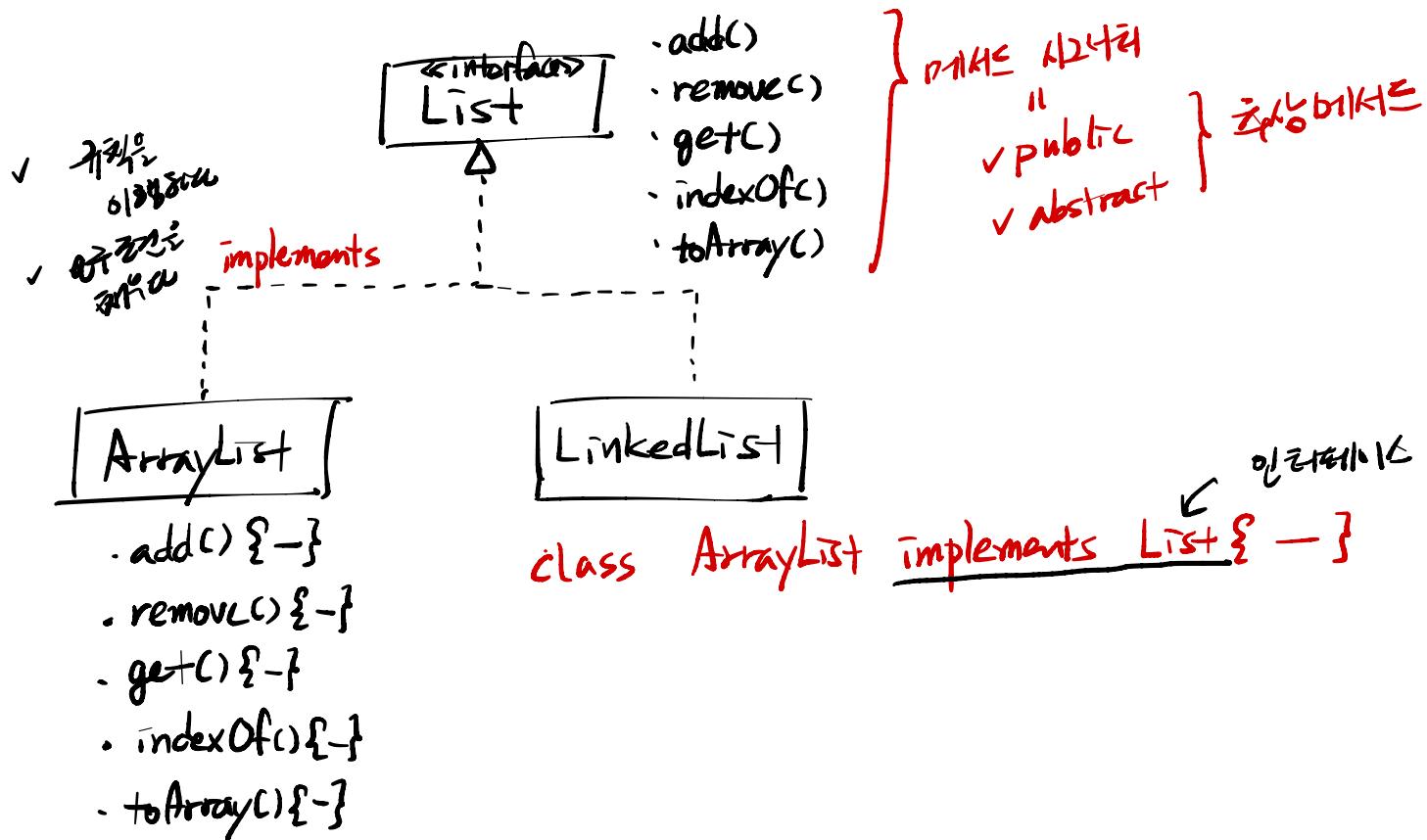
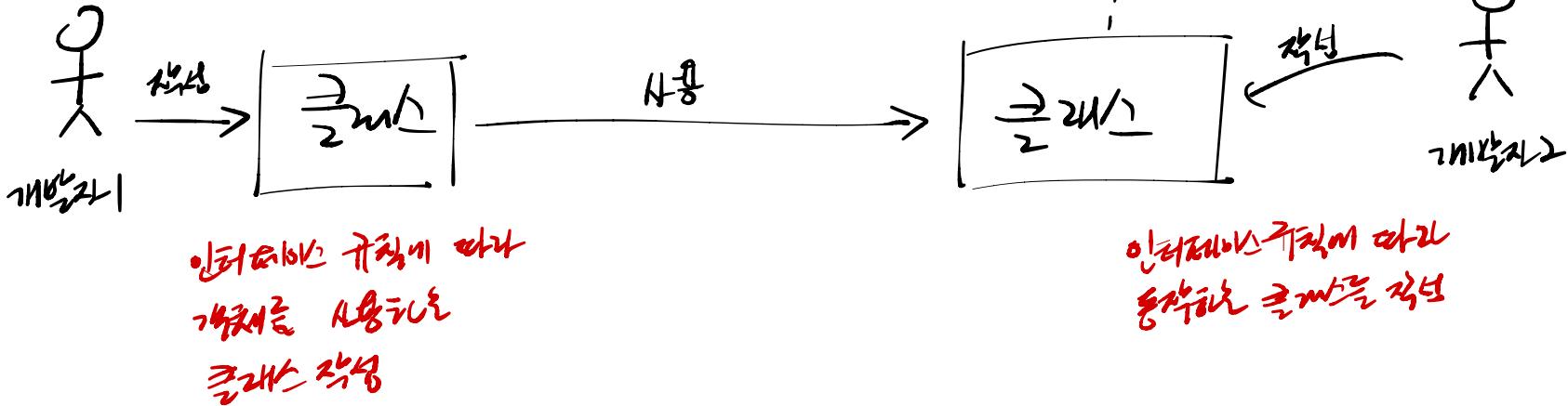
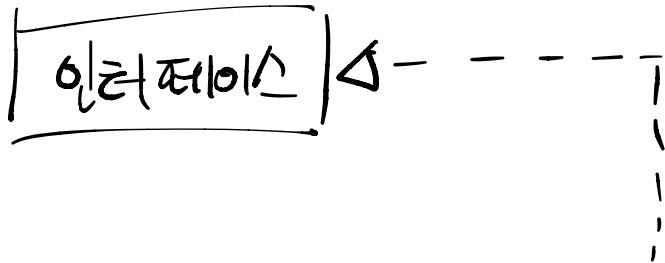


