



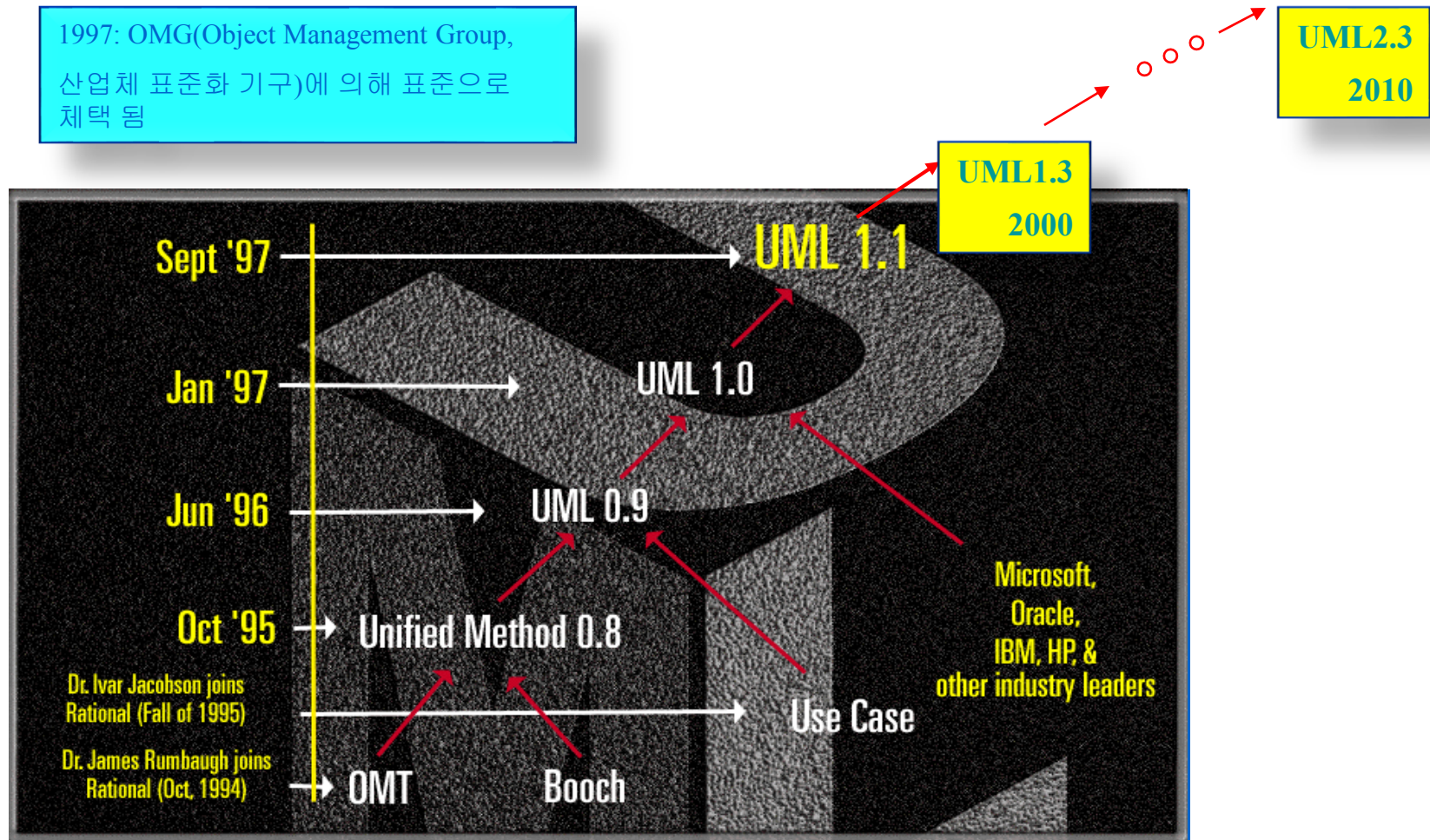
Software & System Modeling UML

•Chap 5 → UML Manual

The Unified Modeling Language (UML)

- **Several different notations for describing object-oriented designs were proposed in the 1980s and 1990s.**
- **The Unified Modeling Language is an integration of these notations.**
- **It describes notations for a number of different models that may be produced during OO analysis and design.**
- **It is now a de facto standard for OO modelling.**

1997: OMG(Object Management Group,
산업체 표준화 기구)에 의해 표준으로
채택 됨



<http://www.omg.org/spec/UML/>

References

- **UML 표기법**

- UML User Guide <http://www.omg.org/spec/UML/2.3>

- **Software Development Process based on UML**

- Rational Unified Process (RUP)

- IBM Rational Unified Process®, or RUP®, is a configurable software development process platform that delivers proven best practices and a configurable architecture.

