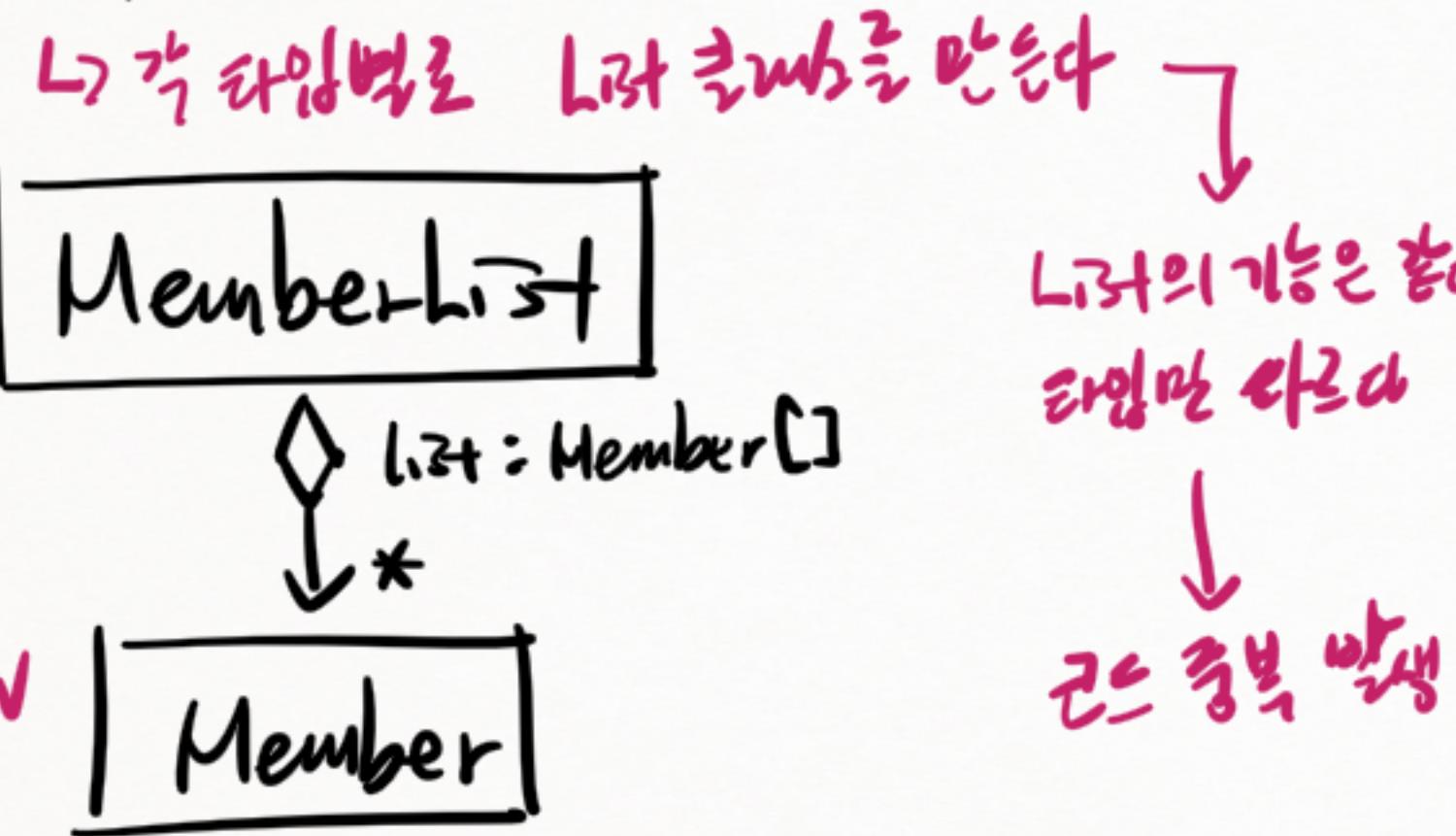


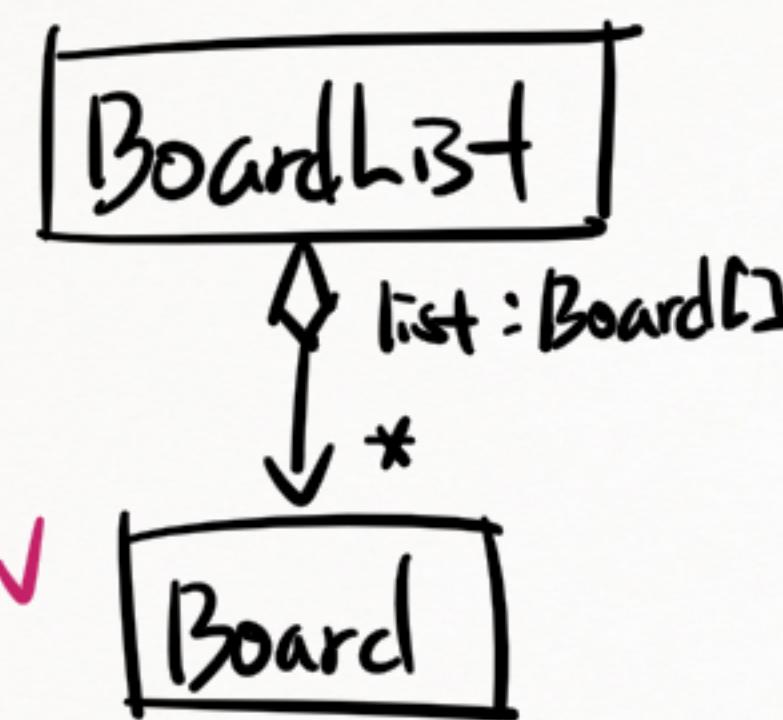
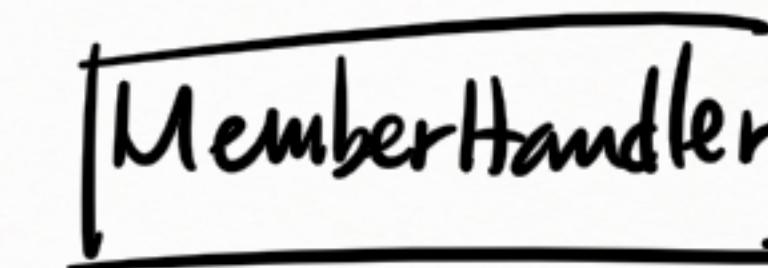
24. 제네리ك (Generic) 적용 : (object 타입처럼 다양한 타입에 대응할 수 있다.)  
특정타입으로 제한할 수 있다.

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.



. Member  
. Board  
:  
:     {  
    } 다양한 타입의  
    인스턴스(즉)를  
    제작할 수 있다.

=> 타입별로  
List를 만들 필요가  
없다!  
↳ 타입 안정성을 해친다.

