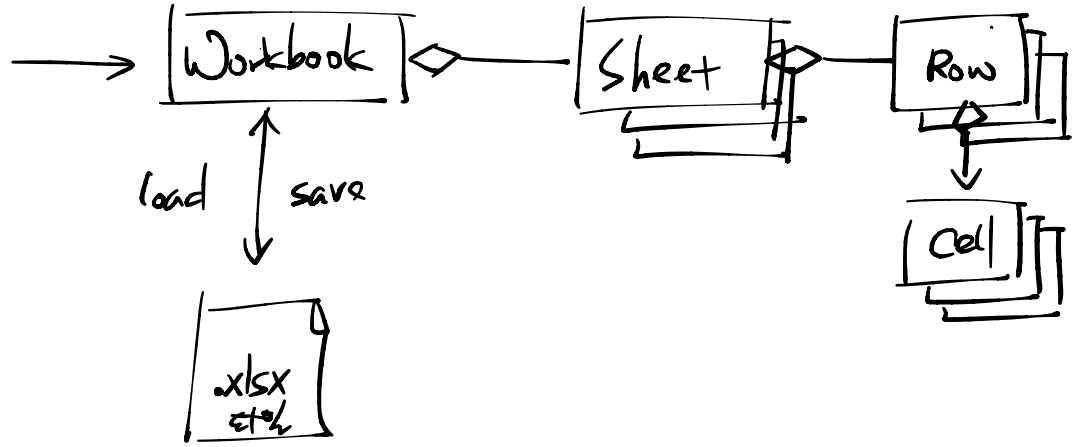
33. 데이터를 엑셀 포맷으로 쓰고/읽기 : Apache POI 활용

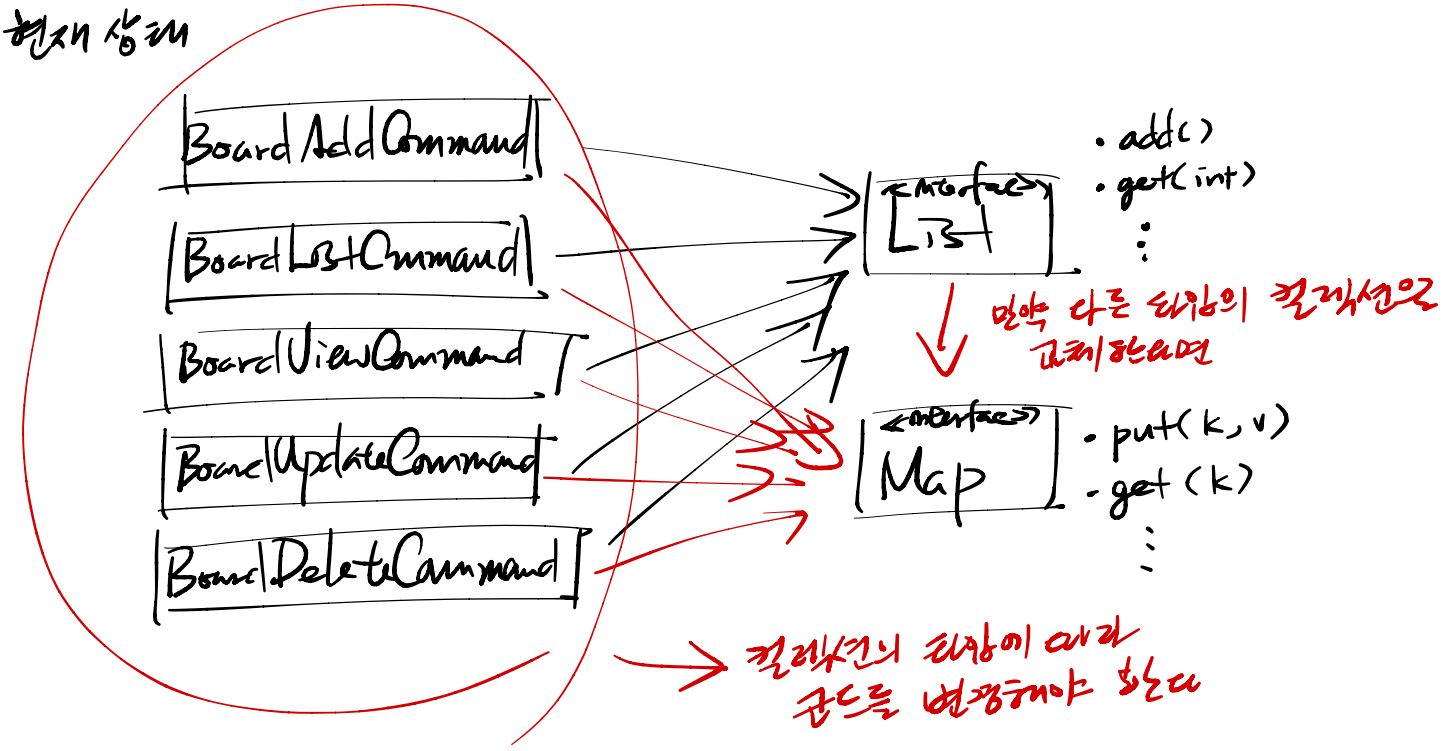
HSSF (엑셀 97)
.xls
XSSF (엑셀 2007)
.xlsx



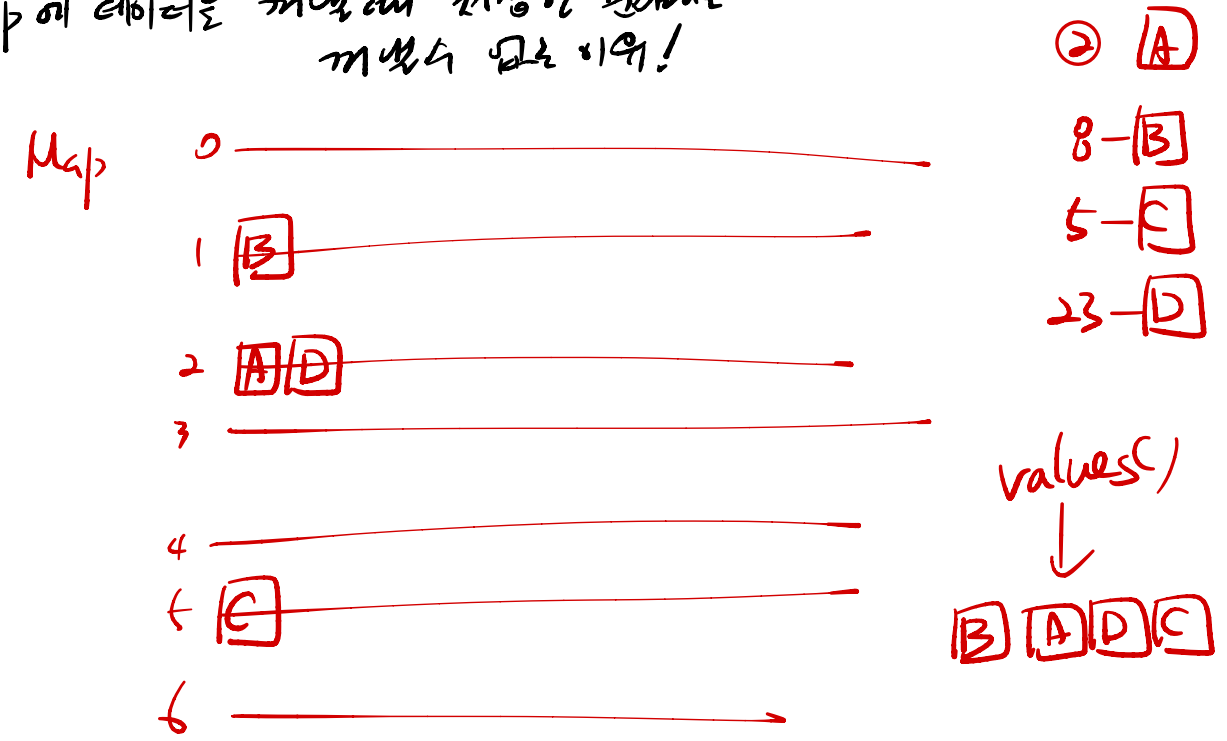
34. DAO가 필요한 이유

↳ 데이터 보관을 캡슐화

① 현재 상태



* Map에 데이터를 저장할 때 저장할 위치를 계산하는 방식 필요 없음!

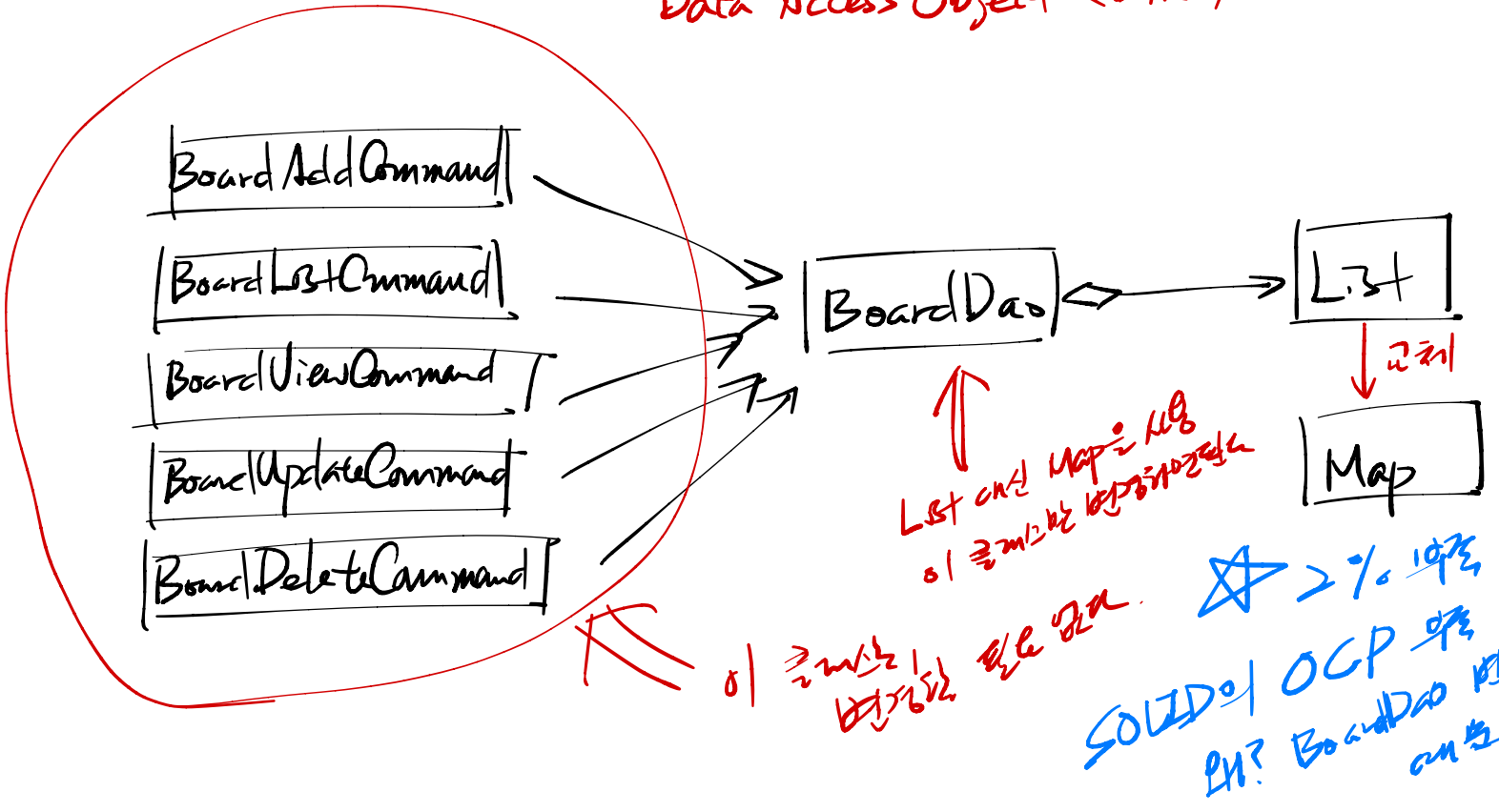


저장할 때 key의 Hash값을 기준으로 저장할 위치를 계산하기 때문에!