- **Supporting Tools**

- ROSE2000

- <http://www.ibm.com/support/kr/ko/> → 트라이얼 및 베타버전 다운로드

UML(Unified Modeling Language)

- OMT + Booch + OOSE +

- Views of UML

Use-case view

functionality of system as perceived by external actor 외부 액터의 관점에서 시스템의 기능적인 측면을 바라보는 뷰

- use-case diagram

Logical view

how the functionality is provided; static structure and dynamic behavior 시스템의 기능들이 내부적으로 어떻게 설계될 수 있는지를 보여주는 뷰

- package diagram, class diagram, state diagram, sequence diagram, collaboration diagram, activity diagram

Component view

implementation modules and their dependency 컴포넌트와 컴포넌트의 관계를 보여주는 뷰

- component diagram

Concurrency view

division of process & processors; also communication and synchronization

시스템 요소들 간의 동기/비동기, 상호작용에 대한 사항 등, 시스템 요소들에 대한 처리방식을 보여주는 뷰.

프로세스와 프로세서의 할당, 효과적인 리스스의 사용, 병렬 수행, 주변환경에서의 이벤트에 대한 처리, 동기화 상호통신을 나타냄. 개발자와 시스템 통합자를 위한 것.

- dynamic diagrams(state, sequence, collaboration diagrams)
- Implementation diagrams (component, deployment diagrams)

Deployment view

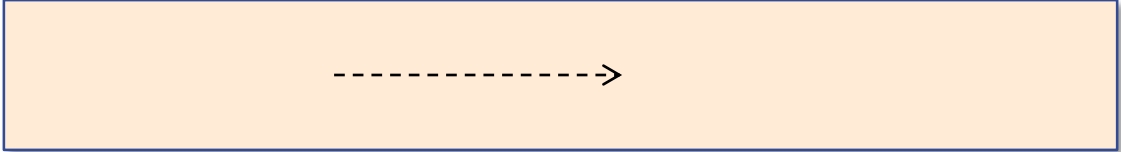
physical deployment; computers & devices 컴퓨터와 주변 장치로 구성된 시스템의 실제 배치를 보여주는 뷰

- deployment diagram

Relationships

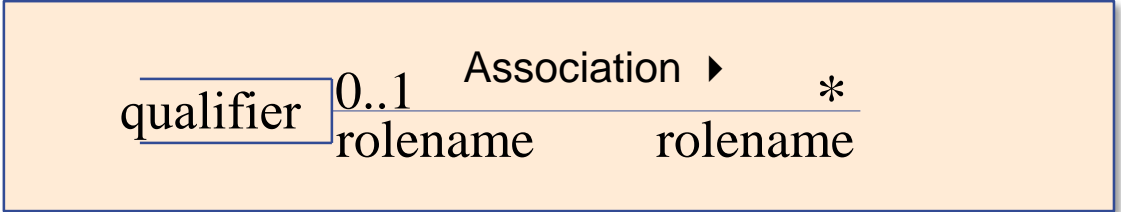
• Dependency

- 어떤 spec에서의 한 변화가 다른 것에 영향을 주는 관계(역은 성립할 필요 없음)



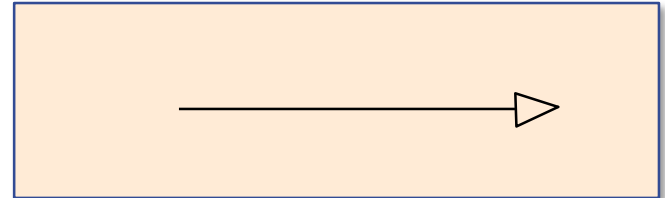
• Association

- structural relationship that specifies that objects of one thing are connected to objects of another
- binary association(common), n-ary도 가능
- name, role, multiplicity



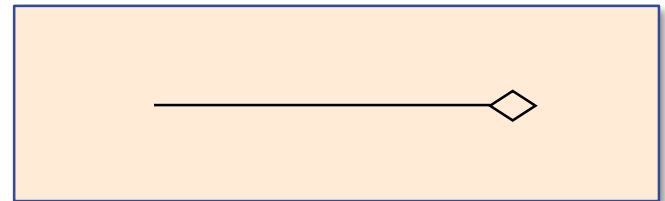
• Generalization

- superclass(parent)와 subclass(child)
- “is-a-kind-of” relationship
- parent가 사용되는 곳이면 어디든 child도 사용가능(역은 not)



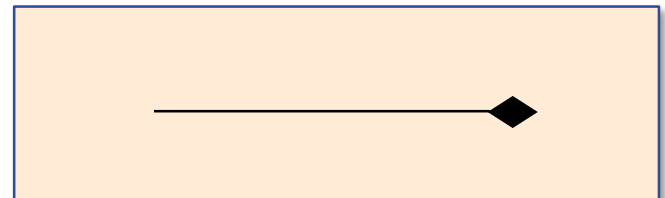
• Aggregation

- “has-a” relationship (“whole/part”)



