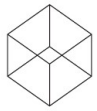
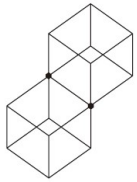
4. 디자인패턴



JAVA 객체 지향 디자인 패턴

UML과 GoF 디자인 패턴 핵심 10가지로 배우는



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)**
  - 불필요하게 처음부터 다시 시작하지 마라

그림 4-1 패턴

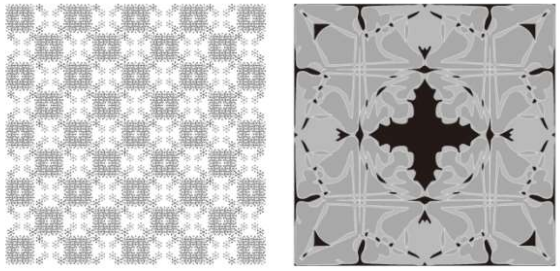
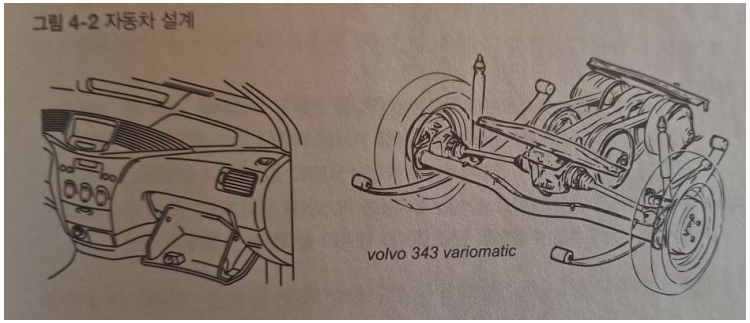


그림 4-2 자동차 설계



❖ 디자인 패턴은?

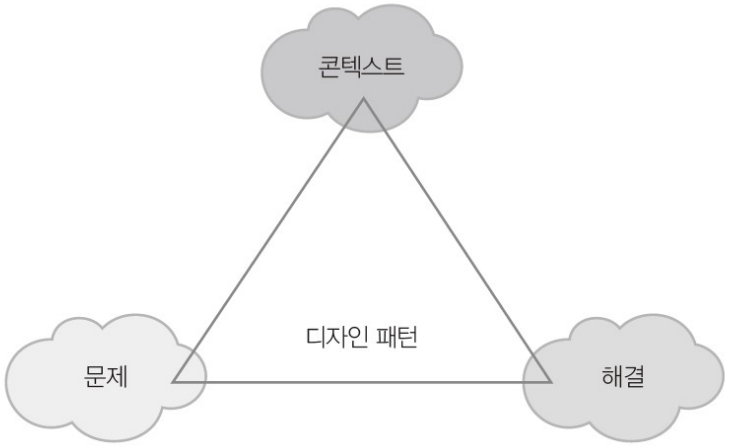
- 소프트웨어를 설계할 때 특정 맥락에서 자주 발생하는 고질적인 문제들이 발생했을 때 재사용할 수 있는 훌륭한 해결책

❖ 패턴

- 사전 : “일정한 형태의 양식이나 유형 “
- 동일한 양식 또는 유형들이 반복되어 나타난다는 의미이며, 문제와 해결책도 동일한 유형이나 양식을 통해 쉽게 찾을 수 있다.

디자인 패턴의 필수적인 3요소

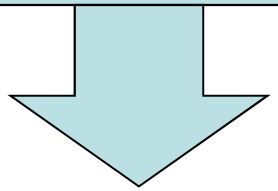
그림 4-3 디자인 패턴의 구성 요소



- 콘텍스트(context): 문제가 발생하는 여러 상황을 기술한다. 즉, 패턴이 적용될 수 있는 상황을 나타낸다. 경우에 따라서는 패턴이 유용하지 못하는 상황을 나타내기도 한다.
- 문제(problem): 패턴이 적용되어 해결될 필요가 있는 여러 디자인 이슈들을 기술한다. 이때 여러 제약 사항과 영향력도 문제 해결을 위해 고려해야 한다
- 해결(solution): 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.

싱글톤 패턴에 대한 상황 설명

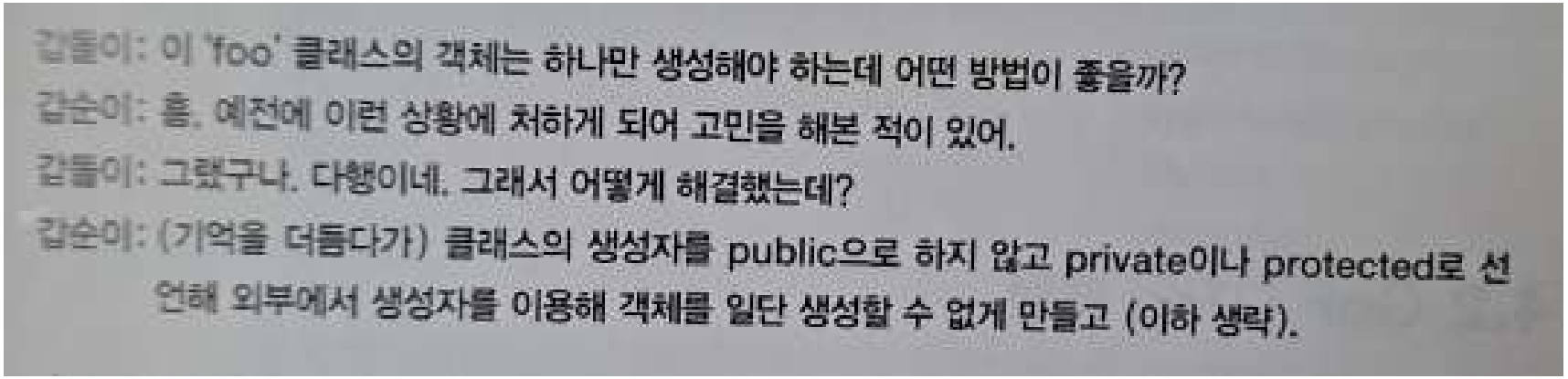
- 콘텍스트: 클래스가 객체를 생성하는 과정을 제어해야 하는 상황
- 문제: 애플리케이션이 **전역적으로 접근하고** 관리할 필요가 있는 데이터를 포함한다. 동시에 이러한 데이터는 **시스템에 유일하다**. 어떤 방식으로 클래스에서 생성되는 객체의 수를 제어하고 클래스의 인터페이스에 접근하는 것을 제어해야 하는가?
- 해결: 클래스의 생성자를 public으로 정의하지 말고 **private이나 protected**로 선언해 **외부에서 생성자를 이용해 객체를 일단 생성할 수 없게 만들고** (이하 생략).



