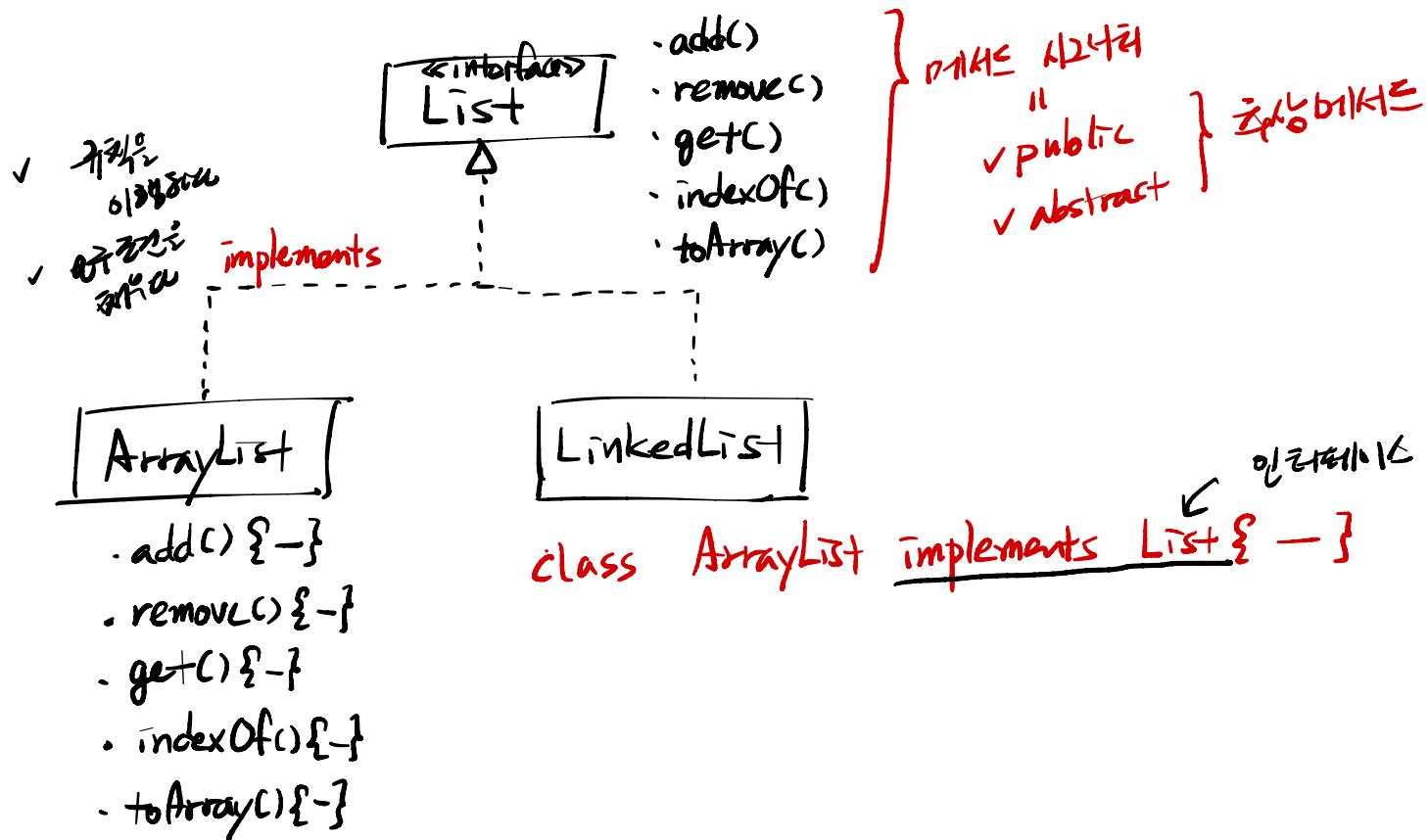
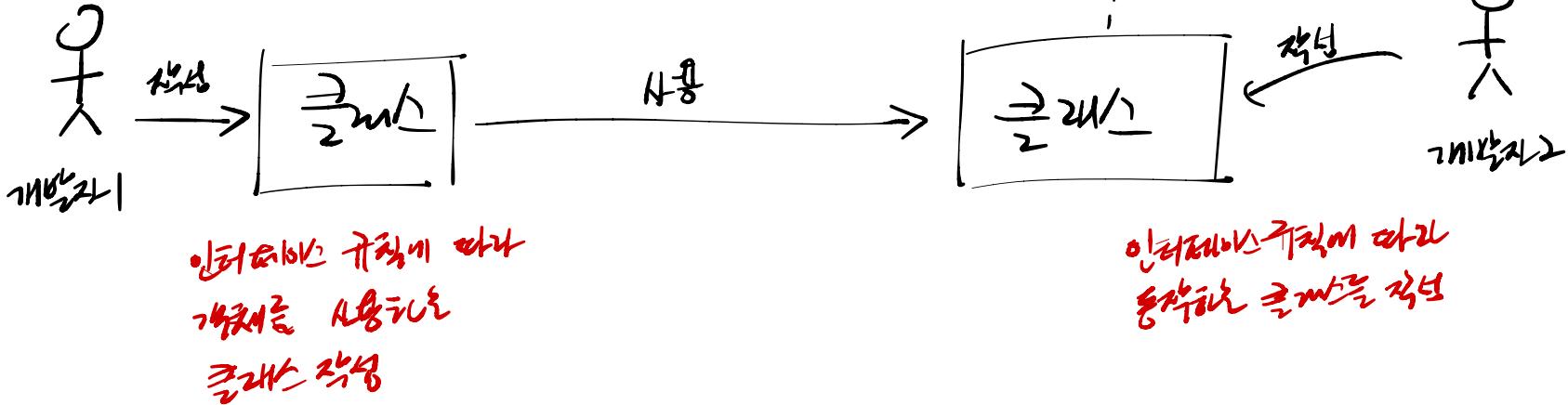
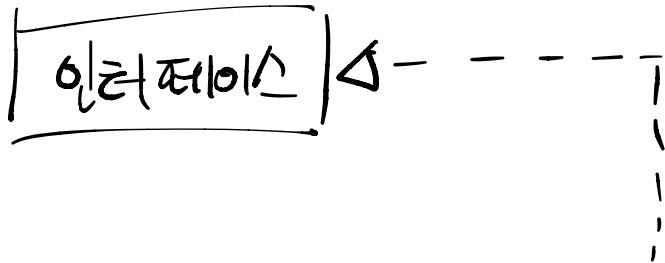


17. 인터페이스를 이용한 구조화된 접근



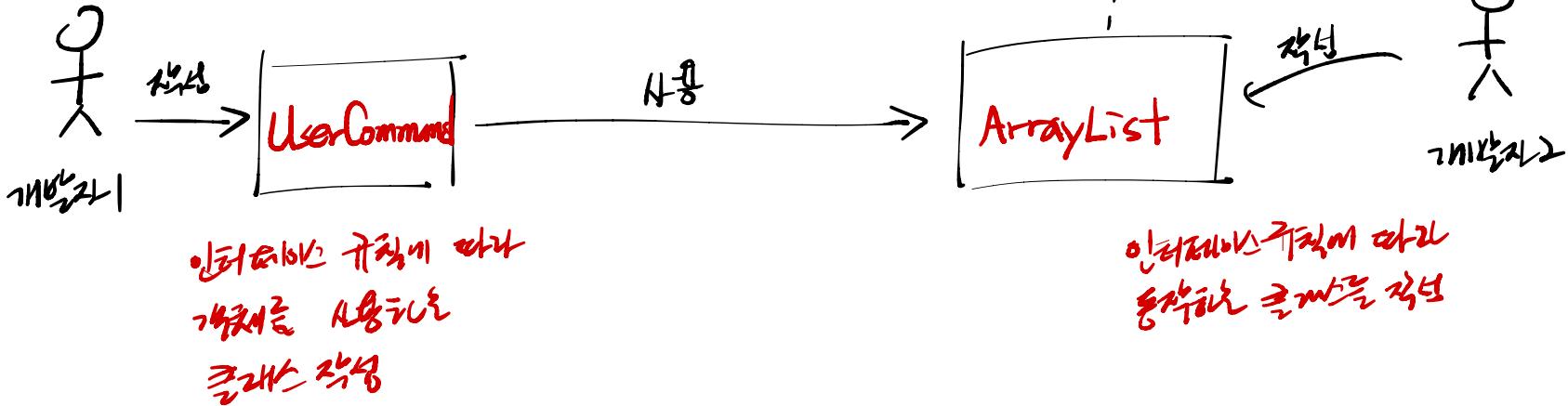
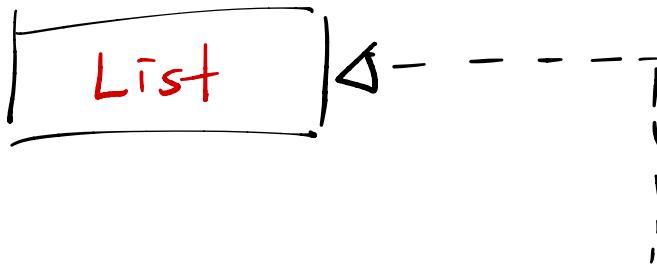
* 인터랙이스

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

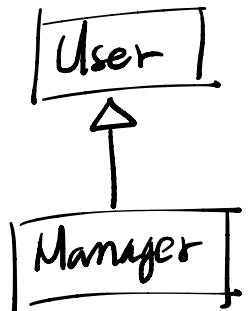


* 인터페이스

인터페이스는 구현체 없이
제공하는 기능의 만족도가 높다.



* instanceof vs getClass()



equals() {
 }
 ==
 }

`equals(Object obj) {`

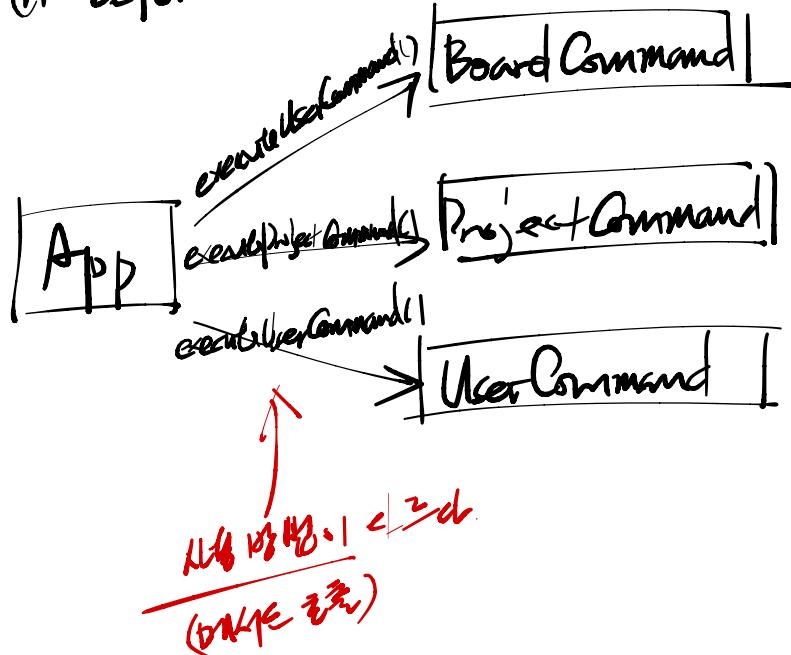
```
if (this.getClass() != obj.getClass()){\n    return false;\n}
```

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

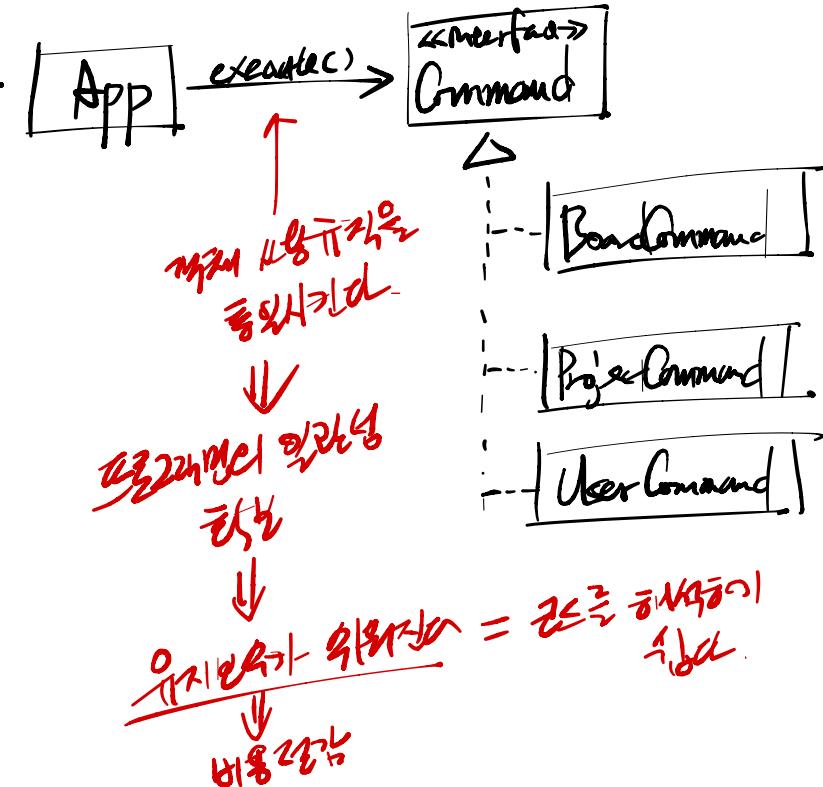
	W ₁	W ₂
ul.equals(str)	F	F
ul.equals(ul)	T	T
ul.equals(m)	F	T

