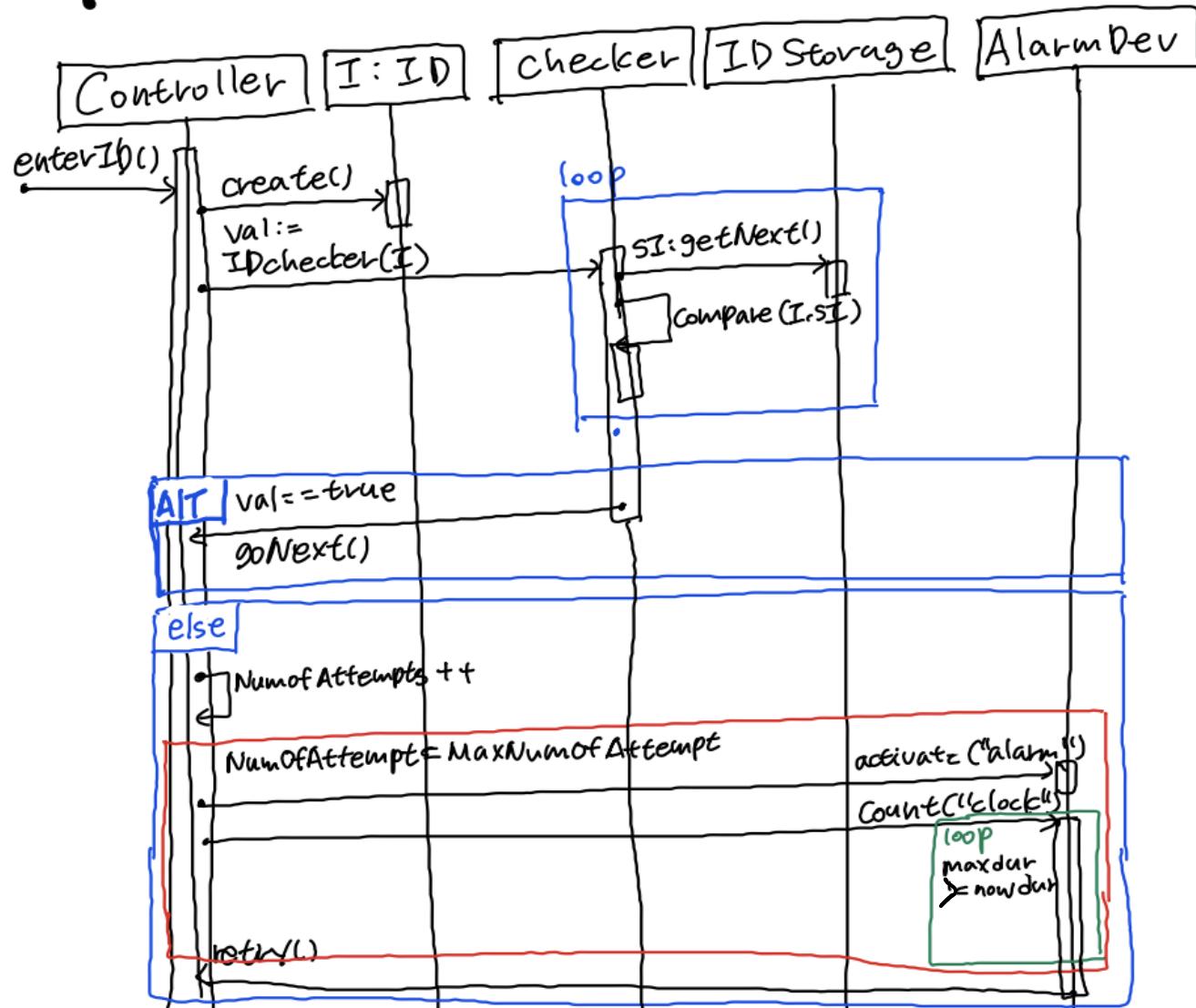


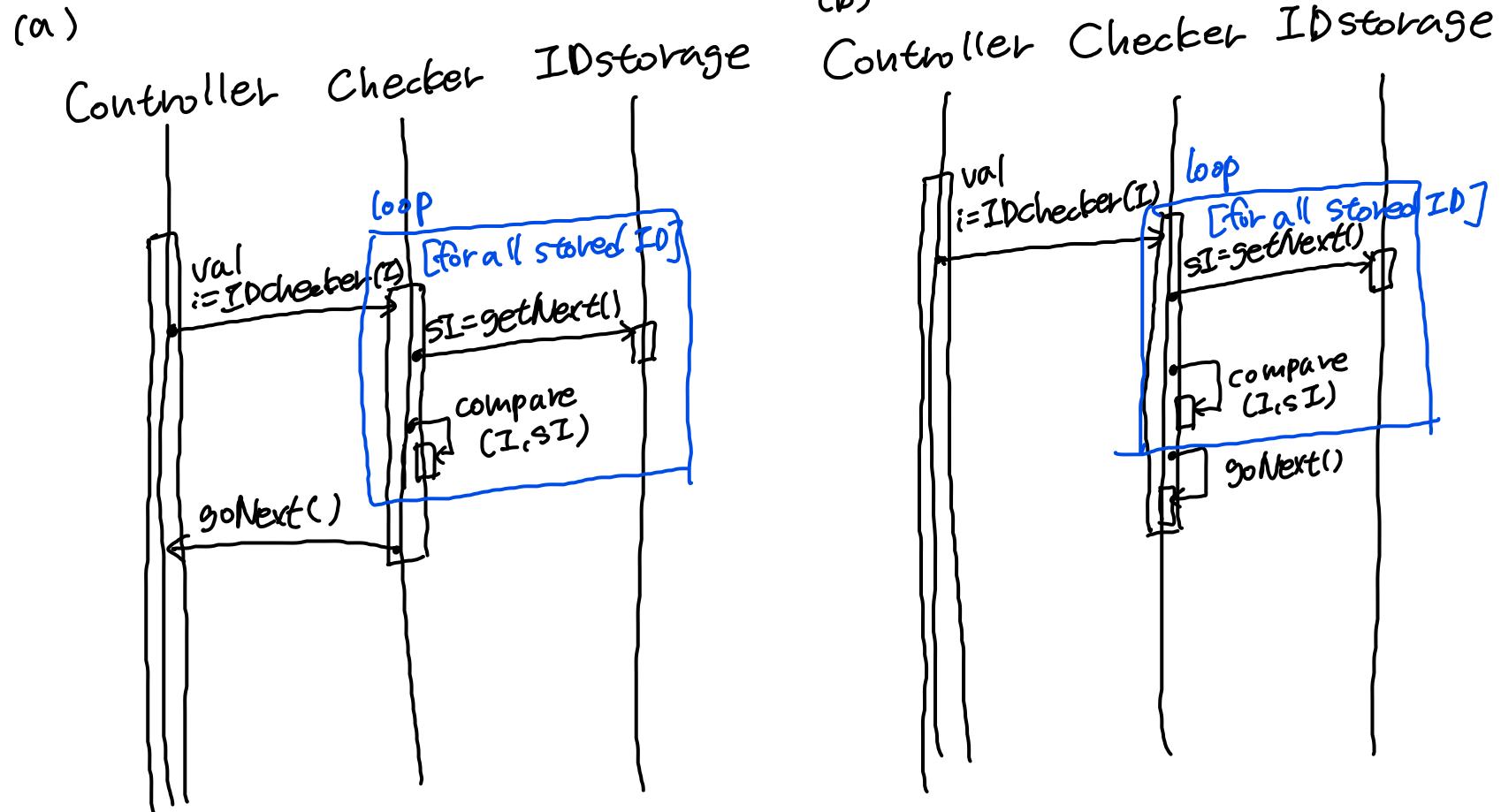
# UC-1



# Variation 1

(a) ID가 올바르다면 Controller로 goNext() (다음 단계로 실행하도록  
구현을 부여하는 기능) 신호 보내기

(b) ID가 올바르다면 checker에서并通过 goNext(), Checker가 다음 단계를  
실행시킬 구현을 가진다.