• Composition

- “has-a” relationship (“whole/part”)
- 강한 집합



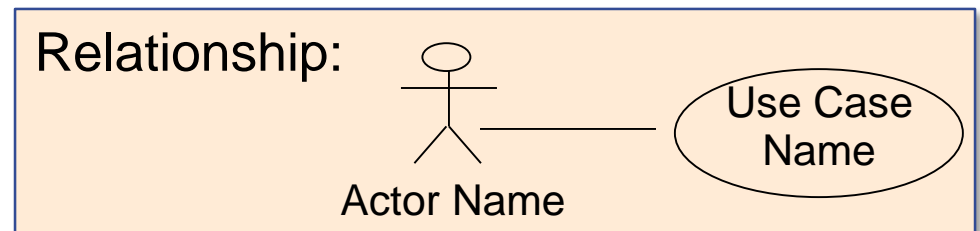
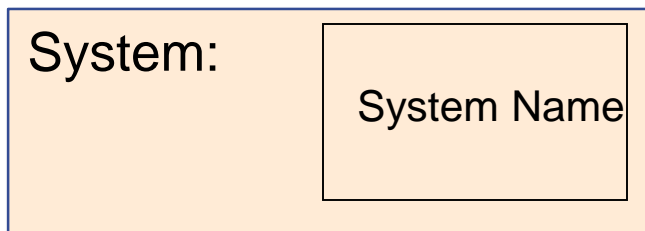
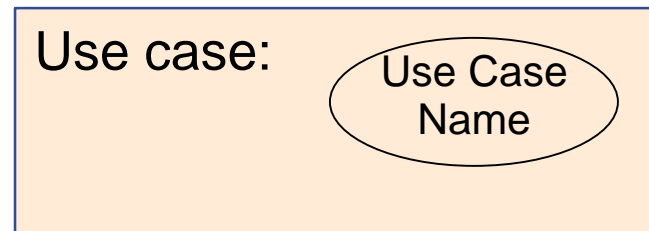
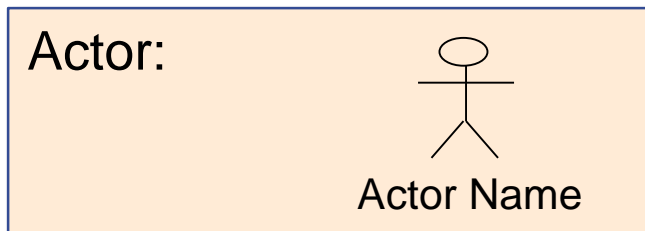


