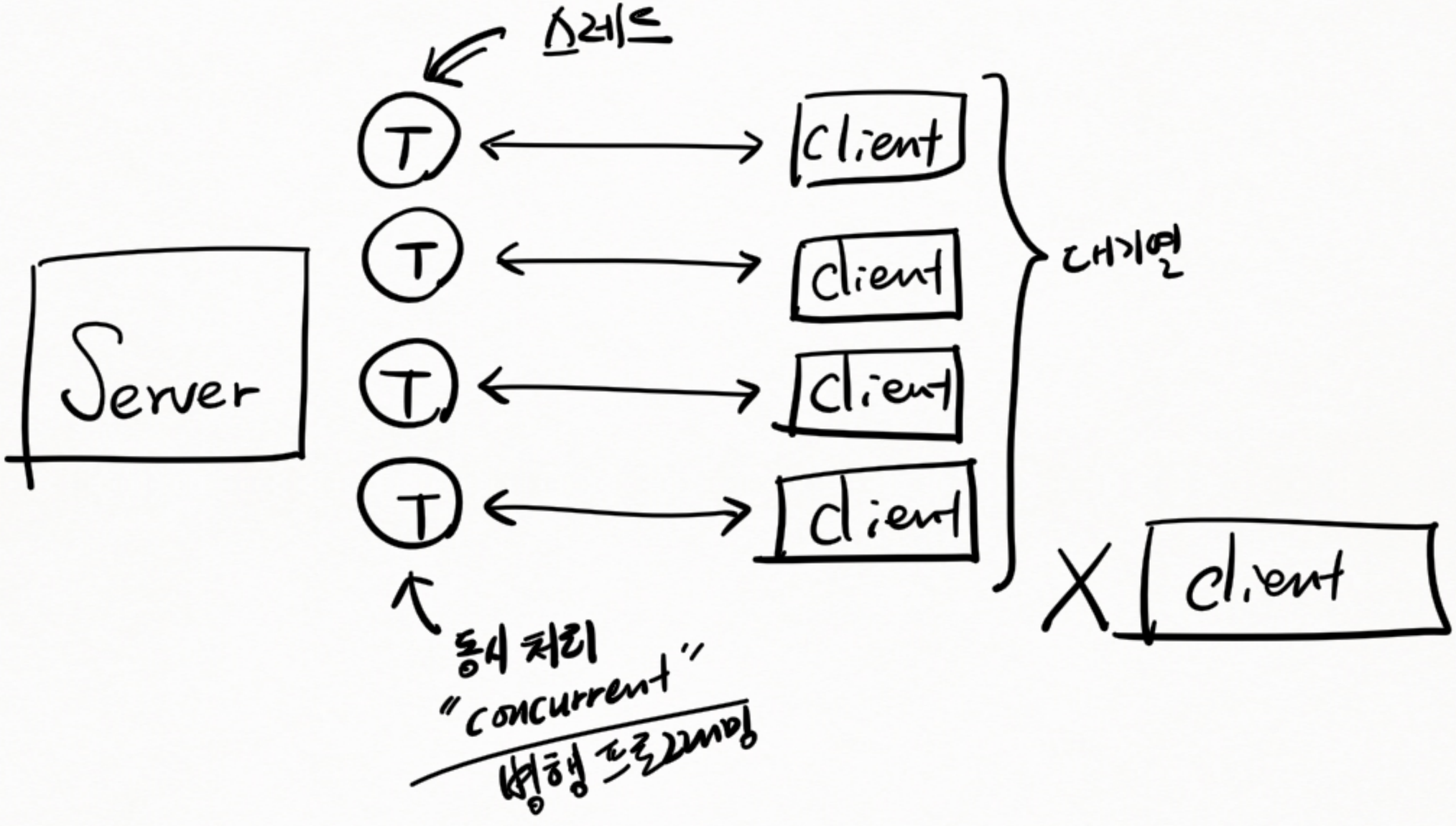

스레드
T
T
T
T
Server
Client
Client
Client
Client
대기열
X
Client
동시 처리
"concurrent"
병행 프로그래밍

멀티태스킹 (multi-tasking)

↳ 동시에 여러작업을 실행

구현방법:

① 멀티 프로세싱 (multi-processing)

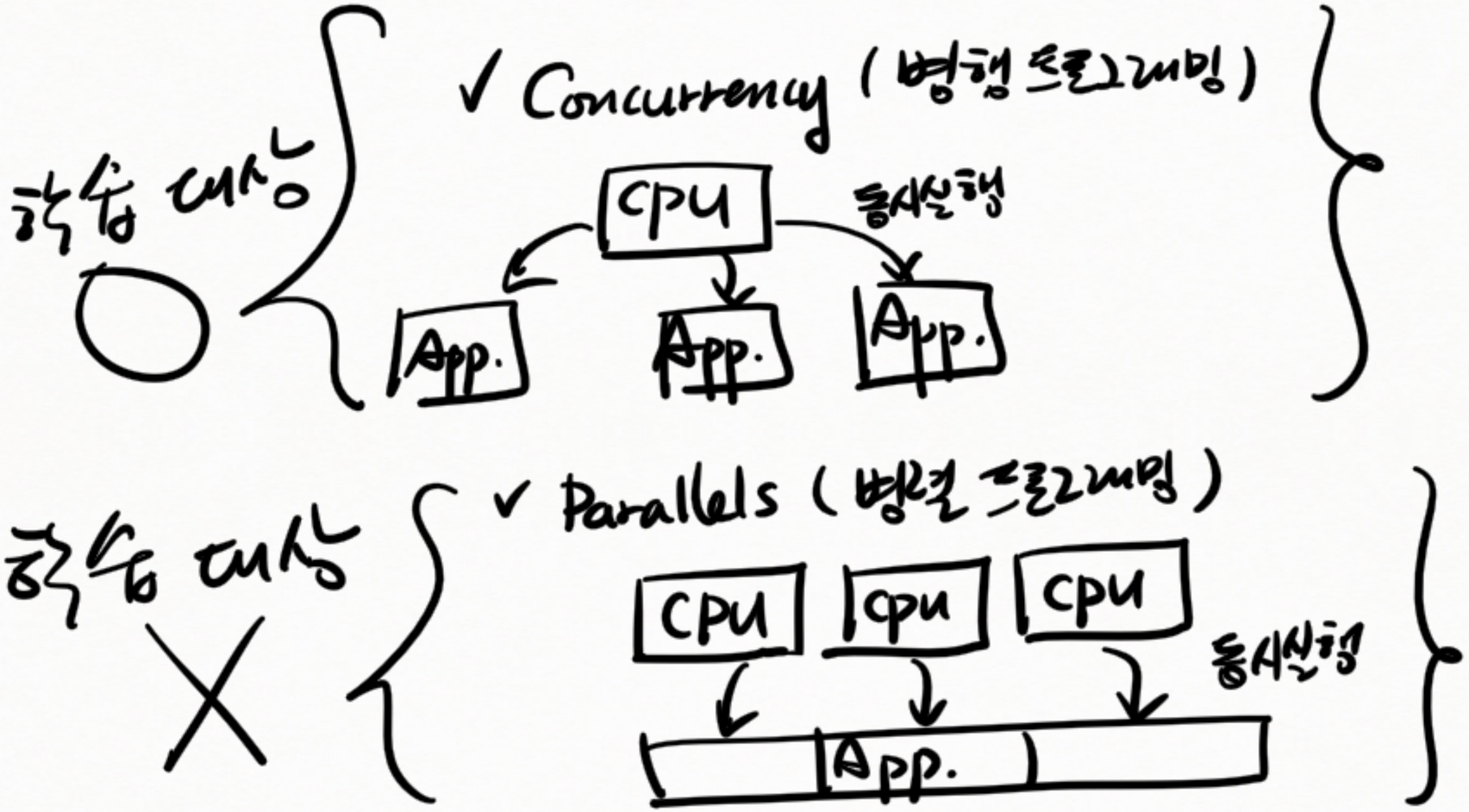
↳ 프로세스를 여러개 복제하여 실행

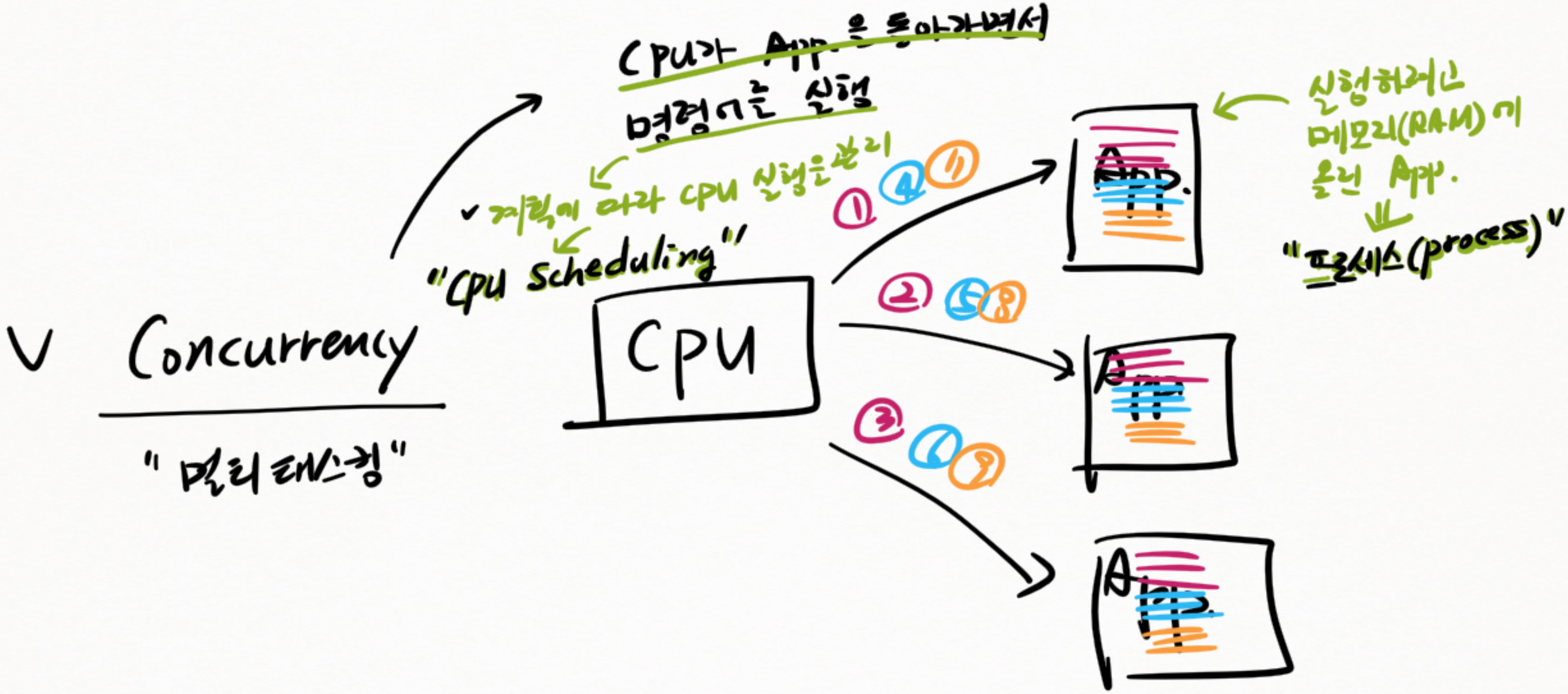
② 멀티 스레딩 (multi-threading)

↳ 프로세스의 작업 중 멀티태스킹이 필요한 작업만 분리하여 여러 개 실행

"멀티 태스킹"

- concurrency
  - ↳ S/W로 병렬
- parallel
  - ↳ H/W 병렬


작업 대상
O
✓ Concurrency (병행 프로그래밍)
CPU
동시실행
App.
App.
App.
작업 대상
X
✓ Parallels (병렬 프로그래밍)
CPU
CPU
CPU
동시실행
App.



