# 4. 디자인패턴

# 학습목표

학습목표

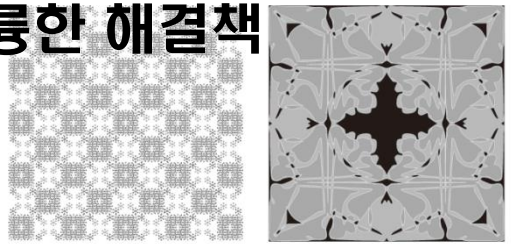
- 디자인 패턴을 만든 동기 이해하기
- 합동과 디자인 패턴 관계 이해하기
- 디자인 패턴 분류하기

# 4.1 디자인패턴의 이해

- 크리스토퍼 알렉산더
  - Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.
  - 바퀴를 다시 발명하지 마라 (Dont reinvent the wheel)
  - 패턴은 비슷하거나 동일한 양식 또는 유형들이 반복되어 나타난다는 의미이며, 문제와 해결책도 동일한 유형이나 양식을 통해 쉽게 찾을 수 있다.

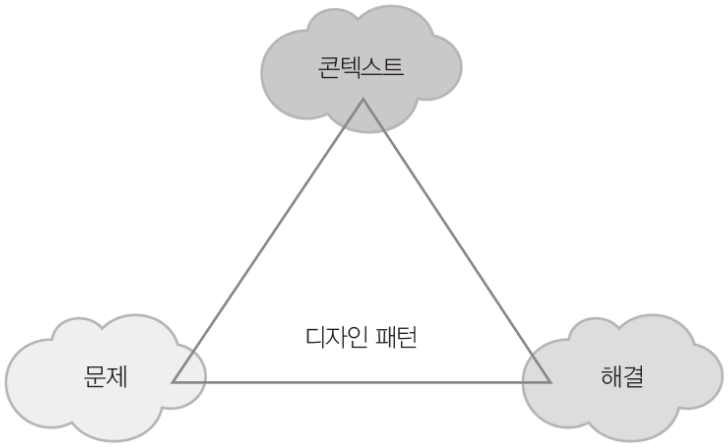
- 소프트웨어를 설계할 때 특정 맥락에서 자주 발생하는 고질적인 문제들이 발생했을 때 재사용할 수 있는 훌륭한 해결책

그림 4-1 패턴



# 디자인 패턴 구조

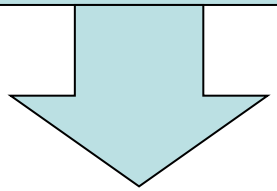
그림 4-3 디자인 패턴의 구성 요소




- 콘텍스트: 문제가 발생하는 여러 상황을 기술한다. 즉, 패턴이 적용될 수 있는 상황을 나타낸다. 경우에 따라서는 패턴이 유용하지 못하는 상황을 나타내기도 한다.
- 문제: 패턴이 적용되어 해결될 필요가 있는 여러 디자인 이슈들을 기술한다. 이때 여러 제약 사항과 영향력도 문제 해결을 위해 고려해야 한다
- 해결: 문제를 해결하도록 설계를 구성하는 요소들과 그 요소들 사이의 관계, 책임, 협력 관계를 기술한다. 해결은 반드시 구체적인 구현 방법이나 언어에 의존적이지 않으며 다양한 상황에 적용할 수 있는 일종의 템플릿이다.

# 디자인 패턴의 구조

- 콘텍스트: 클래스가 객체를 생성하는 과정을 제어해야 하는 상황
- 문제: 애플리케이션이 전역적으로 접근하고 관리할 필요가 있는 데이터를 포함한다. 동시에 이러한 데이터는 시스템에 유일하다. 어떤 방식으로 클래스에서 생성되는 객체의 수를 제어하고 클래스의 인터페이스에 접근하는 것을 제어해야 하는가?
- 해결: 클래스의 생성자를 public으로 정의하지 말고 private이나 protected로 선언해 외부에서 생성자를 이용해 객체를 일단 생성할 수 없게 만들고 (이하 생략).