\* Generic  $\Rightarrow$  Type Parameter

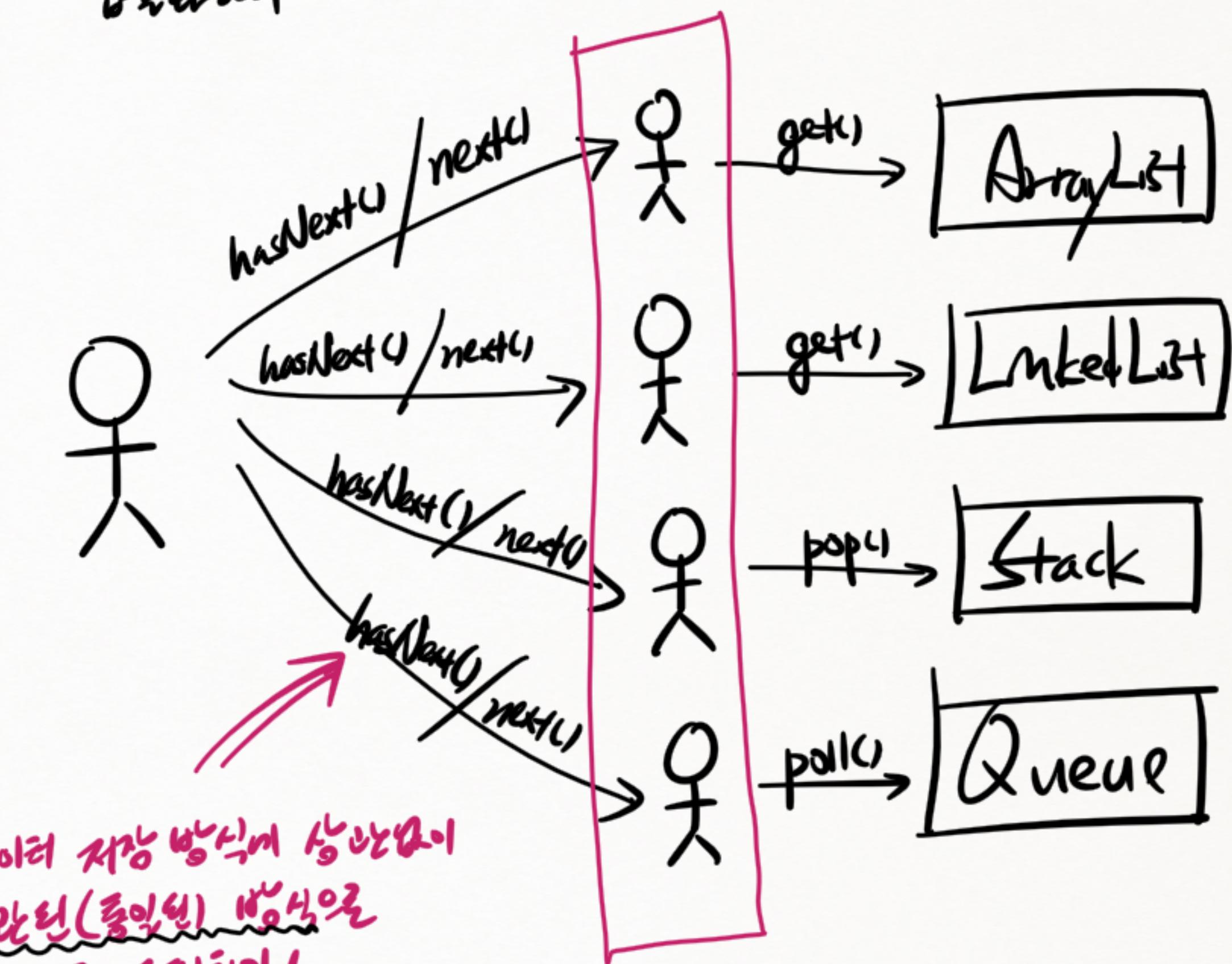