Use Case Diagram

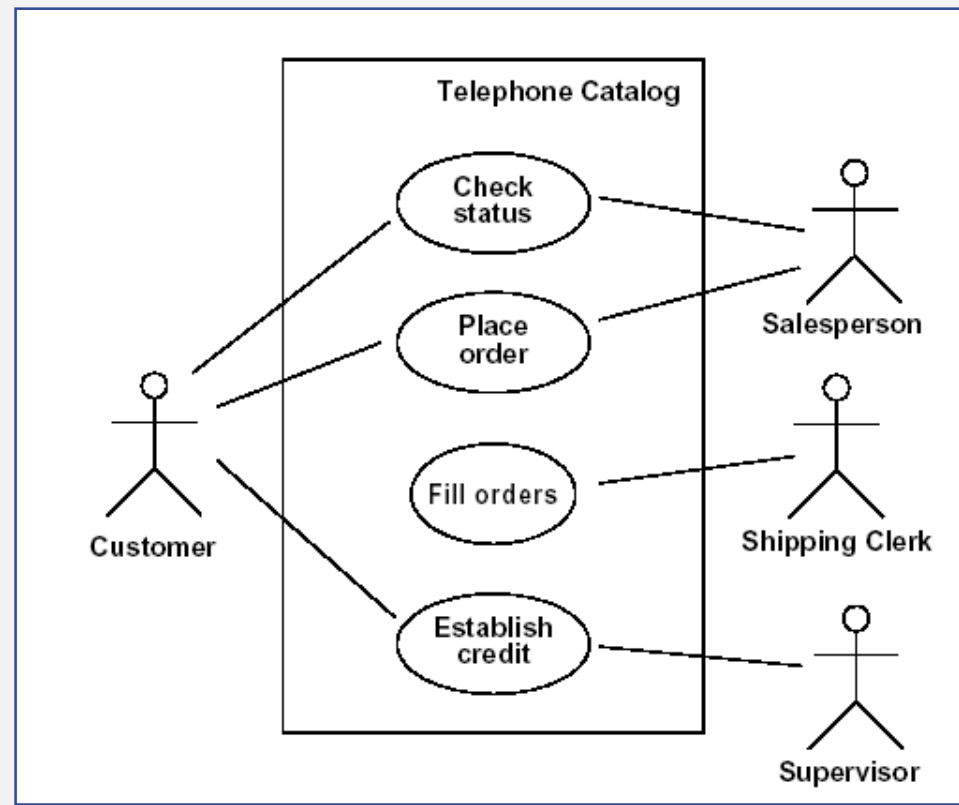
- 요구사항 강의노트 참고

Use Case Diagram

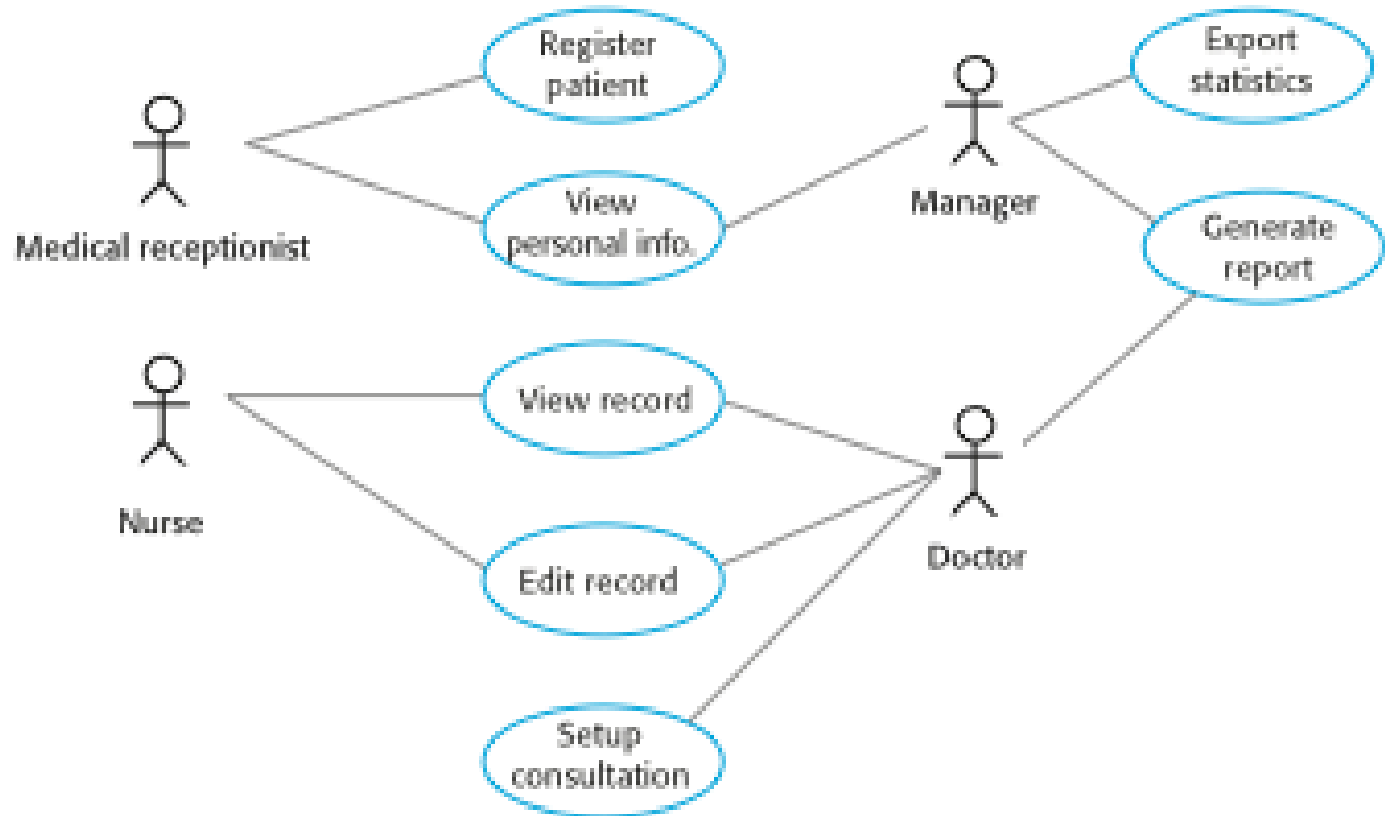
- Actor와 Use Case의 관계를 도식화
 - Actor: 시스템에 대한 사용자 (non-human actor도 가능)
 - Usecase: User-visible function
- Use Case는 시스템이 제공하는 기능을 나타내며, Actor에 의해 바라보는 관점으로 표현된다.



- **Example - Usecase diagram**



예) Use cases for the MHC-PMS



- **Use case modeling**

- 시스템과 사용자간의 인터랙션 식별
 - 각각의 유스케이스는 액터가 정의되어야 한다.
 - 모든 유스케이스를 취합하면 시스템 모습이 된다.
- 고객이 이해하기 쉬움
- 요구사항을 모델링
 - (고객으로 부터) 요구사항을 추출하는 초기단계 동안 수행
 - 프로젝트에 대한 요구 사항을 정의하는 과정에서 상위 수준의 유스케이스 모델 작성
- 소프트웨어 테스트 시나리오 작성의 기반

• Tips for Actor and Usecase

– Actor 추출에 필요한 질문들

- 시스템의 주요기능을 누가 사용하는가?
- 시스템의 운영관리 및 유지를 담당하는 사람은 누구인가?
- 시스템이 만들어내는 결과물을 사용하는 사람은 누구인가?
- 시스템과 연관되어 함께 상호 작용해야 하는 시스템은 무엇인가?
- 시스템과 연관되어 함께 동작해야 하는 하드웨어 장치는 무엇인가?

– Usecase 식별시 고려사항

- 유스케이스는 어떤 일이 처리되는 각 단계가 아니다.
- 유스케이스는 시스템의 도움을 받아 처리하고자 하는 Actor의 요구작업

Scenarios

- Scenarios are real-life examples of how a system can be used.
- Usecase Scenario
 - 유스케이스 이름
 - 유스케이스를 시작하는 행위자
 - 유스케이스의 목표(Optional)
 - 유스케이스가 시작하는데 필요한 선행 조건(Optional)
 - 정상 시나리오 (A description of the normal flow of events)
 - 예외 시나리오 (Optional) (A description of what can go wrong)
 - 동시 발생 가능한 내용 (Information about other concurrent activities)
 - 유스케이스 종료 조건(Optional) A description of the state when the scenario finishes.

