

Software Engineering

5. requirements modeling
(Chapter 5)

Ji Eun Kim
Fall 2024

Topics

- 모델링 기초
- UML
- 정적 모델링
- 동적 모델링
- 제어 모델링
- 모델 검증
- Practice w/ UMLet

Preparation for hands-on: UML Tool Installation

- **Install UMLEt:**

- <https://www.umlet.com/>
- **free license**
- **stand-alone (.jar) or**
- **Eclipse plug-in**

The image shows two screenshots related to UMLEt. On the left is a screenshot of the UMLEt website (umlet.com) featuring a logo, navigation links, and a download section. On the right is a screenshot of the Eclipse IDE interface showing a UML statechart diagram titled "Main". The diagram includes states like "Lifeline A", "Lifeline B", "Lifeline C", and "Lifeline D", and various transitions and events. The Eclipse toolbar and menu bar are visible at the top.

Requirements Modeling

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

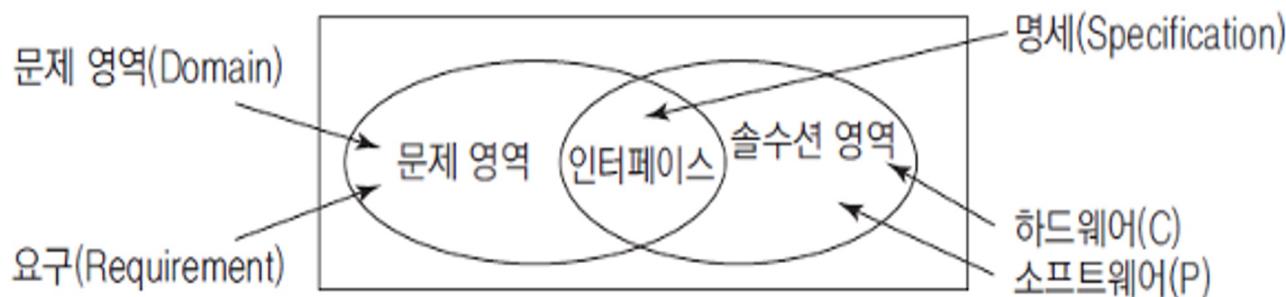


그림 5.1 문제 영역과 솔루션 영역

Modeling

- 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

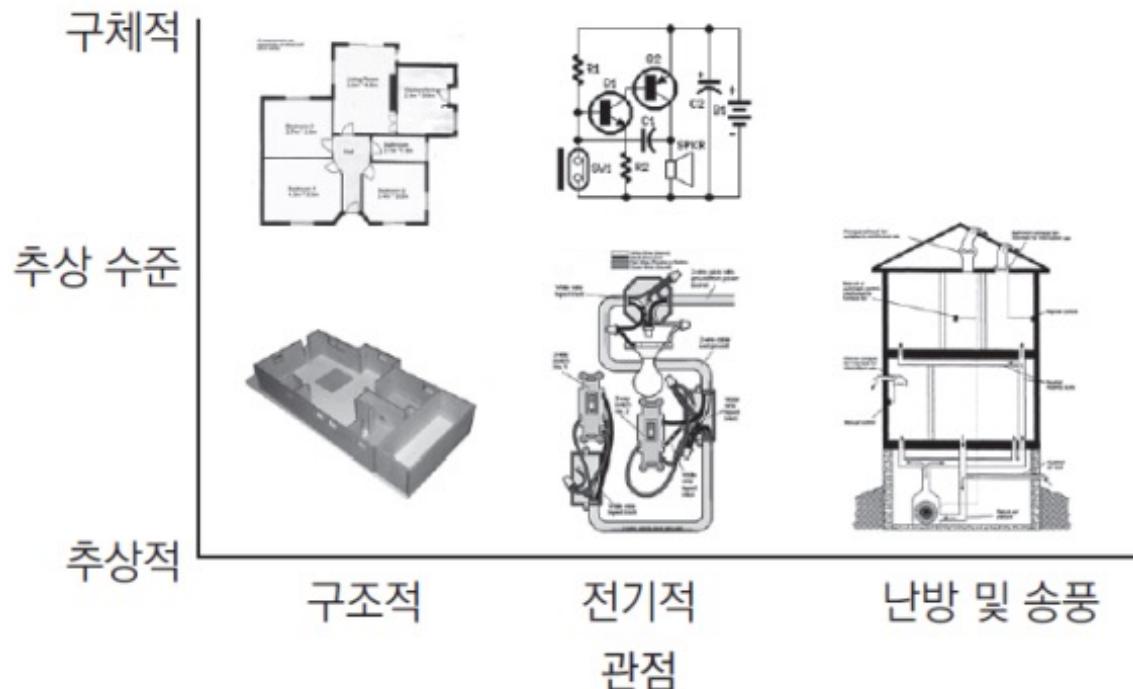
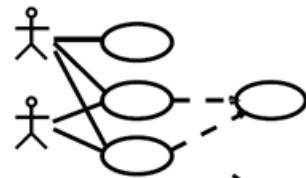
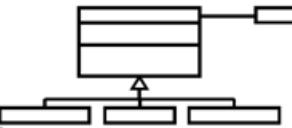


그림 5.5 관점과 추상 수준



사용 사례
다이어그램

기능적 관점



클래스
다이어그램

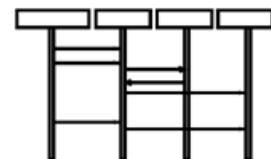
구조적 관점



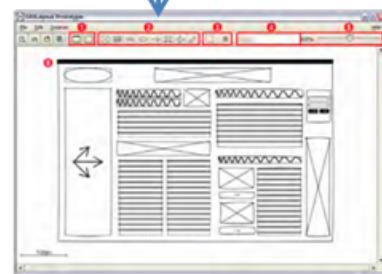
데이터베이스
관점

동적 관점

사용자인터
페이스 관점



순차
다이어그램



UI 레이아웃



ER 다이어그램

소프트웨어와 모델링

- 그래픽 기호와 주석으로 구성된 시각적 다이어그램

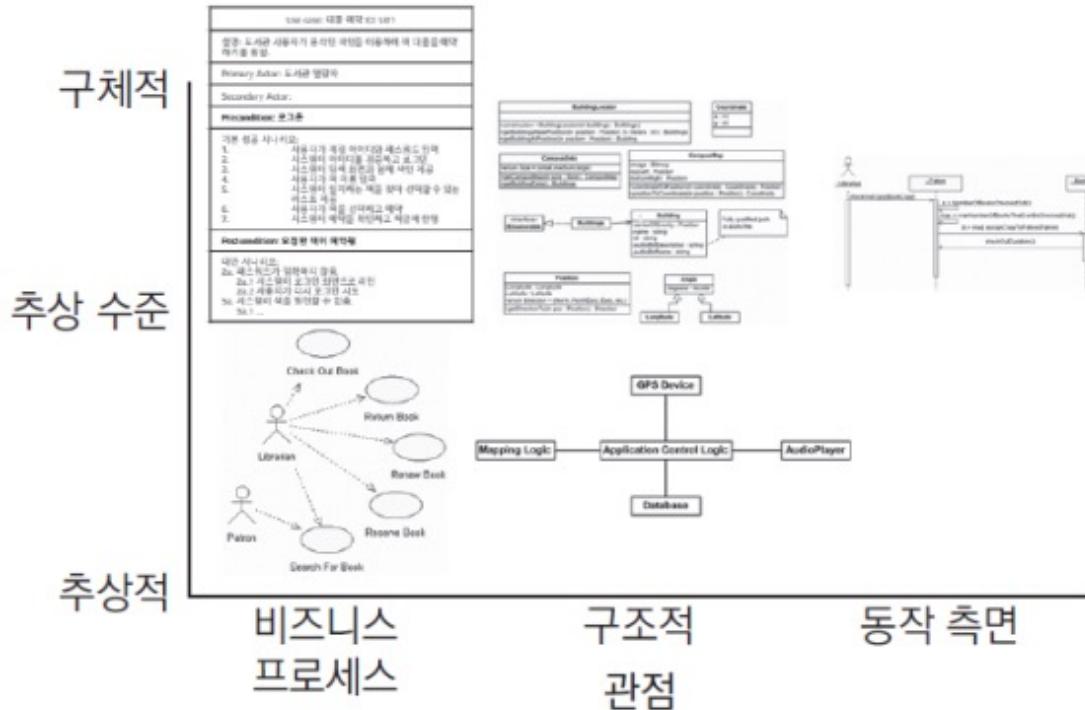


그림 5.6 소프트웨어 모델링의 관점과 추상 수준

UML(Unified Modeling Language)

- 객체지향 소프트웨어를 모델링 하는 표준 그래픽 언어
- 시스템의 여러 측면을 그림으로 모델링
- UML은 소프트웨어 모델링의 공통 언어
- UML 초안은 세 가지 주요 객체 지향 방법론의 통합으로 만들어진 표현
 - Grady Booch의 방법론
 - James Rumbaugh의 OMT(Object Modeling Technique)
 - Ivar Jacobson의 OOSE(Object-Oriented Software Engineering)

UML 다이어그램

- 시스템의 모델링 구성
 - 기능적 관점
 - 구조적 관점
 - 동적 관점
 - UML은 use case가 주도하고 architecture 중심적이며 반복, 점진적인 개발 프로세스를 권장



UML 모델링 과정

1. Requirements (요구) 를 **Use cases** (사용사례) 로 정리하고 Use case diagram 을 작성
2. **Class** 후보를 찾아내고 개념적인 객체 모형을 작성
3. Use case를 기초하여 **Sequence diagram** (순서 다이어그램) 을 작성
4. Class의 속성(attributes), 오퍼레이션 및 Class 사이의 관계를 찾아 객체 (object) 모형을 완성
5. State (transition) diagram (상태 다이어그램)이나 activity diagram 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 예