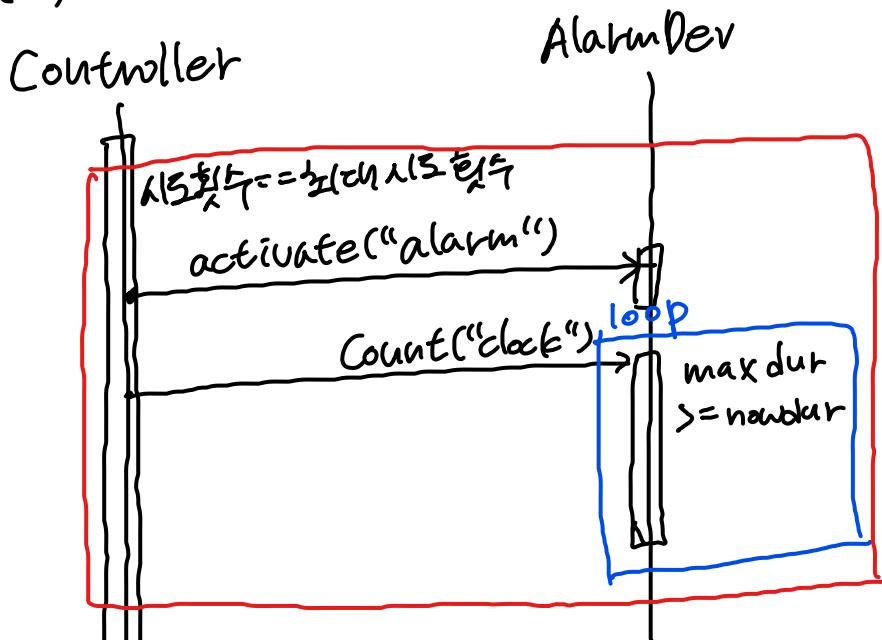


- (b) 일정은, ID가 올바른지 검사하는 기능과 상관없이 새로운 구현한을 갖게 된다. (전문성↓)
- (a) 일정은, Controller가 Checker와의 의존성이 커지지만, Checker 블록의 기능에서 크게 벗어나지 않는다. goNext()는 올바른 ID를 찾았다는 신호이므로 다른 Use Case와 연결시켜 프로그램을 동작할 때, 같은 Controller에서 신호를 받아 처리하는 것이 Checker에서 처리하는 것보다 흐름이 덜 복잡하다.