싱글톤 패턴

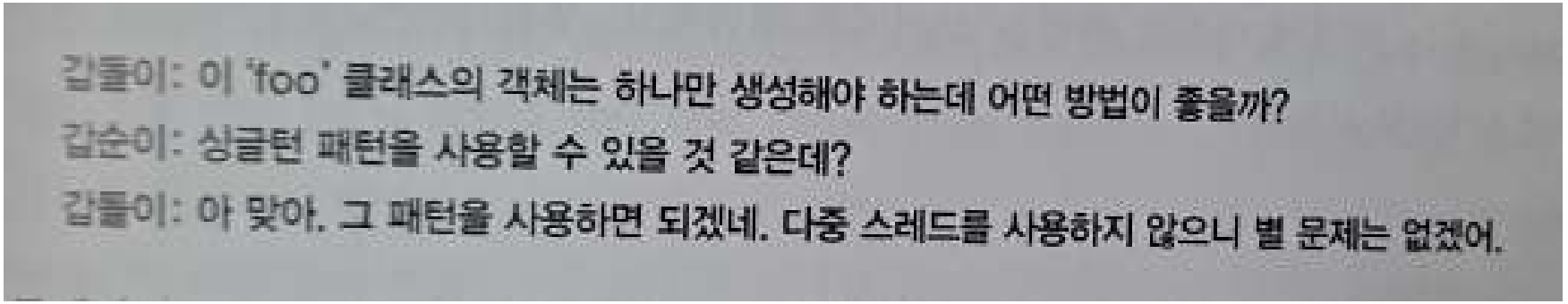
싱글톤 패턴의 초기 설계 과정

❖ 두명의 개발자의 대화

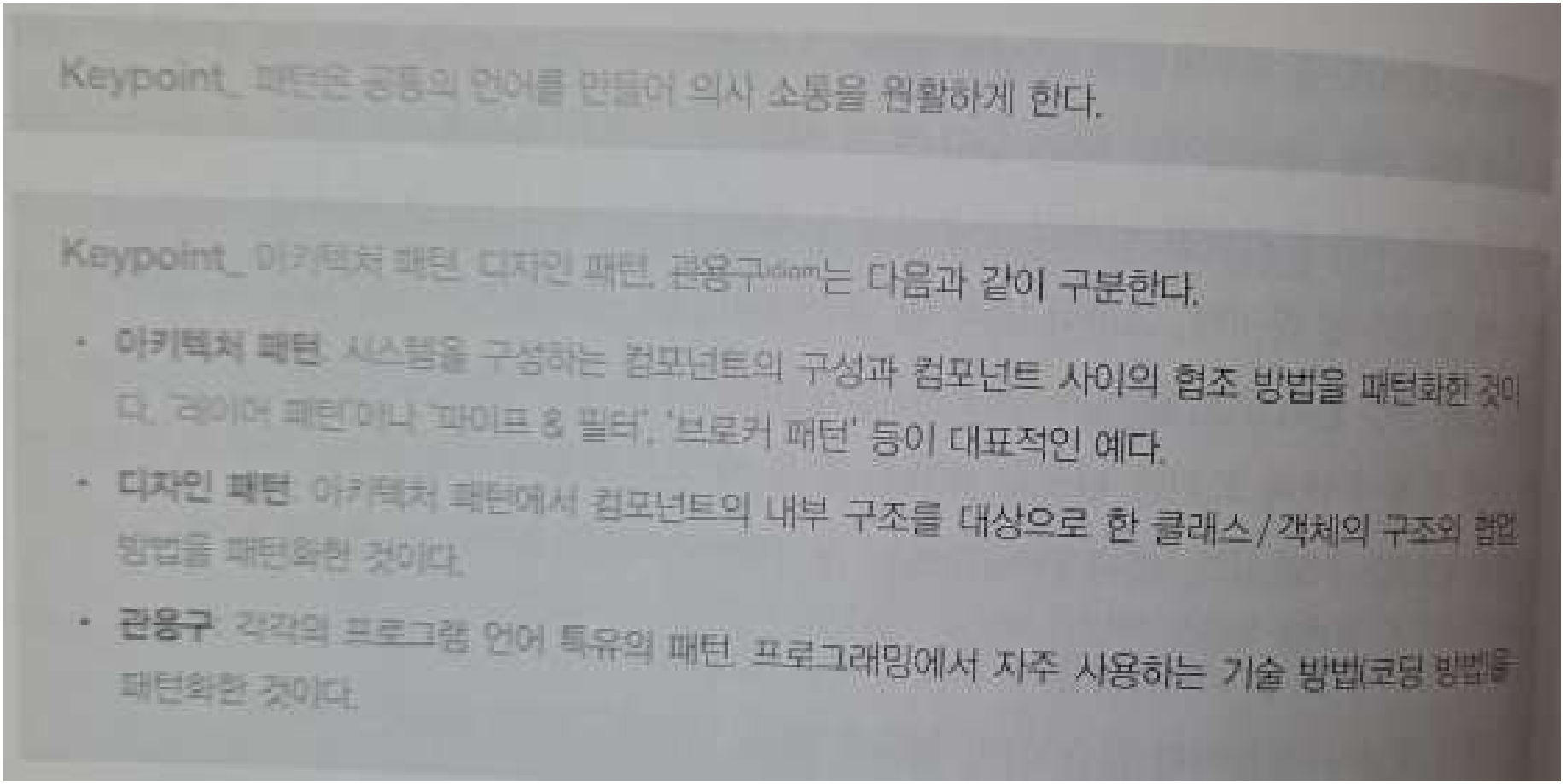


- 문제점에 대한 상세한 해결책 중심의 대화
- 문제를 매우 상세하게 설명

❖ 양 개발자가 싱글톤 패턴을 모두 알고 있을때의 대화



- 패턴은 공동의 언어를 만들어주며
- 팀원 사이의 의사 소통을 원활하게 해줌


Keypoint_ 패턴은 공통의 언어를 만들어 의사 소통을 원활하게 한다.
Keypoint_ 아키텍처 패턴, 디자인 패턴, 관용구idiom는 다음과 같이 구분한다.
• 아키텍처 패턴: 시스템을 구성하는 컴포넌트의 구성과 컴포넌트 사이의 협조 방법을 패턴화한 것이다. '계층 패턴'이나 '파이프 & 필터', '브로커 패턴' 등이 대표적인 예다.
• 디자인 패턴: 아키텍처 패턴에서 컴포넌트의 내부 구조를 대상으로 한 클래스/객체의 구조와 협조 방법을 패턴화한 것이다.
• 관용구: 각각의 프로그램 언어 특유의 패턴. 프로그래밍에서 자주 사용하는 기술 방법(코딩 방법)을 패턴화한 것이다.