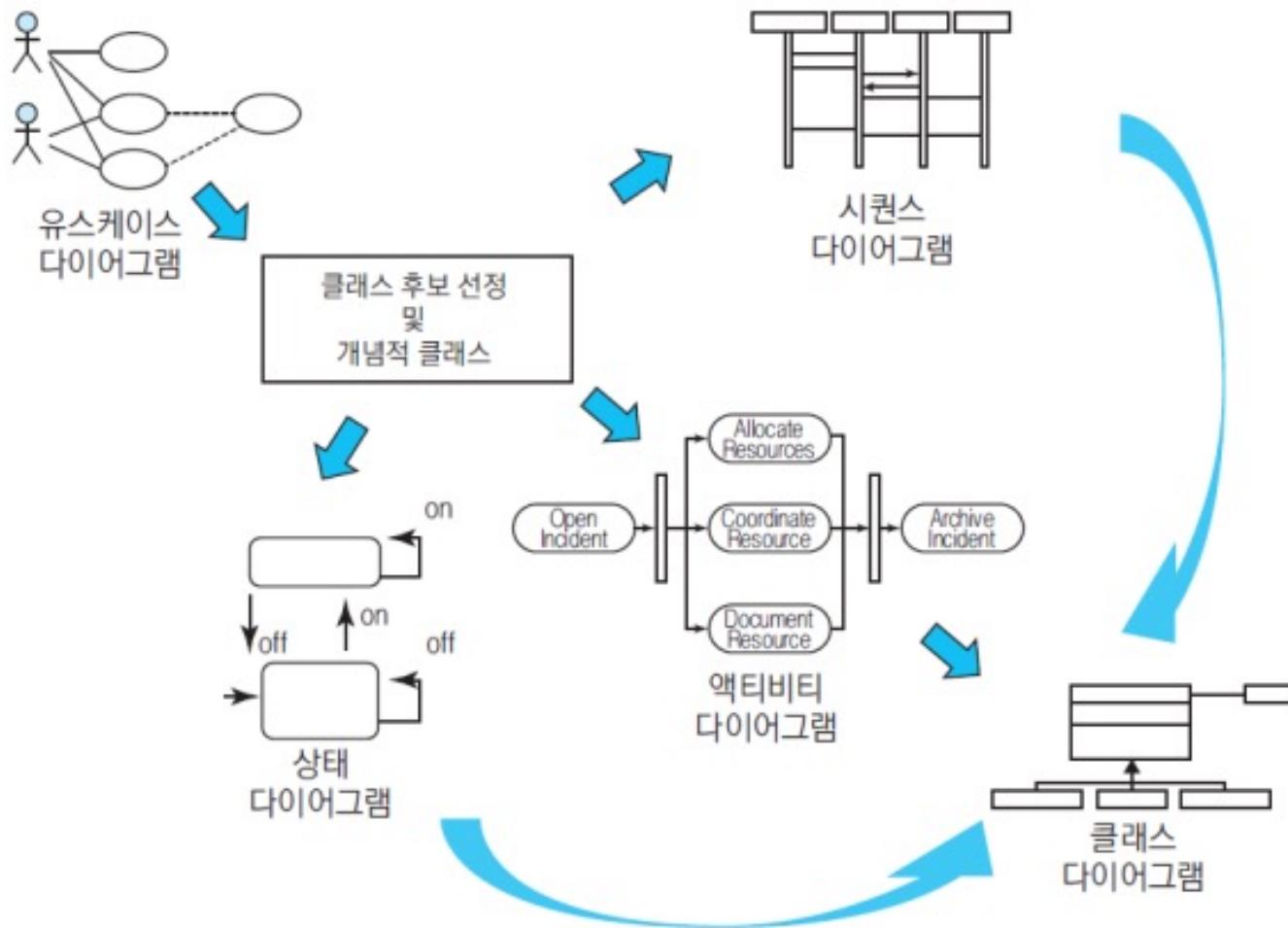


그림 5.11 UML 모델링 과정

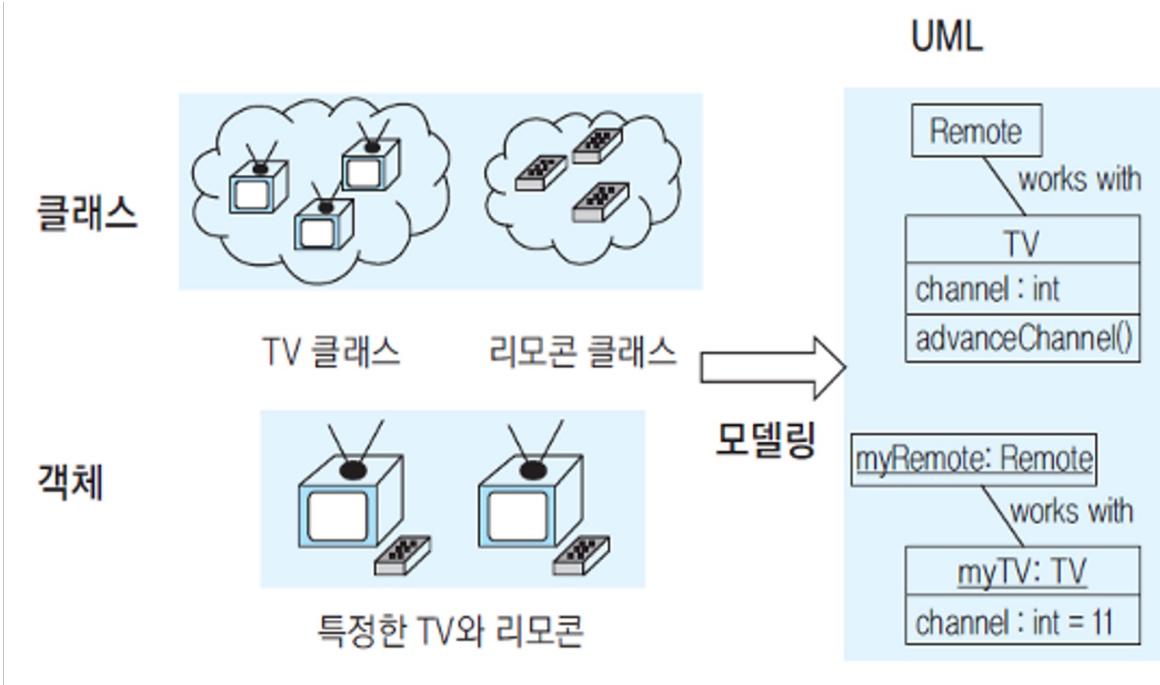
Static Modeling (정적 모델링)

정적 모델링 (Static Modeling)

- 정적 모델
 - 객체들의 공통 구조와 동작들을 추상화 시킨 것
- 객체지향 기본 개념의 이해가 필요
 - 객체와 속성(attributes), 연관 (association), 집합 (aggregation and composition), 상속(generalization/inheritance), 다형성 (polymorphism)
- Class Diagram이 대표적
 - 클래스 및 클래스 사이의 관계를 표현
 - 도메인 개념과 속성

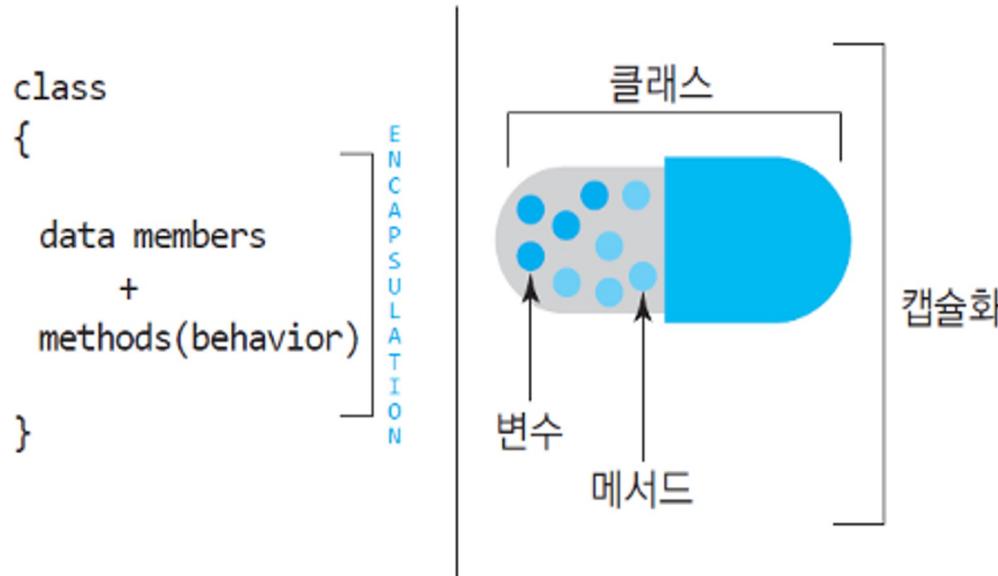
객체와 클래스

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



캡슐화 (Encapsulation)

- 객체의 속성 (attributes) 부분과 오퍼레이션 부분을 하나로 모아서 단위화
- 정보은닉 (Information hiding)



연관 (Association)

- 연관 (Association)
 - 서비스를 제공하는 객체와 서비스를 요청하는 객체가 상호작용하는 관계
 - Relationship (관계)에 의해 표시
- 가시성 (Visibility): 접근 가능성