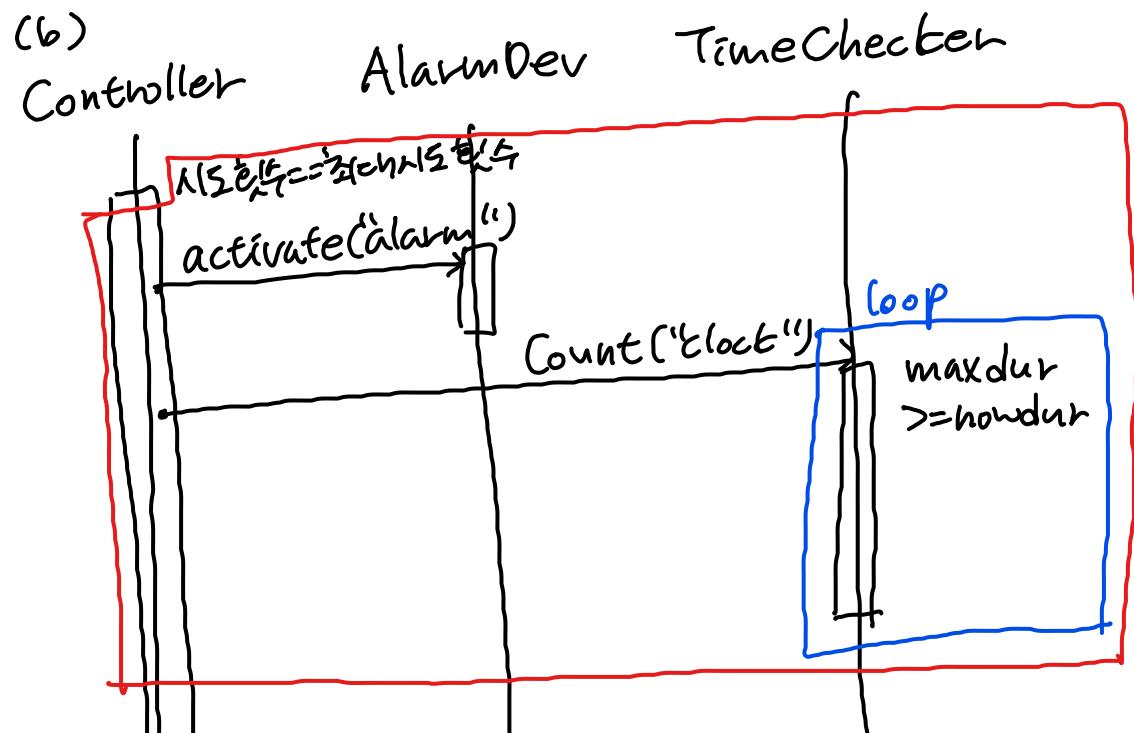
## Variation 2

- (a) AlarmDev에 시간측정 + 알람기능을 한 곳에 모아둔다.  
 (b) TimeChecker Class를 추가하고, AlarmDev는 알람만 울리는 기능을 수행한다.

(a)



(b)