35. 데이터 처리 로직을 캡슐화 하기

→ 별도의 클래스를 만들어 분리 → 재사용성이 높아진다.

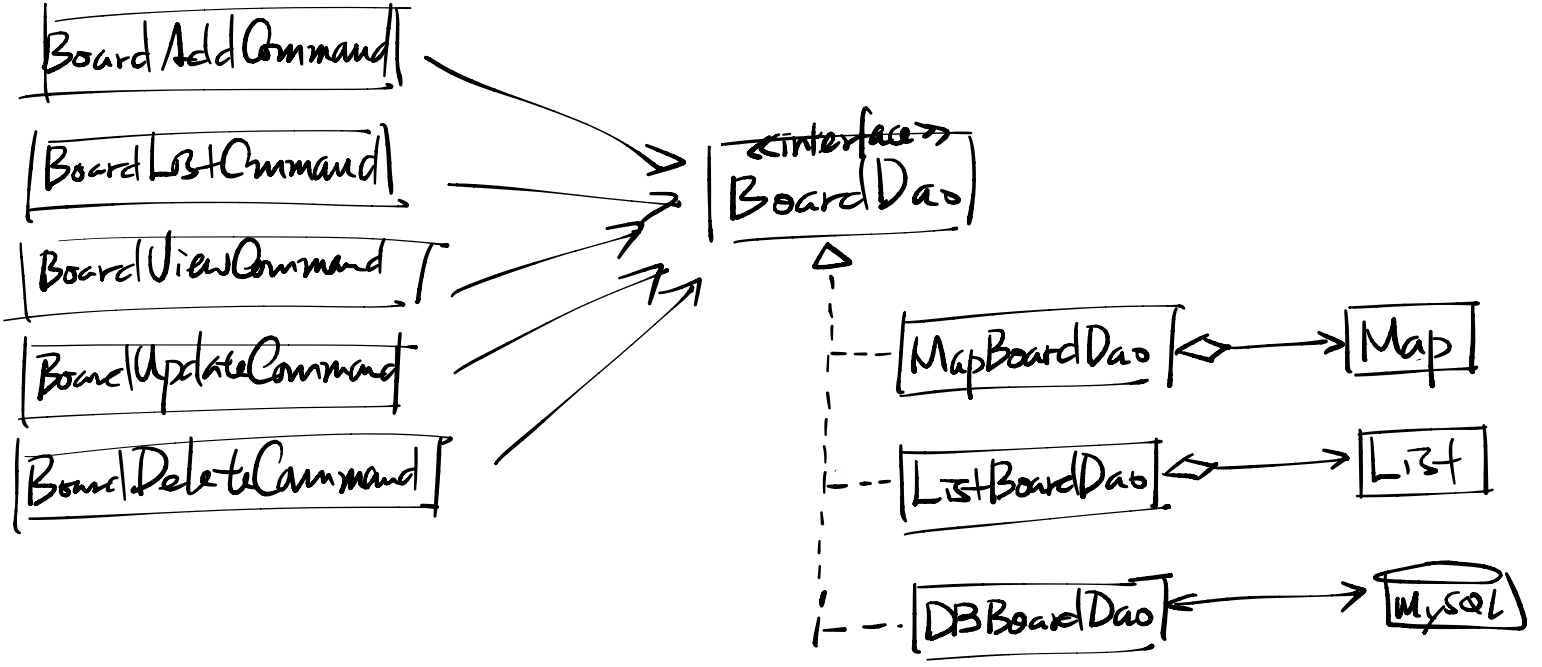
Data Access Object (DAO)



35. 데이터 처리 로직을 캡슐화 하기

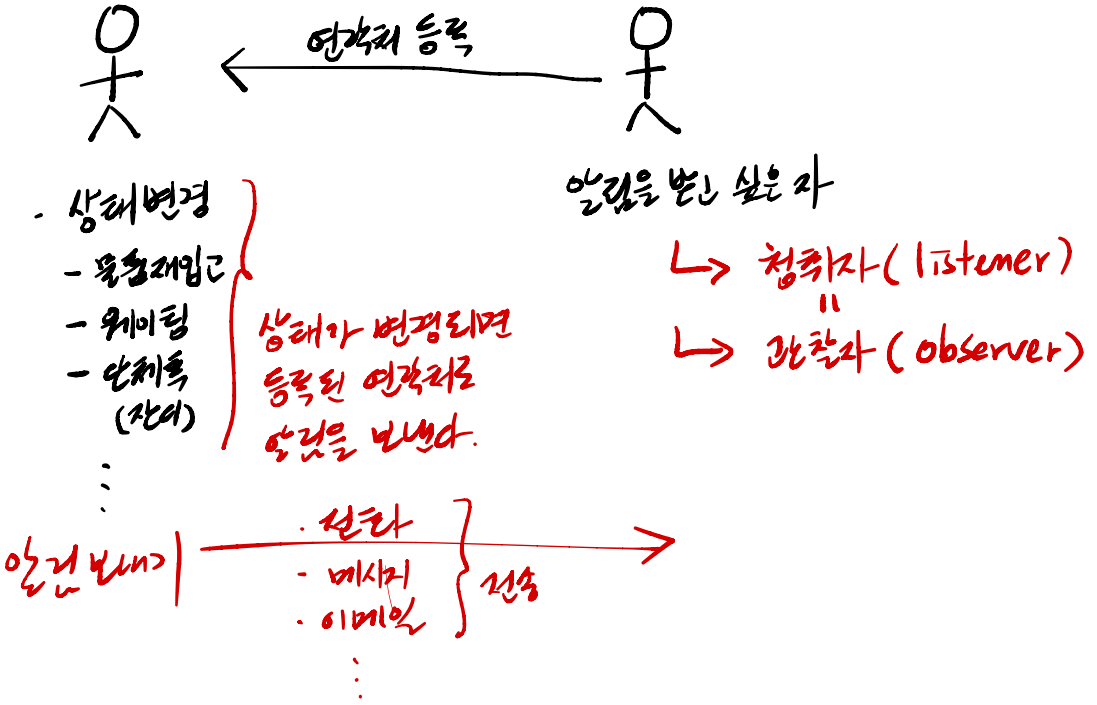
데이터베이스를 처리하는 클래스를 만들어 보자 → 객체지향에서 위임이라고 한다.

Data Access Object (DAO)

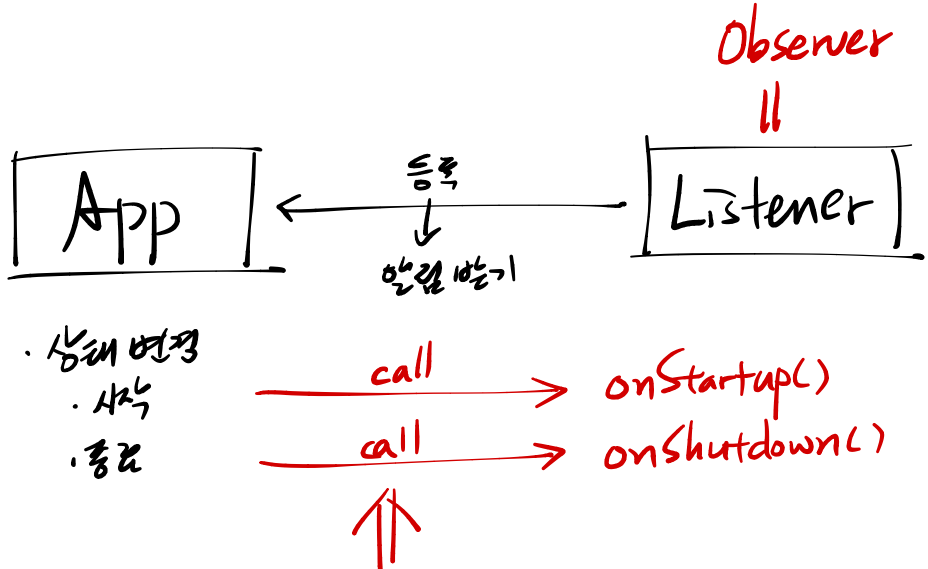


* 36. 알림 받기 : GoF의 Observer 패턴

실생활에)

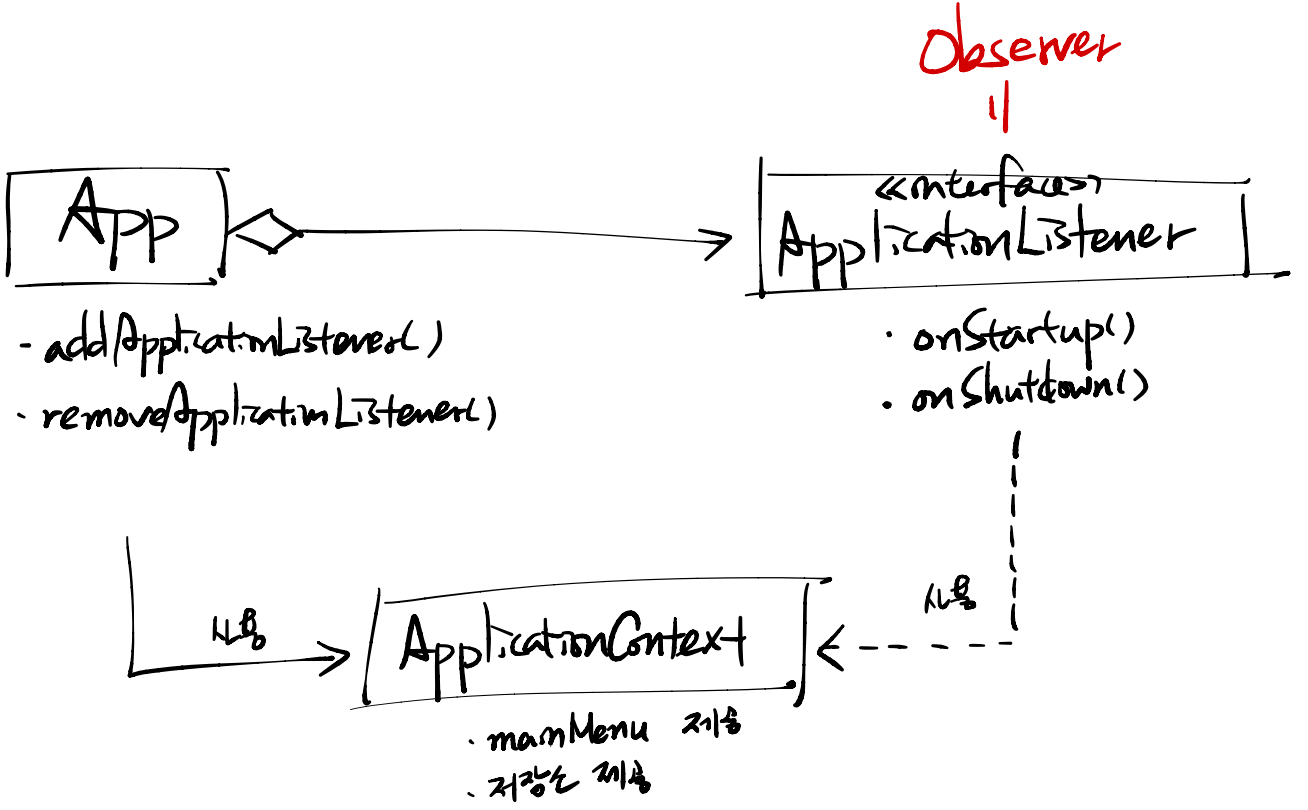


코드 예제)



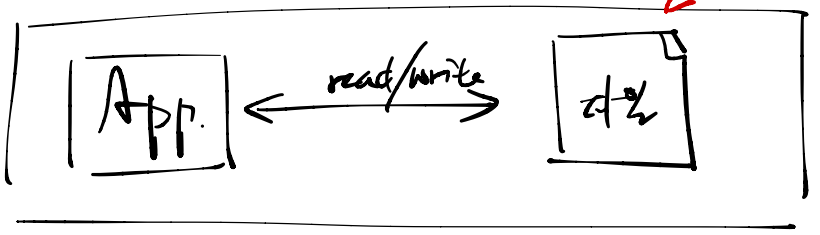
프로그래밍 세계에서 다른 객체에 알림을 주는 방법은 메서드를 호출하는 것이다.