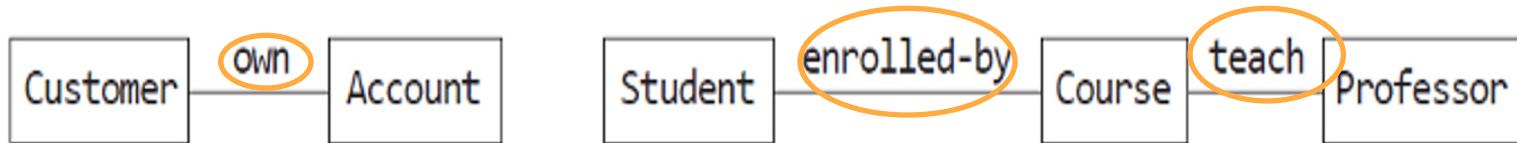


그림 5.14 연관의 사례

상속 (Inheritance)

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

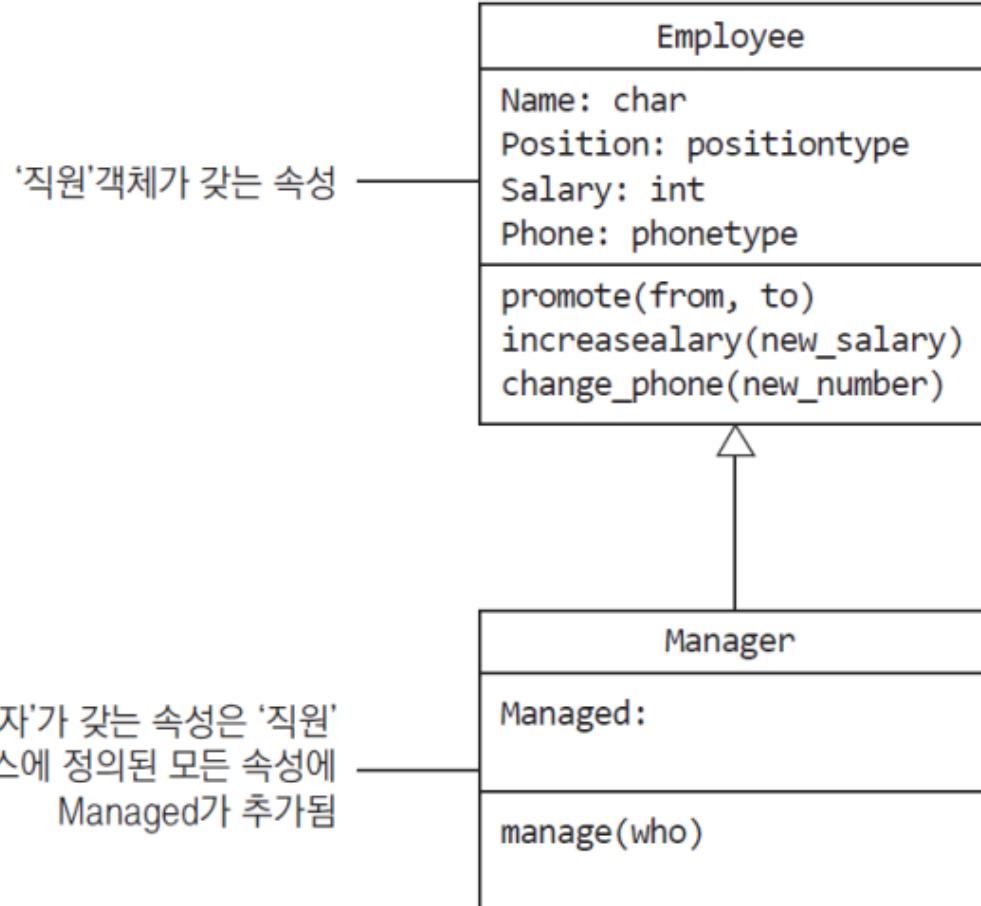
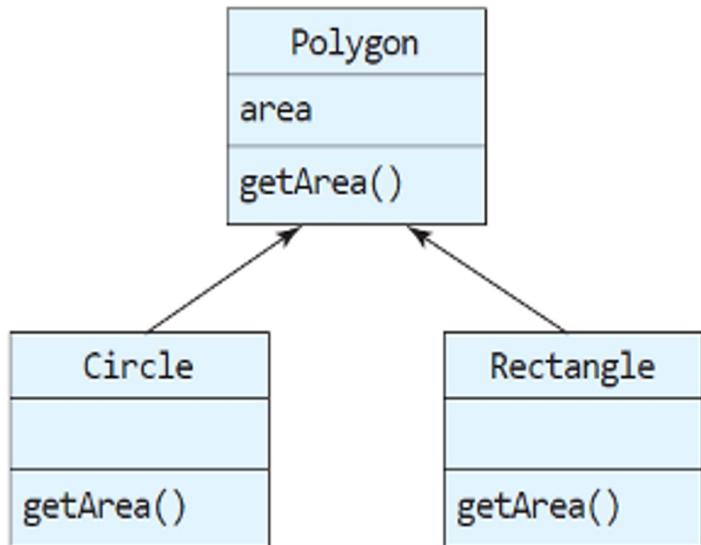


그림 5.16 상속의 예

다형성 (Polymorphism)

- 같은 이름의 메시지를 다른 객체 또는 서브 클래스에 호출할 수 있는 특징



- Polygon, Circle, Rectangle 에 해당되는 getArea() 함수를 구현.
- 생성된 객체의 class에 따라, 같은 method 이름을 사용하여 호출

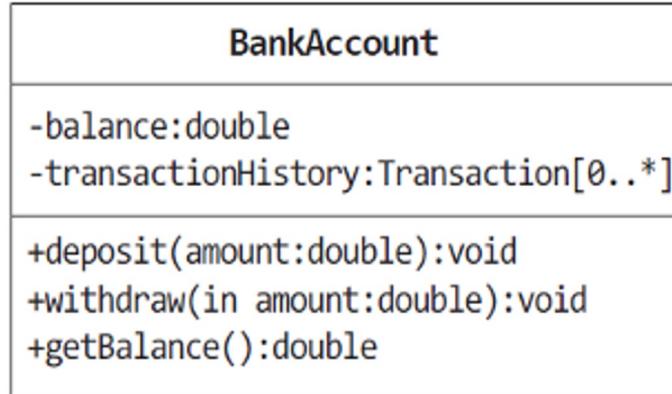
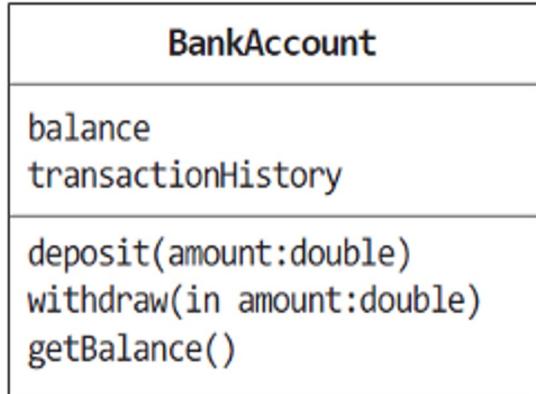
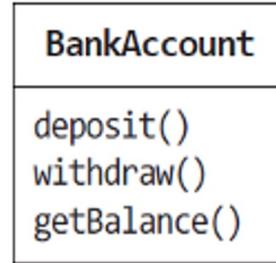
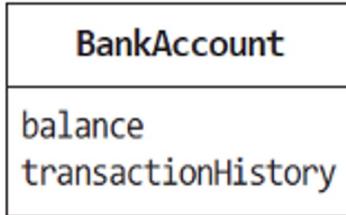
그림 5.17 다형성의 예

Class Representation

- 클래스 심볼
 - 세 개의 부분으로 나누고
 - 클래스의 이름, 클래스의 속성, 오퍼레이션을 적음
 - 추상클래스는 이탈릭체, 인터페이스 클래스는 <<interface>>추가
- 속성 (attributes) : 객체가 가지는 모든 필드를 포함
- 오퍼레이션 (operation) /메소드 (method)
- 아주 흔한 메소드(get/set)는 생략

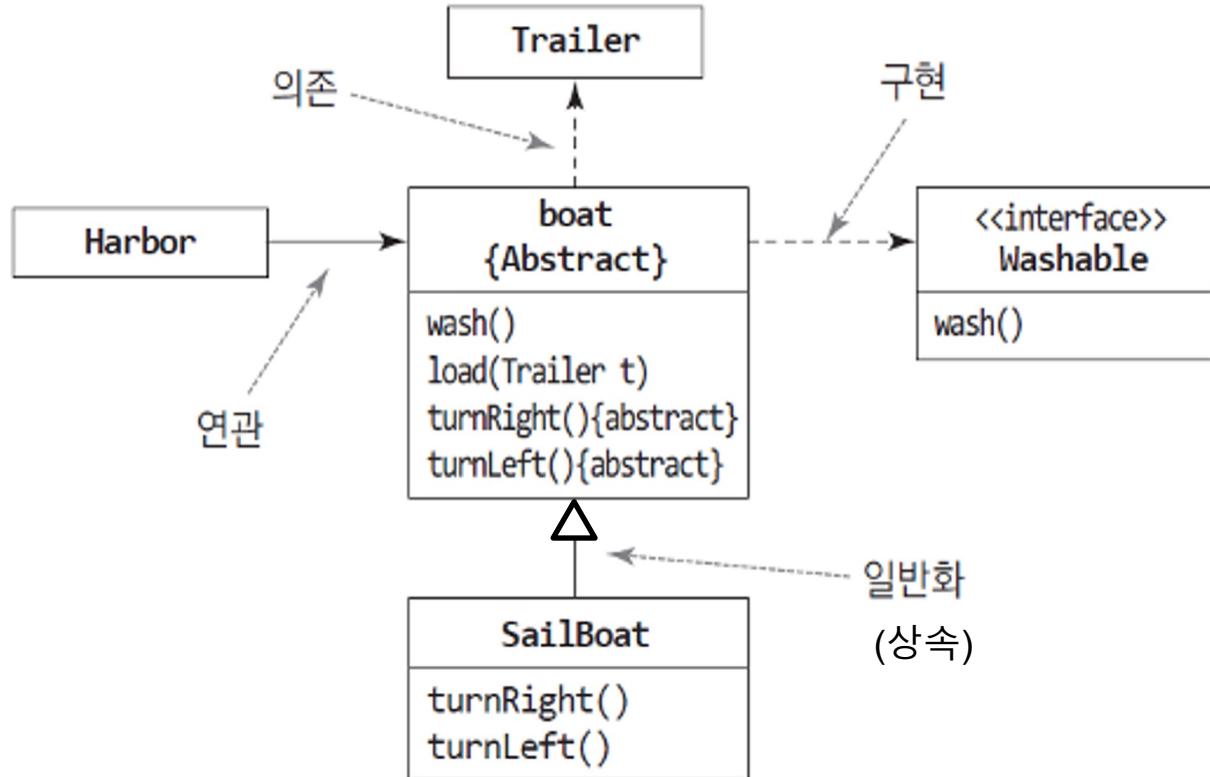
Class Representation Level

- 모델링이 진행되면서 상세화



관계의 표현

- (1) 연관 (association)
- (2) 상속 (inheritance)
- (3) 의존 (dependency)
- (4) 구현 (realization)