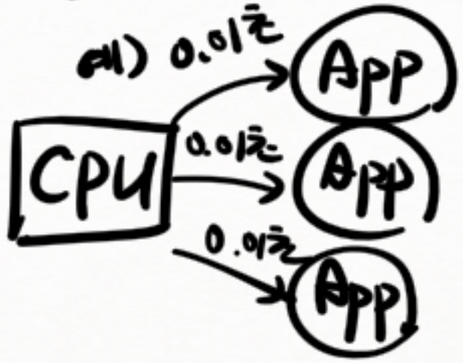
Concurrency

"멀티태스킹"

* CPU Scheduling

↳ 여러 개의 프로세스를 동시에 실행하기 위해
CPU 사용을 관리하는 방법

① Round-Robin
↳ 각 프로세스에 균등한 실행 시간 부여
↳ Windows OS

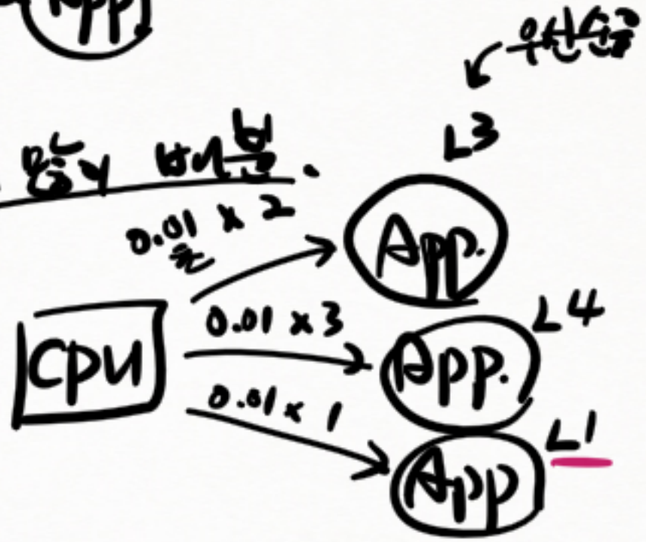


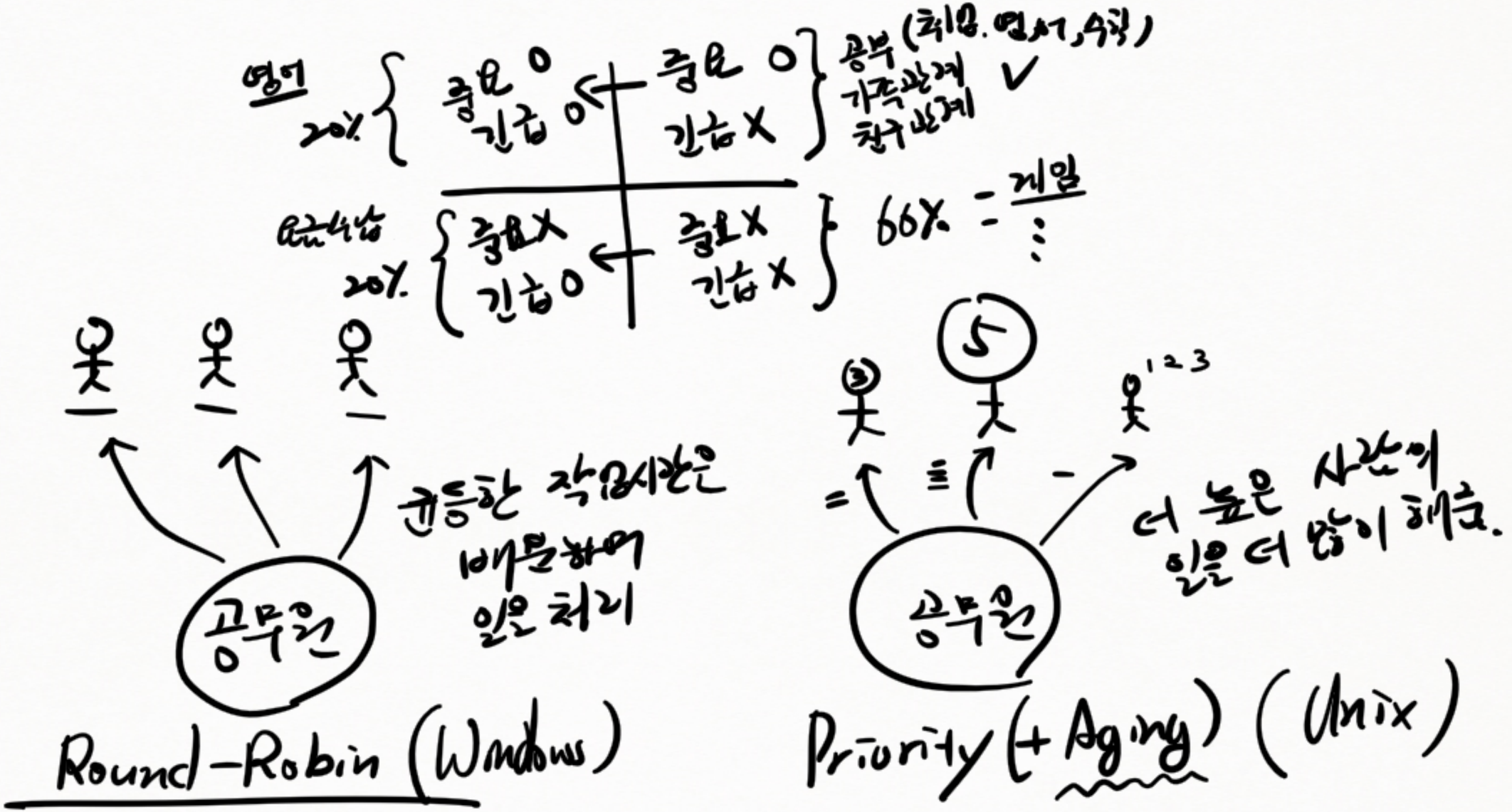
② Priority + "Aging"

↳ 우선 순위가 높은 프로세스에 CPU 사용권한을 더 많이 배분. 더 자주 실행

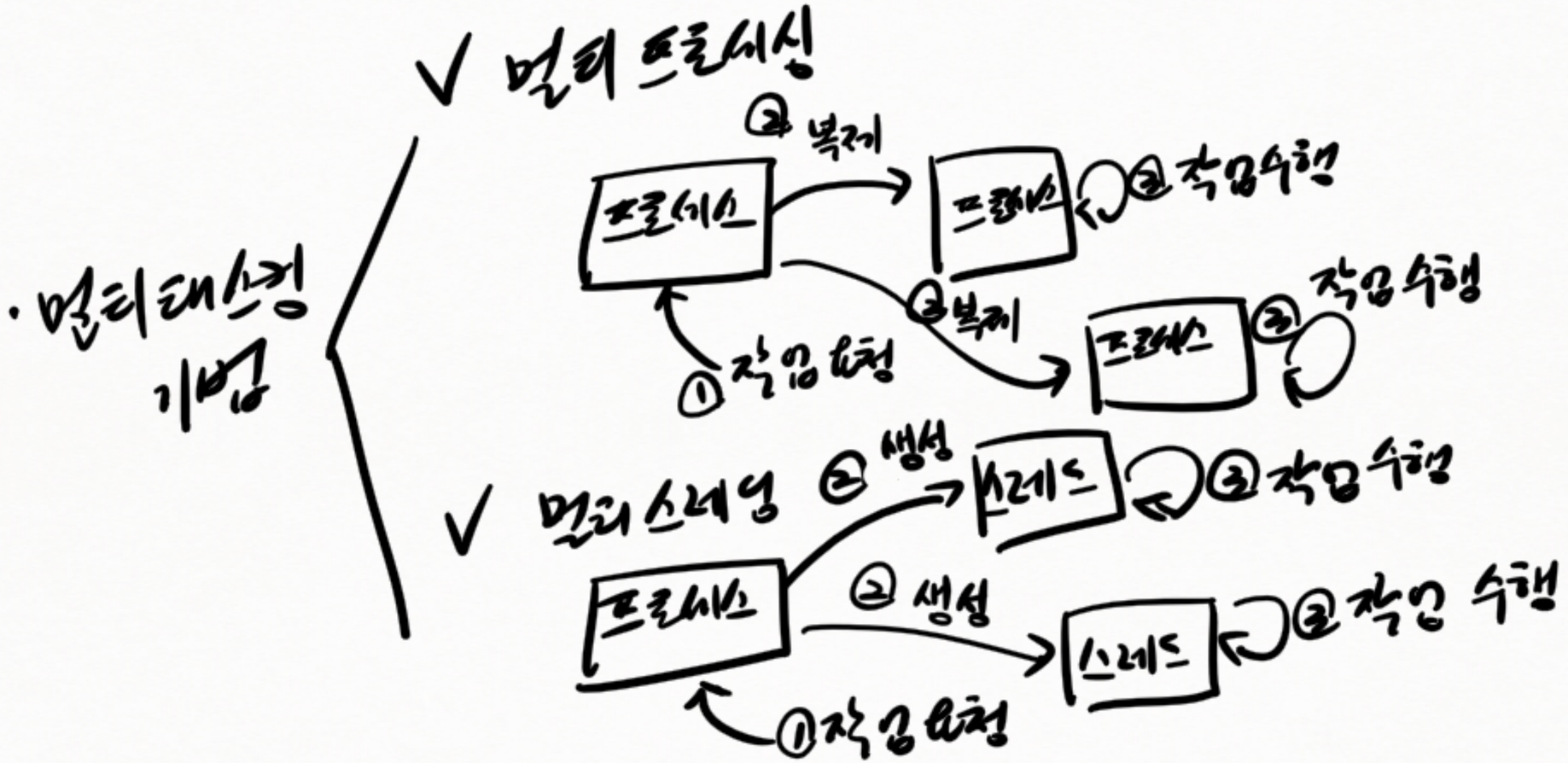
↳ 한 우선 순위가 낮아서 실행이 안될 때마다 우선순위 레벨(age)을 높여서 꼭 실행되게 만든다

↳ Linux, macOS, Unix



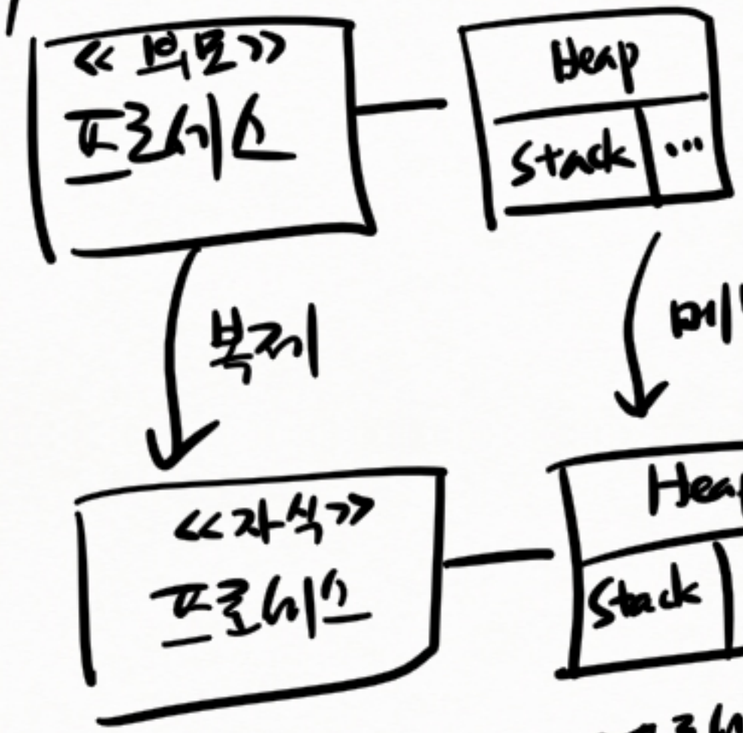

영어
20%
출석 O
긴급 O
출석 O
긴급 X
공부 (취업, 영어, 수학)
가족관계
친구관계
V
Python
20%
출석 X
긴급 O
출석 X
긴급 X
66% = 게임
:
공무원
같은 작업시간을
배분하여
일을 처리
Round-Robin (Windows)
5
1 2 3
공무원
더 높은 사람의
일을 더 많이 해줌.
Priority (+ Aging) (Unix)

· "멀티태스킹" → 한정된 자원(CPU)을 사용하여
"CPU 스케줄링 알고리즘"에 따라
(문제해결방법)
여러 개의 프로세스를 동시에 실행하는 것!


· 멀티태스킹 기법
✓ 멀티프로세싱
② 복제
프로세스
프로세스
③ 작업수행
① 작업요청
② 복제
프로세스
③ 작업수행
✓ 멀티스레딩
② 생성
스레드
③ 작업수행
프로세스
② 생성
스레드
③ 작업 수행
① 작업요청

멀티 프로세싱

"예전에 멀티 테스킹 구현 방식



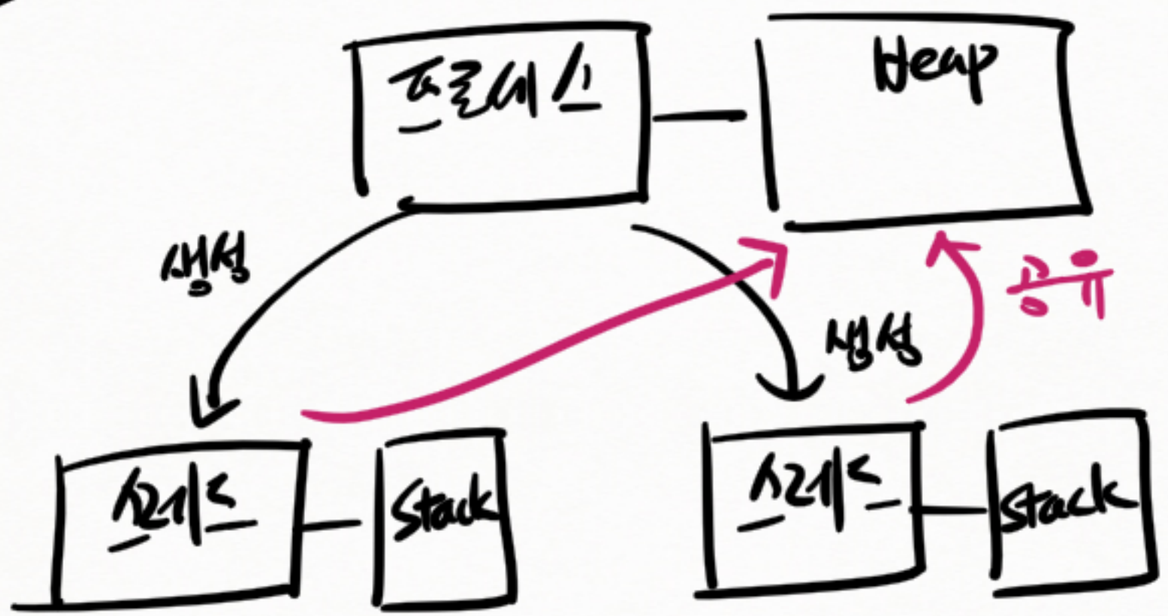
① 순식간 복제(fork)
↓
구현이 쉽다.
(근당)

② 메모리를 그대로 복제
↓
메모리 낭비가 심하다

③ 자식 프로세스로 부모의 중속되지 않는다.