* GoF의 Observer 패턴



31. Application 간의 데이터 공유

현재 상태

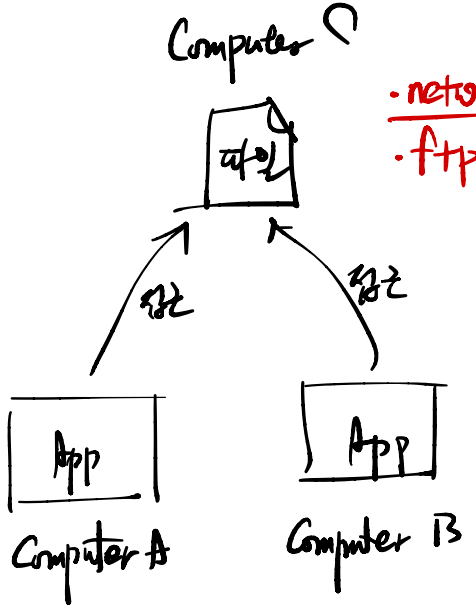


각 App. 이 개별적으로 데이터 관리

App. 끼리 데이터 공유 불가!

31. Application 간의 데이터 공유

① 직접적인 자원 공유



- network drive
- ftp 서버

☆ 문제점

- 동시에 여러 App. 자원에 접근할 수 있다

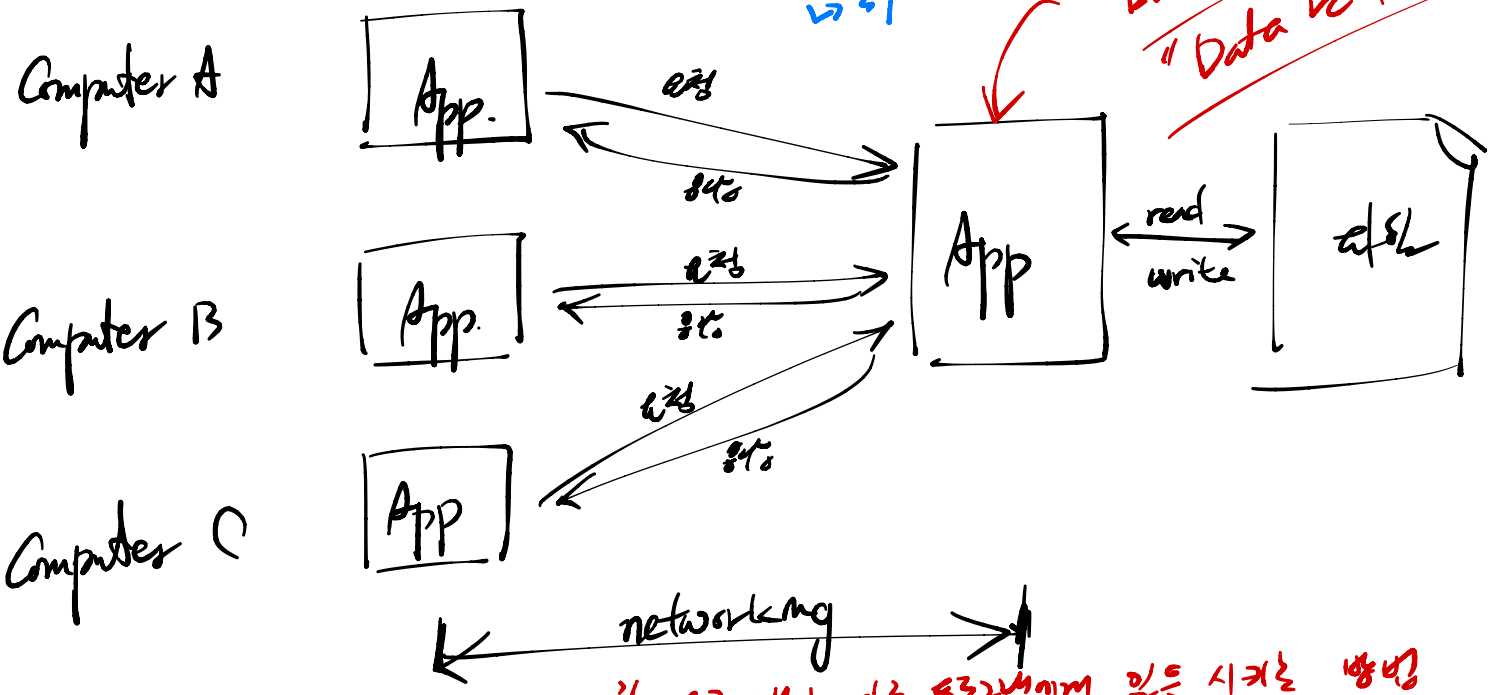
↓

파일을 덮어쓸 수 있다.