* 121 능력과 122 번의 학습자료

① before

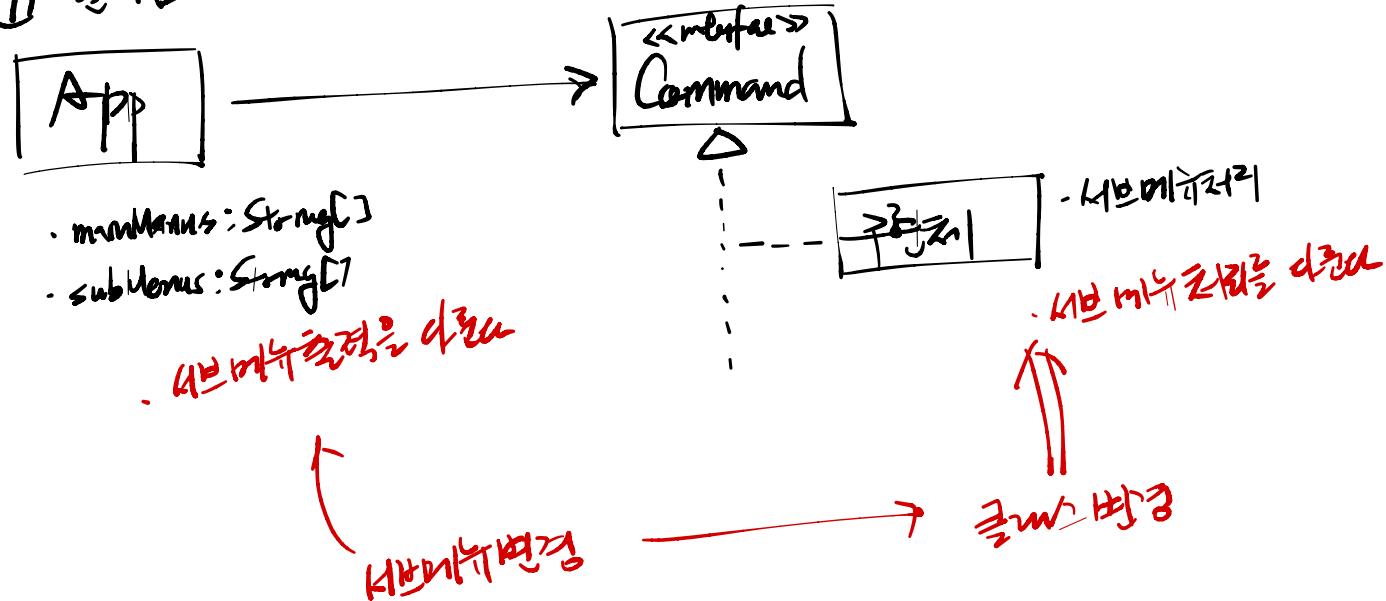


② after



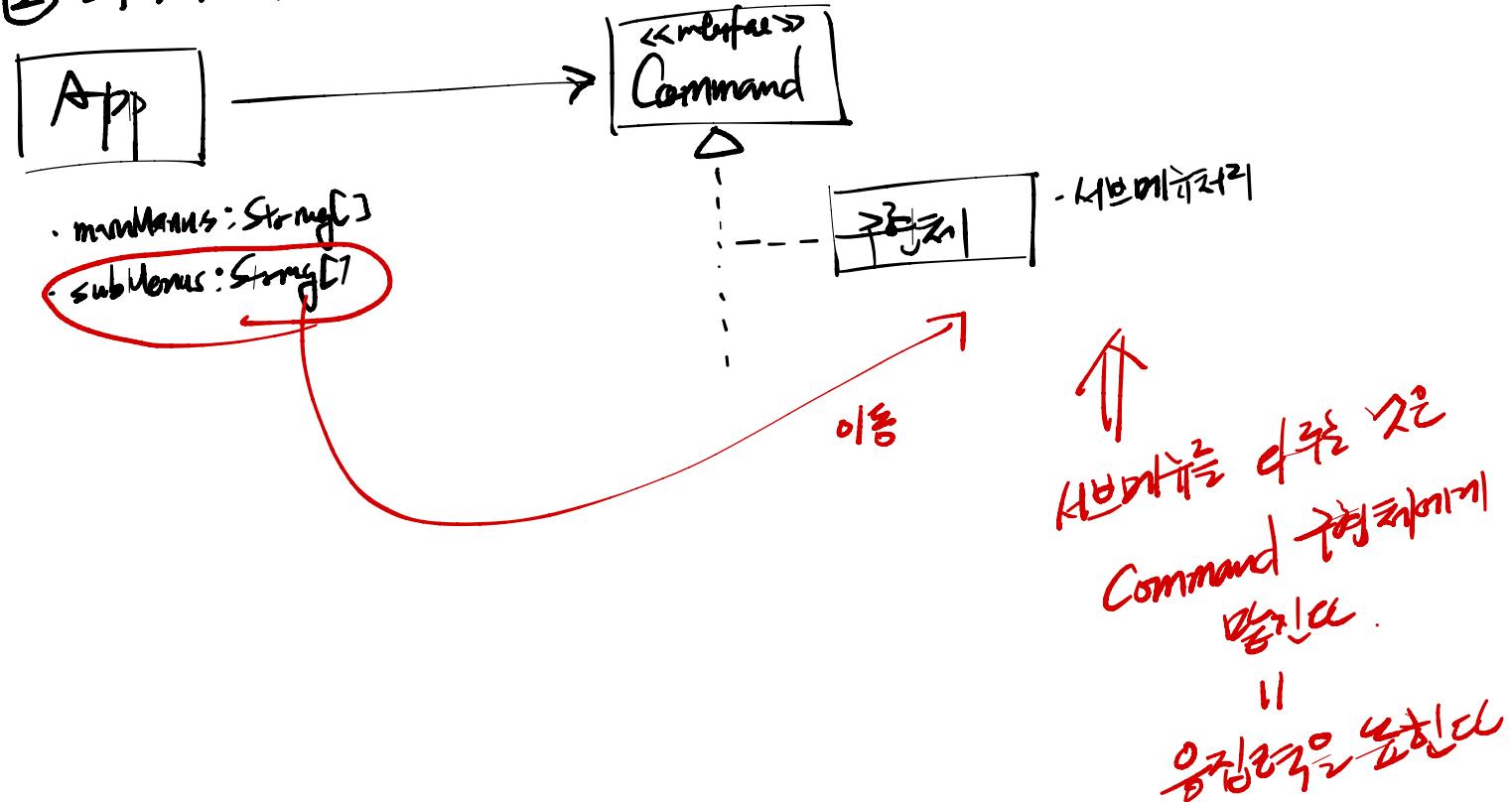
18. 커맨드 패턴

① 응용



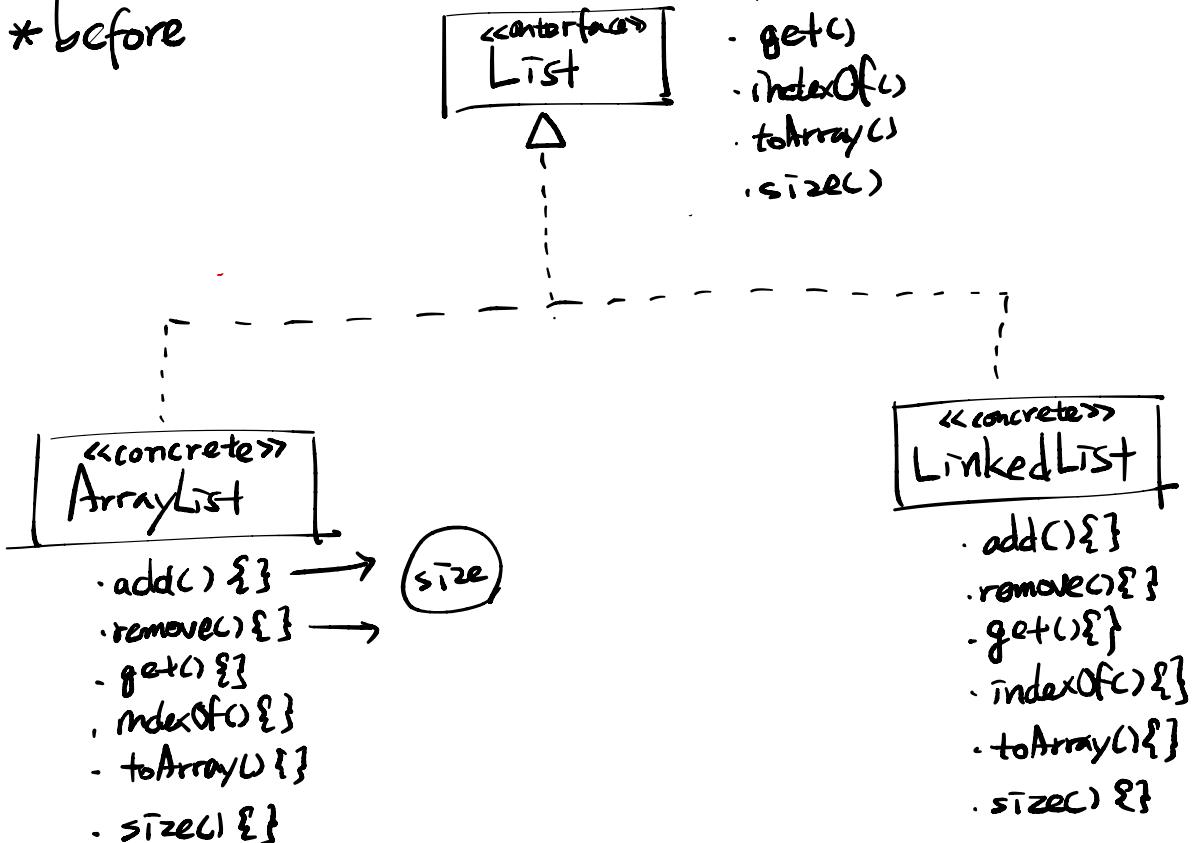
18. 디자인 패턴

② 명령 : GRASP의 High Cohesion & Low Coupling



19. 상속의 Generalization - ①

* before



19. 상속의 Generalization - ①

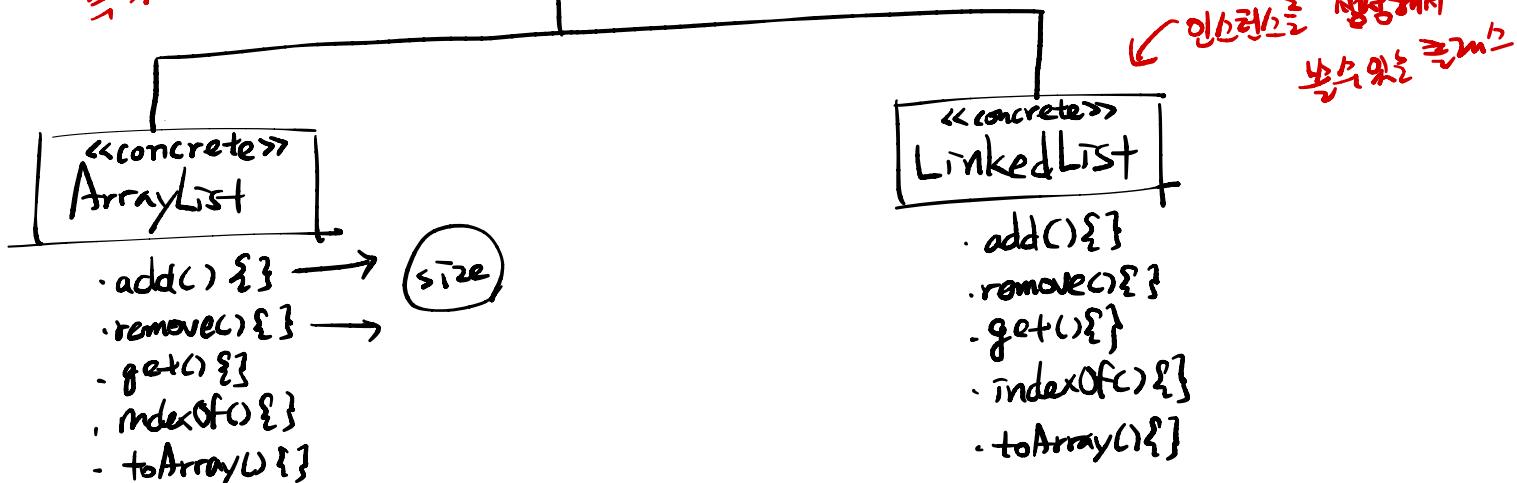
* after

①

상속을 통한
공통 기능을 대상으로
설계하는 경우는
제작할 때
제작할 때
인스턴스 사용하기
쓰기 편리해!