4.2 GoF 디자인 패턴

- GoF(Gang of Four)
  - 에리히 감마/리차드 헬름/랄프 존슨/존 블리시디시
  - 총 23가지의 디자인 패턴
  - 생성(Creational) 패턴: 객체의 생성에 관련된 패턴
  - 구조(Structural) 패턴: 클래스를 조합해 더 큰 구조를 만드는 패턴
  - 행위(Behavioral) 패턴: 알고리즘이나 책임의 분배에 관한 패턴

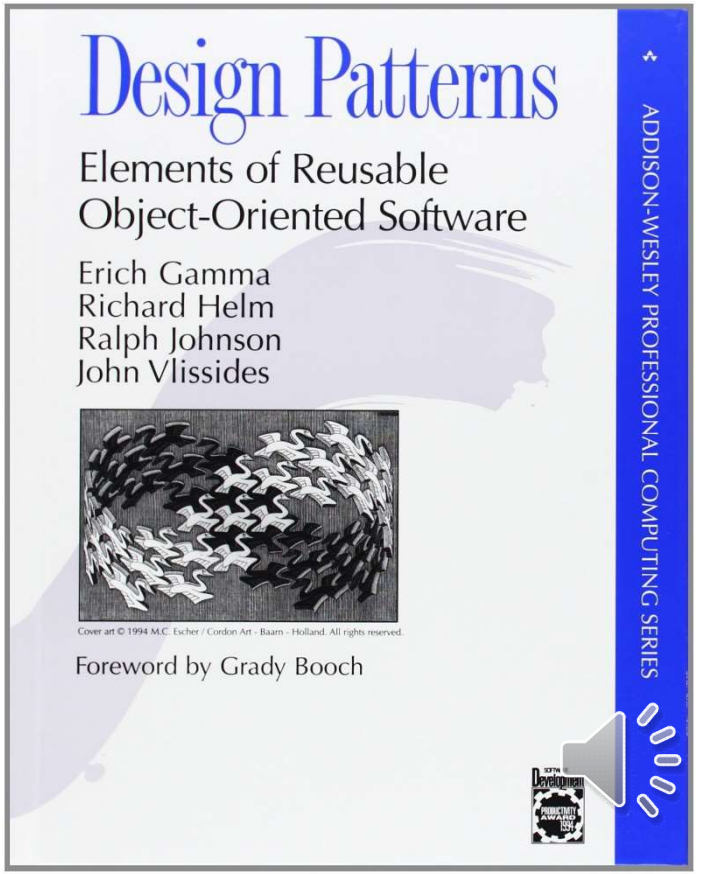


표 4-1 GoF 디자인 패턴의 분류

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

표 4-2. 이 책에서 다루는 10가지 GoF 패턴

패턴 분류	패턴 이름	패턴 설명
생성 패턴	추상 팩토리(Abstract Factory)	구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴
	팩토리 메서드(Factory Method)	객체 생성 처리를 서브 클래스로 분리해 처리하도록 캡슐화하는 패턴
	싱글턴(Singleton)	전역 변수를 사용하지 않고 객체를 하나만 생성하도록 하며, 생성된 객체를 어디에서든지 참조할 수 있도록 하는 패턴
구조 패턴	컴퍼지트(Composite)	여러 개의 객체들로 구성된 복합 객체와 단일 객체를 클라이언트에서 구별 없이 다루게 해주는 패턴
	데코레이터(Decorator)	객체의 결합을 통해 기능을 동적으로 유연하게 확장할 수 있게 해주는 패턴
행위 패턴	옵서버(Observer)	한 객체의 상태 변화에 따라 다른 객체의 상태도 연동되도록 일대다 객체 의존 관계를 구성하는 패턴
	스테이트(State)	객체의 상태에 따라 객체의 행위 내용을 변경해주는 패턴
	스트래티지(Strategy)	행위를 클래스로 캡슐화해 동적으로 행위를 자유롭게 바꿀 수 있게 해주는 패턴
	템플릿 메서드(Template Method)	어떤 작업을 처리하는 일부분을 서브 클래스로 캡슐화해 전체 일을 수행하는 구조는 바꾸지 않으면서 특정 단계에서 수행하는 내역을 바꾸는 패턴
	커맨드(Command)	실행될 기능을 캡슐화함으로써 주어진 여러 기능을 실행할 수 있는 재사용성이 높은 클래스를 설계하는 패턴

4.3 UML과 디자인 패턴

❖ UML패턴의 표현

– 컬레버레이션(collaboration)

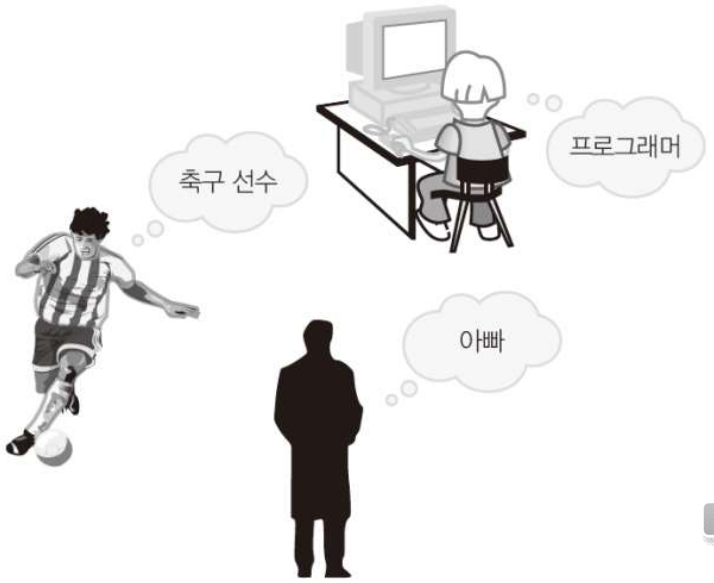
- 1) 구조적인 면
  - ➢ 어떤 요소들이 주어진 목적을 달성하기 위해 협력하는지를 나타냄

- 2) 행위적인 면
  - ➢ 협력을 위한 요소들의 상호작용
  - ➢ ==> 순차 다이어그램 (Sequence Diagram)

4.3.1 컬레보레이션

- 컬레보레이션을 통해 디자인 패턴 기술
- 객체와 역할 사이의 관계
  - 객체는 역할이 아니므로 한 객체가 여러 역할을 수행할수 있음
  - 역할들의 상호작용을 추상화

