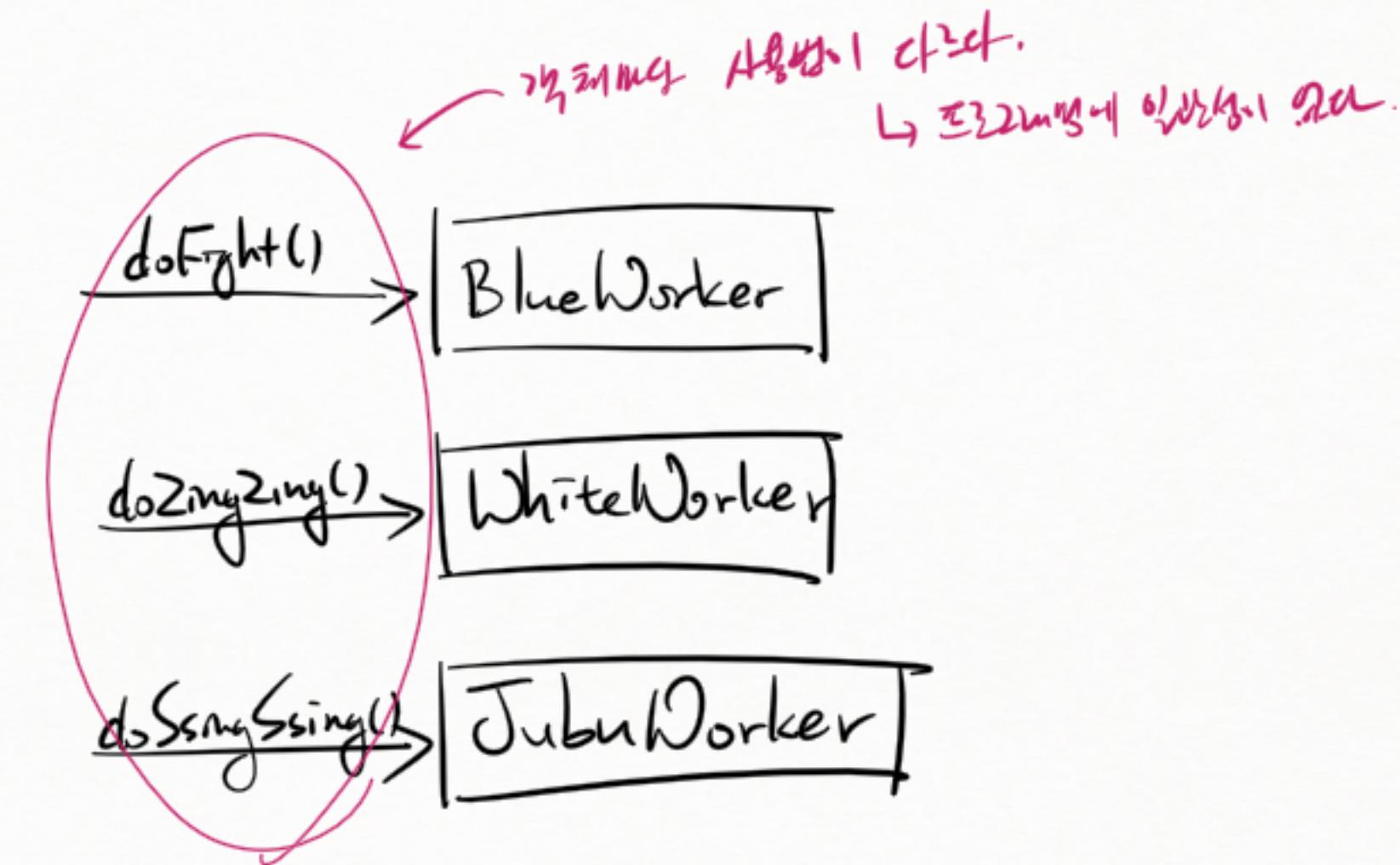
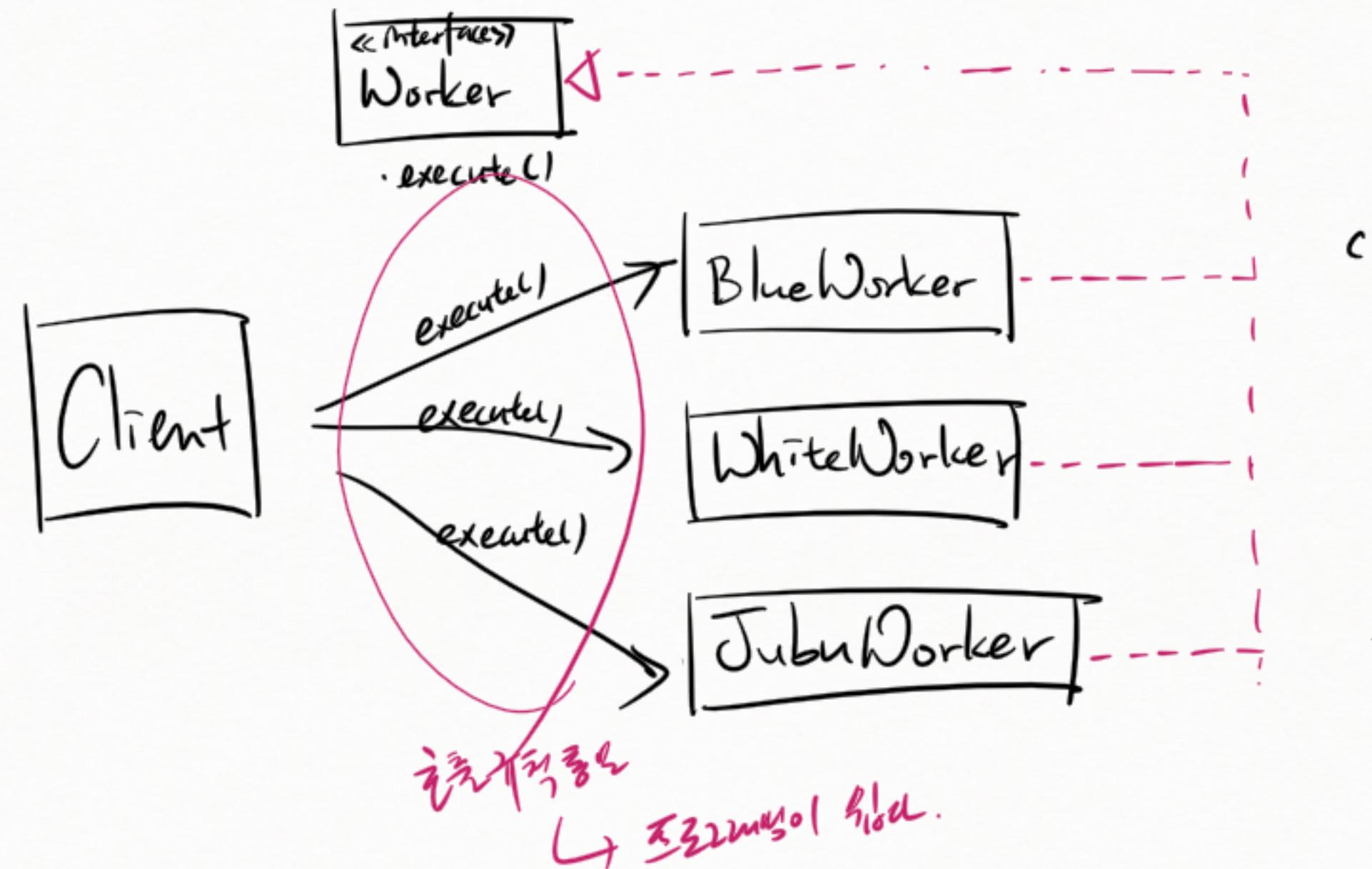


인터페이스 (Interface)

## \* 인터페이스 사용 전

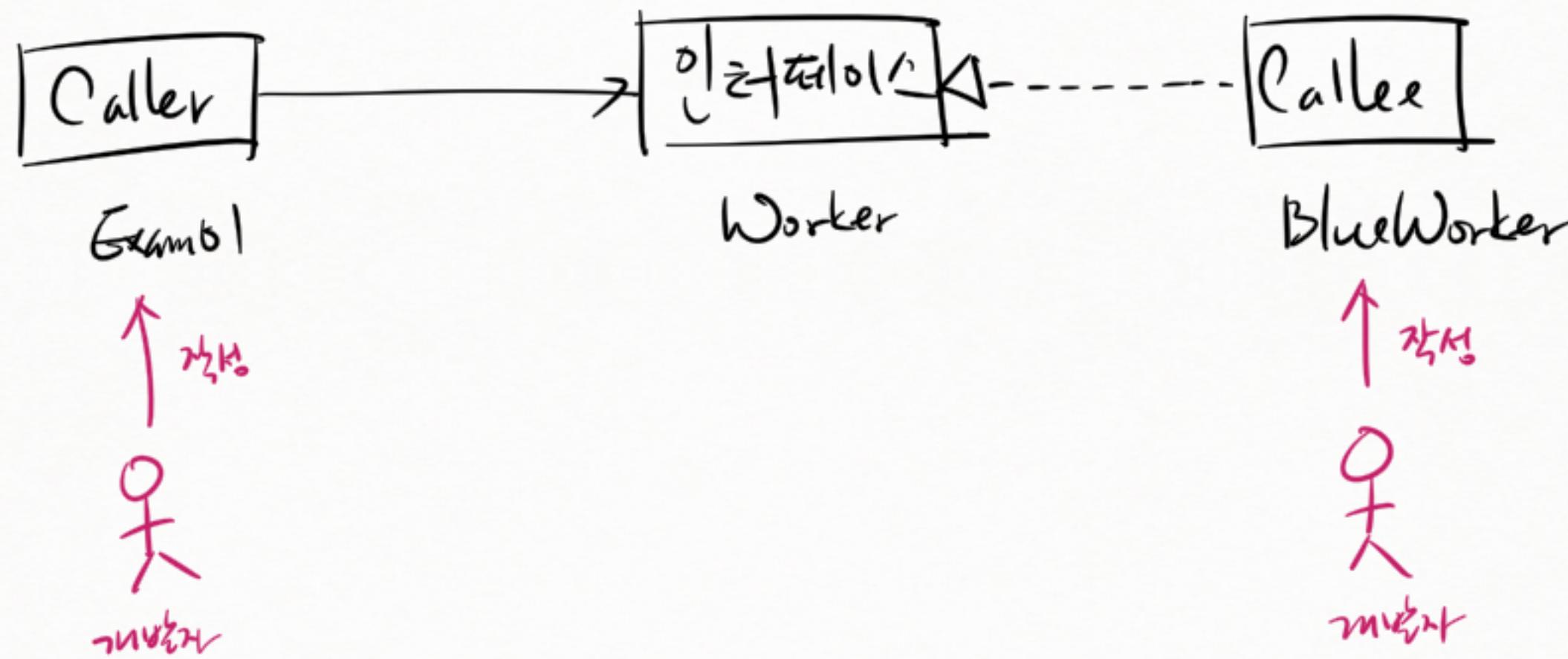


\* 인터페이스 사용 후

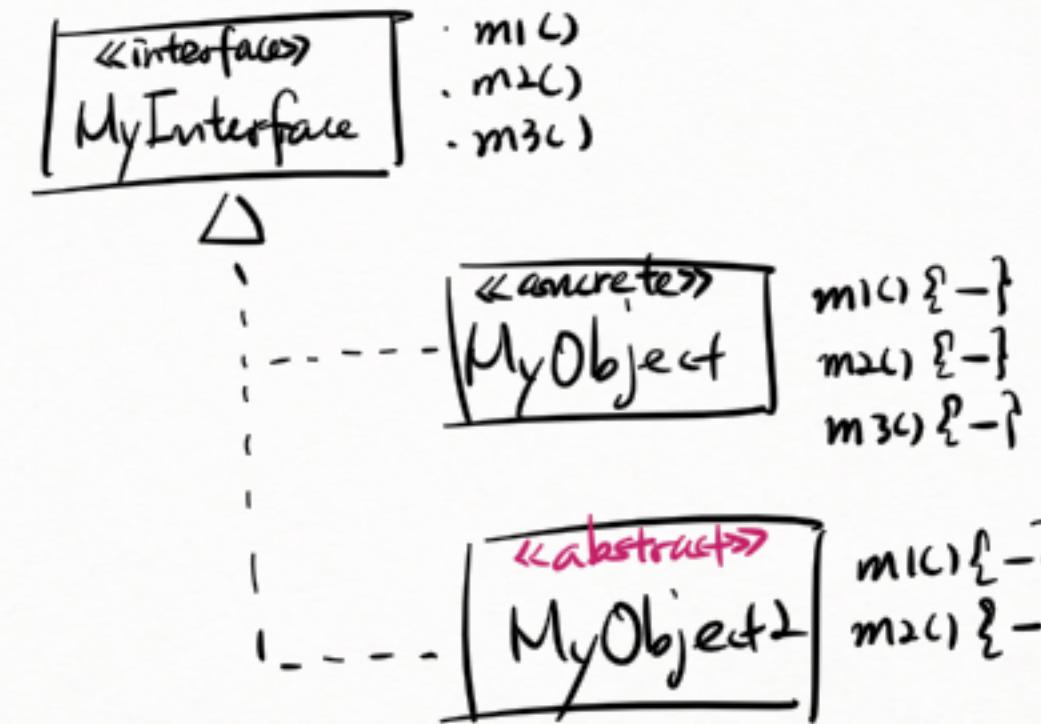


```
class BlueWorker  
implements Worker {  
    ...  
}
```