부모프로세스 종료
↓
자식은 안죽음

"현재의 멀티태스킹 구현 방식" → 멀티 스레딩

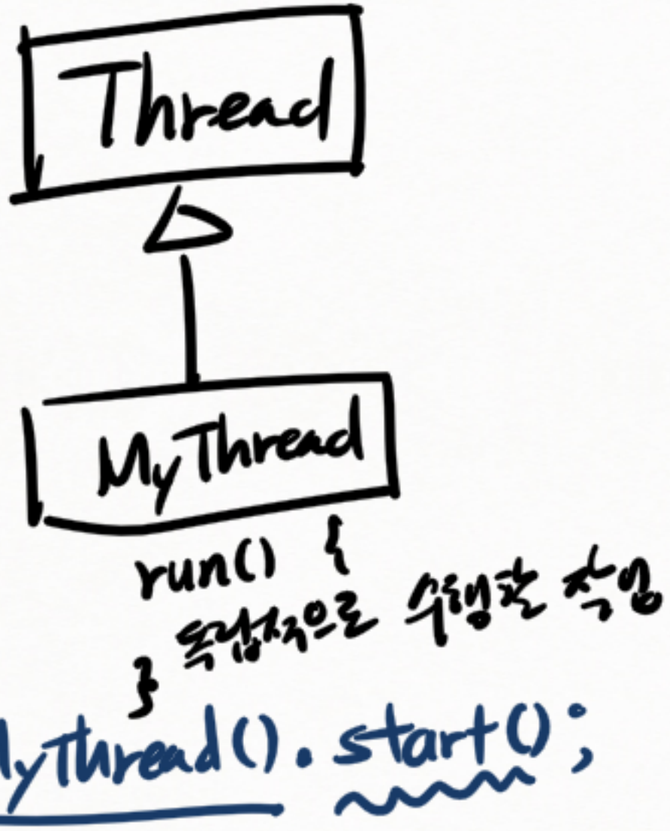


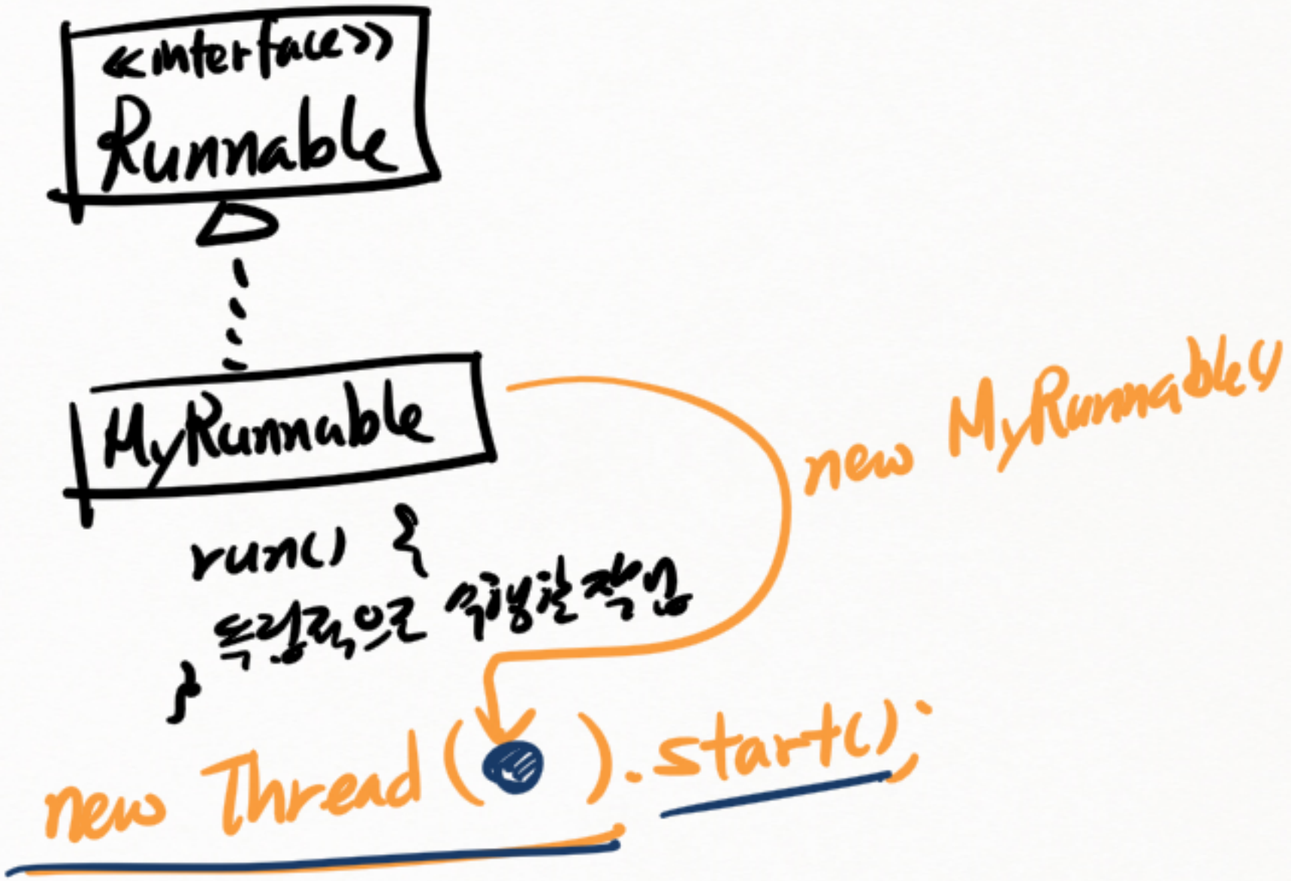
① 스레드 생성
↓
구현이 조금더 복잡

② 프로세스의 Heap은 공유
↓↓
메모리 절약

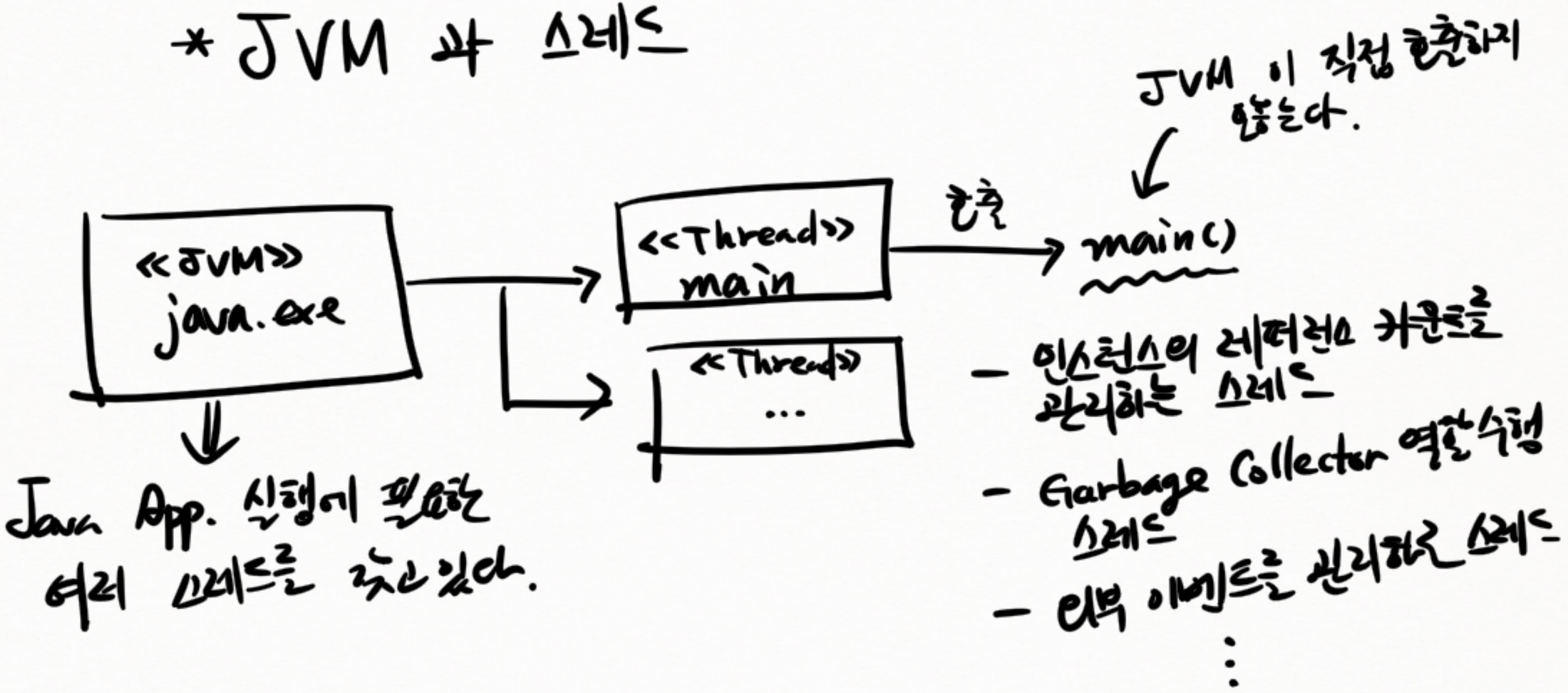
③ 스레드 프로세스에 종속
↙
프로세스 종료 → 스레드 종료

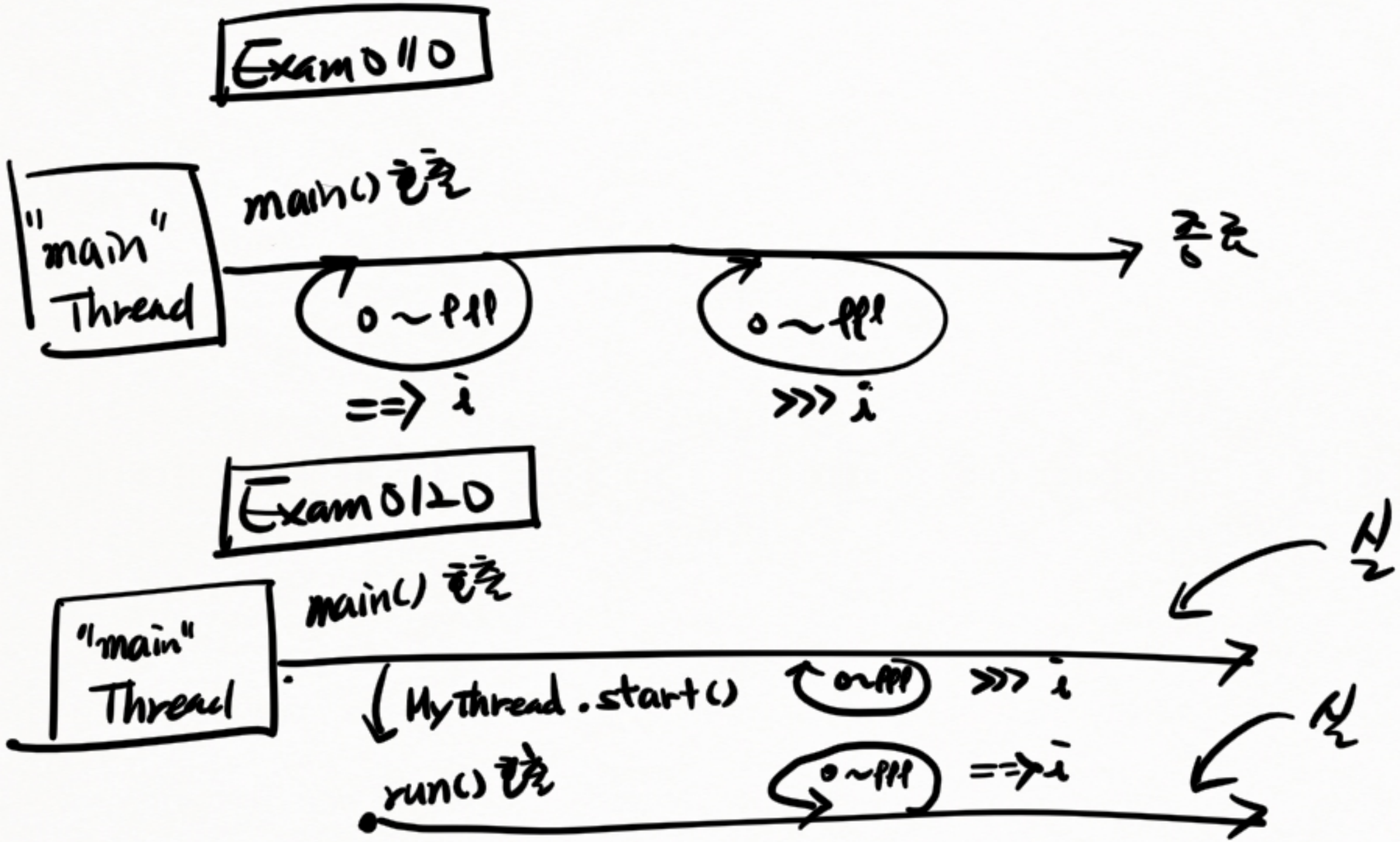
* Java 스레드 구현


Thread
MyThread
run() {
독립적으로 수행할 작업
}
new MyThread().start();