그림 4-4 객체와 역할



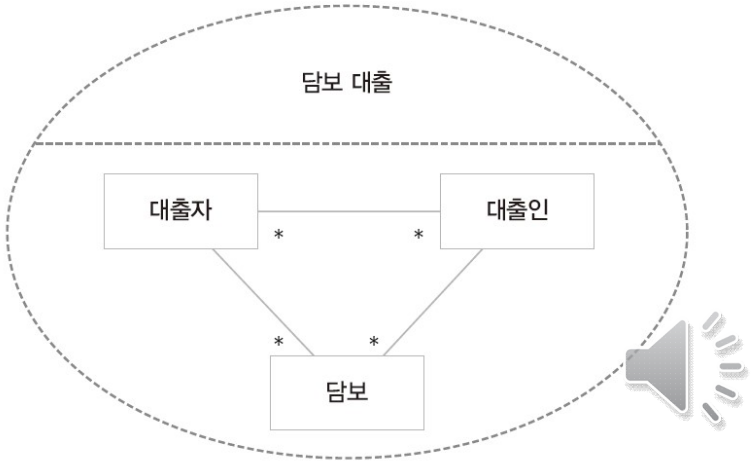
❖ 용어

- Collaboration != 협력다이어그램(collaboration diagram)
- collaboration diagram ➔ communication diagram

❖ Collaboration

- 점선으로 된 타원기호
- 타원 내부에 협력을 필요로 하는 역할들과 그들 사이의 연결 관계를 표현
- 그림 4-5. 담보대출관계를 보여주는 컬러보레이션

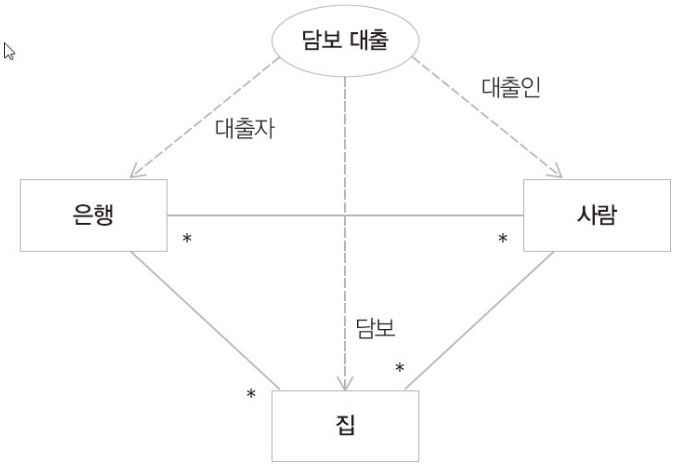
그림 4-5 컬레보레이션



컬레보레이션 어커런스

- Keypoint_컬러보레이션은 객체가 수행하는 역할들의 협력을 표현해준다.
- Collaboration occurrence
  - 구체적인 상황에서의 컬레보레이션 적용
  - 그림 4-6. 은행에서 집을 답보로 대출을 하는 경우

그림 4-6 컬레보레이션 어커런스

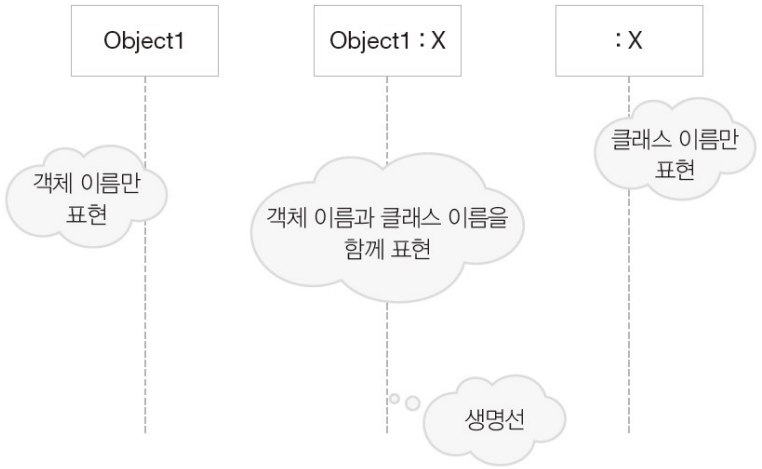


4.3.2 순차 다이어그램

❖ **순차 다이어그램 : 객체들 사이의 메시지 송신과 그들의 순서**

- **객체는 윗 부분에 표시, 왼쪽에서 오른쪽으로 객체들을 나열**
- **생명선 (lifetime) 객체 아래의 점선, 해당 객체가 존재함을 의미**
- **활성구간 (activation) : 좁고 긴 사각형**
  - 실제로 객체가 연산을 실행하는 상태임을 의미

그림 4-7 객체의 3가지 표현



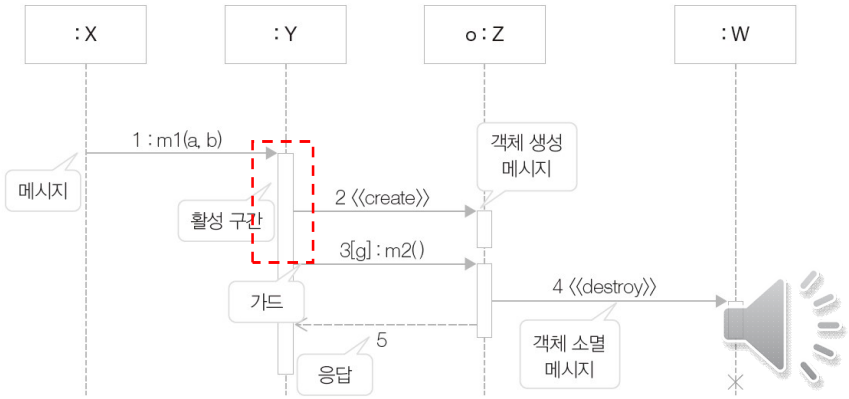
Keypoint_ UML 1.X에서는 객체 이름 아래에 밑줄이 표시되지만 UML 2.0에서는 밑줄을 생략할 수 있다. 그런데 대표적 UML 도구인 starUML을 사용하면 객체 이름에 여전히 밑줄이 표시된다.