\* 인터페이스 와 caller / callee



## \* 인터페이스의 구현



MyInterface ref;

ref = new MyObject();

↑

↑

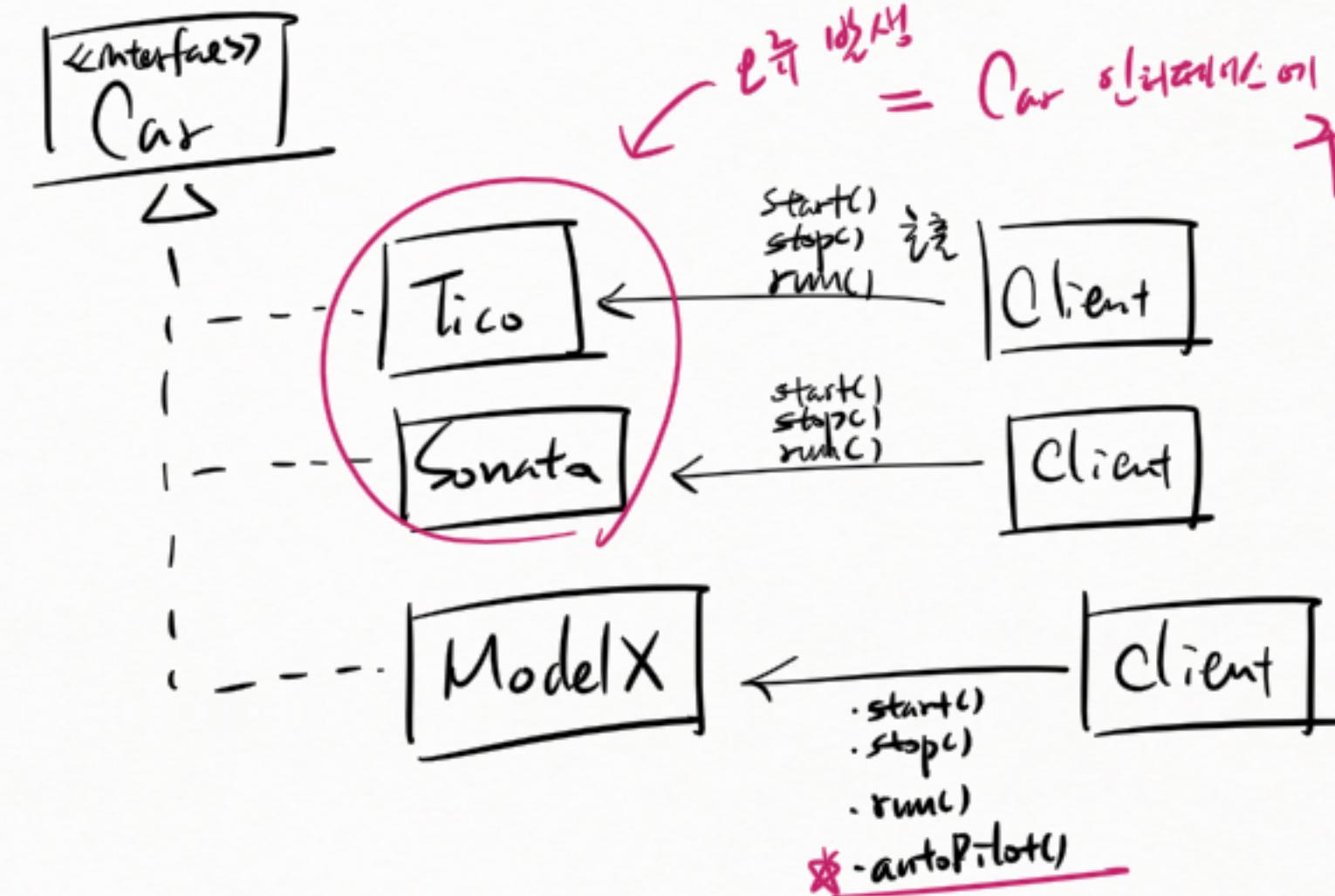
인터페이스를 구현한 클래스의

Ol스턴스 주소

\* default 메서드 사용

```
interface Car {  
    - start();  
    - stop();  
    - run();  
}
```

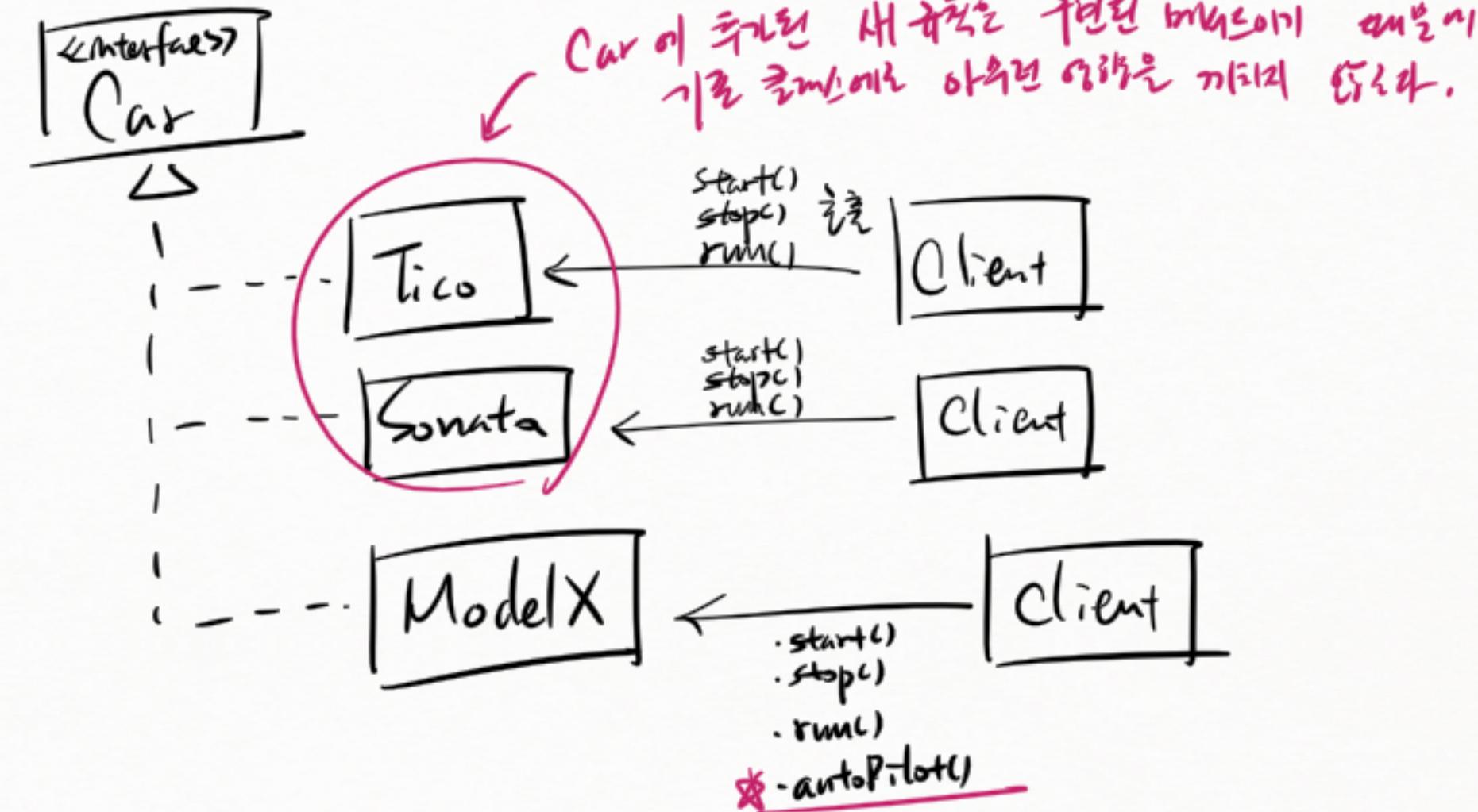
↑  
구현 추가  
+ autopilot();



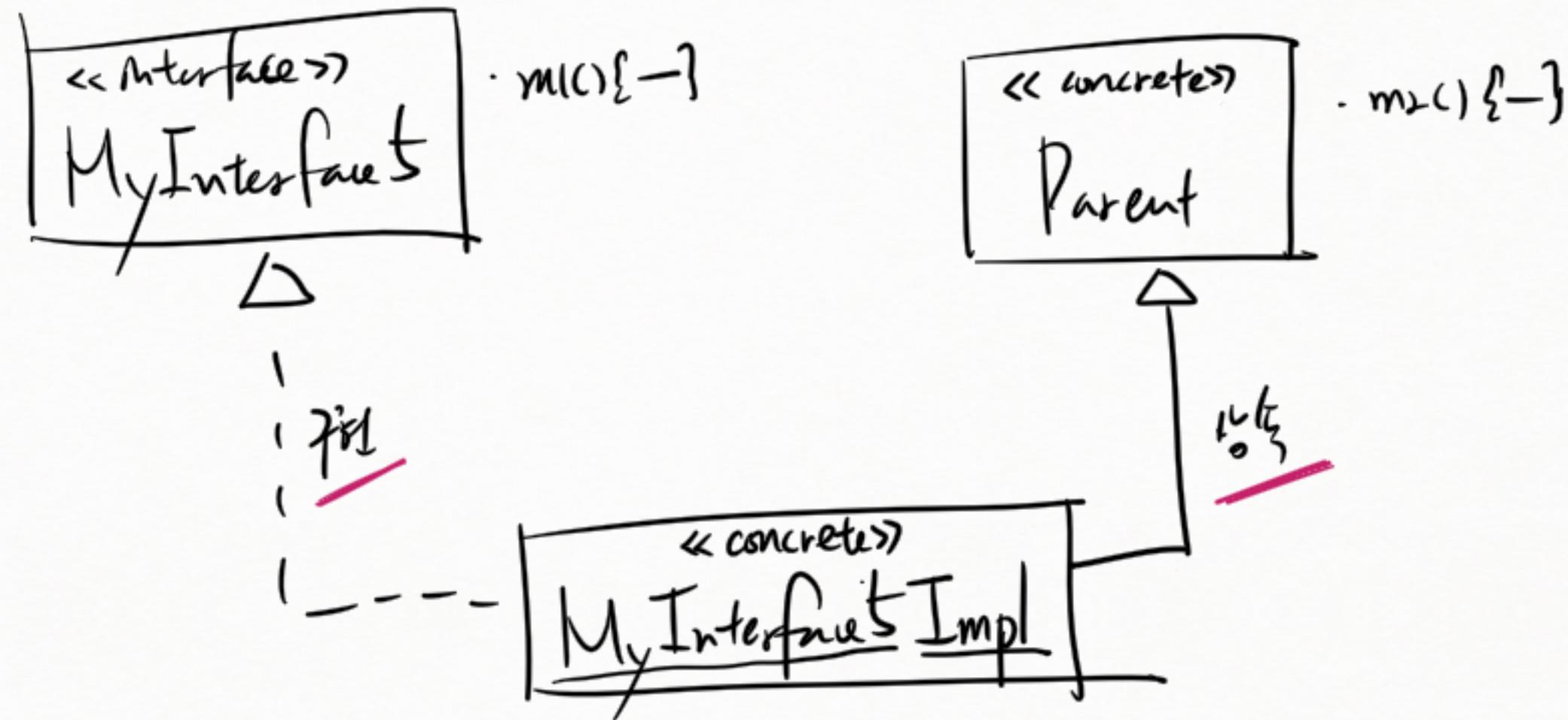
→ = Car 인터페이스에 추가된 새 기능은 구현(정의)되어 있겠지  
증명이다.