17. 인터페이스를 이용한 구조화된 접근



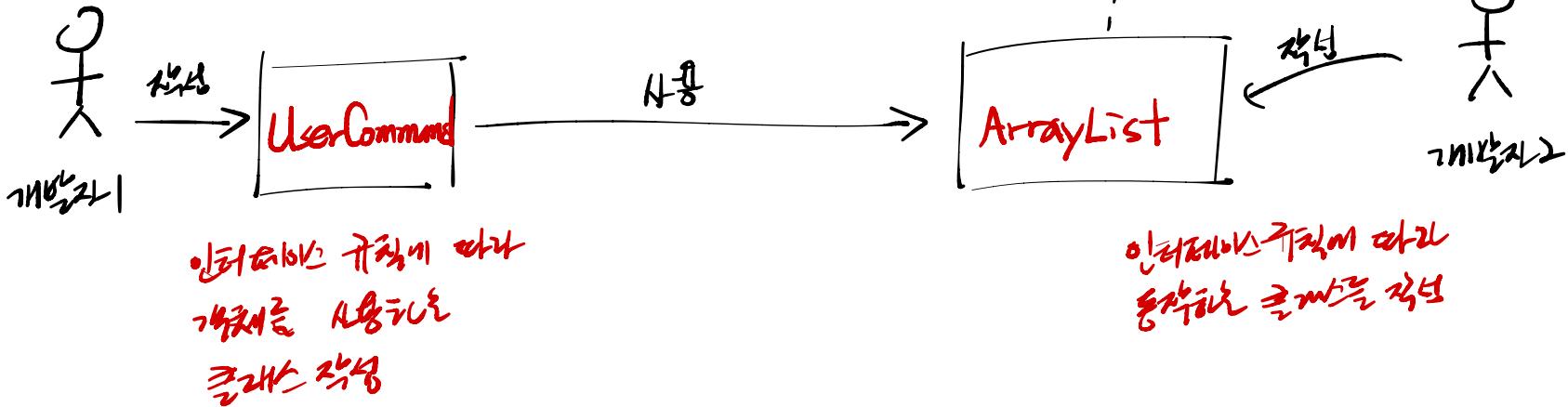
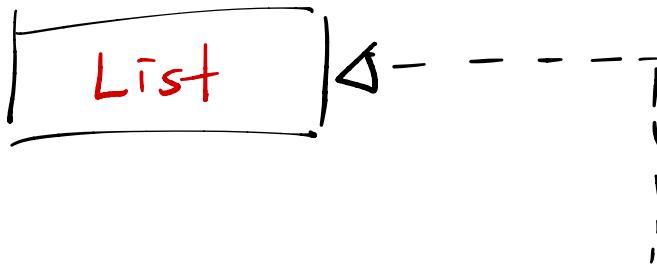
* 인터랙이스

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

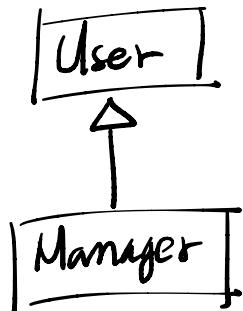


* 인터페이스

인터페이스는 구현체 없이
제공하는 기능의 만족도가 높다.



* instanceof vs getClass()



equals() {
 }
 ==
 }

equals(Object obj) {

```
if (this.getClass() != obj.getClass()){\n    return false;\n}
```

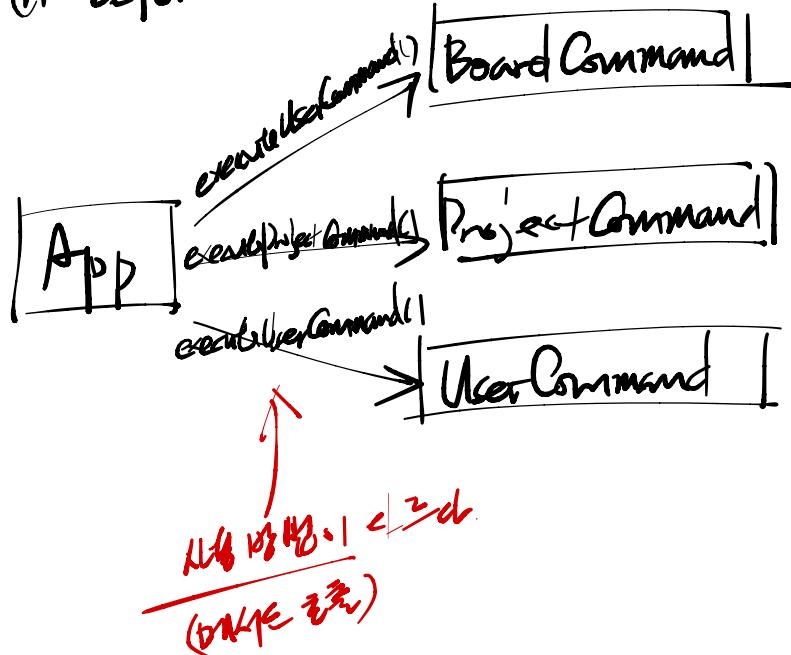
2

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

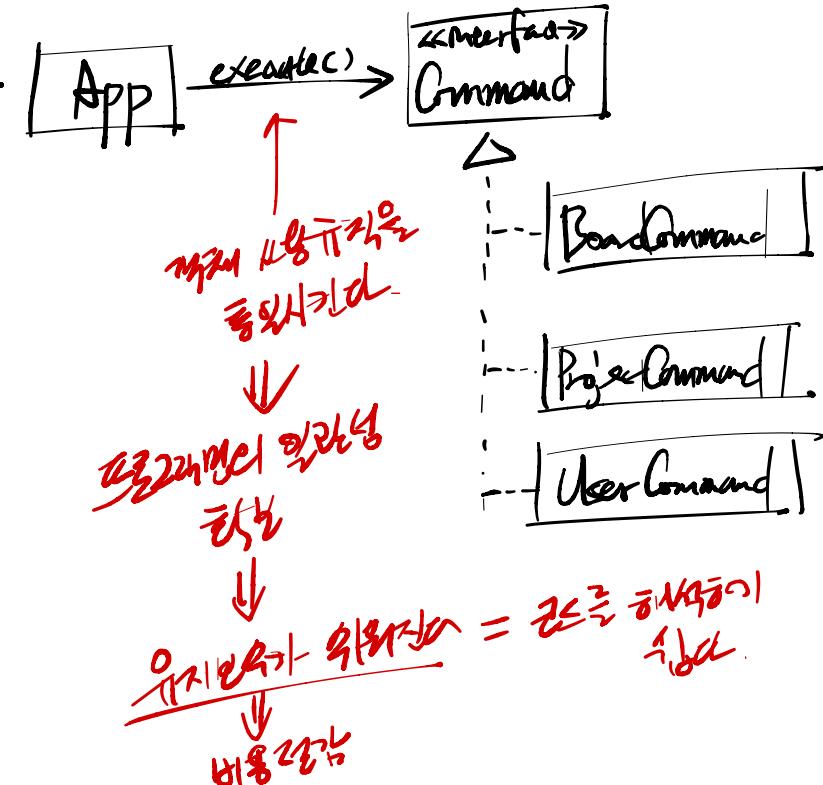
s	list 1	list 2
<code>ul.equals(str)</code>	F	F
<code>ul.equals(ul)</code>	T	T
<code>ul.equals(m)</code>	F	T