메시지

❖ 표기법

- **객체 사이의 메시지 : 화살표**
- **화살표의 시작부분 : 메시지를 송신하는 객체**
- **화살표의 끝부분 : 메시지를 수신하는 객체**
- **머리 부분이 채워지지 않는 , 열려있는 화살표 : 비동기 메시지**
  - 메시지를 송신한 후, 메시지 실행이 끝나기를 기다리지 않고 다음 작업을 바로 수행할수 있음 예) 프린터에 프린트물을 전송하면 프린터 큐에 넣어지고, 요청한 객체는 다른일을 …….
- **머리 부분이 채워진 화살표 : 동기 메시지**
  - 메시지의 실행을 요청하는 객체가 메시지의 실행이 종료될 때까지 다음 작업을 수행할수 없음

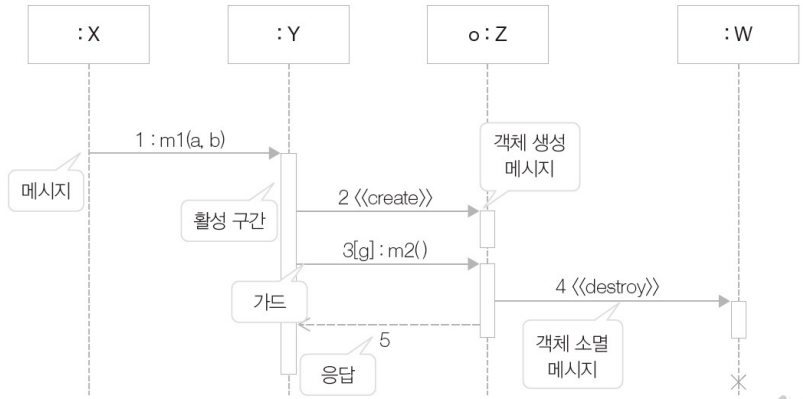
그림 4-8 여러 가지 형태의 메시지 표현



❖ 그림 4-8

- **모두 동기 메시지**
- **스테레오타입을 붙인 메시지**
  - 스테레오타입 : UML모델링 요소들을 모델러의 기준에 따라 새로운 분류를 적용할수 있도록 하용하는 메커니즘 , 모델로 마음대로 정의할수 있음
- **<<Create>>**
  - 객체를 생성하는 메시지
- **<<destroy>>**
  - 객체를 소멸시킬 때
  - 소멸되는 객체의 생명선 끝에 X 표시

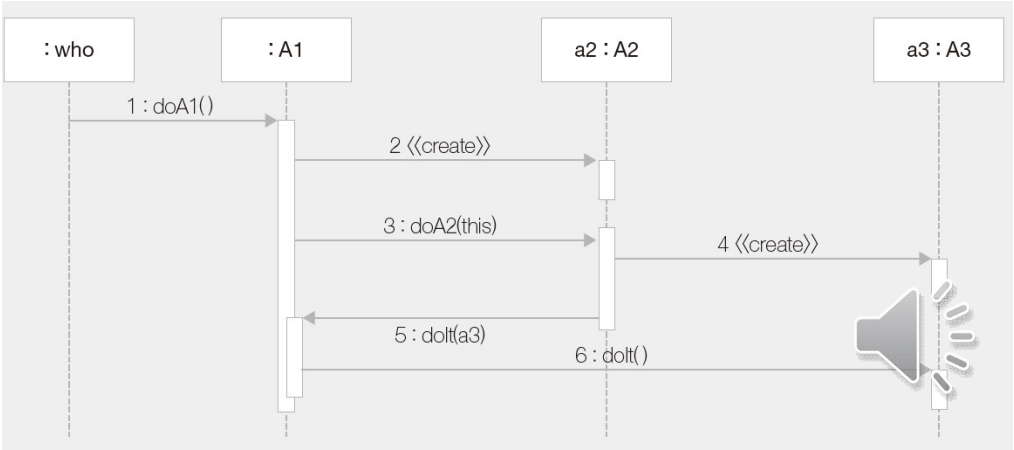
그림 4-8 여러 가지 형태의 메시지 표현



❖ 메시지 표현 형식

[시퀀스 번호] [가드]: 반환 값:=메시지 이름([인자 리스트])

- **메시지 이름을 제외하고는 모두 생략할수 있음**
- **시퀀스 번호 : 보통 생략 가능**
- **가드 (guard) : 메시지가 송신되는데 만족해야하는 조건**
- **점선 화살표 : 응답(reply) 메시지**
  - 응답메세지는 메시지가 종료되었음을 표현
  - 반드시 표현해야하는 것은 아님



체크포인트(p.153)