싱글톤 패턴

# GoF 디자인 패턴

## ❖ GoF(Gang of Four)

- 감마/리차드 헬름/랄프 존슨/존 블리시디시
- 생성 패턴: 객체의 생성에 관련된 패턴
- 구조 패턴: 클래스를 조합해 더 큰 구조를 만드는 패턴
- 행위 패턴: 알고리즘이나 책임의 분배에 관한 패턴

표 4-1 GoF 디자인 패턴의 분류

| | 생성 패턴 | 구조 패턴 | 행위 패턴 |
|---|---|---|---|
| 패턴 이름 | 추상 팩토리(Abstract Factory)<br>빌더(Builder)<br>팩토리 메서드(Factory Method)<br>프로토타입(Prototype)<br>싱글턴(Singleton) | 어댑터(Adapter)<br>브리지(Bridge)<br>컴퍼지트(Composite)<br>데커레이터(Decorator)<br>퍼사드(façade)<br>플라이웨이트(Flyweight)<br>프록시(Proxy) | 책임 연쇄<br>(Chain of Responsibility)<br>커맨드(Command)<br>인터프리터(Interpreter)<br>이터레이터(Iterator)<br>미디에이터(Mediator)<br>메멘토(Memento)<br>옵서버(Observer)<br>스테이트(State)<br>스트래티지(Strategy)<br>템플릿 메서드(Template Method)<br>비지터(Visitor) |

# 4.3 UML과 디자인 패턴

- 컬레보레이션을 통해 디자인 패턴 기술
- 역할들의 상호작용을 추상화

그림 4-4 객체와 역할

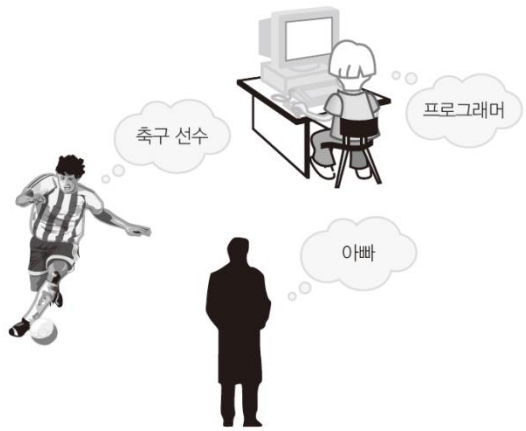
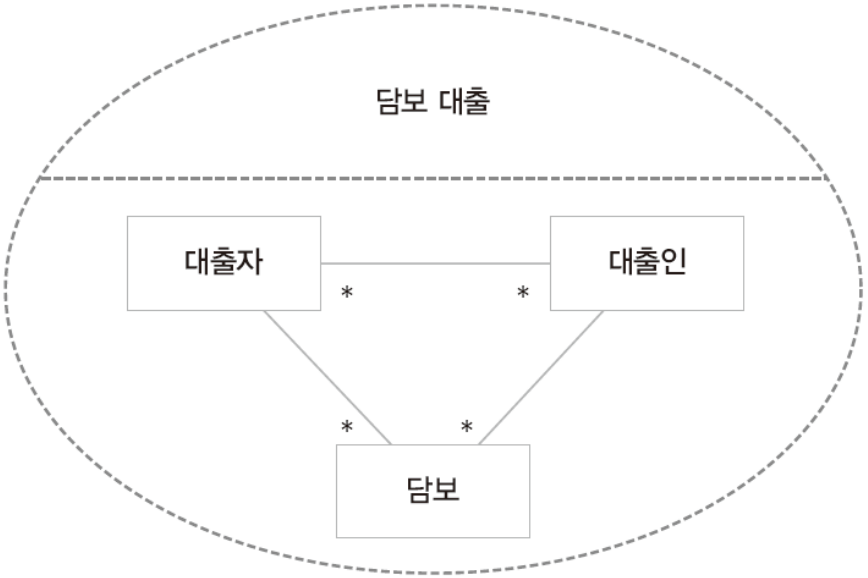



