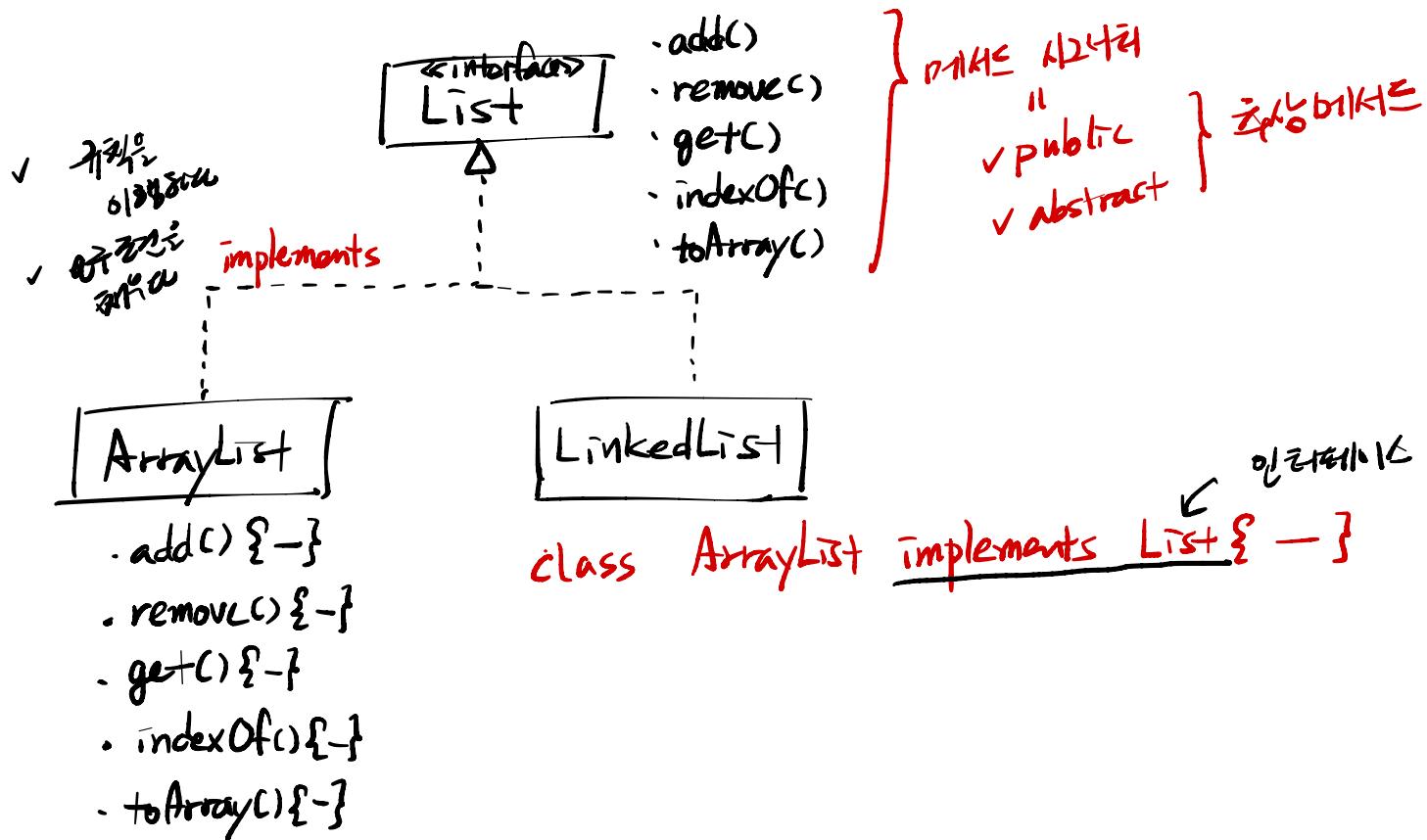
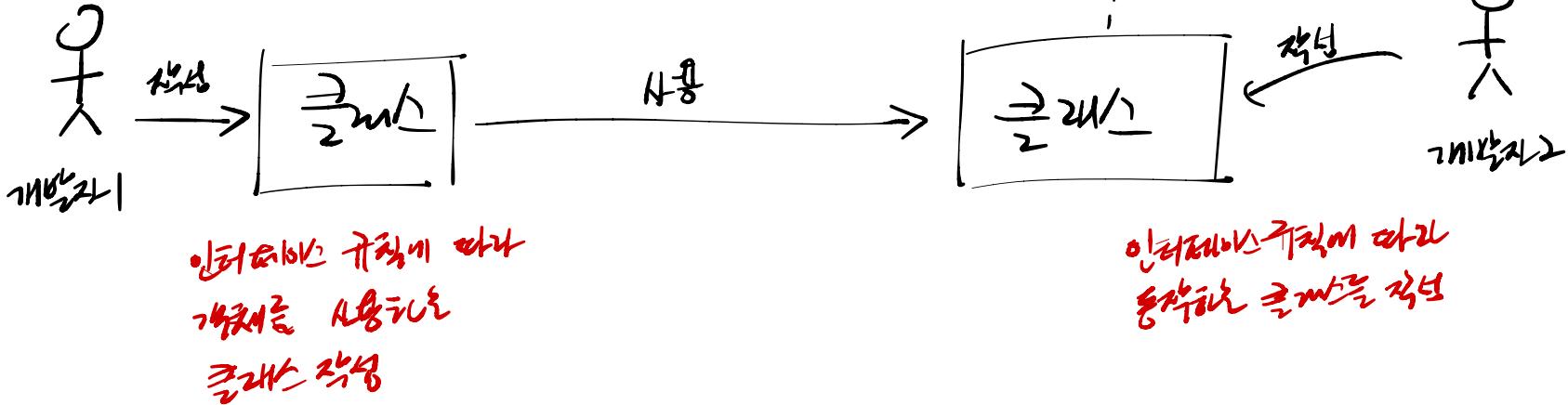
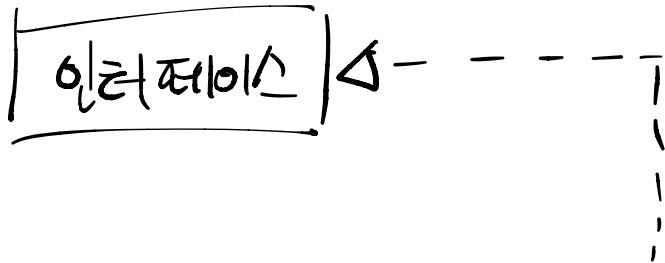


17. 인터페이스를 이용한 구조화된 접근



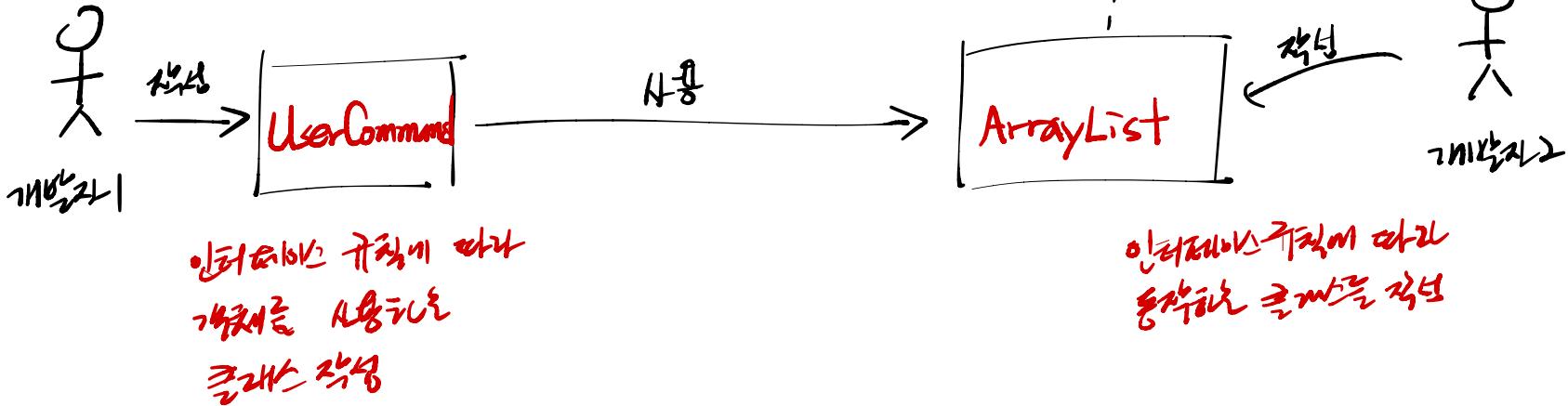
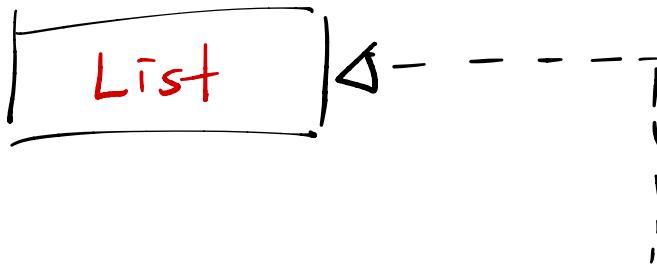
* 인터랙이스

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

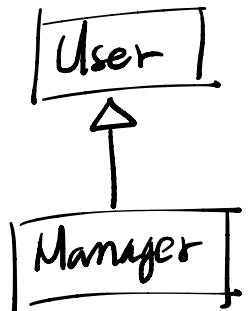


* 인터페이스

인터페이스는 구현체 없이
제공하는 기능의 만족도가 높다.



* instanceof vs getClass()



equals() {
 }
 ==
 }

`equals(Object obj)` {

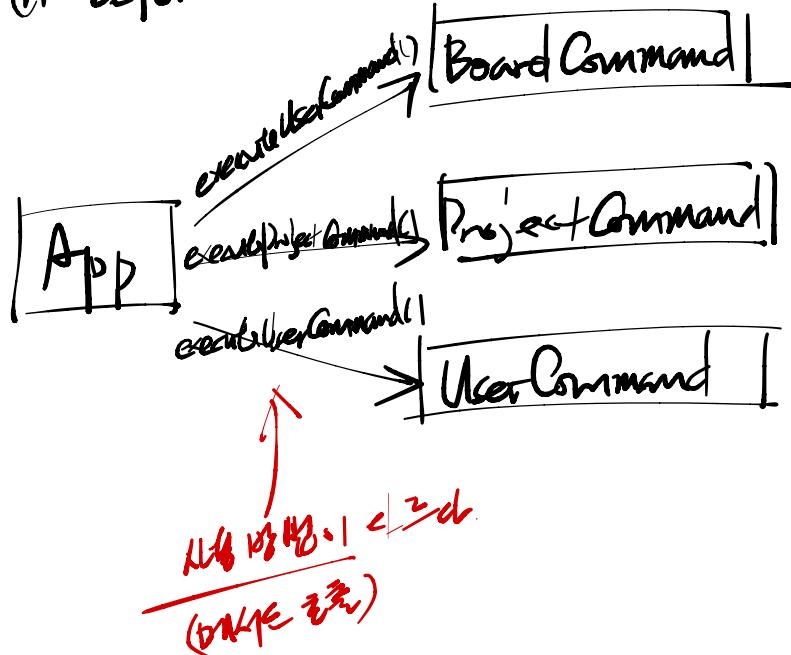
```
if (this.getClass() != obj.getClass()){\n    return false;\n}
```

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

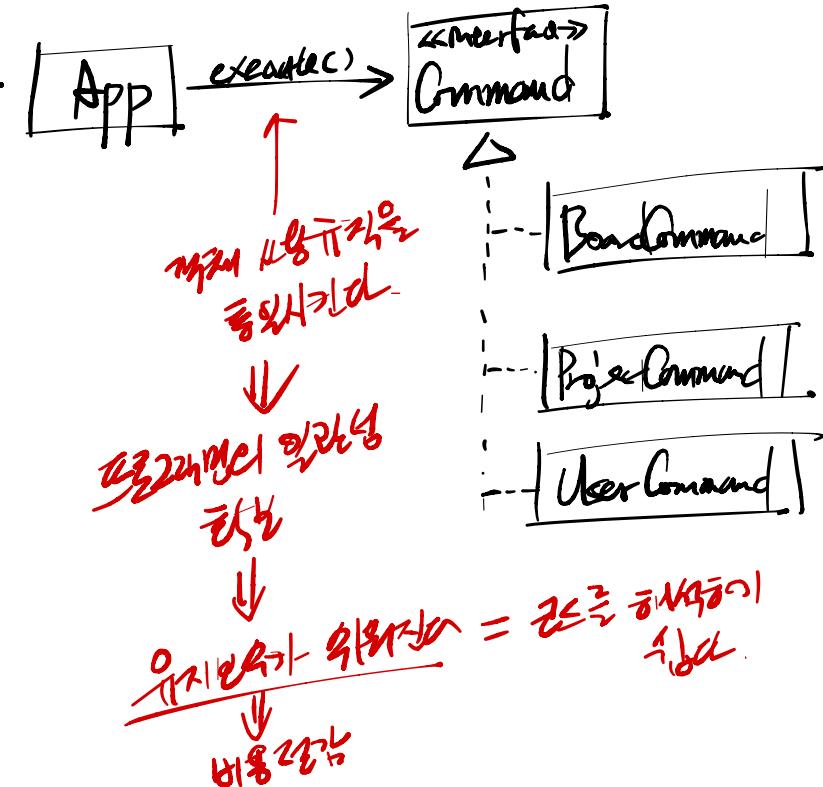
	W ₁	W ₂
ul.equals(str)	F	F
ul.equals(ul)	T	T
ul.equals(m)	F	T