Multiplicity (다중성) Representation

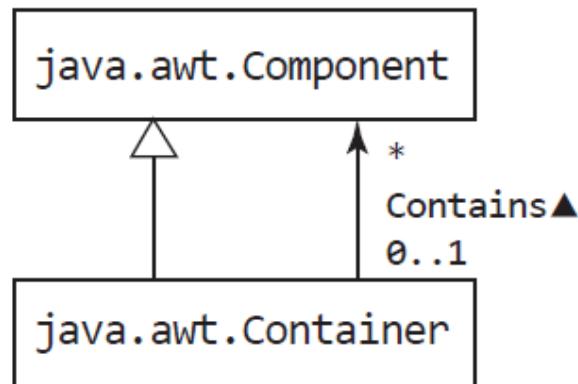


| | |
|-------------|---------------------|
| 1 | 한 개(표시하지 않은 경우 티플트) |
| * | 0 이상 |
| 1..* | 1 이상 |
| 4, 6, 8, 12 | 4, 6, 8, 12개 |
| 10..30 | 10에서 30개 |



그림 5.25 다중성

Multiplicity Representation (cont'd)



- Container에는 0개 이상의 Component가 포함
- 각 Component 최대 하나의 Container에 포함
- Container는 포함된 Component에 접근 가능
반대는 불가능

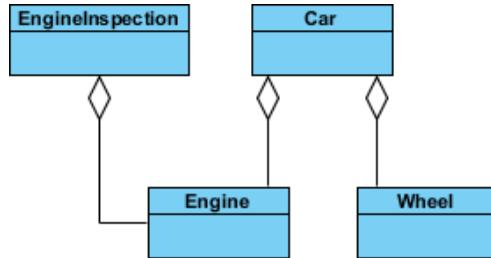
그림 5.26 다중 관계의 표현

클래스 다이어그램의 표현 요소

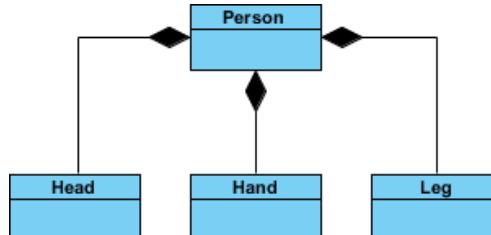
표 5.2 클래스 다이어그램의 표현 요소

| 요소 | 의미 | 표현방법 |
|---------------------------------|---|--|
| 클래스 속성 오퍼레이션 | 클래스는 타입이며 속성과 오퍼레이션은 클래스의 객체의 특징을 결정한다. | 축약형 확장형 클래스 이름 속성 리스트 오퍼레이션 리스트 |
| 상속 | 두 클래스의 일반화 및 상세화 관계 | 클래스 이름 → 슈퍼 클래스 |
| 집합 | 두 클래스 사이의 전체-부분 관계 | 부분 ⇔ 전체 부분 ◆ 전체 |
| 연관 방향 다중도 역할 연관 클래스 | 두 클래스 사이의 관계 연관을 나타내는 클래스 | 클래스 1 [m] [레이블] [n] 클래스 2 [역할1] [역할2] 연관 클래스 |

Aggregation
(Child class exist independently)



Composition
(Child cannot exist without Parent)



UML Tool Installation

- **Install UMLEt:**

- <https://www.umlet.com/>
- **free license**
- **stand-alone (.jar) or**
- **Eclipse plug-in**

The image shows two screenshots of the UMLEt tool. On the left is a screenshot of the umlet.com website, featuring a logo, navigation links like 'Main', 'Download', 'Screenshots', 'More Sample Diagrams', 'FAQ', 'Eclipse Plugin Info', 'Custom Elements', and 'User Comments'. It also includes a 'Bugs or requests' section with links to 'Create code ticket' and 'info@umlet.com'. On the right is a screenshot of the Eclipse IDE interface with the UMLEt plugin installed. The central workspace displays a UML statechart diagram titled 'A complex combined fragment'. The diagram shows four lifelines: Lfeline A, Lfeline B, Lfeline C, and Lfeline D. It includes regions labeled 'combined fragment', 'loop', and 'exitRegion'. Transitions are labeled with actions like 'execute()', 'duration', 'Message', and 'stop'. The Eclipse toolbar and various toolbars are visible at the top, and the Eclipse status bar is at the bottom.

Hands-on with UMLet

- Use case diagram

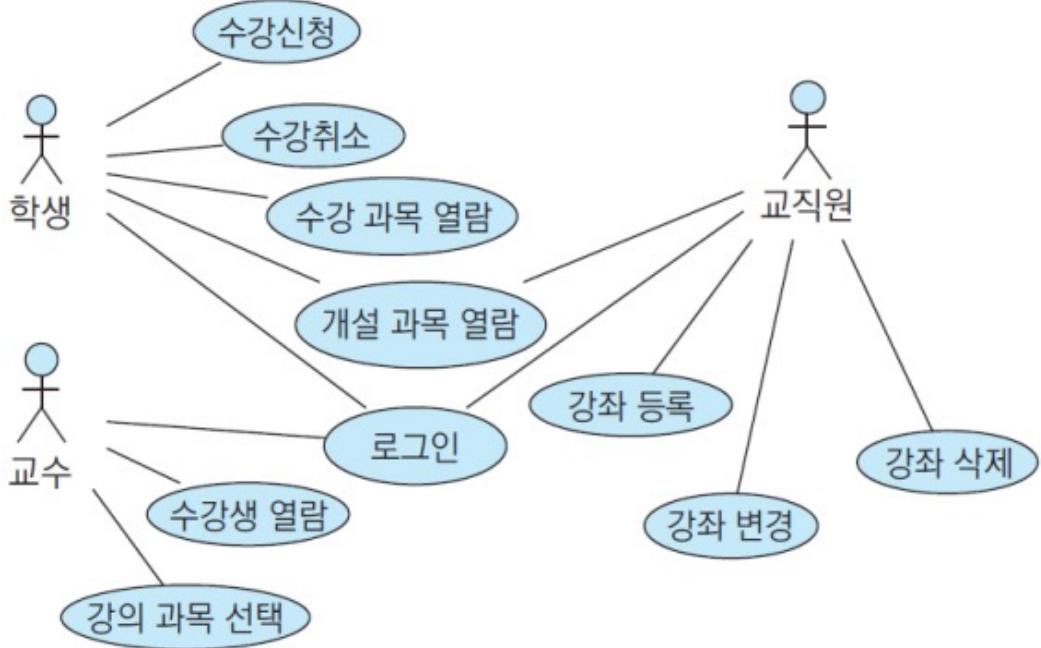


그림 4.11 유스케이스 다이어그램(수강 신청 시스템)

■ 액터



액터는 시스템 밖에 있는 요소로 대상 시스템과 상호 작용하는 사람이나 다른 시스템에 의한 역할이다.
액터는 대상 시스템에게 서비스를 제공하거나 제공 받는다.