* 121 능력과 122 번의 학습자료

① before

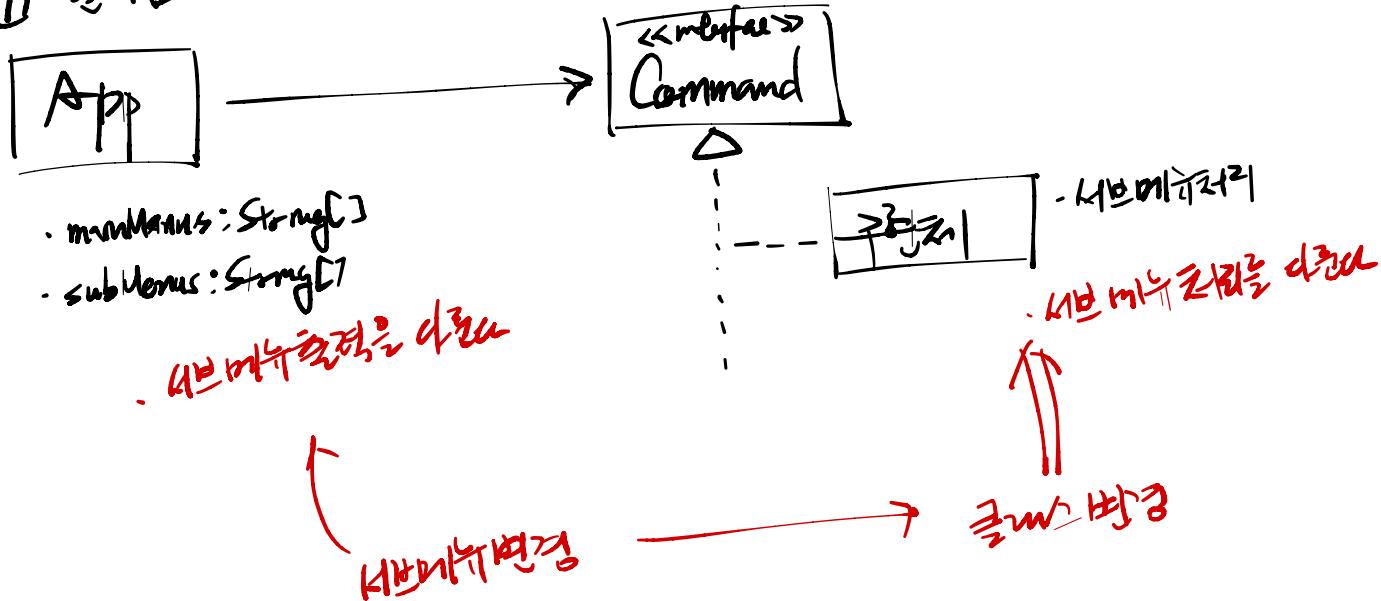


② after



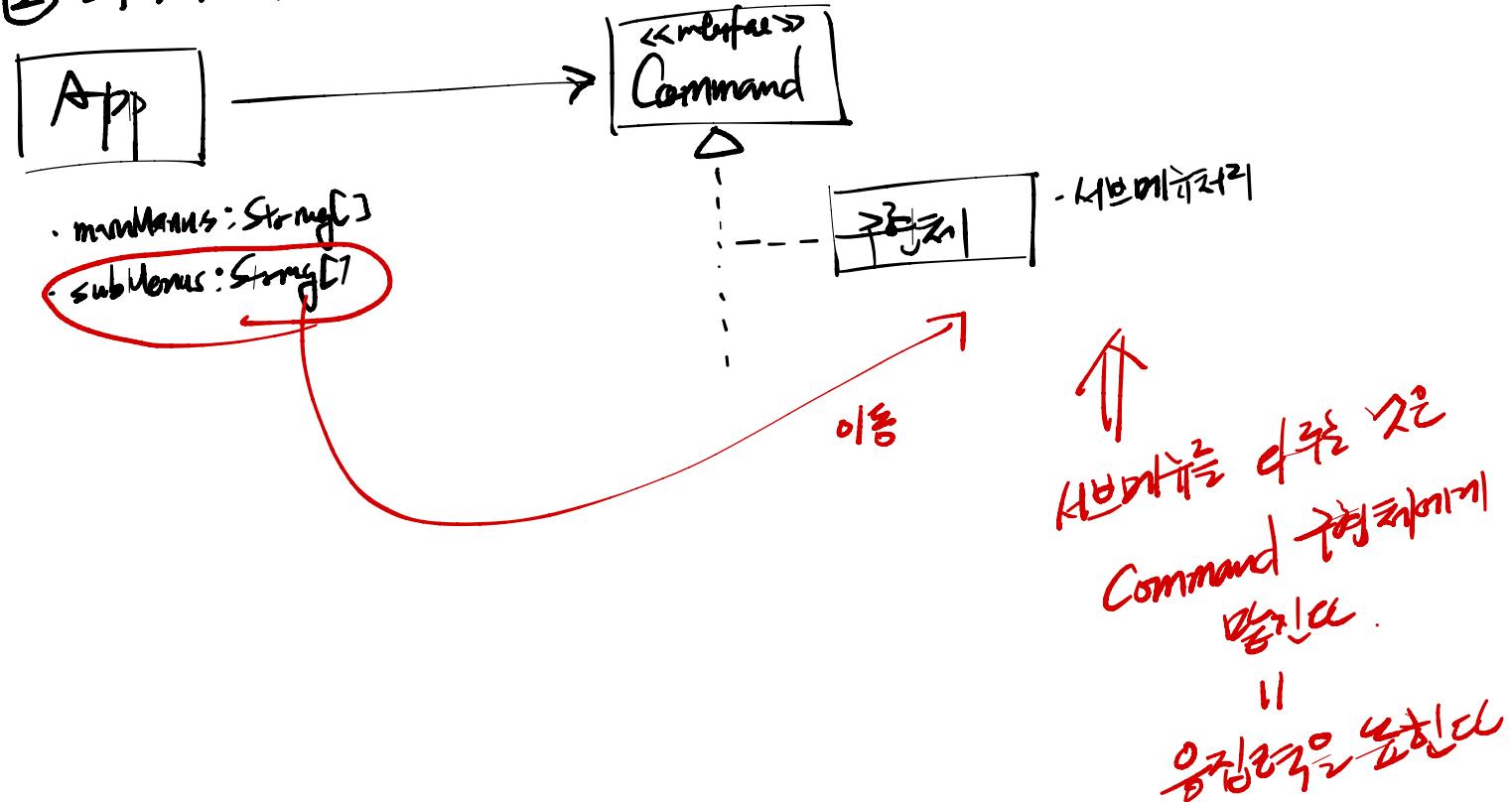
18. 리퍼팅

卷之三



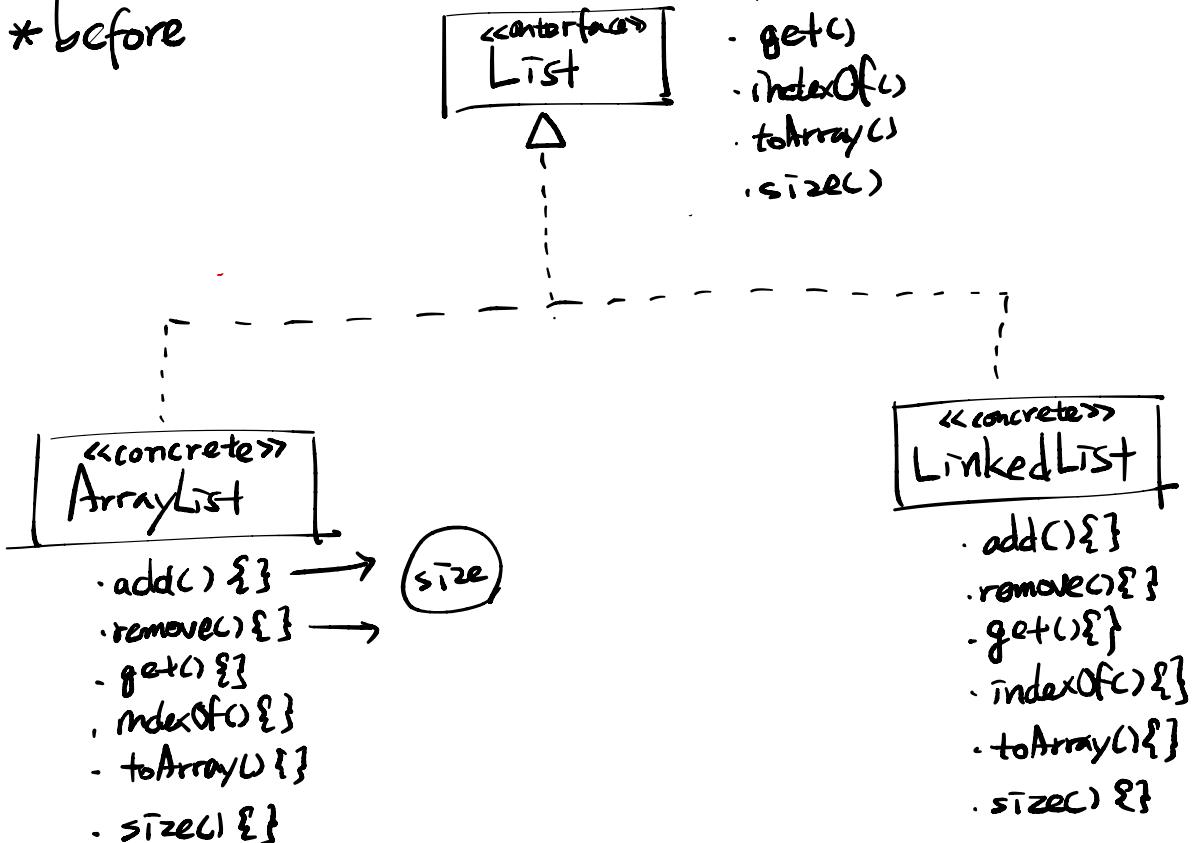
18. 디자인 패턴

② 명령 : GRASP의 High Cohesion & Low Coupling