❖ 그림 4-8의 순차 다이어그램에 해당하는 코드를 작성하라

- ➔ p. 159

그림 4-8 여러 가지 형태의 메시지 표현

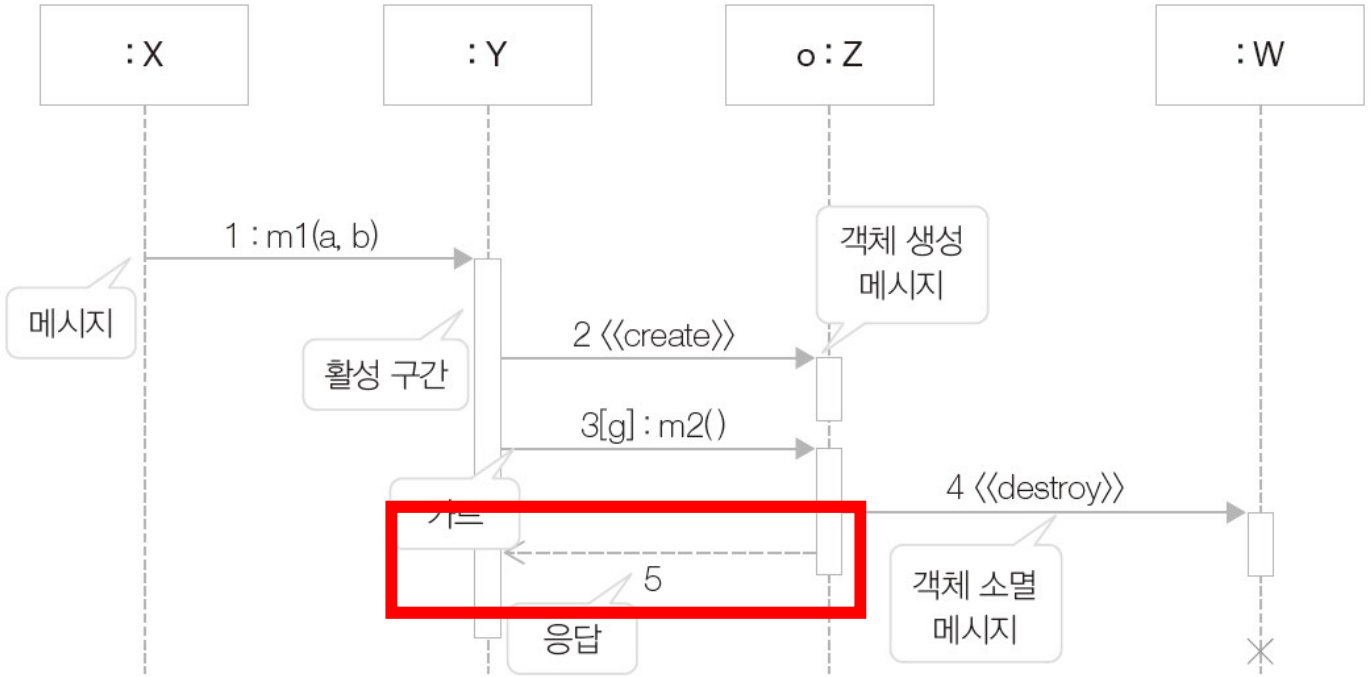
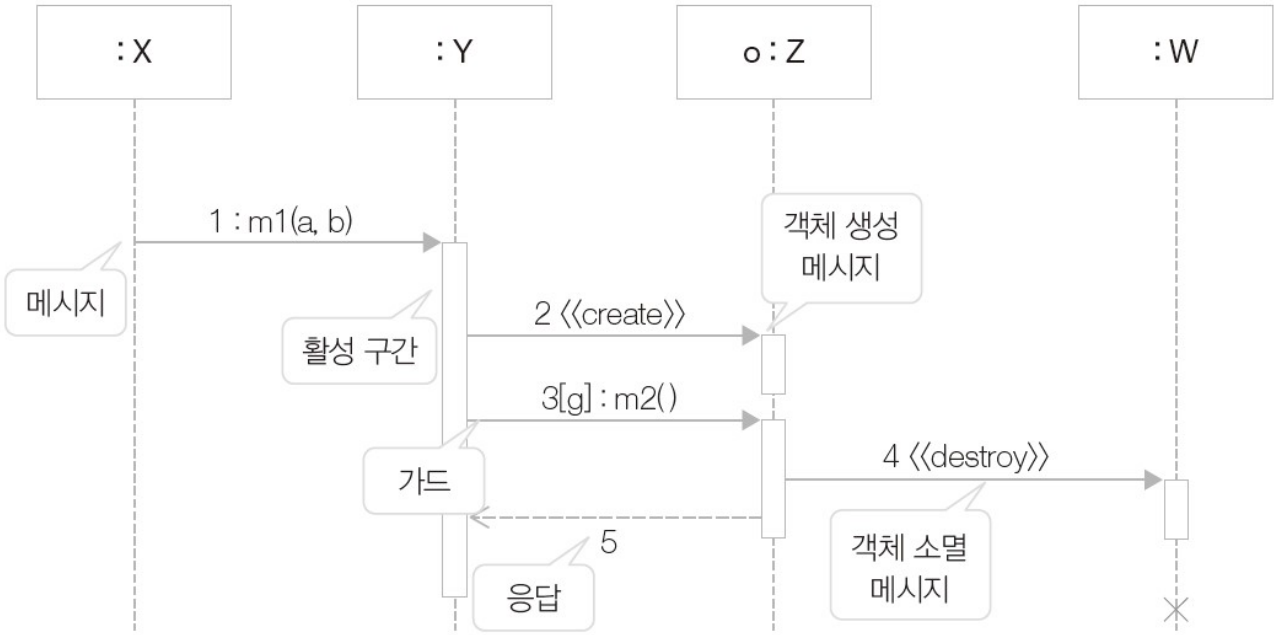


그림 4-8 여러 가지 형태의 메시지 표현




```
public class Y {
  public void m1(T1 a, T2 b) {
    Z o = new Z();
    if (g)
      o.m2();
  }
}

public class Z {
  private W w;

  public void m2() {
    w = null;
  }
}
```

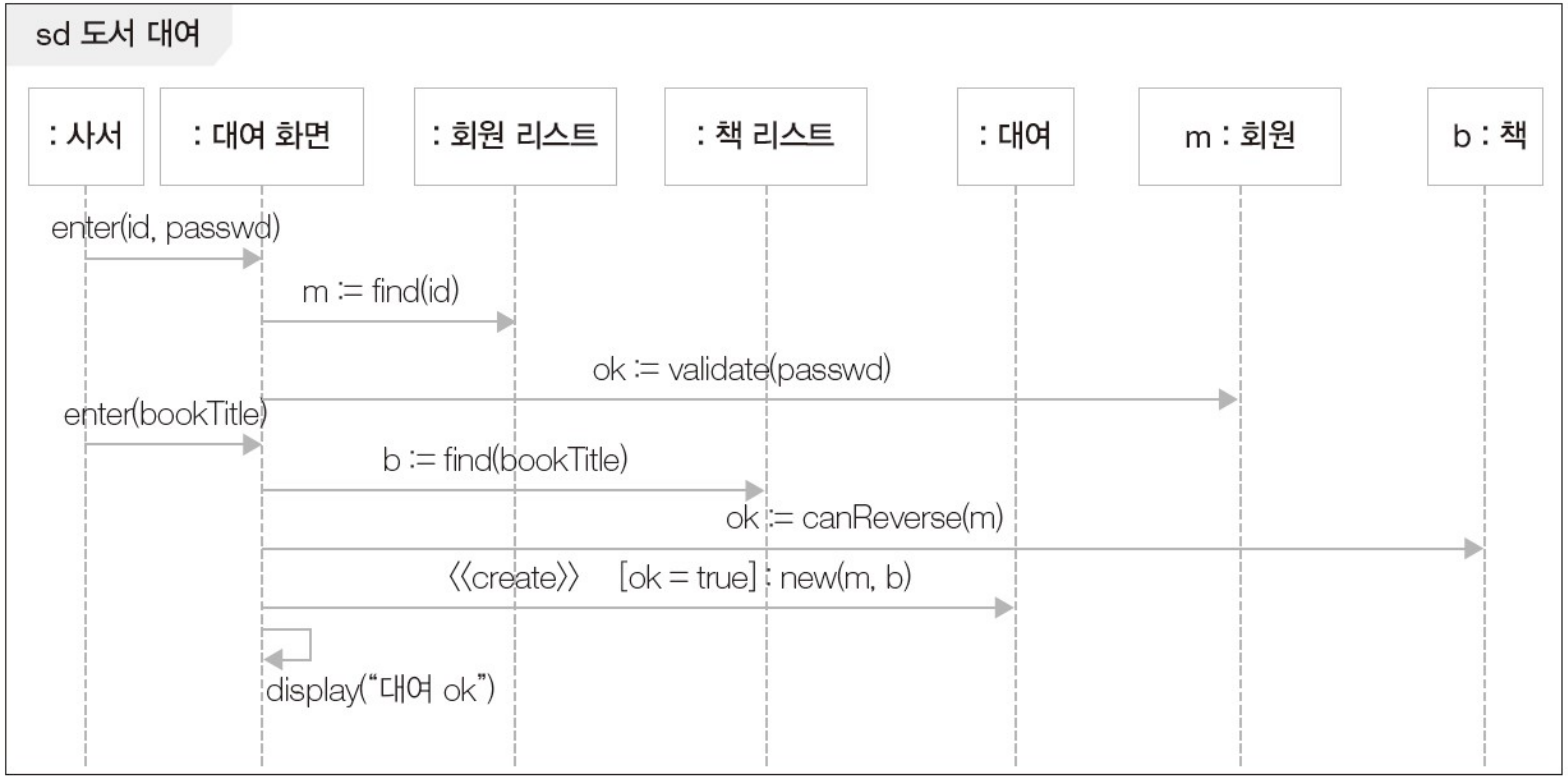
프레임

❖ UML 2.0의 프레임

- 모든 다이어그램에 다이어그램의 경계, 타입, 이름을 포함한 레이블의 장소를 제공하는 프레임을 제공
- 다이어그램을 에워싸는 박스
- 박스 안쪽 모서리에 다이어그램과 이름을 넣음
- 순차 다이어그램의 타입: sd
- 유즈케이스 다이어그램 타입 : uc
- 액티비티 다이어그램 : act