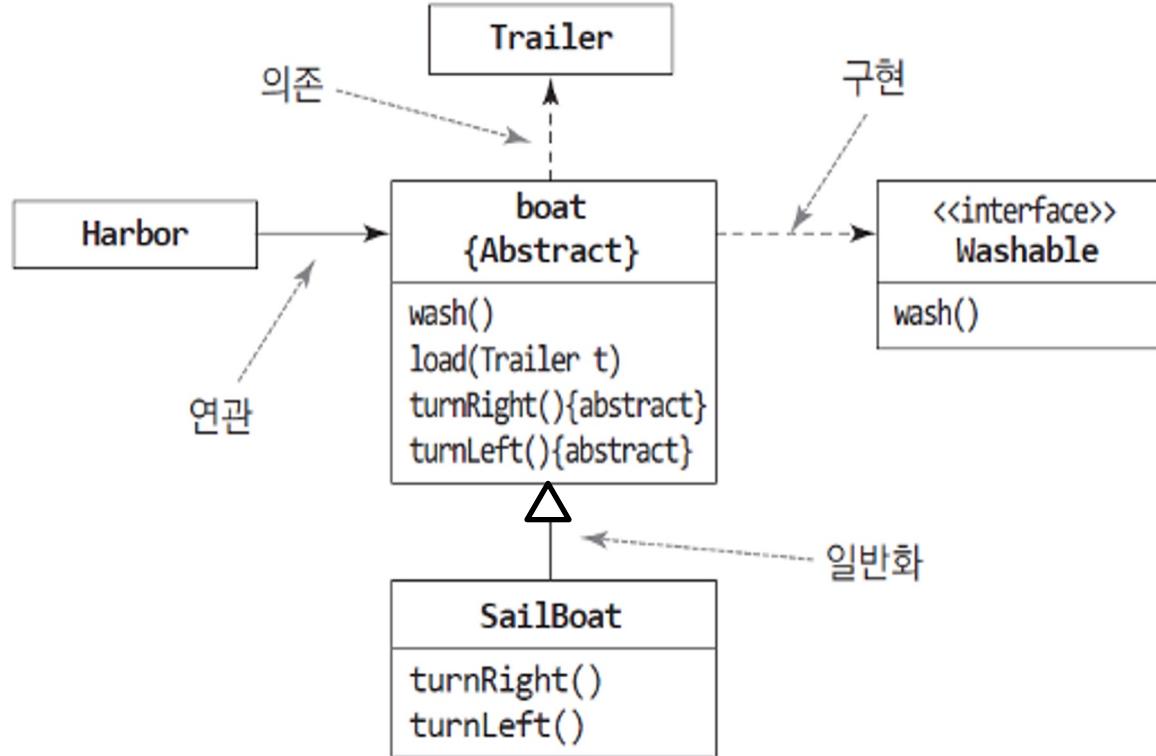
■ 유스케이스



유스케이스는 시스템이 실행하는 일련의 동작으로 특정 액터에게 관찰 가능한 실행 결과를 제공한다.
유스케이스는 시나리오의 집합을 정의한 것이다.

Hands-on with UMLet

- Class diagram



Case Study: 수강 신청 시스템

Requirements:

- 라인 수강신청을 위한 과목 (Course)이 개설되어 학생(Student)들이 강의열람표(ScheduleOfClasses)를 보고 개설된 강좌(CourseSelection)를 신청한다.
- 강좌는 1명의 교수가 담당하여 가르친다.
- 학생은 수강신청 하기 전에 학업이수계획(PlanOfStudy)을 참조하여 신청하며, 선수과목(prerequisite)을 수강하여야 강의를 신청할 수 있다.

Dynamic Modeling (동적 모델링)

동적 모델링 (Dynamic Modeling)

- 동적 측면
 - 소프트웨어가 실행될 때 변경 될 수 있는 뷰
 - 시간의 함수로만 이해
 - 예: 객체 간 상호작용 패턴
- 정적 다이어그램을 보완
- Sequence Diagram, Collaboration Diagram, Activity Diagram, State (transition) diagram
- Tool과 방법론에 따라 (e.g., Rational, 4-1 View) 다양한 diagram을 사용하여 modeling

Sequence Diagram

- 메세지 교환을 시간적 관계로 나타낸 것
- 유스케이스의 이벤트 흐름을 나타내는데 사용
- 유스케이스에 참여하는 객체를 찾아내고 유스케이스 명세에 표현된 동작을 시스템 내부에 존재하는 객체의 메세지 교환 형태로 표현

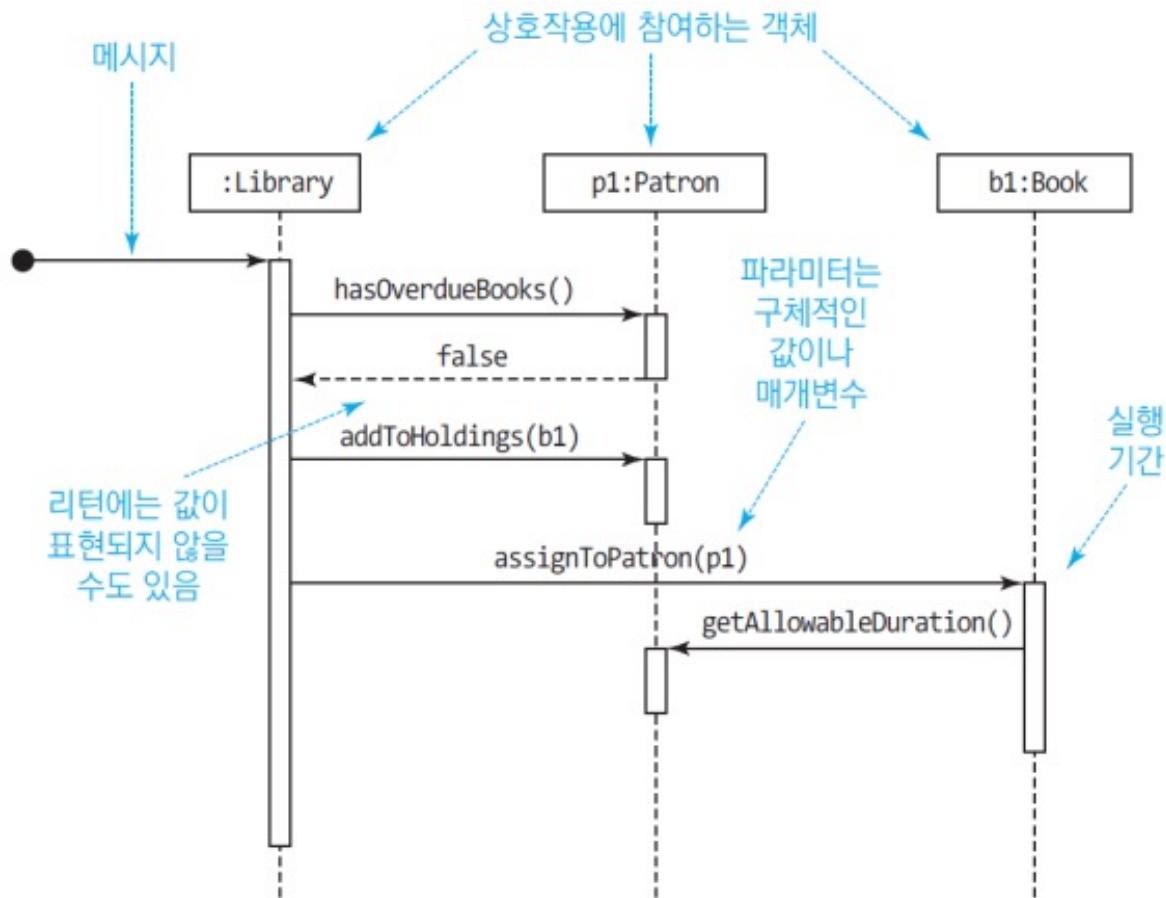


그림 5.31 시퀀스 다이어그램

Sequence Diagram Elements

표 5.3 시퀀스 다이어그램의 기본 요소

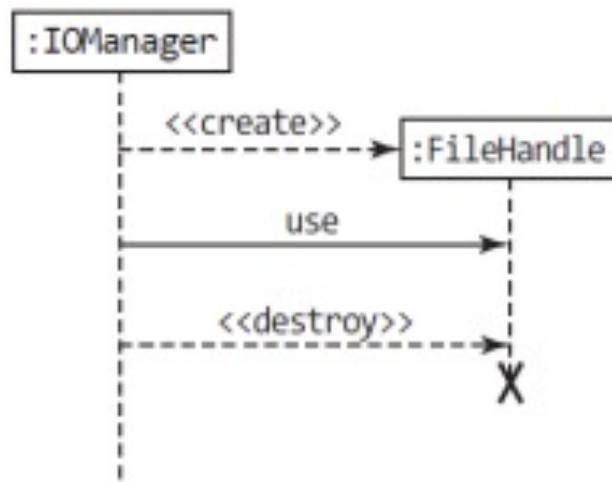
| 요소 | 표현방법 | 의미 | 연결 |
|-------|------|--|-----------------------------------|
| 객체 | | - 특정 클래스의 객체 - 객체의 모임 | |
| 객체집합 | | - 라이프라인 위에 위치하며 콜론 앞은 객체의 이름, 뒤는 클래스의 이름 | 라이프라인 사이에 활성막대와 연결됨 |
| 라이프라인 | | 객체가 시스템에 존재하나 아직 실행되지는 않음을 의미. | 객체를 활성막대와 연결시키며 두 개의 이웃 라이프라인을 연결 |
| 활성막대 | | 시스템에 존재하는 메서드가 막대의 길이만큼 실행됨을 의미. 점선은 라이프라인임. | 객체와 연결됨. 라이프라인과 연결됨 |
| 객체 소멸 | | 라이프라인 맨 위에 연결된 객체가 소멸됨을 의미. | |

객체의 표현 (object:Class)