그림 4-5 컬레보레이션



# 컬레보레이션 어커런스

그림 4-5 컬레보레이션

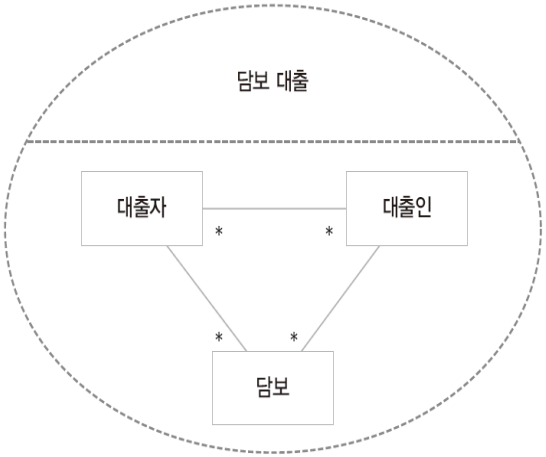
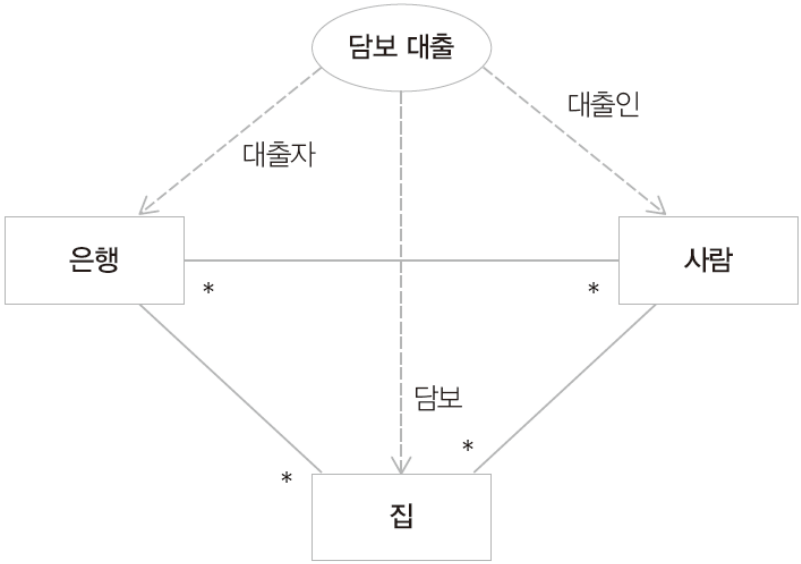


