# Chap 05 요구 모델링

# 목 차

# 요구 모델링

- 고객과 개발자가 무엇이 개발되고 있는지에 동의하는 것을 주된 목적으로 하는 요구 명세를 생성
- 시스템에 대한 형식적 또는 준형식적 설명을 제공

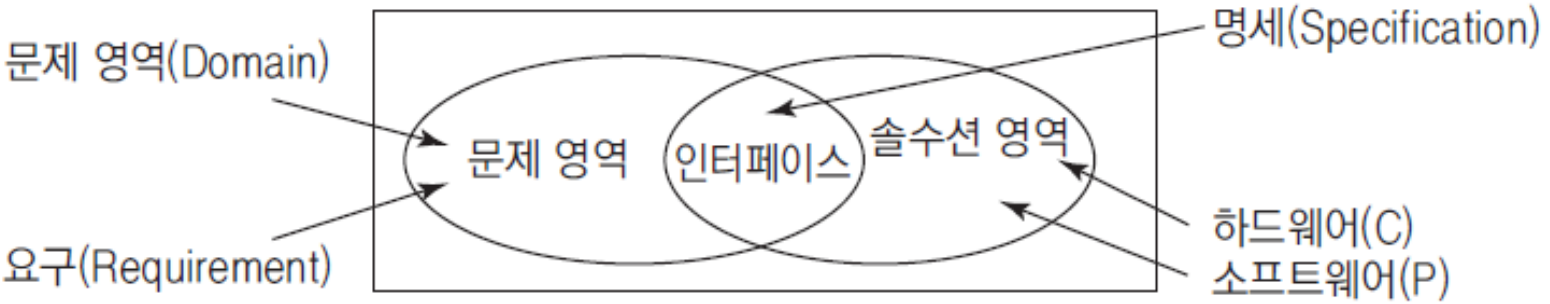



그림 5.1 문제 영역과 솔루션 영역

# 5.1 모델링 기초

- 복잡한 시스템을 다루는 방법
  - 전체를 다루기에는 너무 복잡한 대상을 추상화 또는 단순화
- 모델링을 하는 이유

  (1) 복잡함을 잘 관리하기 위하여

  (2) 형체가 없는 소프트웨어의 구조를 시각화 하기 위하여

  (3) 다른 사람과 커뮤니케이션 하기 위하여

  (4) 문제 도메인 및 제품 요구 사항을 이해하기 위하여

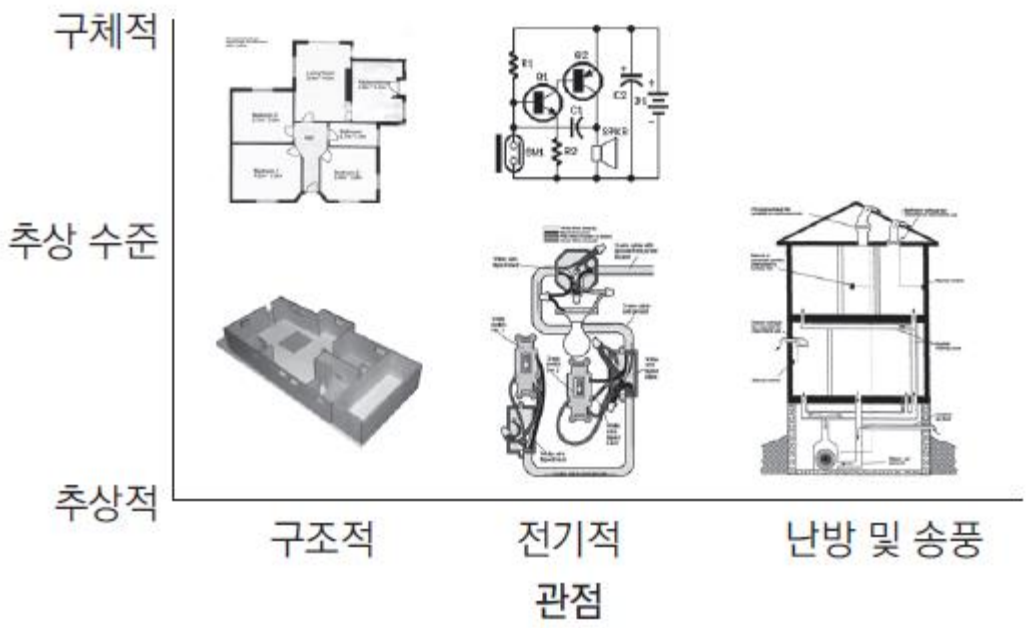
  (5) 개발 중인 시스템을 이해하기 위하여

  (6) 구현하기 전에 잠재적 솔루션을 실험해보기 위하여

  (7) 기존 시스템의 문서화

# 관점과 추상화 수준

- 모델은 특정 관점(perspective)과 추상화 수준(abstraction level)에 따라 달라짐