31. Application 간의 데이터 통신

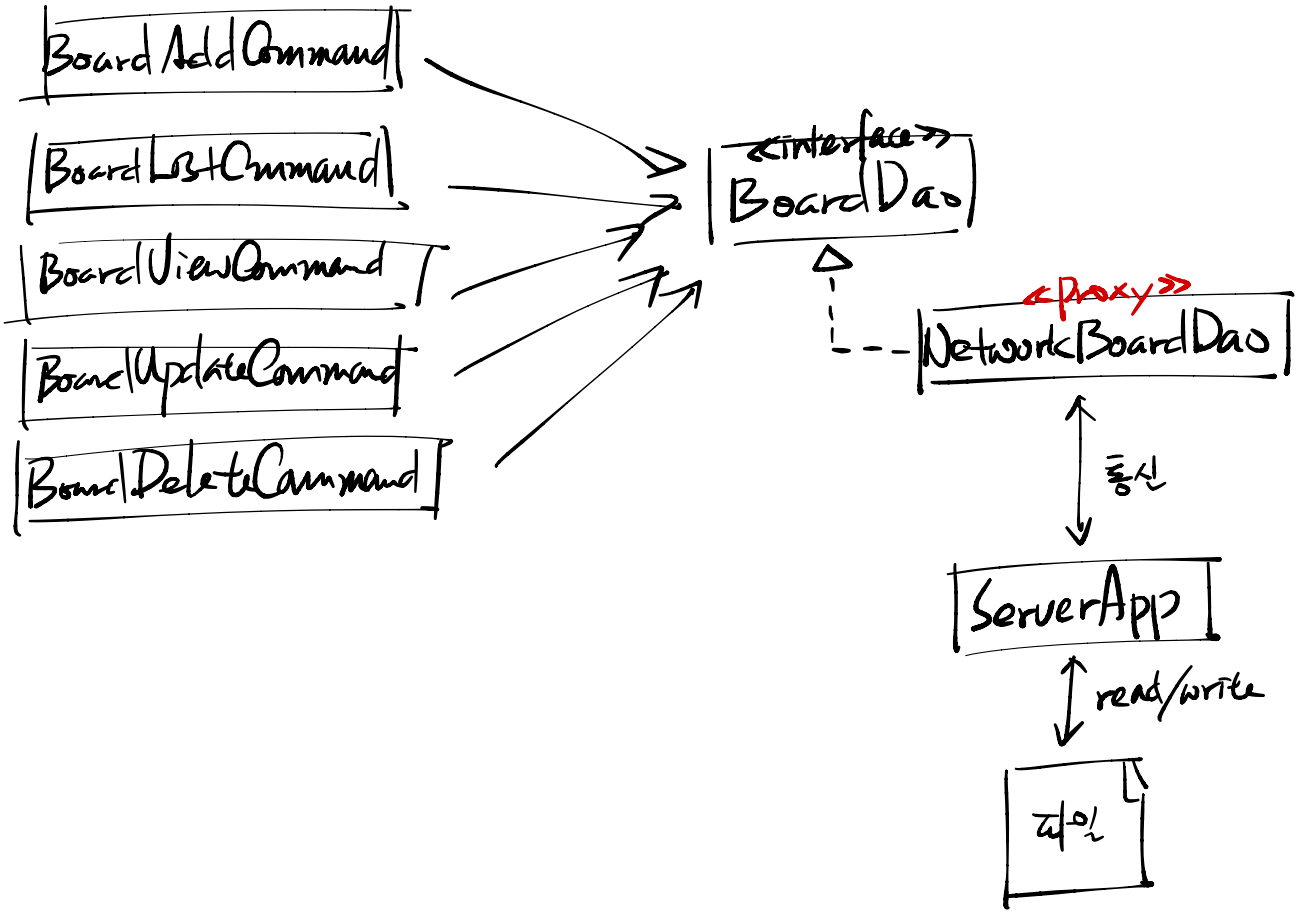
① 일반적인 데이터 통신

- networking prog.
- multi-tasking prog.
  - ↳ multi-threading
  - ↳ synchronous



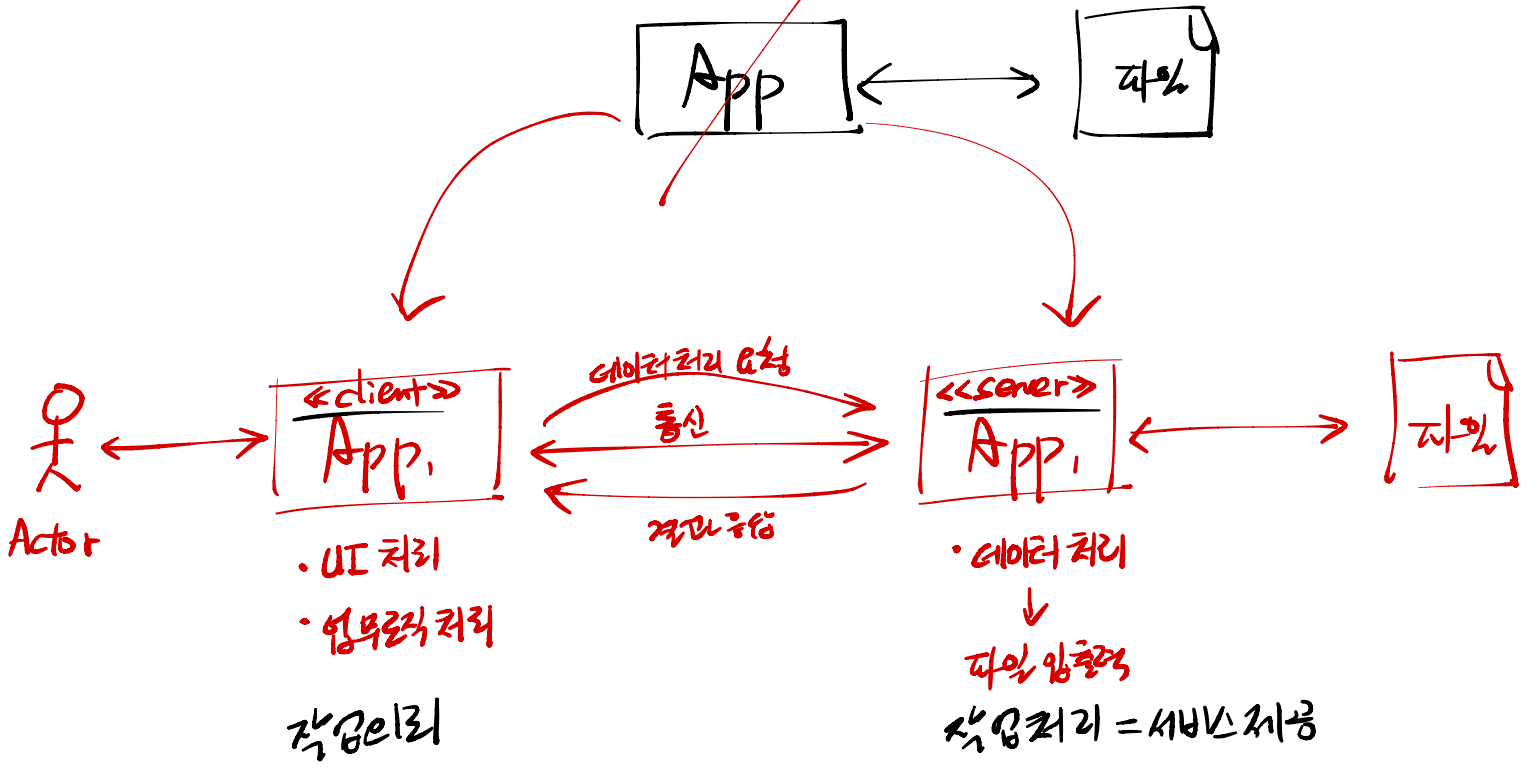
한 프로그램이 (process) 다른 프로그램에게 일을 시키는 방법

(process) ← 실행 중인 프로그램

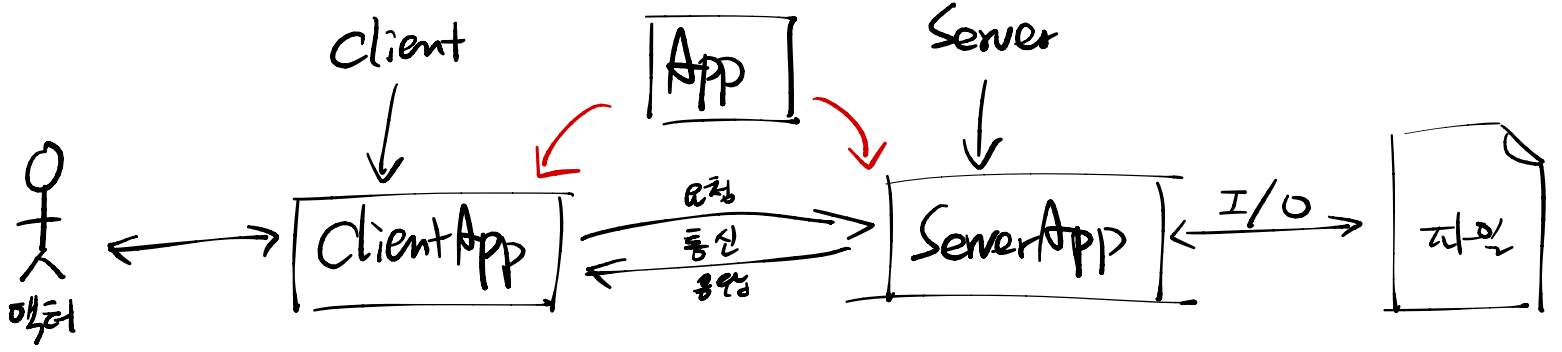

BoardAddCommand
BoardListCommand
BoardViewCommand
BoardUpdateCommand
BoardDeleteCommand
<interface>
BoardDao
<proxy>
NetworkBoardDao
통신
ServerApp
read/write
data

* Software Architecture : Client/Server Architecture
(System)

⇒ CS Application



* CS Architecture 구현

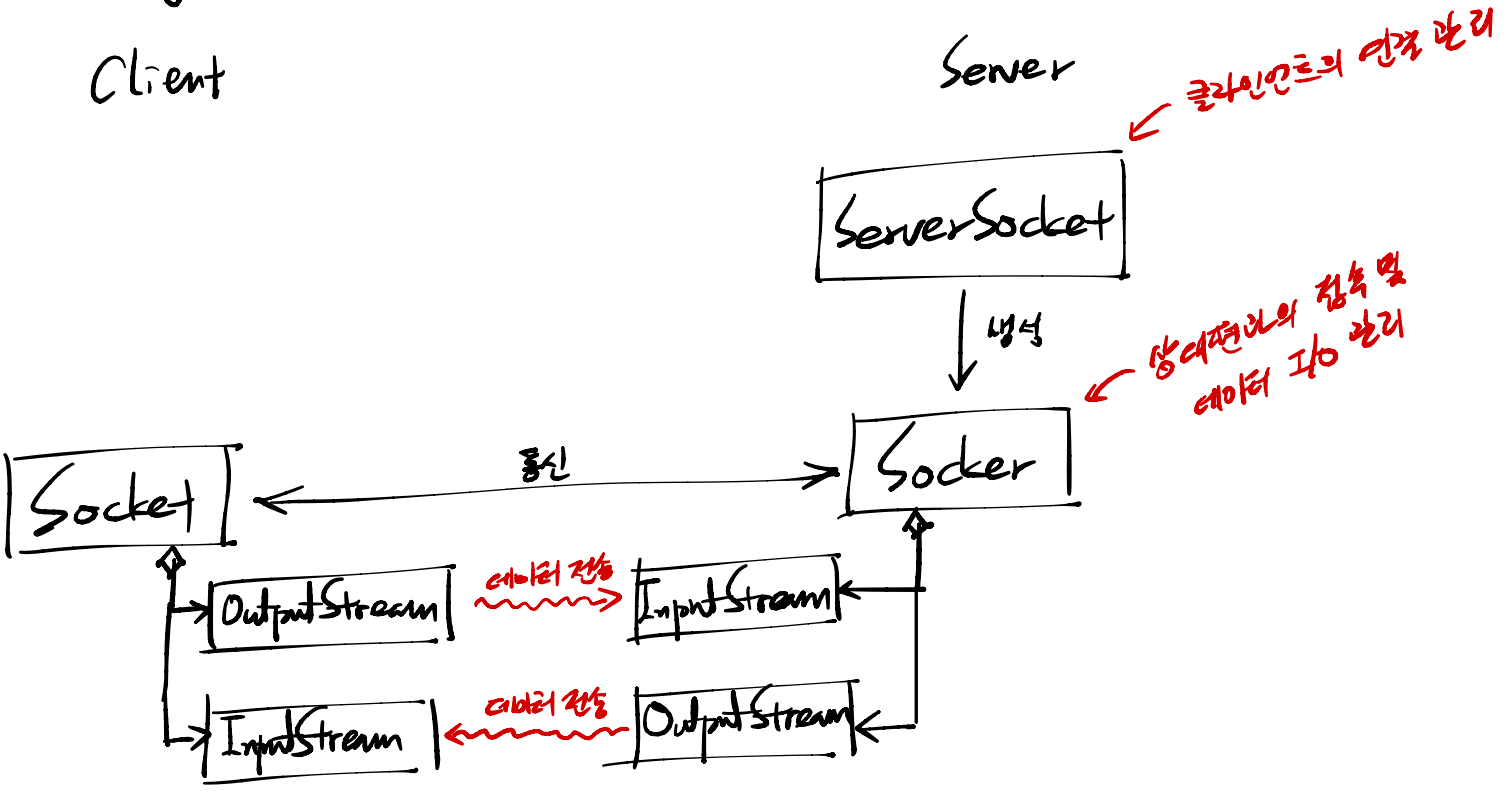


myapp/

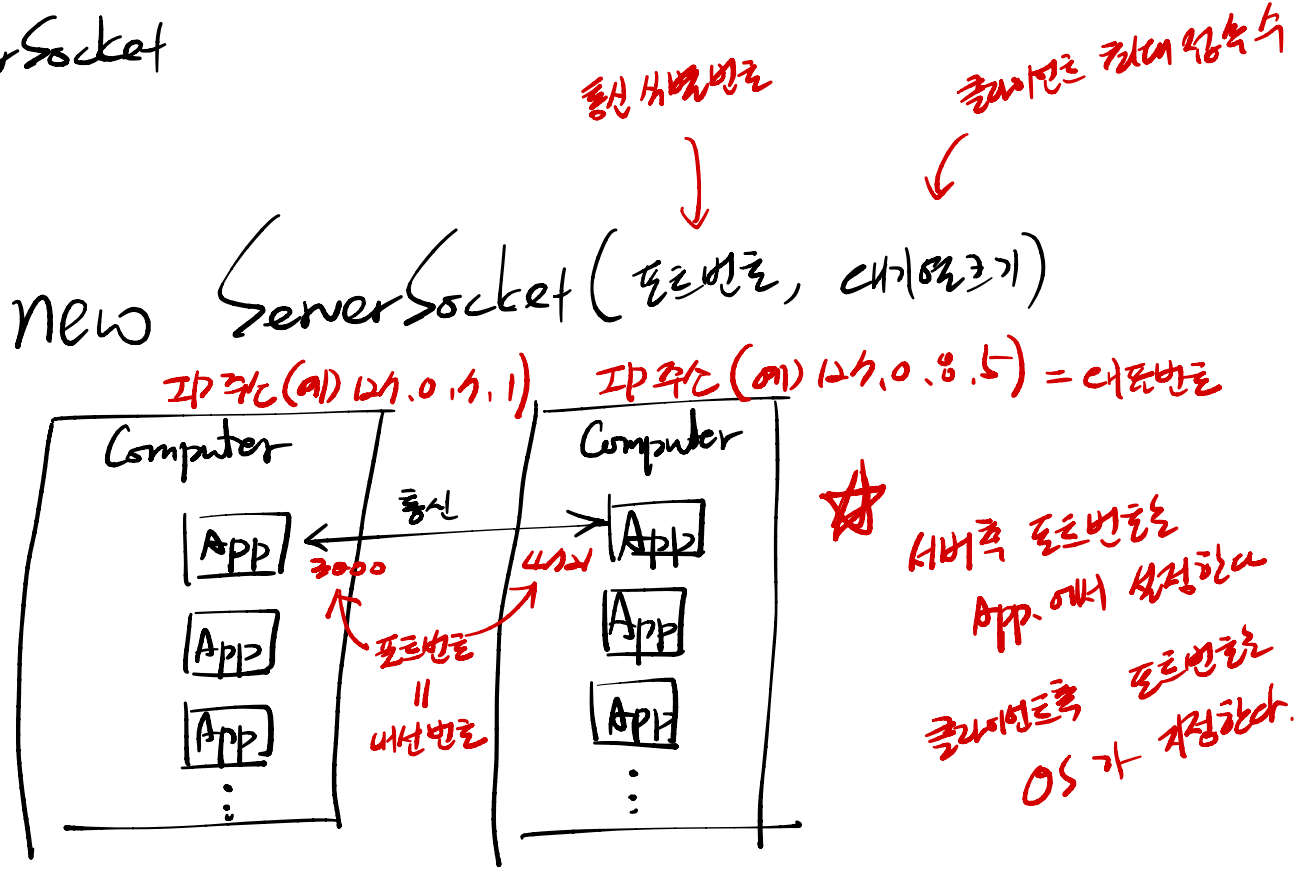
- ~~app/~~
- app-client/
- app-server/

} 프로젝트 디렉토리

* Networking API 사용법



* ServerSocket



* Socket

상대편 IP주소 ↓ 상대편 포트번호 ↓

new Socket(IP주소, 포트번호)

* 나의 포트번호? (클라이언트) → OS가 자동으로 할당.