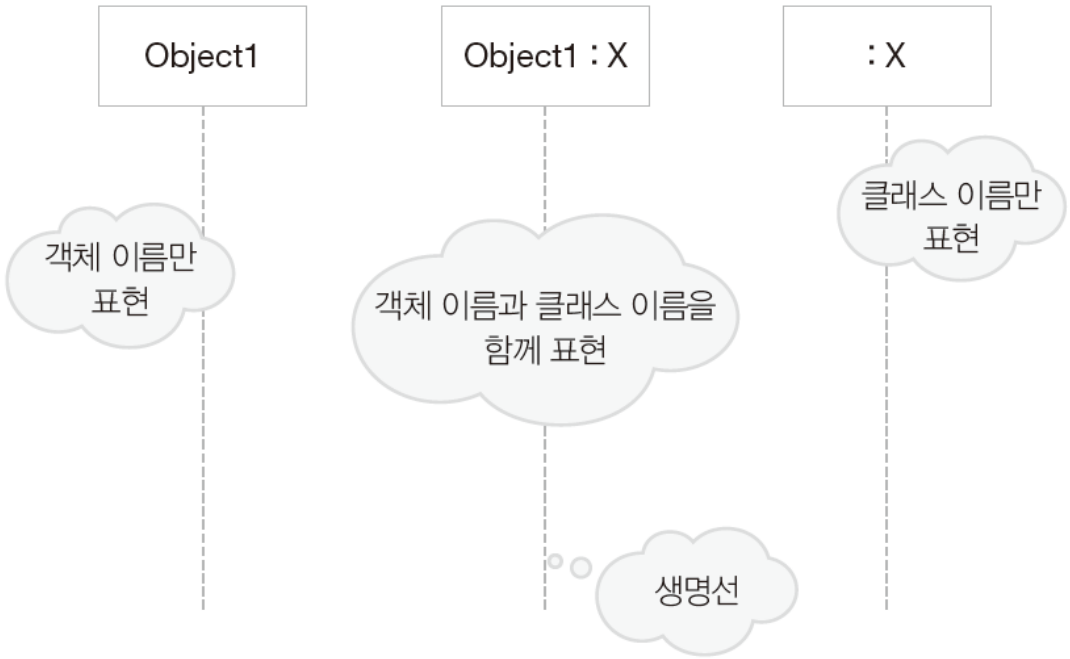

그림 4-6 컬레보레이션 어커런스



# 순차 다이어그램

## ❖ 객체들 사이의 메시지 송신과 그들의 순서

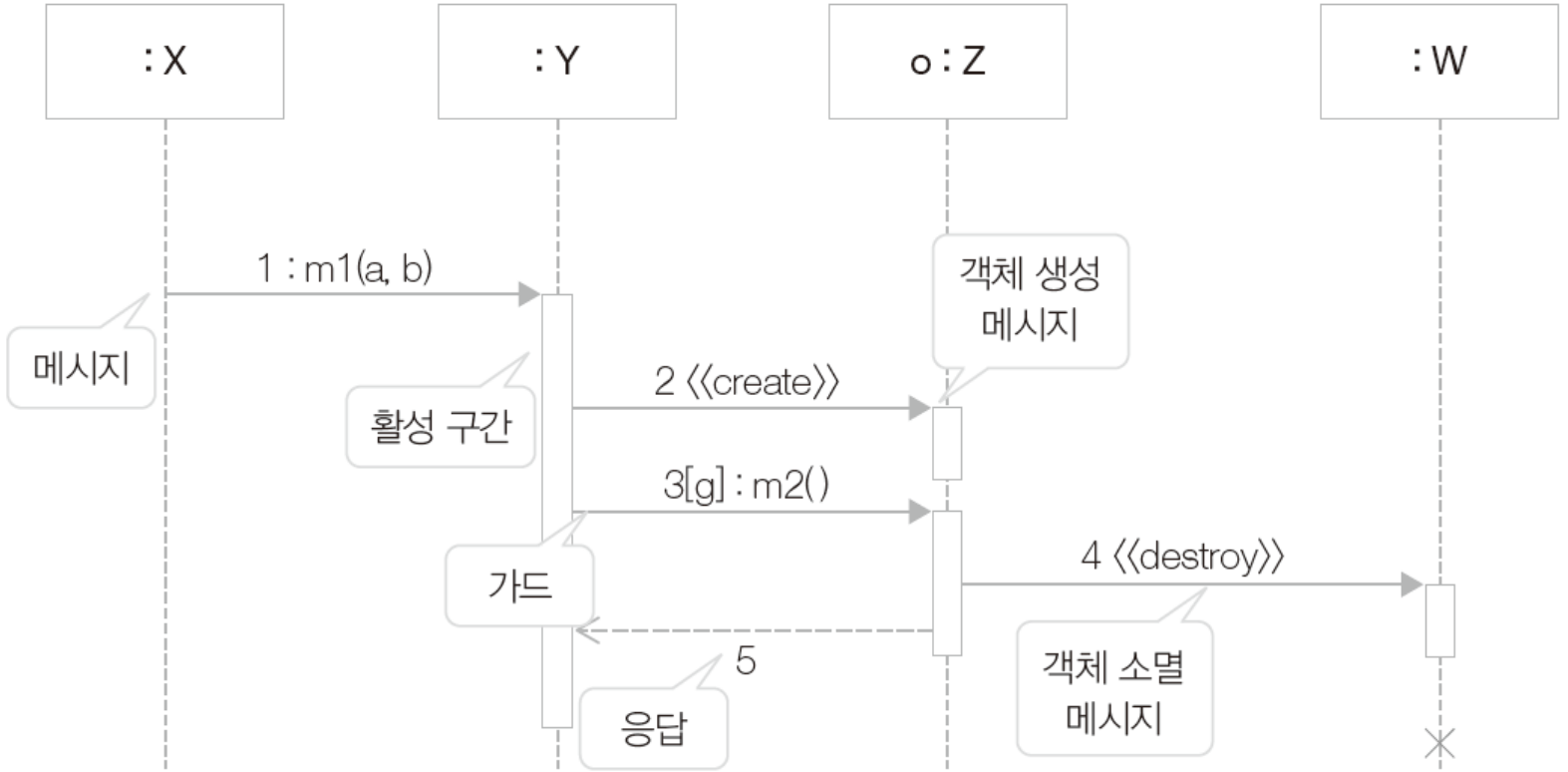
그림 4-7 객체의 3가지 표현