예) MHC-PMS: collecting medical history

INITIAL ASSUMPTION

The patient has seen a medical receptionist who has created a record in the system and collected the patient's personal information(name, address, age, etc). A nurse is logged on to the system and is collecting medical history.

NORMAL

The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name and date of birth are used to identify the patient.

The nurse chooses the menu option to add medical history.

...생략

WHAT CAN GO WRONG

The patient's record does not exist or cannot be found. The nurse should create a new record and record personal information.

Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

... 생략

OTHER ACTIVITIES

Record may be consulted but not edited by other staff while information is being entered.

SYSTEM STATE ON COMPLETION

User is logged on. The patient record including medical history is entered in the database, a record is added to the system log showing the start and end time of the session and the nurse involved.

- Usecase Relationship: **Include(포함):** 유스케이스가 다른 유스케이스를 포함하는 관계 (사용관계(Use))

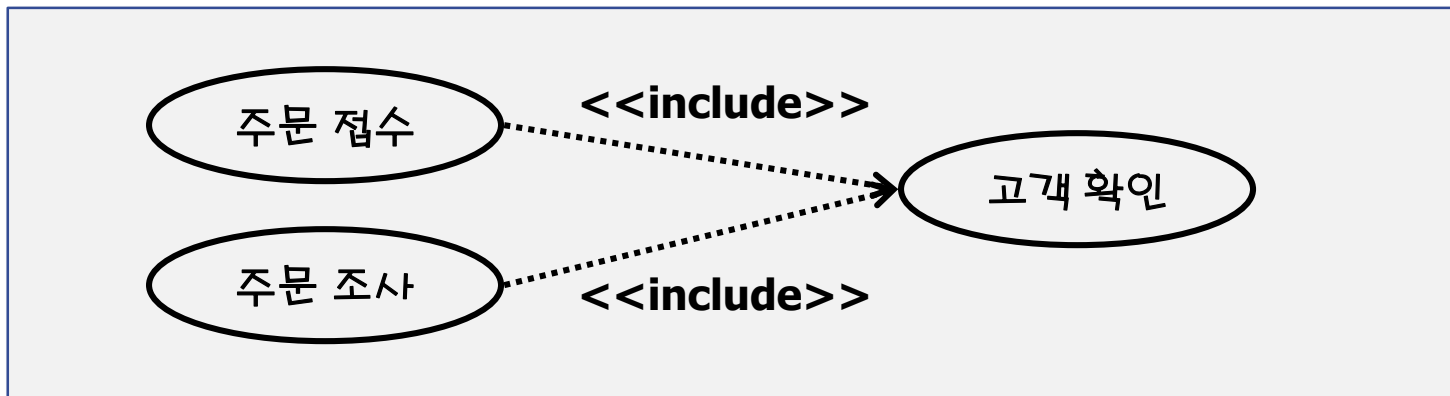
- 여러 유스케이스에서 공통으로 중복되는 시나리오가 있다면 이 시나리오를 따로 분리하여 새로운 유스케이스로 만들고, 새로 만든 유스케이스를 각 유스케이스마다 포함시킨다.
- 포함된 유스케이스는 절대로 단독으로 존재할 수 없으며, 현재 유스케이스의 부분으로만 존재한다.

<<include>>



- 포함관계에 있는 유스케이스 사이를 점선으로 잇고, 포함되어지는 유스케이스 쪽에 화살표 머리를 둔다.
- 연결선 위에는 스테레오타입 <<include>>를 붙인다.

- Example- Include Relationship



- Usecase Relationship: **Include(포함):** 유스케이스가 다른 유스케이스를 포함하는 관계 (사용관계(Use))

- 여러 유스케이스에서 공통으로 중복되는 시나리오가 있다면 이 시나리오를 따로 분리하여 새로운 유스케이스로 만들고, 새로 만든 유스케이스를 각 유스케이스마다 포함시킨다.
- 포함된 유스케이스는 절대로 단독으로 존재할 수 없으며, 현재 유스케이스의 부분으로만 존재한다.