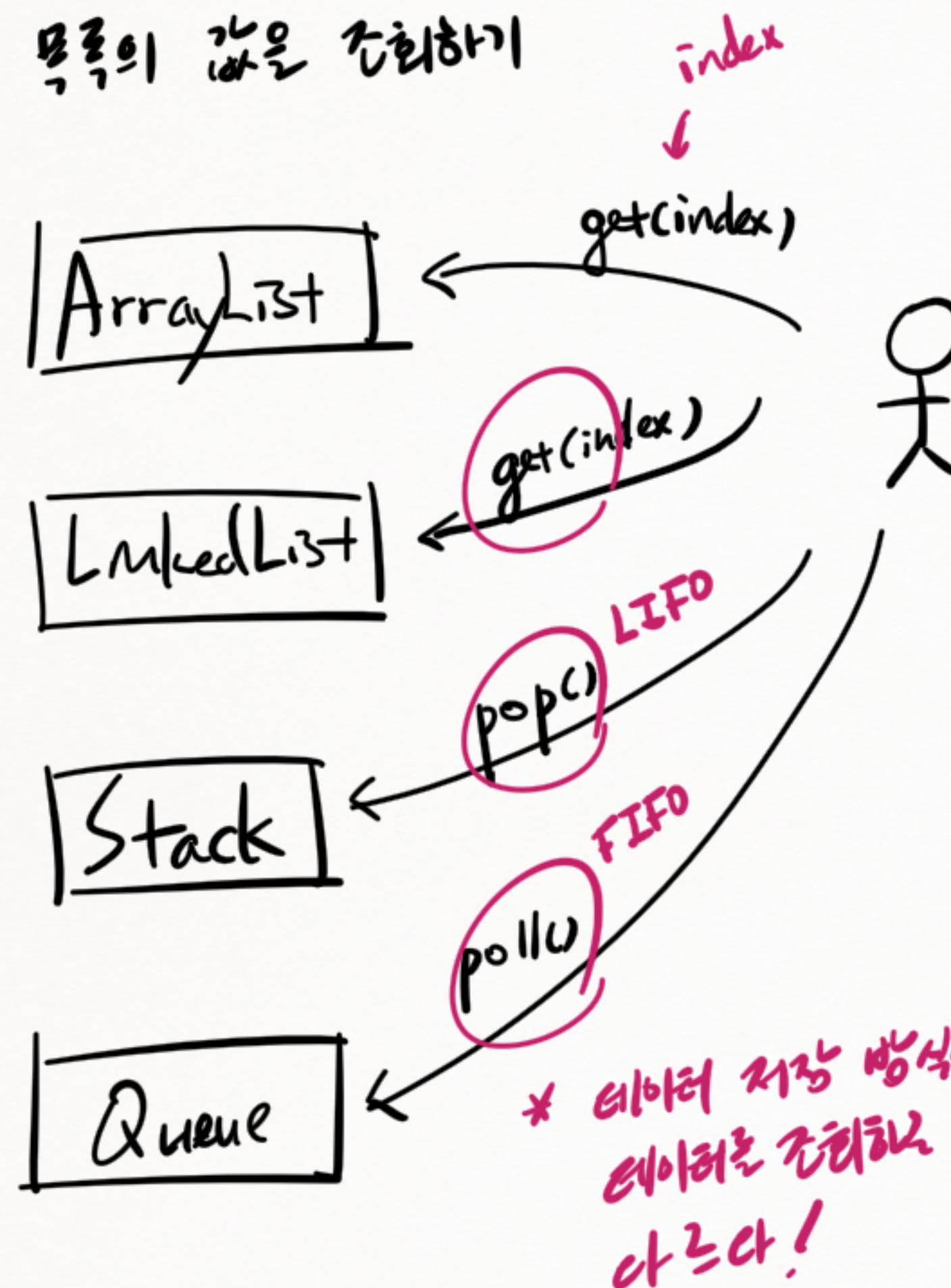
클래스의 타입 변수를 넣는다