# 소프트웨어와 모델링

- 그래픽 기호와 주석으로 구성된 시각적 다이어그램

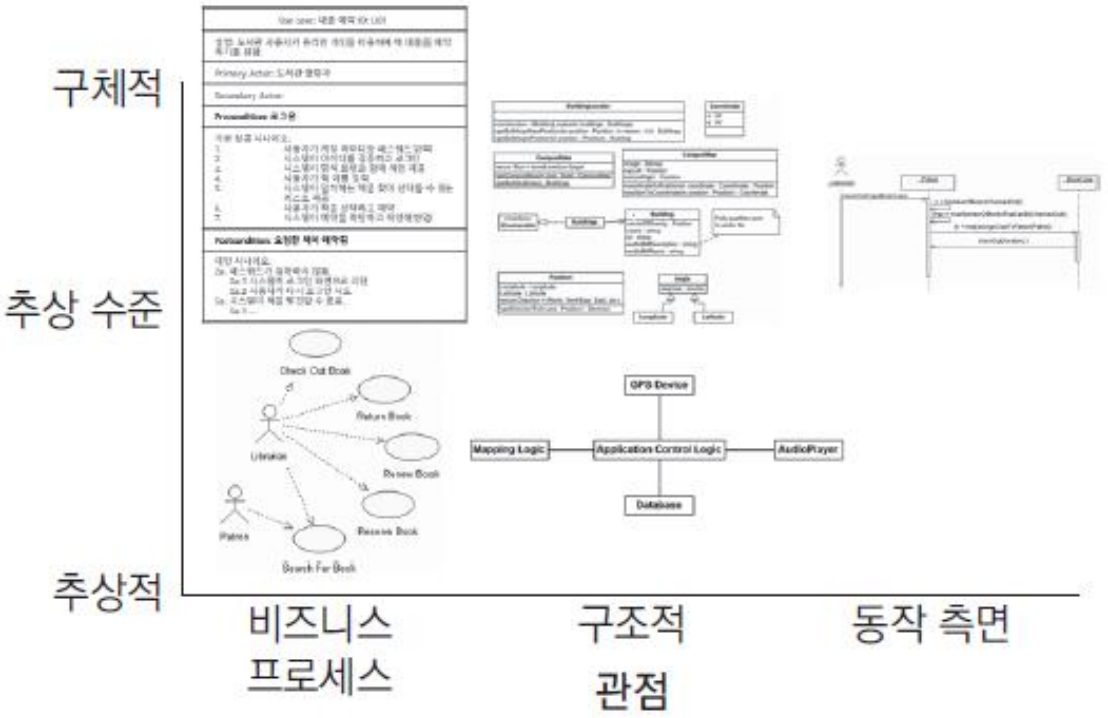



그림 5.6 소프트웨어 모델링의 관점과 추상 수준

# 모델 사이의 관계

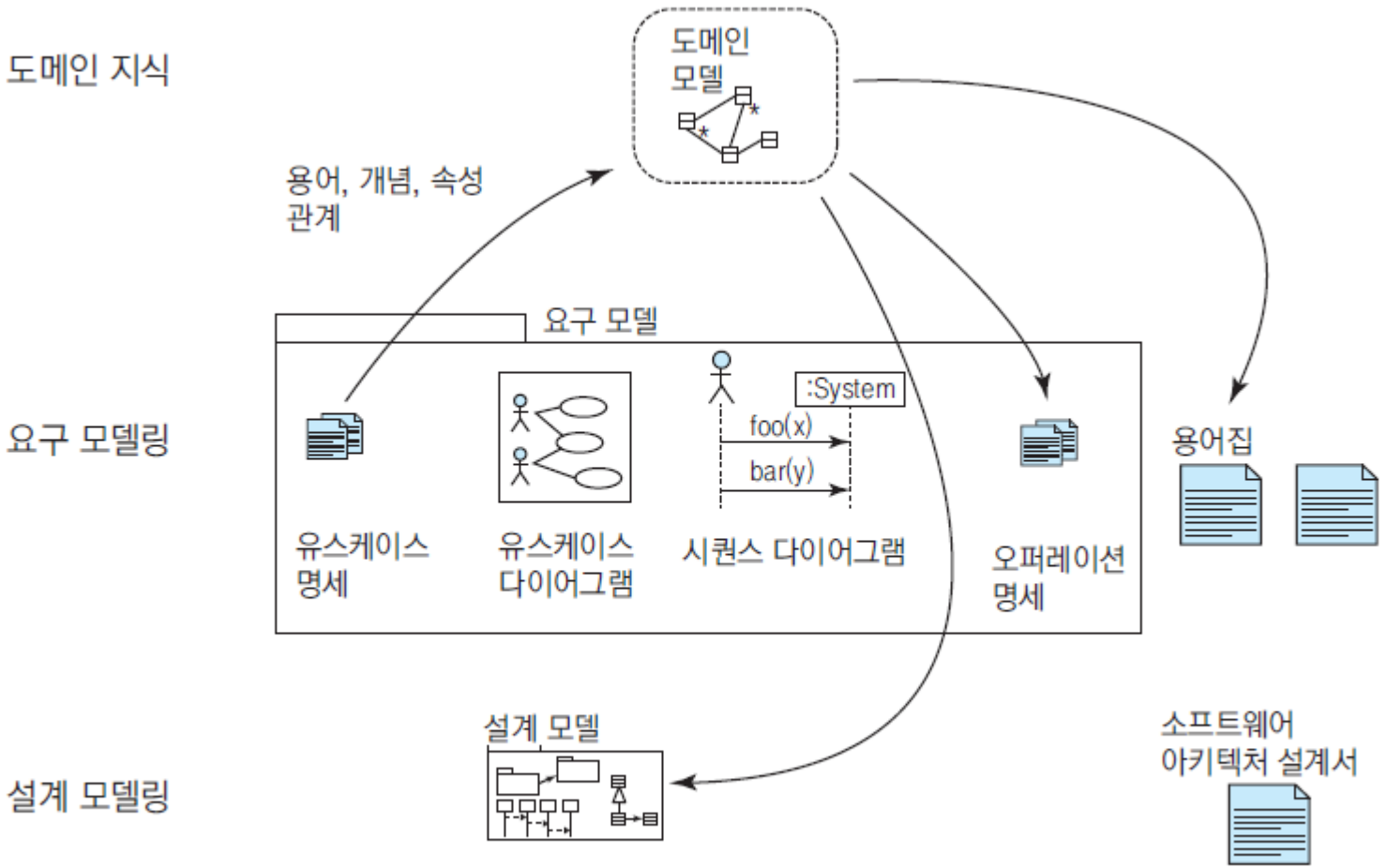



그림 5.7 모델 간의 영향

# 5.2 UML(Unified Modeling Language)

- 객체지향 소프트웨어를 모델링 하는 표준 그래픽 언어
  - 시스템의 여러 측면을 그림으로 모델링
  - 하드웨어의 회로도 같은 의미

- UML은 소프트웨어 모델링의 공통 언어

# UML의 역사

- UML은 OMT(Object Modeling Technique) [James Rumbaugh, 1991]와 Booch[Grady Booch,1994], OOSE(Object-Oriented Software Engineering) [Ivar Jacobson, 1992] 방법의 통합으로 만들어진 표현

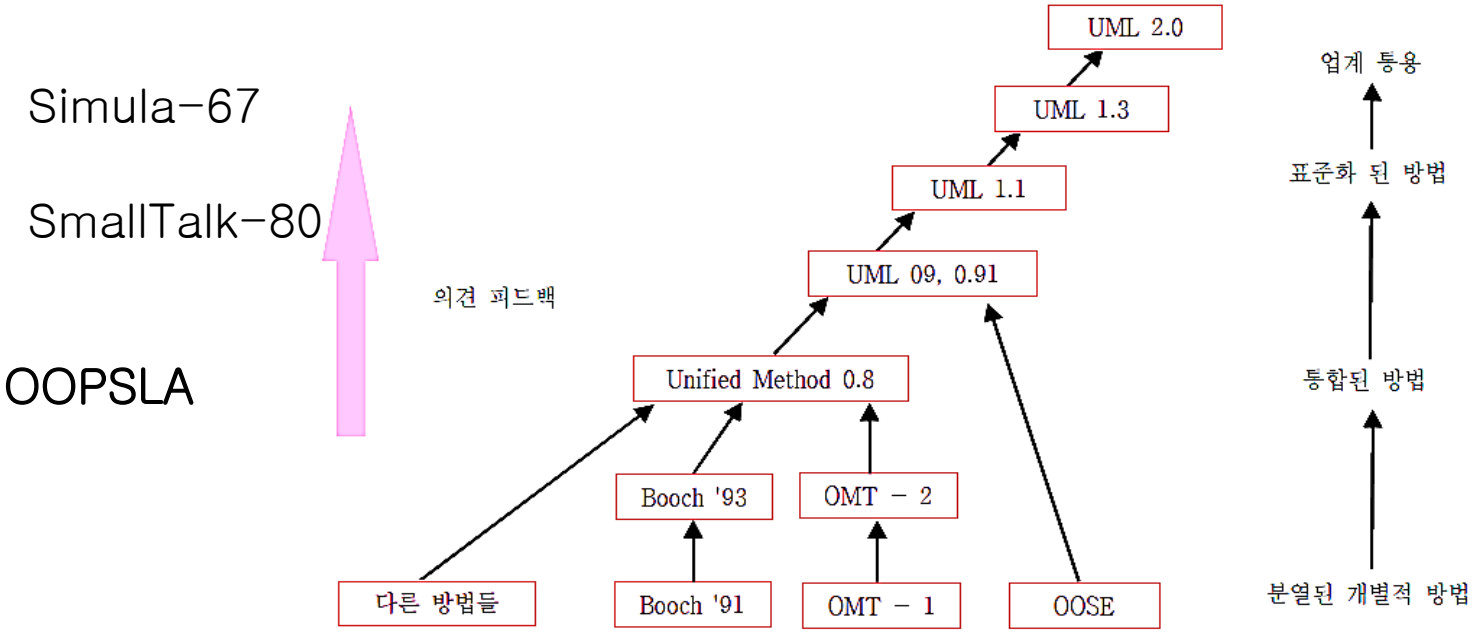
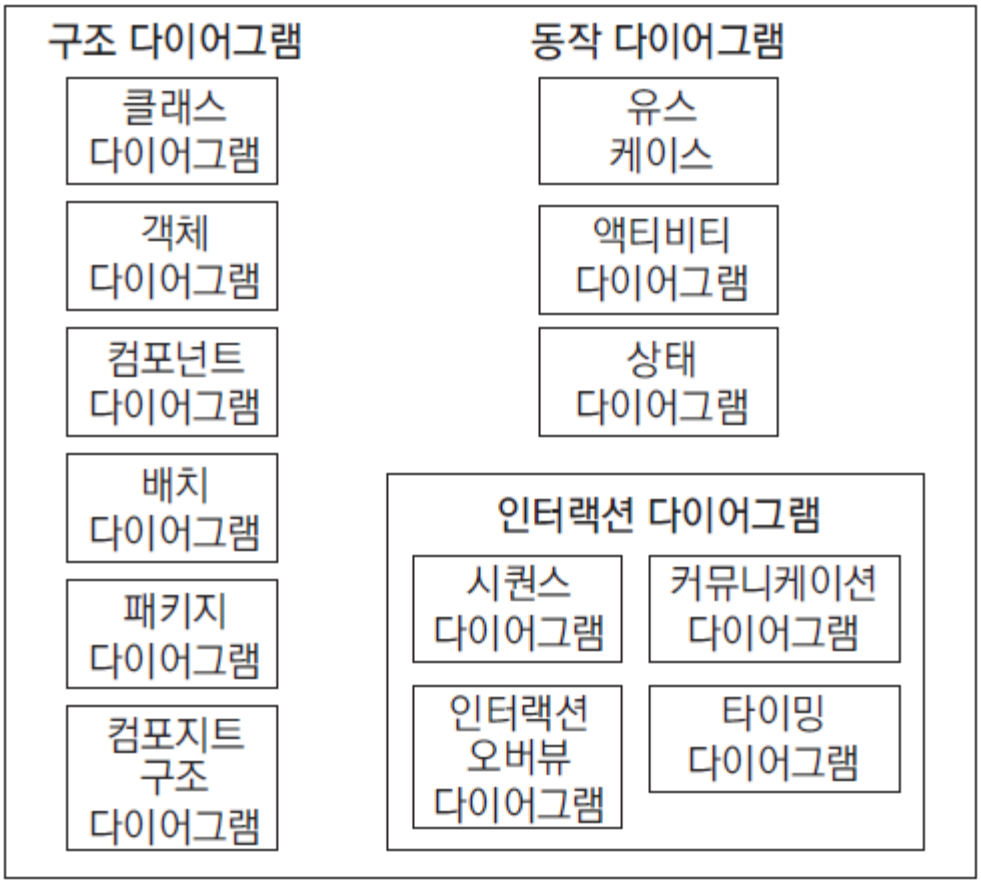