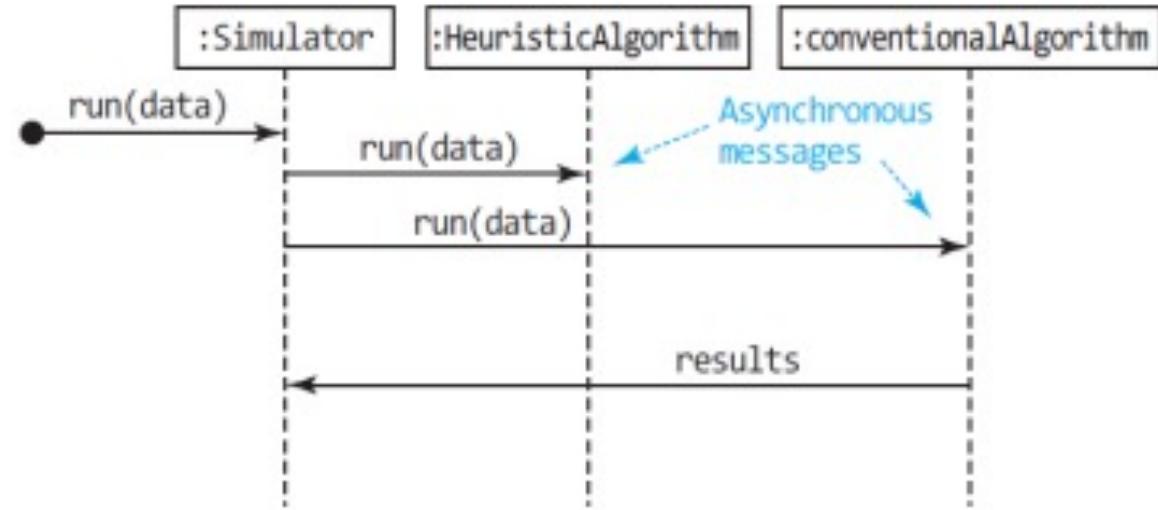
| | |
|-----------------------|-------------------------|
| :Person | Person 클래스의 이름 없는 인스턴스 |
| kim: | 이름 없는 클래스의 kim이라는 인스턴스 |
| kim:Person | Person 클래스의 kim이라는 인스턴스 |
| :Person | Person 클래스의 인스턴스의 모임 |
| <<jsp>> LoginPage: | 스테레오타입 객체 |

그림 5.30 객체의 표현

메세지 표현



(a) 객체 생성과 소멸



(b) 비동기 메시지

그림 5.32 메시지의 표현

Loop and Condition Representation

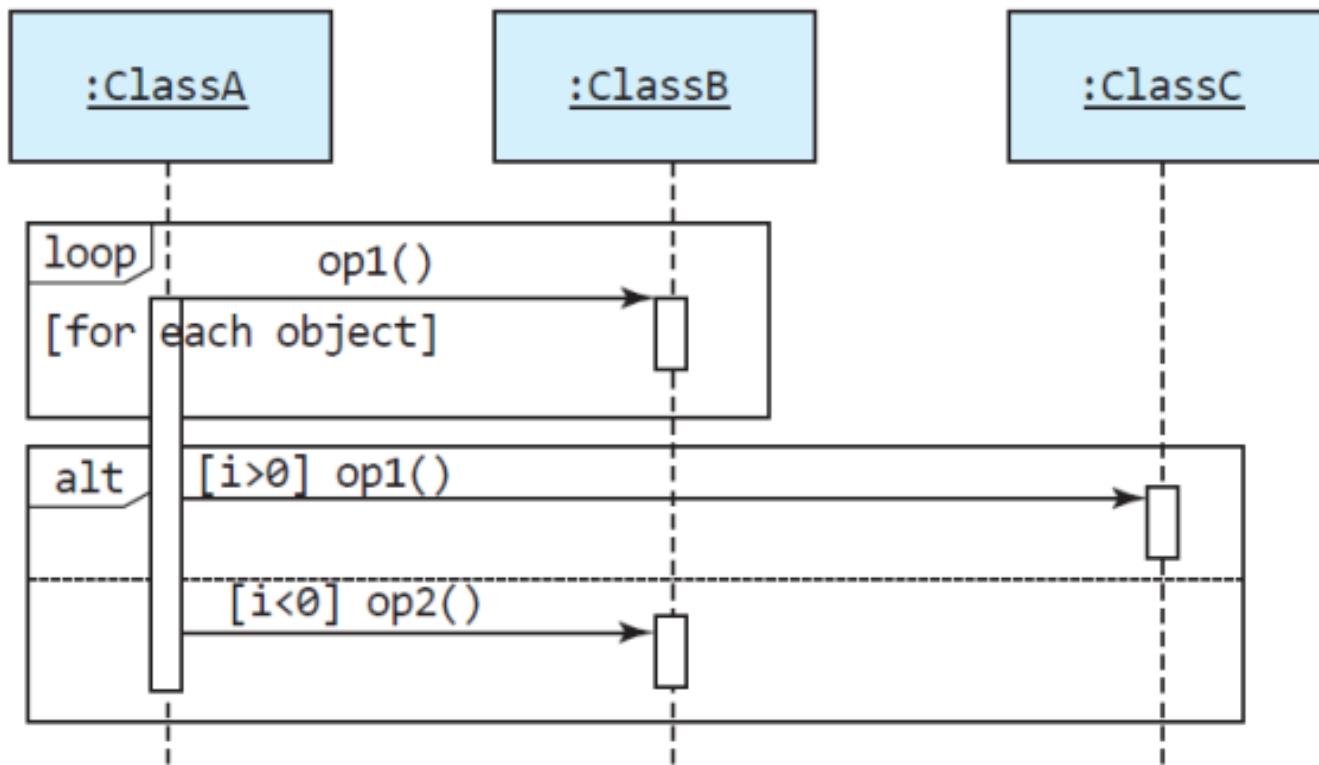


그림 5.33 시퀀스 다이어그램에서 반복과 조건의 예

시퀀스 다이어그램 작성 과정

- Step 1. 참여하는 객체를 파악
- Step 2. 파악한 객체를 X축에 나열하고 라이프라인을 그음
- Step 3. 사용사례에 기술된 이벤트 순서에 따라 객체의 메시지 호출을
화살표로 나타냄

클래스 다이어그램과의 관계

- 시퀀스 다이어그램은 유스케이스나 특정 오퍼레이션 수행을 위한 **메시지 교환에 초점을 두고 모델링한 것**.
- 시퀀스 다이어그램으로 모델링과정에 **유스케이스가 상세화** 되고 더 많은 객체와 서비스가 발견
- **파악한 메시지의 소속과 참여 객체들의 관계는 클래스 다이어그램에 반영할 것**

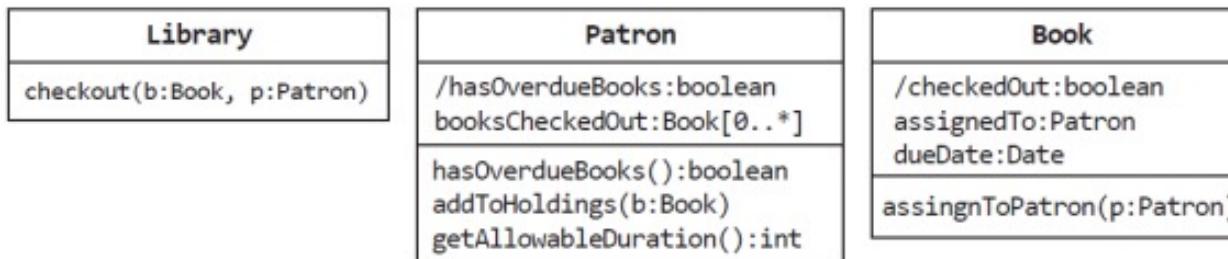


그림 5.35 그림 5.31과 관련된 클래스

Case Study: 수강 신청

- 수강신청 클래스 다이어그램에서 수강신청 유스케이스에 참여하는 객체를 찾는다.
- GUI를 담당하는 객체가 필요하며 수강 신청을 위하여 신청 과목을 선택받는다.
- 다음은 강좌(CourseSelection)에 requestToRegister()메세지를 보내어 학생 등록을 타진한다.
- 수강신청 여부를 판단하기 위하여 과목(Course)에 대한 선수과목(prerequisite)을 알아오는 메시지도 포함한다.
- 선수과목을 수강하였다면 다음에는 등록 정보를 기록한다.
 - 등록 정보를 기록할 객체가 바로 수강신청 유스케이스에서 중요한 역할을 담당하는 Registration객체이다.
- Registration객체를 생성한 후 이를 Student객체와 CourseSelection객체에 연결시키면 완료된다.

Case-Study: 수강 신청 시스템

Collaboration Diagram (협동 다이어그램)

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

Example

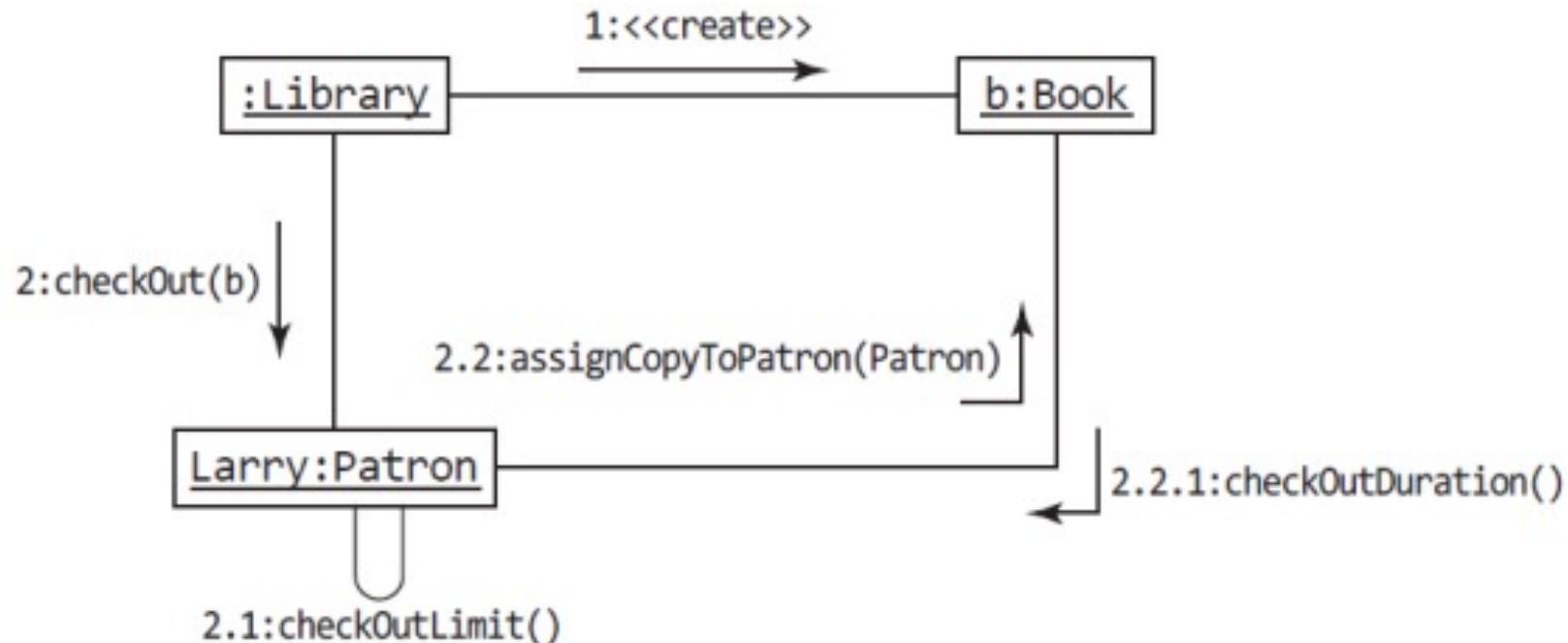


그림 5.36 협동 다이어그램

State Transition Diagram (상태 다이어그램)