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

타입 이름을 넣을 변수 = "Type Parameter"

## 25. Iterator 대신을 활용하여 iterator 처리기능을 기존에 쓰던화하기 → 변환화하기

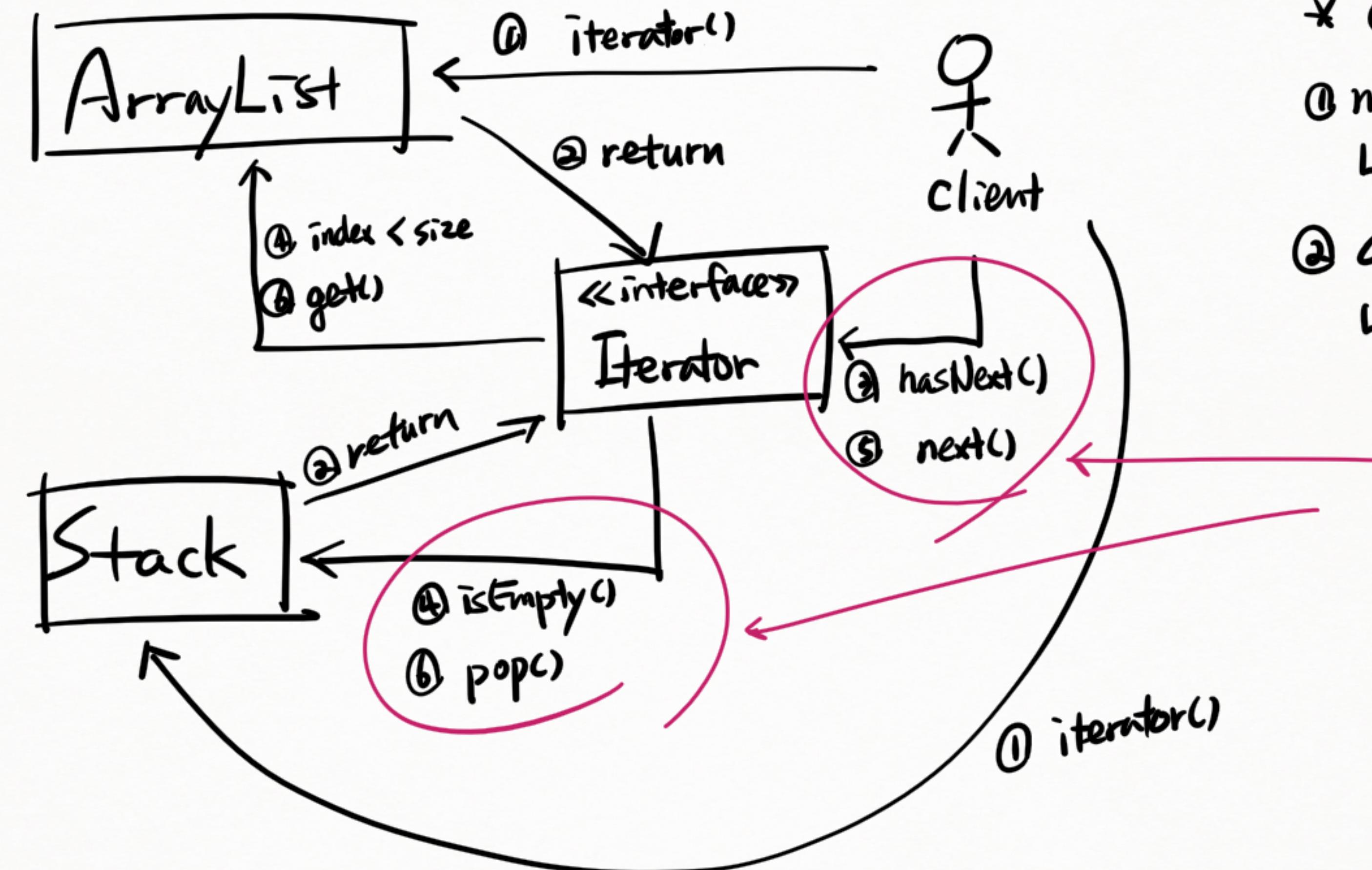
① 목록의 값을 조회하기



\* 데이터 처리 방식에 따라  
일반인(중복인) 방식으로  
데이터를 조회하기!