* 121 능력과 122 번의 학습자료

① before

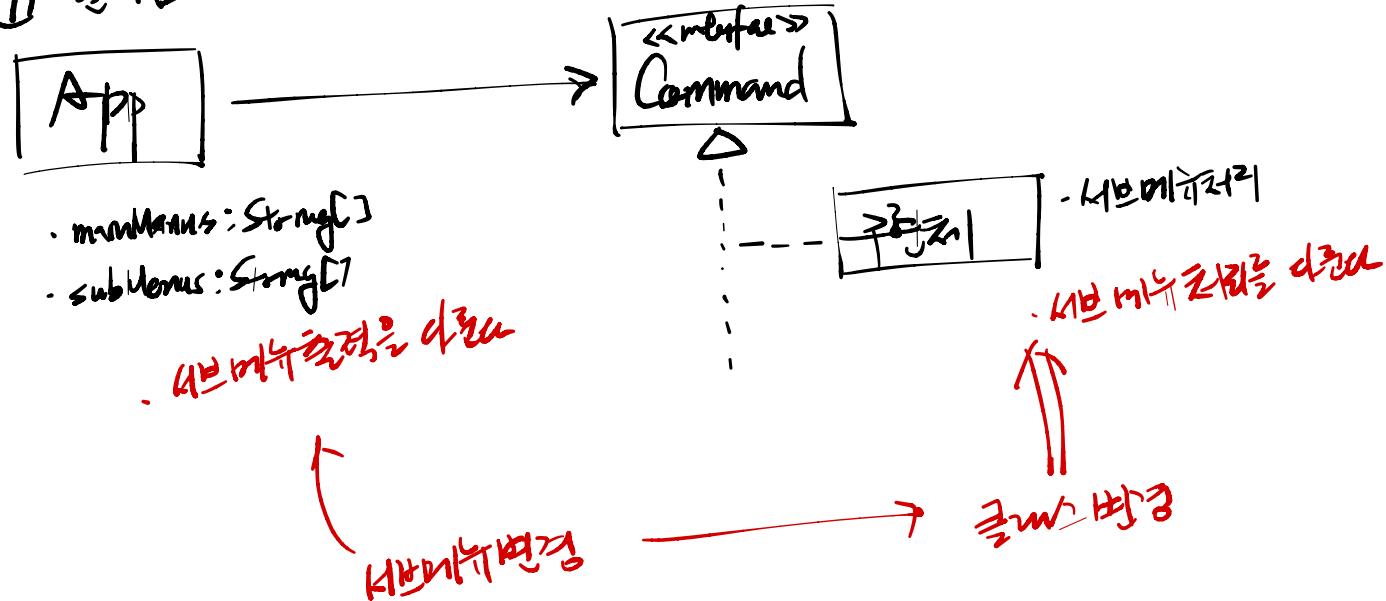


② after



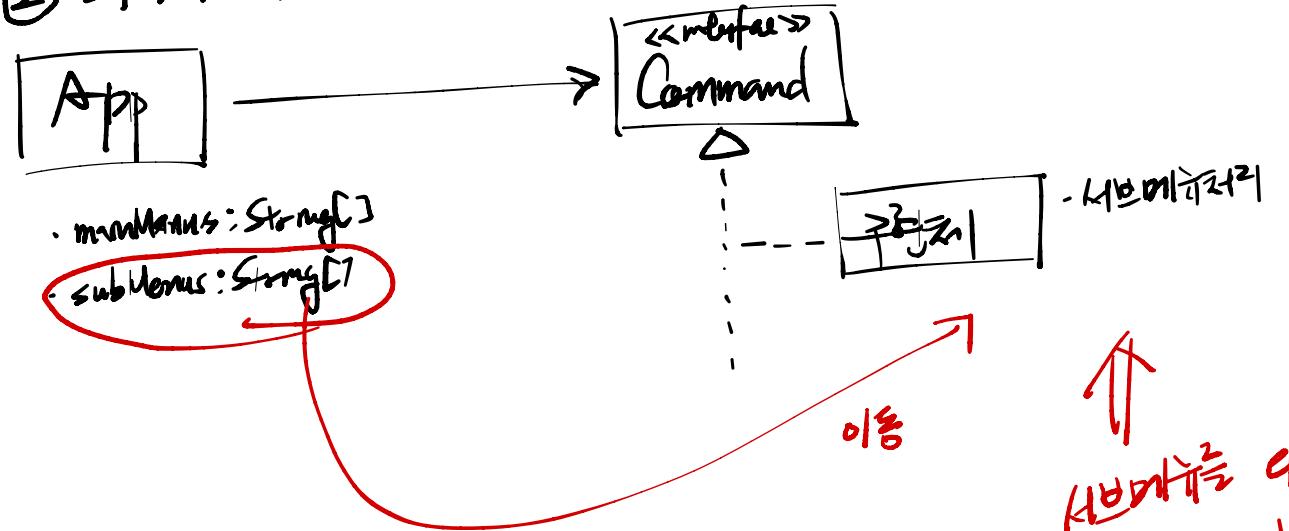
18. 커맨드 패턴

① 응용



18. 디자인 패턴

② 명령 : GRASP의 High Cohesion & Low Coupling



이동

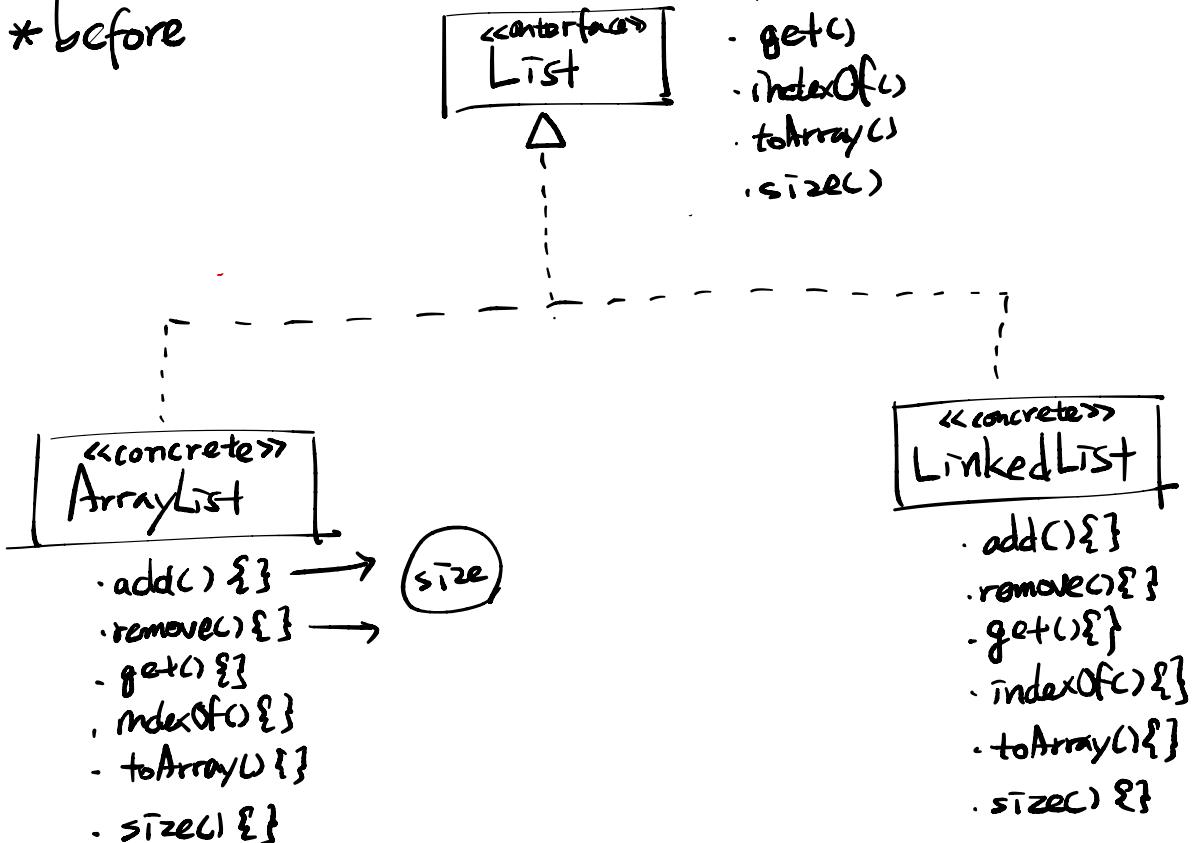
- 메뉴판(메뉴판)

↑
메뉴판을 다루는 역할
Command 구현체로
변경됨.

||
유저에게 응답한다

19. 상속의 Generalization - ①

* before



19. 상속의 Generalization - ①

* after

①

상속을 통한
공통 기능은 대상으로
설정하는 경우
하는 경우

②

인스턴스 사용하기
조작 가능성이 있다!



- add()
- remove()
- get()
- indexOf()
- toArray()
- size()
- size: int
- size() { } →



- add() { } → size
- remove() { } → size
- get() { }
- indexOf() { }
- toArray() { }

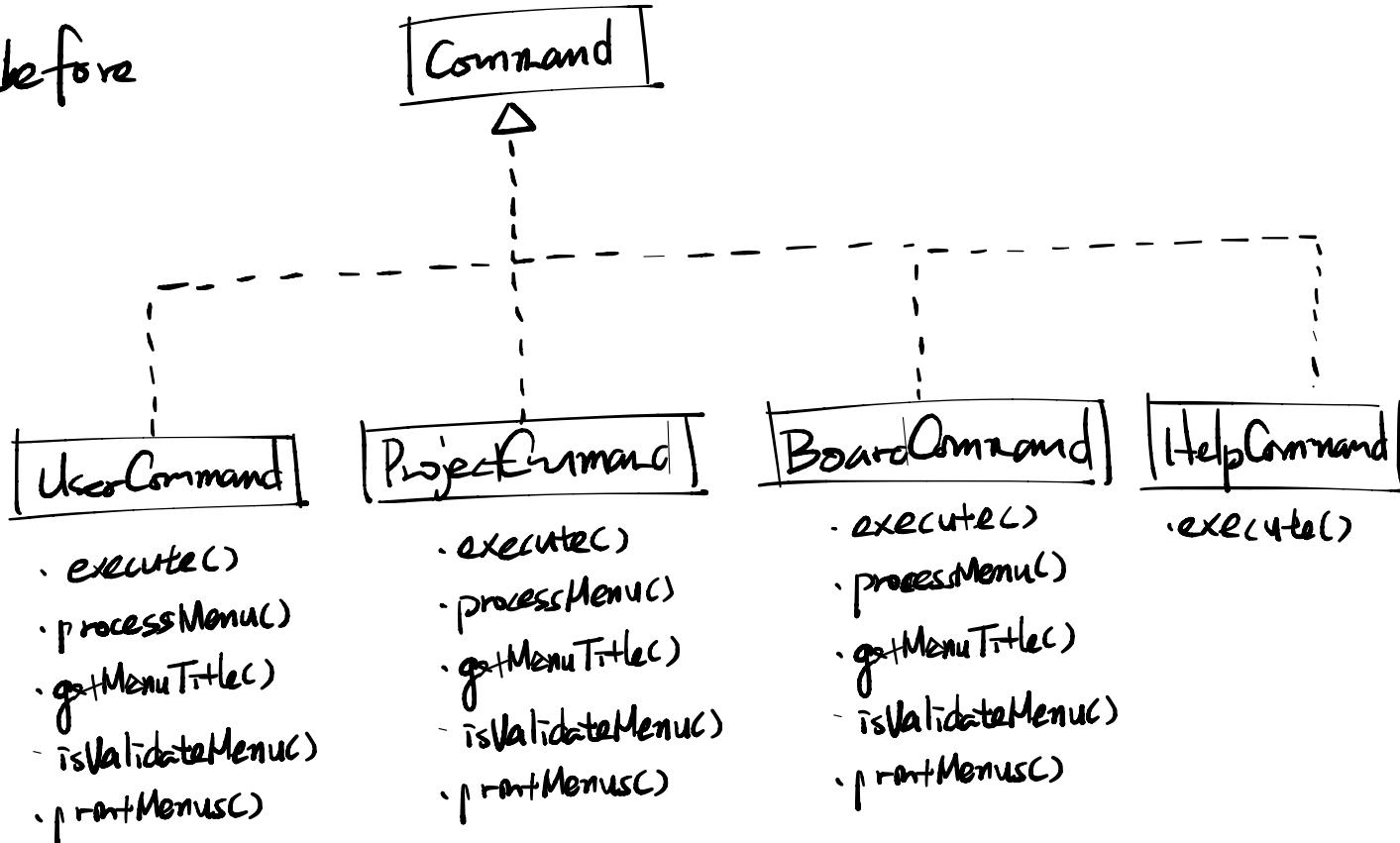


- add() { }
- remove() { }
- get() { }
- indexOf() { }
- toArray() { }

인스턴스를 생성해서
변수 값을 >= m↑

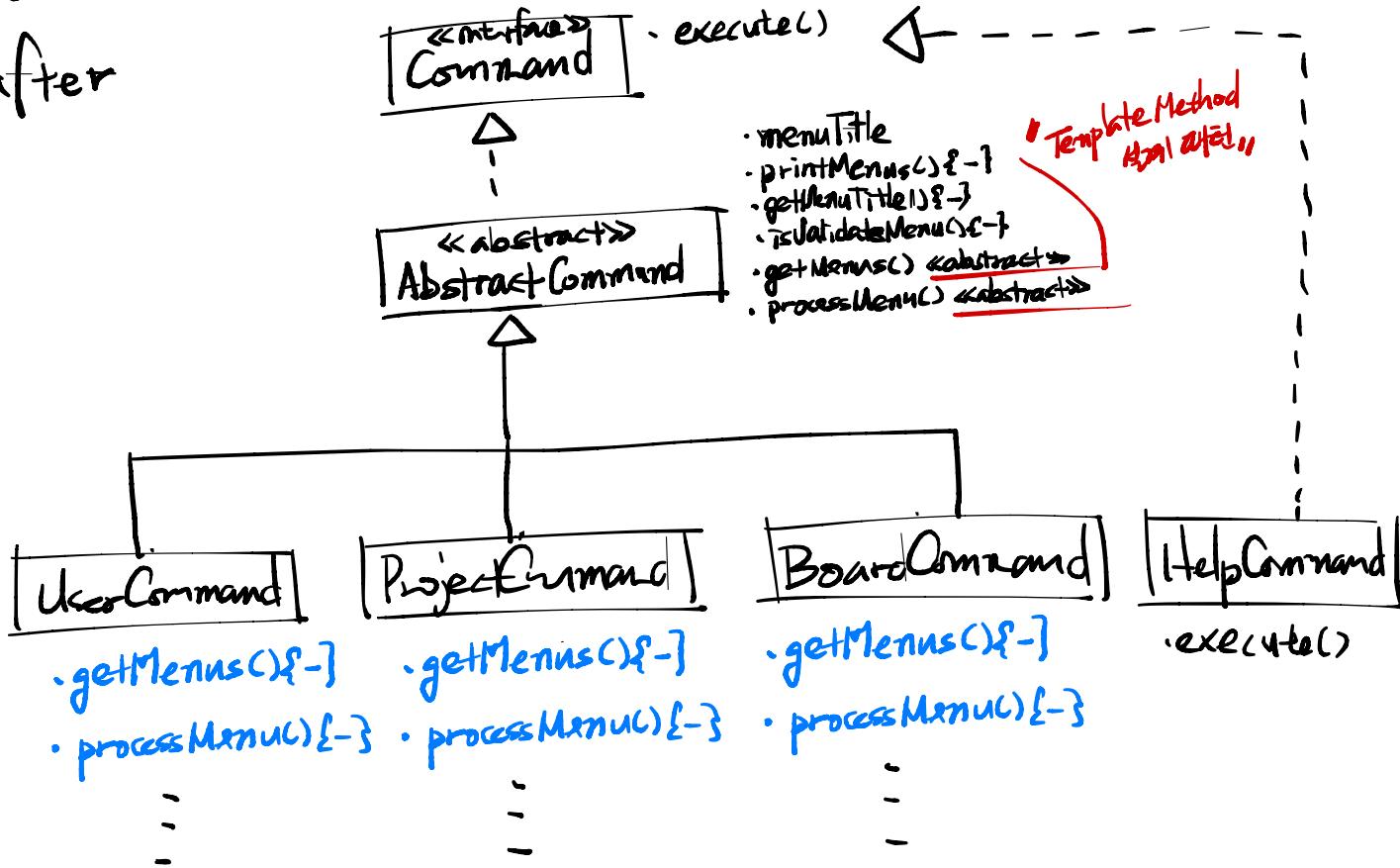
19. 상^k으로 Generalization - ②

* before

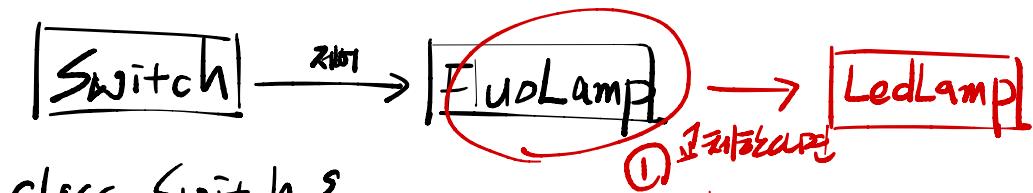


19. 一般化 Generalization - ②

* after



20. SOLID의 DIP와 GIRASP의 Low Coupling



class Switch {

Fluolamp light;
}
=
② 반드시 연결해야 한다.

- ③ Switch 클래스
Fluolamp 클래스와
상호작용 되어야
④ "강한 단점 속성" \Rightarrow 해결?

20. SOLID의 DIP와 GIRASP의 Low Coupling

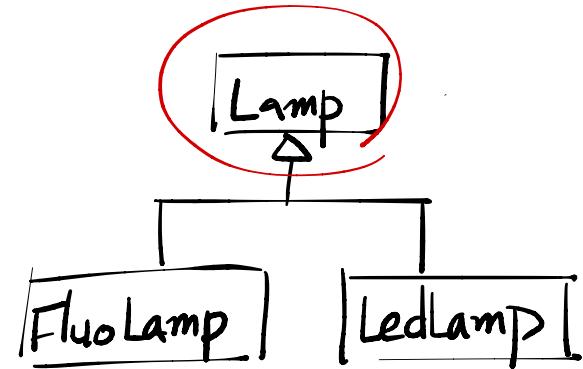


class Switch {

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 {

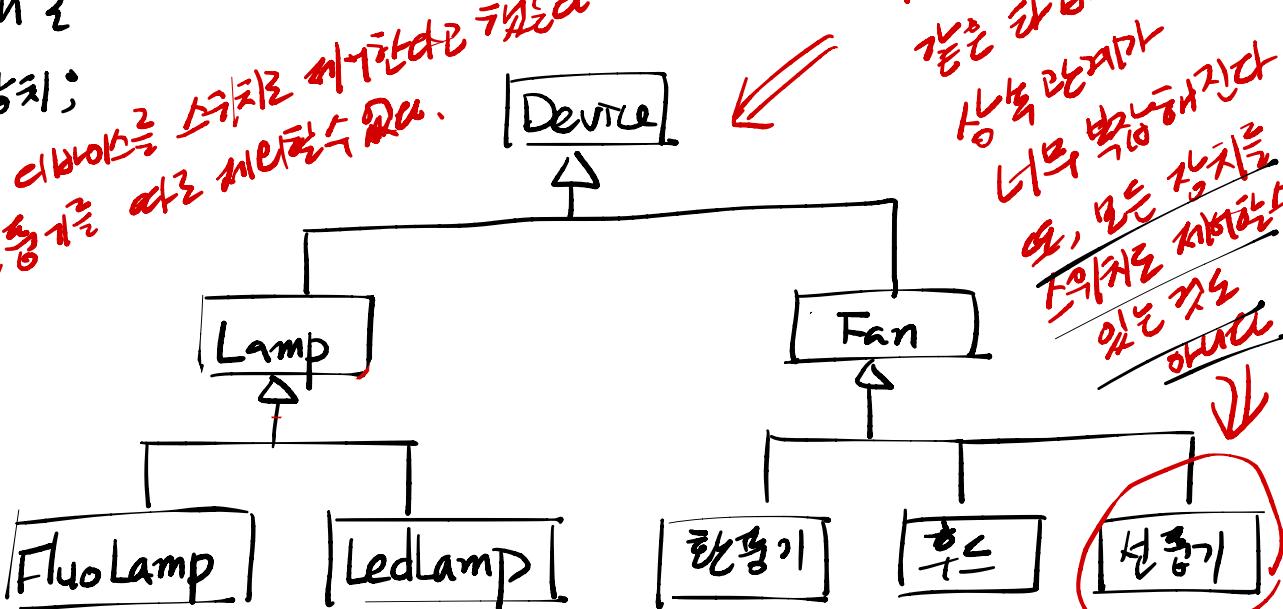
 Device 장치;

}

*② 모든 클래스를 스위치로 연결하는게 아니라
선택적으로 연결하는게 맞다.*

① 여러 장치를 스위치로
제어하고자 하는
경우 타입으로 묶은 모듈
내부 장치가
너무 복잡해지다
오, 모든 장치를
스위치로 연결하는
있는 것도
아니다

③
상속은 "캡슐화"의
유지보수 예요.
유연성 부족.