* 특별한 IP주소

Local 주소 = 로컬호스트IP주소

→ 127.0.0.1 = "localhost"

* Protocol : 데이터 송수신 규칙

client

데이터명 - "users"

작업명령 - "insert" | "list" | "get" |
"update" | "delete"

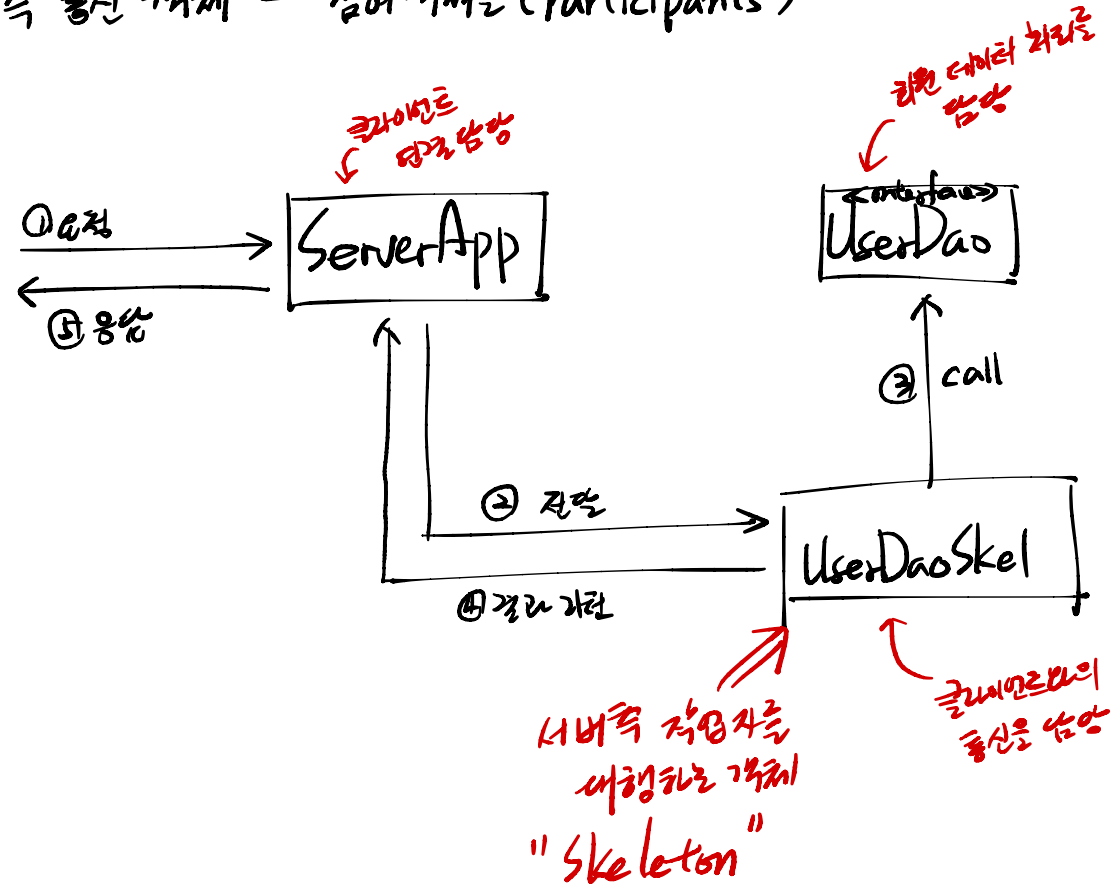
전송데이터 - *

Sever

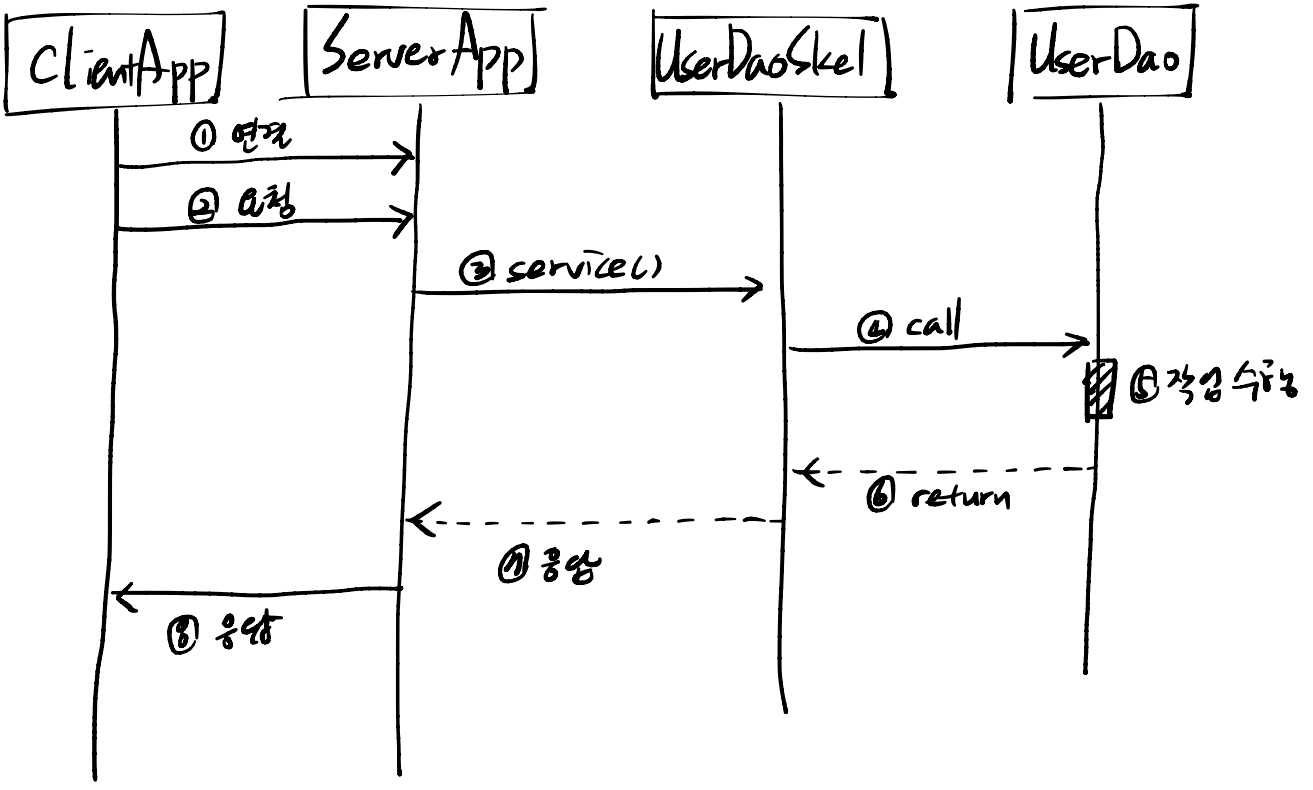
응답상태 - "success" | "failure"

응답데이터 - *

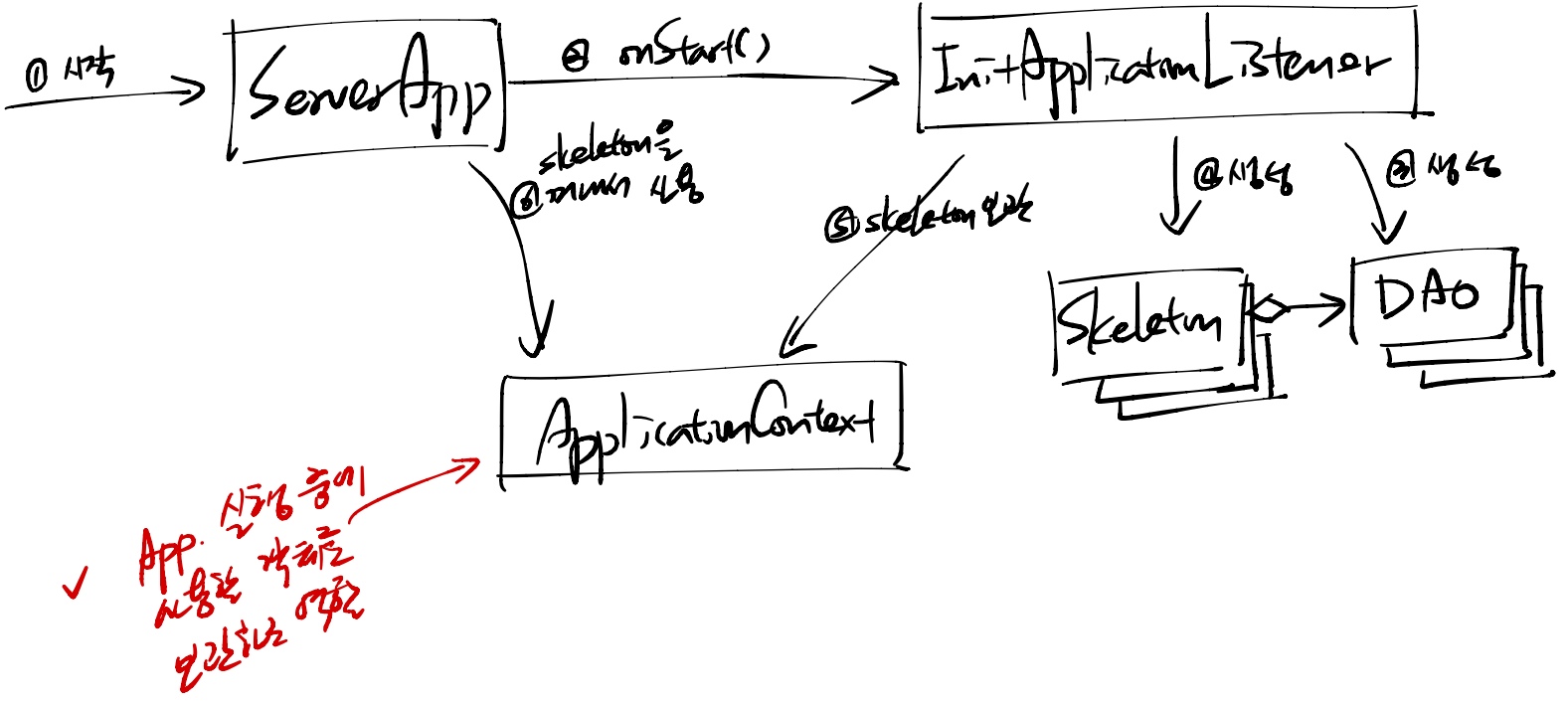
* 서버측 통신 객체 – 참여객체들 (Participants)