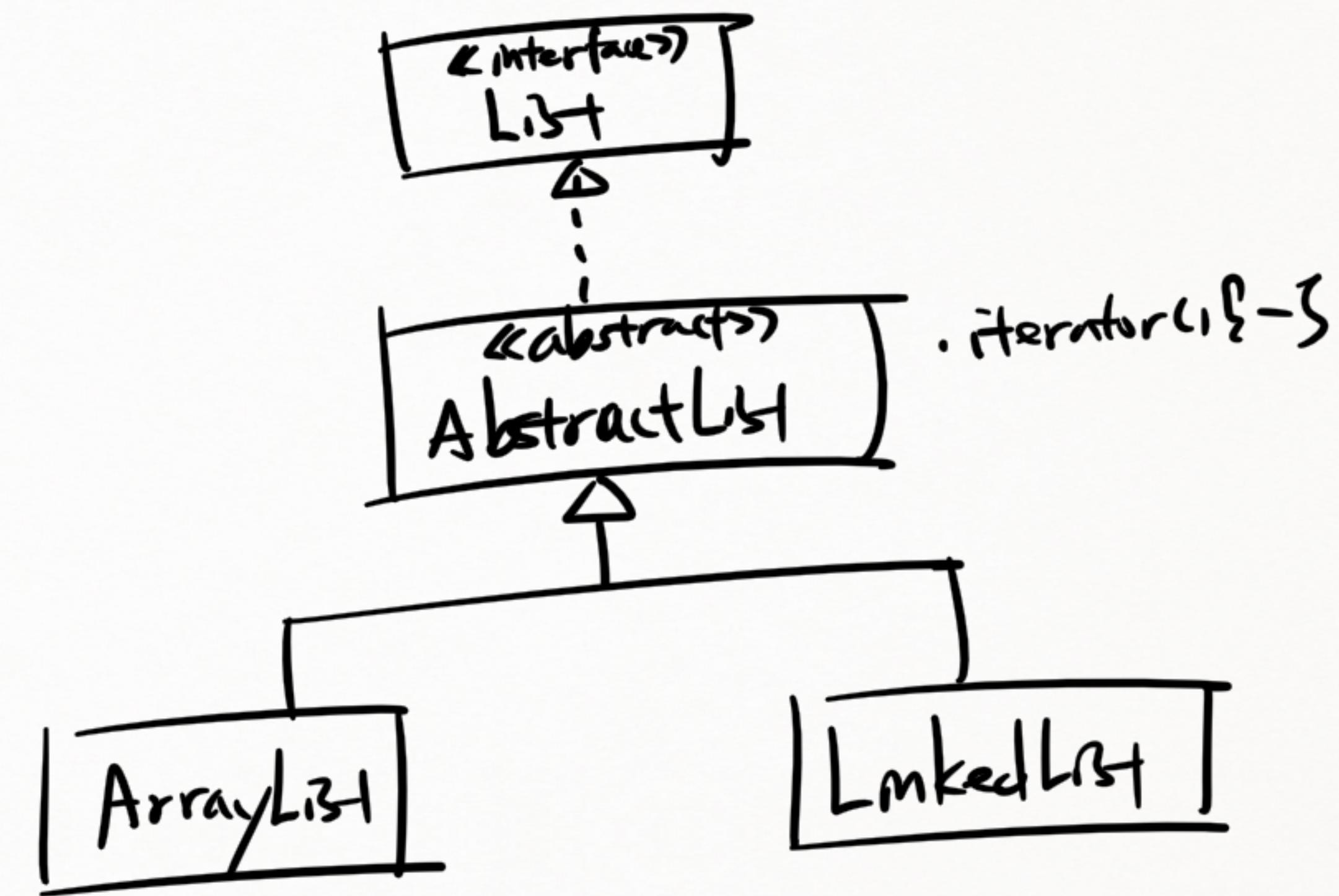