**Keypoint_** UML 1.X에서는 객체 이름 아래에 밑줄이 표시되지만 UML 2.0에서는 밑줄을 생략할 수 있다. 그런데 대표적 UML 도구인 starUML을 사용하면 객체 이름에 여전히 밑줄이 표시된다.

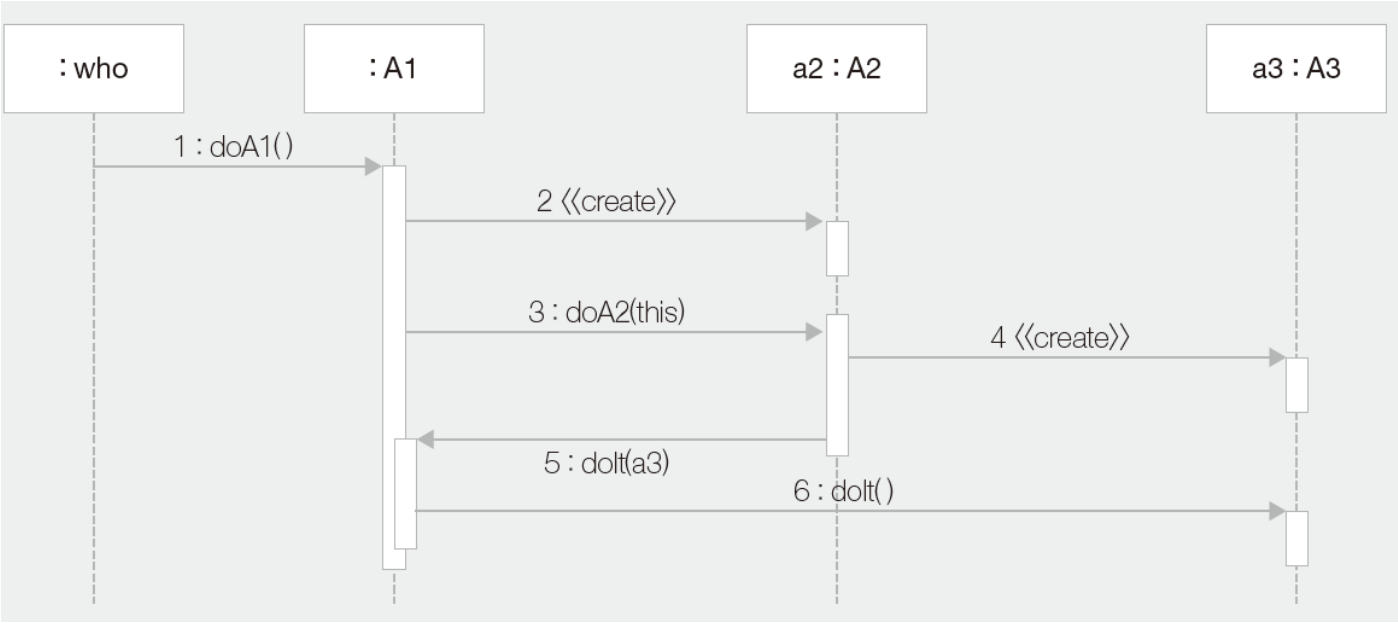