\* default interface 헬퍼

```
interface Car {  
    - start();  
    - stop();  
    - run();  
}  
  
default autopilot() {}
```

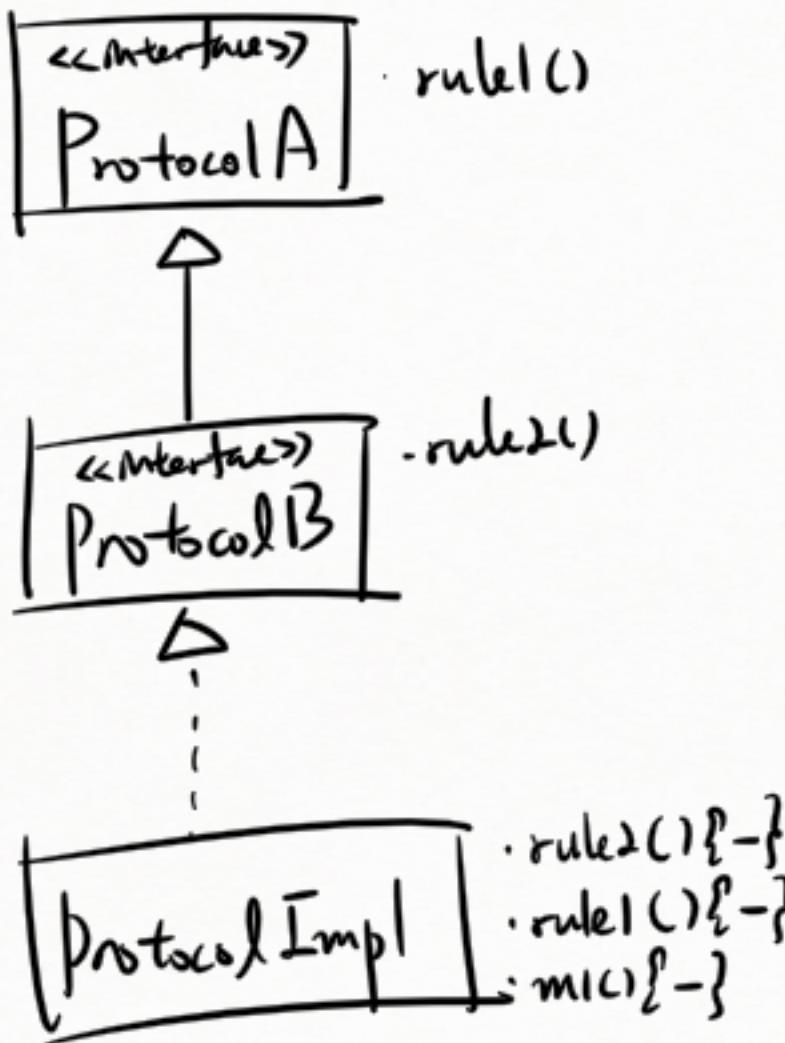


\* static b1/b2



\* 인터페이스 상속 구현

↳ 인터페이스를 통한 메서드 호출 방식



ProtocolImpl obj = new ProtocolImpl();

obj. m1();  
obj. rule1();  
obj. rule2();

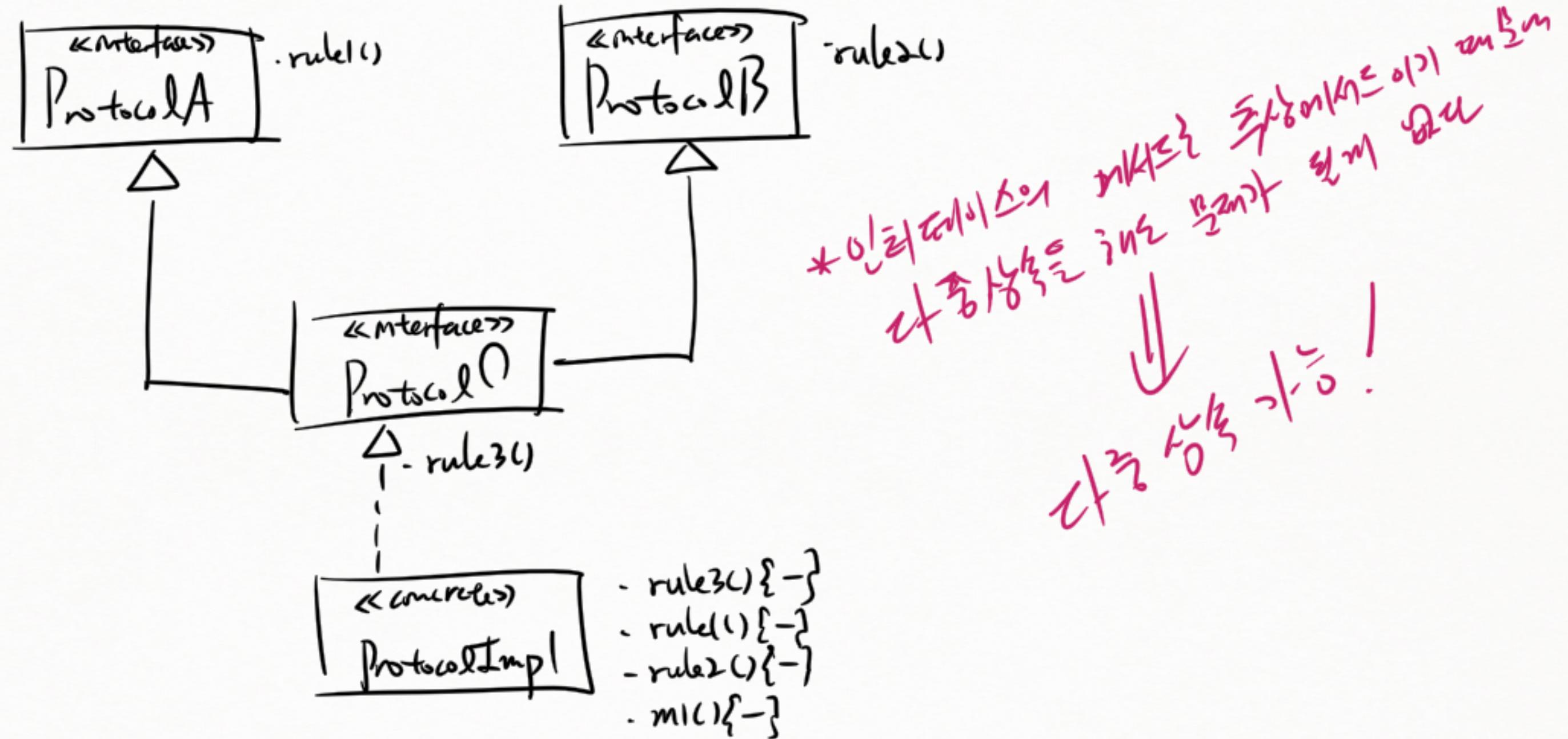
ProtocolB obj2 = obj;

~~obj2. m1();~~  
~~obj2. rule2();~~  
~~obj2. rule1();~~

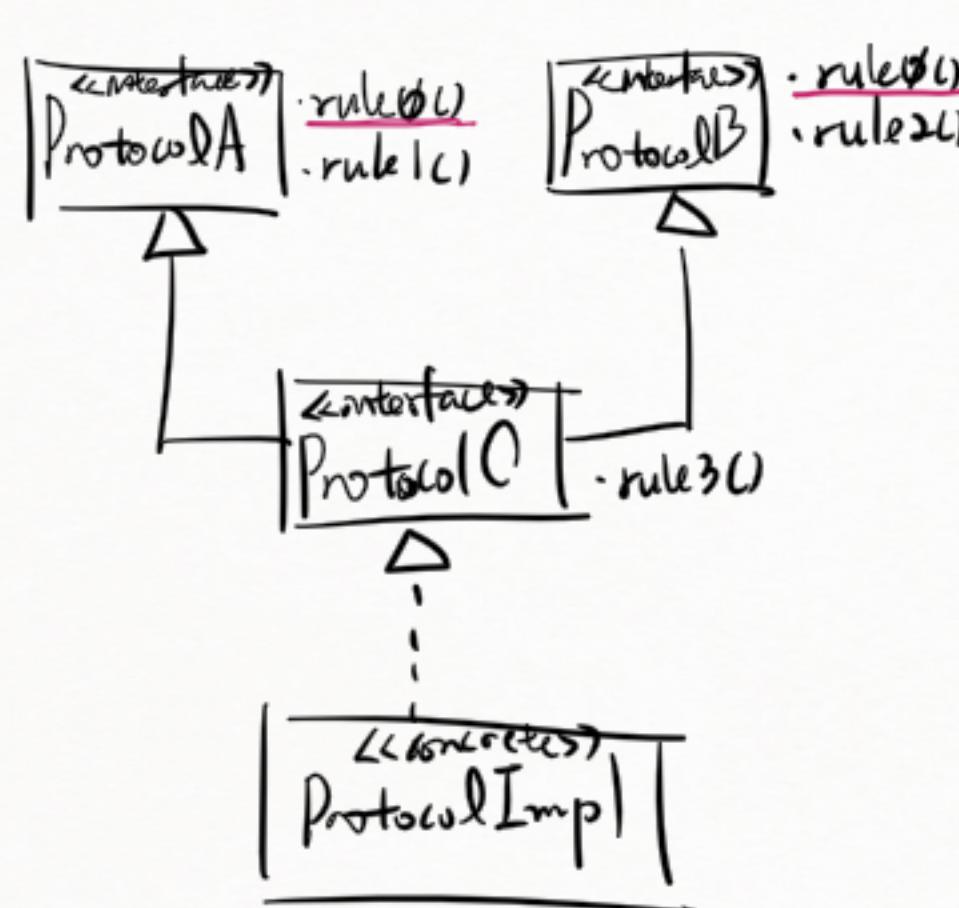
~~obj3. m1();~~  
~~obj3. rule2();~~  
~~obj3. rule1();~~

ProtocolA obj3 = obj;

\* 인터페이스 다중 상속



\* 이전에는 다음과 같은  
가능한 경우

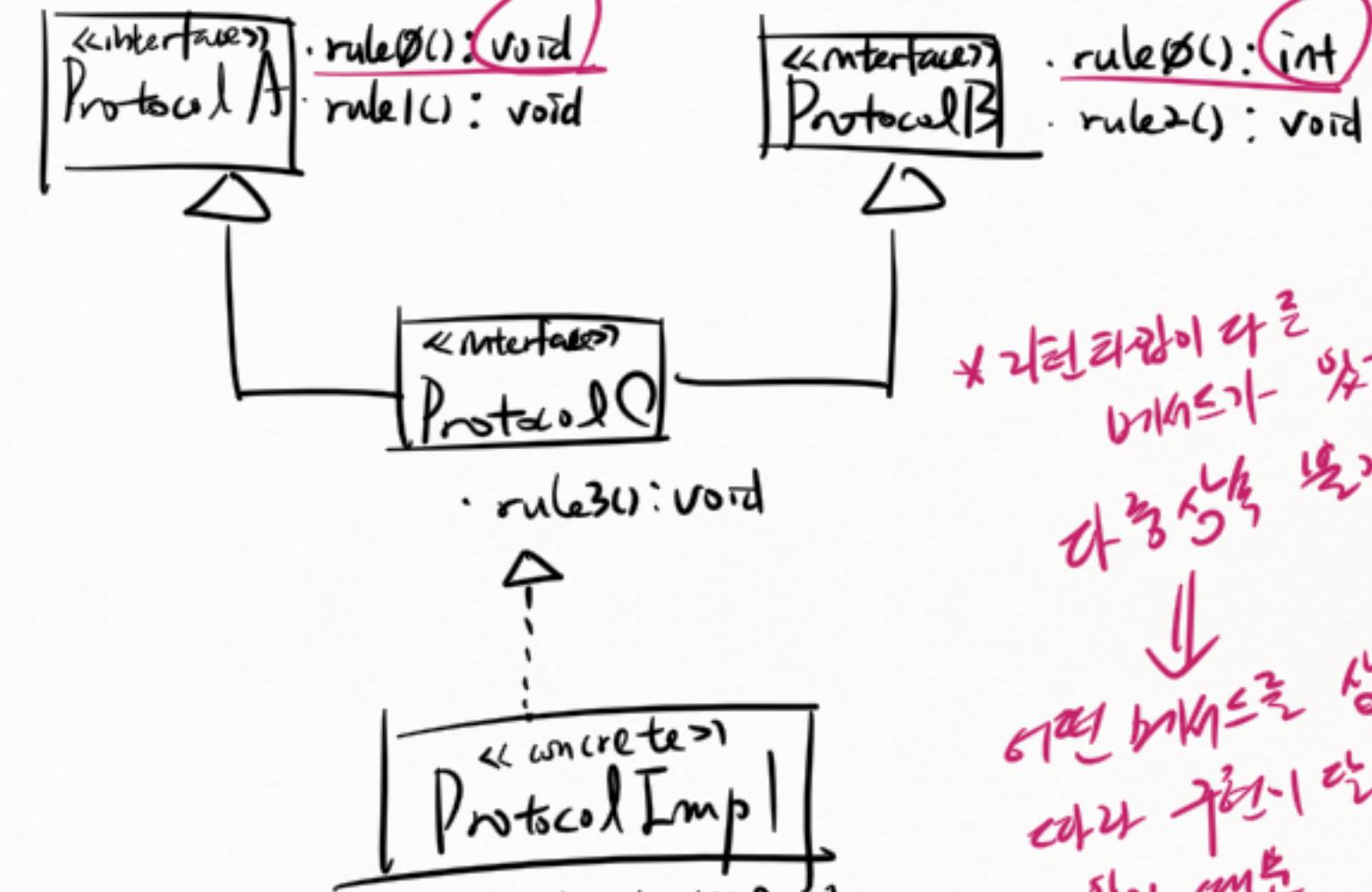


ProtocolA  
rule0()  
rule1()

ProtocolB  
rule0()  
rule2()

ProtocolC  
rule3()

불가능한 경우



이전처럼 다를 수  
없는 H2에 대한  
부모가 다른 두  
자식을 갖다.