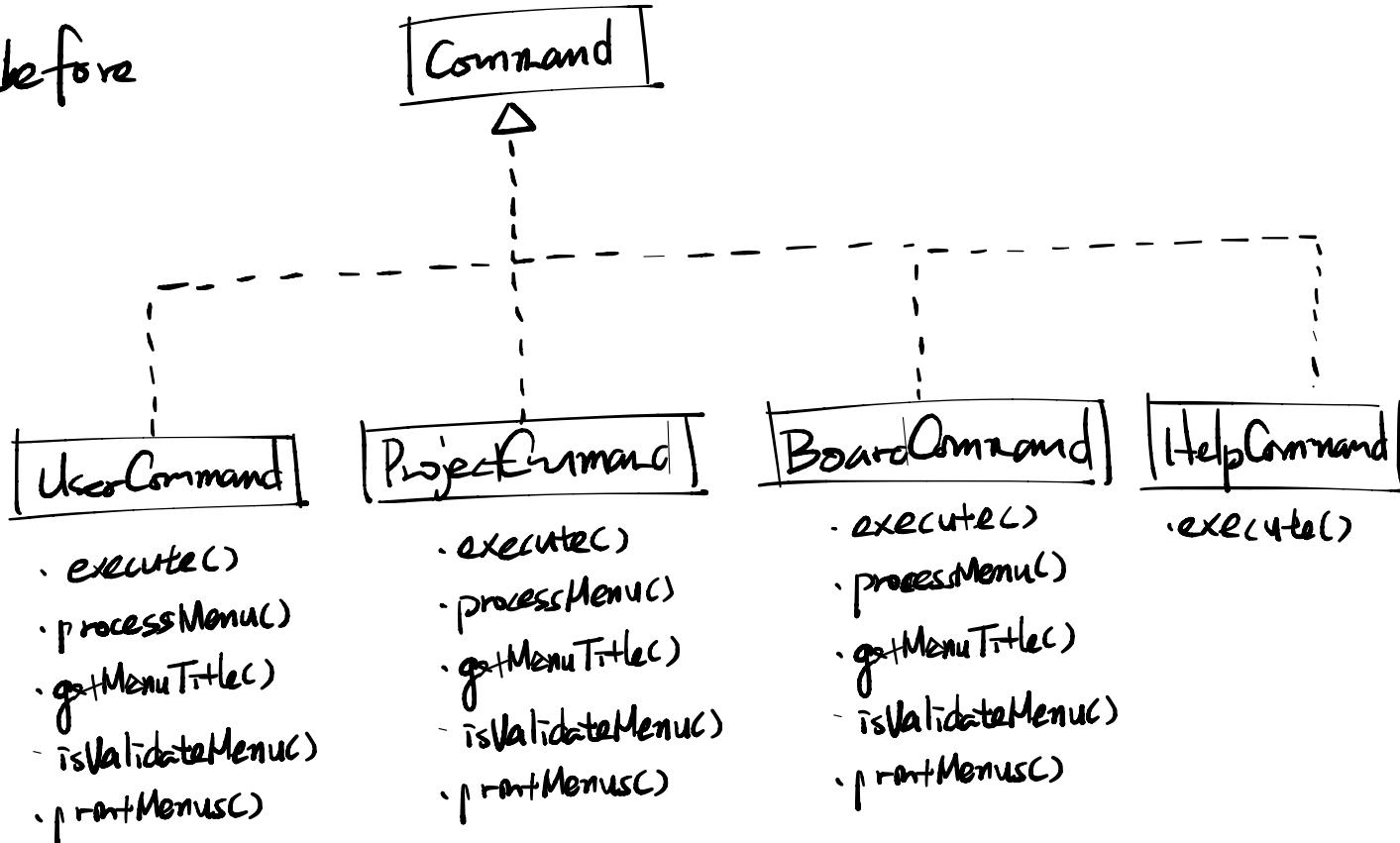
②

인스턴스 사용하기
쓰기 편리해!



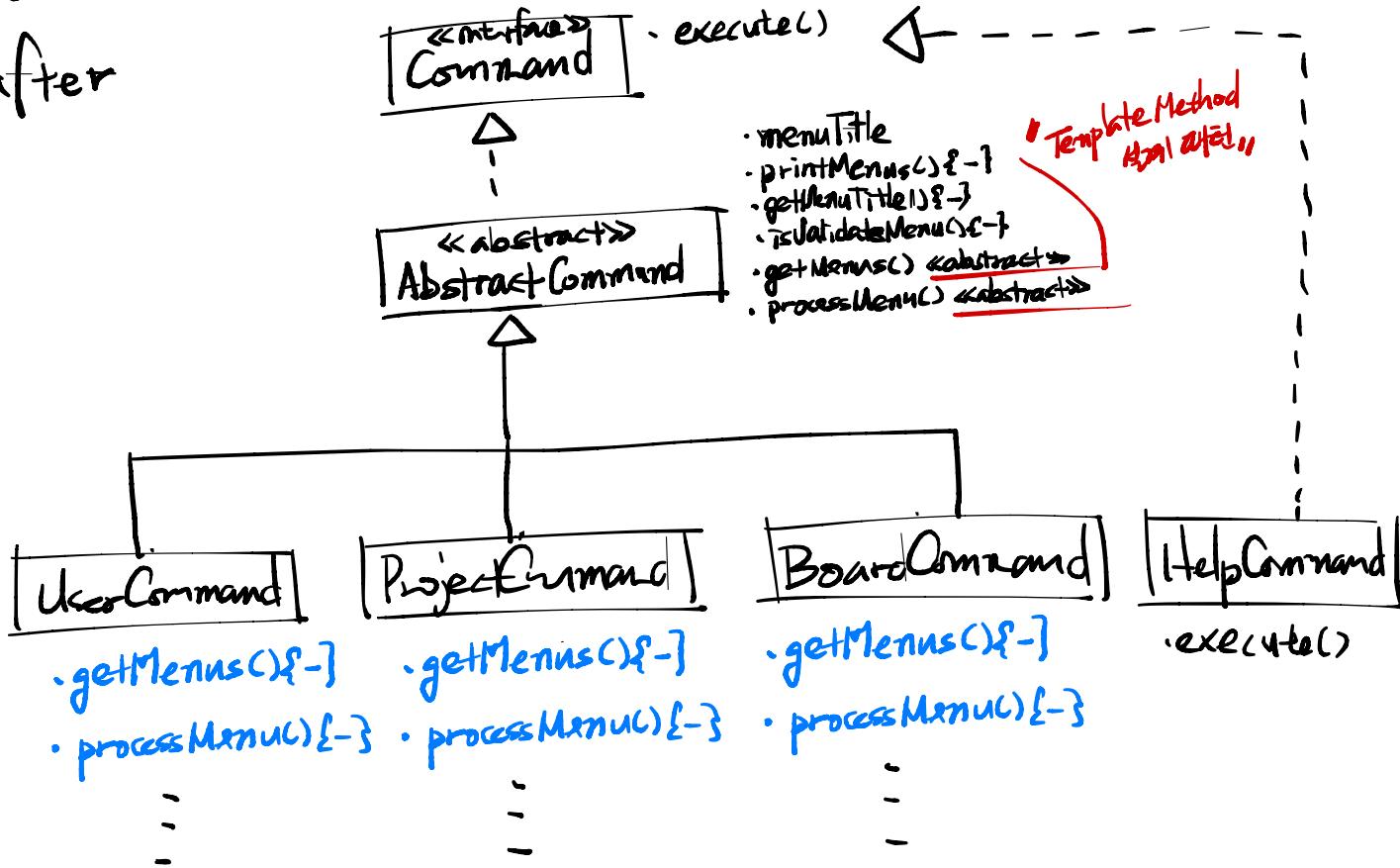
19. 상^k으로 Generalization - ②

* before

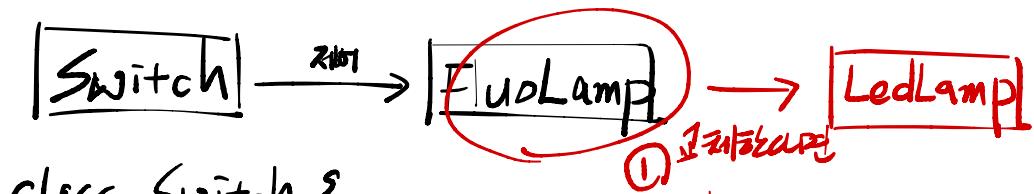


19. 一般化 Generalization - ②

* 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 단점 존재" \Rightarrow 해결?

20. SOLID의 DIP와 GIRASP의 Low Coupling

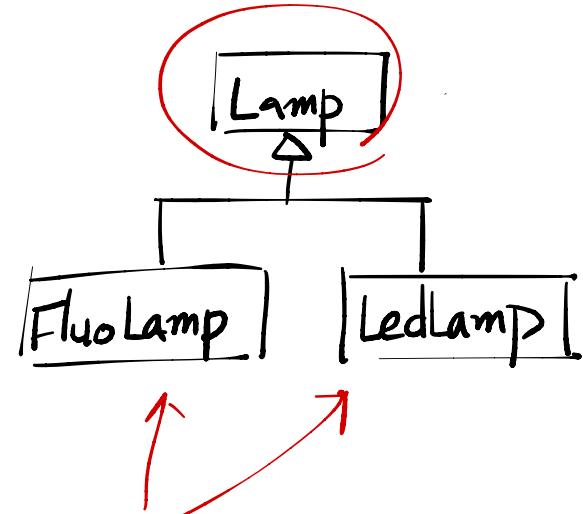


class Switch {

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 공개하여
→ 좋은 태입으로 뷰으면

20. SOLID의 DIP와 GRASP의 Low Coupling



class Switch {

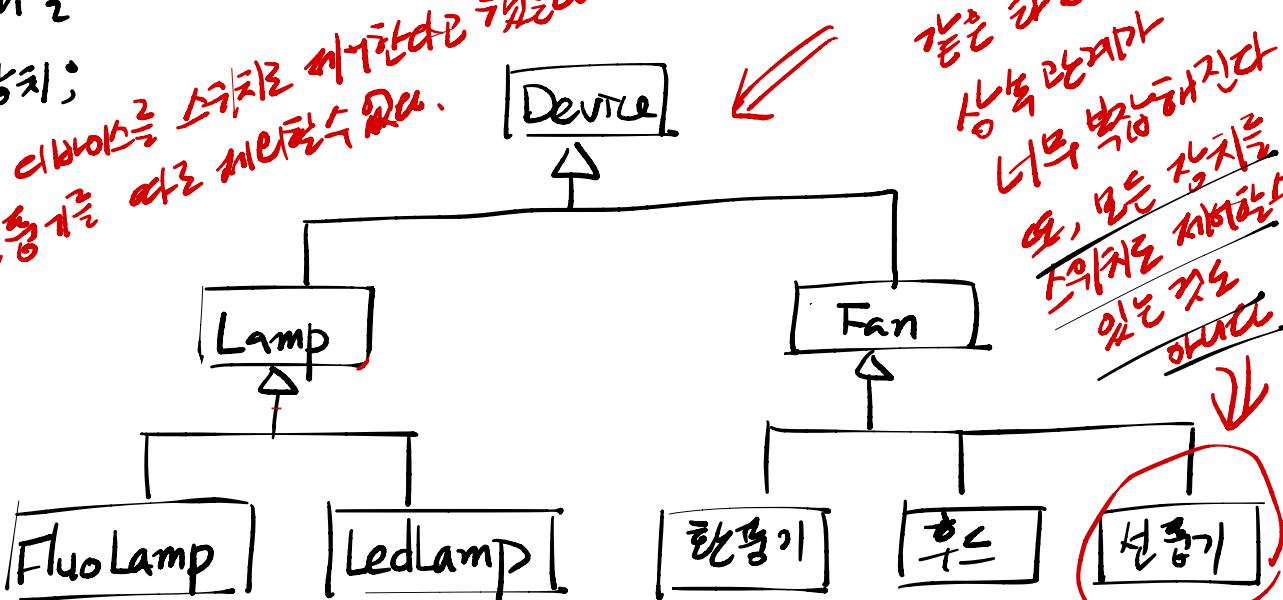
 Device 장치;

}

*② 모든 클래스를 스위치로 대체할 수 있다.
인접 클래스는 제외된다.*

① 여러 장치를 스위치로
대체하거나
같은 태스크로 묶거나
다른 장치가
너무 복잡해지면
여기, 모든 장치를
스위치로 처리하는
있는 것도
있다.

③
상속은 "간접화"의
유익성이 예술,
유연성 부족.