19. 상속의 Generalization - ①

* before



19. 상속의 Generalization - ①

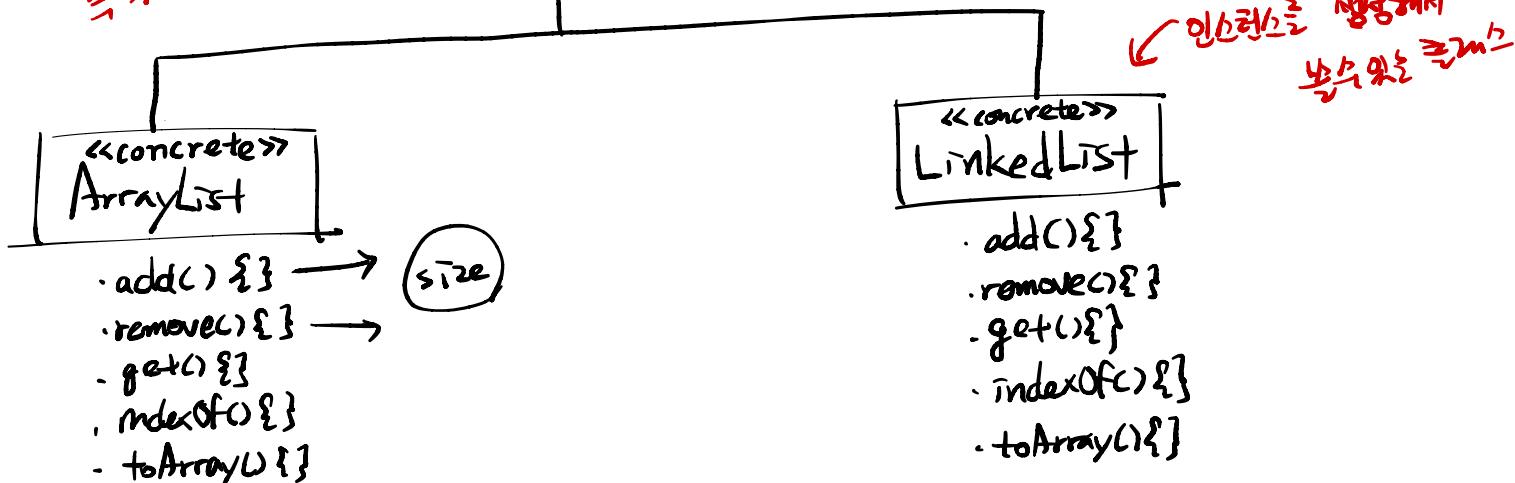
* after

①

상속을 통한
공통 기능을 모아두는
상위 클래스를 만드는
것은 좋음
하지만
이걸로 사용자가
직접 접근하기가!

②

인스턴스 사용하기
조금 더 편리해지겠지!