# 메시지

**그림 4-8** 여러 가지 형태의 메시지 표현



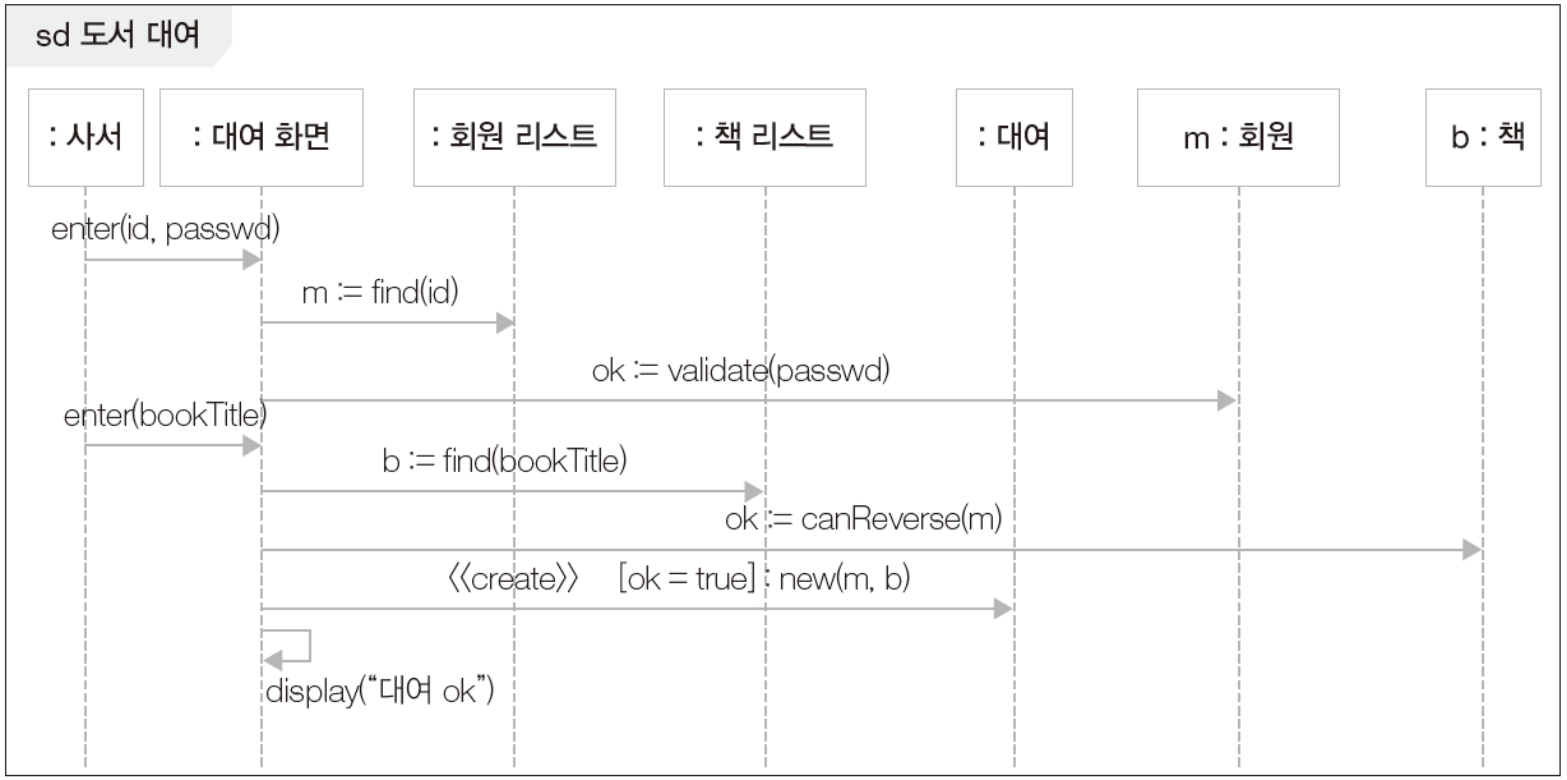
# 체크포인트

- 그림 4-8의 순차 다이어그램에 해당하는 코드를 작성하라.
- 다음 순차 다이어그램에 해당하는 코드를 작성하라.