20. SOLID의 DIP와 GIRASP의 Low Coupling

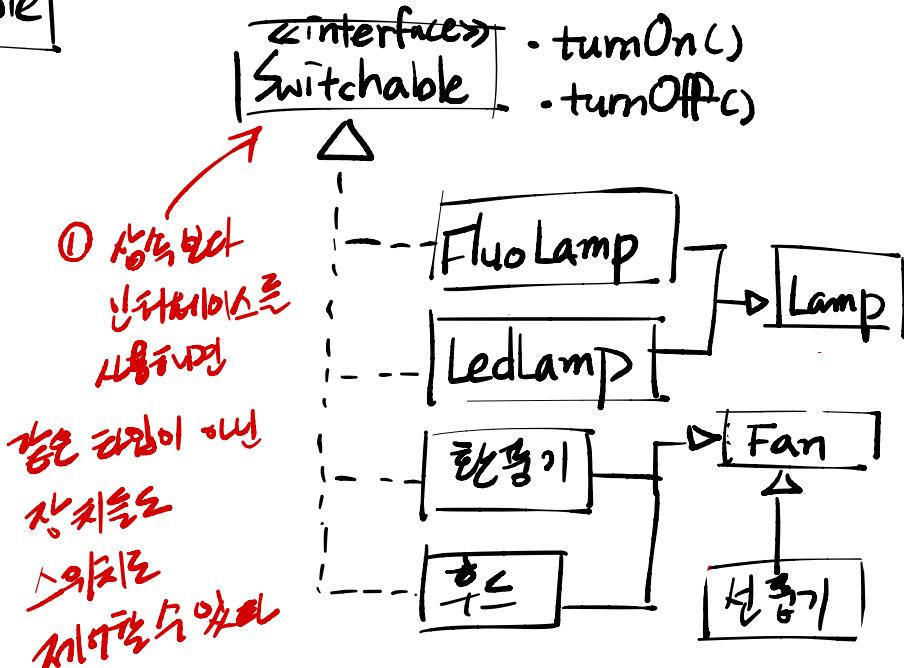


class Switch {

 Switchable 광고;

}

② 어떤 타입이든
Switchable 인터페이스
를 상속하는 것을 가능하게
만들기 가능



↳ ② 캐릭터 유전형 = 유연하다
"액션형" = 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)

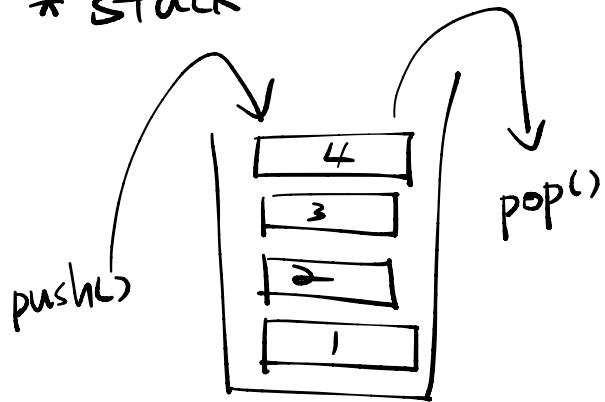
(제어권역)

Inversion of Control (IoC)

- ① Dependency Injection
② Listener = event handler

21. 자료구조 - Stack 와 Queue

* stack



First In Last Out

(FILO)

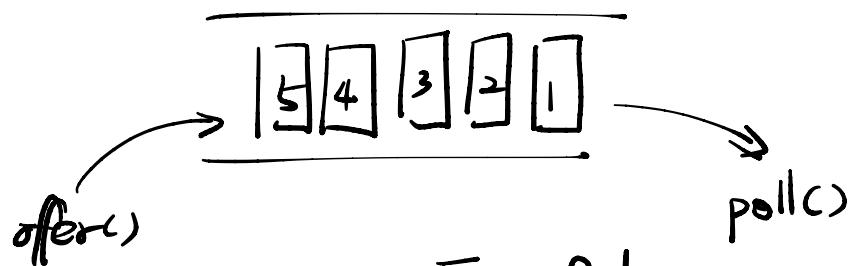
"

Last In First Out

(LIFO)

- ① 뒤로 넣어야 하기
- ② 앞으로 빼야 하기
- ③ 예상하지 못하는 예

* Queue



First In First Out

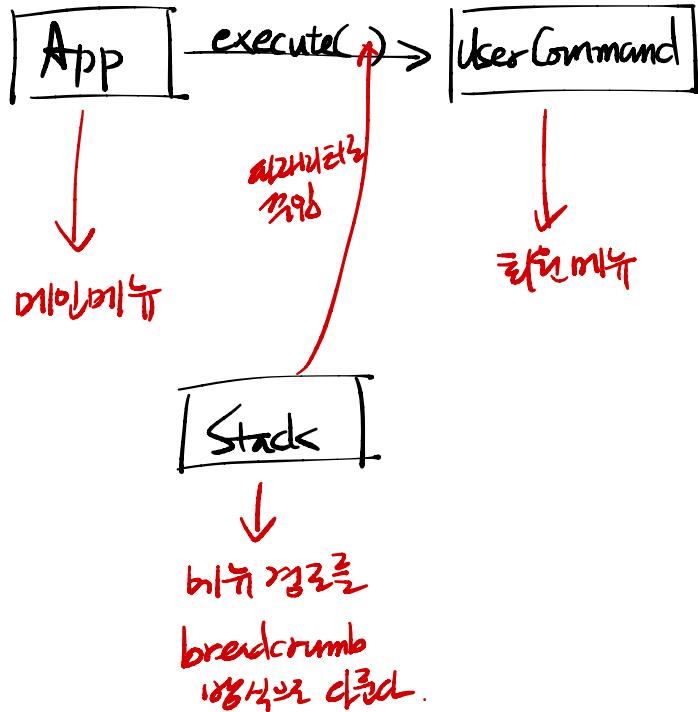
(FIFO)

① 예상

② 예상하지 못한 예상으로 예상 처리

③ 이벤트 처리 = Event Queue

* 디足迹 제목을 소리으로 하기



* String

String str = "";

str += "aaa";

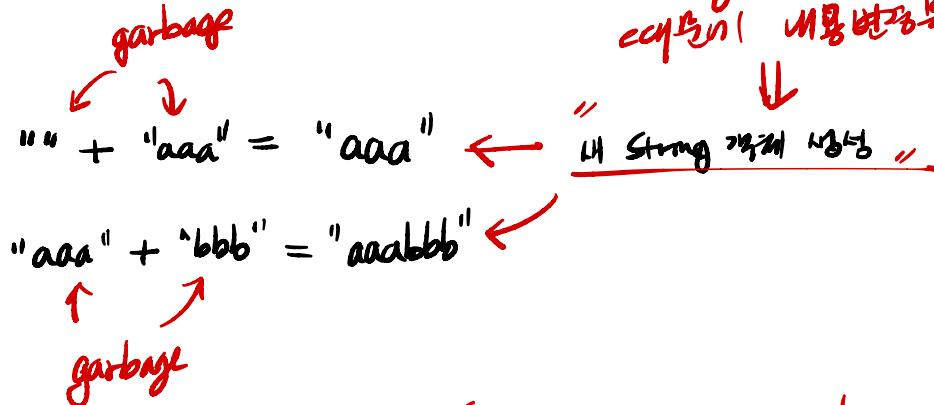
str += "bbb";

★ 왜 Java에서는
String은 오직 같은
StringBuffer는
StringBuilder는
이유는 그다지
아니다.
↳ garbage는 끊임없이
생성되고 해제된다.

thread-Safe

스레드 안전한지 알기
↳ lock/unlock
함수는 안전

StringBuffer, StringBuilder



Strong immutable \Rightarrow 안전한
copy는 만들지 않는다!

문자열은 대체로 같은 1H Strong \Rightarrow 안전하다
1H String은 \Rightarrow garbage \Rightarrow 안전.
문자열은 안전하다.

thread-Safe \Rightarrow 안전한지 알기
↳

thread-Safe \Rightarrow 안전한지 알기
↳ 스레드 안전한지 알기
↳ lock/unlock 함수는 안전
↳ obj锁 \Rightarrow 안전

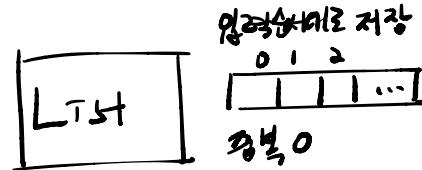
22. Iterator 깊이 파악

이전 문서화하는 강점인 하위 목록

· 재사용성↑ · 유지보수성↑ · SOLID / GRASP 부합

문서
수집하는
수집하는
방법이
다르다!

인덱스(정수)
get()



toArray()

Set

한시점으로 저장하는 형식

0	1	2	...

정복 X

key 가짐
get()

Map

기본적으로 값 저장

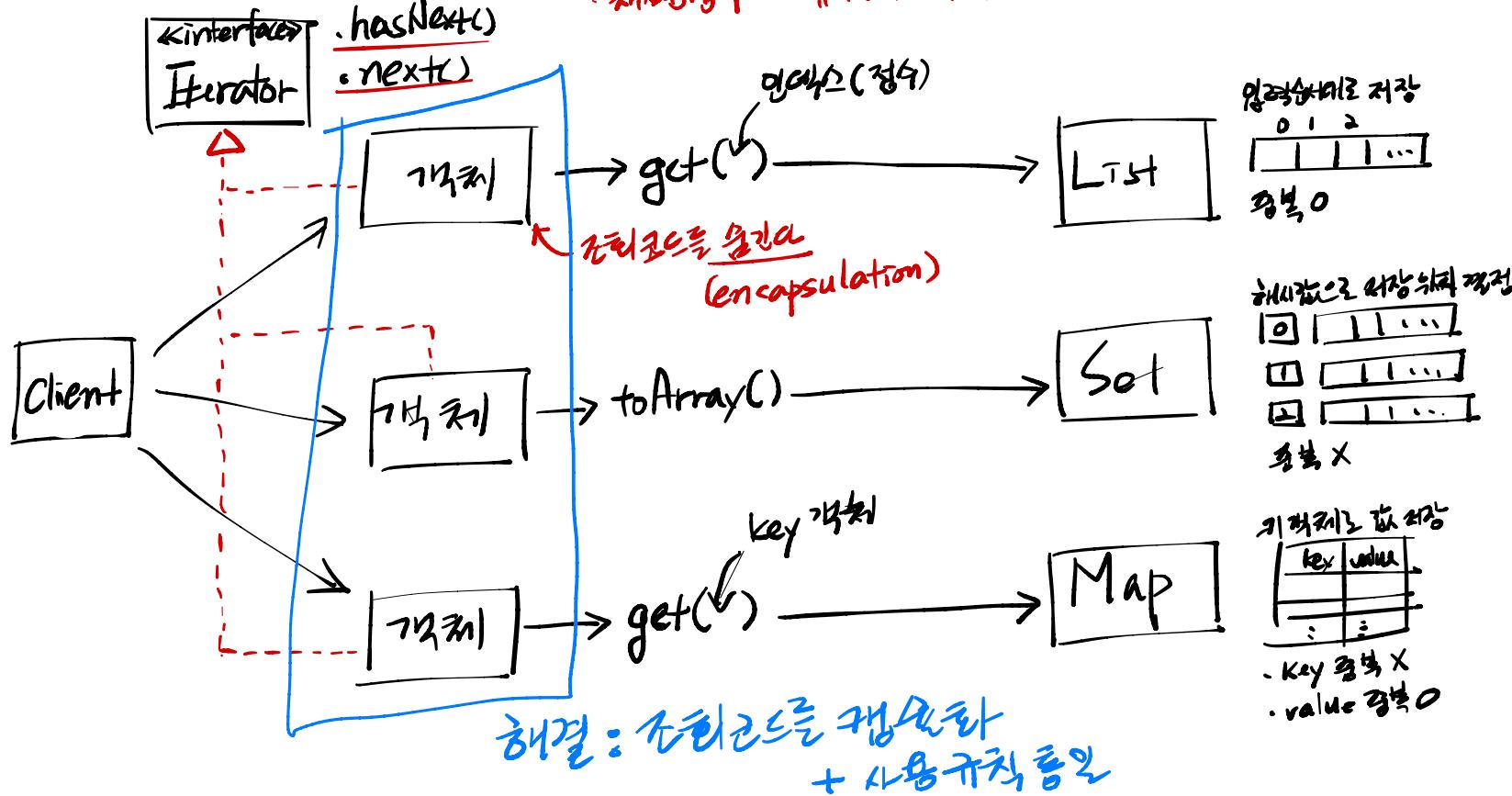
key	value

- Key 정복 X
- value 정복 O

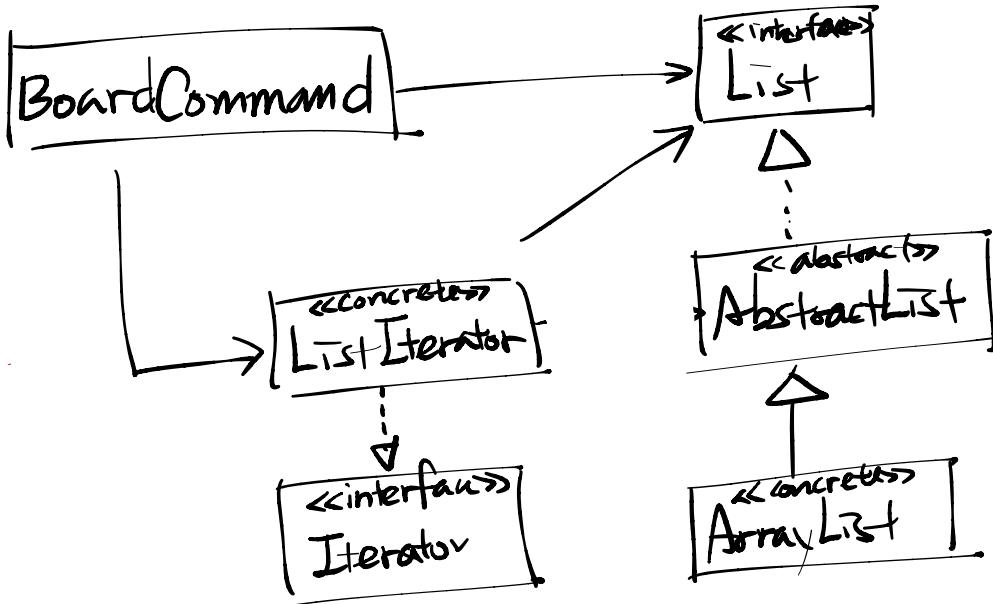
22. Iterator 기능적 패턴

이전 문서화하는 강제된 흐름 방식

· 재사용성↑ · 유지보수성↑ · SOLID / GRASP 부합

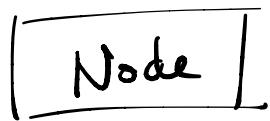


22. Iterator 틀 구조



23. 중첩 쿨러스

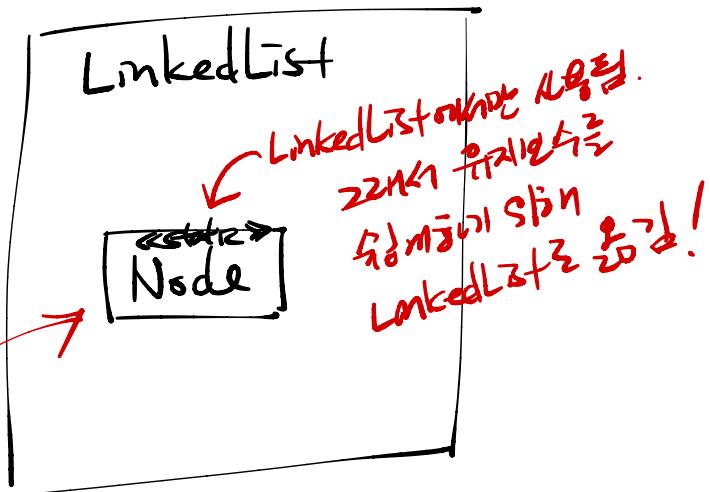
before



package member class

Nested class

after



LinkedList

Linked List \leftarrow $22 \rightarrow 11 \rightarrow 21 \rightarrow 12 \rightarrow 13 \rightarrow 25 \rightarrow N$