«interface»
Runnable
MyRunnable
run() {
독립적으로 수행할 작업
}
new MyRunnable()
new Thread( ).start();

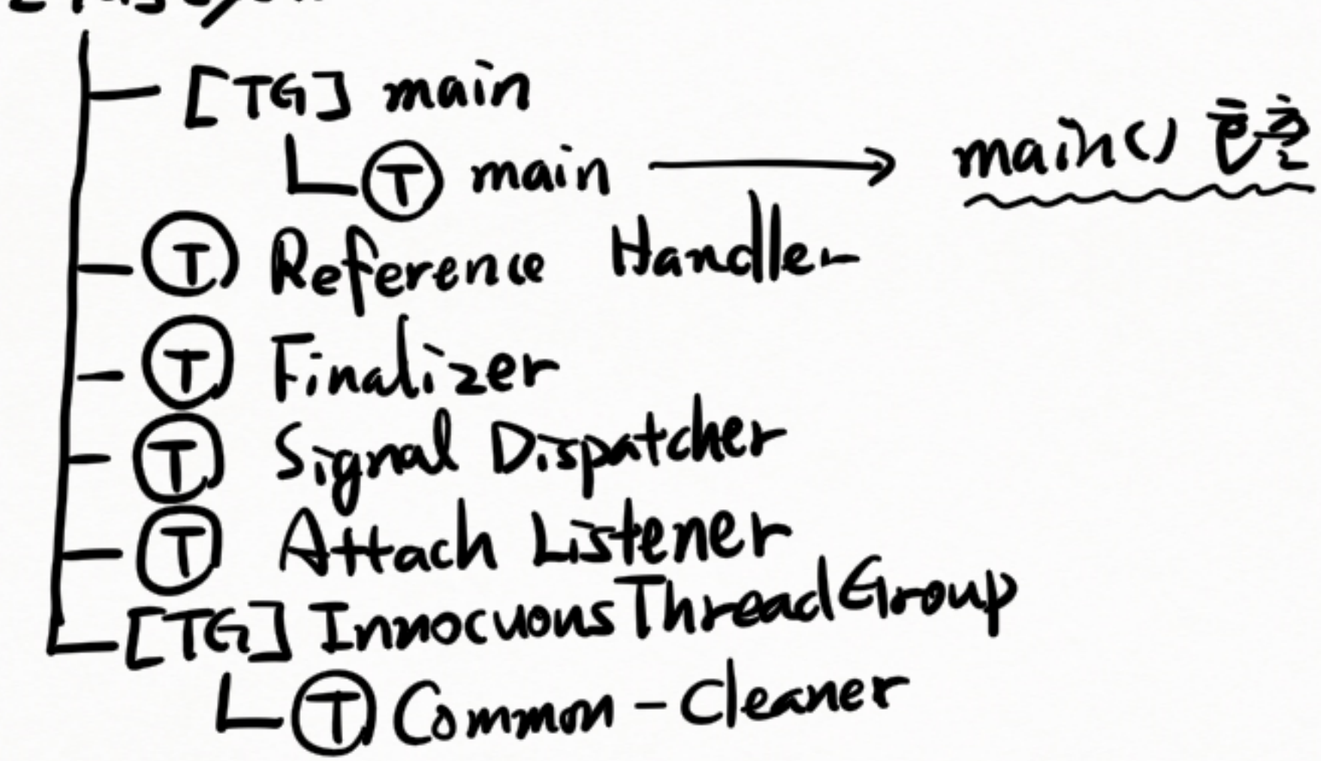
* JVM 과 스레드


JVM 이 직접 호출하지 않는다.
«JVM»
java.exe
«Thread»
main
«Thread»
...
호출
main()
Java App. 실행에 필요한 여러 스레드를 갖고 있다.
- 인스턴스의 레퍼런스 카운트를 관리하는 스레드
- Garbage Collector 역할수행 스레드
- 이벤트를 관리하는 스레드
⋮

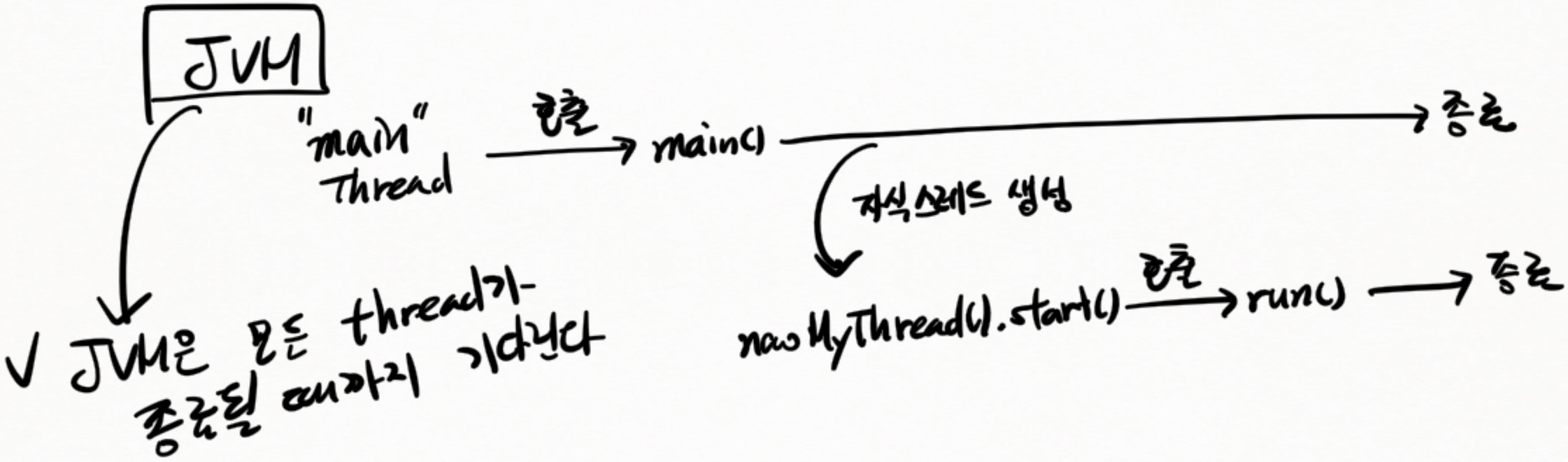

Exam0110
"main" Thread
main() 호출
0 ~ 999
==> i
0 ~ 999
>>> i
종료
Exam0120
"main" Thread
main() 호출
MyThread.start()
0~999
>>> i
run() 호출
0~999
==> i
N
N

JVM Thread 계층

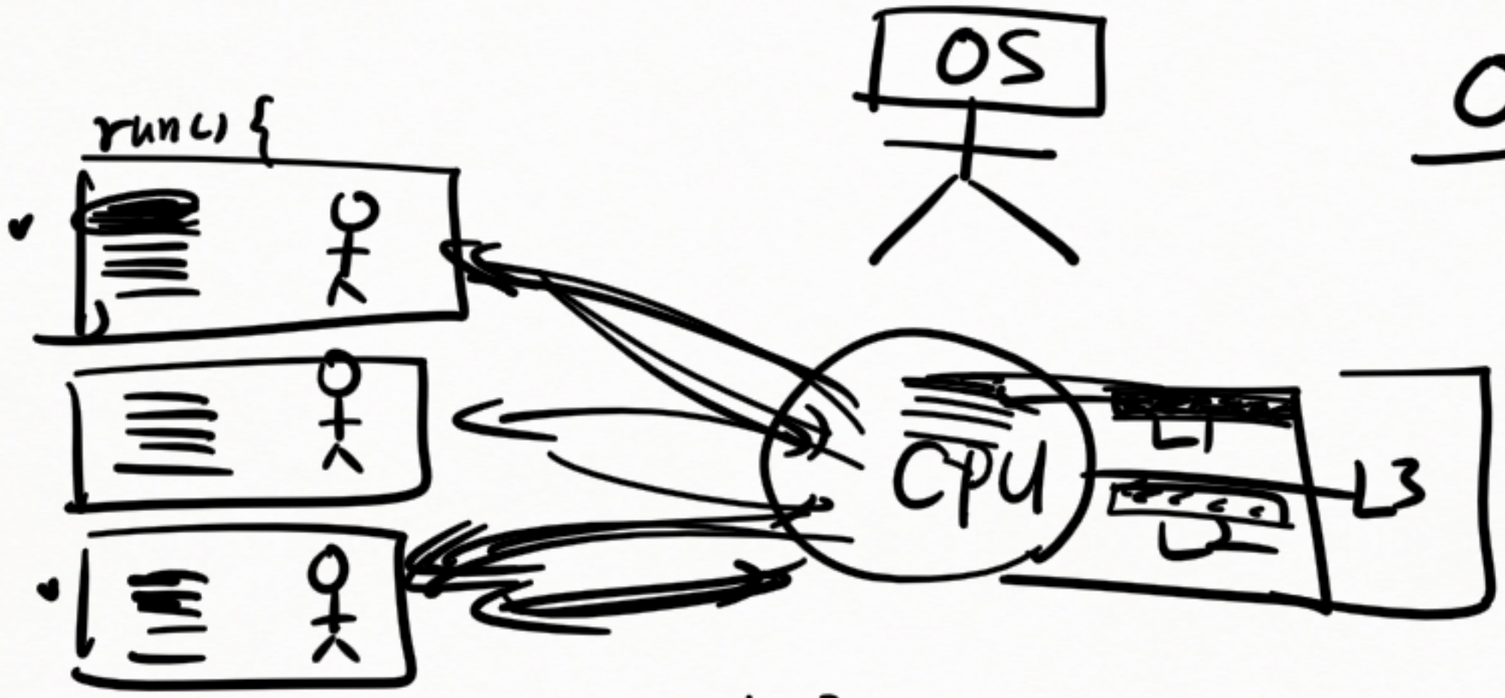
JVM 프로세스



프로세스와 스레드의 실행 흐름



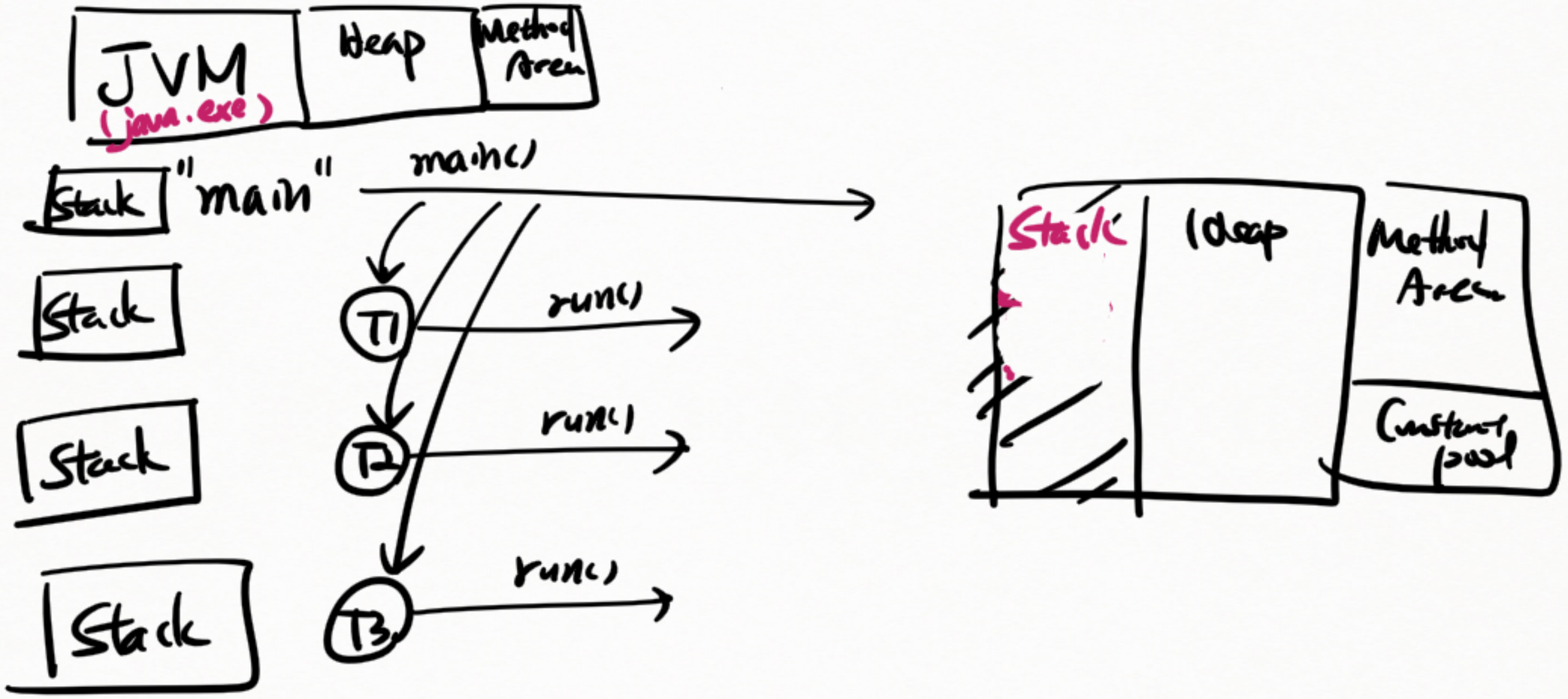
Thread.join()