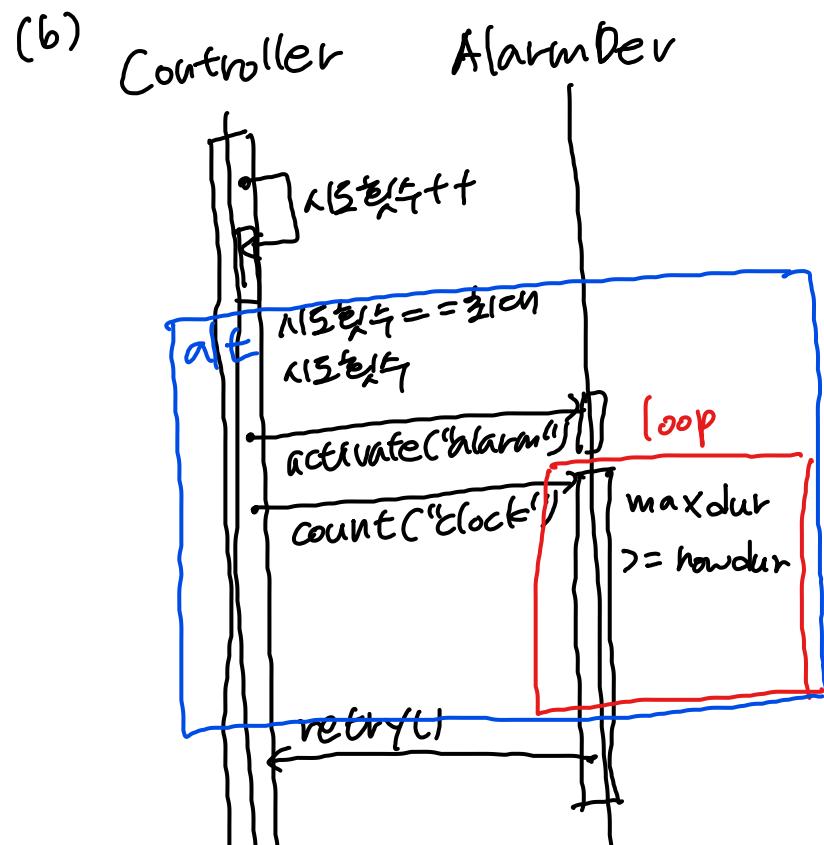
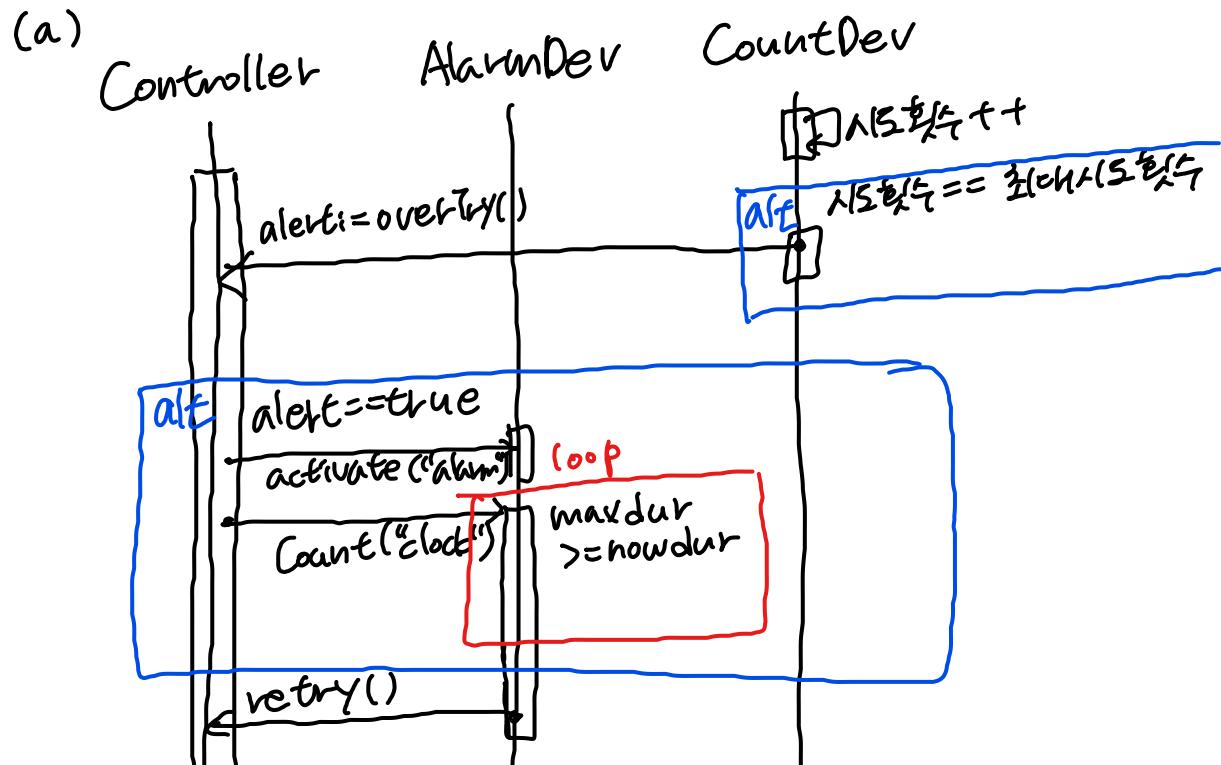
(a)의 경우 (b)보다 하다의 클래스에 더 많은 메소드를 포함한다. 그러나 시간 측정과 알람 기능을 함께 하는 것은 비슷한 기능을 하기 때문이다. 오히려 클래스를 2개로 나눌 경우 전문성을 높이지겠지만, 더욱 개별적으로 하다의 Controller에서 communication 할 class가 불필요하게 많아진다.