- 동작을 수신 이벤트와 이에 대한 응답을 기반으로 상태 사이의 전환으로 모델링

예: 도서관 시스템에서 책의 상태

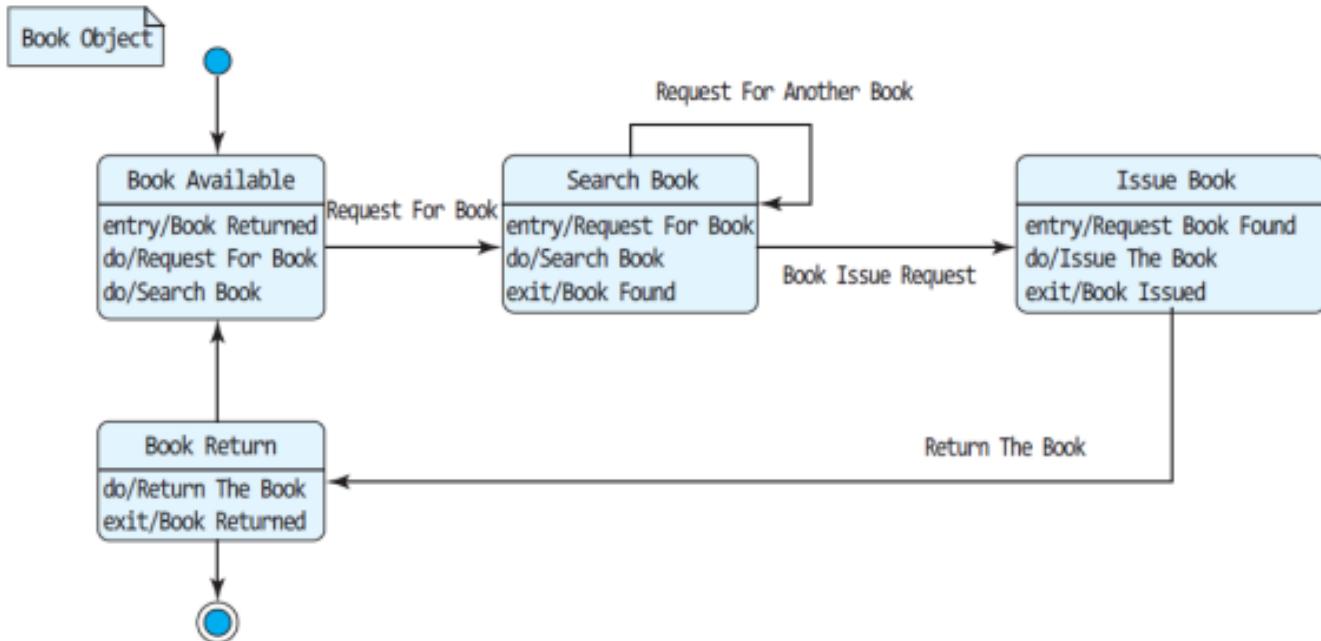


그림 5.37 상태 다이어그램

State (Transition) Diagram Elements

표 5.4 상태 다이어그램의 기본 요소

| 요소 | 표현방법 | 연결 |
|-------|-----------------|---|
| 상태 | | 단순 상태: 다른 상태를 품고 있지 않은 단순한 상태. 서브시스템이나 객체의 조건이나 상황이다. |
| 시작 상태 | | 시스템이 시작되었을 때 머무르는 가상의 상태 |
| 종료 상태 | | 시스템이 종료되었음을 나타내는 상태 |
| 트랜지션 | | 하나의 상태에서 다른 상태로 이벤트에 의하여 변화됨 |
| 레이블 | $e[exp]/a1; a2$ | 이벤트(e)가 발생하고 가드조건(exp)이 참이면 트랜지션이 일어나고 액션 a1, a2가 실행됨 |

Activity Diagram

- Activity 사이의 제어 흐름을 보여주는 일종의 흐름도
 - Activity: 계산 또는 프로세스
 - Transition (전환): 액티비티에서 다른 액티비티로 제어가 넘어감
 - Branch (분기): 진위 조건 (Guard expressions, condition)
 - Merge

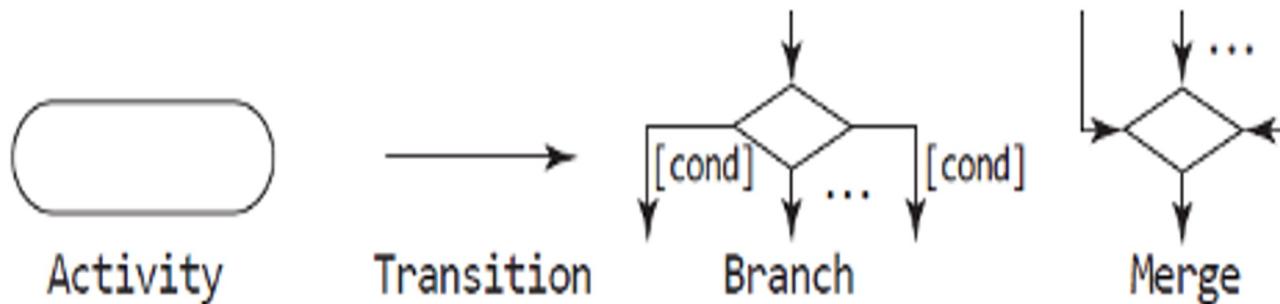
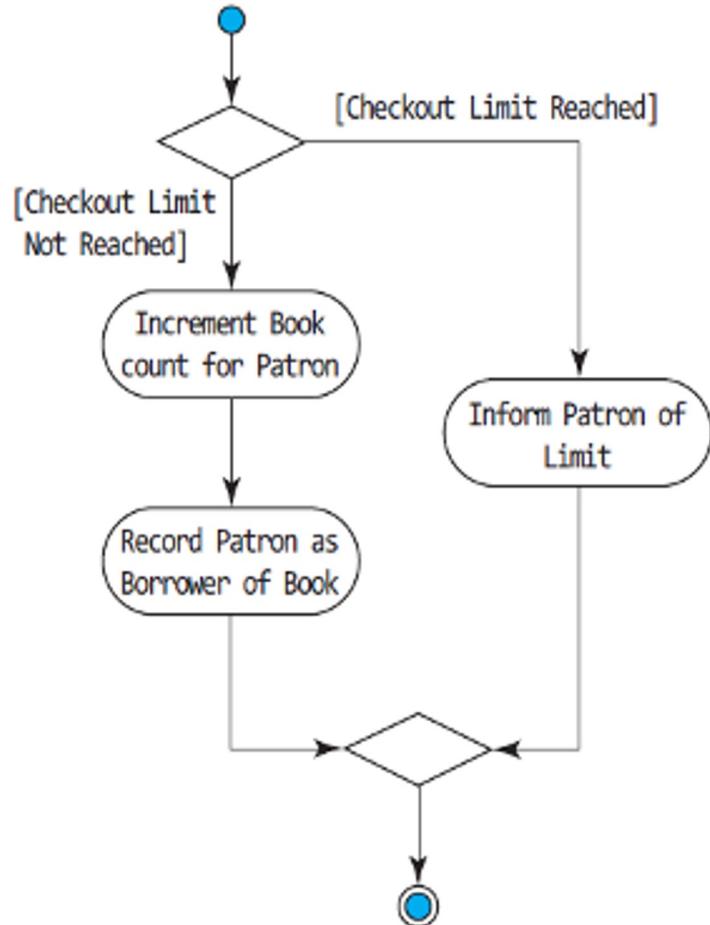