의인화

비선점형 OS ← DOS + Windows 3.x

선점형 OS ← Windows 9x -

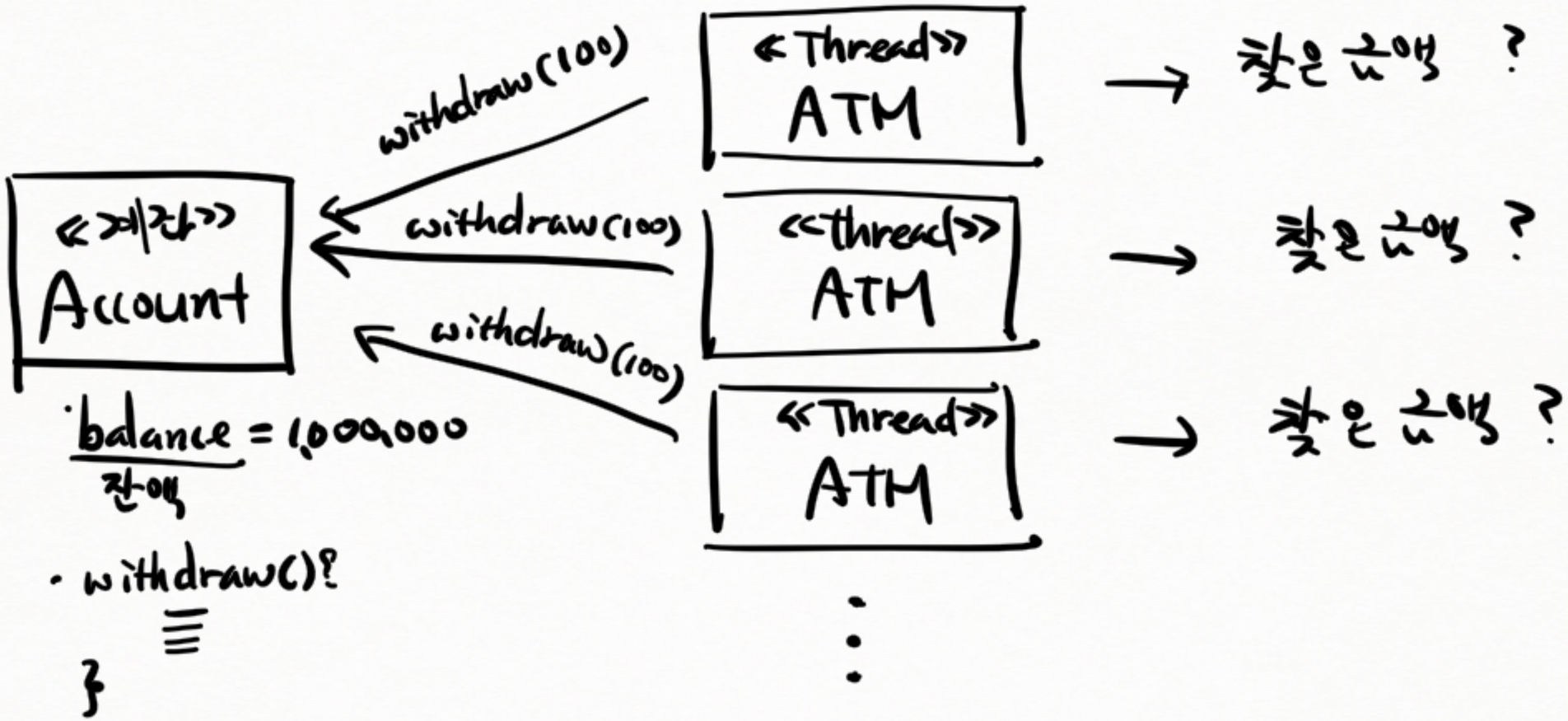

JVM
(java.exe)
Heap
Method Area
Stack
"main"
main()
Stack
T1
run()
Stack
T2
run()
Stack
T3
run()
Stack
Heap
Method Area
Constant pool

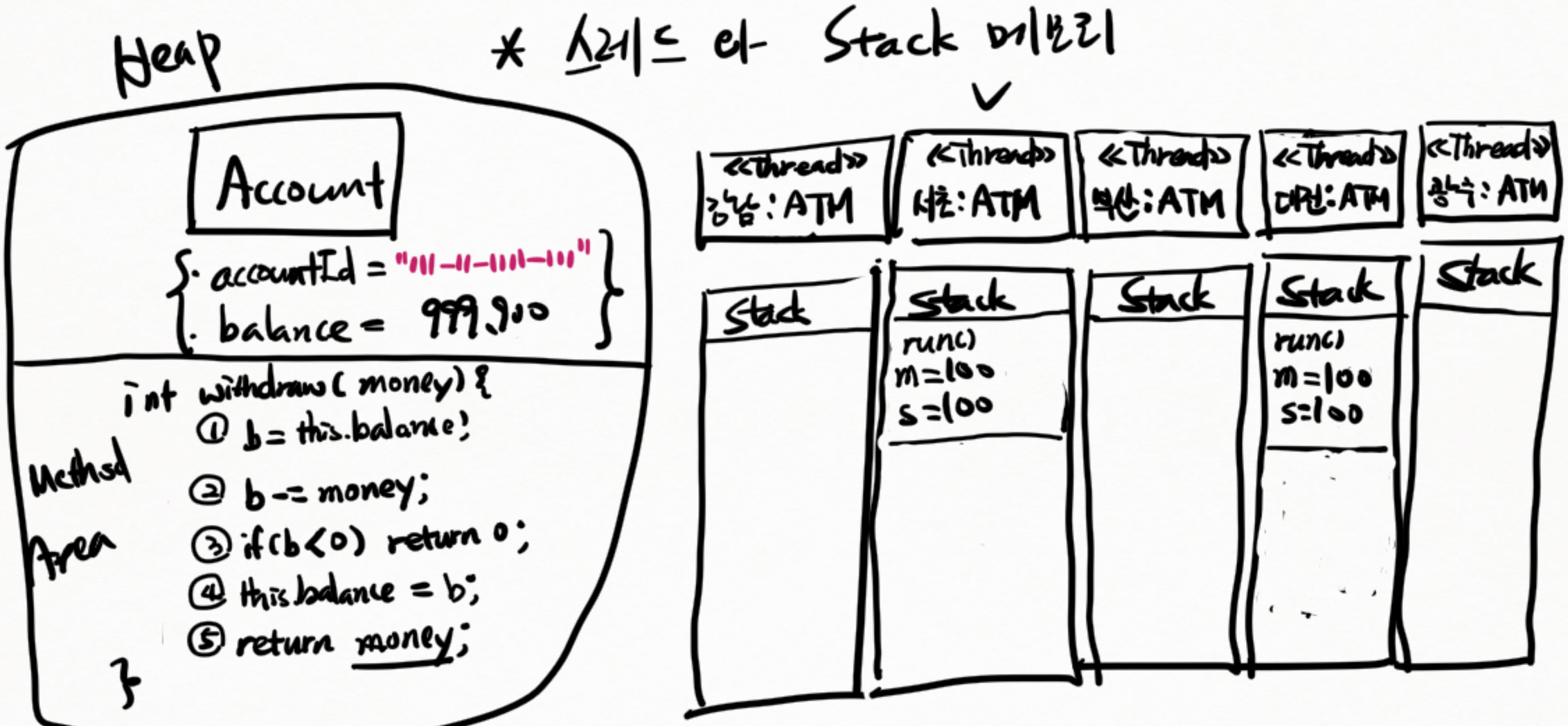
"Context Switching"