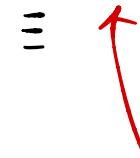


20. SOLID의 DIP와 GIRASP의 Low Coupling



class Switch {

 Switchable 광고;



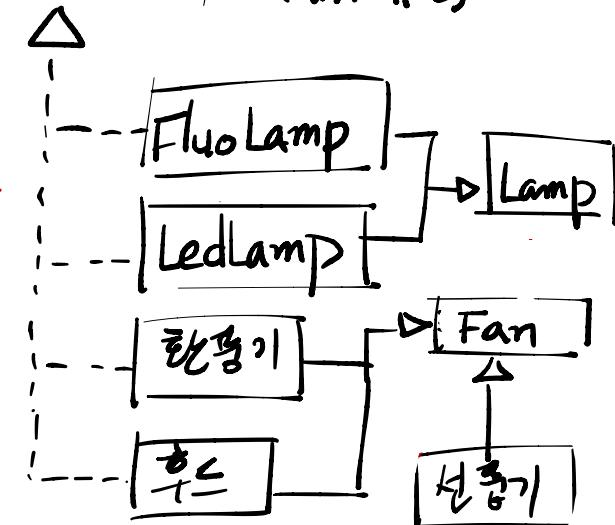
② 어떤 타입이든
Switchable 인터페이스
를 상속하는 것을 가능하게
만들기 가능

Switchable 인터페이스

- turnOn()
- turnOff()

① 품목마다
구현해야 하는
수행방법

같은 추상이 아니
구현체들도
다르므로
제작할 수 있다



↳ ② 품목마다 더 유연하다 “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 역할 분리)

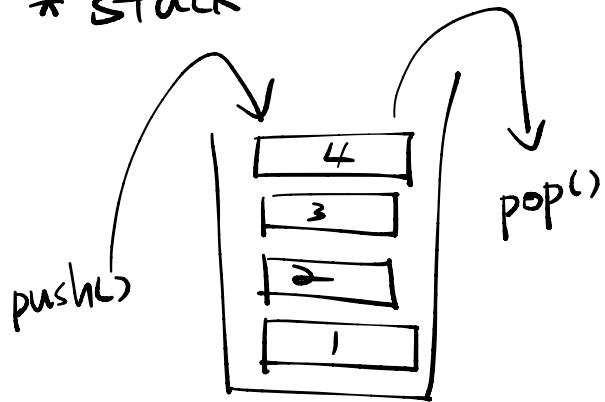
(제어권 분리)

Inversion of Control (IoC)

- ① Dependency Injection
② Listener = event handler

21. 자료구조 - Stack 와 Queue

* stack



First In Last Out

(FILO)

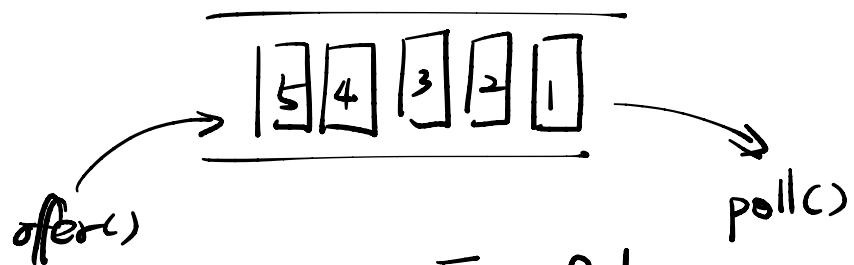
"

Last In First Out

(LIFO)

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

* Queue



First In First Out

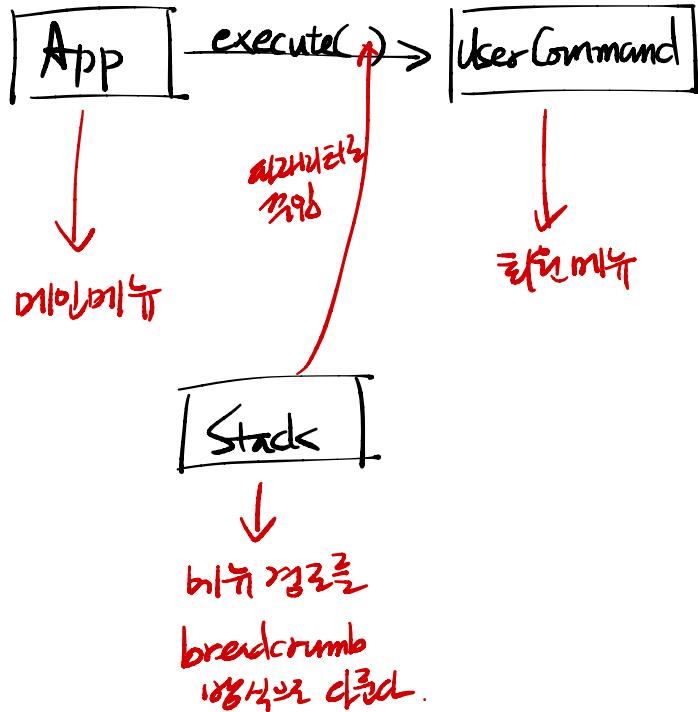
(FIFO)

① 예상

② 예상하지 못한 예상으로 예상 처리

③ 이벤트 처리 = Event Queue

* 디足迹 제목을 소리으로 하기



* String

String str = "";

str += "aaa";

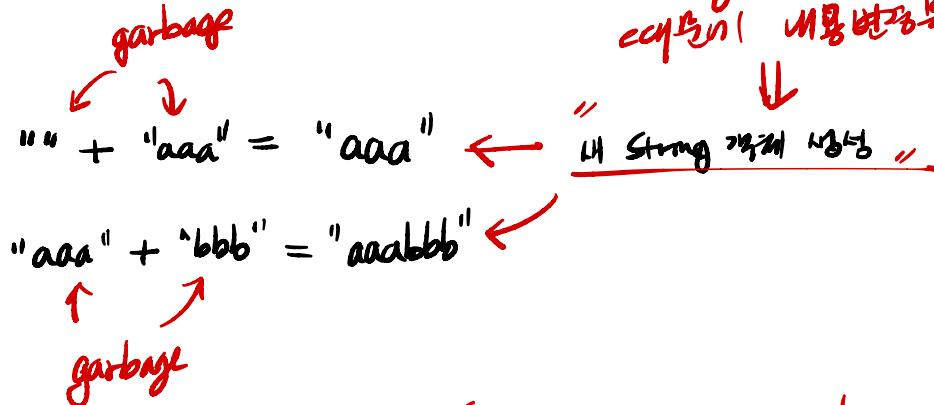
str += "bbb";

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

thread-Safe

스레드 안전한 이유
↳ lock/unlock
함수를 사용하는

StringBuffer, StringBuilder



문자열은 대체로 같은 1H String은 안전하지
않은 String은 항상 garbage가 있다.
문자열은 항상.

↳ thread-Safe



mutable처럼 사용

thread-Safe
↳ 스레드 안전한
↳ 스레드 안전한
↳ lock/unlock을 통해
↳ obj锁을 통해
↳ thread-safe

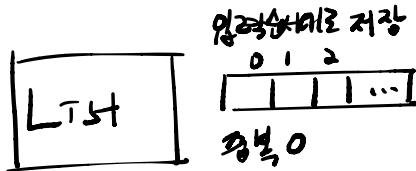
22. Iterator 깊이 파악

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

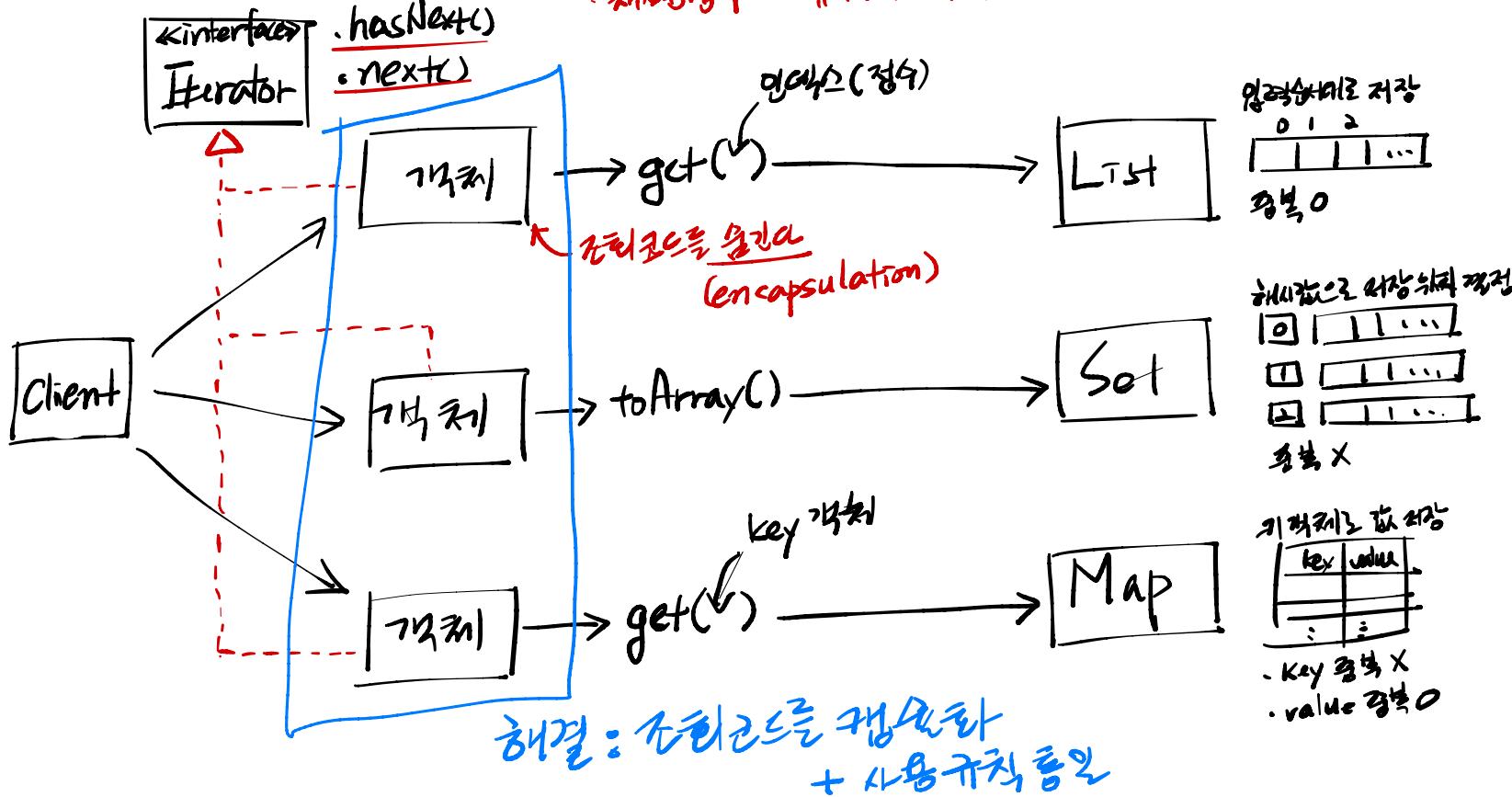
key	value

- Key 정복 X
- value 정복 O

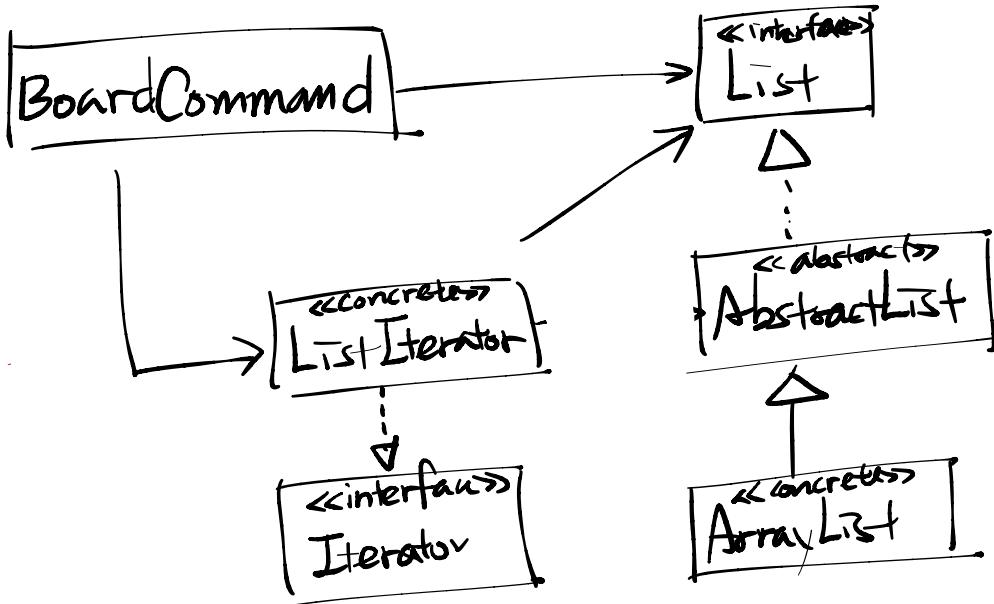
22. Iterator 기능적 패턴

이전 문서화하는 강제된 흐름 방식

· 재사용성↑ · 유지보수성↑ · SOLID / GRASP 부합

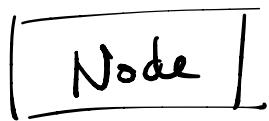


22. Iterator 틀 구조



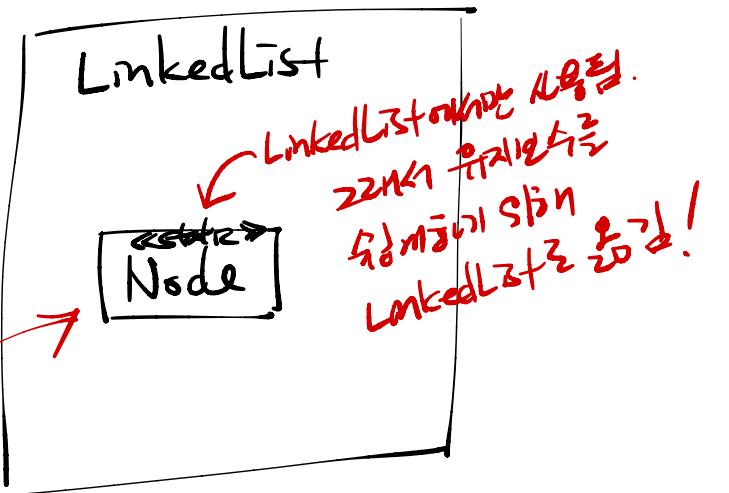
23. 중첩 클래스

before



↑
package member class

after

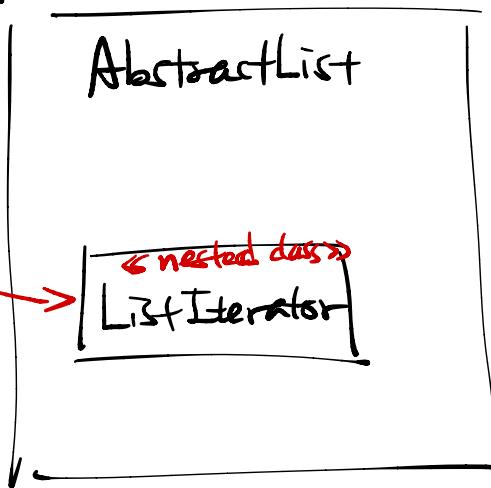


23. 중첩 클래스

before



after



* enhanced for 문법

for(변수선언 : 배열 또는 Iterable 구현체)