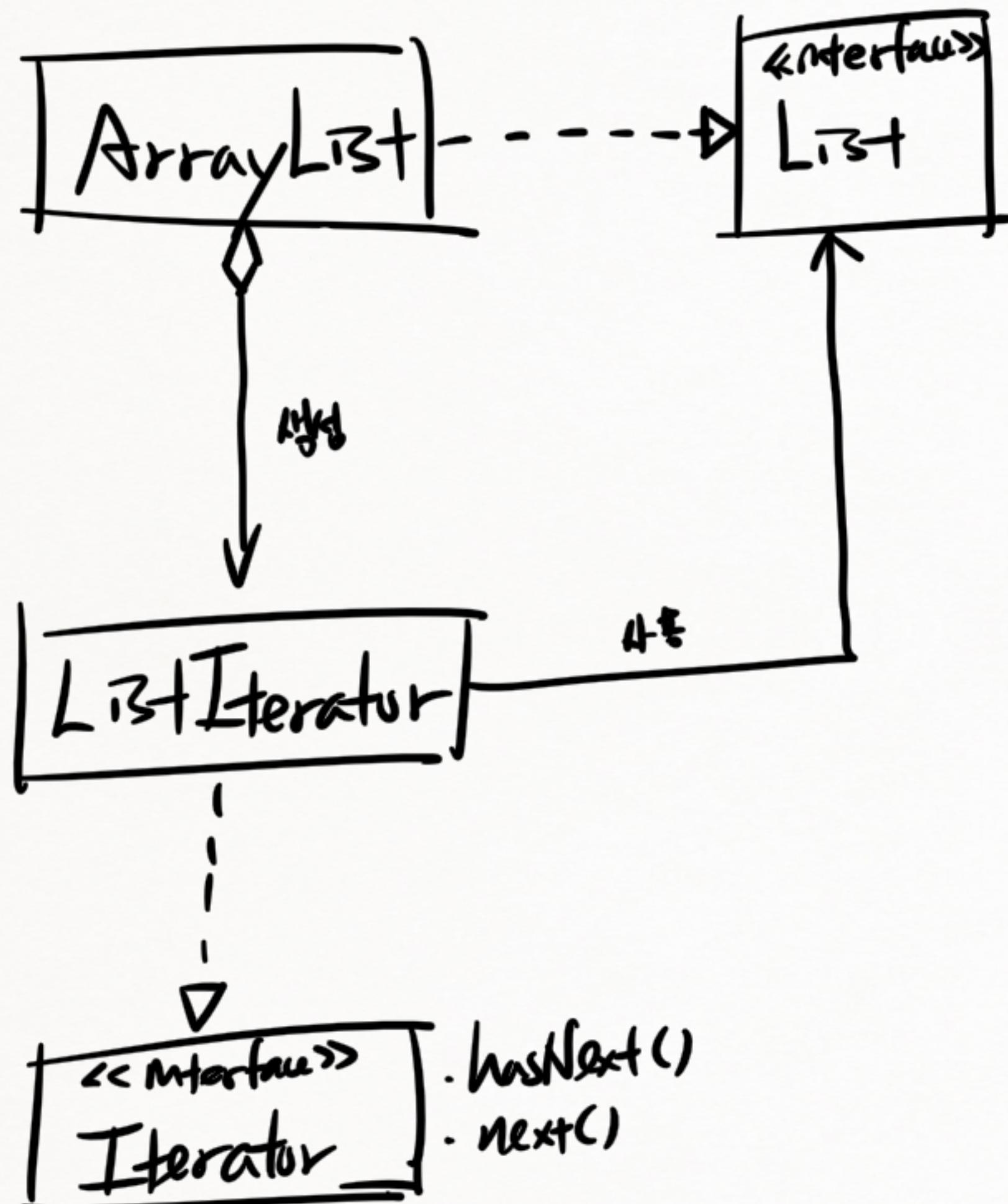
Iterator  
(데이터를 순회하는 일을 하는 객체)