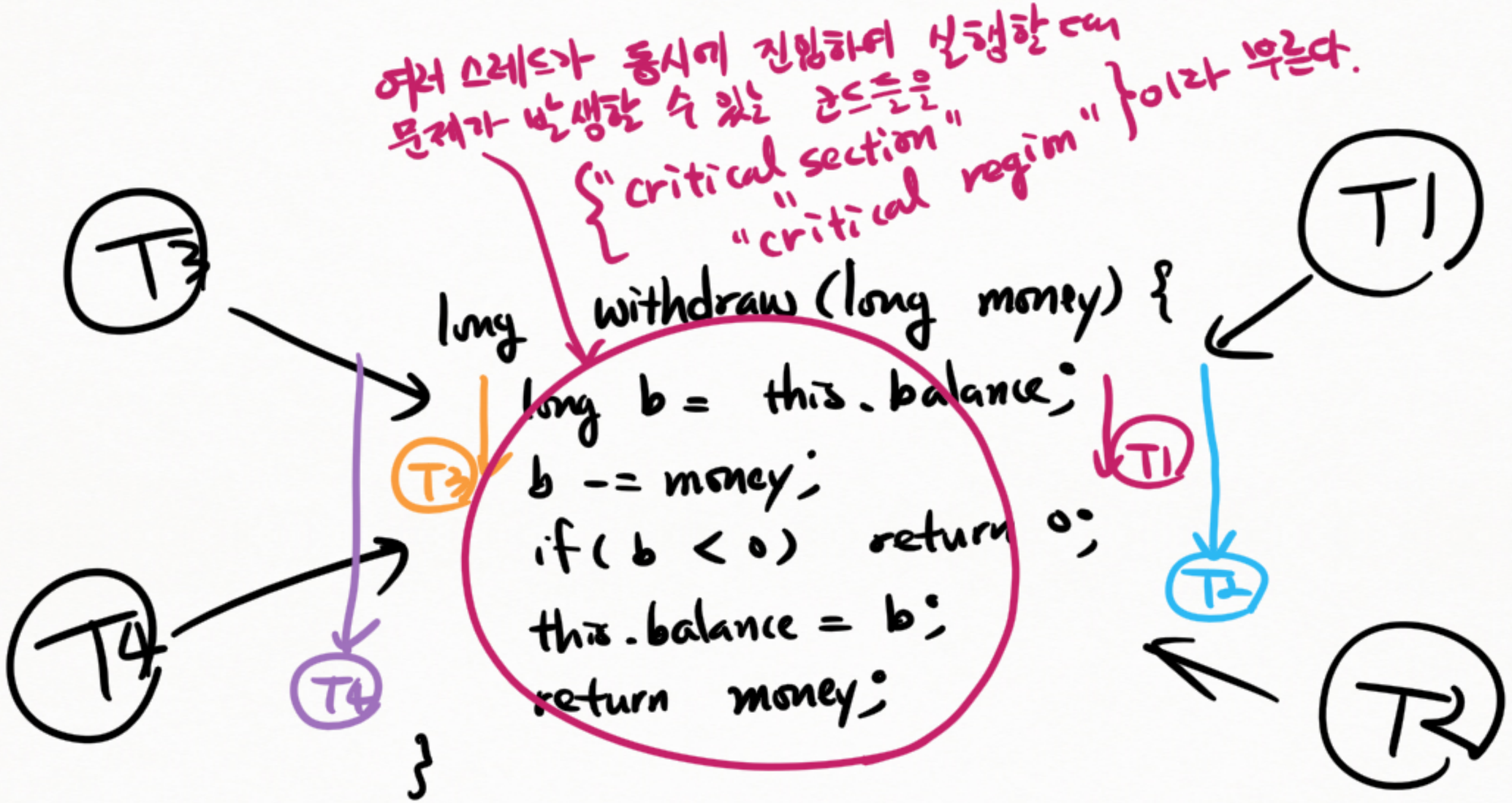
컨텍스트 상황 문맥

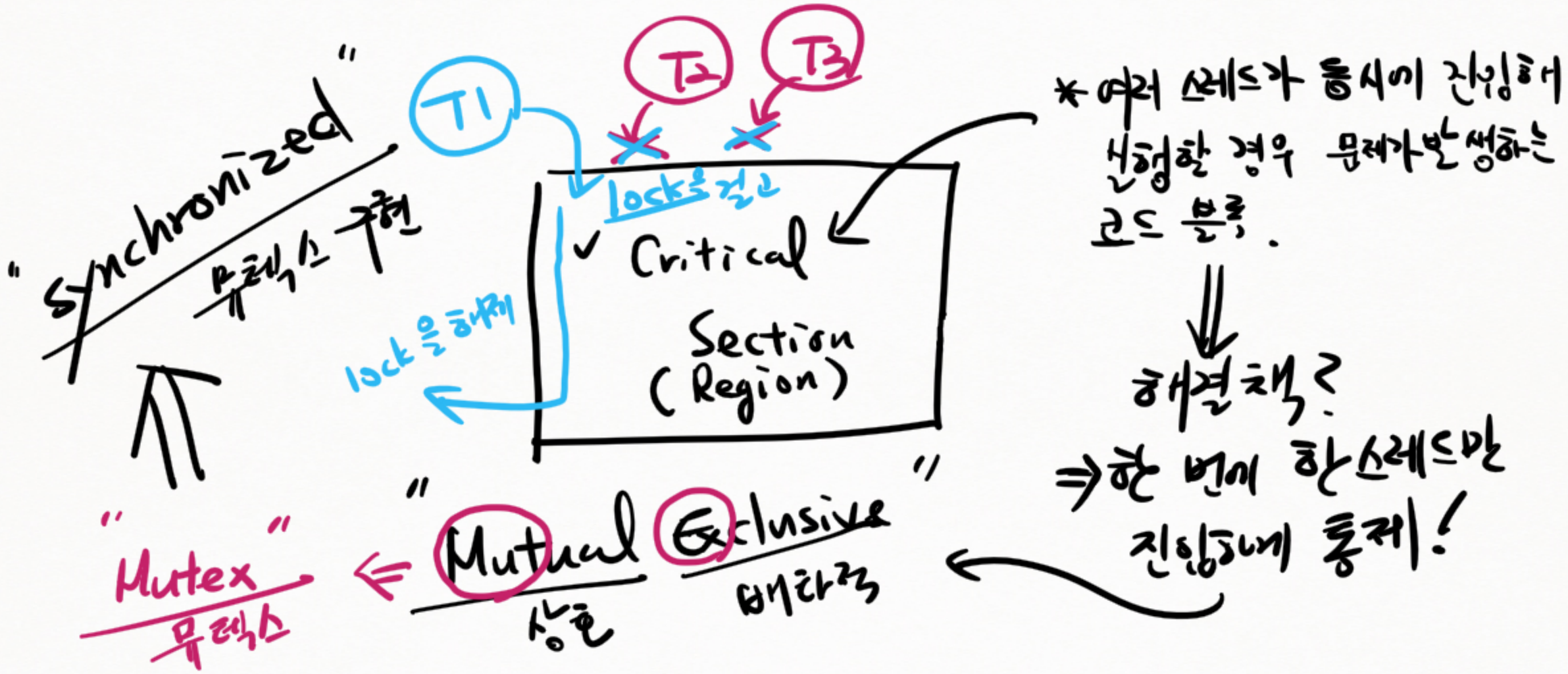
↳ 프로세스나 스레드가 실행되기 위해 실행정보를 읽어내고 저장하는 것.

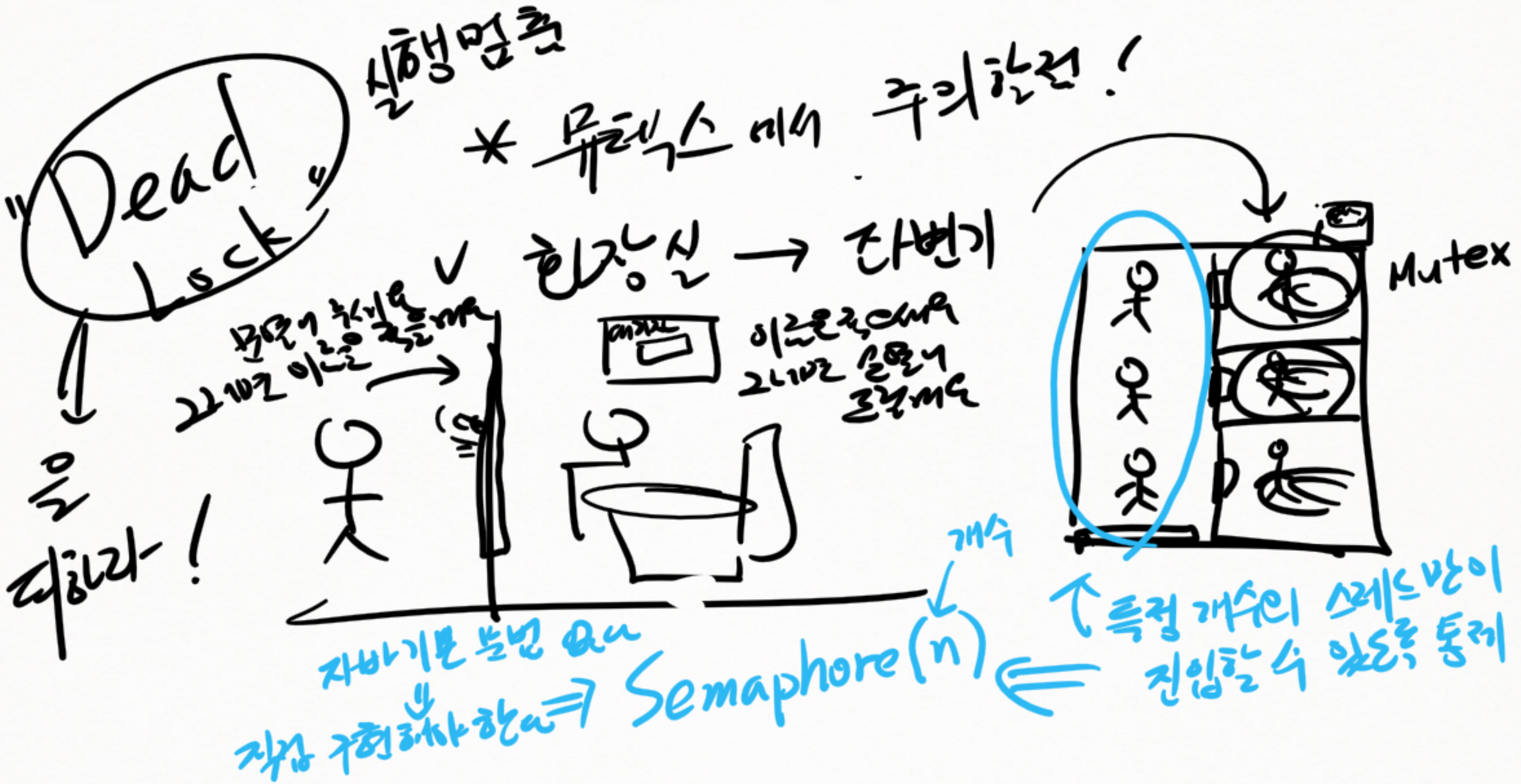
* 비동기 실행의 문제점


«Thread»
ATM
withdraw(100)
«계좌»
Account
withdraw(100)
«thread»
ATM
withdraw(100)
«Thread»
ATM
찾은 금액 ?
찾은 금액 ?
찾은 금액 ?
balance = 1,000,000
잔액
withdraw()?
}


Heap
* 스레드 별 Stack 메모리
Account
accountId = "111-11-1111-111"
balance = 999,900
int withdraw(money){
① b = this.balance;
② b -= money;
③ if(b<0) return 0;
④ this.balance = b;
⑤ return money;
}
Method Area
<<Thread>> 강남:ATM
<<Thread>> 서초:ATM
<<Thread>> 역삼:ATM
<<Thread>> 대전:ATM
<<Thread>> 광주:ATM
Stack
run()
m=100
s=100


여러 스레드가 동시에 진입하여 실행할 때
문제가 발생할 수 있는 코드들을
{"critical section"
"critical regim"}이라 부른다.
T3
T1
T4
T2
T4
long withdraw(long money) {
long b = this.balance;
b -= money;
if(b < 0) return 0;
this.balance = b;
return money;
}


T1
T2
T3
lock을 걸고
lock을 해제
Critical
Section
(Region)
"synchronized"
무텍스 구현
"Mutex"
무텍스
Mutual
상호
Exclusive
배타적
* 여러 스레드가 동시에 진입해
선형할 경우 문제가 발생하는
코드 블록.
해결책?
⇒ 한 번에 한 스레드만
진입하게 통제!


"Dead Lock"
을 피하라!
실행명령
* 뮤텍스에서 주의할점!
화장실 → 공중변기
Mutex
개수
Semaphore(n)
특정 개수의 스레드만이 진입할 수 있도록 통제

Semaphore(n) → 최대 n 개의 스레드 진입 가능

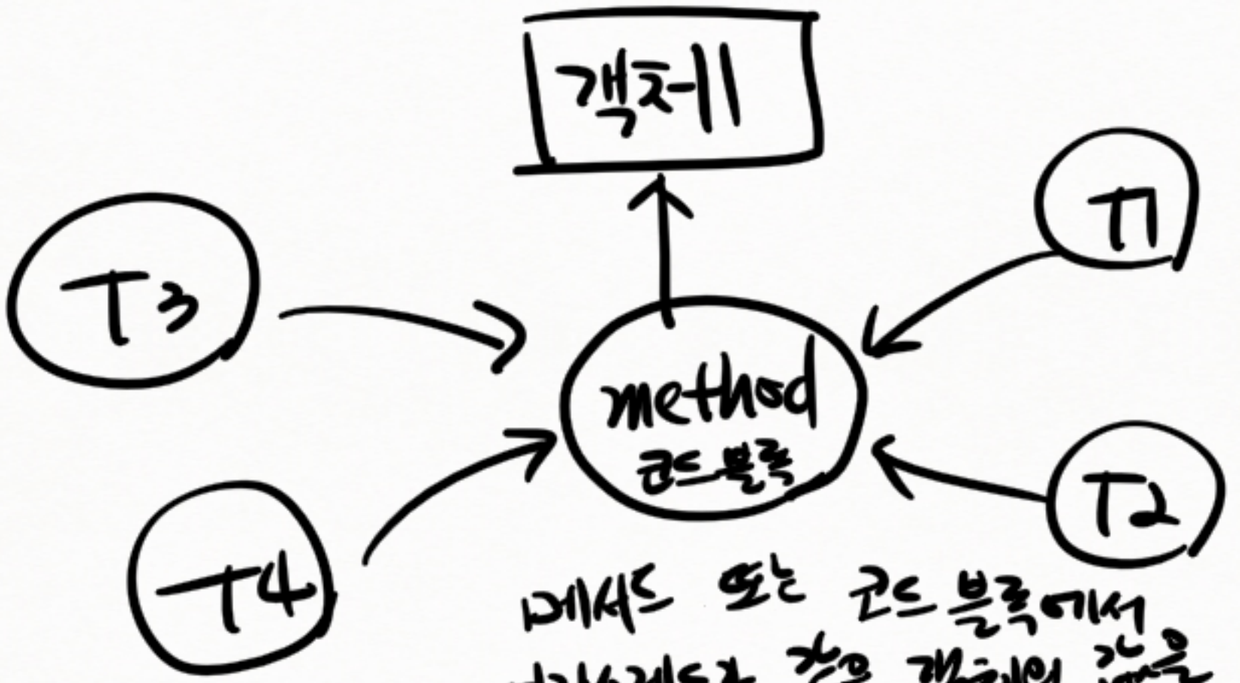
Semaphore(1) → 최대 1 개의 스레드만 진입가능

=

"Mutex"