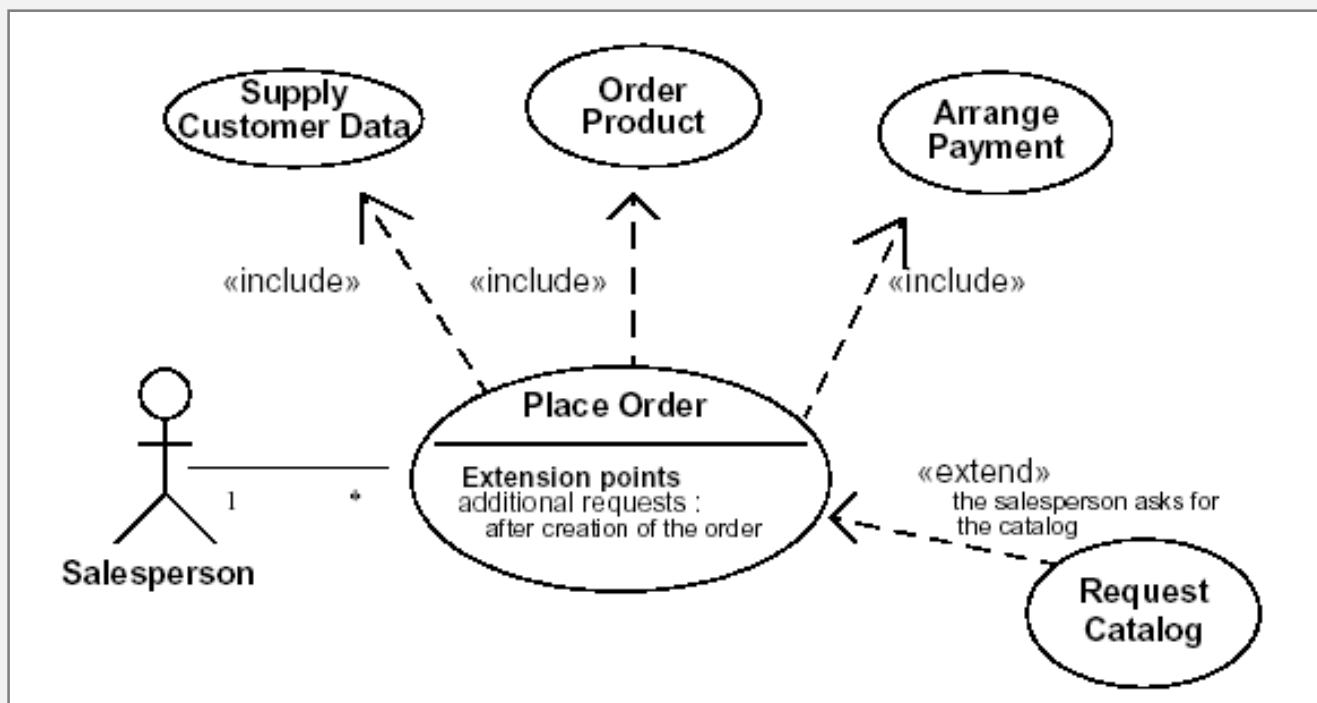
<<include>>



- 포함관계에 있는 유스케이스 사이를 점선으로 잇고, 포함되어지는 유스케이스 쪽에 화살표 머리를 둔다.
- 연결선 위에는 스테레오타입 <<include>>를 붙인다.

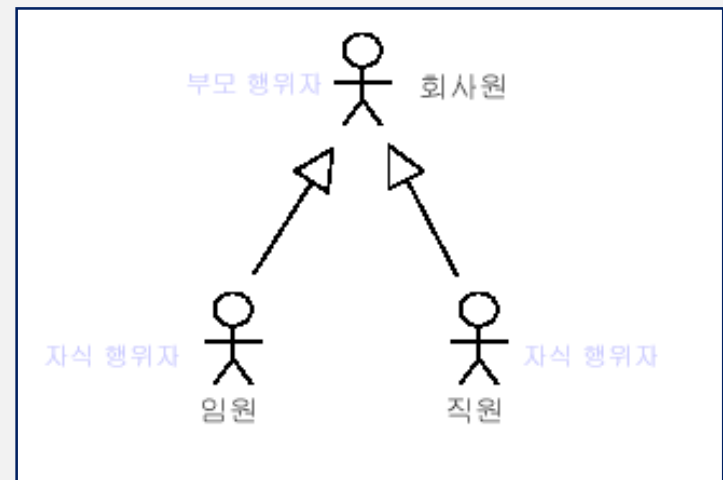
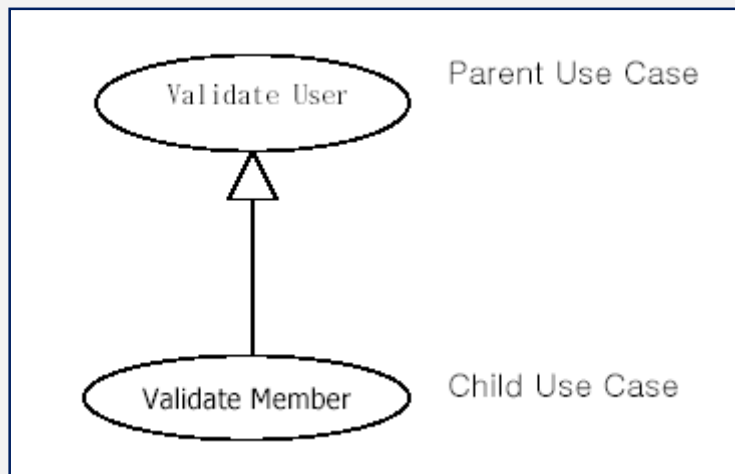
• Example- Extend Relationship

- 정상적인 흐름에서는 Place Order 기본 유스케이스가 실행되지만 확장점(Extension point)의 조건을 만족한다면 Request Catalog 확장 유스케이스가 실행된다.



- Usecase Relationship: **Generalization(일반화)**

- 유스케이스를 상속하는 것을 의미
- 자식 유스케이스는 부모 유스케이스의 모든 행동과 의미를 물려 받으며, 여기에 자신 만의 행동을 추가할 수 있다.
- 일반화 관계는 Actor 사이에도 적용할 수 있다.