❖ 그림 4-9. 도서관에서 회원들에게 도서를 대여하는 과정

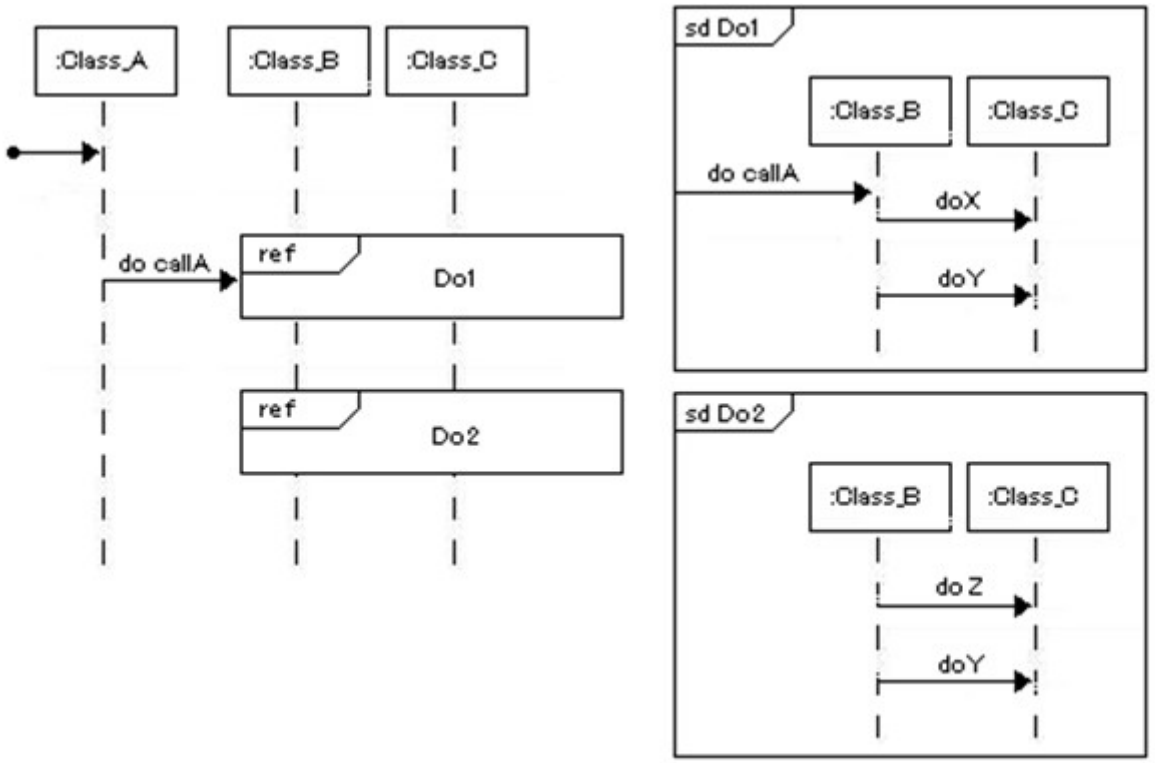
그림 4-9 프레임을 사용한 순차 다이어그램



❖ 프레임의 장점

- 다이어그램 외부에서 특정 다이어그램을 참조하는 것이 쉬워짐
- 일부분이 다른 곳에서 재사용될때
- 매우 복잡한 상호작용을 표시할때 분리해서 작성후 참조할수 있도록
- Ref : 다른 순차 다이어그램을 참조….

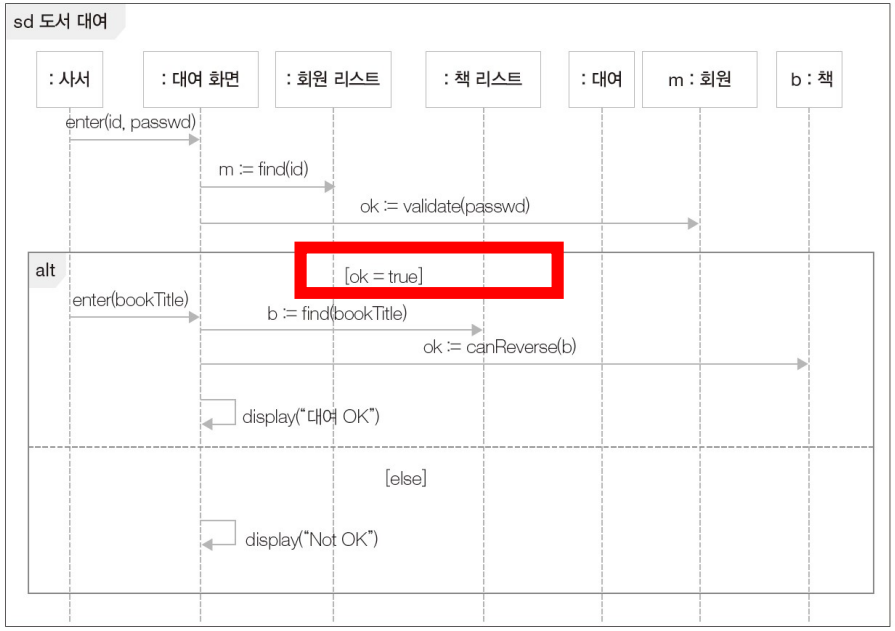
참조 : ref


: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

❖ 그림 4-10. alt 키워드

- 그림 4-9의 도시 대여 시나리오에 , 올바르지 않은 비밀번호를 입력한 경우를 고려해 확장한 순차 다이어그램
- Alt 키워드를 사용해, 상호작용을 조건에 따라 선택적으로 수행하도록 함
- 조건은 프레임의 윗부분에 명시적으로 표현