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

for(String name : names) { }

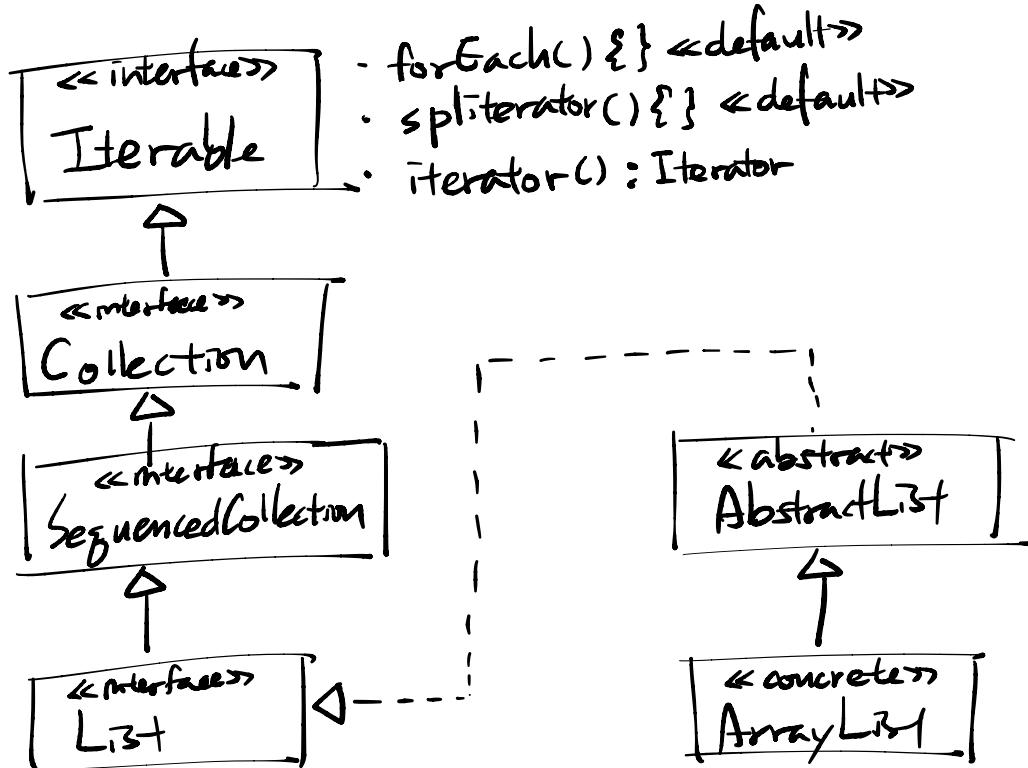
변수

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

for(Object item : list) { }

Iterable 구현체

* Iterable ↗^{3연자}



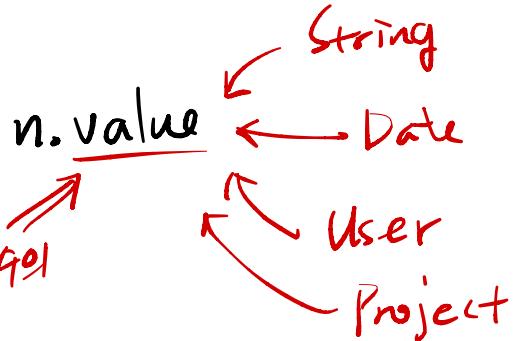
24. Generic 타입 사용하기

```
class Node {
```

```
    Object value;
```

특정 타입의
인스턴스
만들기 위해
제한할 수
있을까?

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



다형화 변수의
타입
하나의 타입의
제한을 해야
하는 경우.

⇒ Generic 타입

24. Generic 블록 사용하기

class Node<what> {

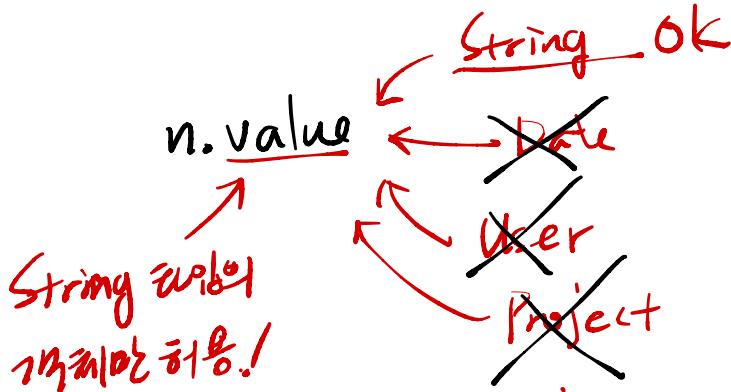
what value;

what이
어떤 타입인지

선택할 때
제한한다

타입 제한자 = 타입 정보를 넣은 변수

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



제한자는 아니
타입 제한자로써는
타입 제한자로
제한 가능

* Type Parameter

↑ 타입 명보를 뱉는 변수

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

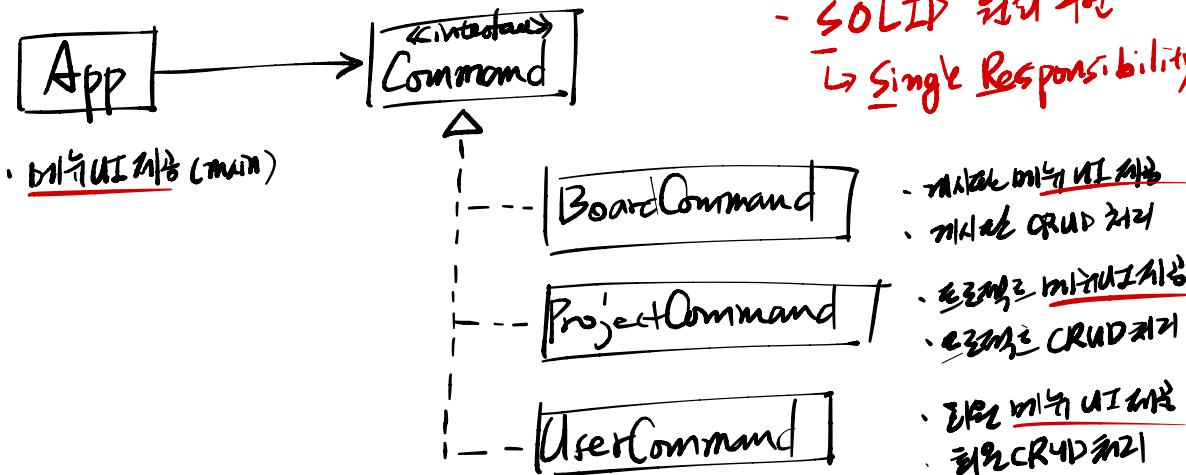
26. GoF의 Composite 패턴 대안

* before

↳ 개체가 트리 구조로 포함 관계임.

문제

- 메뉴나 UI 툴은 굳은 결합
 - Command 패턴과 여러 모의 일을 처리
- 대안
- 멀티 UI 툴은 굳은 결합 → 개체를 분리
 - SOLID 원칙 적용
 - ↳ Single Responsibility Principle



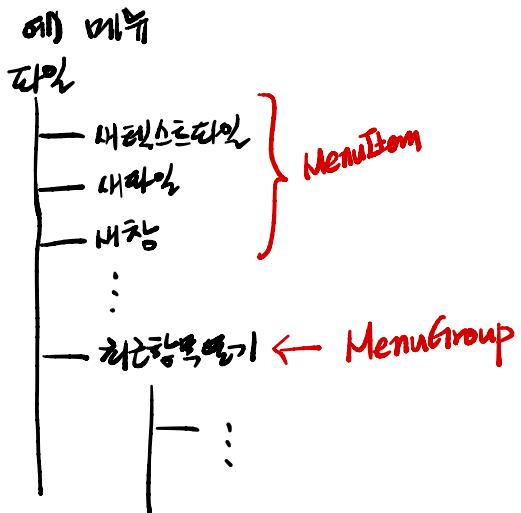
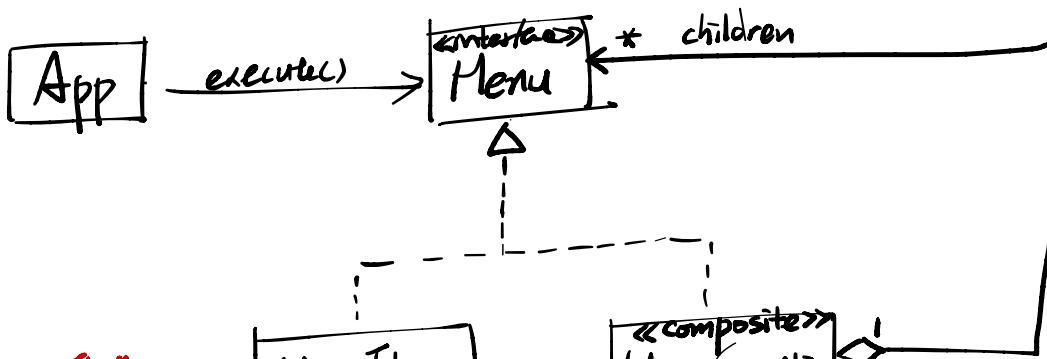
- 기존 UI 툴은 굳은 결합
- 기존 UI 툴은 굳은 결합
- 프로젝트 멀티 UI 툴

26. GoF의 Composite 패턴

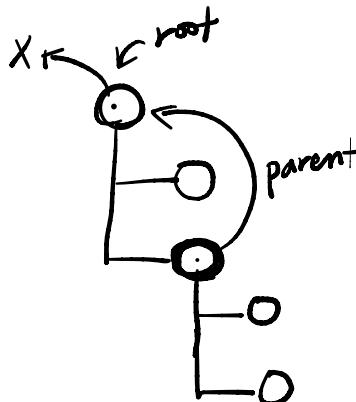
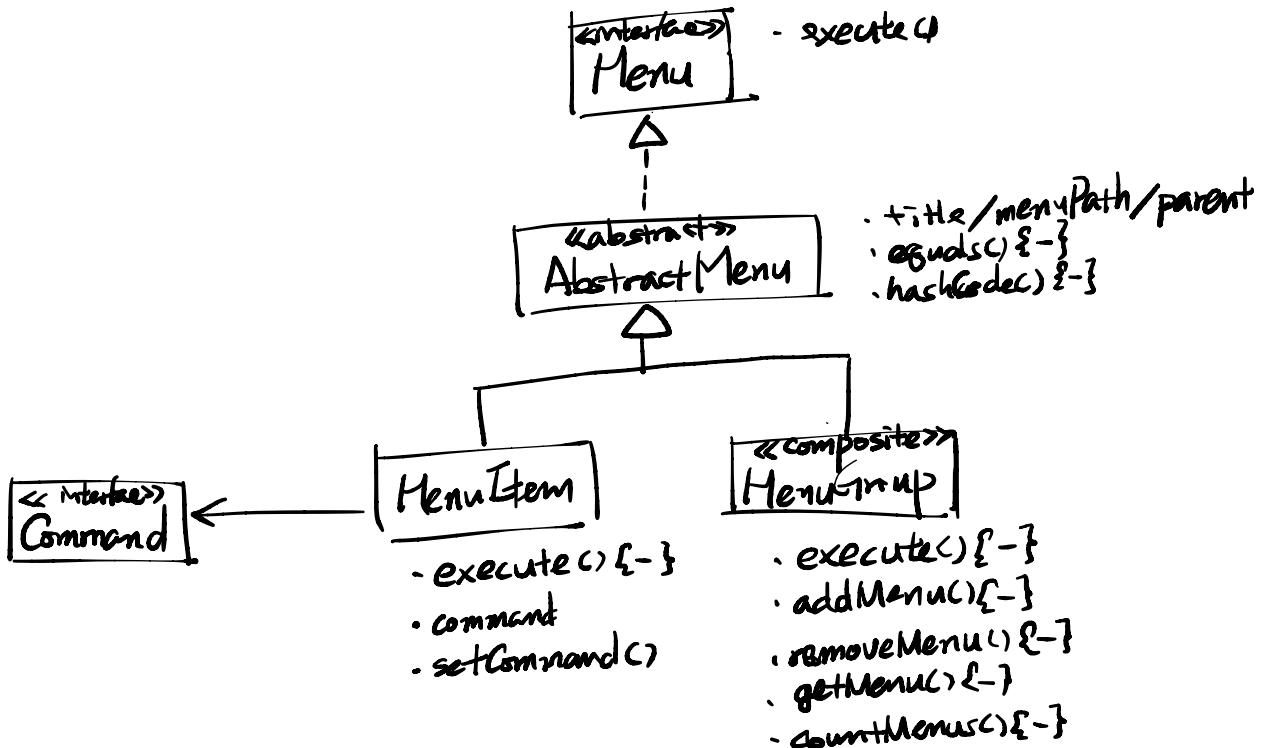
* after

↳ 개체가 트리 구조로 표현 관계임. ↳ ② 파일 시스템

- { ① 메뉴
② 조직도
③ 팀원 }

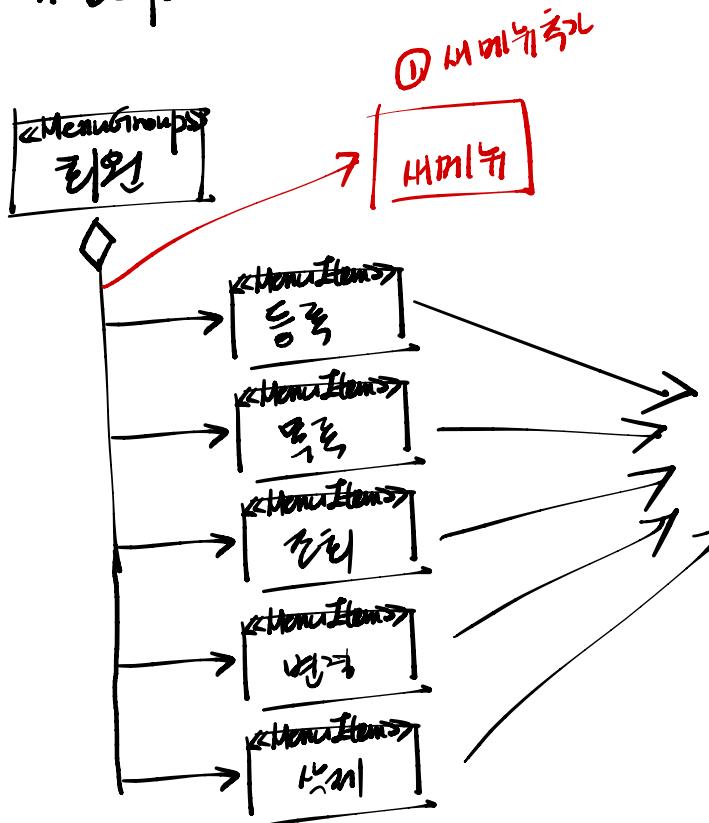


26. GOF의 Composite 패턴



27. 메뉴의 메뉴추가 기능을 개선하자 : GoTo Command 패턴처럼

* before



① MenuItem 추가
② MenuItem 처리할
코드를 추가
↓
SOLID의 OCP 원칙을
위반함

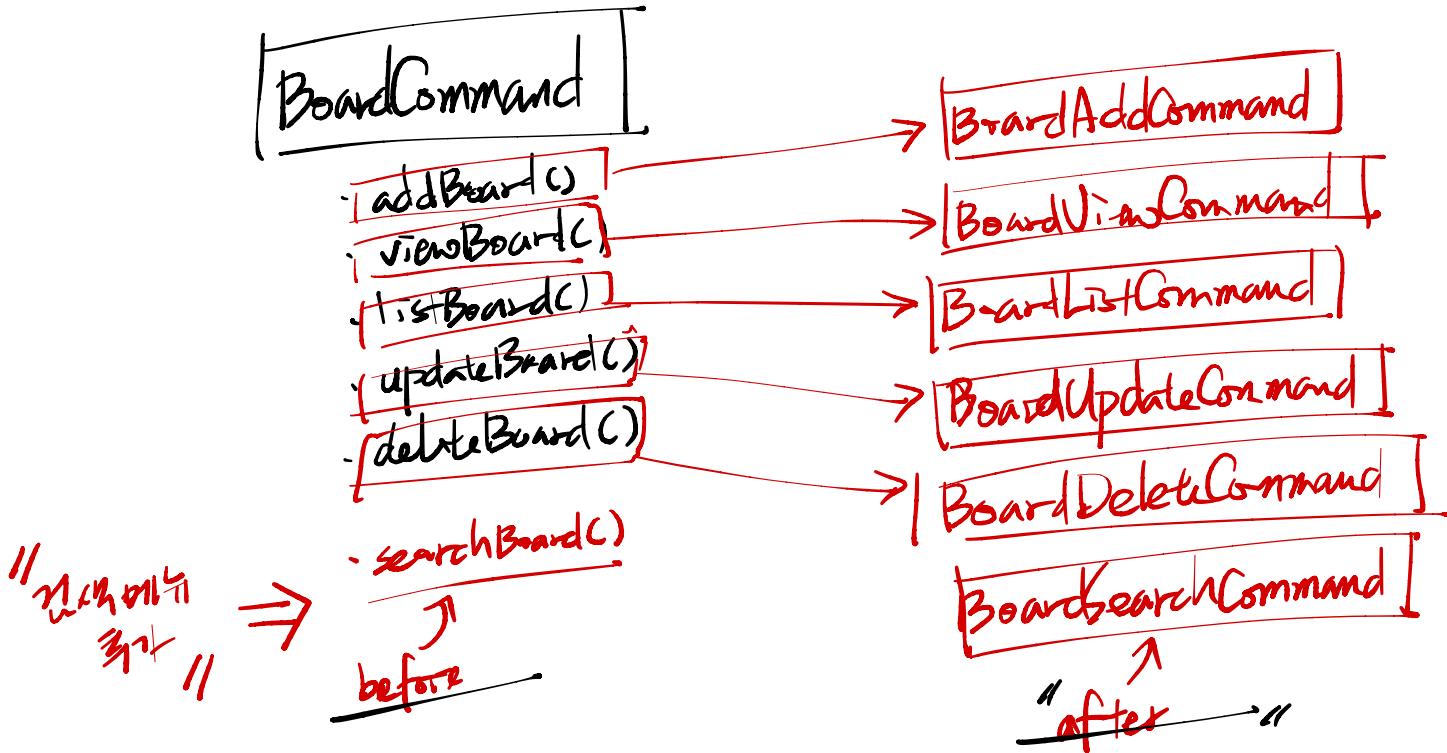


- MenuItem { }

기능 추가 / 버그 수정 시
기능이 많아 코드 관리가
어려워짐.