## \* Iterator mechanism (구조원리)



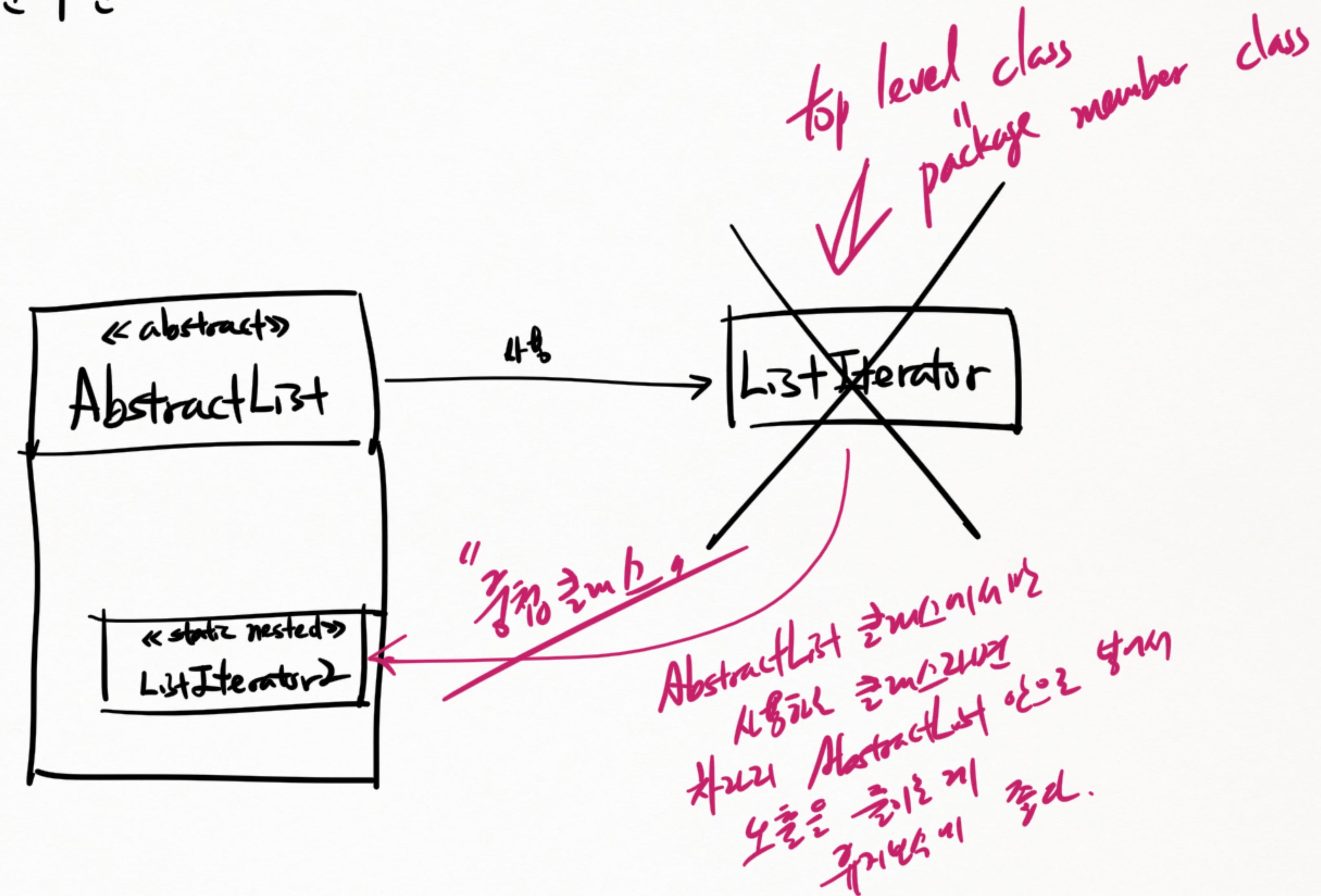
## \* Iterator 퀘션 구조

① 퀘션 - top level class



## \* Iterator 퀘션 구현

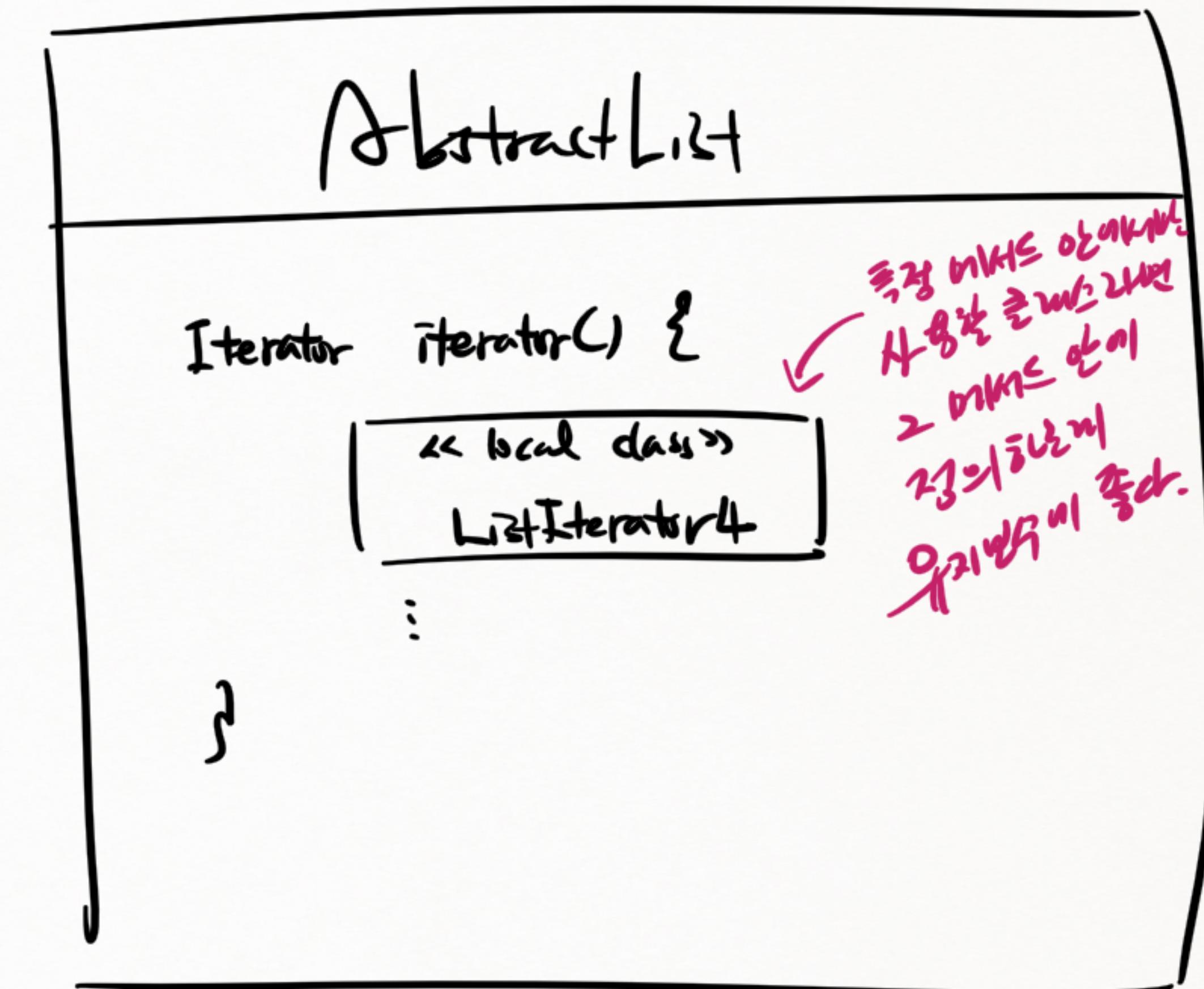