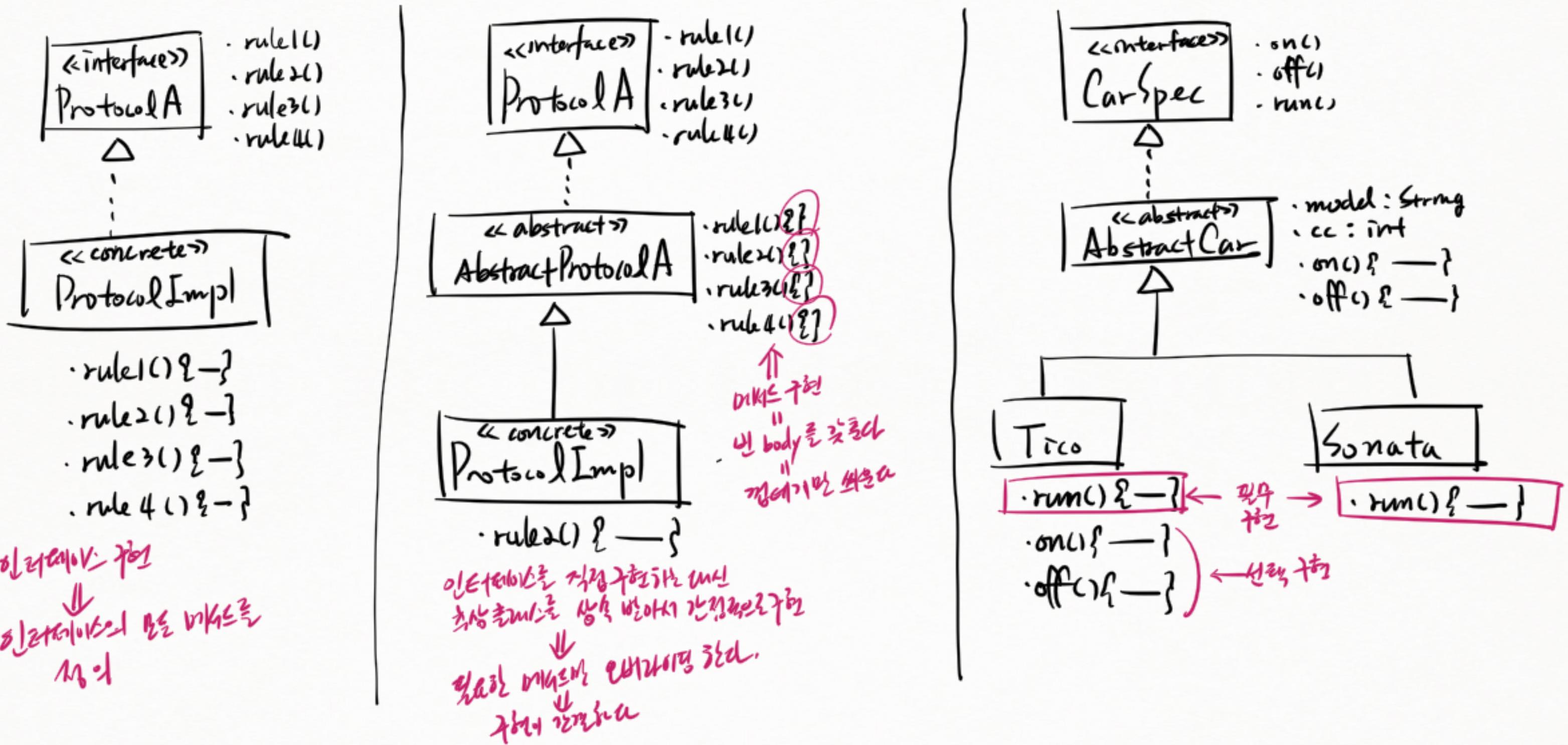
• void rule0()  
• int rule0()

\* 이런 이유로 다를 수  
없는 부모가 있기 때문에  
다른 자식을 두고  
있지 않아!

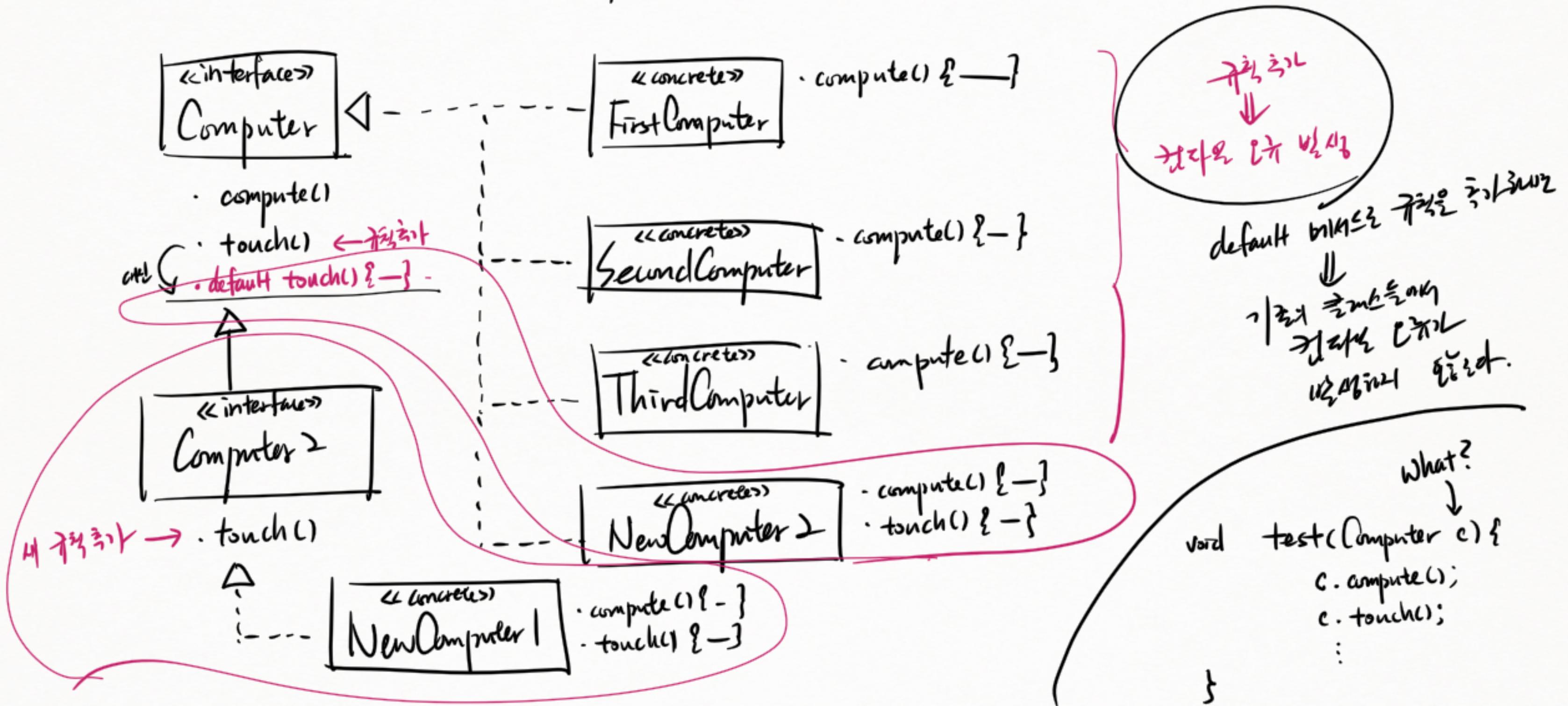
↓

여전히 부모는 상속 받는 부모  
보다 자식이 더 많지  
않기 때문

## \* 인터페이스와 추상 클래스



\* 인터페이스의 default 메소드



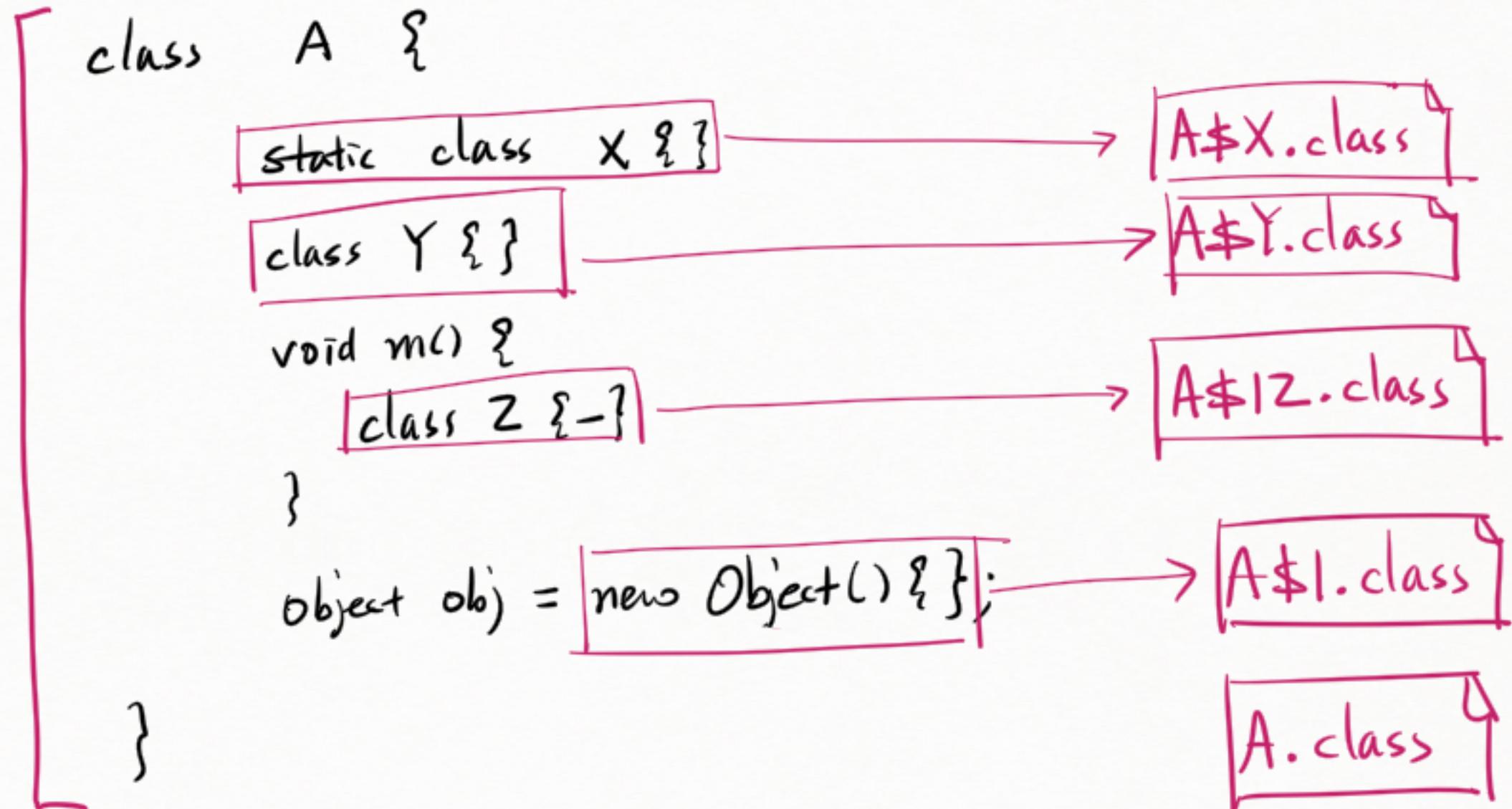
깊은  
정적  
클래스  
( nested class )

\* 중첩 클래스의 종류  
nested

```
class A {  
    static class X {} } ← static nested class  
    class Y {} } ← non-static nested class  
    void m() {  
        class Z {} } ← local class  
    }  
    Object obj = new Object() {}; ← anonymous class  
}
```

\* 중첩 클래스와 .class 파일

A.java



\* static nested class 와 non-static nested class의 차이

class A {

  static class X { }

  class Y { }

}

Y obj = new Y();

    ↓  
    new Y(this);

    ↳ Initialization  
    ↳ 바깥 클래스의 객체 주소  
    ↳ 초기화 코드로 사용된다

static class X {  
  X() { }  
}

class Y {

  A this\$0;

  Y(A arg) {

    this\$0 = arg;

  }

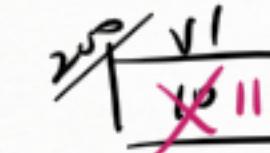
    ↳ 컨структор가 기본 생성자 추가

    ↳ 컨структор,

    ↳ 바깥 클래스의  
    ↳ 객체 주소를 받을  
    ↳ 레퍼런스를 추가.