그림 5.38 액티비티 다이어그램의 기본 요소

Example



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

Swimlane

- Swimlane(스윔레인): activity에 대한 responsibility를 할당

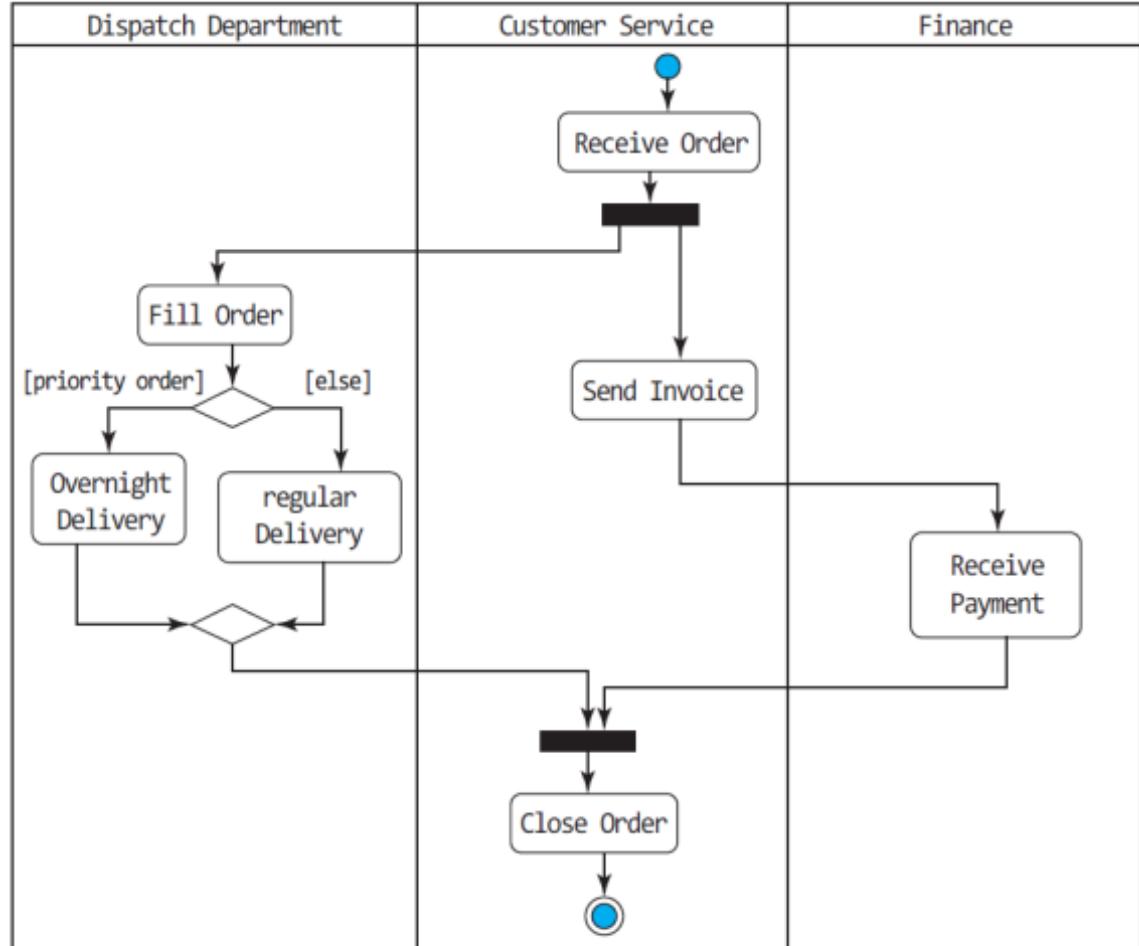


그림 5.40 병렬과 스윔레인이 표시된 액티비티 다이어그램

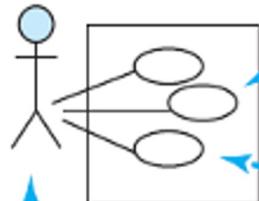
모델 검증

- 모델 검증 방법
 - 리뷰: 워크스루, 인스펙션
 - 테스팅
 - Formal Method (e.g. Z Specification)
 - 프로토타이핑
 - 요구 추적

일관성 체크

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

유스케이스 다이어그램

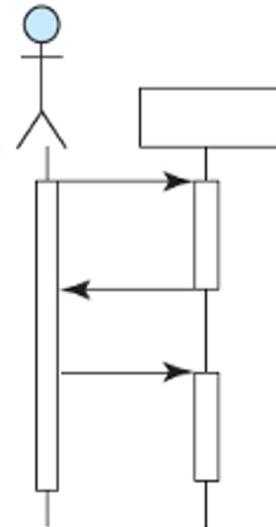


각 액터에 적어도
하나 이상의
유스케이스가 있는가?

각 유스케이스를 구동하는
액터가 있는가?

각 유스케이스에 대
한 명세가 기술되었
나?
해당되는 시퀀스 다
이어그램이 있는가?

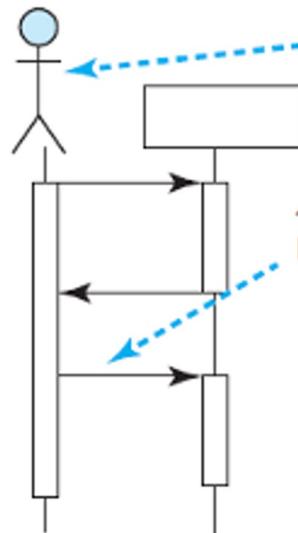
시퀀스 다이어그램



일관성 체크 (cont'd)

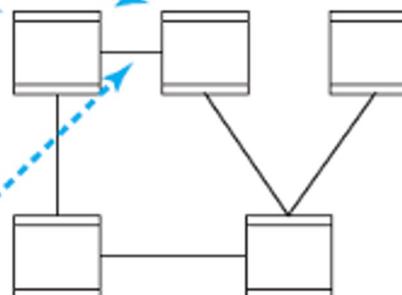
- 시퀀스 다이어그램 안에 포함된 클래스와 메시지가 클래스 다이어그램에 빠지지 않고 표현되었는지 체크

시퀀스 다이어그램



모든 클래스가 클래스
다이어그램에 포함되어 있나?

클래스 다이어그램



시퀀스 다이어그램에 표현된 각 메서드에
대하여

- 메시지를 보내는 클래스와 받는 클래스가
클래스 다이어그램에서 연결되어 있는가?
- 메시지를 보내는 클래스 안에 메시지의
호출이 있는가?
- 메시지를 받는 클래스 안에 메시지의 정
의가 있는가?

일관성 체크 (cont'd)

- 상태 다이어그램과 클래스 다이어그램을 크로스체크

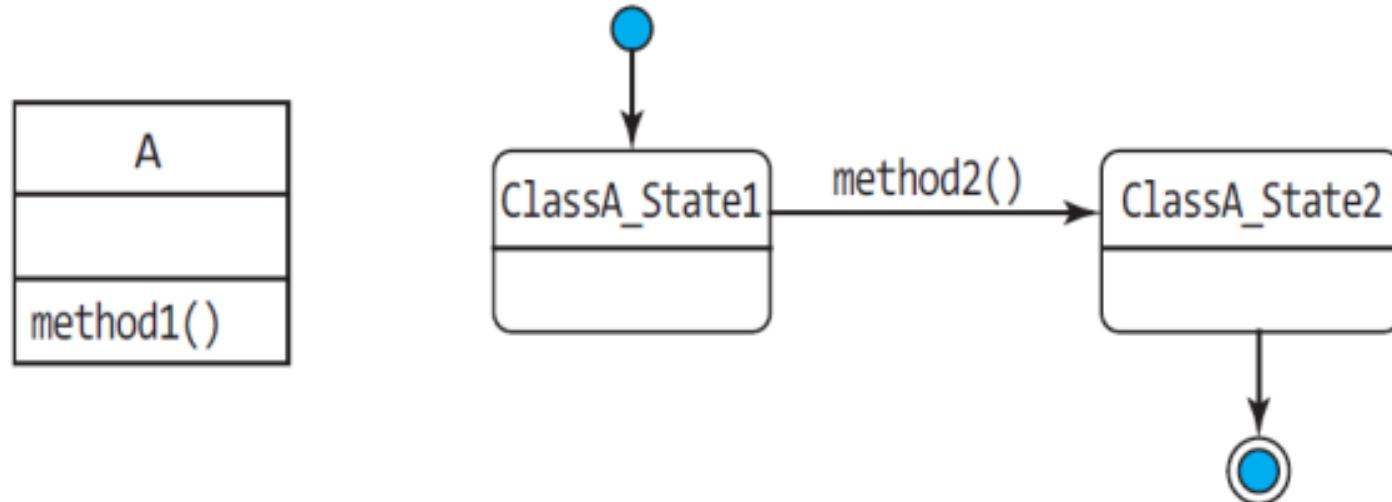


그림 5.43 클래스 A에 `method2()` 선언이 누락됨

표 5.5 모델 일관성 확인을 위한 체크리스트

| 다이어그램 | 체크 항목 |
|-------------|---|
| 유스케이스 다이어그램 | <ul style="list-style-type: none"> 각 유스케이스는 액터가 있는가? 각 액터는 적어도 하나의 유스케이스가 있는가? 유스케이스는 명세화 되었나? 시퀀스 다이어그램과 매치되는가? |
| 클래스 다이어그램 | <ul style="list-style-type: none"> 다른 다이어그램에 있는 클래스가 모두 클래스 다이어그램에 있는가? |
| 시퀀스 다이어그램 | <ul style="list-style-type: none"> 객체가 모두 클래스 다이어그램에 있는가? 각 메시지가 호출되나? 보내는 클래스와 받는 클래스가 연관되어 있나? 메시지 보내는 클래스 안에 메서드 호출이 있는가? 받는 클래스 안에 메시지가 정의되어 있나? |
| 상태 다이어그램 | <ul style="list-style-type: none"> 상태 다이어그램의 클래스가 클래스 다이어그램에 있나? 각 트랜지션을 구동하는 이벤트가 있는가? 각 이벤트를 초기화 하는 객체가 확실히 있나? 상태는 속성 값의 서로 다른 조합인가? 어떤 조합인지 분명한가? 모든 속성이 클래스 다이어그램에 있나? 트랜지션을 위한 메서드 호출이 있나? 메서드 호출이 새로운 상태를 위하여 속성값이 바뀌나? 메서드 호출에서 트랜지션에 대한 조건을 체크하는가? 트랜지션에서 액션을 수행하는가? |

References

- [1] 소프트웨어 공학의 모든 것, 최은만, 생능출판
- [2] UMLet