22nd Jan 2019
Homework Session
Linked List

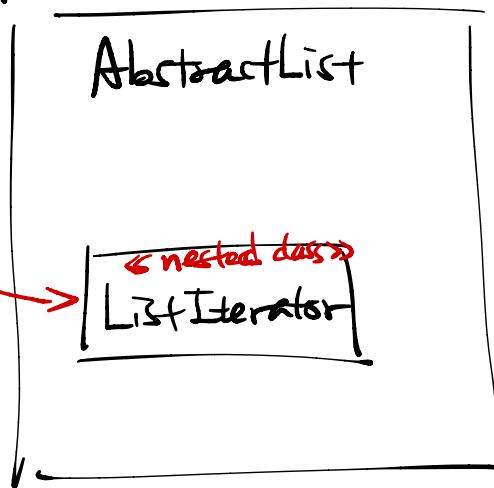
22.11.21
Homework Session
Linked List Review!

23. 중첩 클래스

before



after



* enhanced for 문법

for(변수선언 : 배열 또는 Iterable 구현체)

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

for(String name : names) { }

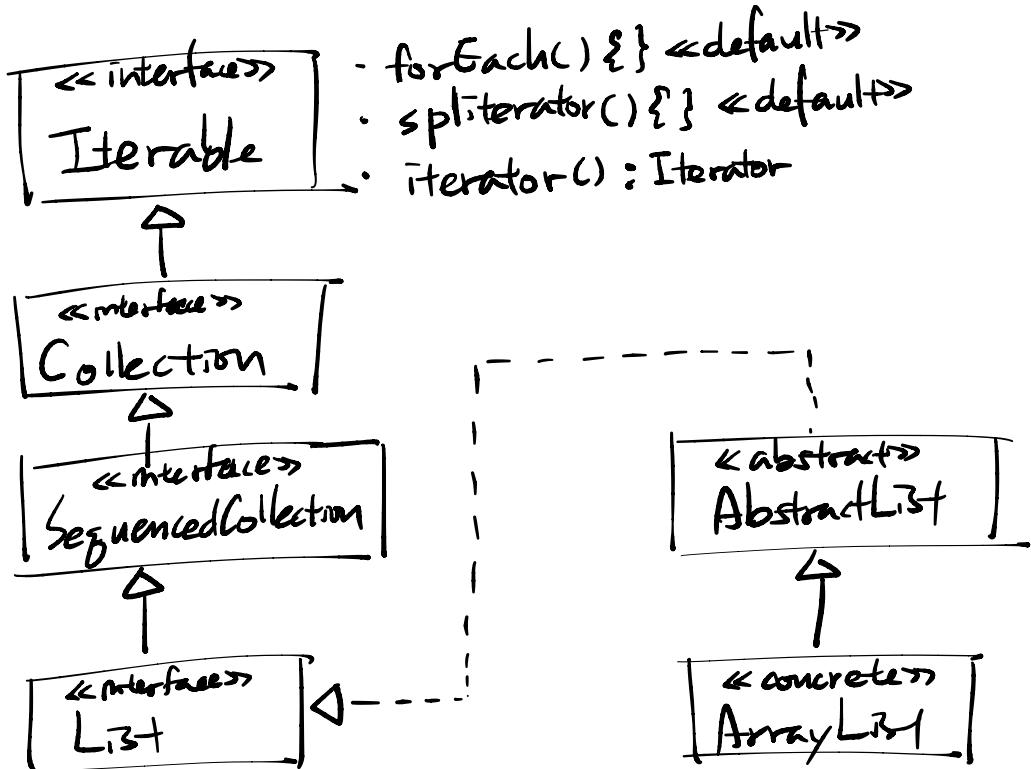
변수

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

for(Object item : list) { }

Iterable 구현체

* Iterable ↗^{3연자}



24. Generic 타입 사용하기

```
class Node {
```

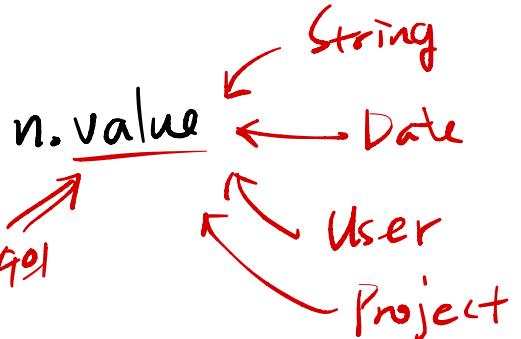
```
    Object value;
```

특정 타입의
인스턴스
만들기 위해
제한할 수

있어야 하는
제한할 수
있어야 한다.

있을까?

```
Node n = new Node();
```



다형화 변수의
특성
하여 유동화의
사용하는 경향이
있음.

⇒ Generic 타입

24. Generic 블록 사용하기

class Node<what> {

what value;

what이
어떤 타입인지

선택할 때
제한한다

타입 제한자 = 타입 정보를 넣은 변수

제한자에서
타입 제한자에 들어가면
타입 제한자에
해당하는 타입
만 가능하다

Node<String> n = new Node<String>();

n.value ~~Date~~ ~~User~~ ~~Project~~ ...

String 타입의
가능성이 있다!

String OK

* Type Parameter

↑ 타입 명보를 뱉는 변수

이거 { T (Type)
E (Element)
K (key)
V (Value)
S, U, V

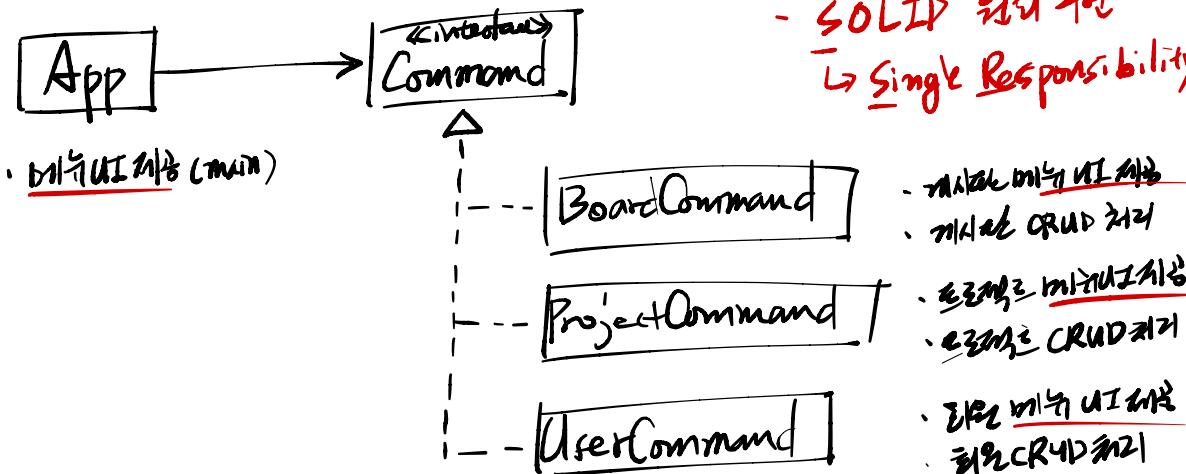
26. GoF의 Composite 패턴 대안

* before

↳ 개체가 트리 구조로 포함 관계임.

문제

- 메뉴나 UI 툴은 같은 정체
 - Command 패턴과 여러 모의 일을 처리
- ↳
- 메뉴나 UI 툴은 같은 정체 → 개체를 분리
 - SOLID 원칙 구현
 - ↳ Single Responsibility Principle

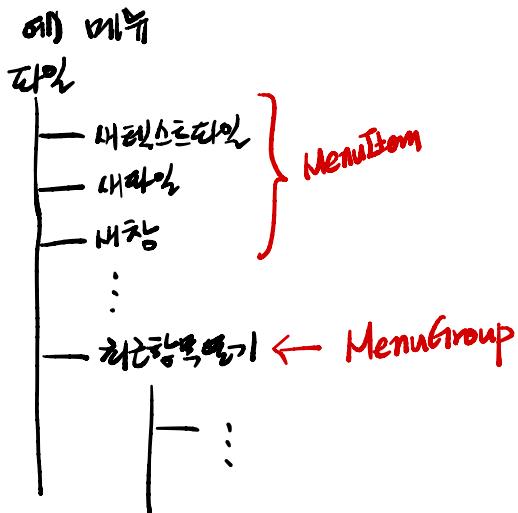
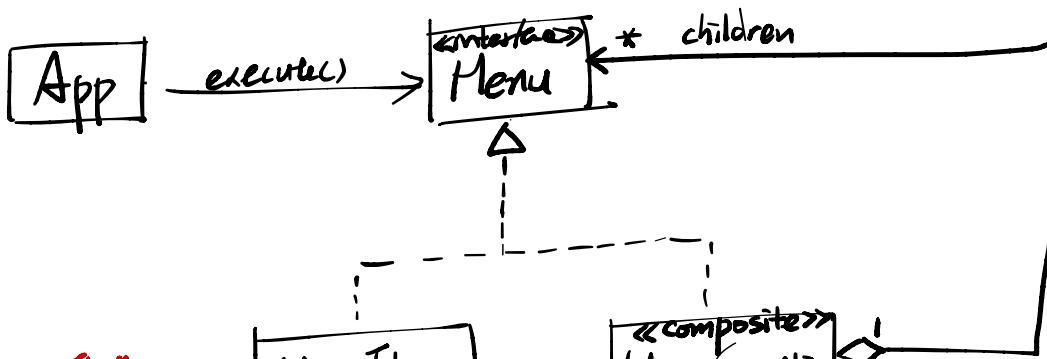


26. GOF의 Composite 패턴 대안

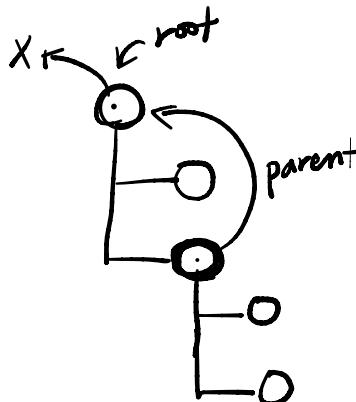
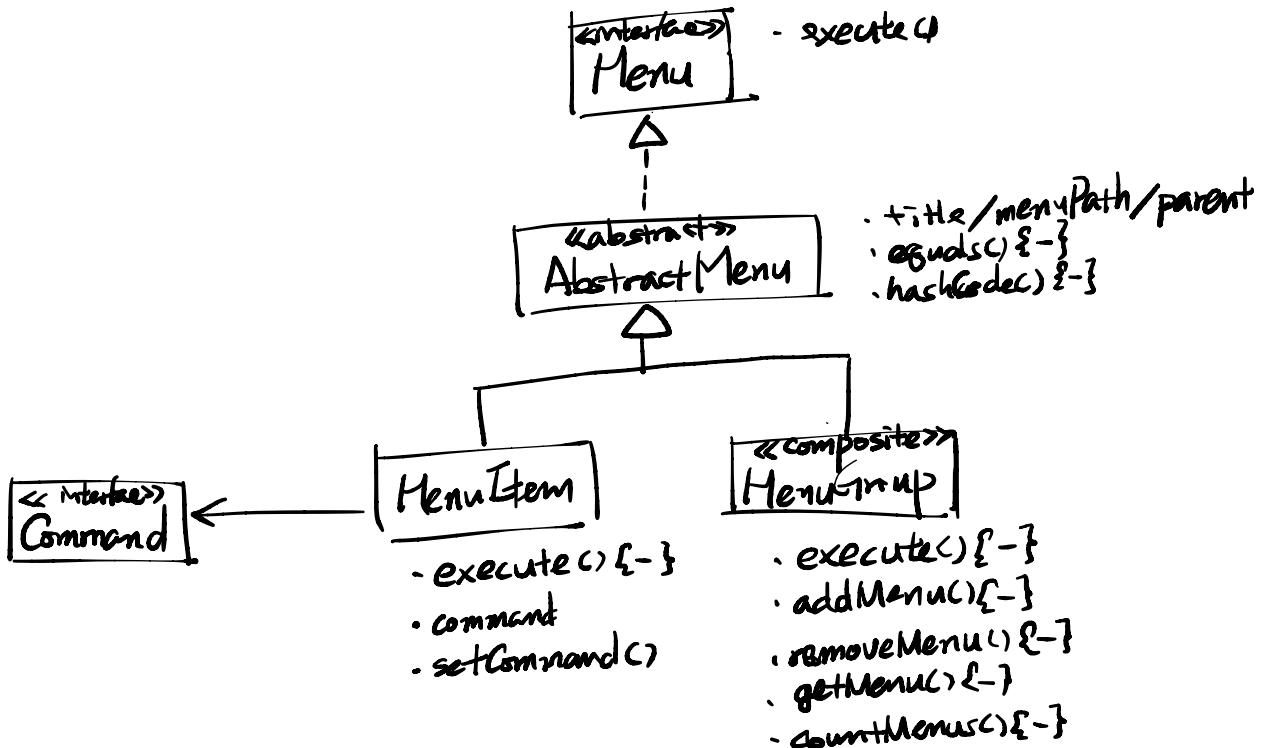
* after

↳ 개체가 트리 구조로 포함 관계임.