## Variation 3

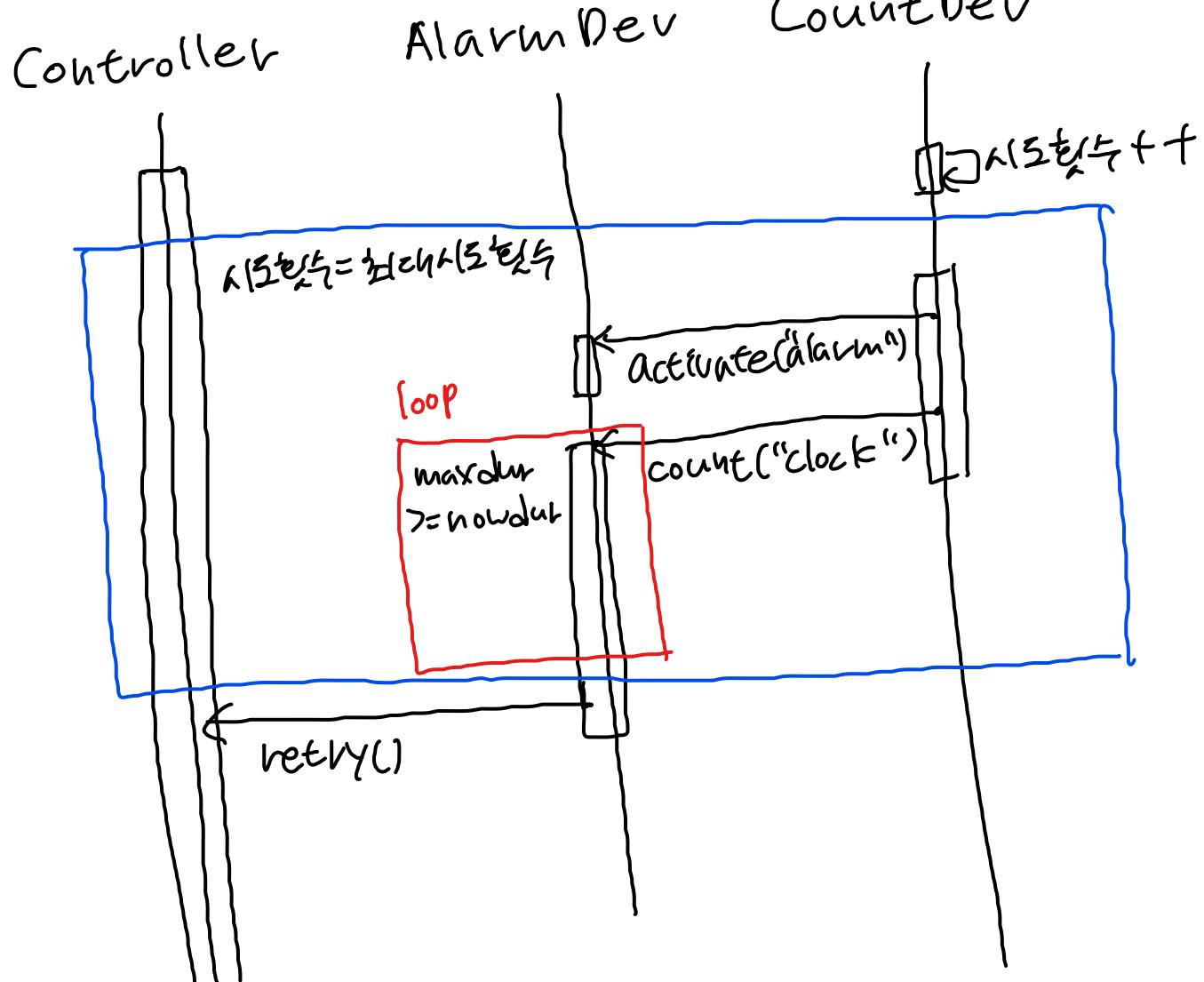
(a) CountDev class에서 시도횟수를 측정하여 최대시도횟수 초과 유무를 알린다.

(b) Controller 내에 시도횟수를 저장한다.

(c) CountDev class에서 시도횟수 측정과 최대시도횟수를 비교하여 알람을 울리는 신호를 보낸다.



(c)



(a) 일 경우, CountDev에 단지 시도횟수의 field만 가지고 있고, 여러한 기능도 수행하지 않으므로 전문성과 응집도 모두 높지 않고 지나치게 개별적이다.

(b) Controller의 active(), Count() 기능은 시도횟수를 가지고 수행하기 때문에 '시도횟수' field를 Controller에 넣어 전문화, 비슷한 기능 특성끼리 모인다.

object oriented communication 도 가능하다.

(c) work balance가 CountDev에 치우쳐 있어 상대적으로 Controller의 기능이 많다.

# Class Diagram