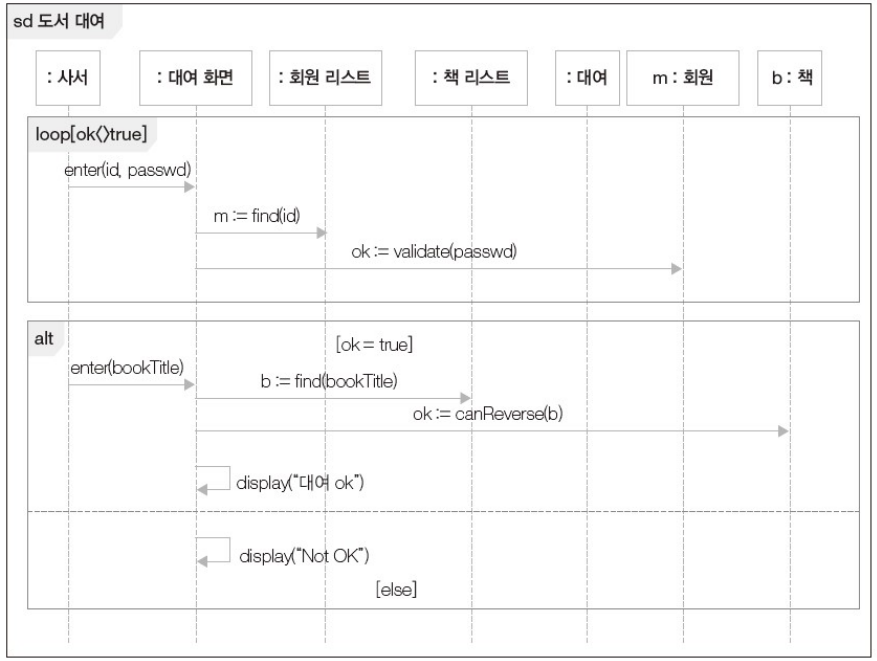


loop

❖ Loop 키워드

- 반복적인 상호작용을 나타냄
- 그림 4-11. 올바르지 않은 비밀번호를 입력했을 때 올바른 비밀번호를 입력하도록 도서대여 시나리오를 변경한 그림
- 반복 조건은 [가드] 형식으로 나타냄

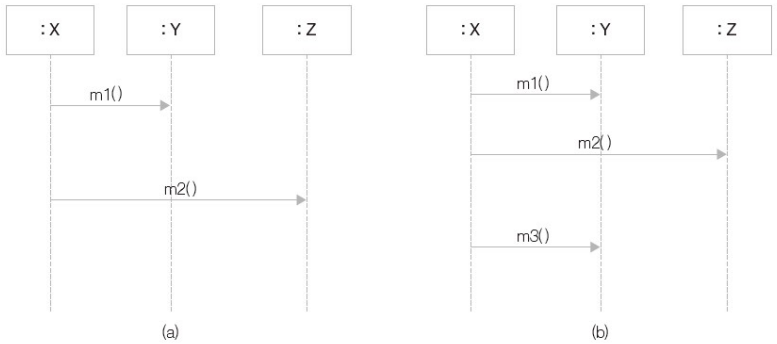
그림 4-11 loop 키워드



Keypoint_ 이 책에서는 언급하지 않았지만 유용한 상호작용 연산으로는 다음과 같은 것이 있다.

- opt: 특정 조건에서만 상호작용을 선택적으로 수행한다.
- par: 동시에 실행되는 상호작용을 수행한다.
- break: C 프로그래밍 언어의 break 키워드와 같이 특정 상호작용 그룹을 빠져나갈 때 사용한다. 하지만 break 연산자 부분은 수행한다.

그림 4-12 연관이나 의존 관계가 존재하지 않는 순차 다이어그램



❖ 그림 4-12

- **(a), (b) 모두 클래스 x의 인스턴스가**
  - y와 z의 인스턴스에 메시지를 보낼수 있어야 함
- **연관 또는 의존관계**
- **시스템 설계 초기 단계에서는 연관관계로 표시**
- **구현 직전의 단계에서는 구별해주는 것이 바람직**

- **(b)**
  - M3() 메시지를 더 송신
  - 클래스 x의 인스턴스 입장에서는 m1 메시지를 받는 클래스 y의 인스턴스를 기억할 필요
  - 클래스 x에서는 지역변수나 인자형태가 아닌, 클래스의 속성을 이용해 클래스 Y의 인스턴스를 저장할 필요
  - 이는 클래스 X와 클래스 Y가 연관관계가 형성
- **(a)**
  - Y와 Z의 인스턴스들이 m1과 m2 메시지를 처리한후 더는 필요없다고 생각함
  - 따라서 의존관계

4.3.3 순차 다이어그램과 클래스 다이어그램의 관계

그림 4-12 연관이나 의존 관계가 존재하지 않는 순차 다이어그램

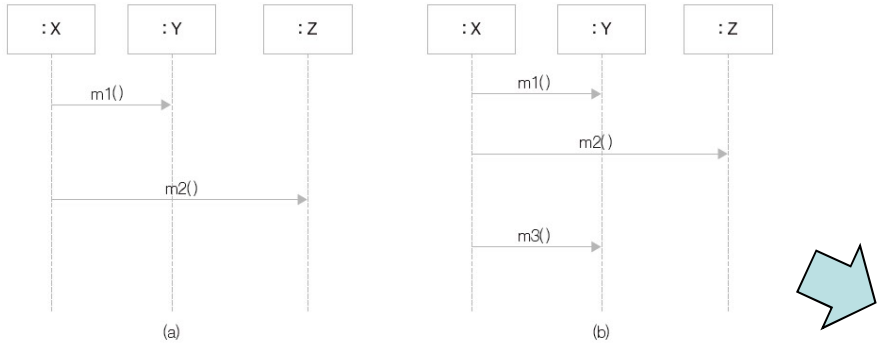
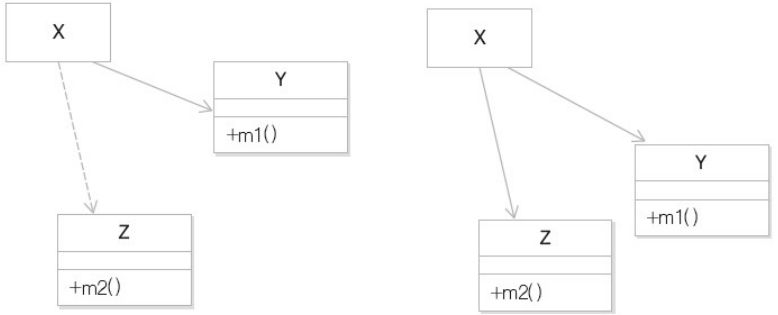


그림 4-13 그림 4-12를 순차 다이어그램으로 수정



순차 다이어그램 (a)에 대응하는 클래스 다이어그램

순차 다이어그램 (b)에 대응하는 클래스 다이어그램