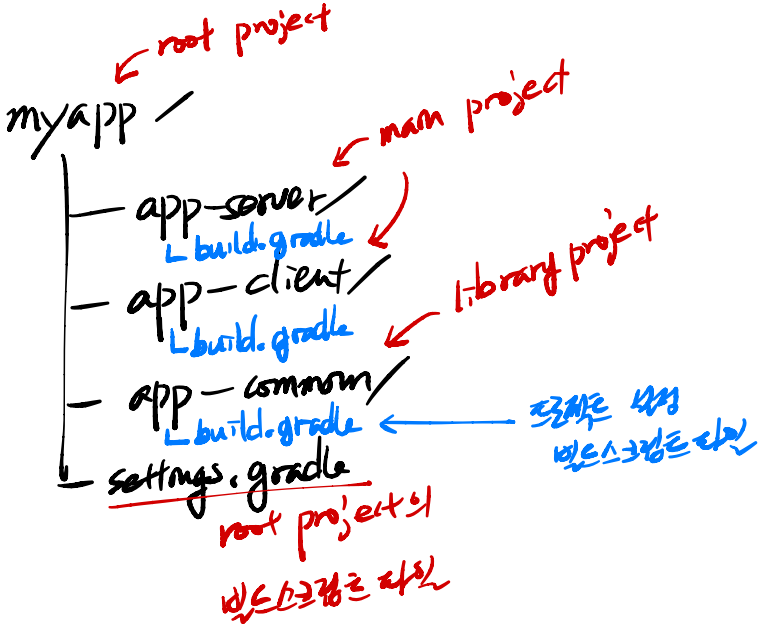
* 서버측 통신객체 - Sequence Diagram (시간에 따라 객체의 상호작용을 표현)