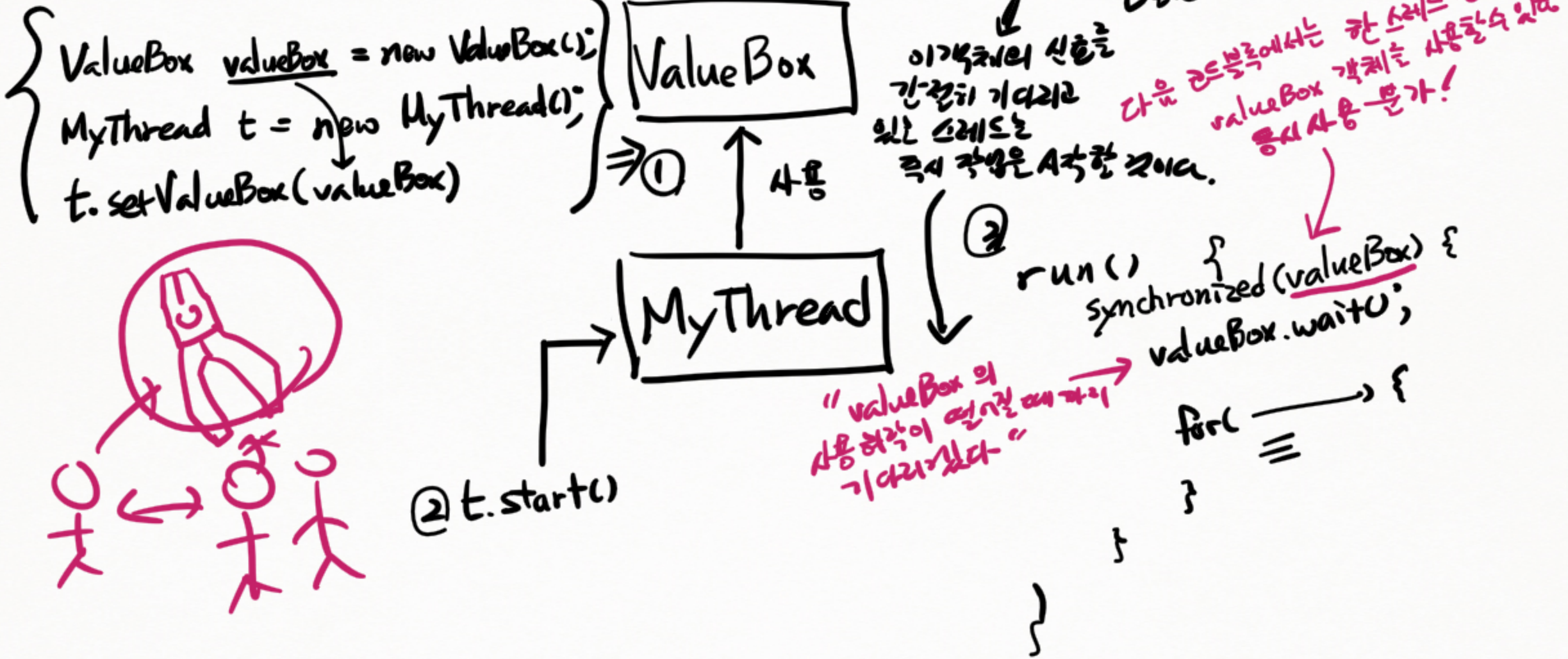
* 비동기 실행

즉 Mutex 상태로 만든다.



메서드 또는 코드블록에서
여러스레드가 같은 객체의 값을 동시에 변경하려 할때
문제 발생

해당 메서드나 코드블록을
synchronized로
동기화 처리하면 된다

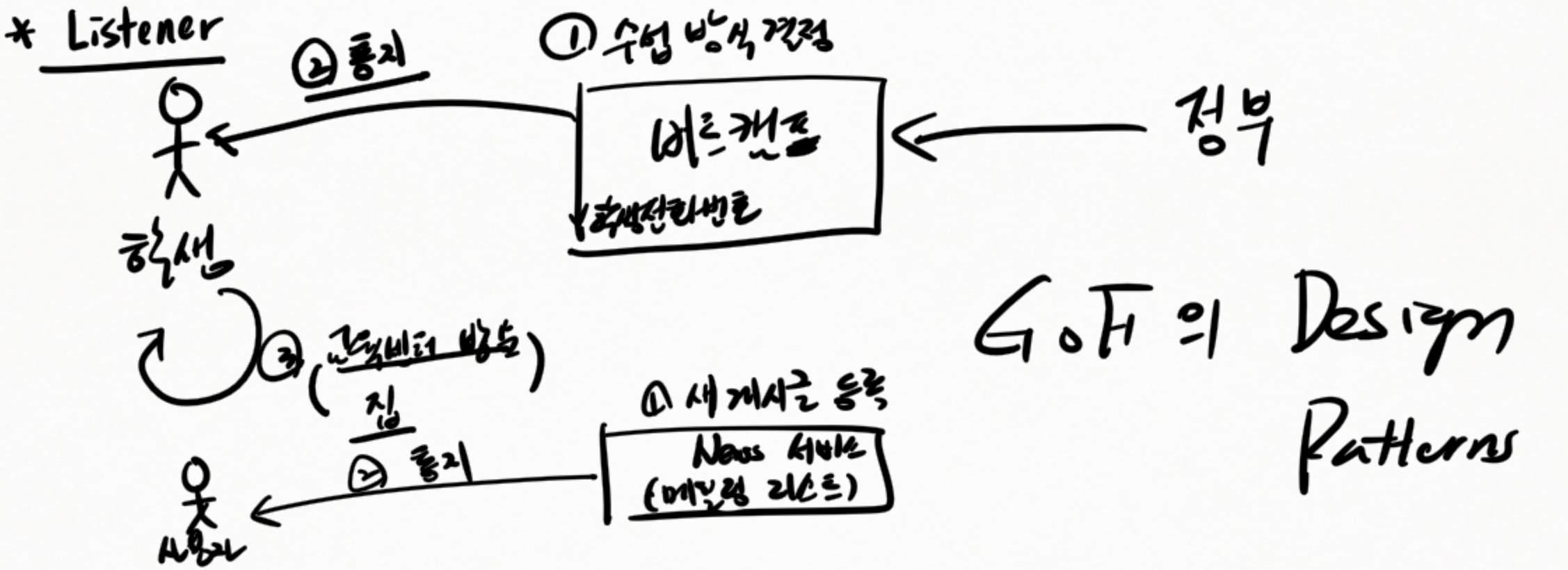

ValueBox valueBox = new ValueBox();
MyThread t = new MyThread();
t.setValueBox(valueBox)
ValueBox
①
사용
MyThread
② t.start()
④ main 스레드
set(3int()) 호출
이 객체에 대해 notify()를
호출한 CL
이 객체의 신호를
간절히 기다리고
있는 스레드를
즉시 깨워준다.
③
다음 코드블록에서는 한 스레드만이
valueBox 객체를 사용할수 있도
록 사용 ~~한~~가!
run() {
synchronized (valueBox) {
valueBox.wait();
for
}
}
}
"valueBox의
사용허락이 떨어질 때까지
기다리겠다"

(관찰자) = (수신자) Listener

* Observer 패턴

↳ 특정 객체의 상태 변화에 따라 작업을 수행하고 싶을 때

✓ "Observer"
=
* Listener



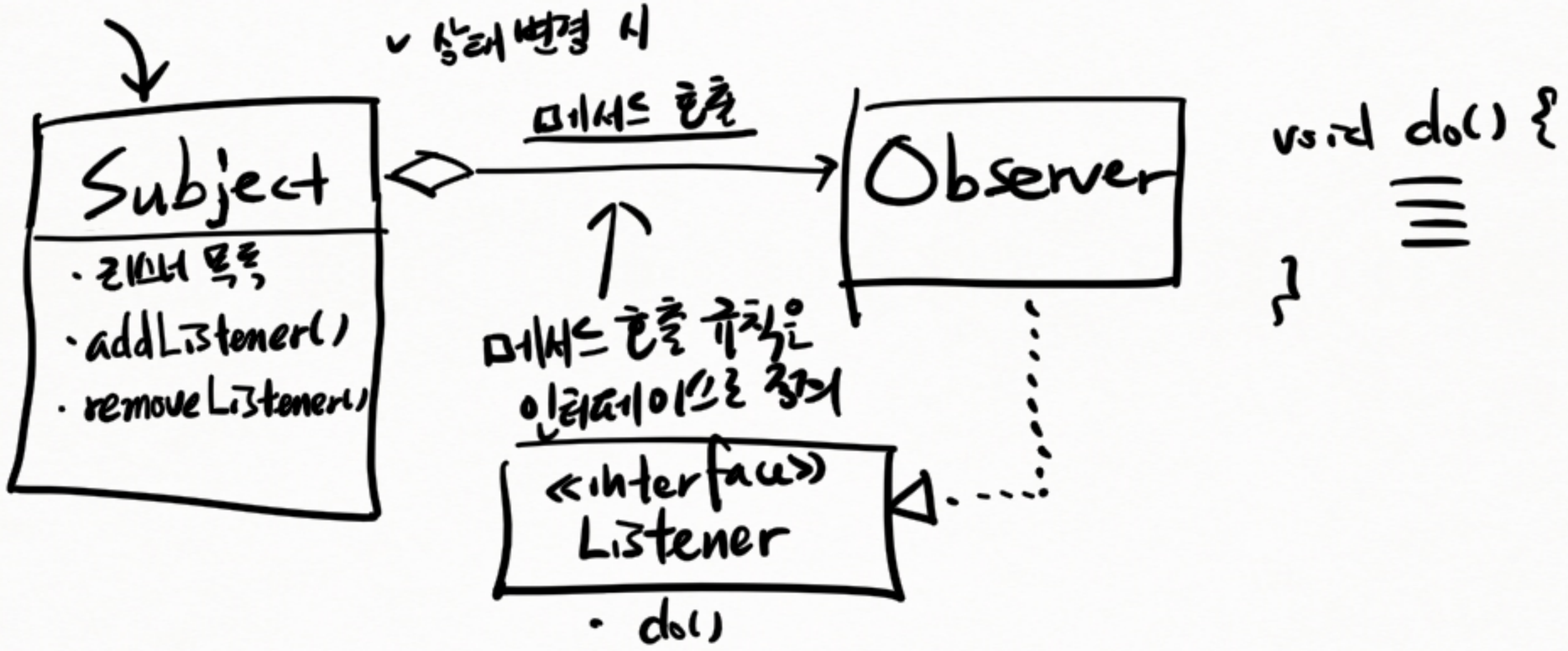
GoF의 Design Patterns

Listener = publish / subscribe
(등록/발행) (수신/구독)

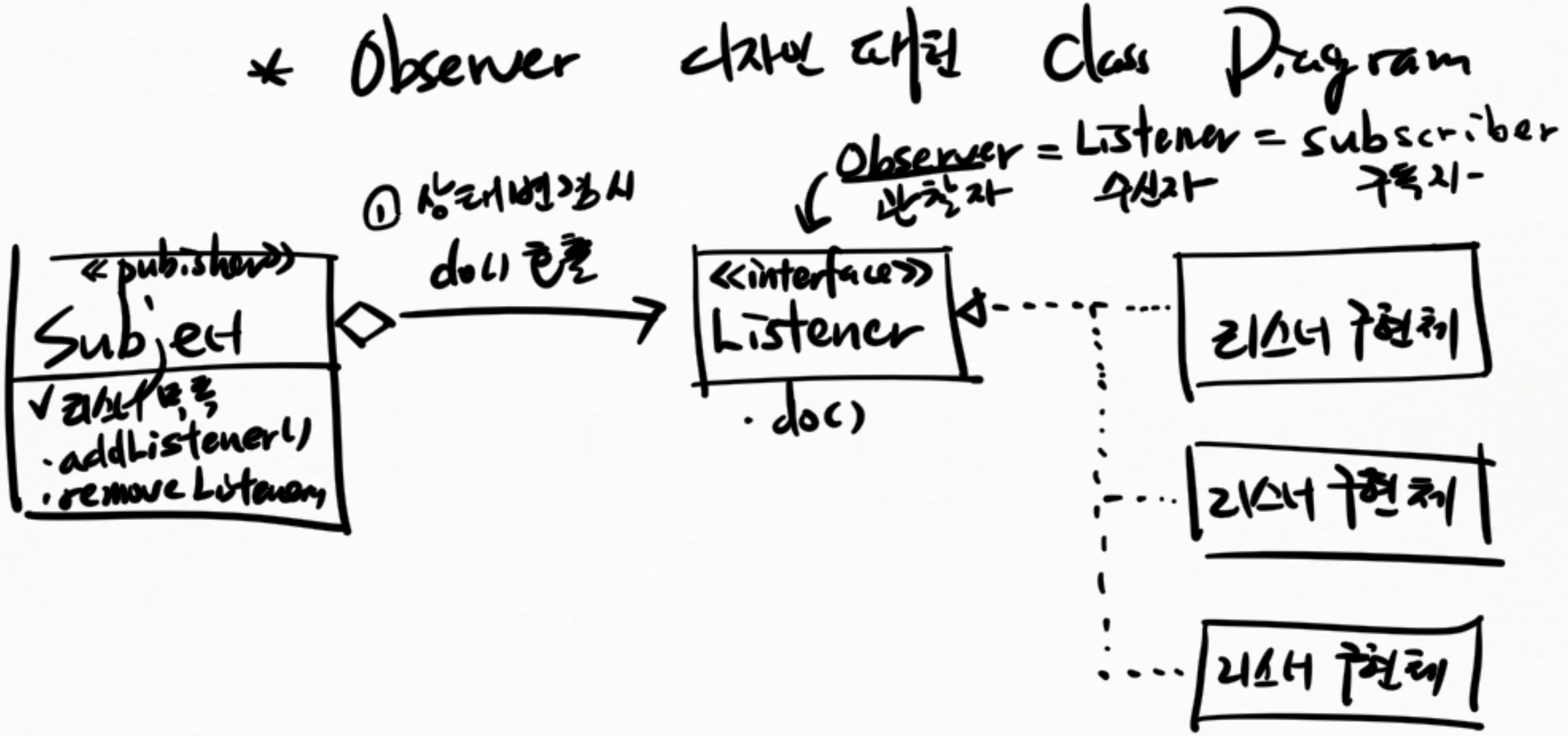
v 이벤트를 받을 서버를 구현
(publisher)

* Observer 패턴의 Class Diagram

v 상태변경 시



* Observer 디자인 패턴 Class Diagram



* Car 클래스에 Observer 패턴 적용 전!