Class Diagram

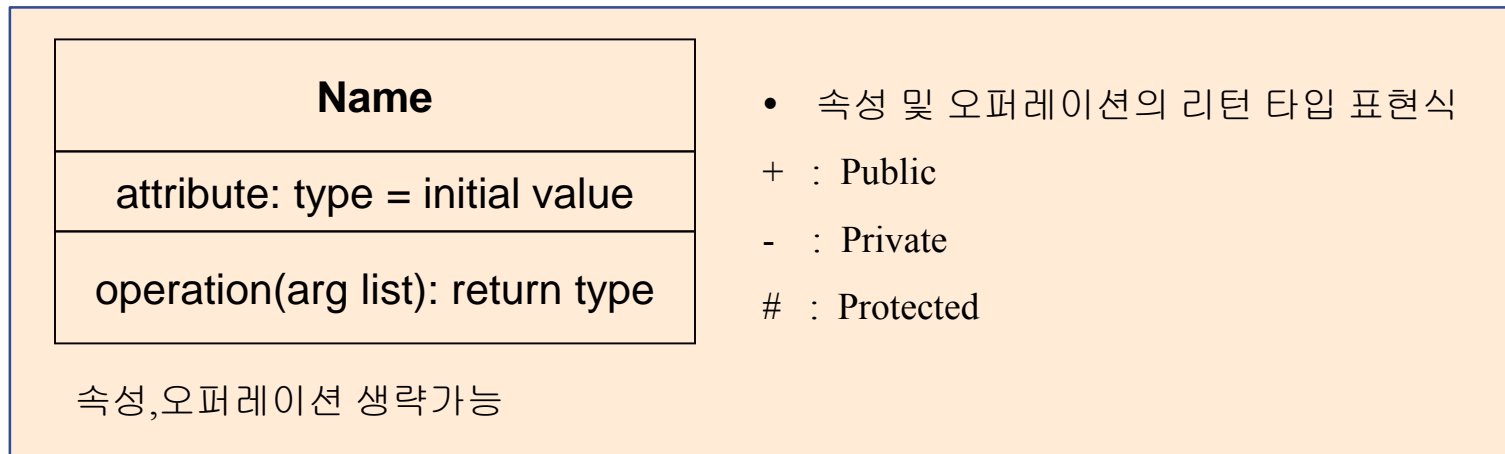


Class Diagram

- 클래스도는 클래스들과 그들의 관계(Relationship)를 표시

- Class

- 클래스는 임의의 유사한 객체들을 명세하기 위한 스펙
- 각 객체들을 생성해내기 위한 **템플릿**으로 사용
- 속성 (attribute) + 메소드 (method, operation)



• Class 를 추출하는 Tip

- 하나의 **주제**(theme)를 갖고 있어야 한다.
- 클래스 이름은 추상화를 가장 특징화하는 하나의 **명사**이어야 한다.
- 의뢰인과의 대화(분석 도출과정)에서 명사는 클래스 이름이 될 가능성이 높고, 동사는 클래스의 Operation (method)가 될 가능성이 높다.
- 고객, 사용자와의 상담(시나리오) + 문제 도메인 지식 + 기술적 경험 + 일반적 상식에서 추출한다.