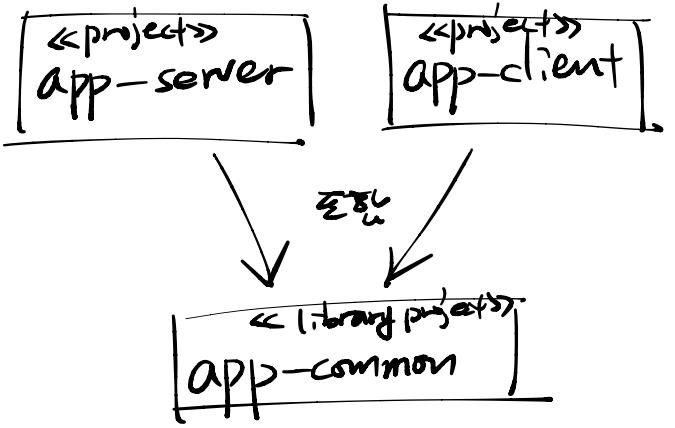


* ServerApp 과 Skeleton



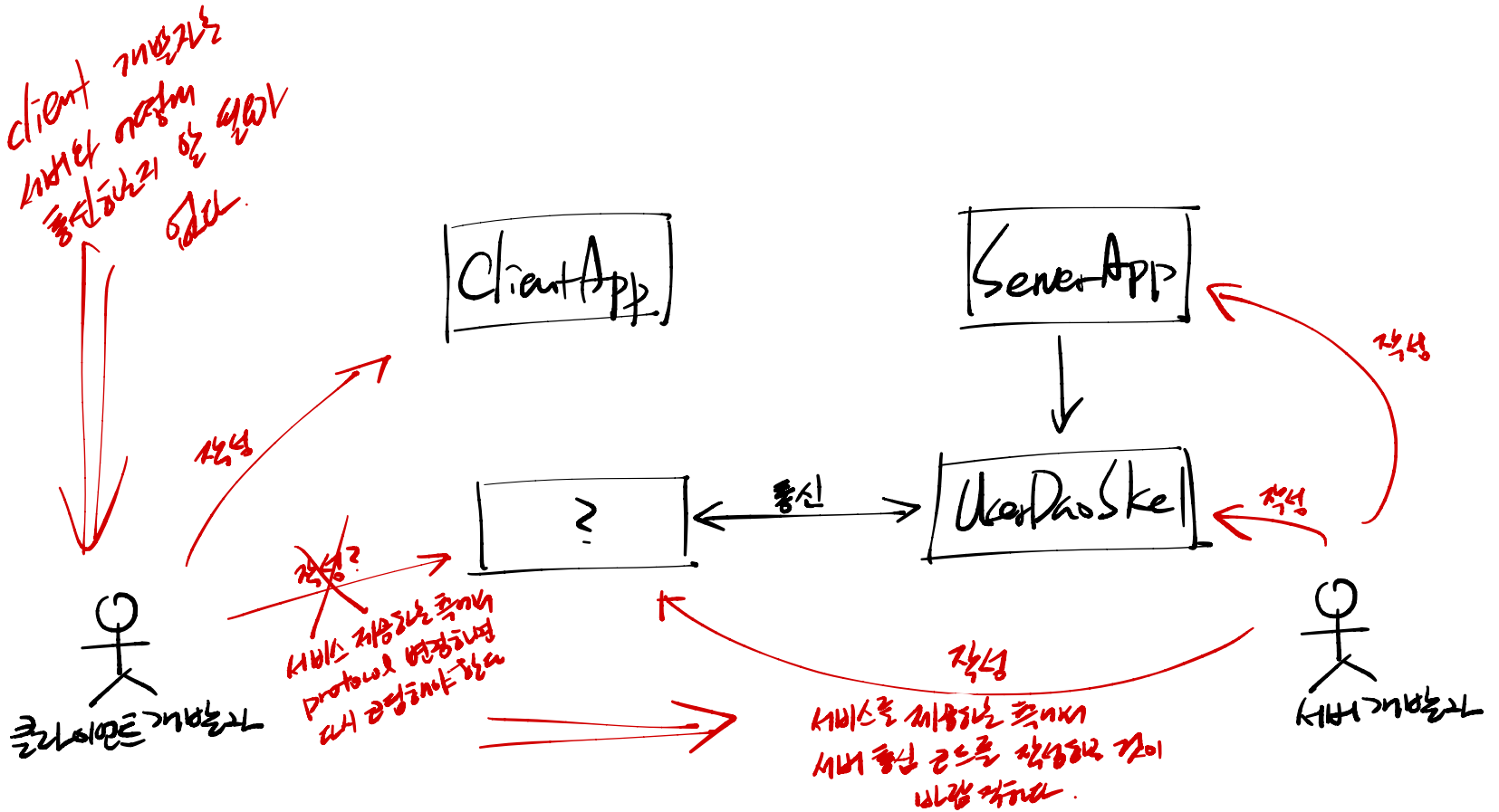
* 라이브러리 프로젝트 - 프로젝트간의 코드 공유하는 방법



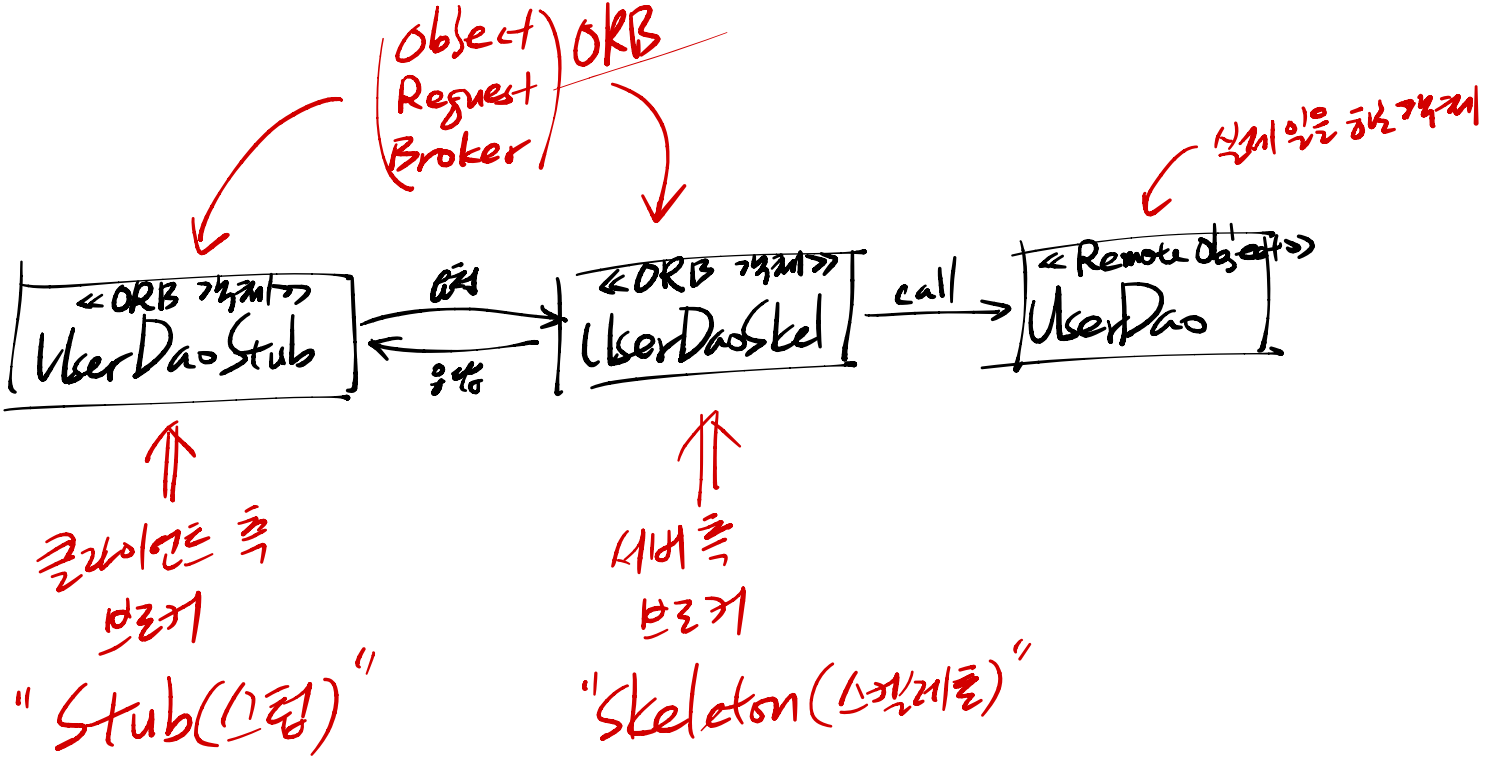


- VO 계층
- Net 계층

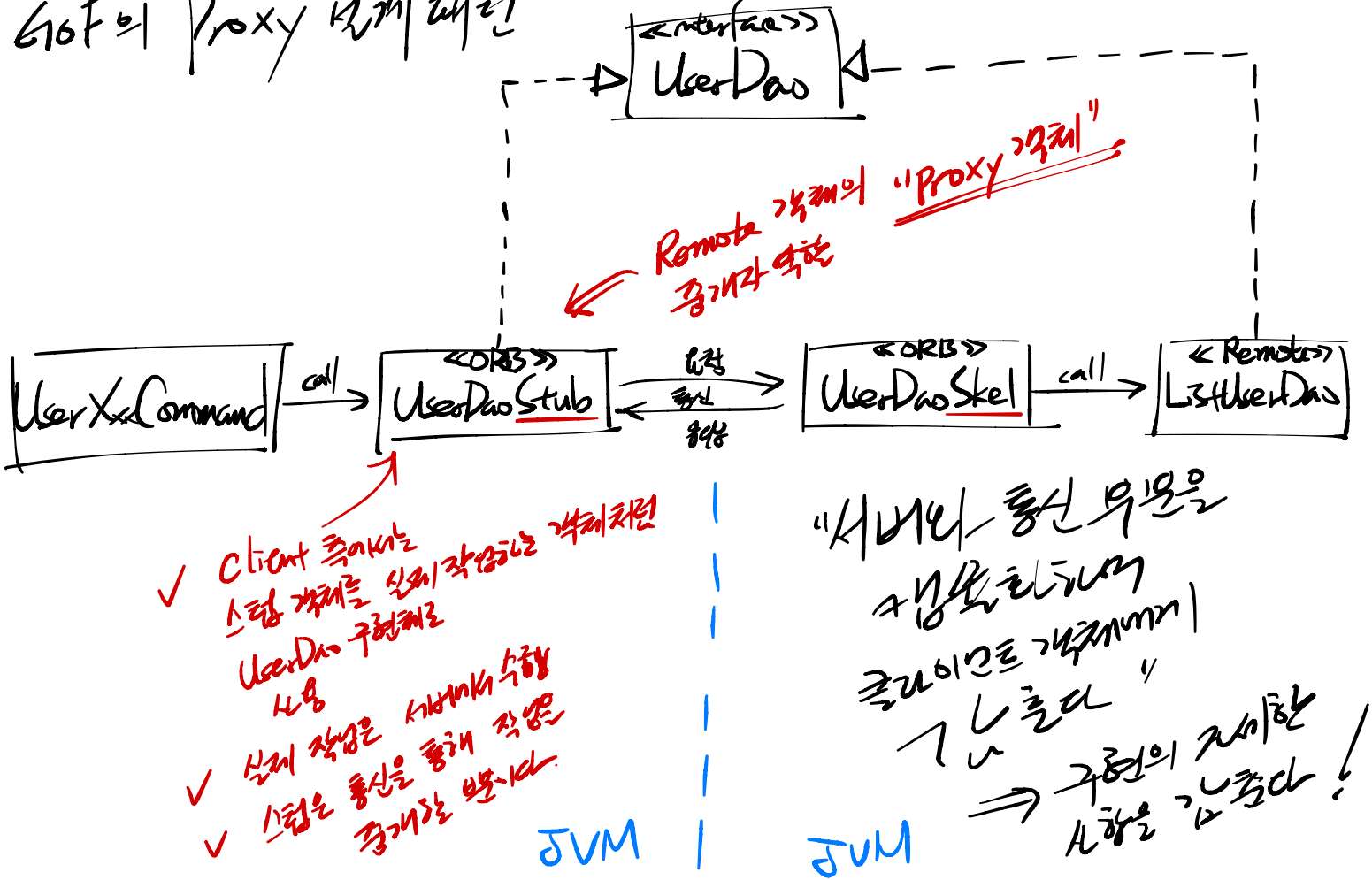
* Server 개발자 / Client 개발자



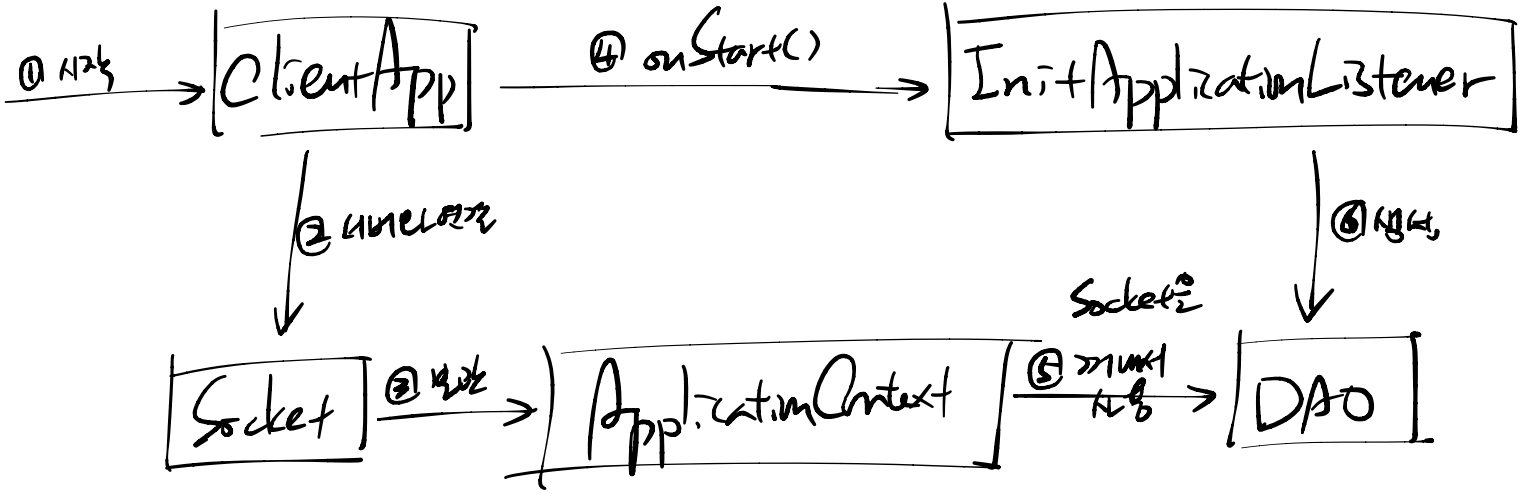
* Skeleton 과 Stub



✱ GoF의 Proxy 디자인 패턴

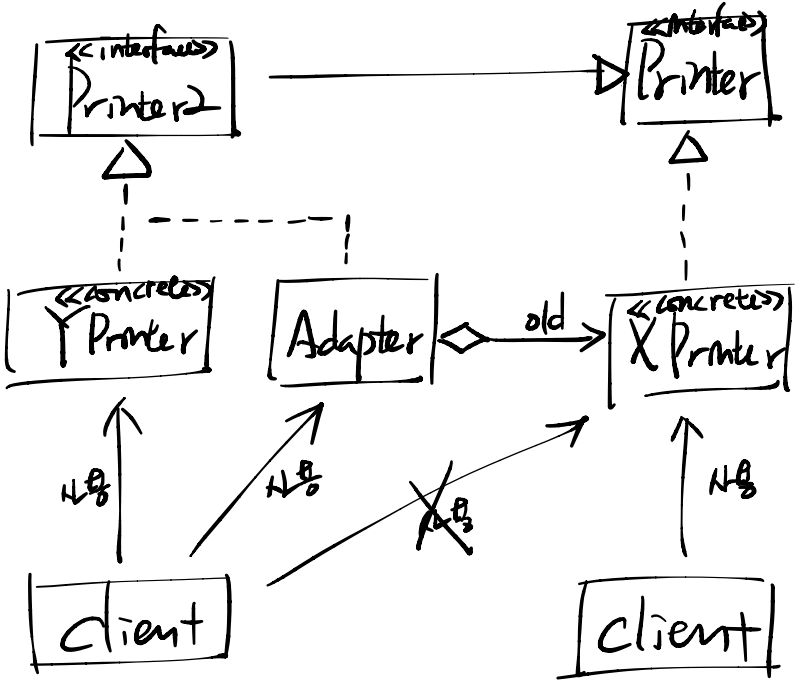


* Client Application 과 Stub

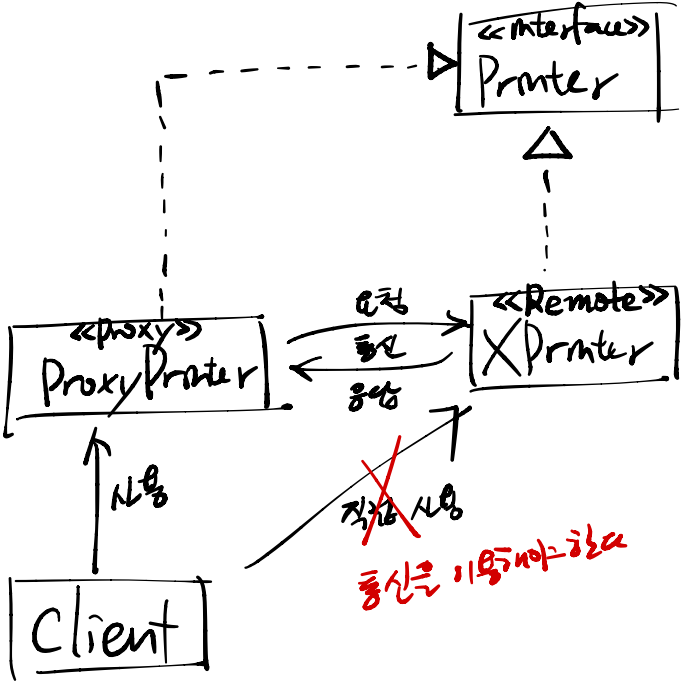


* Adapter 패턴 vs Proxy 패턴

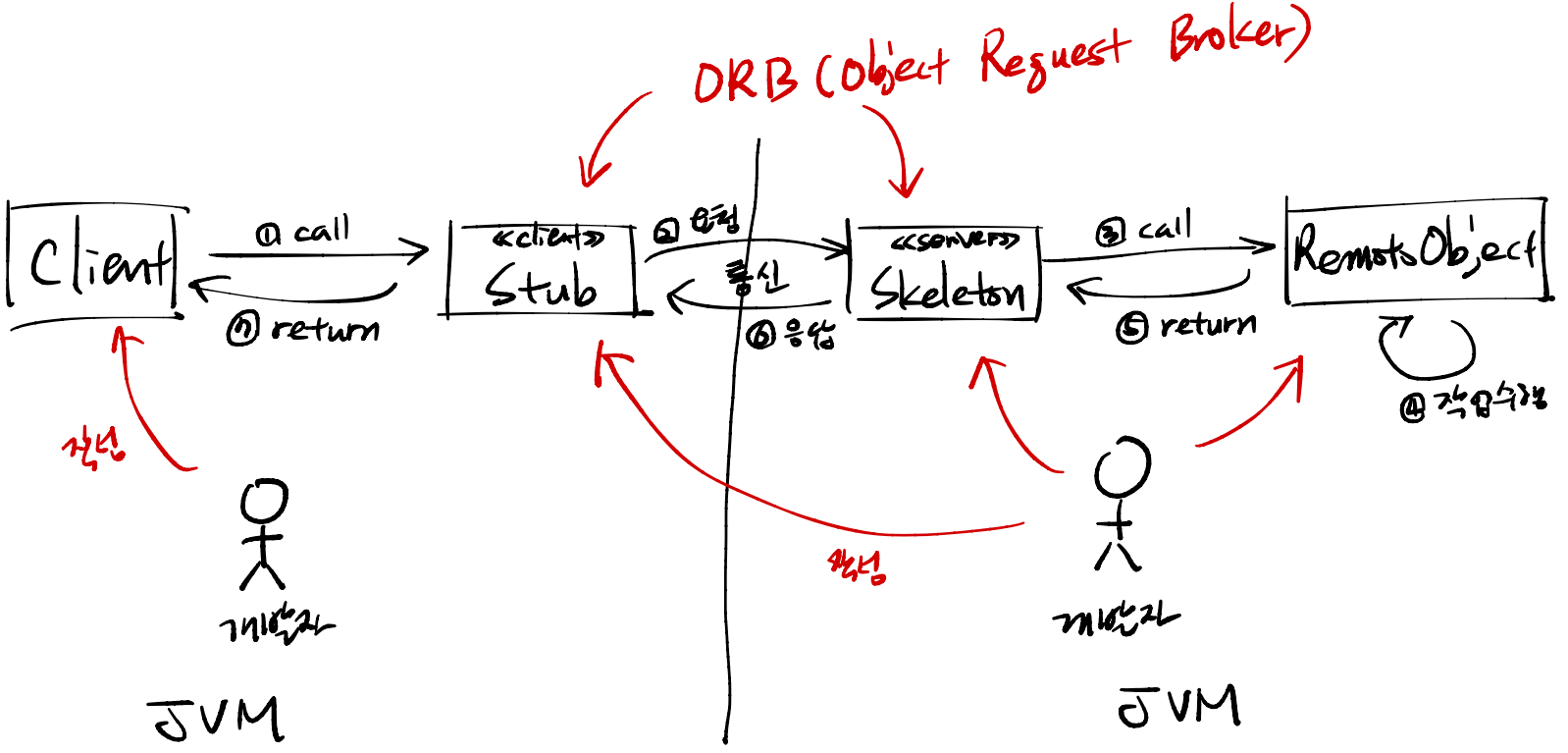
① Adapter



② Proxy



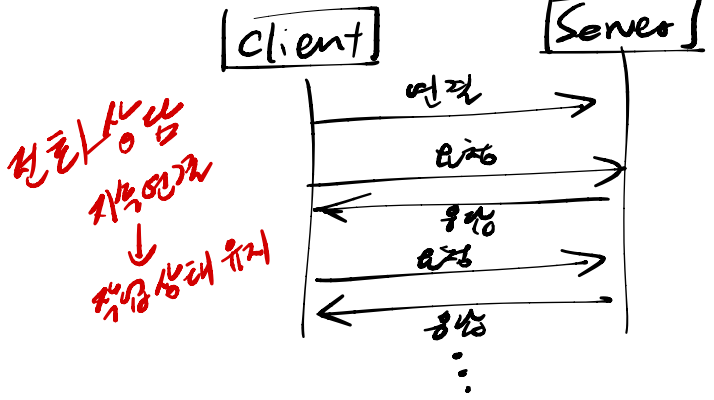
* Remote Object 사용
  ↳ 다른 JVM에서 만든 객체
  (프로세스)