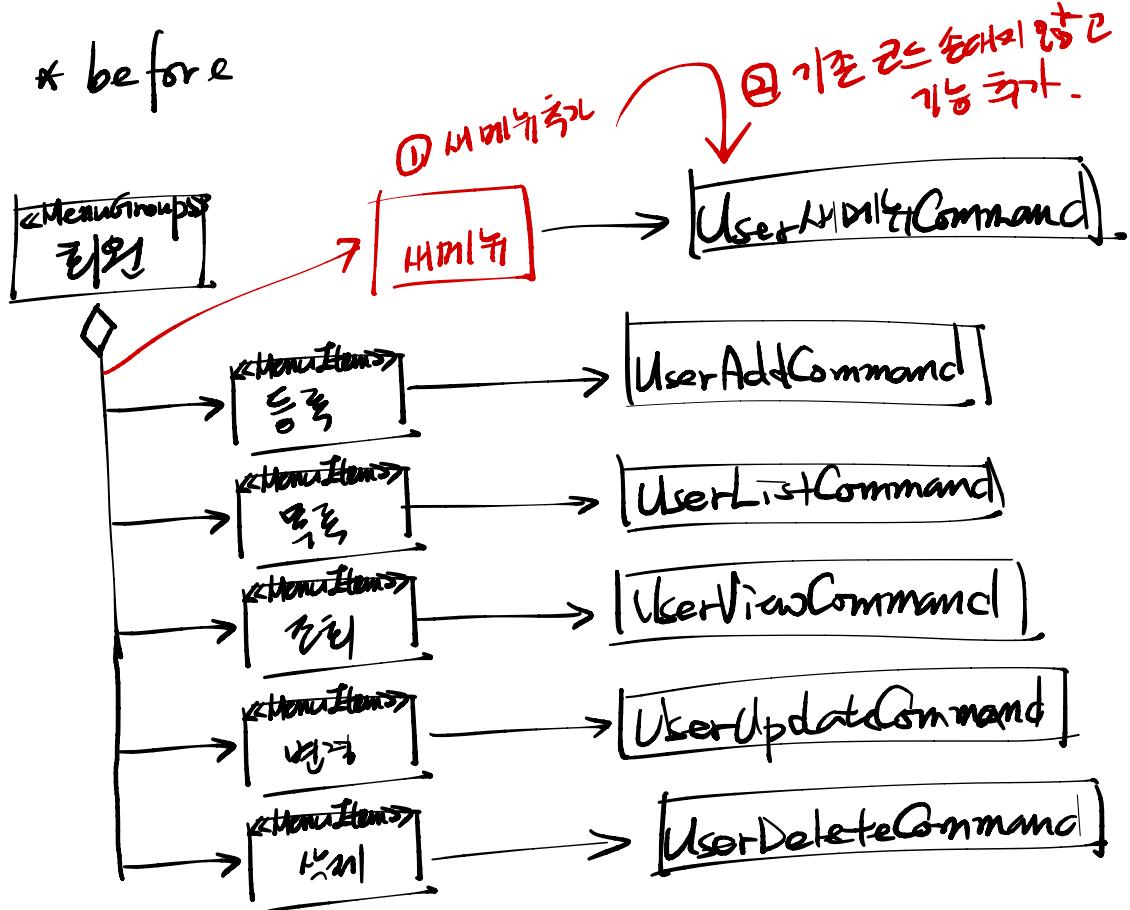
↑
기능 증가
↓
기능 증가

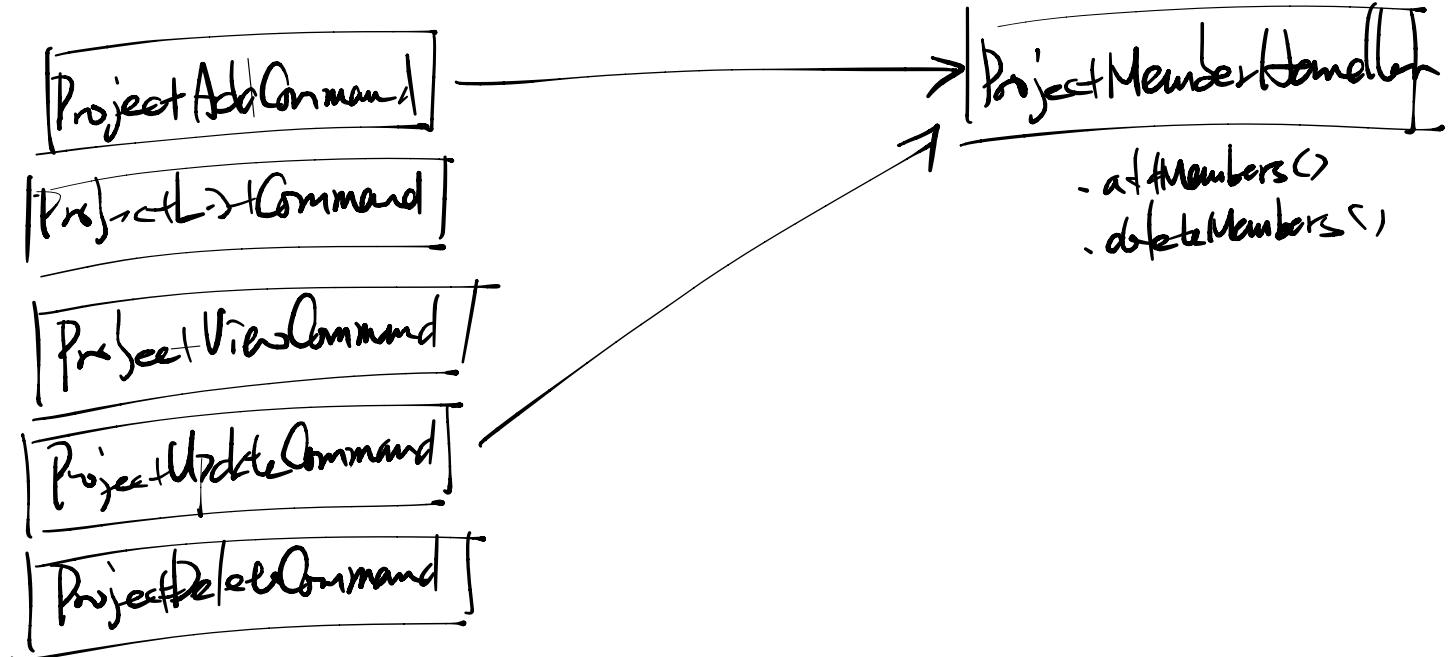
27. 게시판의 목록관리 기능을 구현하자 : GoTo Command from list
↳ list → list



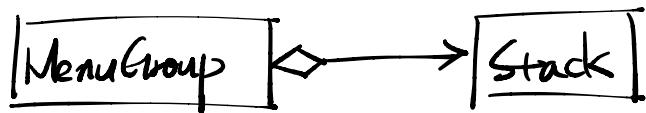
27. 메뉴의 명령처리 기능을 구현하자 : GoTo Command 패턴 틀

* before



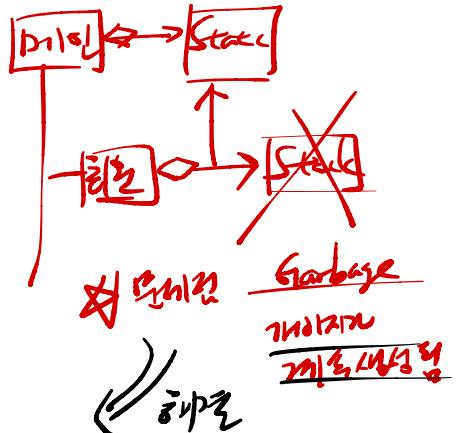
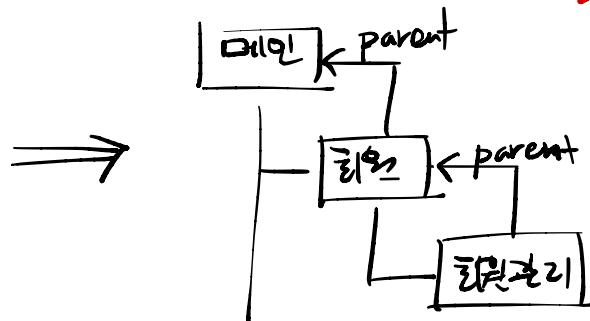
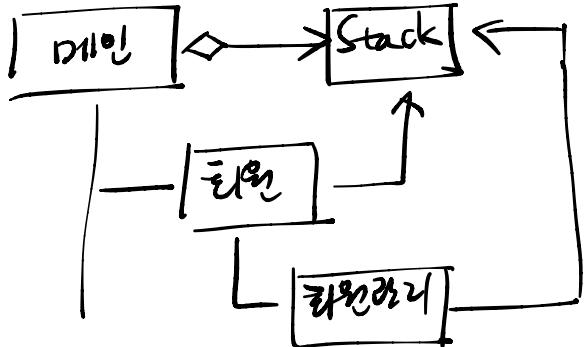


* MenuGroup 의 경로 메시지를 MenuItem로 분리



• 메뉴이동은 단순히 맵

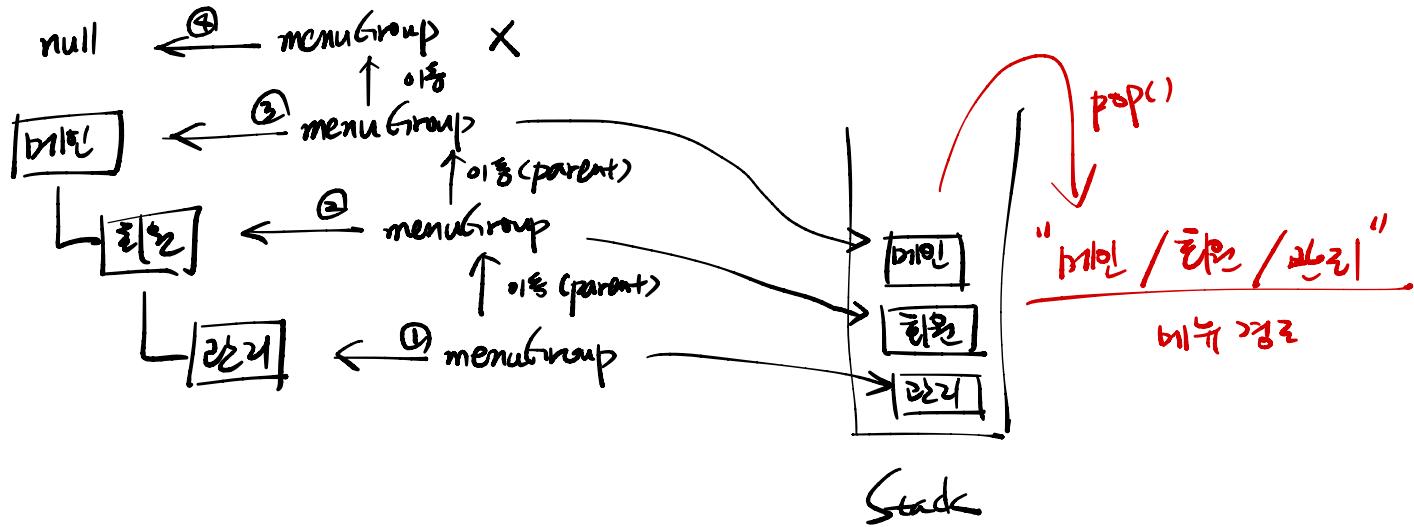
①)