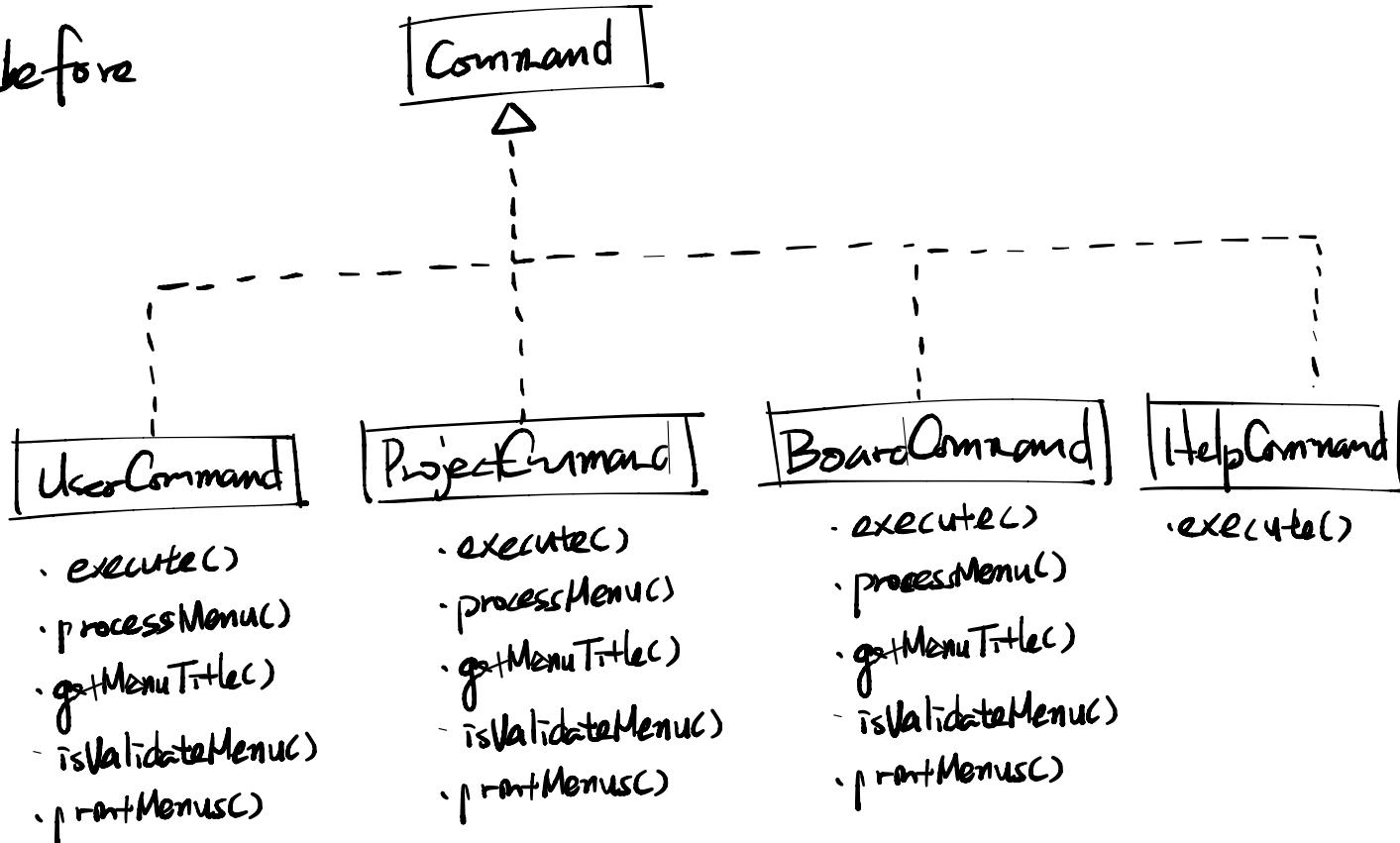
- add()
- remove()
- get()
- indexOf()
- toArray()
- size()

- **size: int**
- **size() { } }**

- **add() { } }**
- **remove() { } }**
- **get() { } }**
- **indexOf() { } }**
- **toArray() { } }**

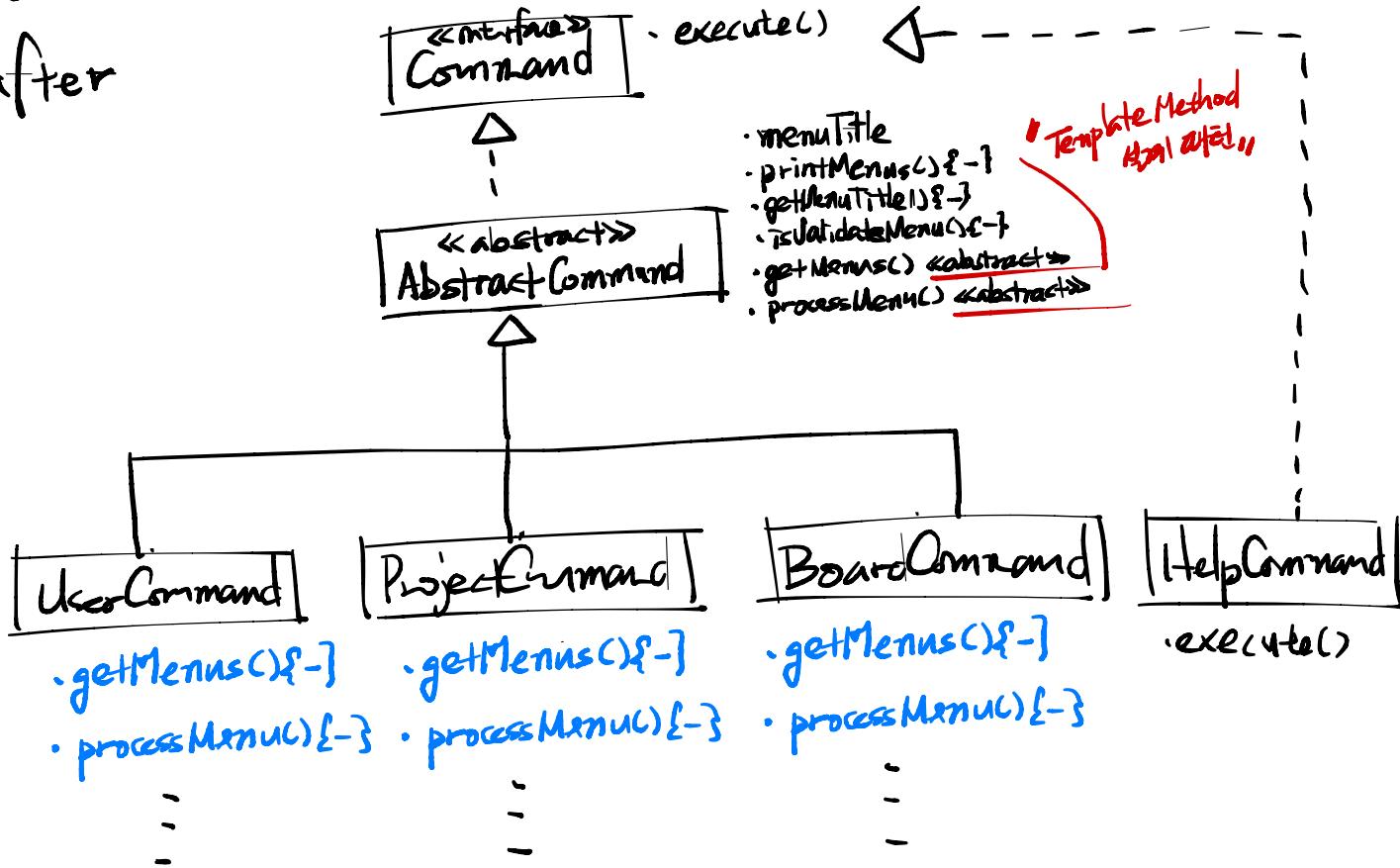
19. 상^k으로 Generalization - ②

* before

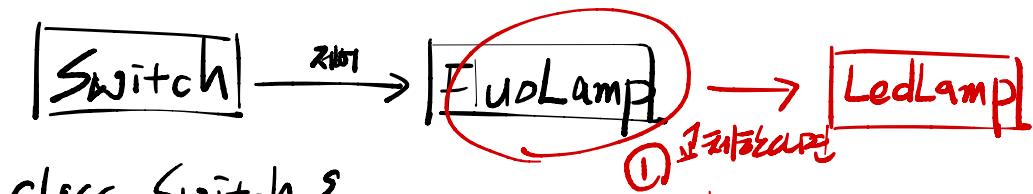


19. 一般化 Generalization - ②

* after



20. SOLID의 DIP와 GIRASP의 Low Coupling



class Switch {

 FluoLamp light;
 \equiv ② 반드시 연결해야 함.

③ Switch 클래스가

FluoLamp 클래스와

상호로 연결되어야

④ "강한 단계 관계" \Rightarrow 해결?