그림 5.9 ▶ UML의 진화

# UML 다이어그램

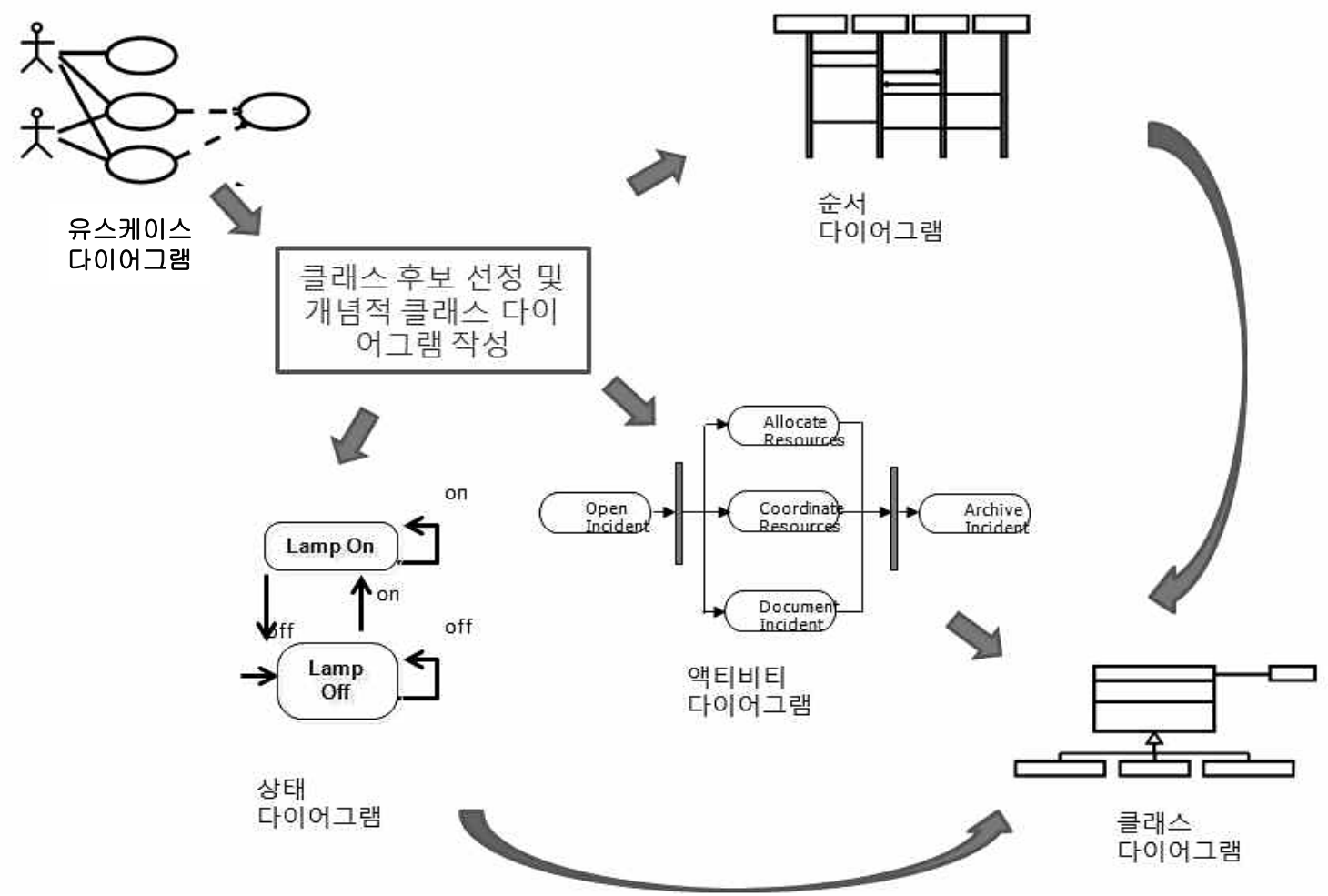
- 시스템의 모델링은 기능적 관점, 구조적 관점, 동적 관점으로 구성



# UML 모델링 과정

1. 요구를 유스케이스로 정리하고 유스케이스다이어그램을 작성
2. 클래스 후보를 찾아내고 개념적인 객체 모델(도메인모델)을 작성
3. 유스케이스를 기초하여 시퀀스 다이어그램을 작성
4. 클래스의 속성, 메소드 및 클래스 사이의 관계를 찾아 객체 모델(도메인모델)을 완성 - 클래스다이어그램
5. 상태 다이어그램이나 액티비티 다이어그램 등 다른 다이어그램을 추가하여 UML 모델을 완성
6. 서브시스템을 파악하고 전체 시스템 구조를 설계
7. 적당한 객체를 찾아내거나 커스텀화 또는 객체를 새로 설계

# UML 모델링 과정

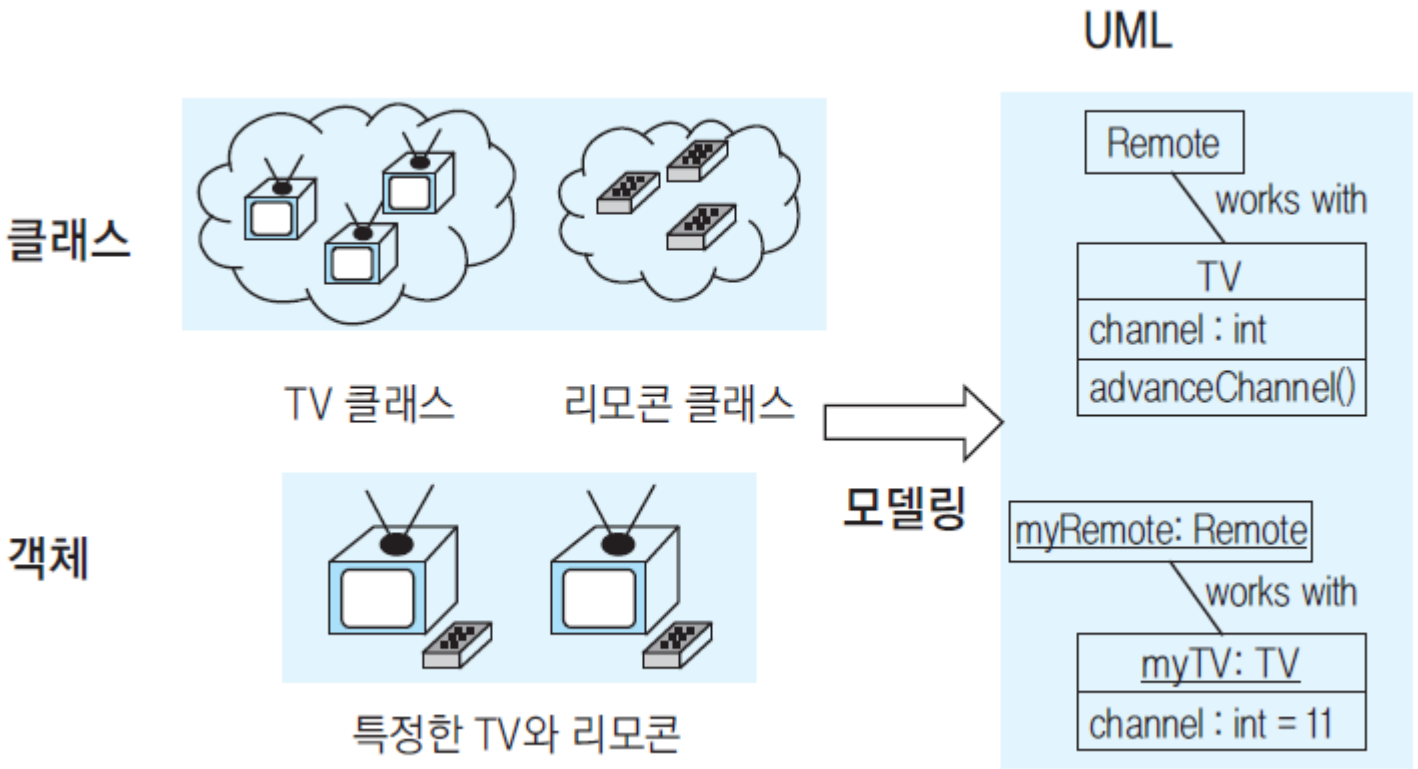

유스케이스
다이어그램
순서
다이어그램
클래스 후보 선정 및
개념적 클래스 다이
어그램 작성
Allocate
Resources
Open
Incident
Coordinate
Resources
Archive
Incident
Document
Incident
액티비티
다이어그램
on
Lamp On
on
off
off
Lamp
Off
상태
다이어그램
클래스
다이어그램

# 5.3 정적 모델링

- 정적 모델
  - 객체들의 공통 구조와 동작들을 추상화 시킨 것
- 객체지향 기본 개념의 이해가 필요
  - 객체와 속성, 연관, 집합, 상속, 다형성
- 클래스 다이어그램이 대표적
  - 클래스 및 클래스 사이의 관계를 표현
  - 도메인 개념과 속성

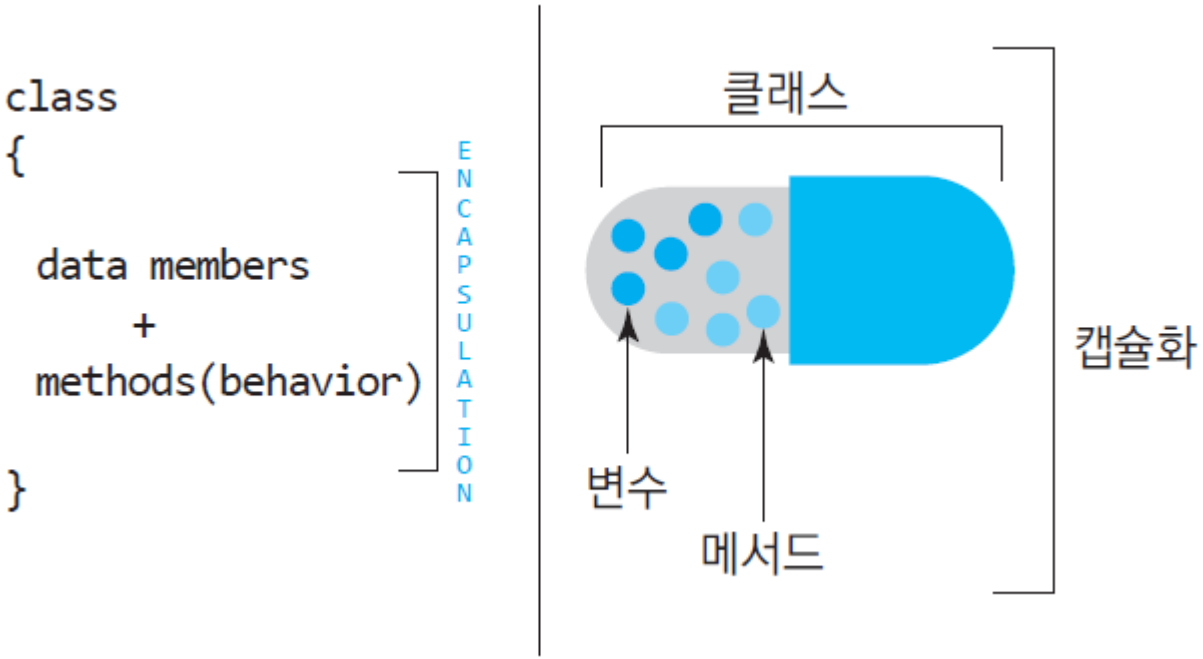
# 객체와 클래스

- 객체: 상태, 동작, 고유 식별자를 가진 모든 실체
- 클래스: 공통 속성을 공유하는 객체 집합에 대한 정의



# 캡슐화

- 객체의 속성 부분과 행위(메소드) 부분을 하나로 모아서 단위화
- 정보은닉(Information Hiding)