✓ getMenuPath() Root ->

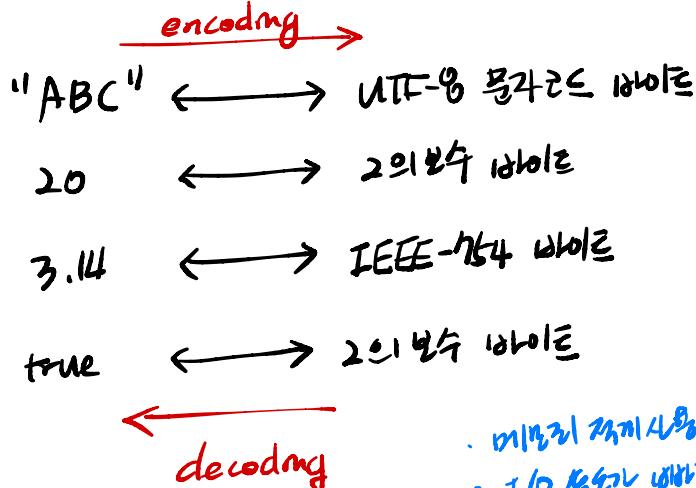
메뉴경로를 메시션

→ getMenuPath() 구조 예시



28. File I/O API 활용 : ① 데이터형의 데이터 암호화

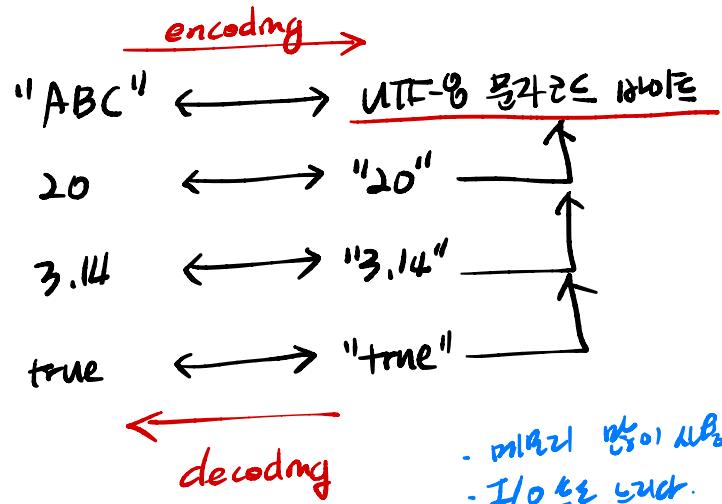
* binary data I/O



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

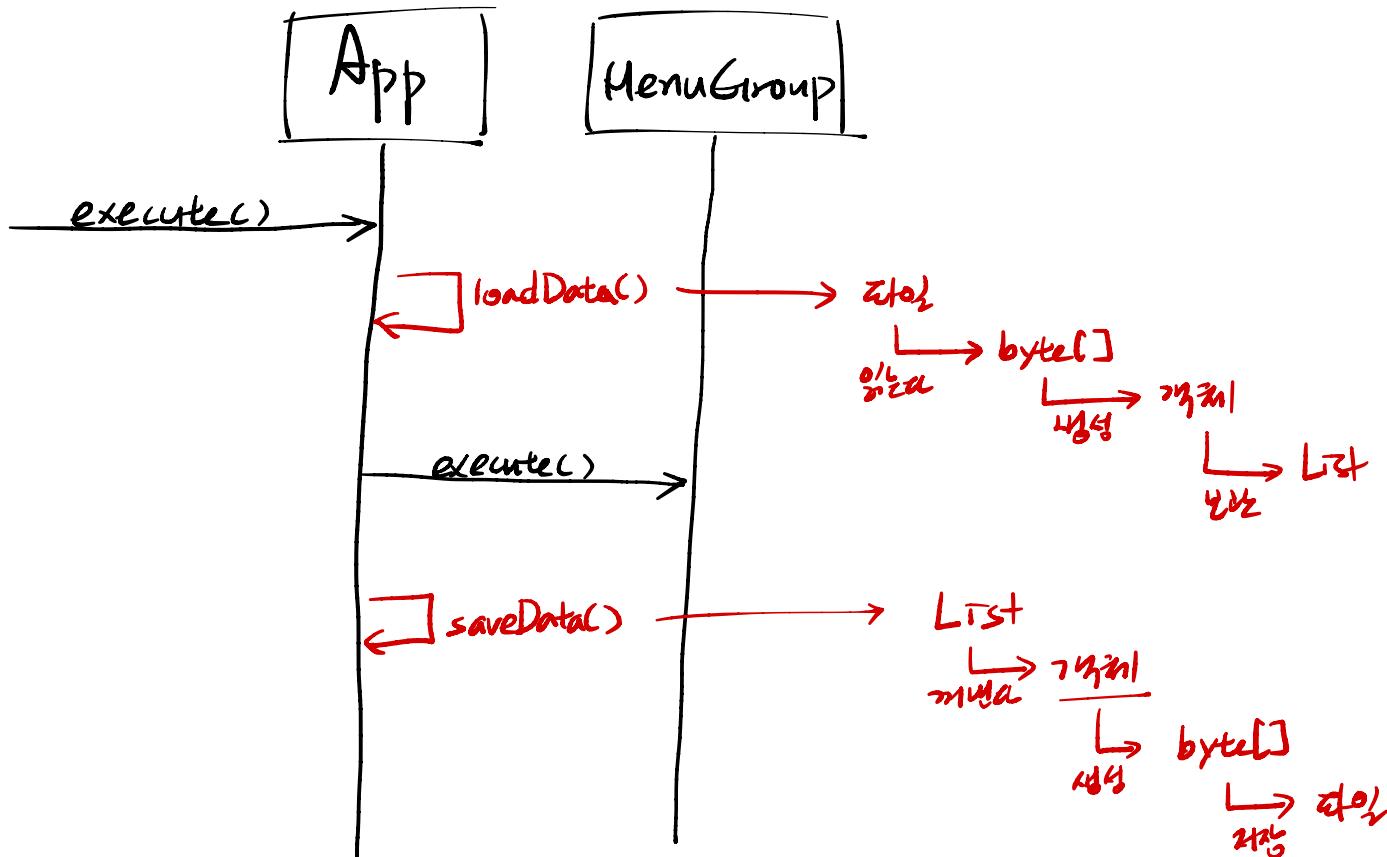
↳ 진정한 데이터를 이용한 I/O

* text data I/O



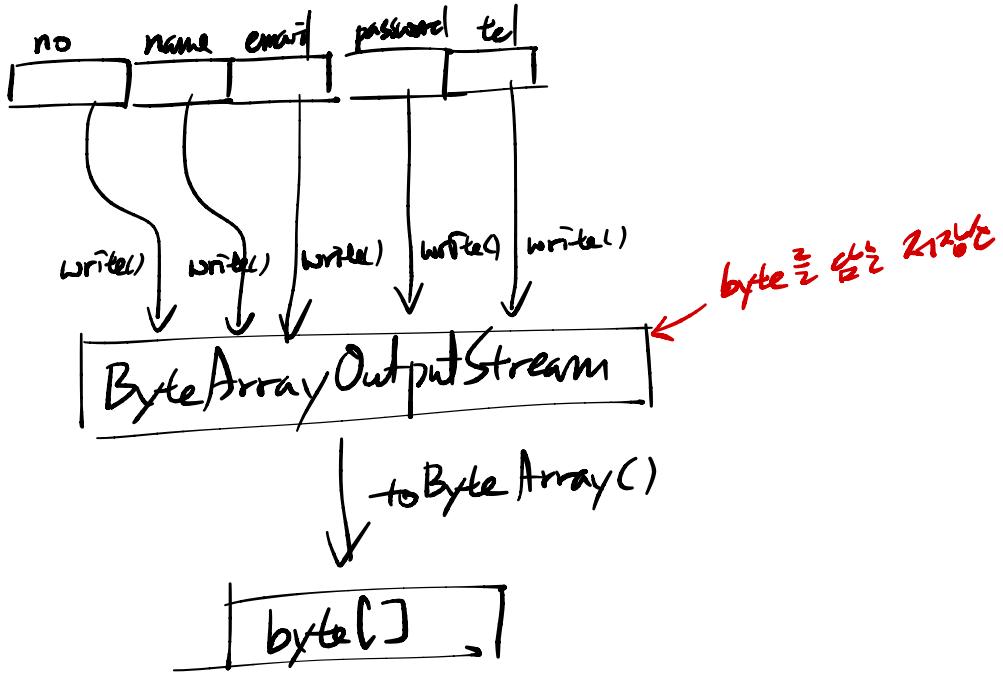
- ②) TXT, HTML, CSS, JavaScript, XML,
Properties, JAVA 등

↳ 텍스트 형식으로 이용한 I/O



* گذاشتیم که byte[] یک مجموعه

User گذاشتیم



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

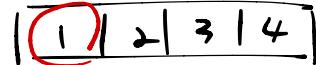
↳ 32비트 | 10101010101010101010101010101010



1byte 단위
32비트

write()

↓ 16비트(1010101010101010)만 쓸 때



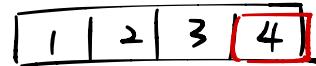
→ 1bit 이용
→ write()



16bit 단위
→ write()

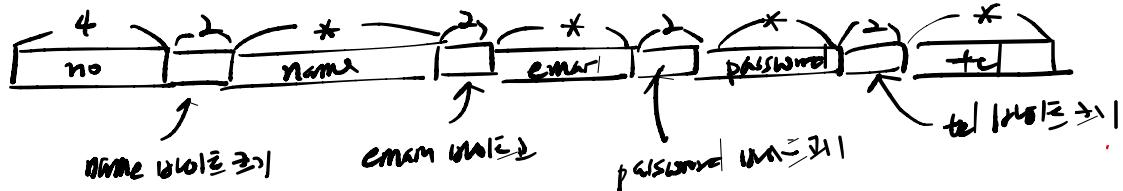


8bit 단위
→ write()

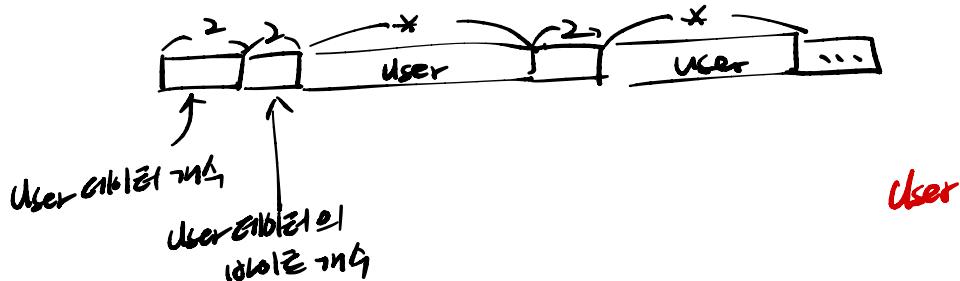


→ write()

* 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

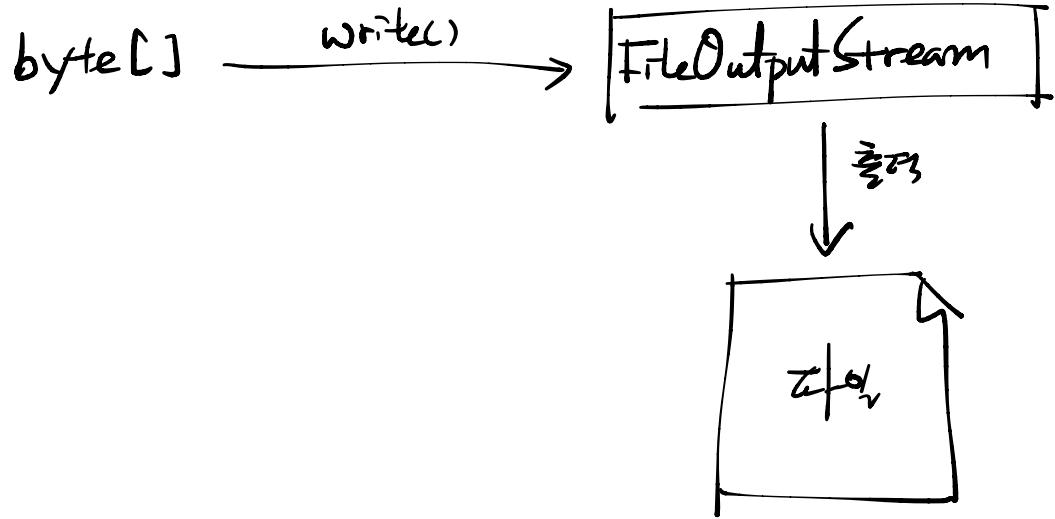
2 byte - password

* byte

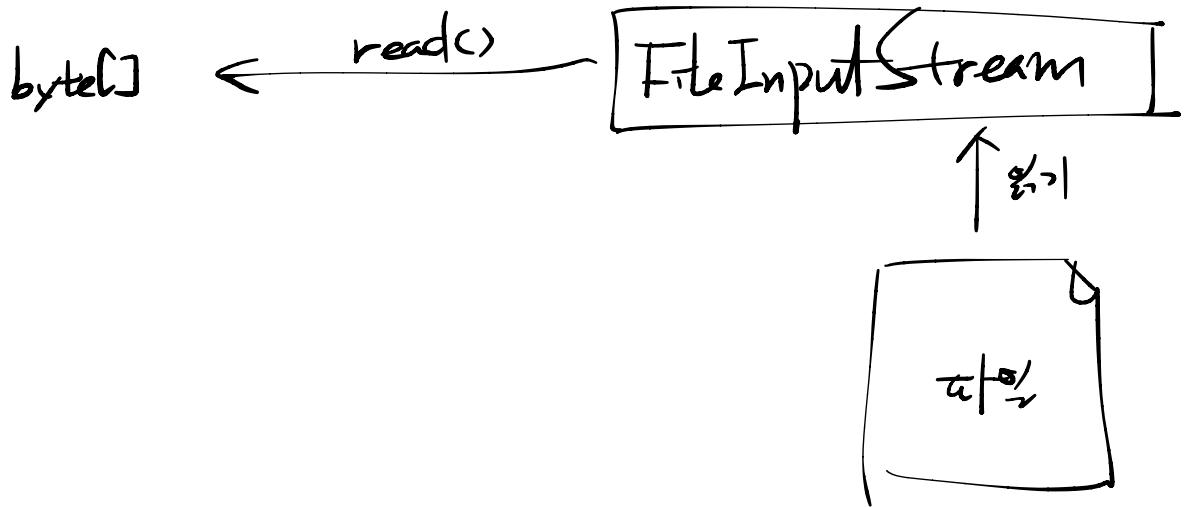
2 byte - tel

* byte

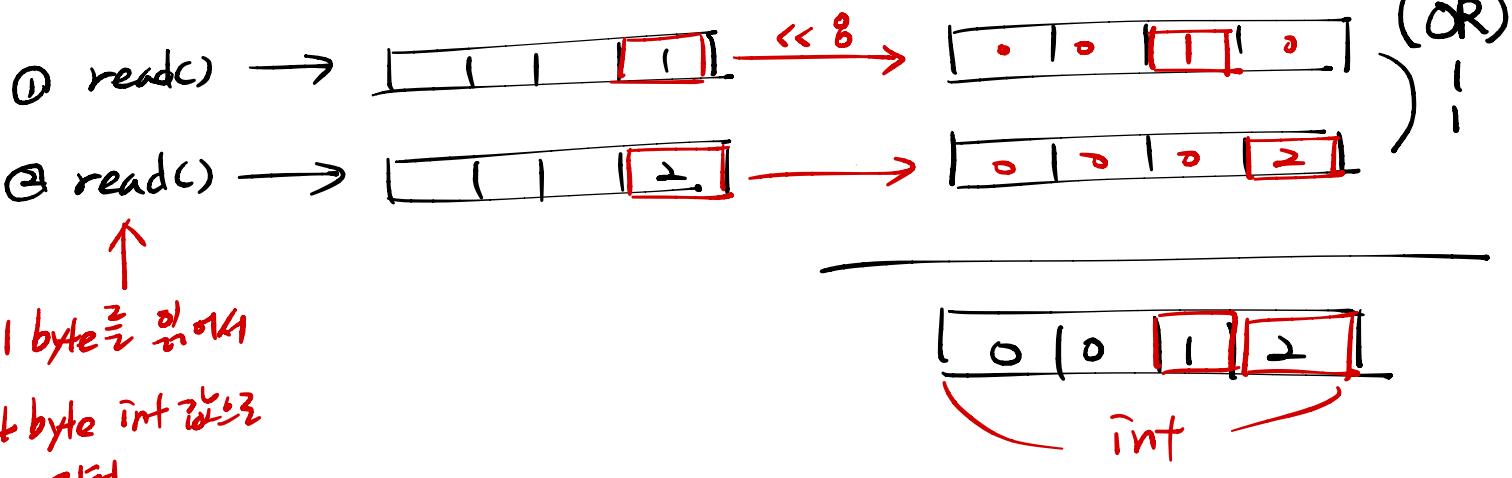
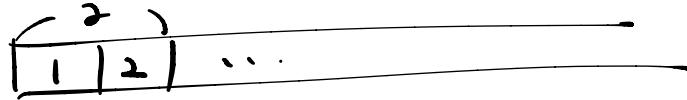
* `byte[]` $\xrightarrow{\text{写出}}$ `txt`



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



* `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

•

•

•

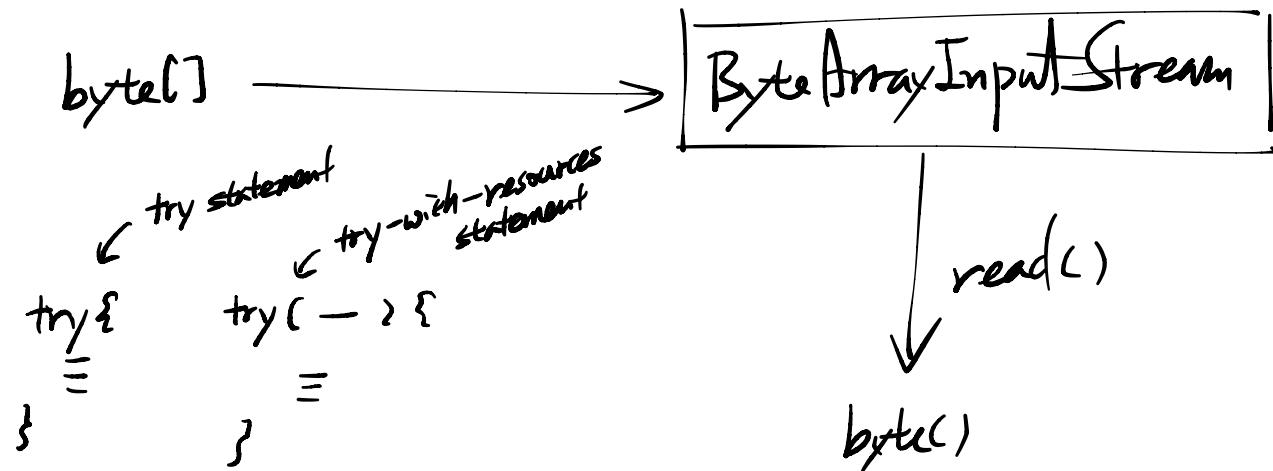
(read() << 8) | read() \rightarrow int

(read() << 8) | read() \rightarrow int

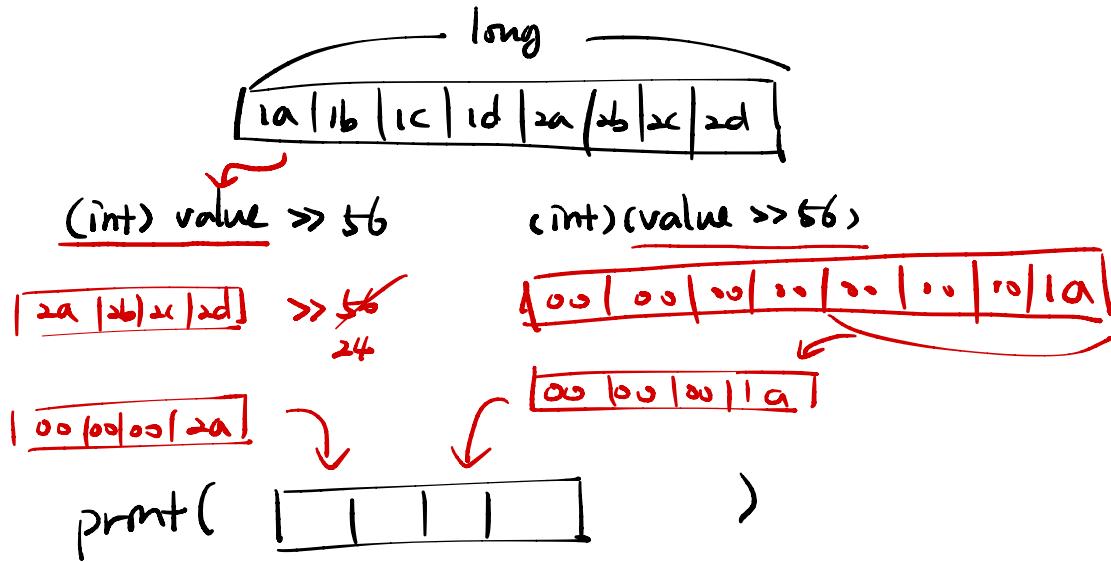
byte[] bytes = new byte[]:

→ read(bytes);