- { ① 메뉴
- ② 그룹
- ③ 파일 시스템

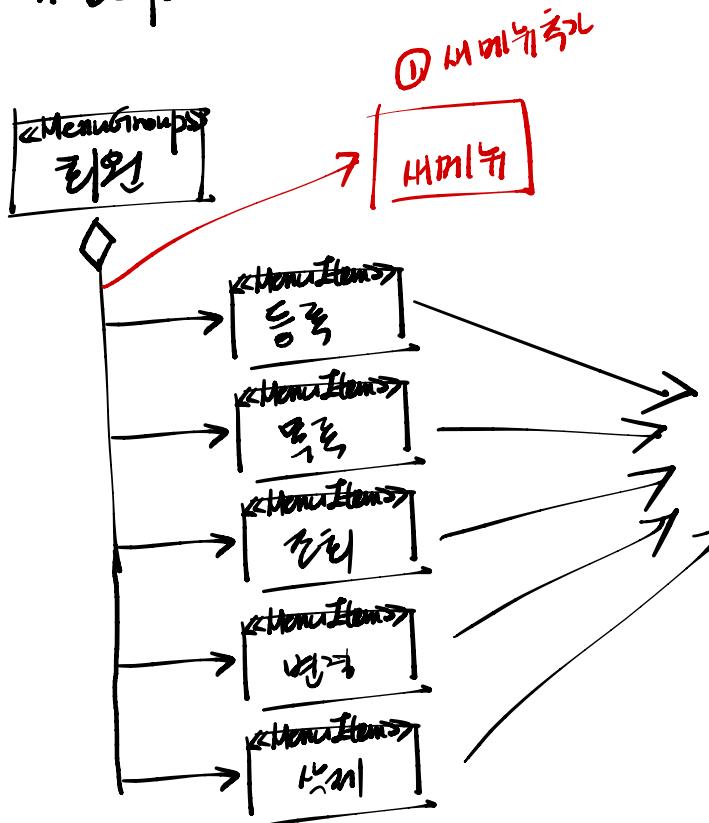


26. GOF의 Composite 패턴



27. 메뉴의 메뉴추가 기능을 개선하자 : GoTo Command 패턴처럼

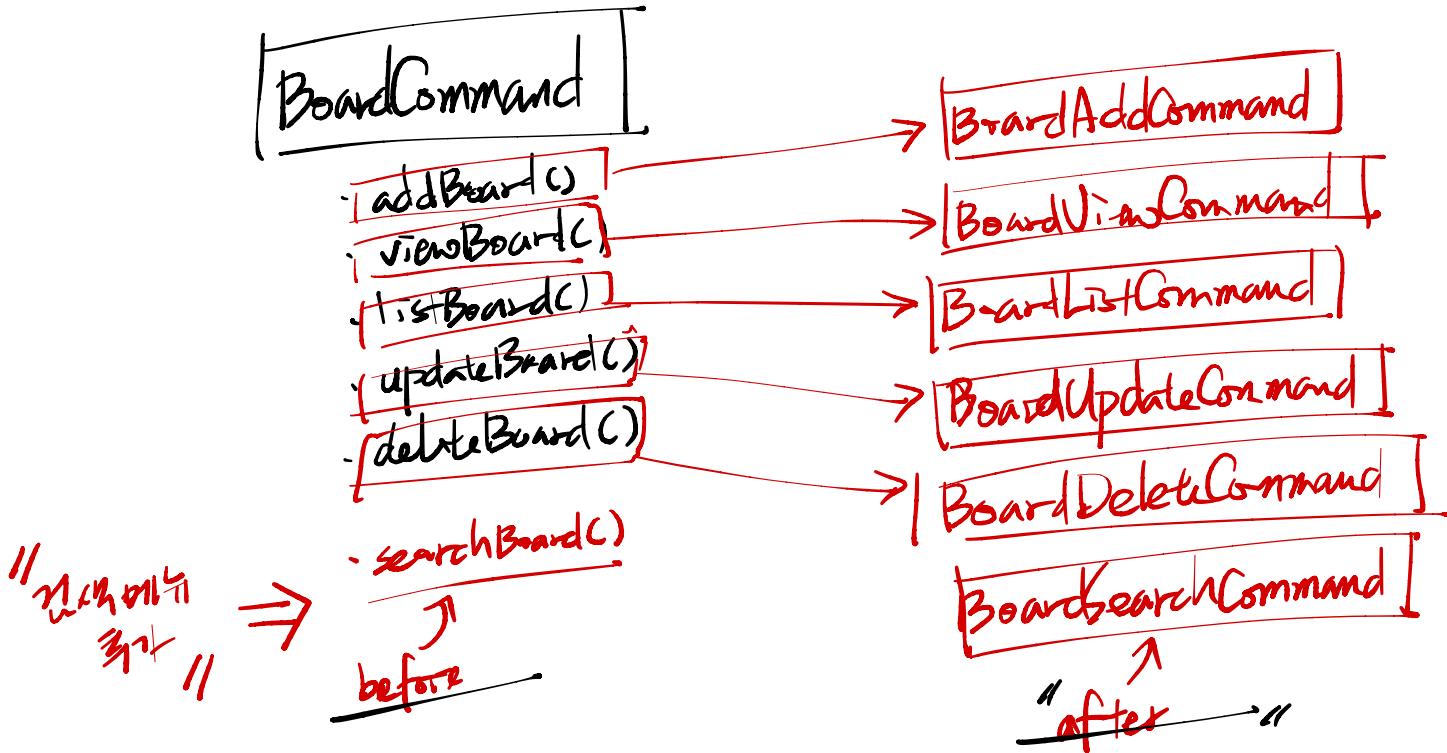
* before



② 메뉴를 처리할
코드를 추가
↓
SOLID의 OCP 원칙을
기반한
기능추가/기능제거
가능성이 있는 구조.

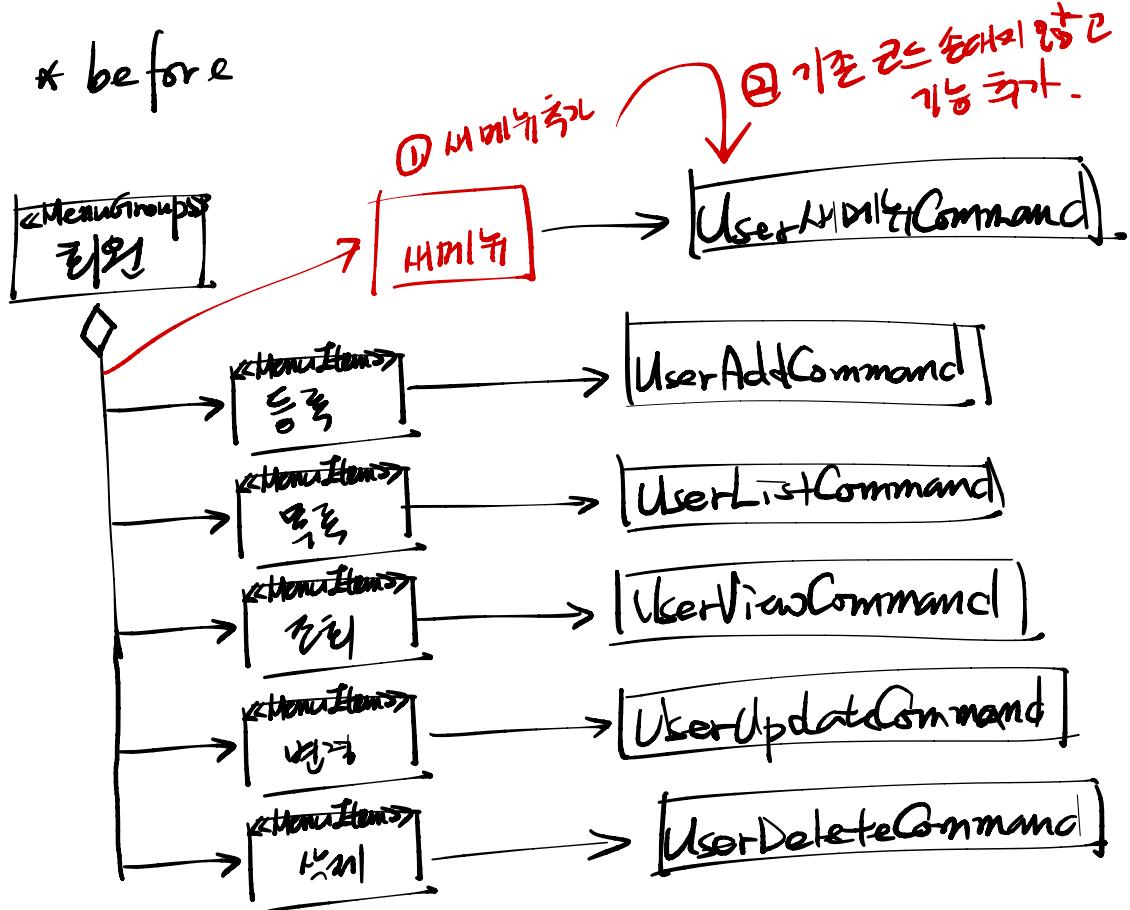
↑
기능추가/기능제거
가능성이 있는 구조.
↓
기능제거
기능추가

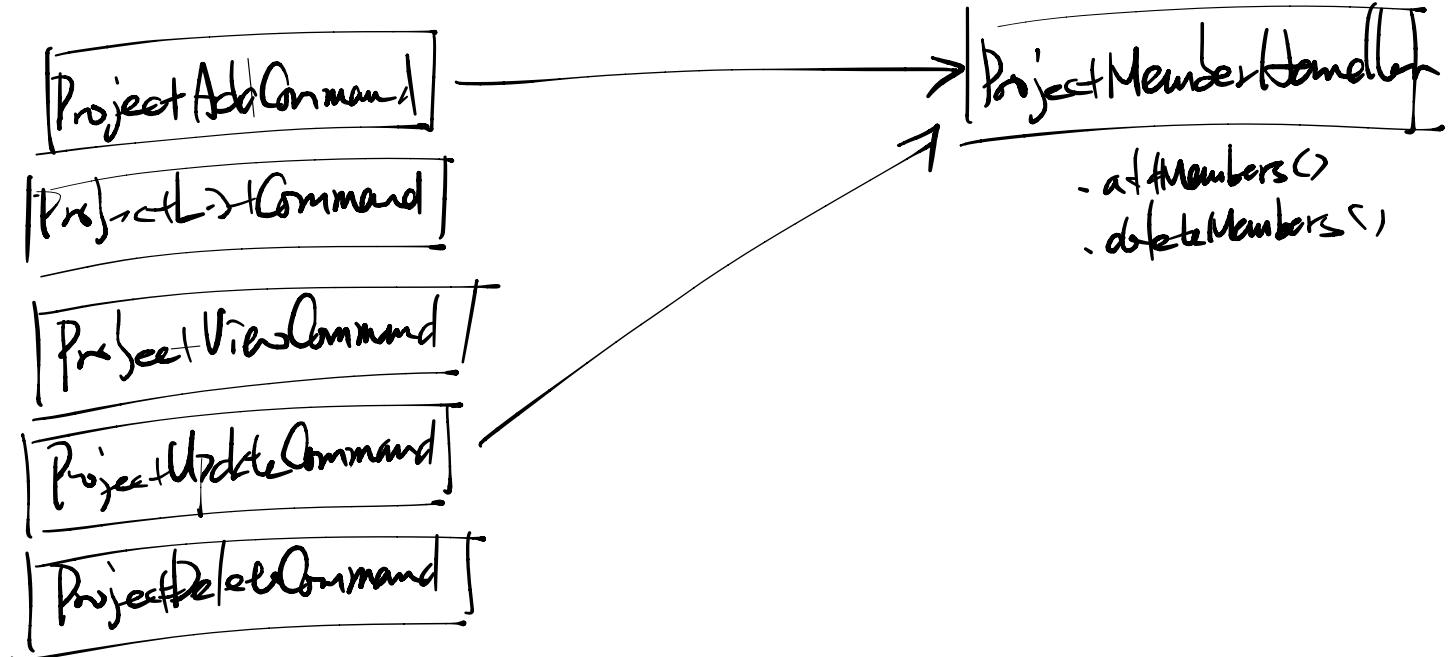
27. 게시판의 목록관리 기능을 구현하자 : GoTo Command from list
↳ list → list