# 연관

- 연관
  - 서비스를 제공하는 객체와 서비스를 요청하는 객체가 상호작용하는 관계

- 가시성
  - 객체의 접근 가능성

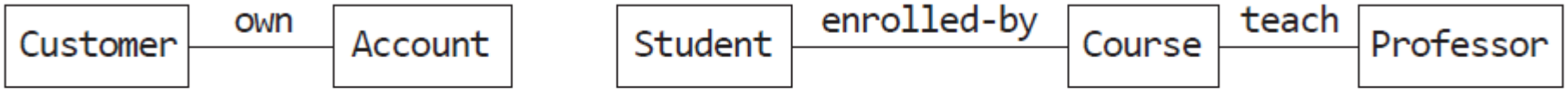
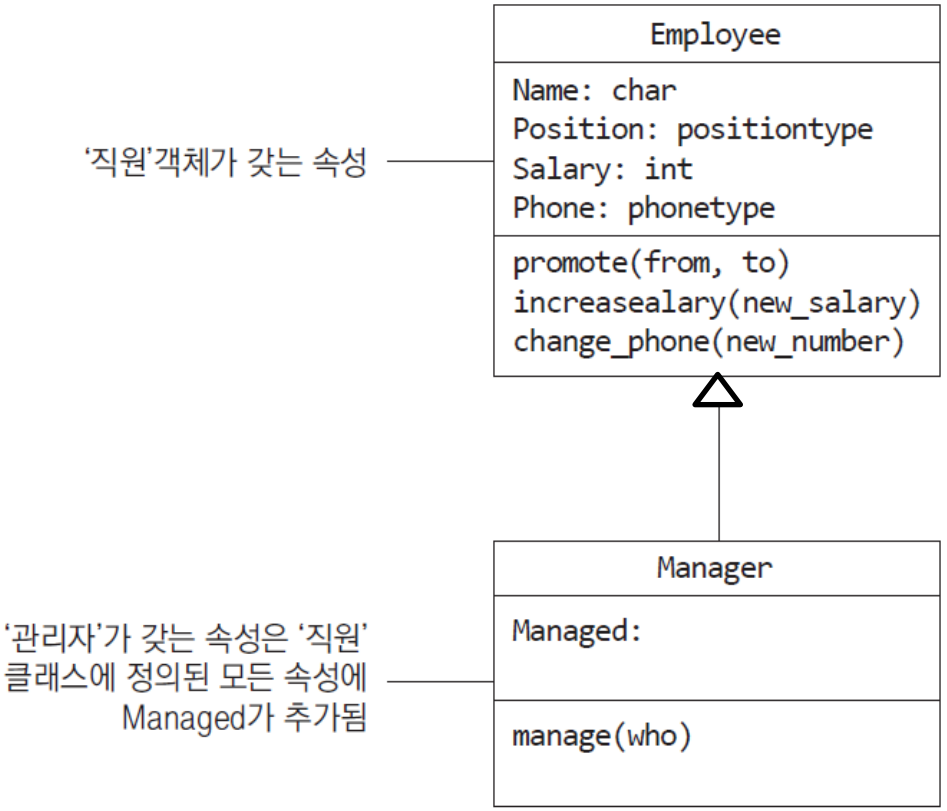



그림 5.14 연관의 사례

# 상속

- 일반화된 클래스가 갖는 속성과 연산을 하위 개념의 구체화된 클래스가 그대로 물려받는 것



# 다형성

- 같은 이름의 메시지를 다른 객체 또는 서브 클래스에 호출할 수 있는 특징
  - twoDShape.getArea():

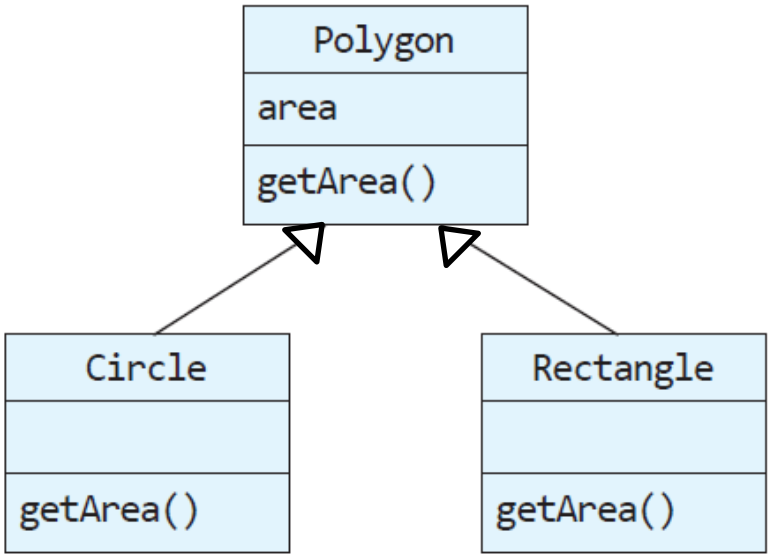


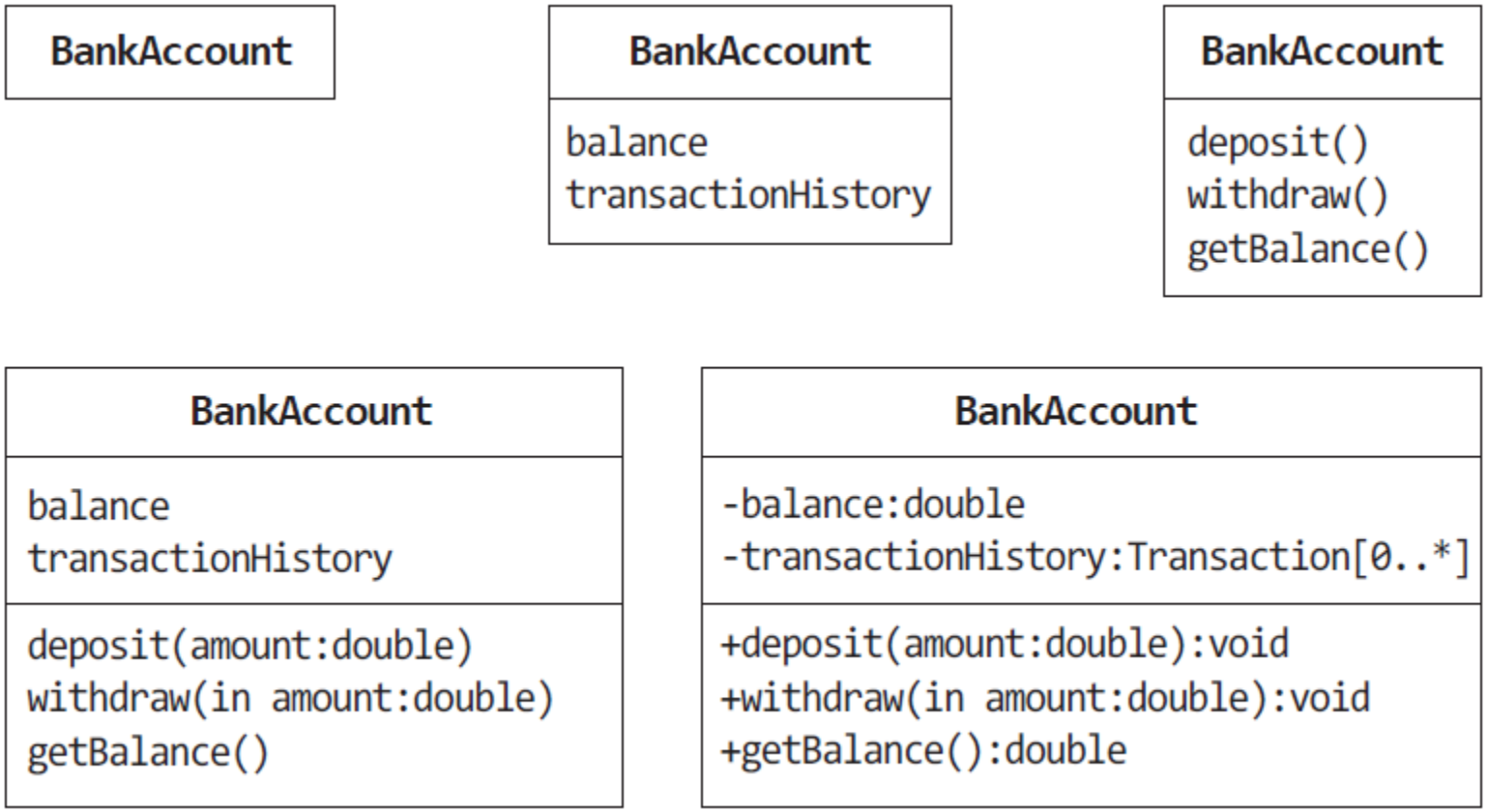

그림 5.17 다형성의 예

# 클래스의 표현

- 클래스 심볼
  - 세 개의 부분으로 나누고
  - 클래스의 이름, 중간에는 클래스의 속성, 아래 부분은 메소드를 적음
  - 추상클래스는 이탤릭체, 인터페이스 클래스는 <<interface>>추가
- 속성 : 객체가 가지는 모든 필드를 포함
- 메소드
  - 아주 흔한 메소드(get/set)는 생략