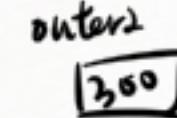
    ↳ 생성자에서  
    ↳ 바깥 클래스의 인스턴스를  
    ↳ 받을 수 있도록  
    ↳ 파라미터를 추가

B3 outer = new B3();

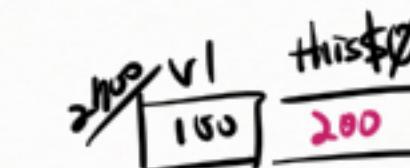
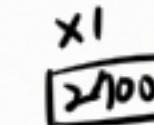


outer.v1 = 11;

B3 outer2 = new B3();



B3.X x1 = ~~outer~~.new X();



x1.test();  
↑ this

B3.X x2 =

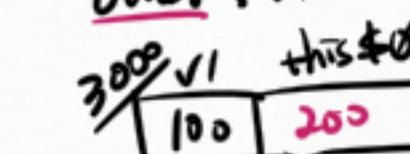


x2.test();  
↑ this

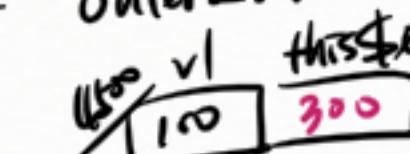
B3.X x3 =



~~outer~~.new X();



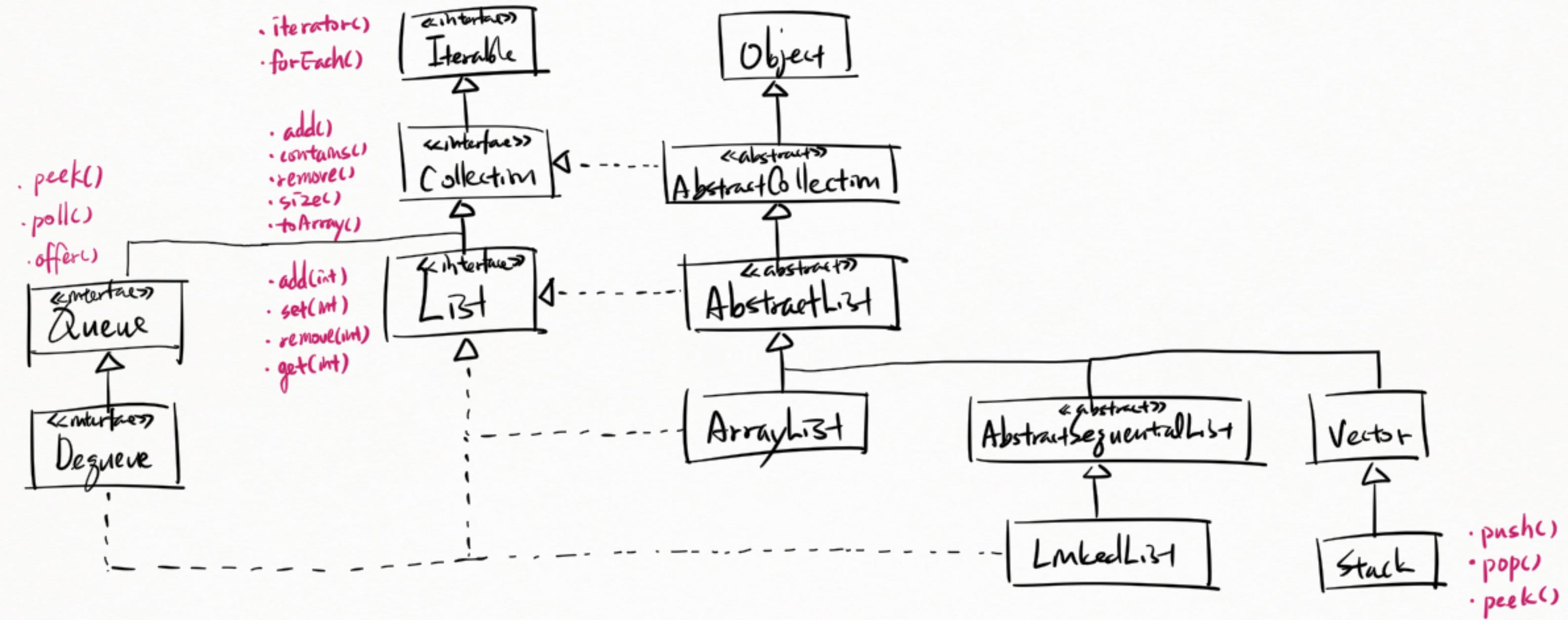
~~outer2~~.new X();



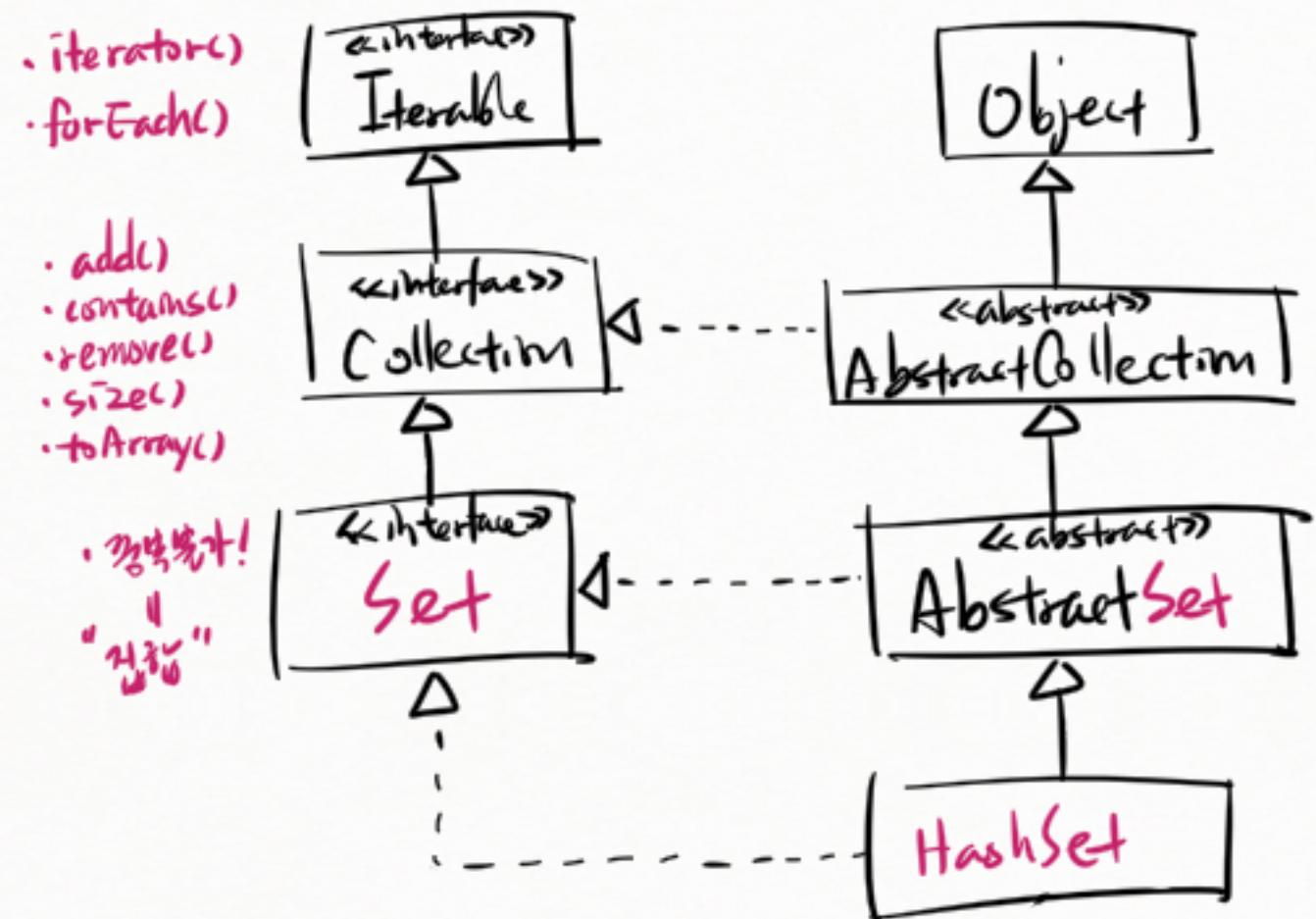
450  
x3.test();

Collection API

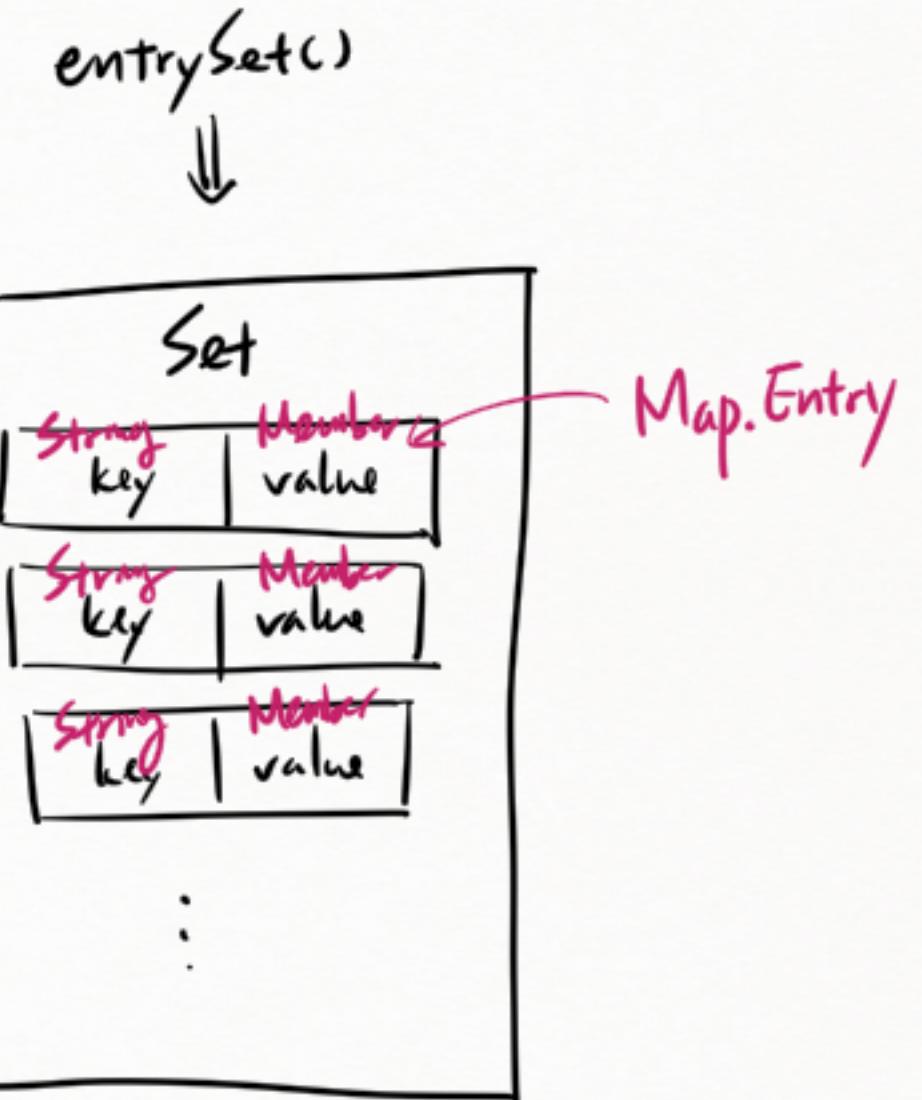
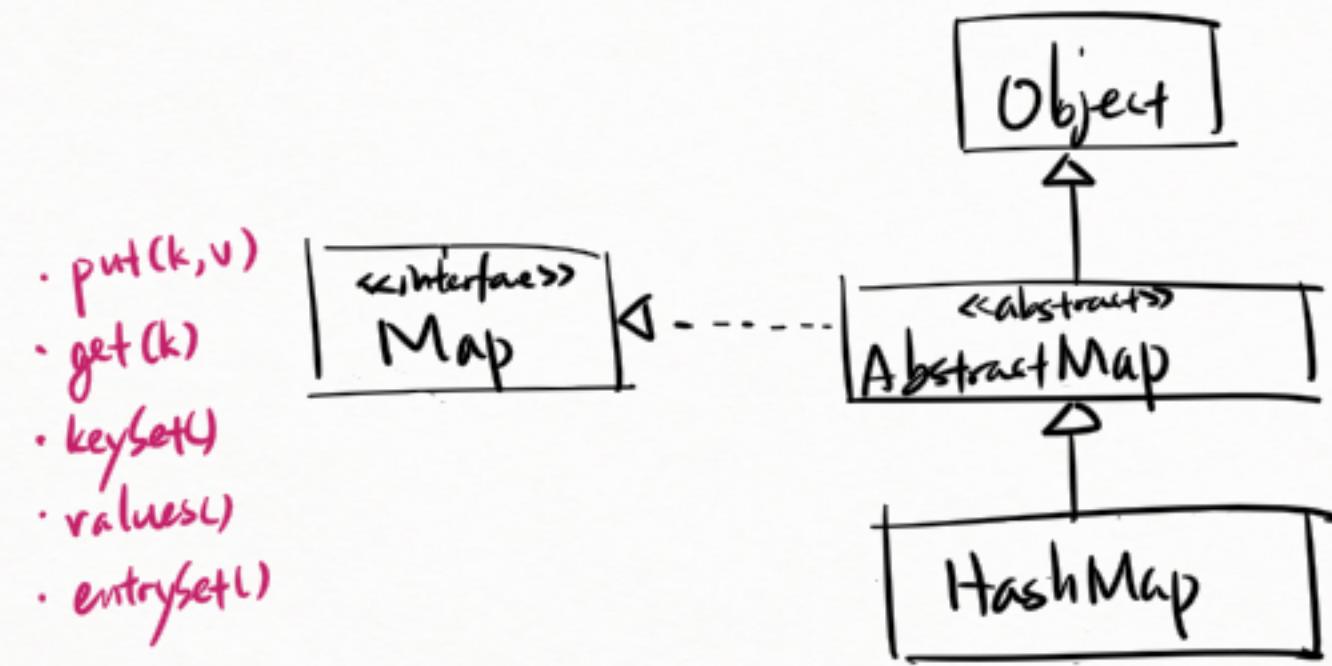
## \* List の構成と階層 (hierarchy)