① v1.0 → 배포

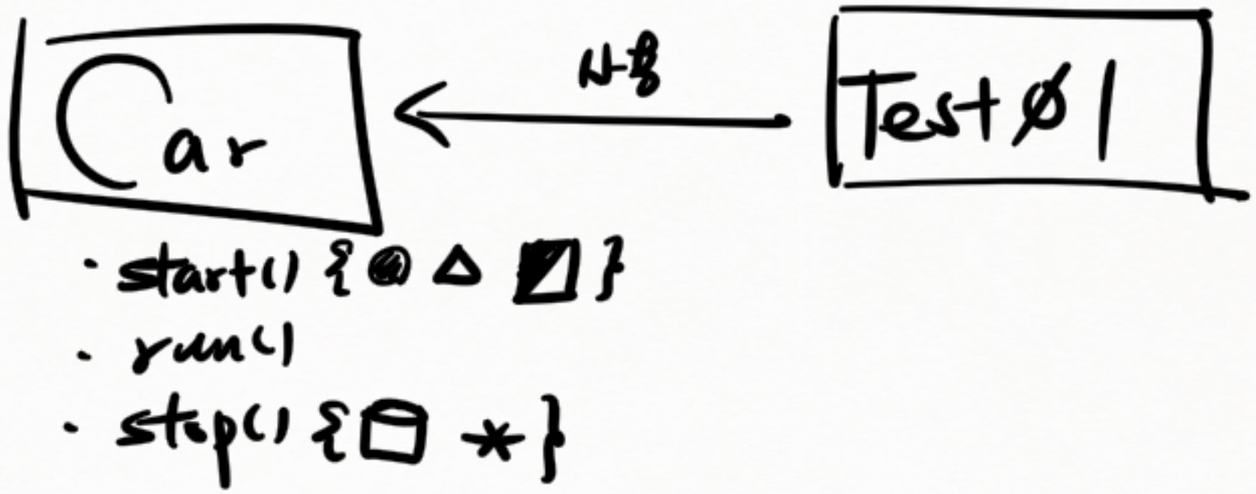
② v1.1 → 안전벨트 착용전사 ●

③ v1.2 → 엔진On 전사 △

④ v1.3 → 변데이터On 전사 ◪

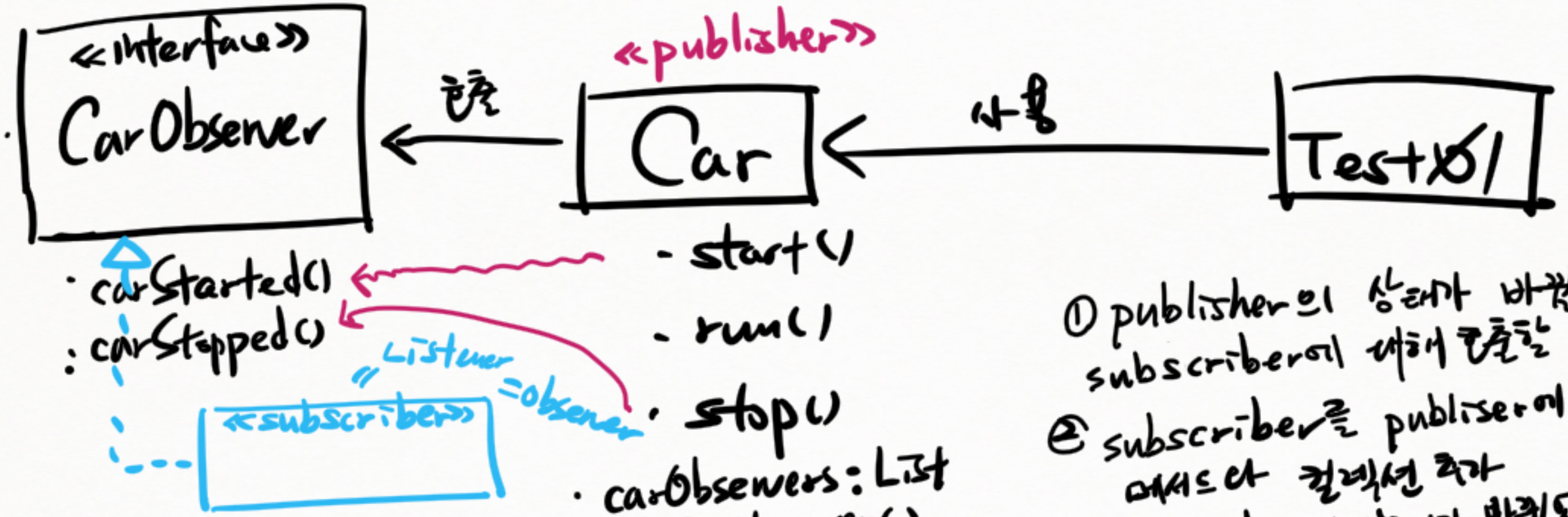
⑤ v1.4 → 시동 끌때, 경고등 끄기 ▢

⑥ v1.5 → 번쿠션 작동 안함 *



* Car 클래스에 Observer 패턴 적용 후

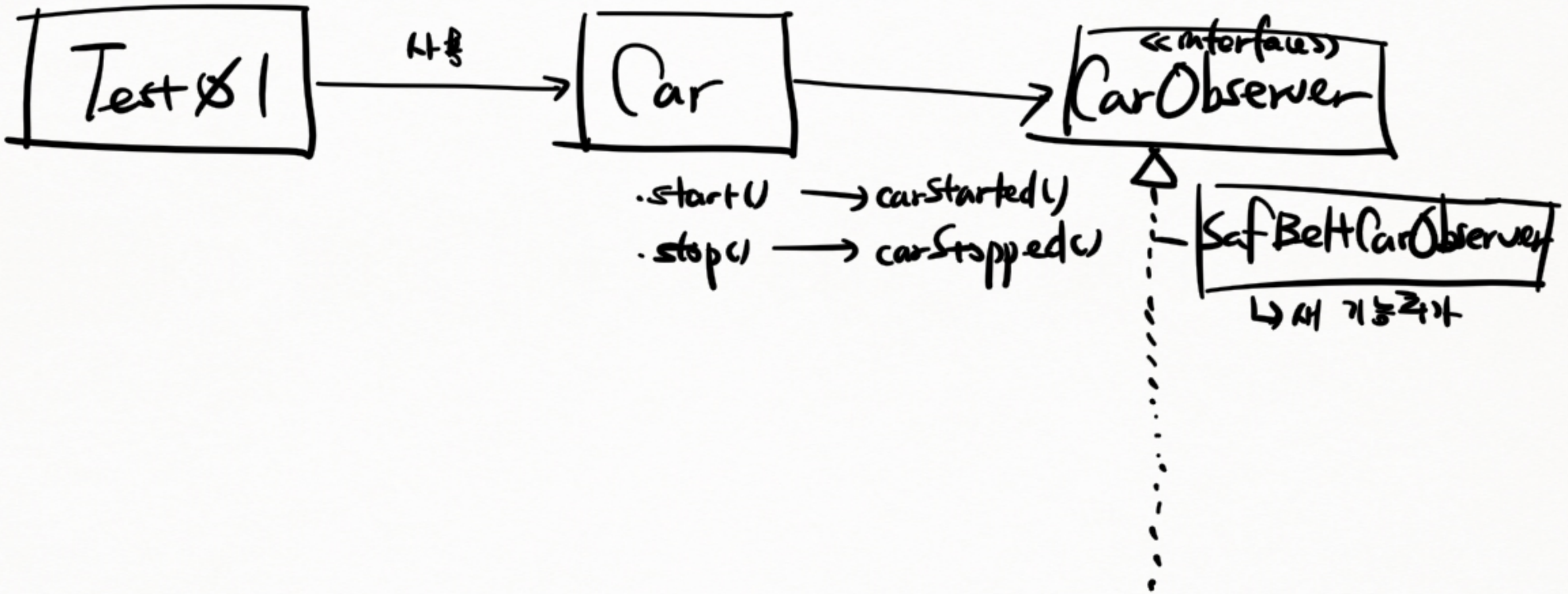
Observer = Listener = subscriber



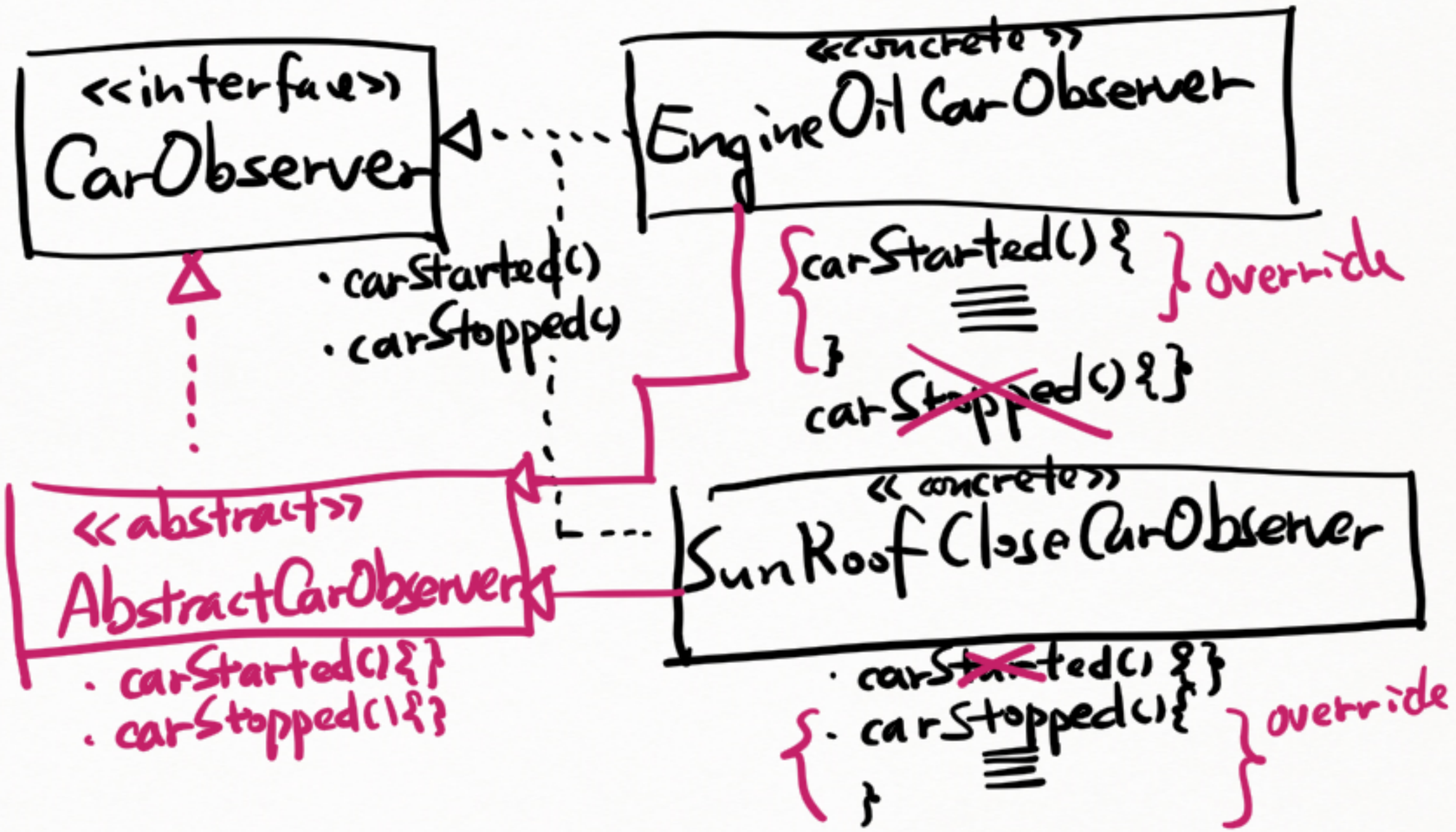
① publisher의 상태가 바뀔 때 마다 subscriber에 대해 호출할 메서드 규칙 정의

② subscriber를 publisher에 등록하고 제거할 메서드와 컬렉션 추가

③ publisher의 상태가 바뀌었을 때 subscriber에게 통지하는 코드 추가


Test01
사용
Car
<<interface>>
CarObserver
.startC) → carStarted()
.stop() → carStopped()
SafBeltCarObserver
↳새 기능추가

* 추상클래스 구현의 핵심


<<interface>>
CarObserver
· carStarted()
· carStopped()
<<concrete>>
EngineOilCarObserver
carStarted() { }
override
carStopped() {}
<<abstract>>
AbstractCarObserver
· carStarted() {}
· carStopped() {}
<<concrete>>
SunRoofCloseCarObserver
· carStarted() {}
· carStopped() { }
override

* 자바 실습 프로젝트에 Observer 패턴 적용