* ByteArrayInputStream



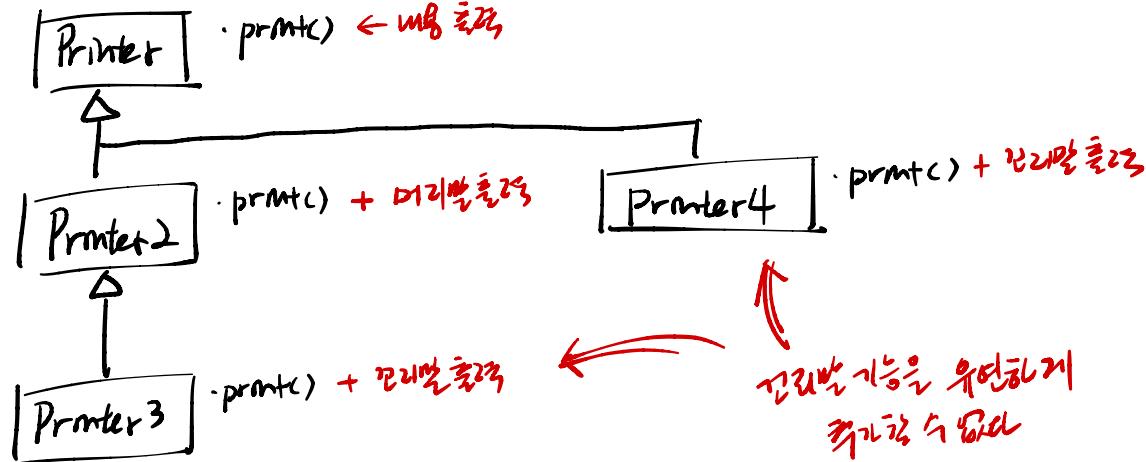
* 18|20|24 26 28 32 36 38 40 44 48 52 56 58 60 64 68 72 76 80 84 88 92 96 100 104 108 112 116 120 124 128 132 136 140 144 148 152 156 160 164 168 172 176 180 184 188 192 196 198 200 204 208 212 216 220 224 228 232 236 240 244 248 252 256 260 264 268 272 276 280 284 288 292 296 298 300 304 308 312 316 320 324 328 332 336 340 344 348 352 356 360 364 368 372 376 380 384 388 392 396 398 400 404 408 412 416 420 424 428 432 436 440 444 448 452 456 460 464 468 472 476 480 484 488 492 496 498 500 504 508 512 516 520 524 528 532 536 540 544 548 552 556 560 564 568 572 576 580 584 588 592 596 598 600 604 608 612 616 620 624 628 632 636 640 644 648 652 656 660 664 668 672 676 680 684 688 692 696 698 700 704 708 712 716 720 724 728 732 736 740 744 748 752 756 760 764 768 772 776 780 784 788 792 796 798 800 804 808 812 816 820 824 828 832 836 840 844 848 852 856 860 864 868 872 876 880 884 888 892 896 898 900 904 908 912 916 920 924 928 932 936 940 944 948 952 956 960 964 968 972 976 980 984 988 992 996 998 1000



29. File I/O API : Decorator 클래스 활용하기

기능 확장

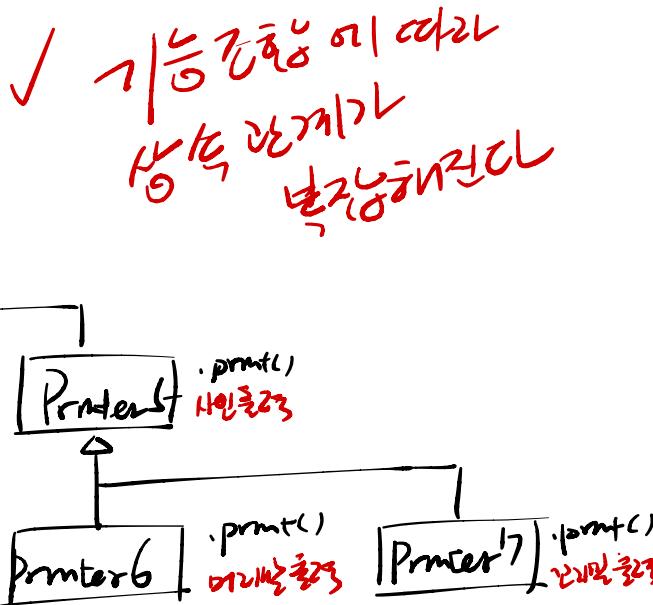
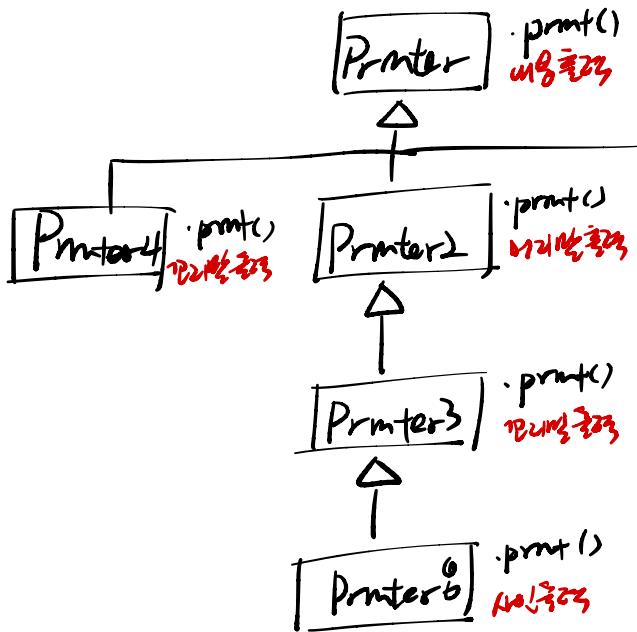
① 상속



* 단점
· 기능 추가 ↓
 ↳ 상속부분은
 ↳ 일부 기능 처리 불가

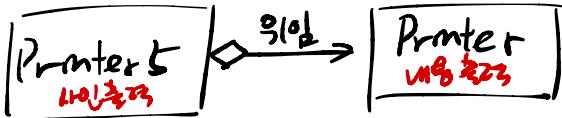
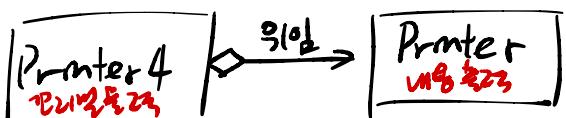
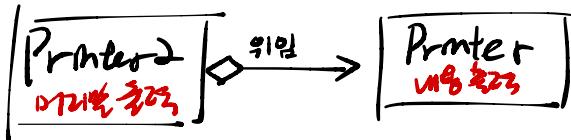
기존 기능을
우선화하지
추가할 수 없음

* 삼수를 이용한 기능 확장 시 복제 상황



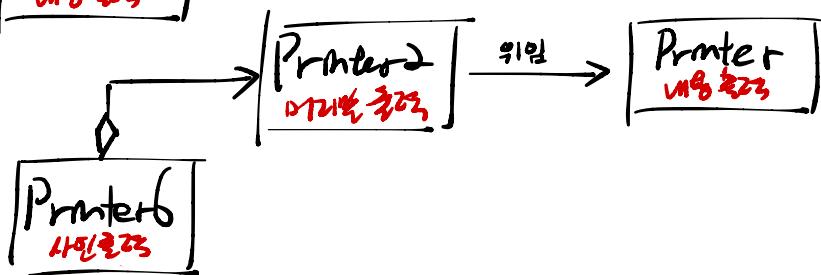
✓ 기능 확장 시 캐스팅
기능 구현하기
복제 가능

② 프린터 간의 이동한 기능 확장

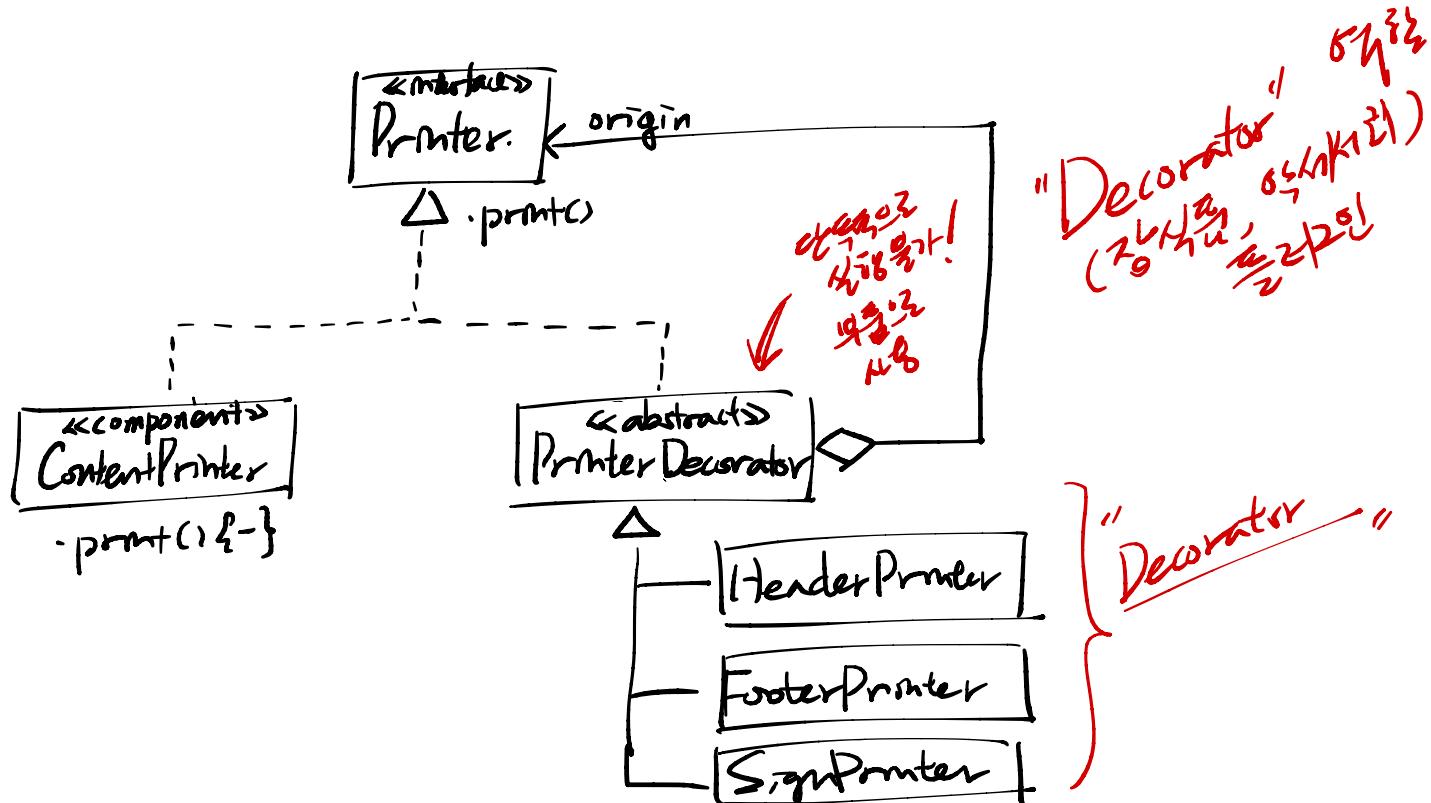


* BANRI
- 상부기관 ~~관련~~ 관계 부설기관
(기능 분할 관리)

- 기능을 관리하는 기관
주관기관이 관리를 해야 한다.



- ③ GOF의 Decorator 패턴은 기능을 확장하는 패턴
- ↳ 여러 기능은 상황에 따라 결합되는 경우에 유용
 - ↳ 기능은 풀고자 하거나 조합하기 힘들 때



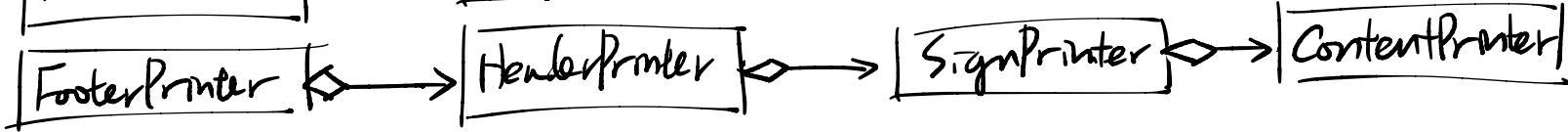
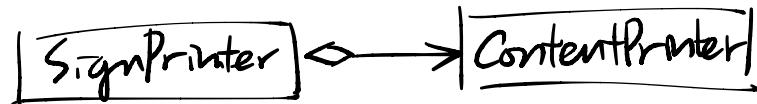
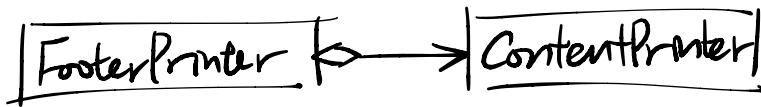
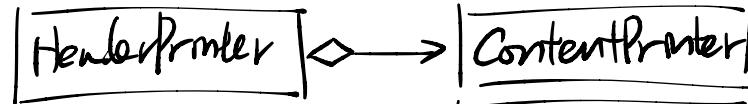
* Printer 2단

ContentPrinter

HeaderPrinter

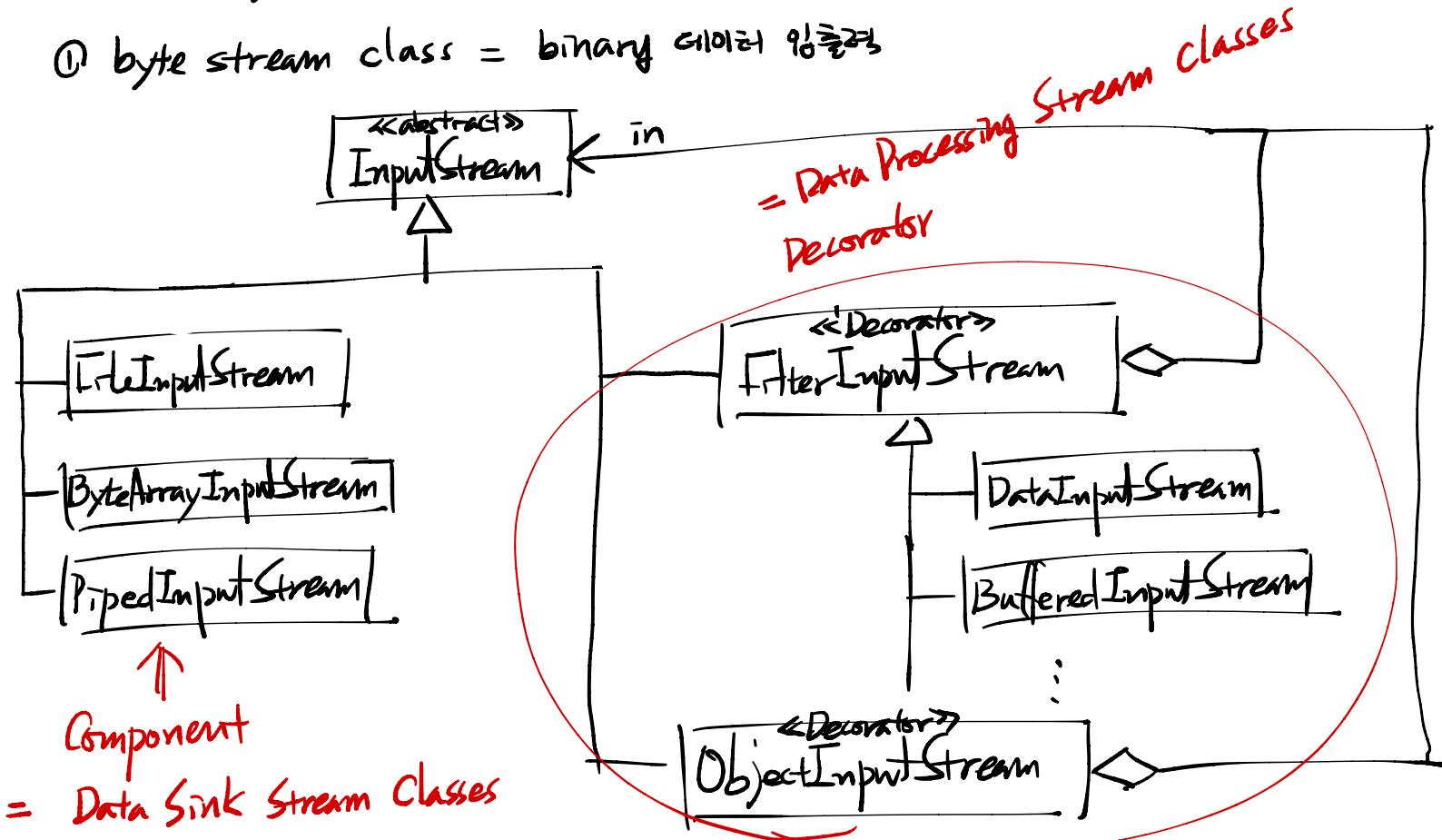
FooterPrinter

SignPrinter



* File I/O API 와 Decorator 패턴

① byte stream class = binary 데이터 처리 클래스



* DataInputStream / DataOutputStream



`int` → `byte[]`



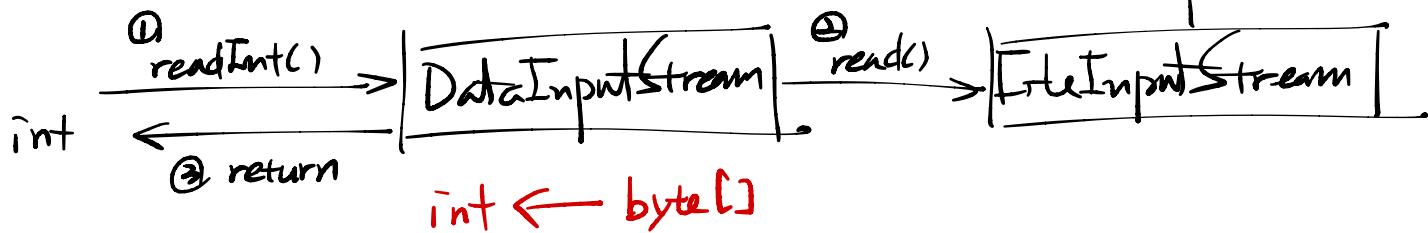
"Decorator" = "Data Processing Stream class"