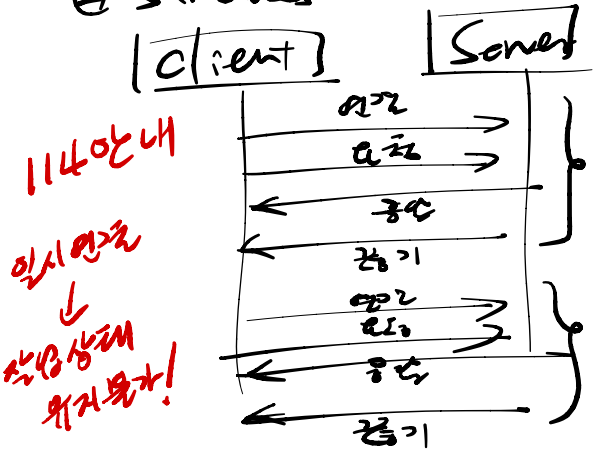
38. Stateful vs Stateless

* Stateful 방식 – 한번 연결한 후 여러번 요청/응답 수행
  Stateless 방식 – 한번 연결한 후 한번 요청/응답 수행

① Stateful



② Stateless



* 통신 방식

connection-oriented

↑

연결형 통신

(TCP 통신)

↓

FTP, Telnet 등

↓ + 보안

SSH

전화

connectionless

↑

비연결성 통신

(UDP 통신)

↓

ping, broadcasting 등

편지

FTP, Telnet, SSH, 스트리밍, 채팅 등

Stateful – 한 번 연결 시 여러 번 송수신 (요청/응답)

상담

지속연결 ⇒ 작업 상태보관

- 기존 작업 결과를 바탕으로 이어지는 작업을 처리할 때 적합
- 요청/응답 메시지 = 연결 사이에 주고받음
- 적은 수의 연결 = 메모리에 연결정보 유지

HTTP(1.1, 1.1, 2), 메신저 등

Stateless – 한 번 연결 시 한 번 송수신 (요청/응답)

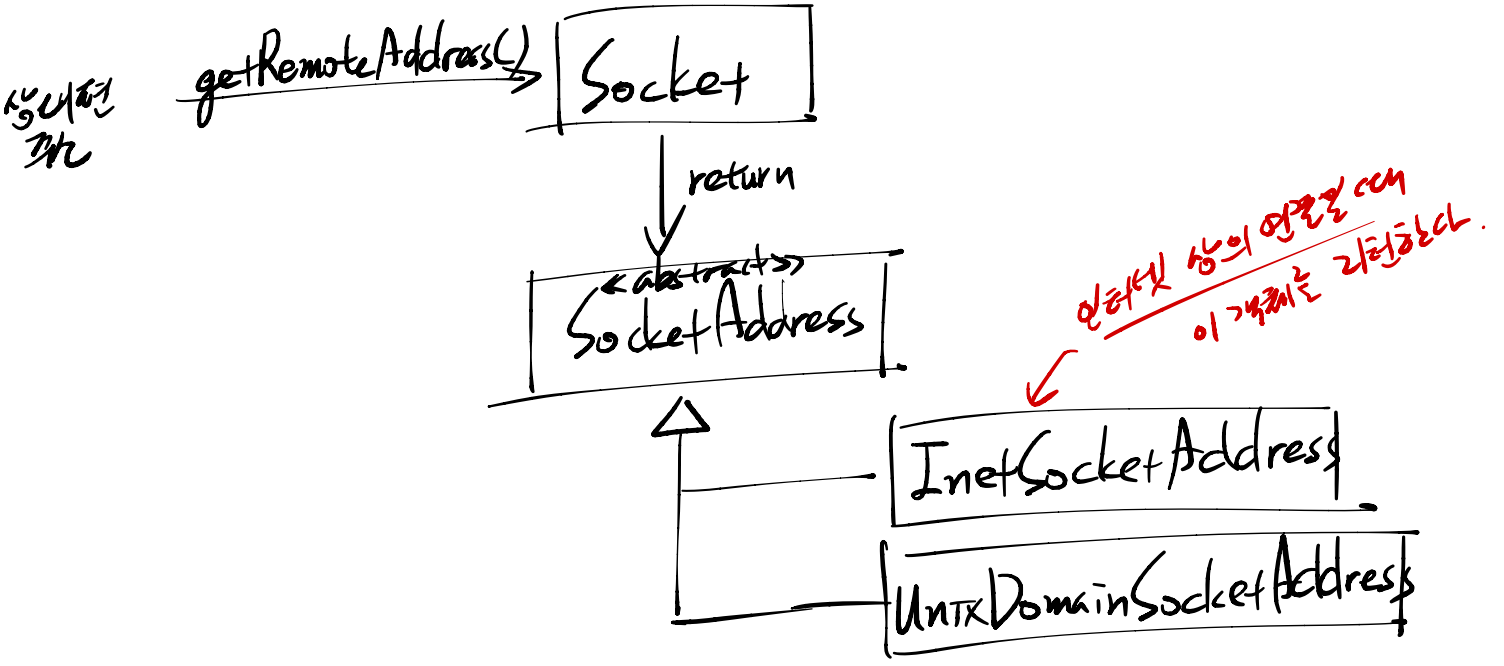
내비 안내

일시연결 ⇒ 작업 상태 유지 불가

- 요청 때마다 독립적으로 수행해야 하는 작업을 처리할 때 적합
- 요청/응답 스트림 = 매번 연결
- 많은 수의 연결 = 연결정보 유지 안함

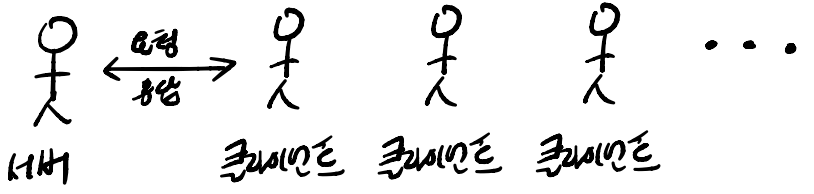
HTTP3

* Socket 관련


상대편 정보
getRemoteAddress()
Socket
return
<<abstract>>
SocketAddress
인터넷 상의 연결일 경우 이 객체를 리턴한다.
InetSocketAddress
UnixDomainSocketAddress

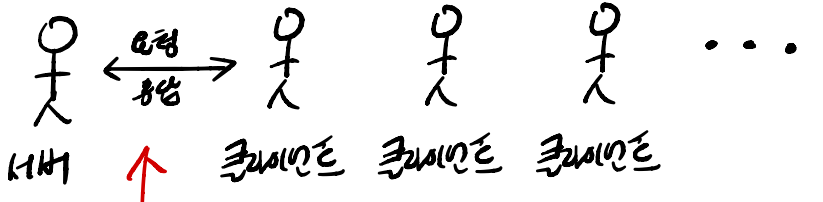
39. 멀티스레드 모델

① 다중 클라이언트 요청 처리 — Stateful



✓ 서버와 연결된 클라이언트가
연결을 끊을 때까지
다른 클라이언트를 기다려야 한다.
✓ 항상 순서대로 처리 한다.

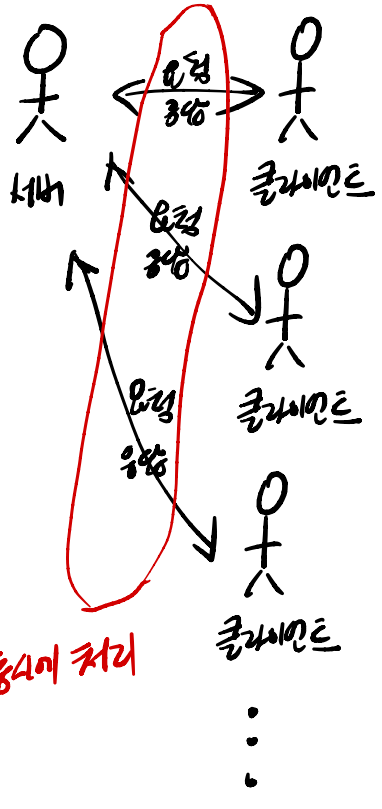
② 다중 클라이언트 요청 처리 — Stateless



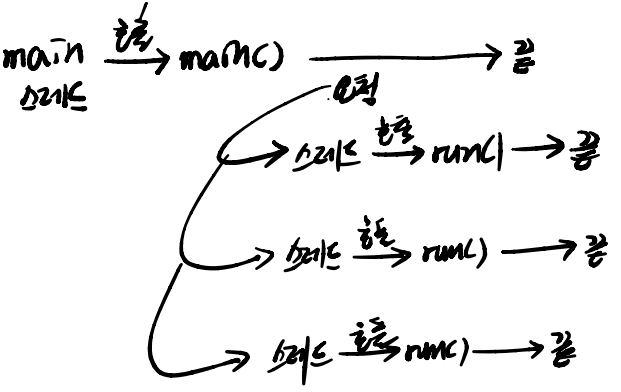
한번 요청과 응답을 수행하면
바로 연결을 끊는다.

✓ 한번연결 → 한번 요청/응답
↓
Stateful 방식보다 대기 시간이 짧다.
✓ 항상 순서 대로 처리

③ 다중 클라이언트 요청 처리 — Multi-threading ⇐ stateful & stateless 방식의 근원적인 프레임 해결



동시에 처리



요청처리에 시간이 오래걸리면
그 시간만큼 대기해야 하므로

⇓

클라이언트 요청을 순차적으로 처리!

↓ 왜?

병행으로 처리!

⇓

클라이언트 요청 하나를
main의 실행 흐름과
분리해서 따로 실행!

* 스레드를 만드는 방법 — main 실행흐름에서 따로 실행흐름을 만들어내는 방법

```
class 실행흐름명 extends Thread{
    void run(){
        따로 실행할 코드
    }
}
```

```
실행흐름.start();
```

↓

별도의 실행흐름이 시작된다.