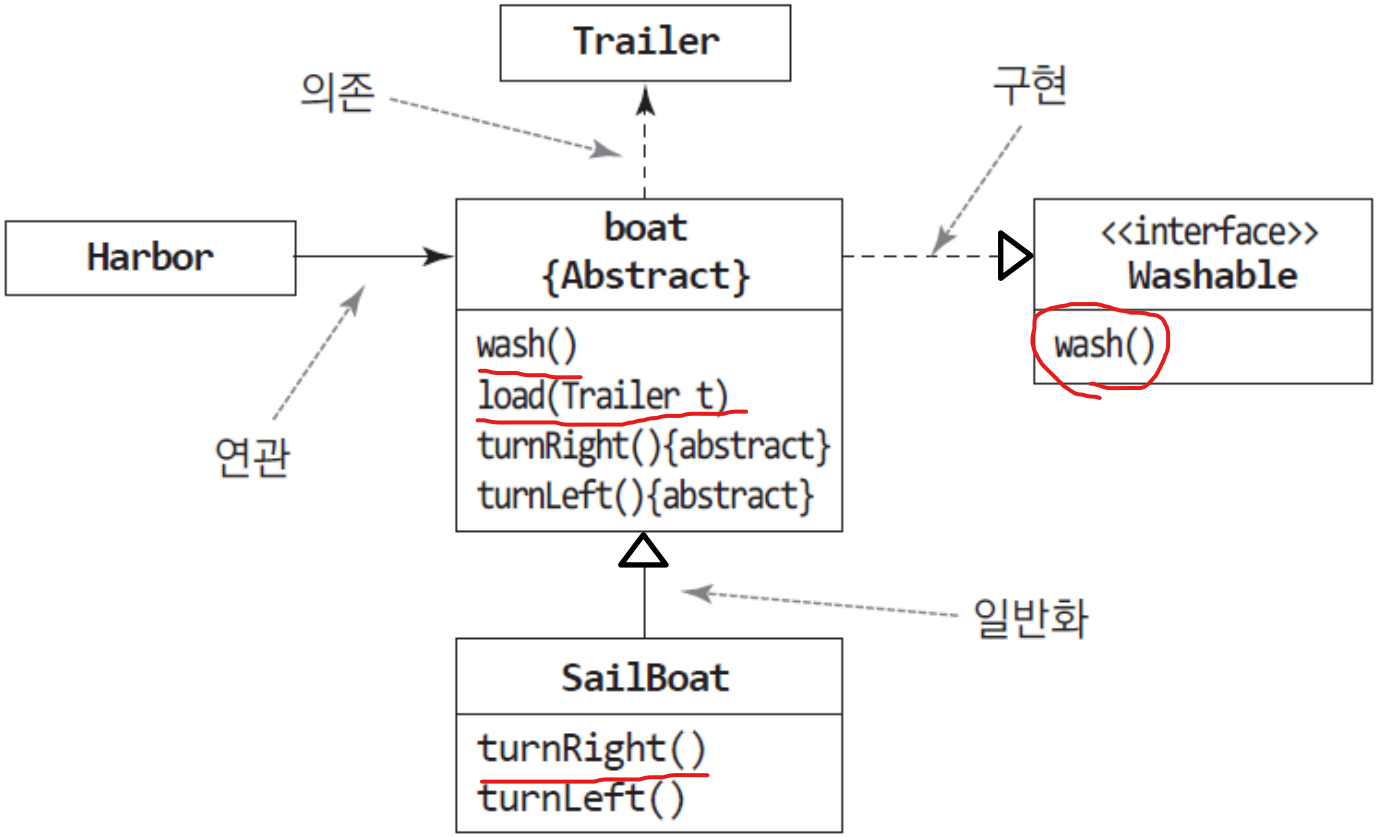
# 표현 수준

- 모델링이 진행되면서 상세화



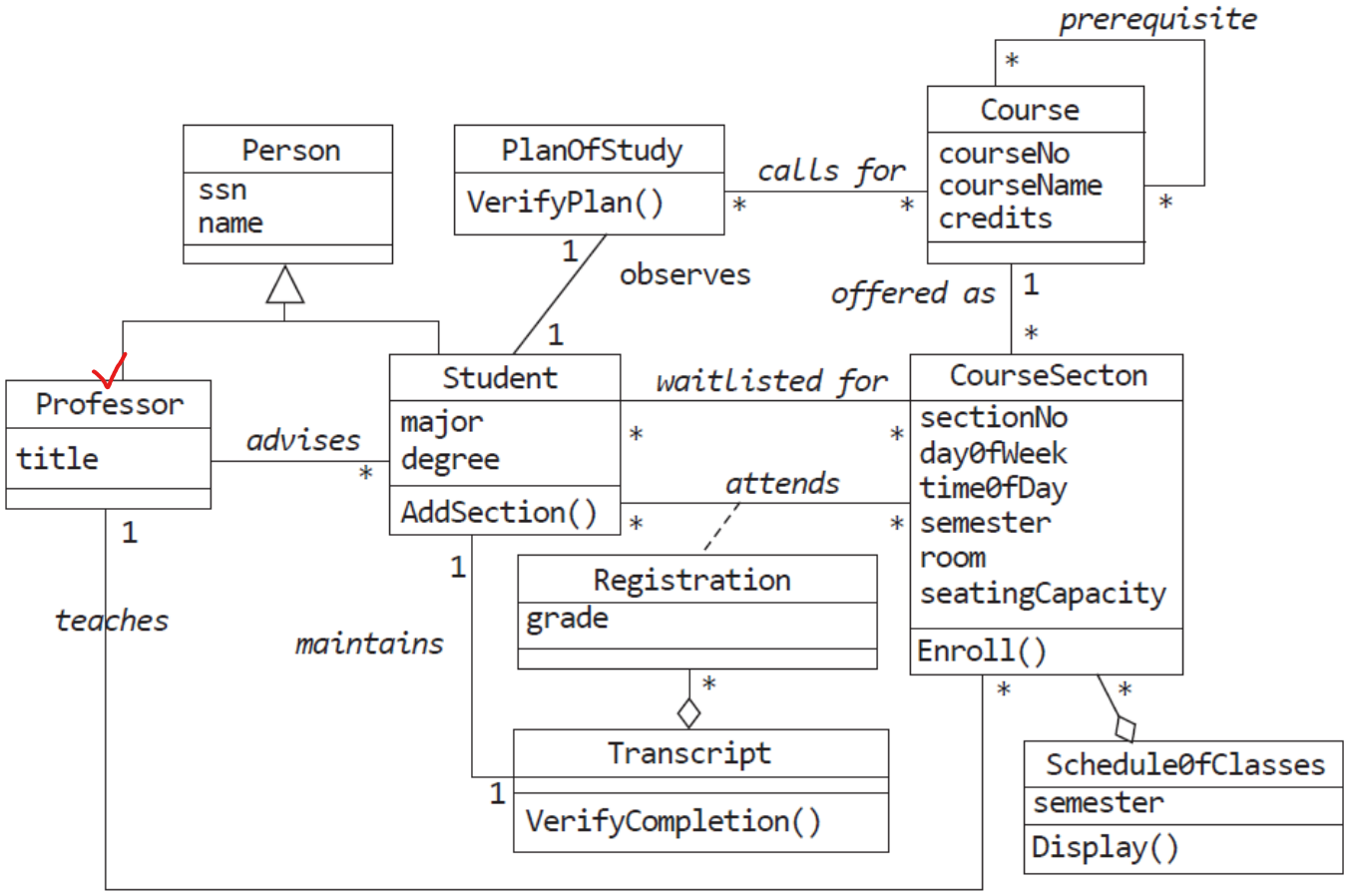
# 관계의 표현

(1) 연관
(2) 상속
(3) 의존
(4) 구현



# 사례

- 수강 신청 시스템

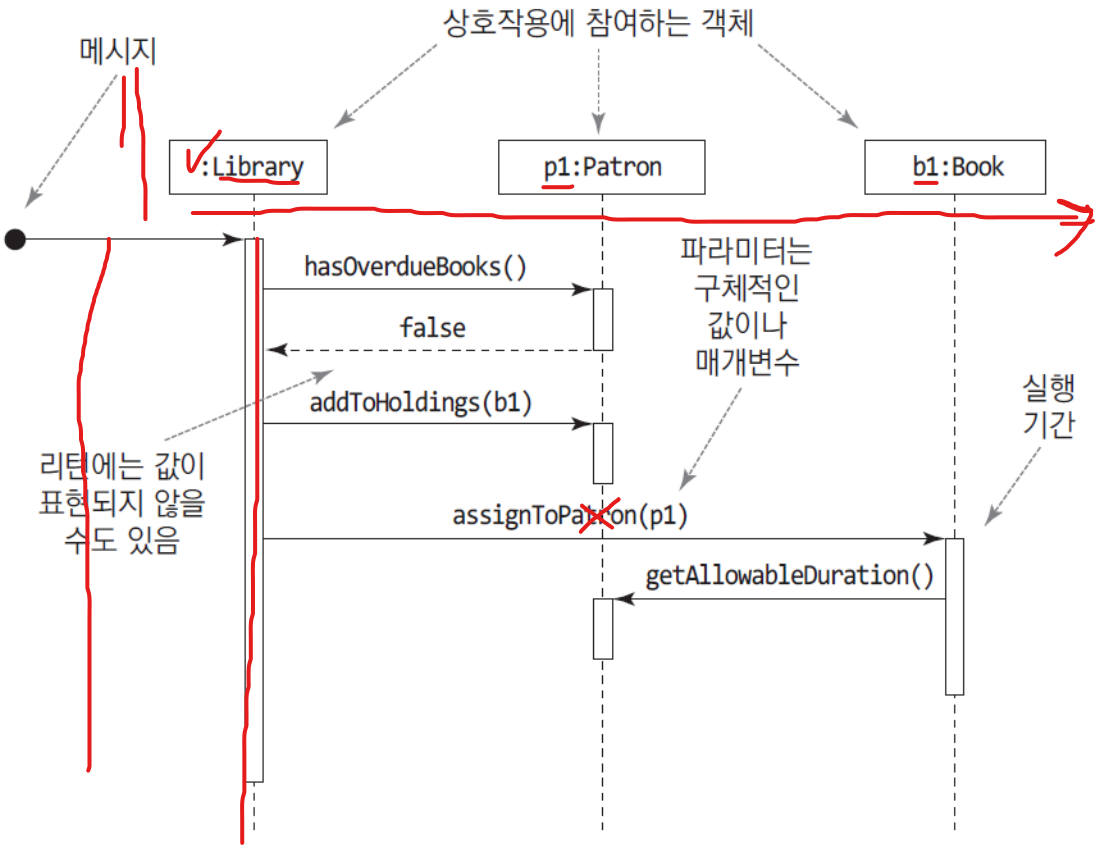

prerequisite
Person
ssn
name
PlanOfStudy
VerifyPlan()
calls for
Course
courseNo
courseName
credits
observes
offered as
Professor
title
advises
Student
major
degree
AddSection()
waitlisted for
CourseSecton
sectionNo
day0fWeek
time0fDay
semester
room
seatingCapacity
Enroll()
attends
Registration
grade
teaches
maintains
Transcript
VerifyCompletion()
Schedule0fClasses
semester
Display()

# 5.4 동적 모델링

- 동적 측면
  - 소프트웨어가 실행될 때 변경 될 수 있는 뷰
  - 시간의 함수로만 이해
  - 예: 객체 간 상호작용 패턴
- 정적 다이어그램을 보완
- 상호작용 다이어그램
  1. 시퀀스 다이어그램
  2. 협동 다이어그램

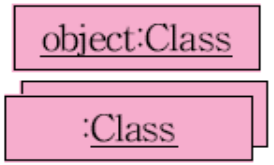
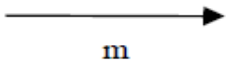