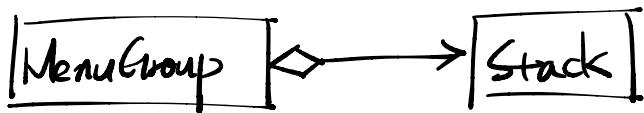
27. 메뉴의 명령처리 기능을 구현하자 : GoTo Command 패턴 틀

* before



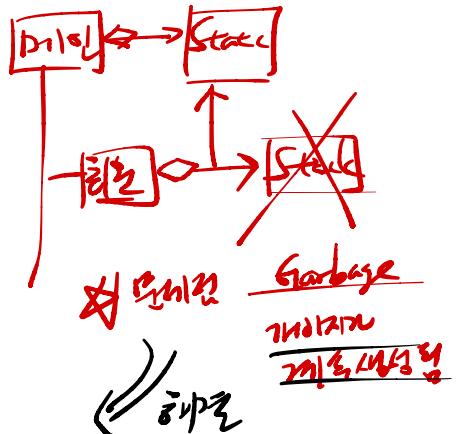
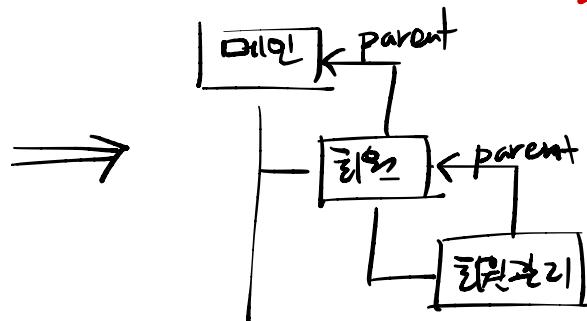
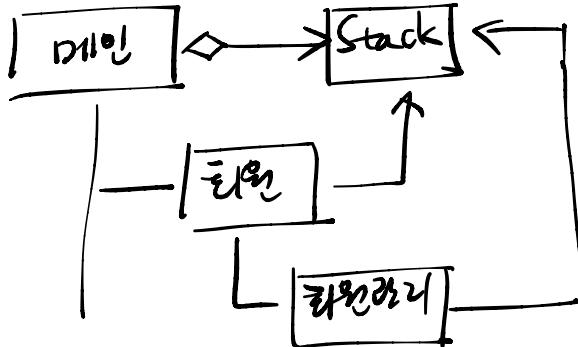


* MenuGroup 의 경로 메시지를 MenuItem로 분리



• 메뉴이동은 단순히 맵

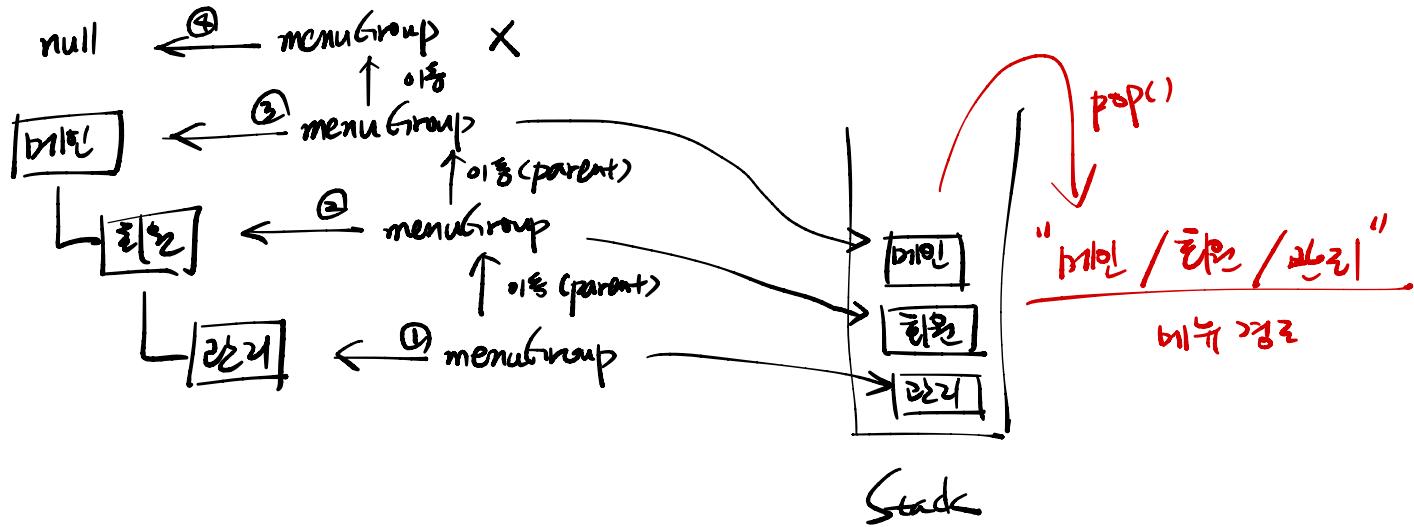
①)



✓ getMenuPath() D1101 $\xrightarrow{?}$ -

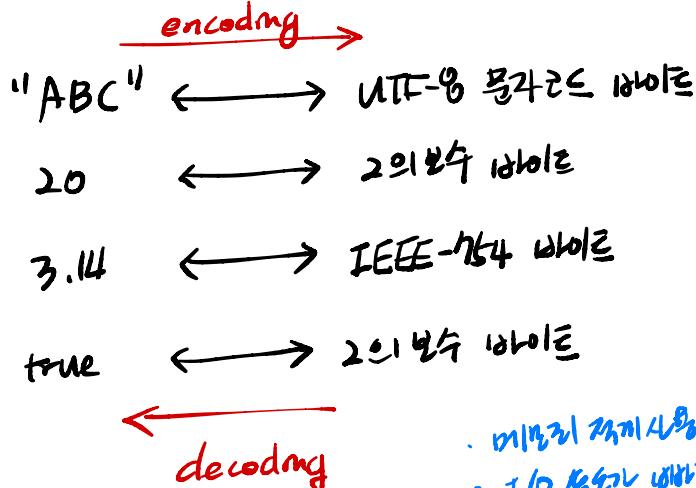
메뉴경로를 메시션

→ getMenuPath() 구조 예시



28. File I/O API 활용 : ① 데이터형의 데이터 암호화

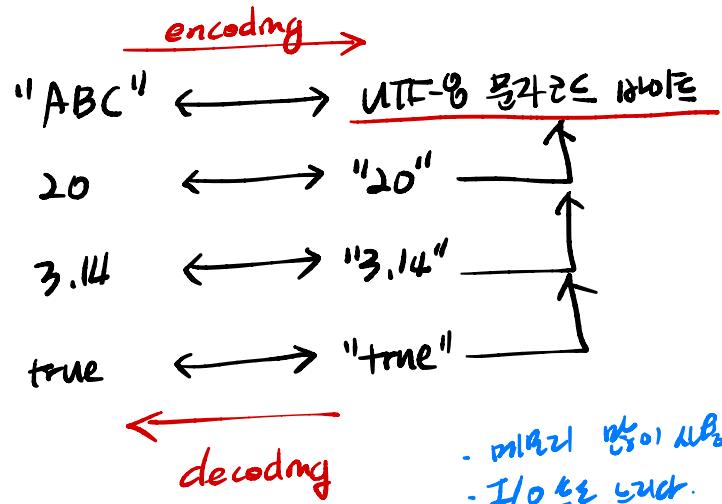
* binary data I/O



- ①) PDF, PPT, DOC, GIF, JPEG,
MP3, MP4, AVI, WAV, HWP,
EXE 등

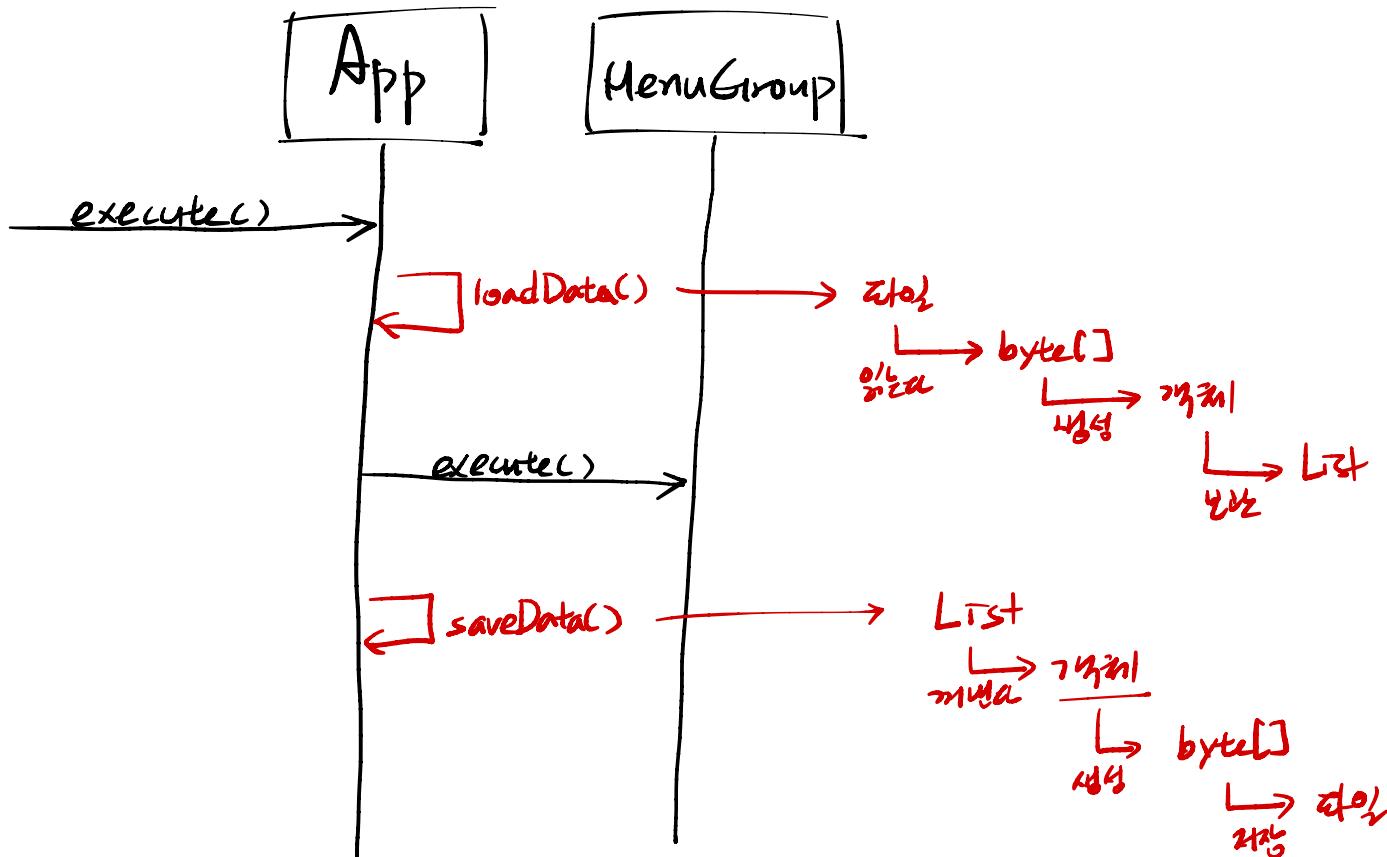
↳ 진정한 데이터를 이용한 I/O

* text data I/O



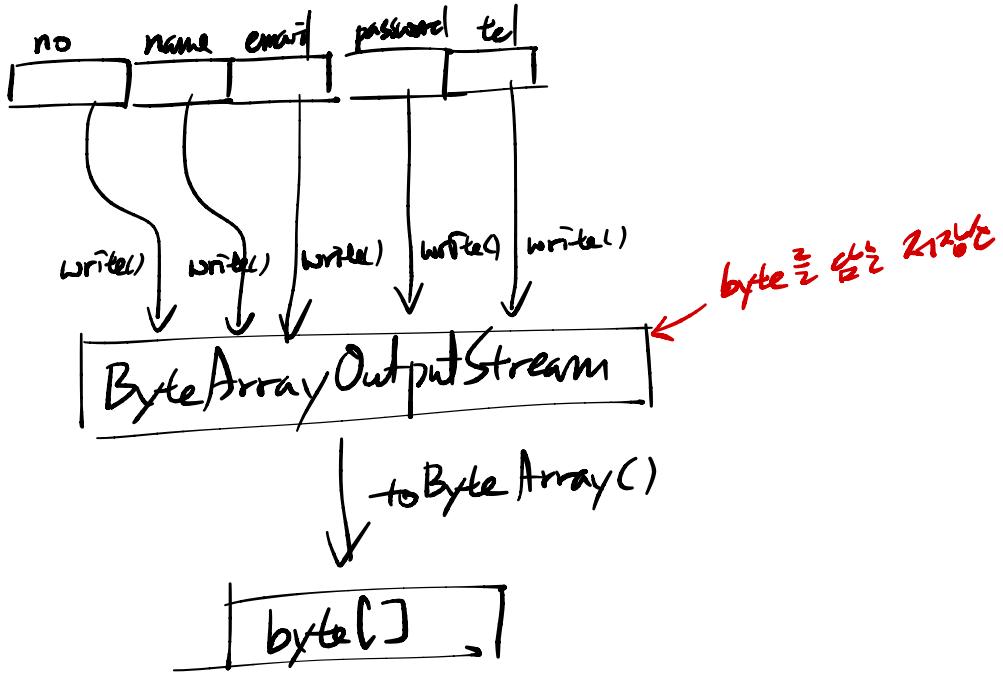
- ②) TXT, HTML, CSS, JavaScript, XML,
Properties, JAVA 등