## \* Set သုတေသနများ ရှိခိုင်း (hierarchy)

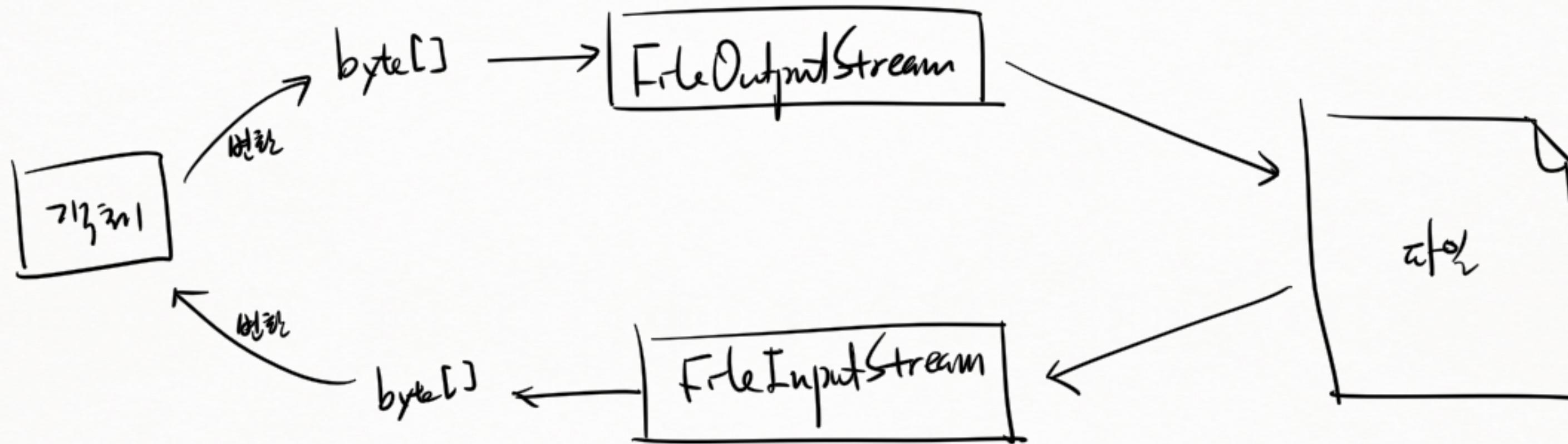


\* Map の構成と階層構造 (hierarchy)



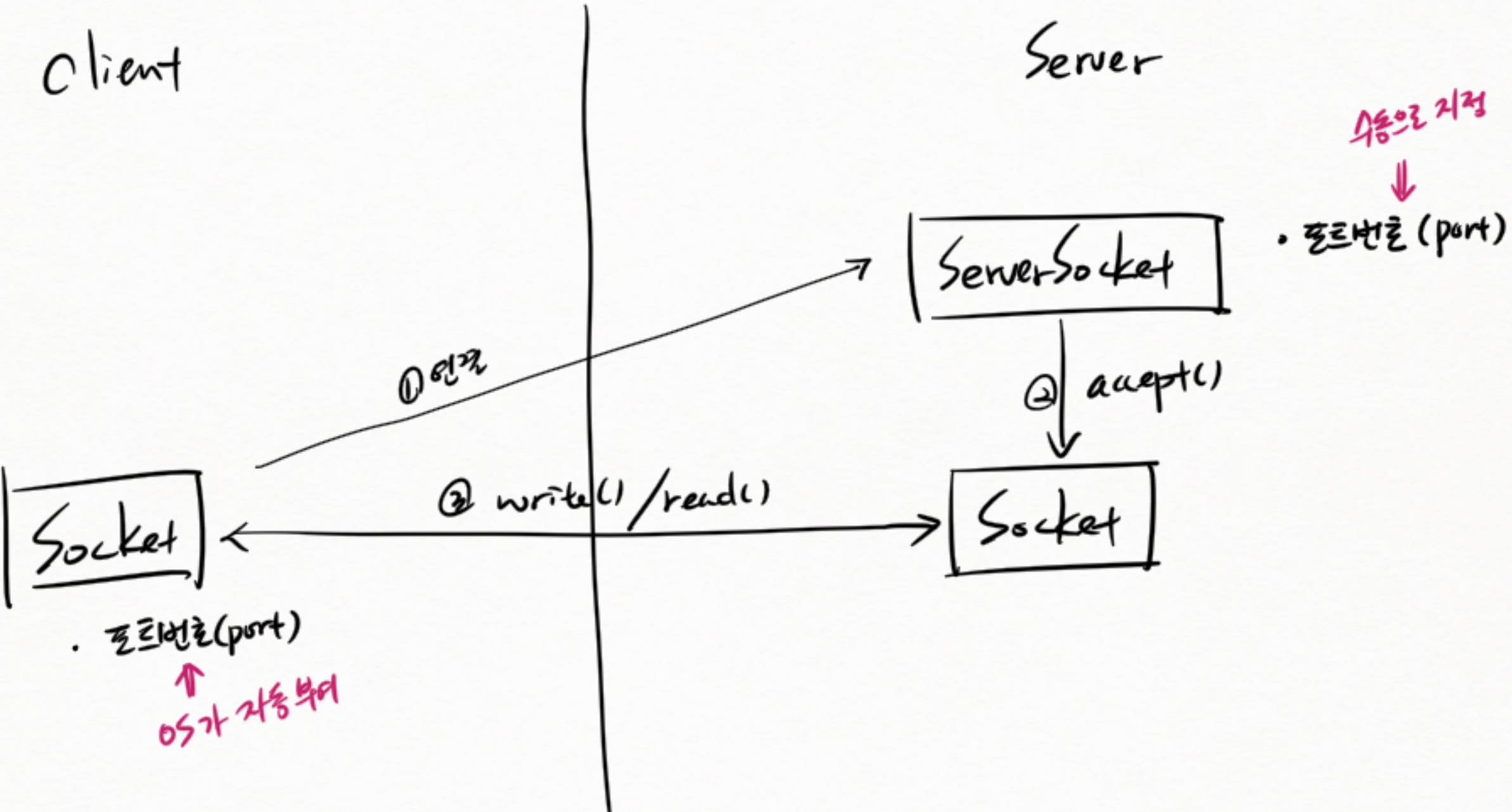
File I/O API

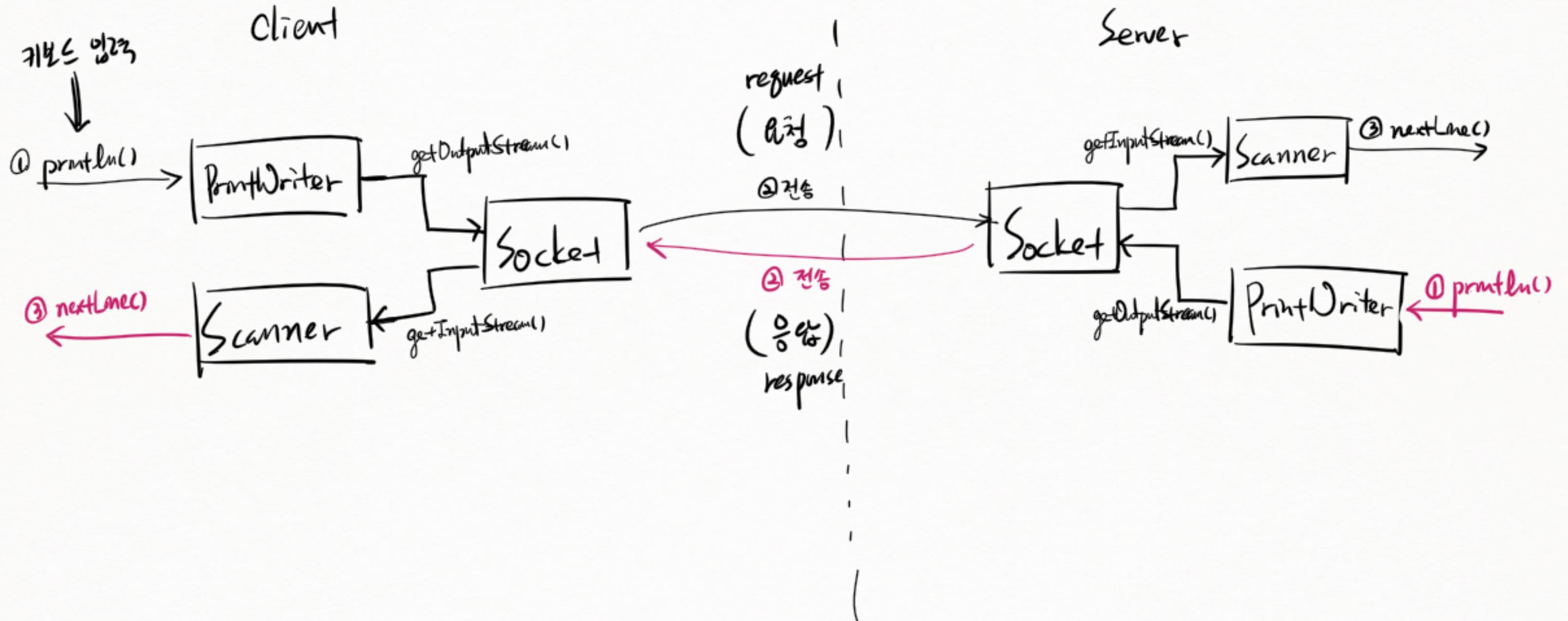
## \* FileOutputStream / FileInputStream



Networking

# Client / Server





## \* Networking 연결 방식

Connection-Oriented (연결 지향)

예) 전화, WWW, LOL, 키글리드, 웹메일, 유튜브

프로토콜  $\Rightarrow$  TCP

Connectionless (비연결 지향)

예) 네트워크, 대화, 멘시, ping

프로토콜  $\Rightarrow$  UDP

Stateful

예) 상호전화

프로토콜: SSH, Telnet, FTP, 채팅



Stateless

예) 웹 앤내



\* Connection-Oriented

- ✓ 단기적인 연결에만 사용을 허용
- ✓ 단기적인 후 데이터 전송  $\Rightarrow$  신뢰성 높음.

(TCP) HTTP 1/2

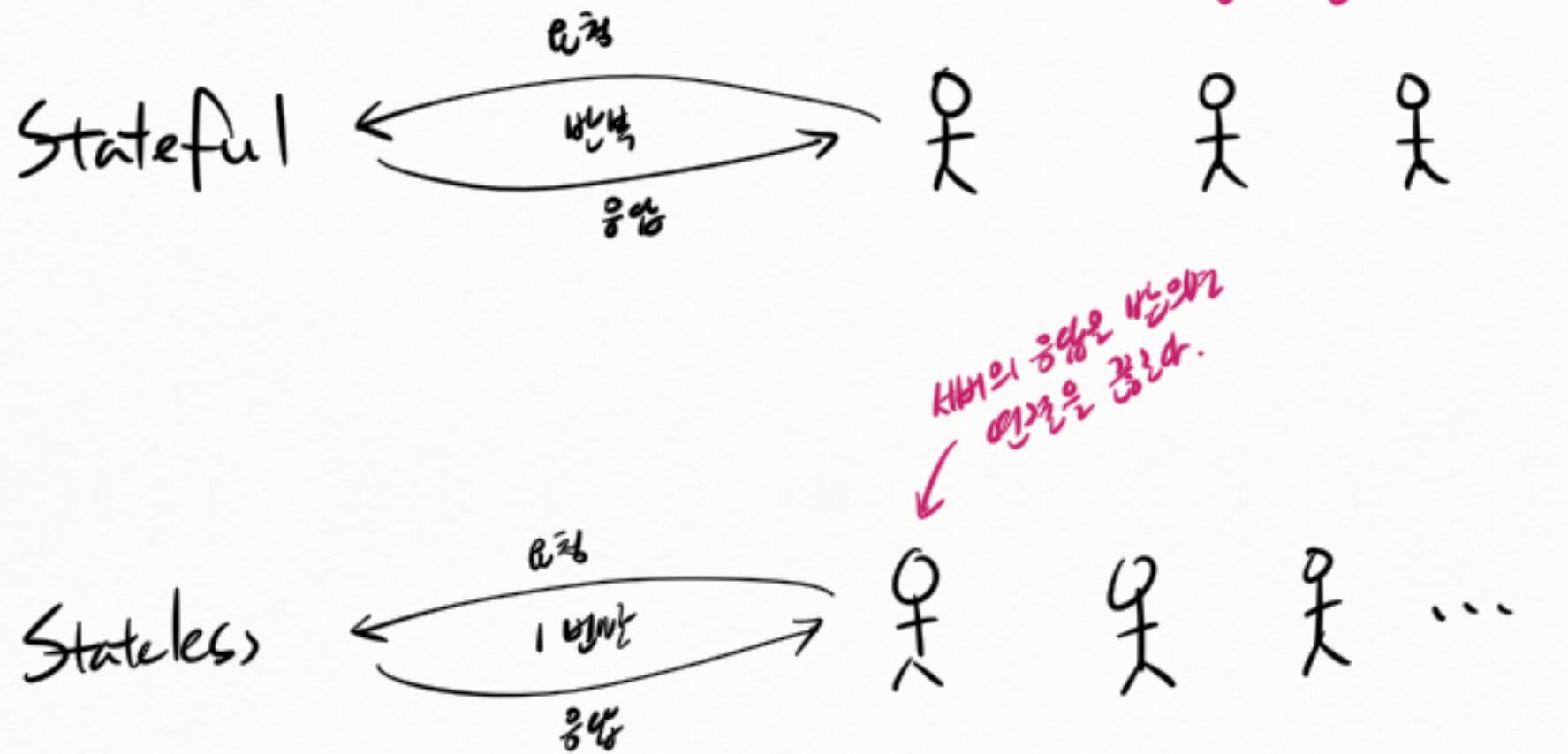
\* Connectionless

↓

(UDP) HTTP 3

- ✓ 단기적인 접속  $\Rightarrow$  신뢰성이 낮음.
- ✓ 단기적인 접속 데이터 전송  $\Rightarrow$  신뢰성이 낮음.