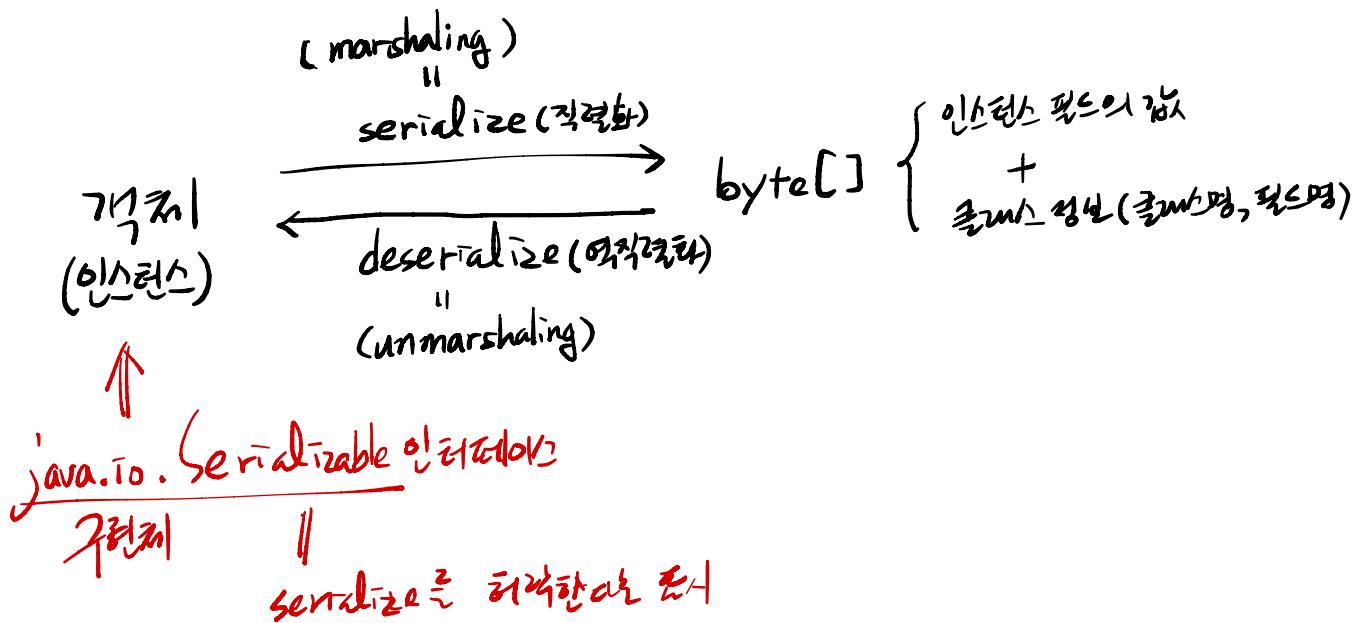
③ ↗ ↘



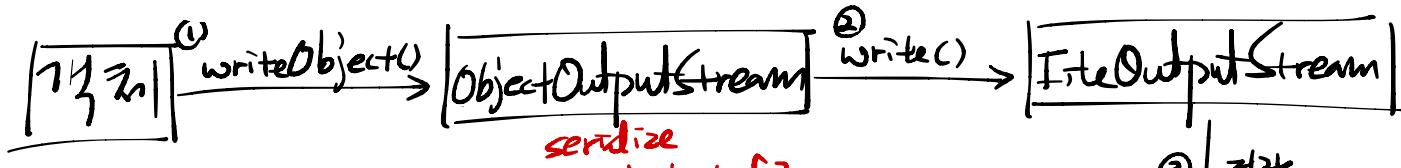
↗ ↘



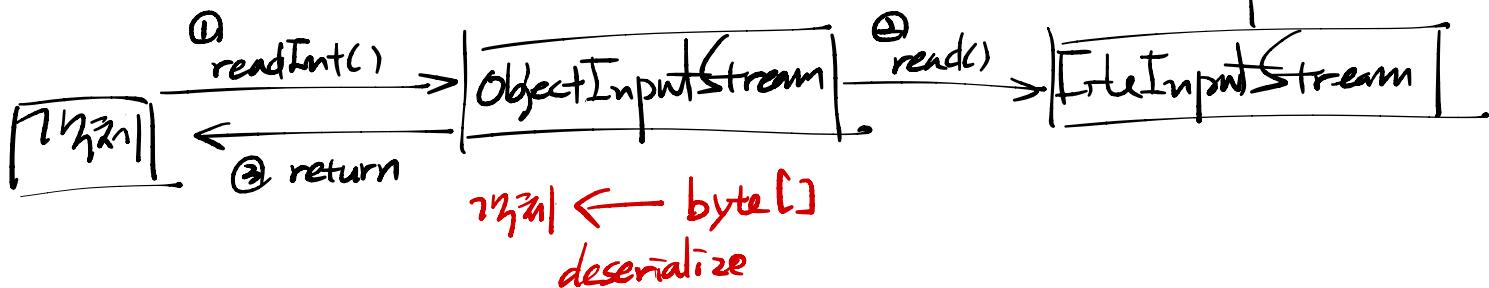
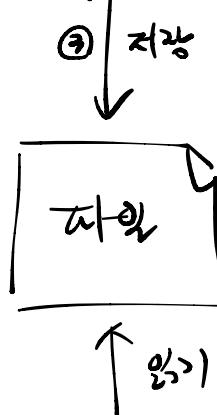
30. File I/O API : 객체 직렬화 / 역직렬화



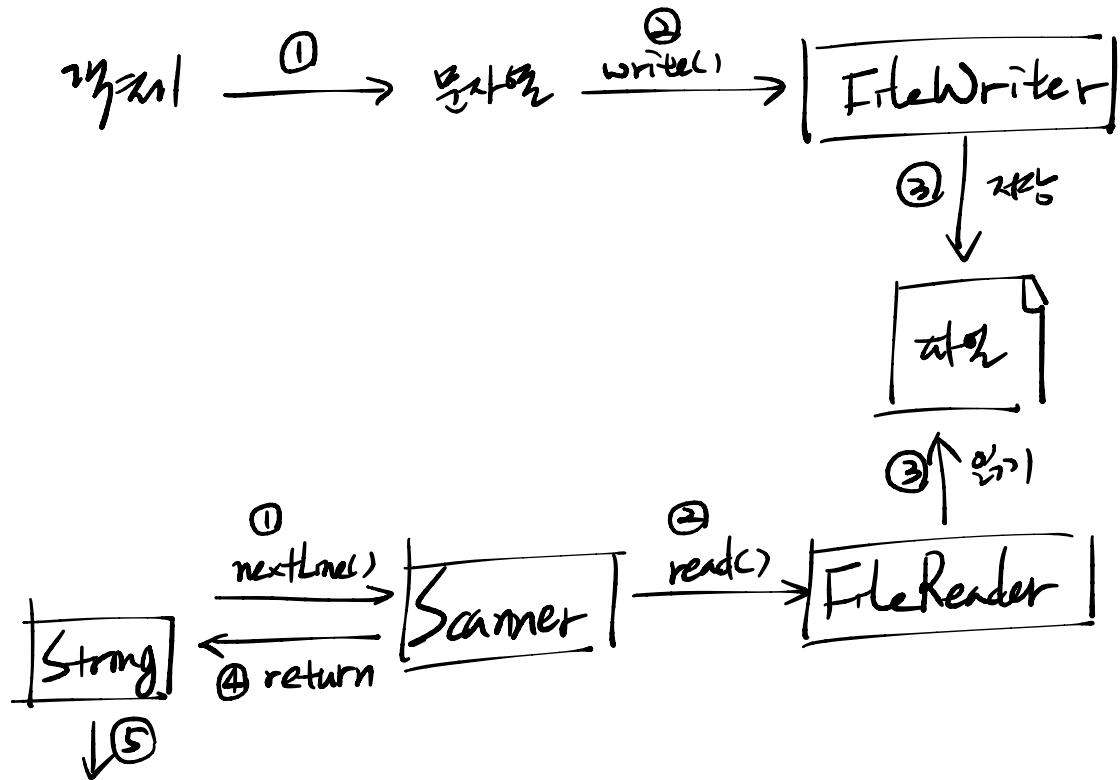
* ObjectInputStream / ObjectOutputStream



"Decorator" = "Data Processing Stream class"

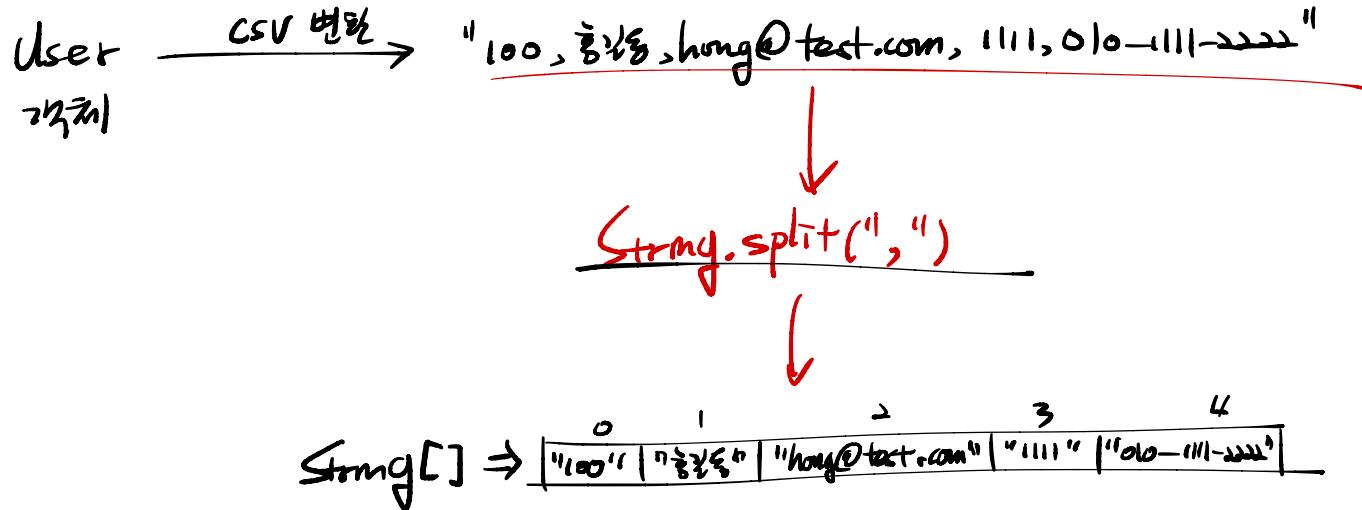


31. File I/O API : 파일을 읽어들이는 CSV 입출력



답변

* CSV 例外의 String.split() 딜리전



* 32. File I/O API : JSON 토막 입출력

JavaScript Object Notation 문법을 가져다서 막대 토막

	JavaScript 토막	JSON 토막
문자열	"문자열" '문자열'	"문자열"
프로퍼티명	프로퍼티명 "프로퍼티명" '프로퍼티명'	"프로퍼티명"

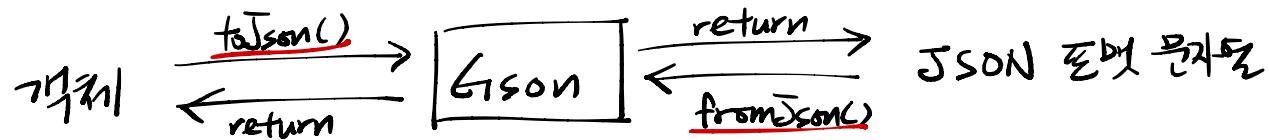
a) 객체

```
{
  "name": "홍길동",
  "age": 20,
  "working": true,
  "address": {
    "postNo": "01100",
    "city": "seoul"
  }
}
```

b) 목록

```
[ "aaa", "bbb", 20, true ]
[ { - }, { - }, { - } ]
```

* JSON 스펙(<http://json.org>)



* Gson 구조

① 기본 설정으로 생성

`new Gson()`



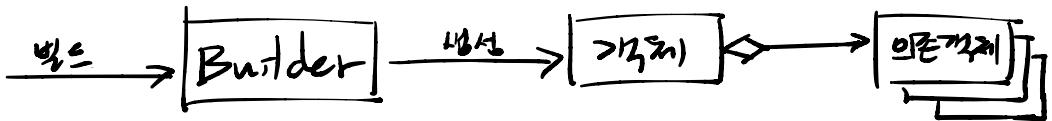
`Gson` 구조

② 빌더 패턴 구조

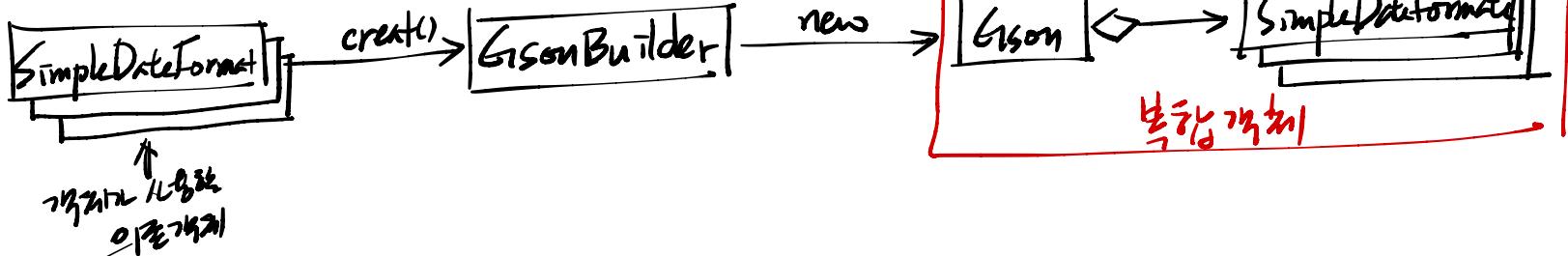


* Gson은 구조화 생성과 GOF의 Builder 패턴
 ↳ 생성자 외부 → 전문 (기본 + 창문 + 인터페이스 + 외부파인드 + 망장)
 + 응선

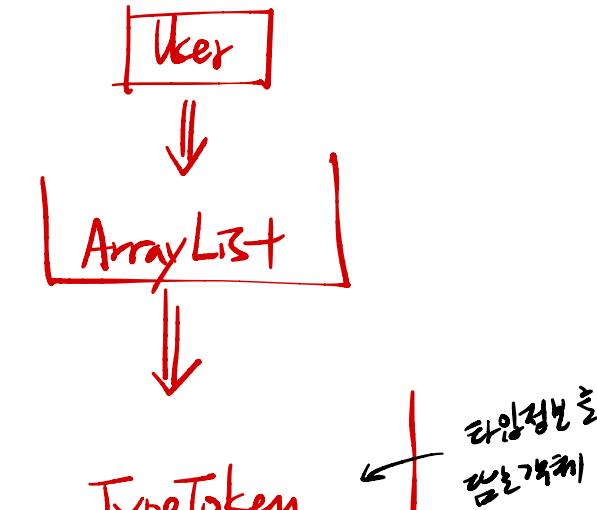
전통적이라 전문을 찾는 것처럼
 여러가지의 의존객체(응선)가 복합적으로 조합된 객체를
 만들었을 때 사용하는 패턴이다



a) GsonBuilder



* JSON 목록 데이터 $\xrightarrow{\text{알기}}$ 클래스형 객체



"[{}, {}, ...]"

new Gson().fromJson(Json문자열, 클래스형 타입 정보);
 Strong TypeToken

답변 1 :

new TypeToken<ArrayList<User>>() {}

* JSON 목록 데이터 $\xrightarrow{\text{읽기}}$ 커리어전기부제 II

`new Gson().fromJson(Json문자열, 현재년 타입 정보);`

Ques 2 : TypeToken.getParameterized(List.class, User.class)

* 여러 사용자를 위한 규칙 정의