↳ 텍스트 형식으로 이용한 I/O



* گذاشتیم که byte[] یک مجموعه

User گذاشتیم



* write(int): int $\frac{32}{2}=3$

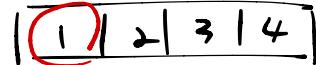
↳ 32비트 | 10101010101010101010101010101010



1byte 단위
32비트

write()

↓ 16비트(1010101010101010)만 쓸 때



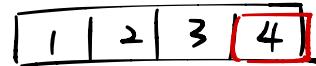
→ 1bit 이용
→ write()



16bit 단위
→ write()

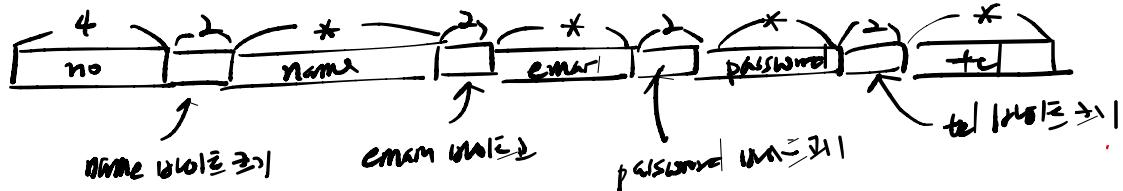


8bit 단위
→ write()

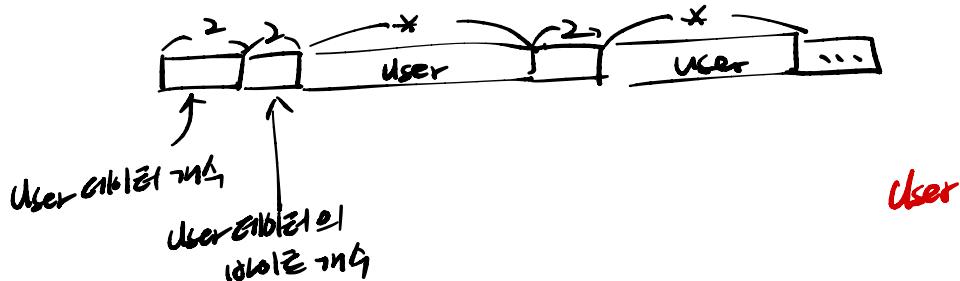


→ write()

* User 데이터 형식



* user.data 파일 형식 (File Format)



2 byte - User 데이터 형식

2 byte - User 데이터 블록은 3byte

4 byte - no

2 byte - name 블록은 3byte

* byte - name 블록은

2 byte - email

* byte

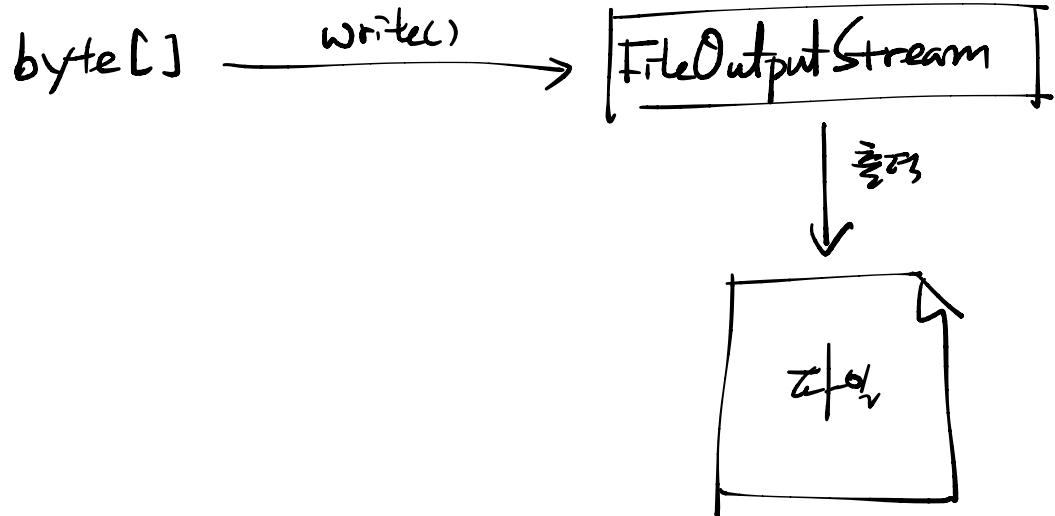
2 byte - password

* byte

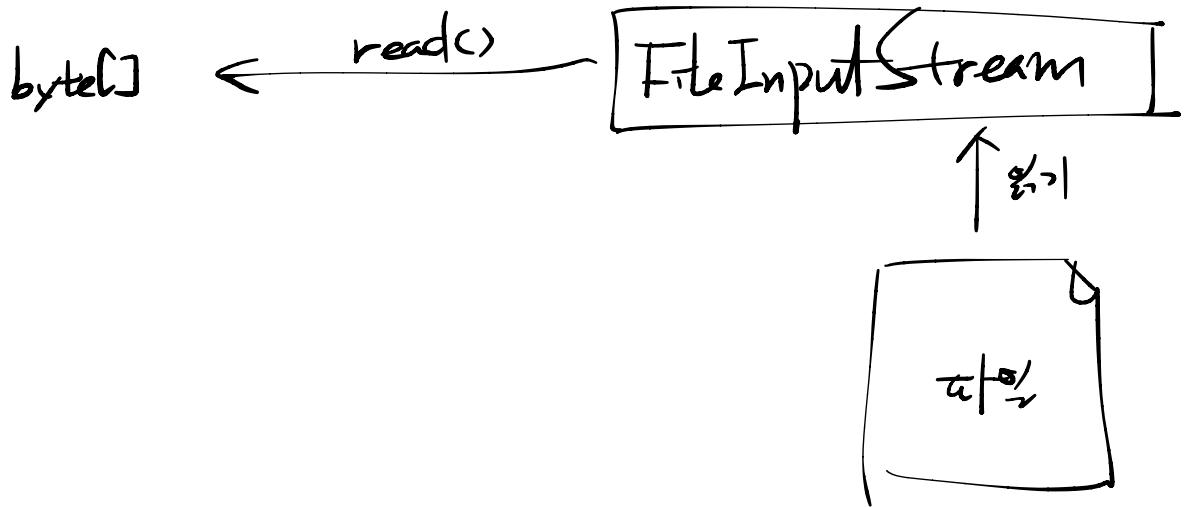
2 byte - tel

* byte

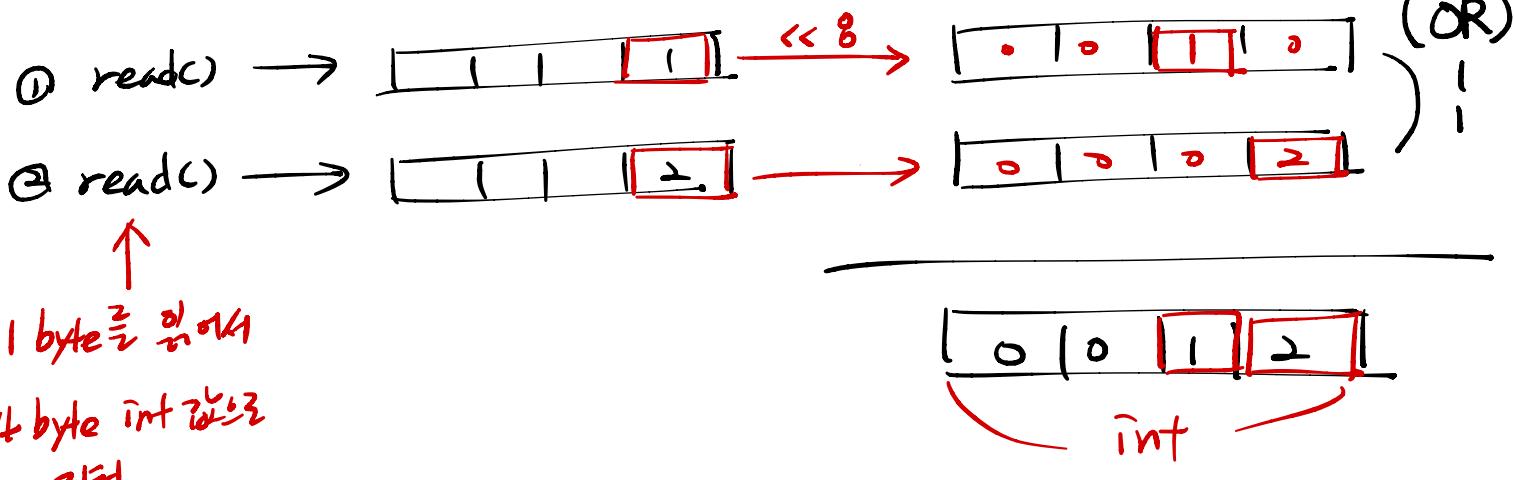
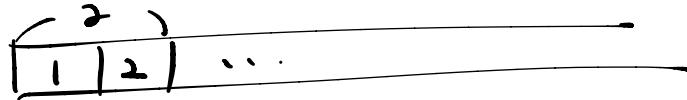
* `byte[]` $\xrightarrow{\text{写出}}$ `txt`



* $\text{ FileInputStream } \xrightarrow{\text{getchar}} \text{byte[]} \rightarrow \text{byte}[]$



* `byte[]` → `int`



* byte[] → User

2 byte - User id \rightarrow int \rightarrow (read() << 8) | read() \rightarrow int

2 byte - user id \rightarrow user id \rightarrow int \rightarrow (read() << 8) | read() \rightarrow int

4 byte - no

2 byte - name \rightarrow name \rightarrow string

* byte - name \rightarrow string

2 byte - email

* byte

2 byte - password

* byte

2 byte - tel

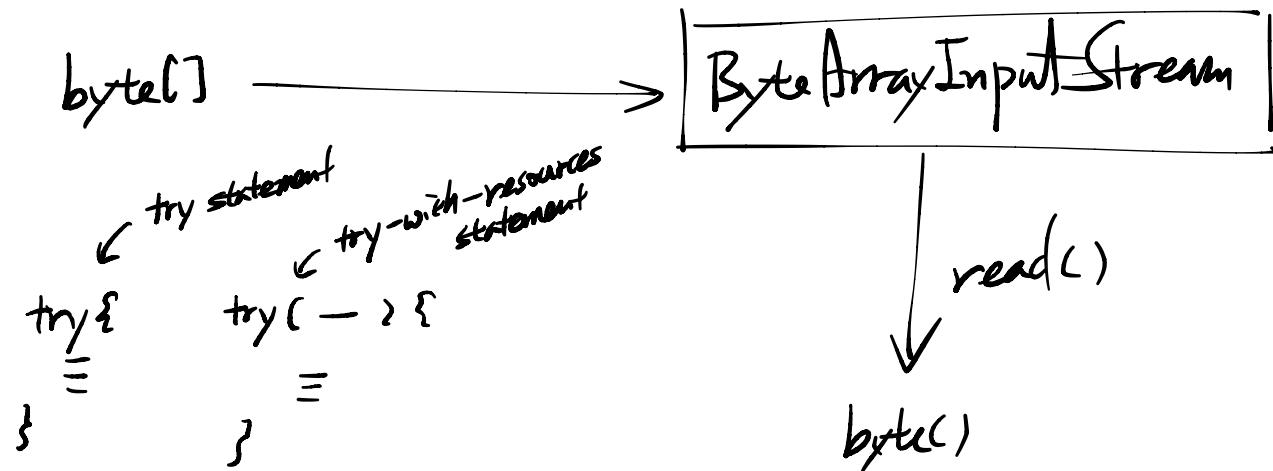
* byte

•
•
•

byte[] bytes = new byte[]:

\rightarrow read(bytes);

* ByteArrayInputStream



* 18|20|24 26 28 32 36 38 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 104 108 112 116 120 124 128 132 136 140 144 148 152 156 160 164 168 172 176 180 184 188 192 196 198 200 204 208 212 216 220 224 228 232 236 240 244 248 252 256 260 264 268 272 276 280 284 288 292 296 298 300 304 308 312 316 320 324 328 332 336 340 344 348 352 356 360 364 368 372 376 380 384 388 392 396 398 400 404 408 412 416 420 424 428 432 436 440 444 448 452 456 460 464 468 472 476 480 484 488 492 496 498 500 504 508 512 516 520 524 528 532 536 540 544 548 552 556 560 564 568 572 576 580 584 588 592 596 598 600 604 608 612 616 620 624 628 632 636 640 644 648 652 656 660 664 668 672 676 680 684 688 692 696 698 700 704 708 712 716 720 724 728 732 736 740 744 748 752 756 760 764 768 772 776 780 784 788 792 796 798 800 804 808 812 816 820 824 828 832 836 840 844 848 852 856 860 864 868 872 876 880 884 888 892 896 898 900 904 908 912 916 920 924 928 932 936 940 944 948 952 956 960 964 968 972 976 980 984 988 992 996 998 1000