# 시퀀스 다이어그램

- 시스템의 동작을 정형화하고 객체들의 메시지 교환을 울타리 형태로 시각화하여 나타낸 것



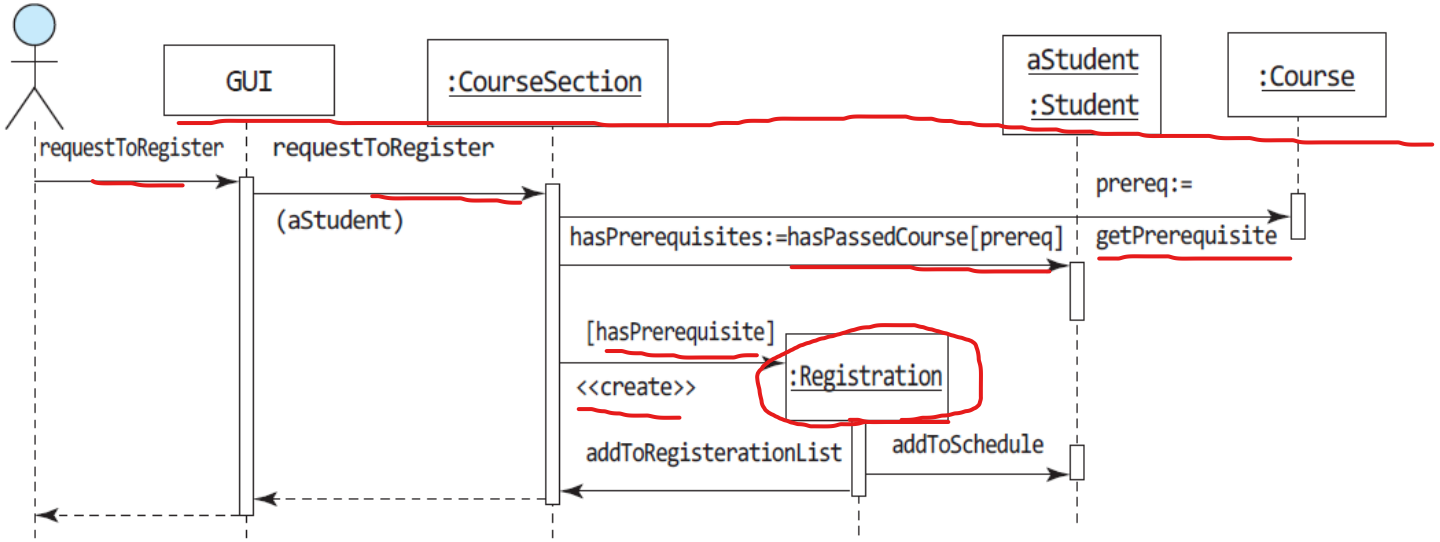
# 시퀀스 다이어그램의 요소

표 5.3 ▶ 시퀀스 다이어그램의 기본 요소

| 요소 | 표현방법 | 의미 | 연결 |
|---|---|---|---|
| 객체<br>객체집합 |  | - 특정 클래스의 객체<br>- 객체의 모임<br>- 라이프라인 위에 위치하며 콜론 앞은 객체의 이름, 뒤는 클래스의 이름 | 라이프라인 사이에 활성막대와 연결됨 |
| 라이프라인 |  | 객체가 시스템에 존재하나 아직 실행되지는 않음을 의미. | 객체를 활성막대와 연결시키며 두 개의 이웃 라이프라인을 연결 |
| 활성막대 |  | 시스템에 존재하는 메소드가 막대의 길이만큼 실행됨을 의미. 점선은 라이프라인임. | 객체와 연결됨. 라이프라인과 연결됨 |
| 객체 소멸 |  | 라이프라인 맨 위에 연결된 객체가 소멸됨을 의미. | |
| 메시지 호출 |  | 한 객체에서 다른 객체로 메시지를 보냄을 의미. 즉 함수가 호출됨 | 상호작용하는 두 객체를 연결함 |
| 프레임 |  | 시퀀스 다이어그램의 일부로 반복 또는 택일 구조의 묶여진 조각. | |

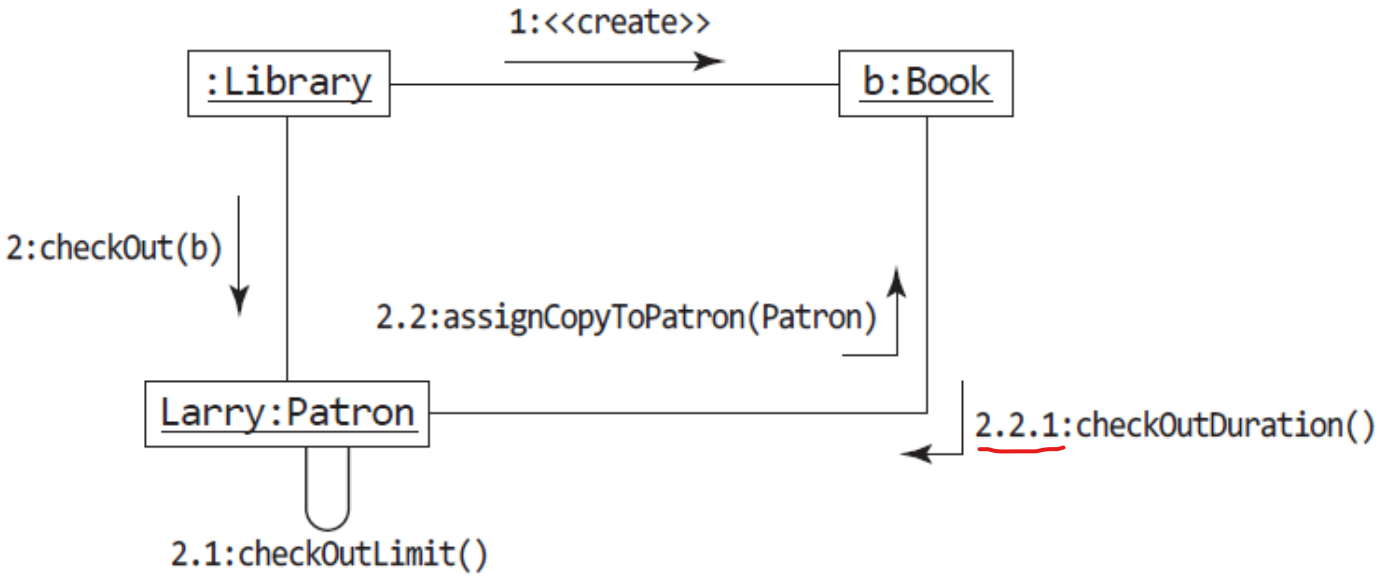
# 시퀀스 다이어그램 작성 과정

- Step 1. 참여하는 객체를 파악
- Step 2. 파악한 객체를 X축에 나열하고 라이프라인을 그음
- Step 3. 유스케이스에 기술된 이벤트 순서에 따라 객체의 메시지 호출을 화살표로 나타냄
- 사례: 수강신청 유스케이스