20. SOLID의 DIP와 GIRASP의 Low Coupling

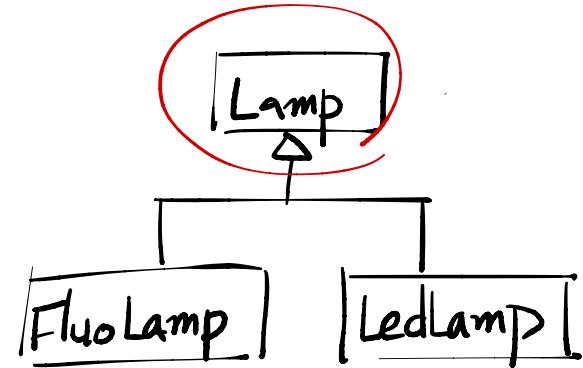


class Switch {

Lamp light;
 }
 =
 ② 아름다운 예술을 활용하여
 여러 종류의 형상을
 제작할 수 있다

③
Lamp
가정만 제거?
Lamp

⇒ ④ 소위 카드 다른 제품을 제작할 수 있어야 하는가?



① 두 클래스의 부모를 공유하는
→ 같은 작업으로 뷰으면

20. SOLID의 DIP와 GRASP의 Low Coupling



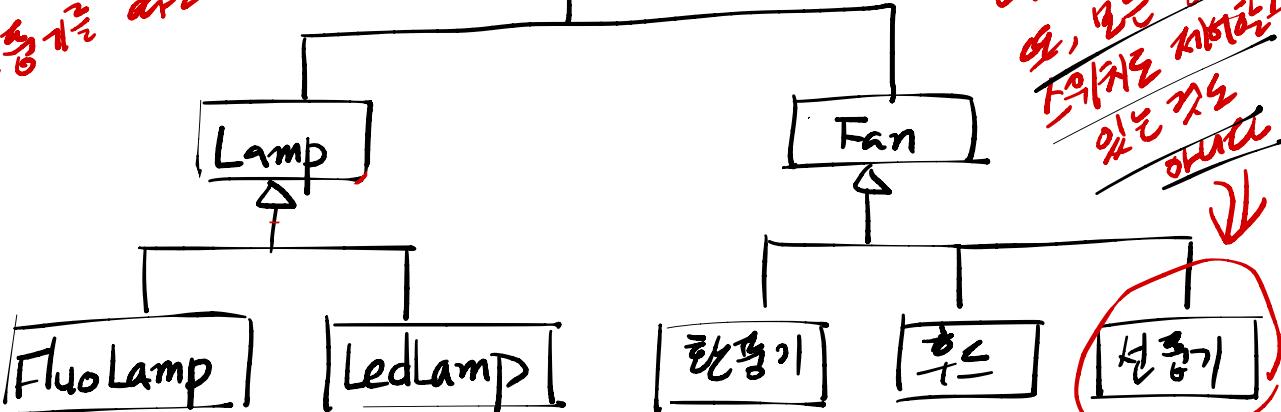
class Switch {

Device 장치;

}

② 모든 클래스를 스위치로 연결하는게 아니라
선택적으로 연결하는게 맞다.

① 여러 장치를 스위치로
연결하는건 고급화로 무관한데
같은 태깅으로 묶어보면
더욱 편리할 것이다
그리고 선택적으로
연결하는게 맞다
여기서도 선택하는
것은 괜찮은 선택이다



③
상속은 "캡슐화"의
유지는데에도,
유연성 부족.

20. SOLID의 DIP와 GIRASP의 Low Coupling

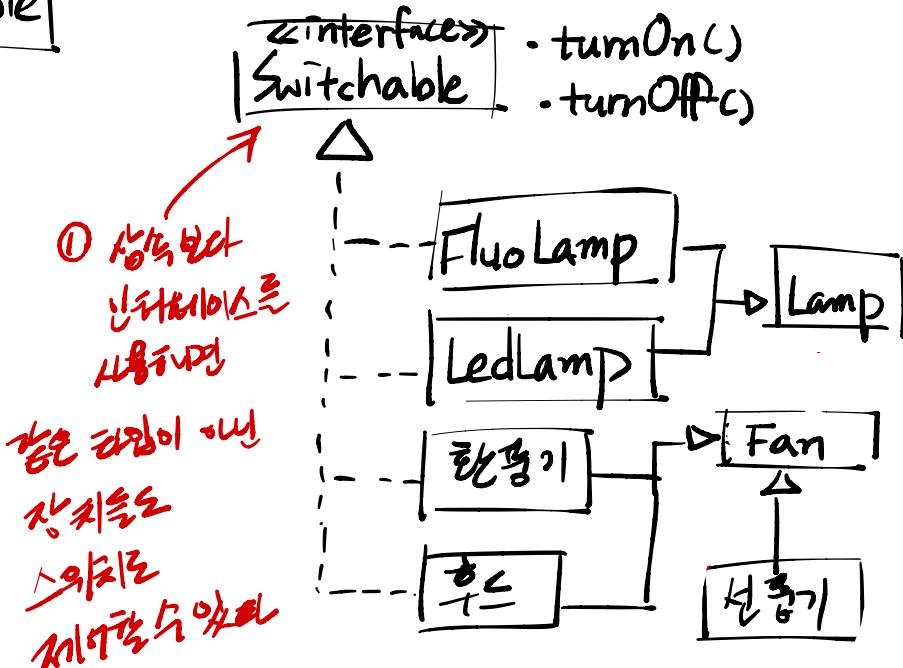


class Switch {

 Switchable 장치;

}

② 어떤 타입의든
Switchable 구조에
적합하는 것을 알아내기
가능하게



↳ ② 쌍방향 인터페이스로
“약관화” = 느슨한 연결

Low Coupling

* SOLID - Dependency Inversion Principle (DIP)

↳ 의존 구조를 확장 만들지 않고

외부에서 주입 받는 방식.



class Switch {
 Switchable 광고;

① 노드한 광고로
전환한 후

FluoLamp light1 = new FluoLamp();

Switch switch = new Switch(light1);

② Switch가 의존 구조를
설정하는 것 아니고
외부에서 주입 받는
방식으로 전환하면

- ③
- ✓ 광고가 된다
 - ✓ 광고가 된다.

↳ 의존 구조를
간단히 만들어 주입하는
스위치의 용도를 헤아릴 수 있다.