\* Stateful 허용하고 Stateless 허용하지



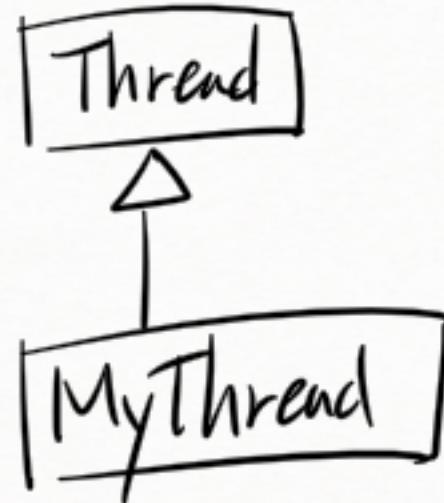
한 클라이언트가  
여러 시스템 연결되어 있어도  
다른 클라이언트의  
대기 시간이 걸어진다.

한번 연결이  
한번의 요청/응답만 처리  
↓  
다른 클라이언트의 연결 대기 시간이  
걸려.  
↓  
어떻든 클라이언트의 요청을 처리

Thread

\* 스레드 생성 및 실행

① 키트레드 만들기



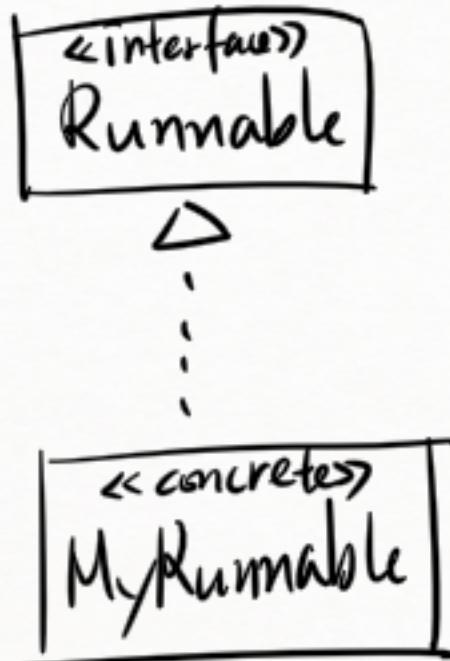
```
MyThread t = new MyThread();
t.start();
```

② 일반 클래스 키트레드 만들기

```
new Thread() {
    run() {
    }
}.start();
```

\* 스레드 생성 및 실행

③ Runnable 향상 활용



MyRunnable r = new MyRunnable();

Thread t = new Thread(r);

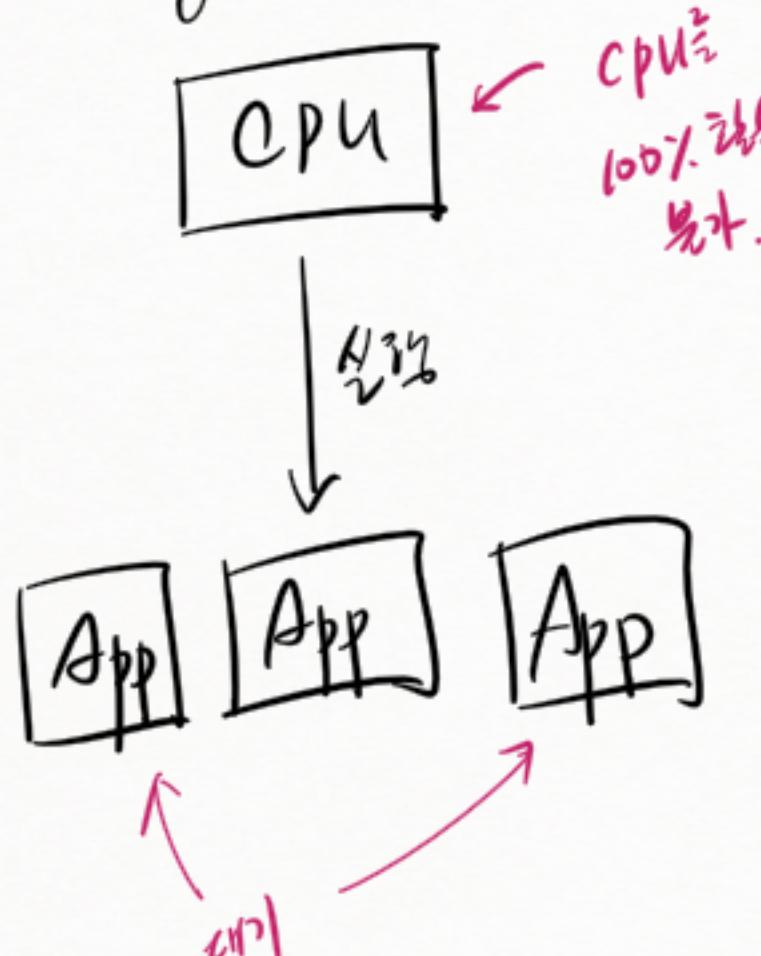
t.start();

④ 빌드 편의를 위한 Runnable 향상

```
new Thread( new Runnable() {
    run() {
    }
}).start();
```

## \* 멀티 태스킹 (Multi-tasking)

## ① Single-tasking



811) Dots

4211

卷之三

25  
짜봉 1  
1000m

11212

四

①  $\rightarrow$  121<sup>2807</sup>  
 ②  $\rightarrow$  121<sup>4446</sup>  
 ③  $\rightarrow$  11<sup>224</sup>

$$\textcircled{1} \rightarrow \text{Zn}^{2+} \text{NH}_3 \left\{ \begin{array}{l} \text{Zn}^{2+} \text{NH}_3^+ \\ \text{Zn}^{2+} \text{NH}_3^+ \\ \text{Zn}^{2+} \text{NH}_3^+ \end{array} \right.$$

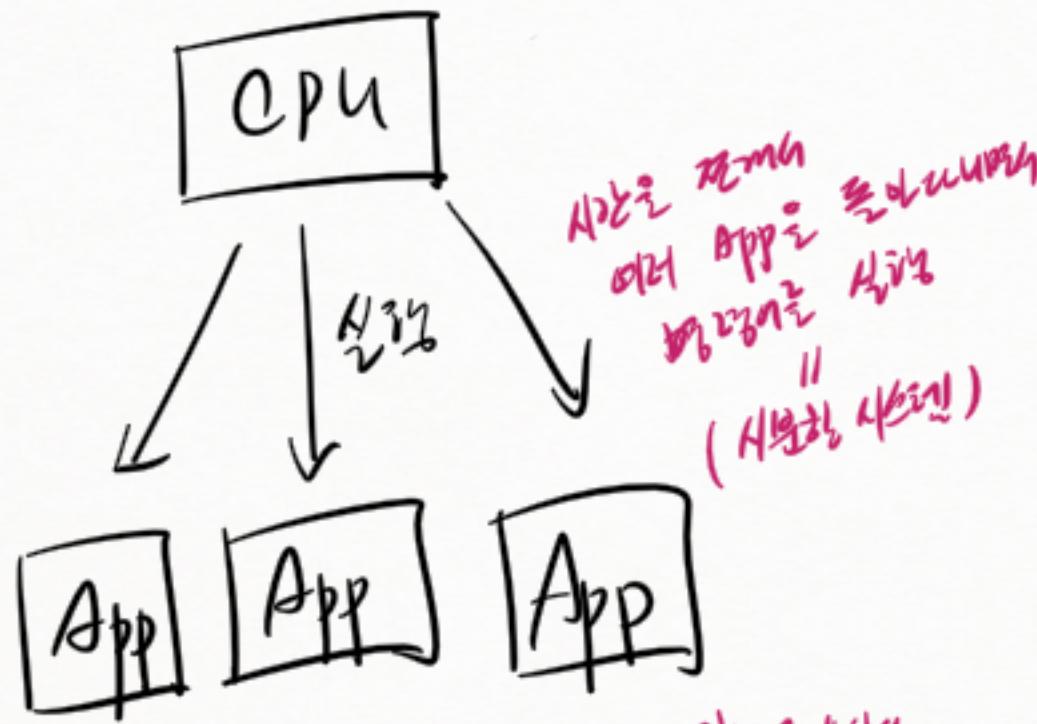
人  
口  
大  
全  
人  
口  
大  
全  
2008.02.1  
四月

제작자  
제작자  
(제작자)  
↓  
!! 의 (제작자)  
제작자 (제작자) 제작자  
↓  
제작자 제작자  
(제작자)

1)  $\frac{1}{2} \sin x - \frac{1}{2} \cos x$

## \* 멀티 태스킹 (Multi-tasking)

### ② Multi-tasking



여기 App이 해석 및 실행하는 것을 말함.

(1) Windows

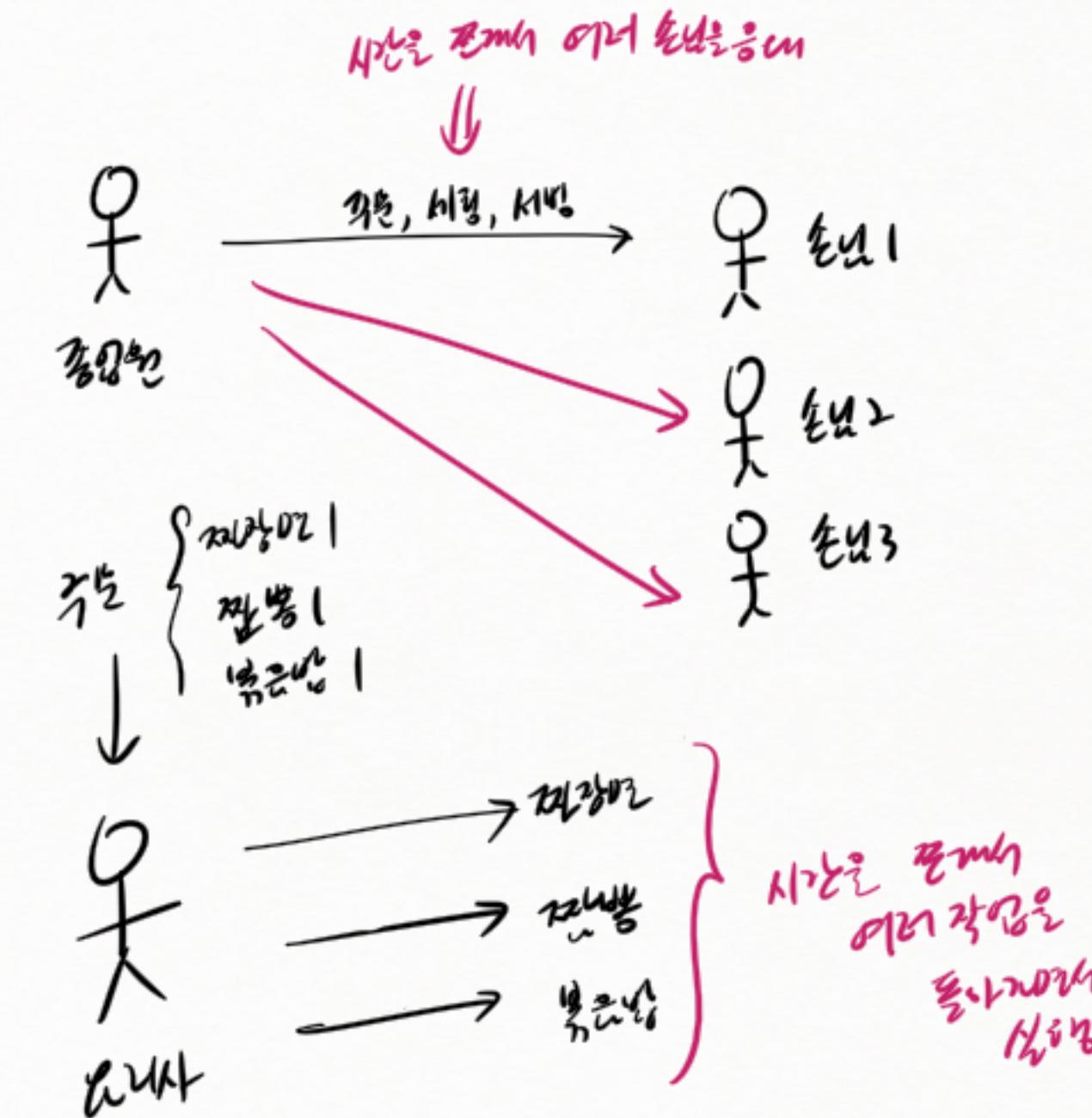
Linux

:

H2(1)

여기 CPU는 여러 App을 동시에 처리하는 것과 같다.  
여기 CPU는 여러 App을 동시에 처리하는 것과 같다.  
(여기 CPU는 여러 App을 동시에 처리하는 것과 같다.)