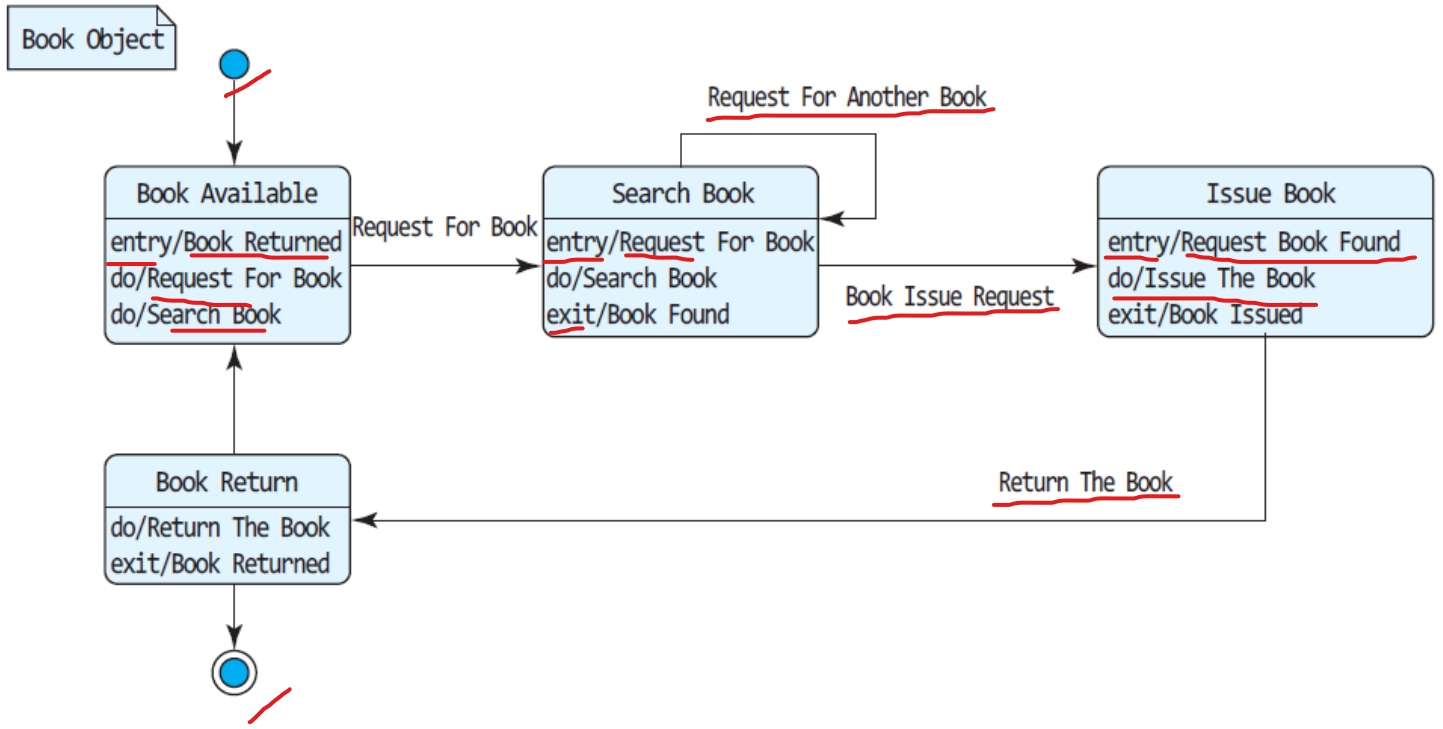
# 협동 다이어그램

- 두 가지 조합
  - 상호작용에 필요한 객체들 간의 링크를 포함한 객체 다이어그램
  - 상호작용을 정의하는 객체 간의 메시지



# 상태 다이어그램

- 동작을 수신 이벤트와 이에 대한 응답을 기반으로 상태 사이의 전환으로 모델링
- 예: 도서관 시스템에서 책의 상태



# 5.5 제어 모델링

- **액티비티 다이어그램**
  - 액티비티 사이의 제어흐름을 보여 주는 일종의 흐름도
- 액티비티
  - 계산 또는 프로세스
- 전환
  - 액티비티에서 다른 액티비티로 제어가 넘어감
- 분기
  - 진위 조건

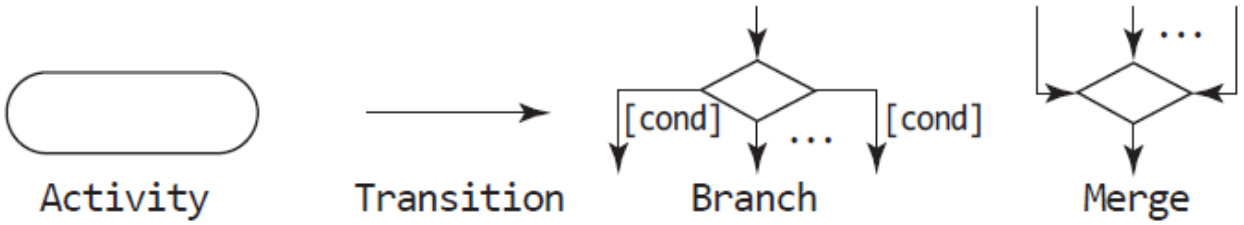
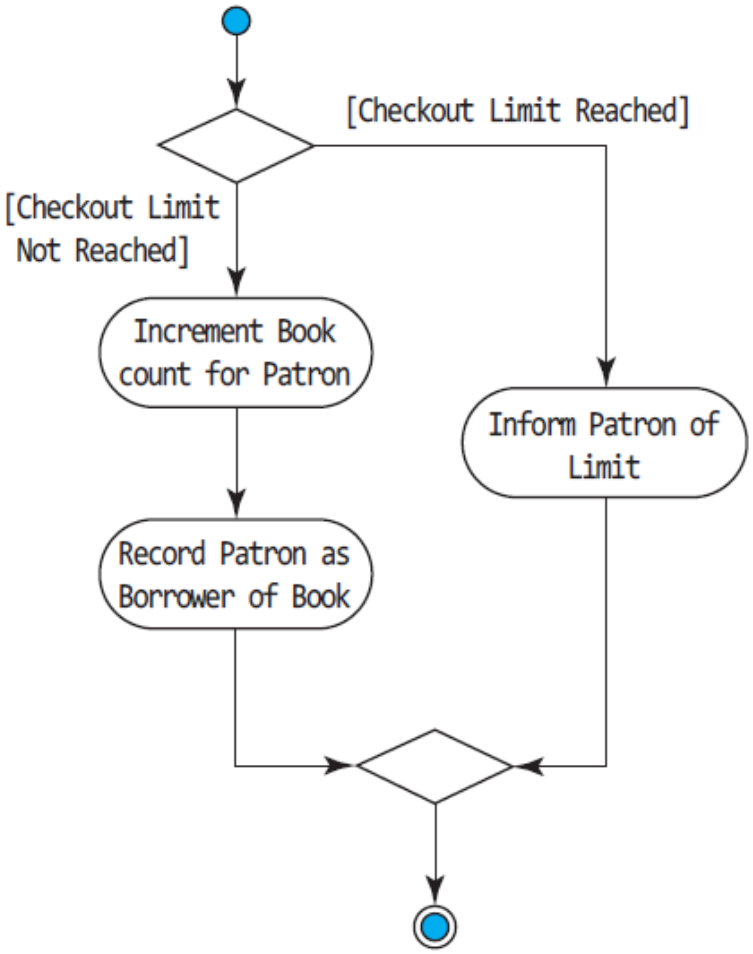



그림 5.38 액티비티 다이어그램의 기본 요소

# 액티비티 다이어그램



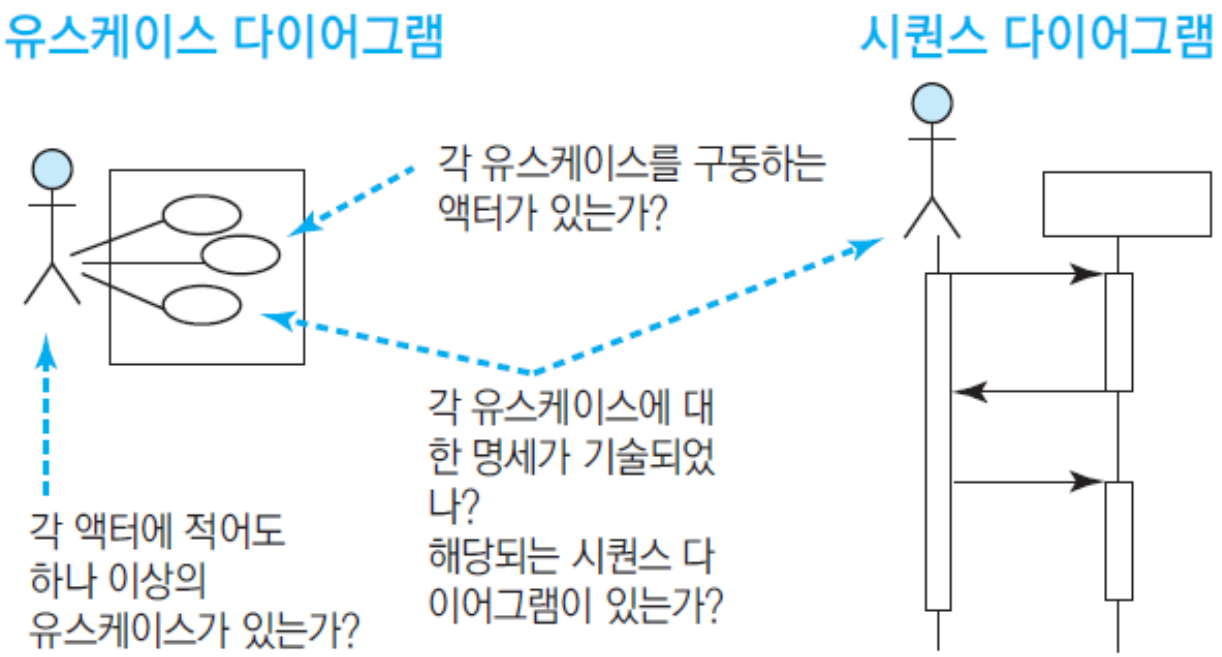

```
class Patron {
  int booksCheckedOut;
  . . .
  public void checkOutBook(Book b) {
    if (booksCheckedOut < limit()){
      booksCheckedOut++;
       b.assignCopyToPatron(this);
    }
    else
      WriteLn("Failed");
    }
  }
}
```