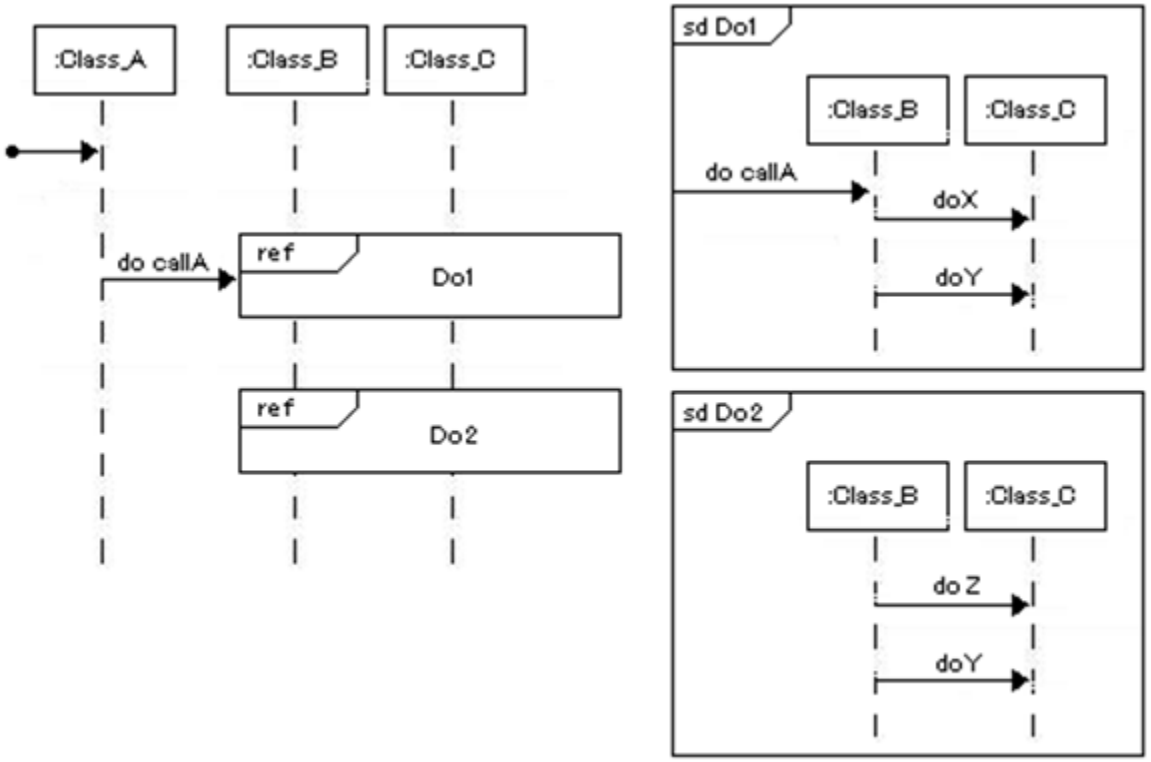


# 프레임

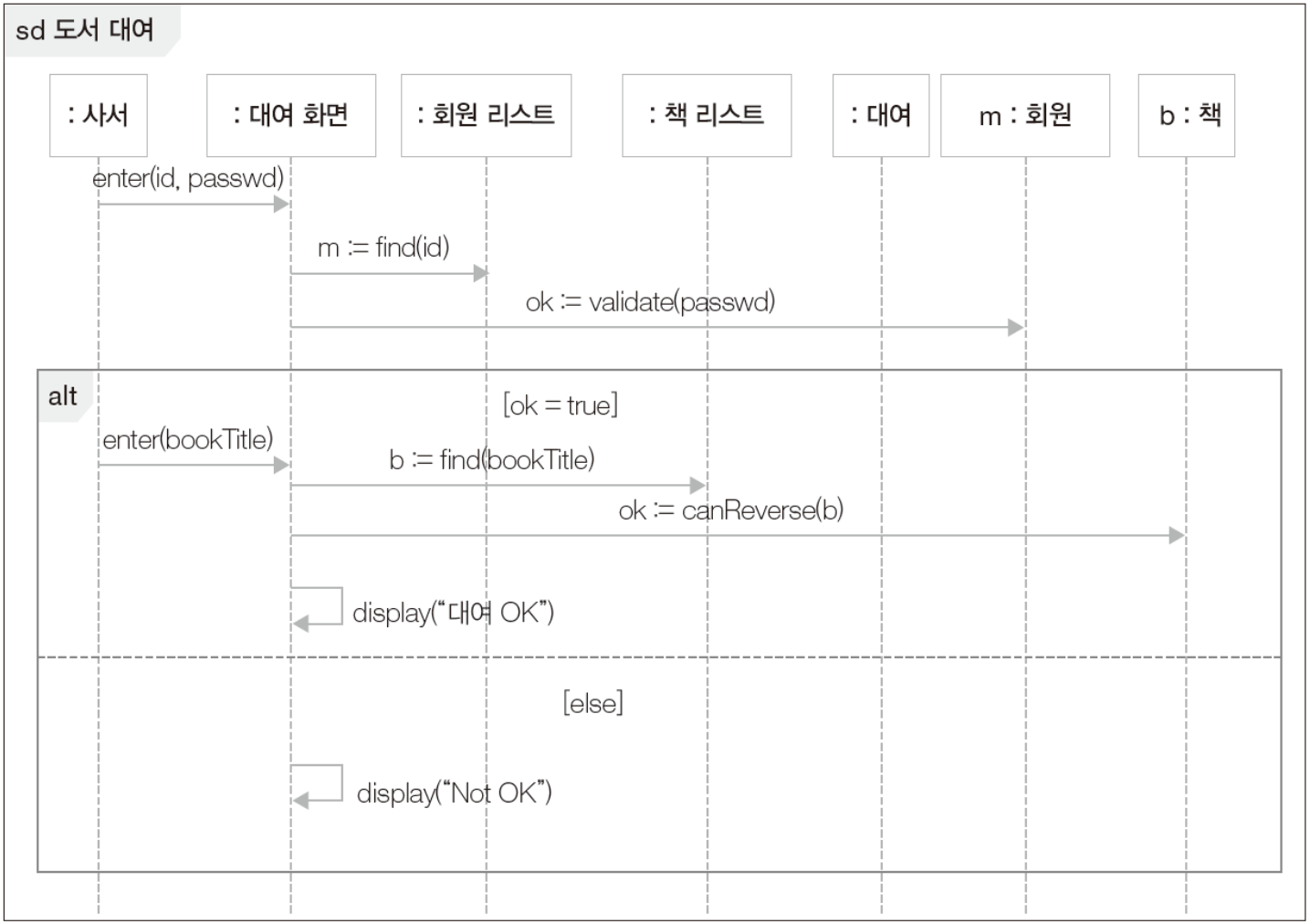
그림 4-9 프레임을 사용한 순차 다이어그램



# 상호작용


:Class_A
:Class_B
:Class_C
do callA
ref
Do1
ref
Do2
sd Do1
:Class_B
:Class_C
do callA
doX
doY
sd Do2
:Class_B
:Class_C
do Z
doY

# alt


sd 도서 대여
: 사서
: 대여 화면
: 회원 리스트
: 책 리스트
: 대여
m : 회원
b : 책
enter(id, passwd)
m := find(id)
ok := validate(passwd)
alt