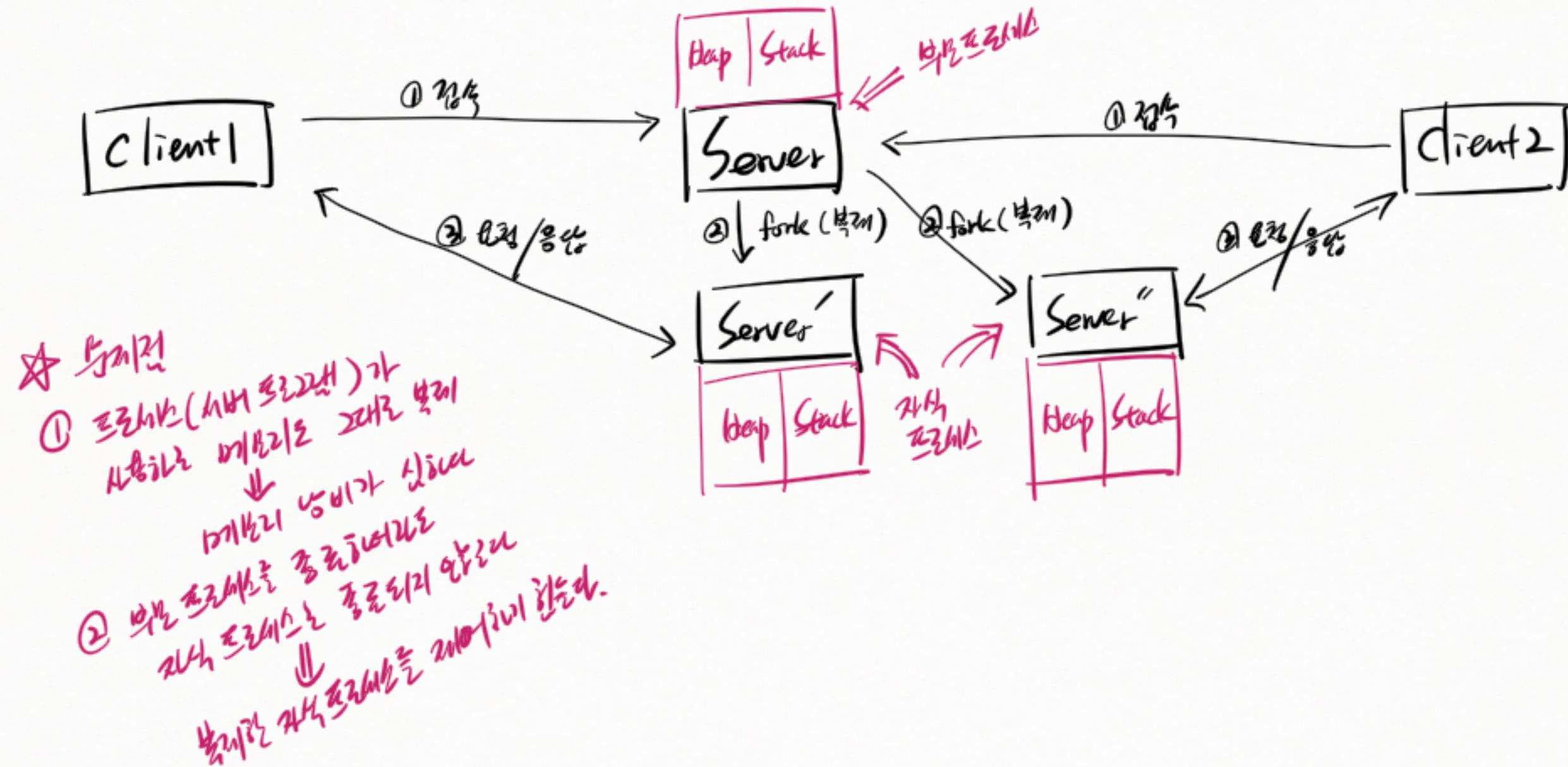
H2(2)



여기 CPU는 여러 App을 동시에 처리하는 것과 같다.

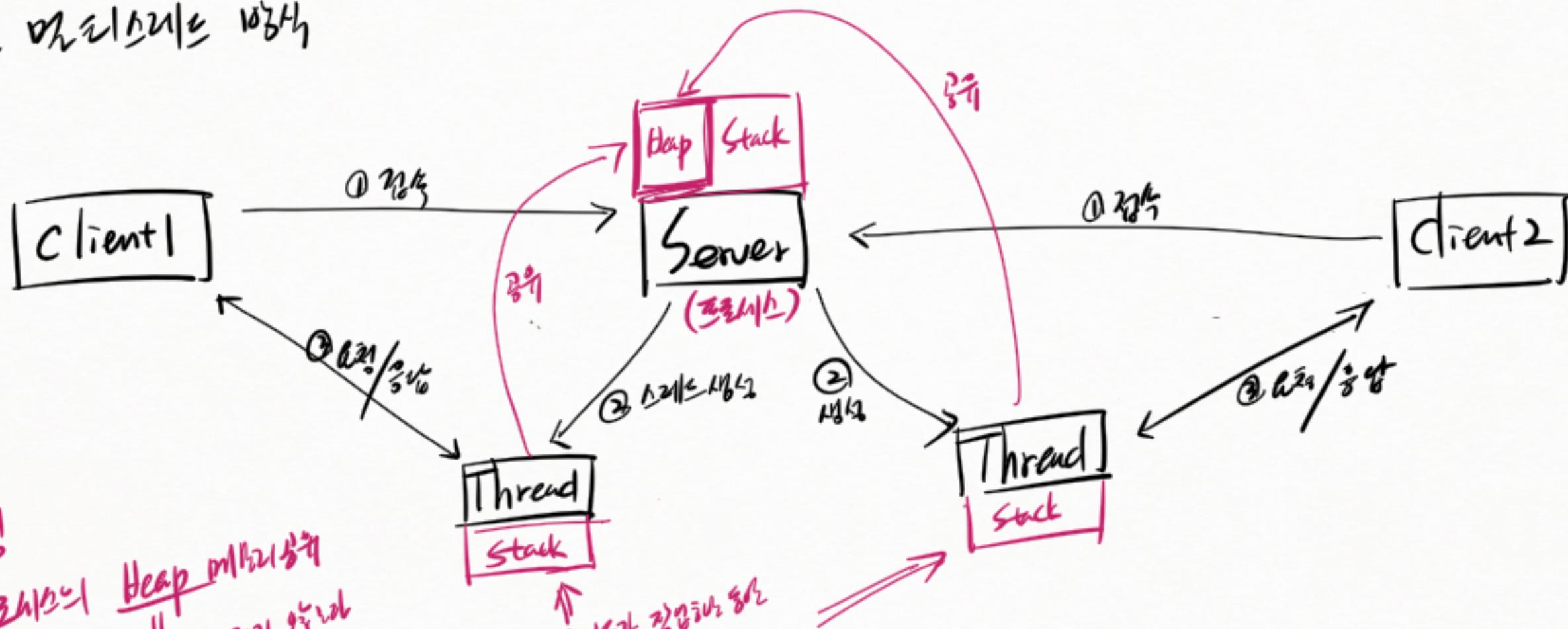
\* 자동 클라이언트 요청 처리에 따라 인터넷 사용

## ① 프로세스 복제 기법 (fork())



\* 다음 클라이언트 요청 처리에 맞춰 구현한 내용

## ② 마리스마스 1084h



\* 320

① 프로세스의 Heap은 멘리에 있습니다.

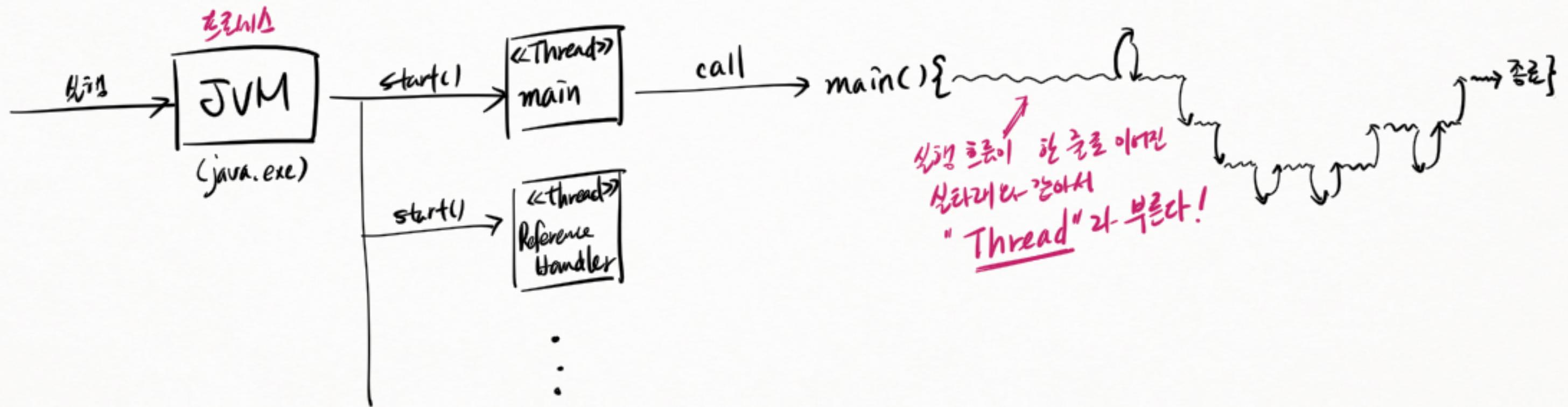
110의 ~~High~~  
110의 ~~High~~ ↓  
110의 중복 사용을 고려한 예상  
↓  
110의 예상

→ 1200

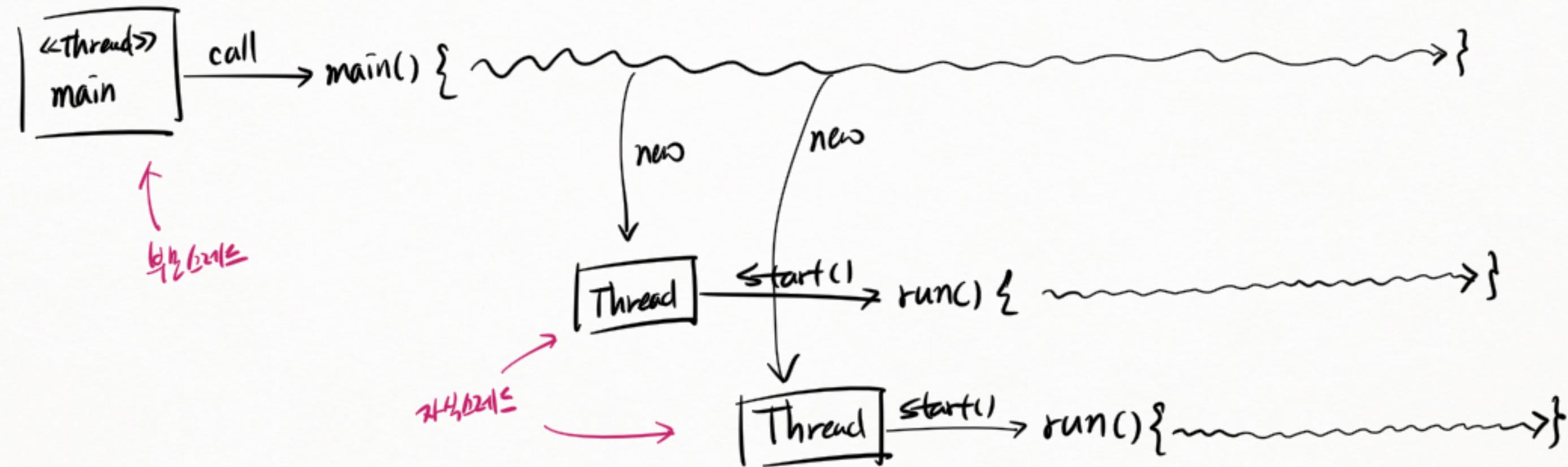
②  $\text{프로토} \xrightarrow{\text{전기화}} \text{네트워크} \xrightarrow{\downarrow} \text{네트워크} \text{설정/구성}$

↑  
수리드가 짹밥이니  
H.C.B. 매콤니

## \* JVM 과 스레드



\* မျှော်လုပ်ခဲ့သူတော်



\* CPU Scheduling