이 유연해진다

* DI

SOLID

Dependency
Inversion
Principle

(의존성 역전 원칙)

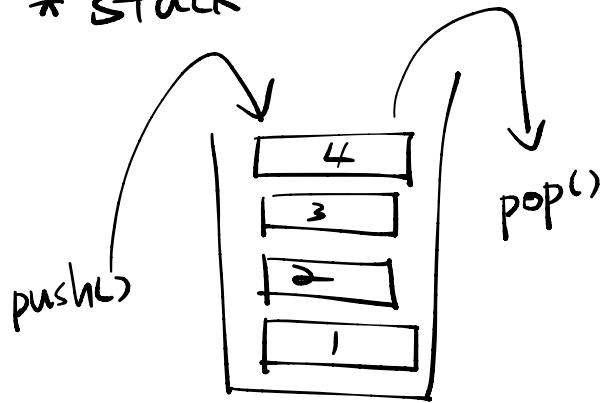
(제어권역)

Inversion of Control (IoC)

- ① Dependency Injection
② Listener = event handler

21. 자료구조 - Stack 와 Queue

* stack

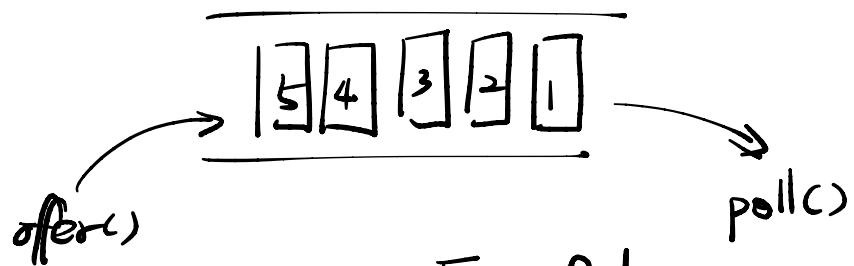


First In Last Out
(FILO)

"
Last In First Out

(LIFO) ① 뒤로나오기 ② 앞으로나오기
③ 예상외나오기

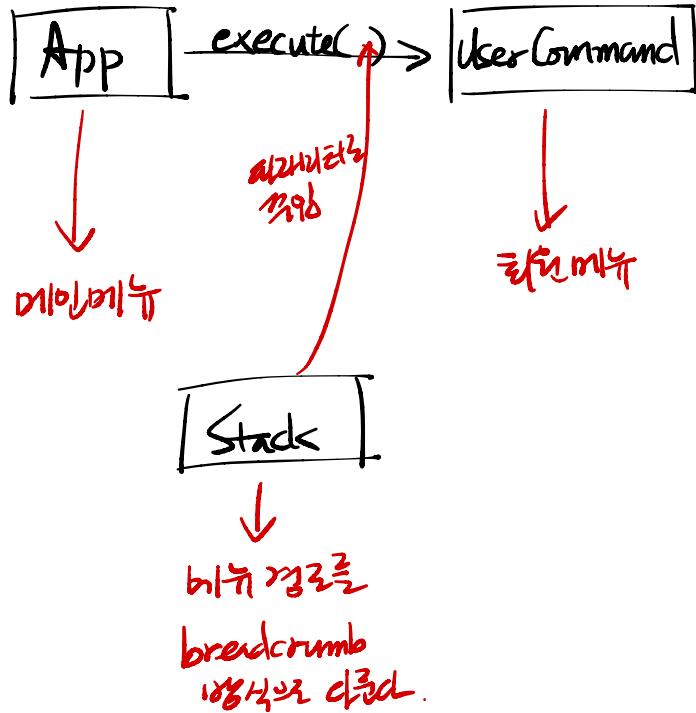
* Queue



First In First Out
(FIFO)

- ① 예상
- ② 앞으로나오면서 큐를 나온다 = 정상 처리
- ③ 이벤트 처리 = Event Queue

* 디足迹 제목을 소리으로 하기



* String

String str = "";

str += "aaa";

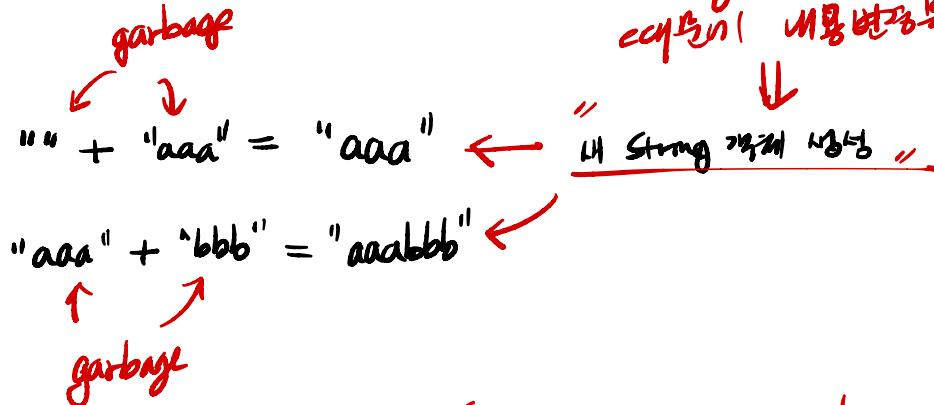
str += "bbb";

★ 왜 Java에서는
String은 오직 같은
StringBuffer는
StringBuilder는
StringBuffer와
StringBuilder는
같은가?
↳ garbage는 끊임없이
생성되고 해제된다.

thread-Safe

스레드 안전한지 알기
↳ lock/unlock
함수는 안전

StringBuffer, StringBuilder



문자열은 대량을 만들 때 String은 안전하지
않아 String->String->garbage가 된다.
문자열은 안전하다.

↳ thread-Safe

thread-Safe
↳ 스레드 안전
↳ 스레드 안전한지 알기
↳ lock/unlock 함수는 안전
↳ obj--> lock/unlock

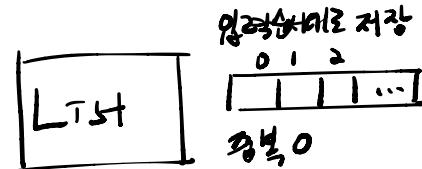
22. Iterator 깊이 파악

이전 문서화하는 강점인 하위 목록

· 재사용성↑ · 유지보수성↑ · SOLID / GRASP 부합

문서
서로 다른
데이터를
주회하는
방법이
다르다!

인덱스(정수)
get()



toArray()

Set

한시점으로 저장하는 목록

0	1	2	...

정복 X

key 가짐
get()