[ok = true]
enter(bookTitle)
b := find(bookTitle)
ok := canReverse(b)
display("대여 OK")
[else]
display("Not OK")

# loop

그림 4-11 loop 키워드

# 순차 다이어그램과 클래스 다이어그램

그림 4-12 연관이나 의존 관계가 존재하지 않는 순차 다이어그램

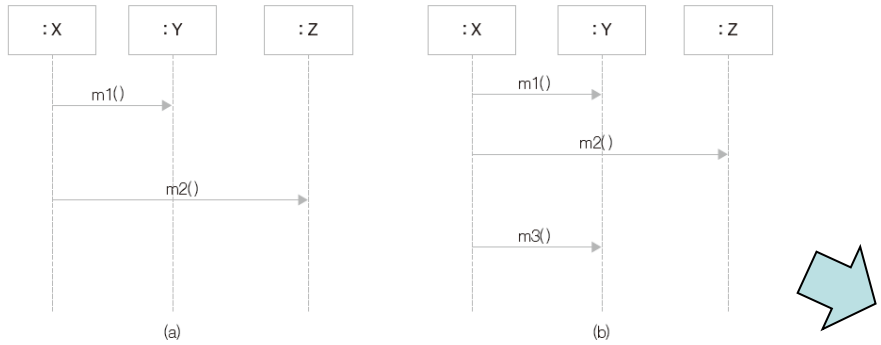
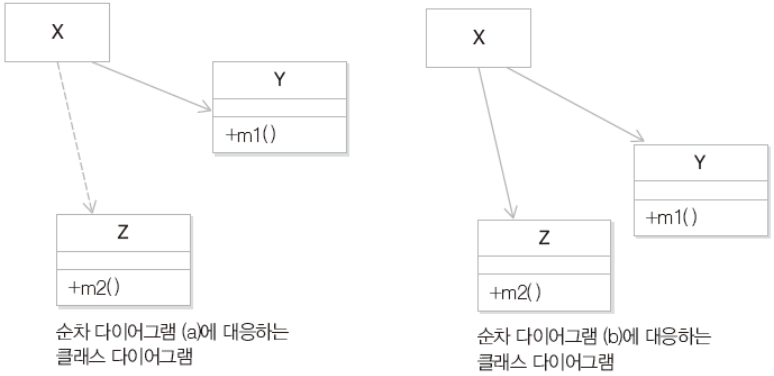



그림 4-13 그림 4-12를 순차 다이어그램으로 수정




순차 다이어그램 (a)에 대응하는 클래스 다이어그램

순차 다이어그램 (b)에 대응하는 클래스 다이어그램