# 5.6 모델 검증

- 모델 검증 방법
  - 리뷰: 워크스루, 인스펙션
  - 테스팅
  - 정형적 방법
  - 프로토타이핑
  - 요구 추적

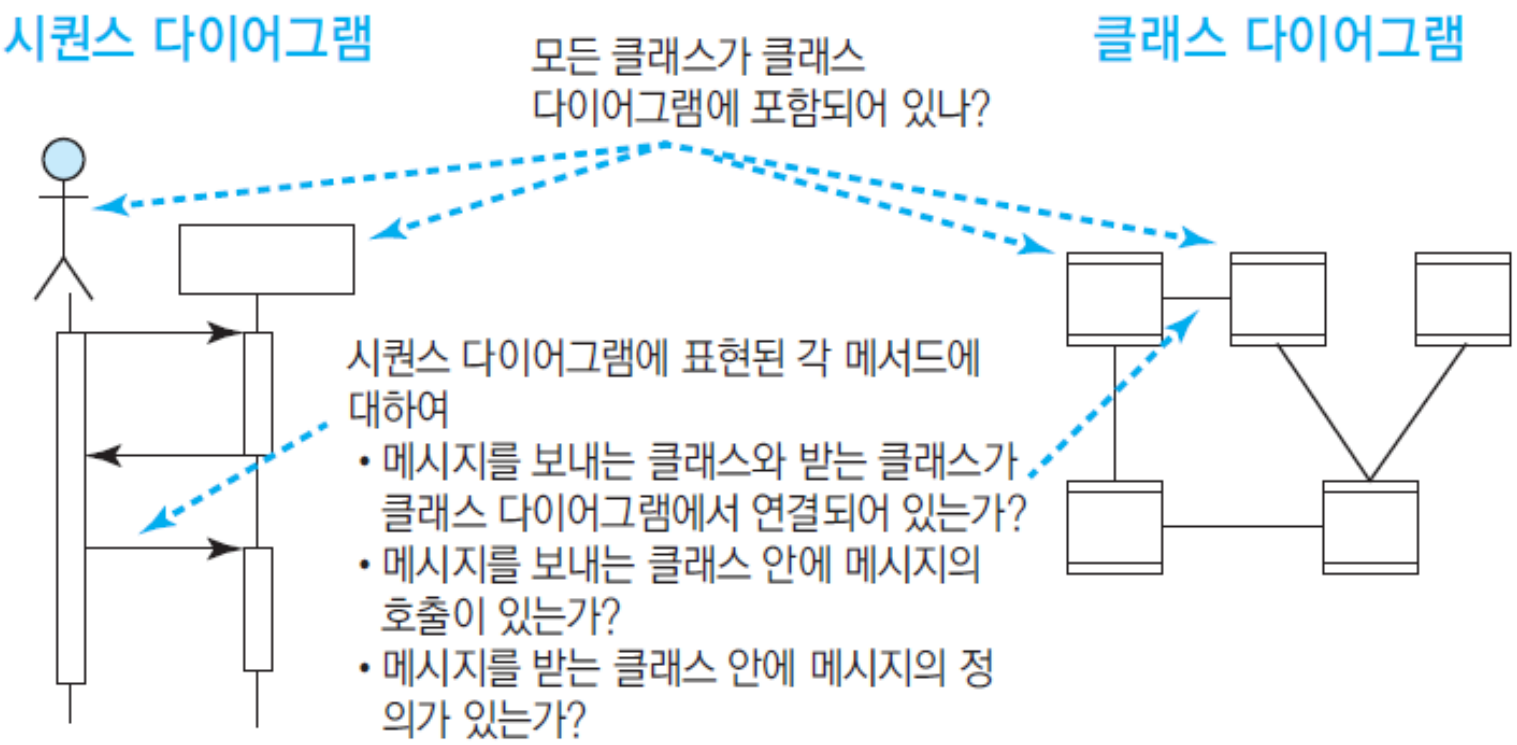
# 일관성 체크

- 유스케이스 다이어그램과 시퀀스 다이어그램
  - 유스케이스에 대한 명세가 기술되어 있고 매칭되는 시퀀스 다이어그램이 있는지



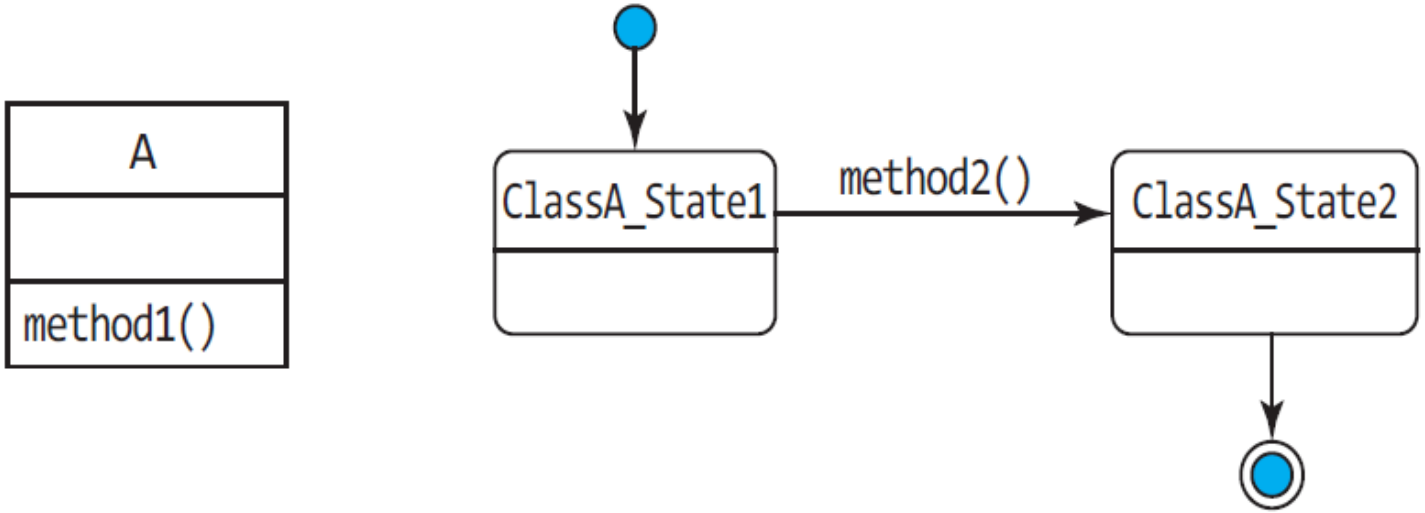
# 일관성 체크

- 시퀀스 다이어그램 안에 포함된 클래스와 메시지가 클래스 다이어그램에 빠지지 않고 표현되었는지 체크



# 일관성 체크

- 상태 다이어그램과 클래스 다이어그램을 크로스 체크



# 일관성 체크

표 5.5 모델 일관성 확인을 위한 체크리스트

| 다이어그램 | 체크 항목 |
|---|---|
| 유스케이스 다이어그램 | • 각 유스케이스는 액터가 있는가?<br>• 각 액터는 적어도 하나의 유스케이스가 있는가?<br>• 유스케이스는 명세화 되었나?<br>• 시퀀스 다이어그램과 매치되는가? |
| 클래스 다이어그램 | • 다른 다이어그램에 있는 클래스가 모두 클래스 다이어그램에 있는가?<br>• 모든 클래스가 속성에 대한 get/set 메소드가 있는가? |
| 시퀀스 다이어그램 | • 객체가 모두 클래스 다이어그램에 있는가?<br>• 각 메시지가 호출되나?<br>• 보내는 클래스와 받는 클래스가 연관되어 있나?<br>• 메시지 보내는 클래스 안에 메서드 호출이 있는가?<br>• 받는 클래스 안에 메시지가 정의되어 있나? |
| 상태 다이어그램 | • 상태 다이어그램의 클래스가 클래스 다이어그램에 있나?<br>• 각 트랜지션을 구동하는 이벤트가 있는가?<br>• 각 이벤트를 초기화 하는 객체가 확실히 있나?<br>• 상태는 속성 값의 서로 다른 조합인가?<br>• 어떤 조합인지 분명한가?<br>• 모든 속성이 클래스 다이어그램에 있나?<br>• 트랜지션을 위한 메서드 호출이 있나?<br>• 메서드 호출이 새로운 상태를 위하여 속성값이 바뀌나?<br>• 메서드 호출에서 트랜지션에 대한 조건을 체크하는가?<br>• 트랜지션에서 액션을 수행하는가? |

# 감사가 행복을 불러온다

- 감사는 사람들의 삶의 질에 영향을 미치는 매우 중요한 요소다.
- 감사하는 마음을 지닌 사람은 그렇지 않은 사람에 비해 더 행복하고,
- 더 열정적이며, 정서적으로 뚝뚝하면서도 덜 외롭고, 덜 우울하며 걱정도 적었다.
- 감사를 훈련하면 더 많이 행복해하고 더 낙관적으로 생각한다.
- 로버트 에몬스, 심리학자

- 행복하면 저절로 감사한 마음이 들지만,
- 그렇지 않을 때도 훈련을 통해 감사할 수 있습니다. 감사하는 마음이 역으로 행복을 만들어내기도 합니다.
- 감사하는 마음이 노화를 늦추기도 한다고 합니다.