Map

기본적으로 값 저장

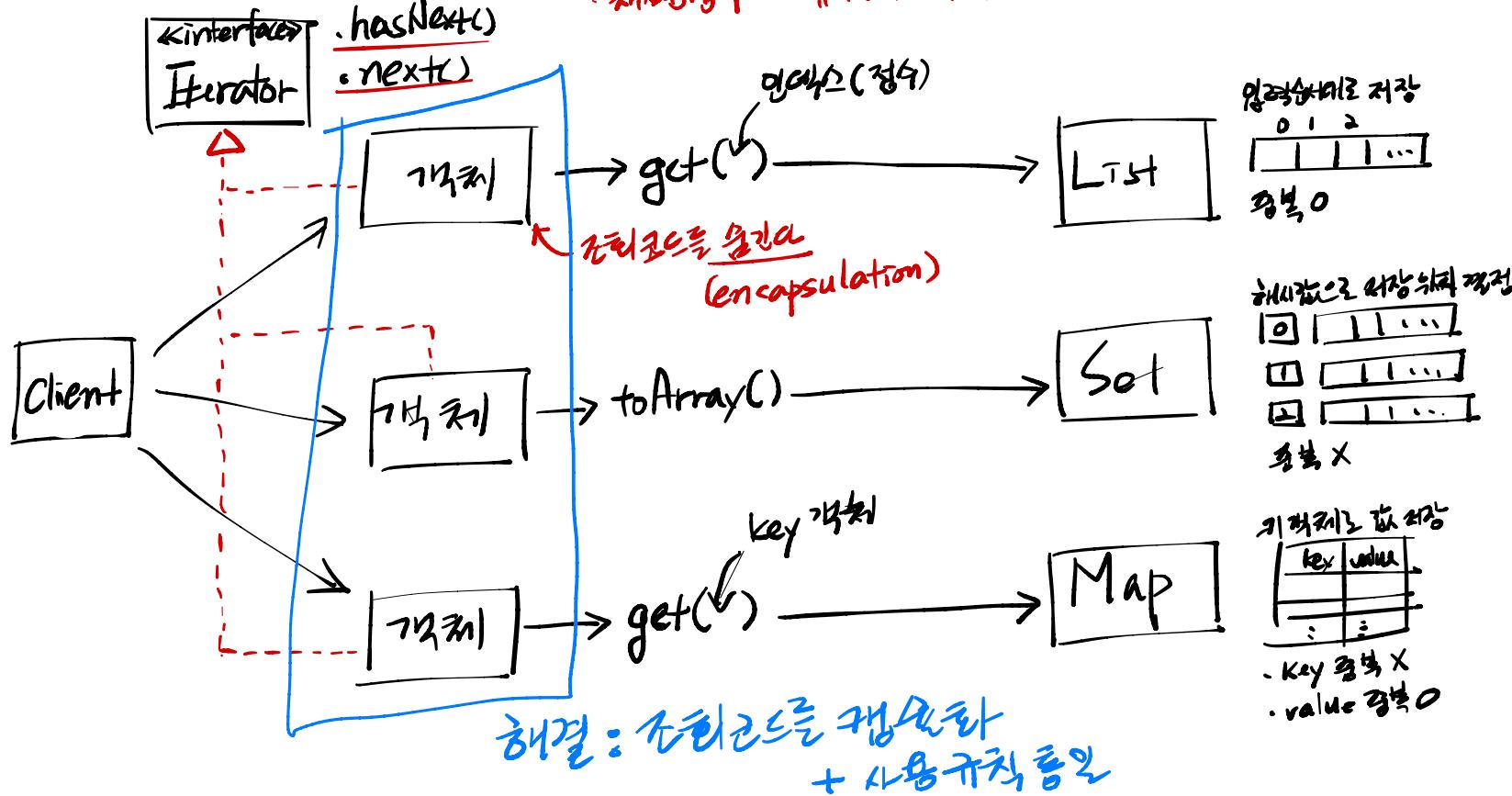
key	value

- Key 정복 X
- value 정복 O

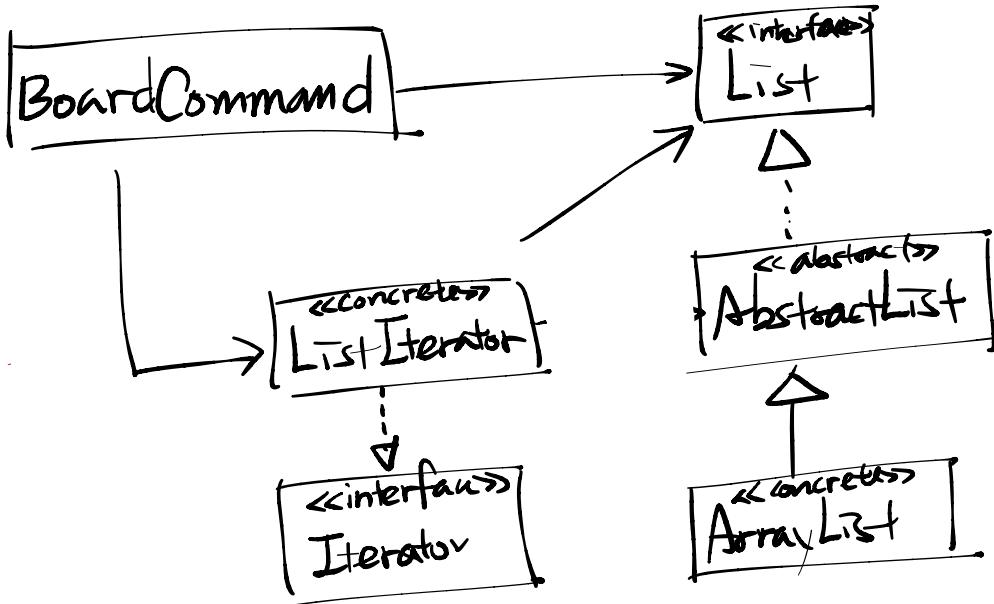
22. Iterator 기능적 패턴

이전 문서화하는 강제된 흐름 방식

· 재사용성↑ · 유지보수성↑ · SOLID / GRASP 부합

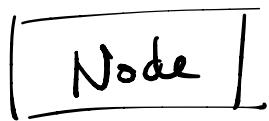


22. Iterator 틀 구조



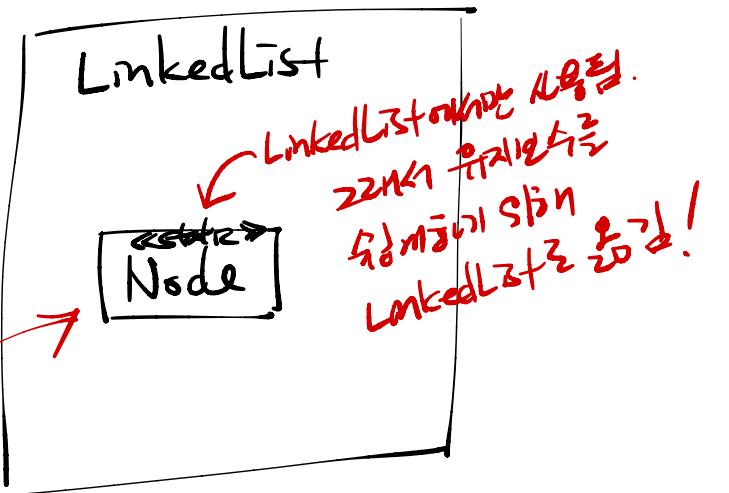
23. 중첩 클래스

before



↑
package member class

after

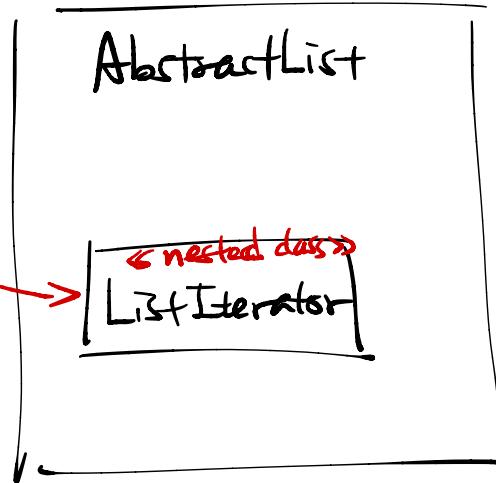


23. 중첩 클래스

before



after



* enhanced for 문법

for(변수선언 : 배열 또는 Iterable 구현체)

ex) String[] names = {"홍길동", "임꺽정", "유관순"};

for(String name : names) { }

변수

ex) ArrayList list = new ArrayList();
list.add("홍길동"); list.add("임꺽정"); list.add("유관순");

for(Object item : list) { }

Iterable 구현체

* Iterable ↗^{3연자}