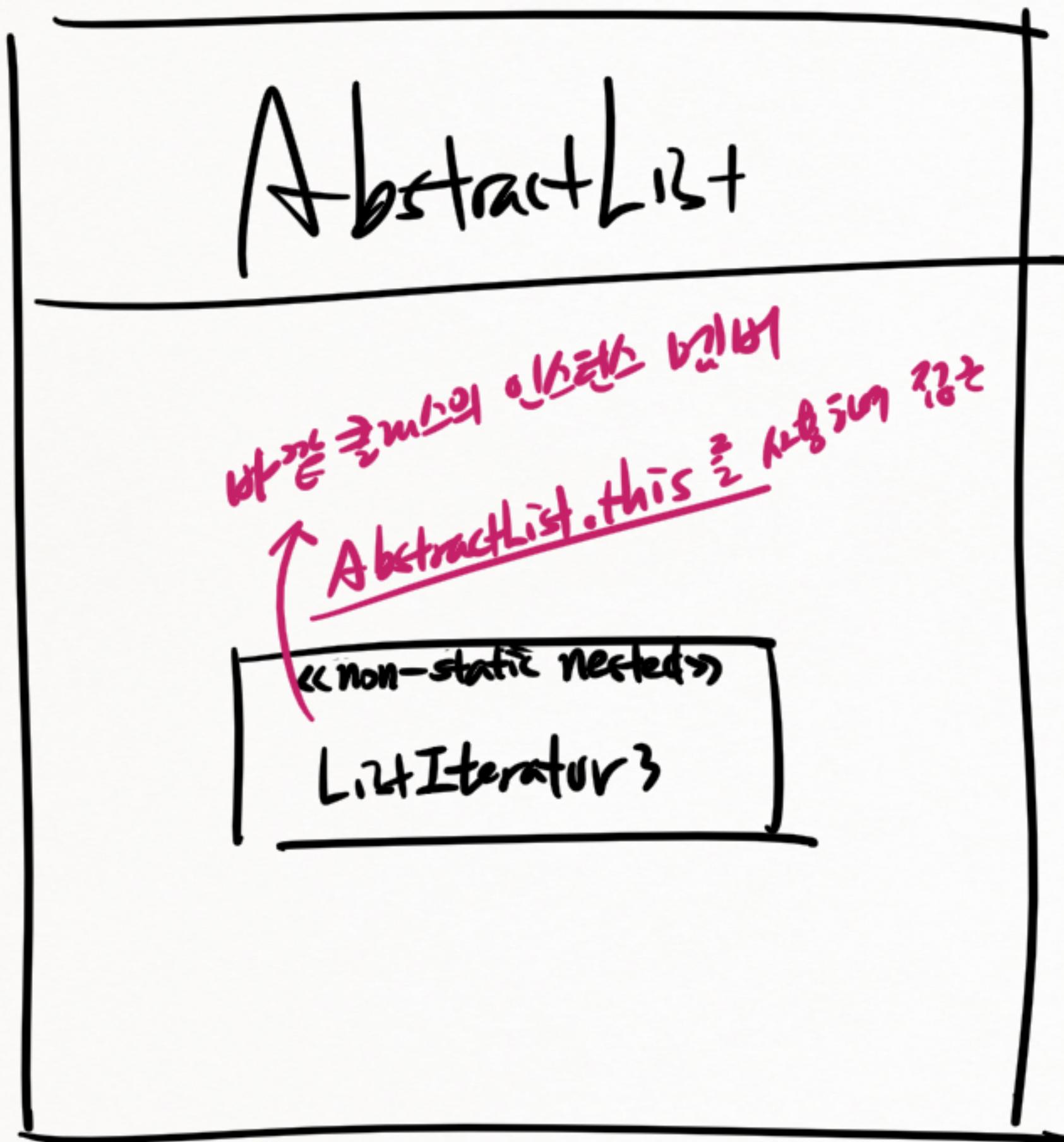
② 구현 — static nested class



\* Iterator 파편 구현

③ 구현 - non-static nested class

④ 구현 - local class



## \* Iterator 파편 구현

### ⑤ anonymous class