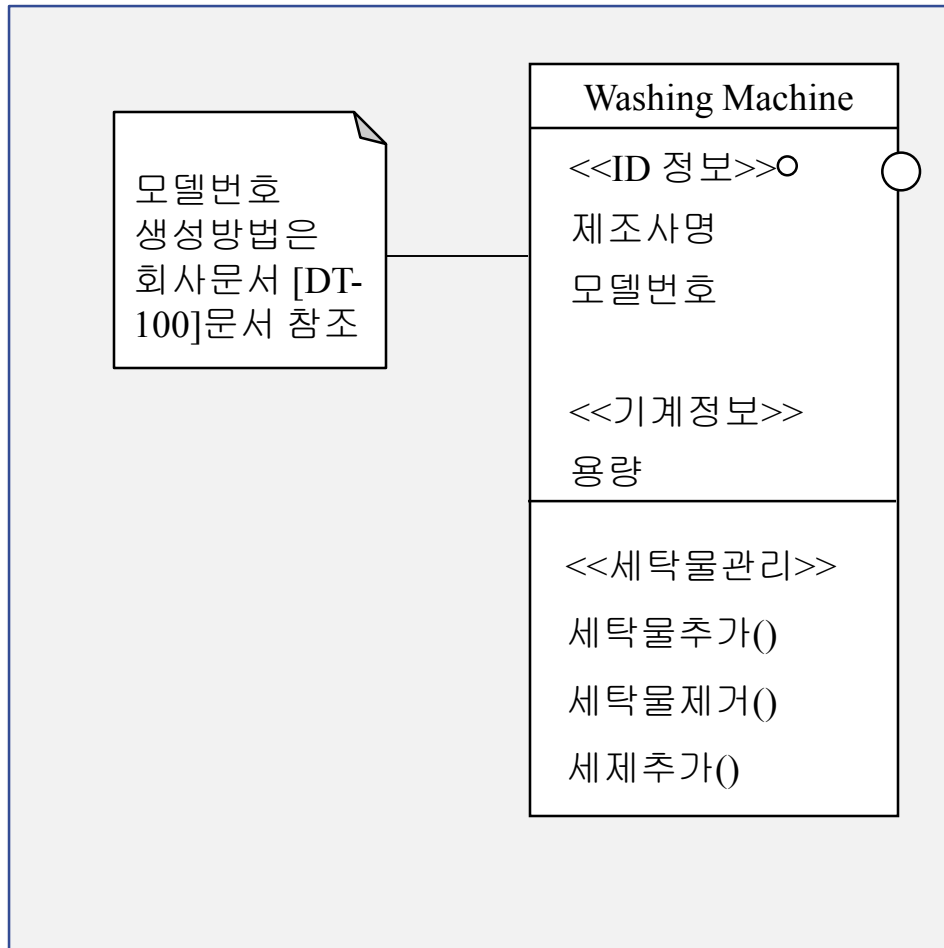
Step 1: 명사 추출 → 클래스 후보

Step 2: 클래스로 선정된 명사와 관련된 명사와 형용사 추출 → 속성 (attribute) 후보

Step 3: 클래스와 관련된 동사 추출 → 메소드 후보

Step 4: 클래스와 클래스 사이의 관계를 나타내는 동사 추출 → 관계 (relation) 후보

• Example- 세탁기 클래스



스tere오 타입을
사용하여 속성
또는 오퍼레이션
리스트를 구분할
수 있다

• Example- Window

Window

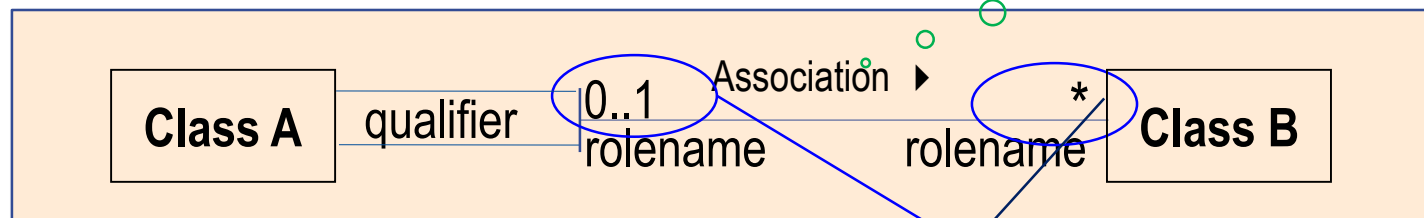
<i>Window</i>
<i>size: Area</i> <i>visibility: Boolean</i>
<i>display()</i> <i>hide()</i>

<i>Window</i>
<i>+size: Area = {100,100}</i> <i>#visibility: Boolean = true</i> <i>+default-size: Rectangle</i> <i>#maximum-size: Rectangle</i> <i>-xptr:XWindow</i>
<i>+display()</i> <i>+hide()</i> <i>+create()</i> <i>-attachXWindow(xwin:Xwindow)</i>

• Class 사이의 relationship- **Association**

- 클래스가 개념적으로 서로 연결되어 있음을 말한다.

▶ 또는 ◀ 을 사용하여 관계의 방향을 나타낼 수 있다.

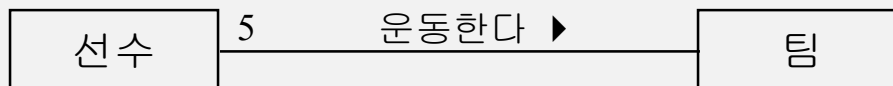


하나의 클래스가 여러 클래스와 연관을 맺을 수 있다.

Multiplicity: 연관되어 있는 두 클래스 사이에서 한 클래스의 객체와 관계를 가질 수 있는 다른 클래스의 객체 개수

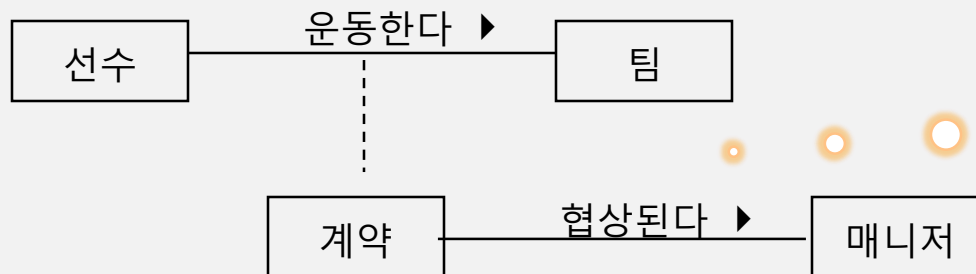
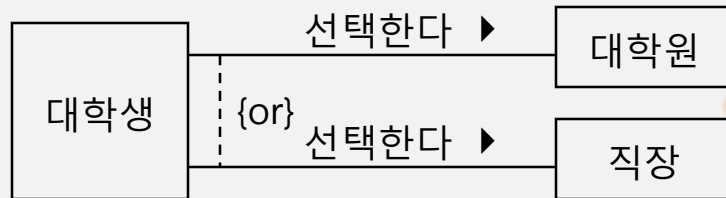
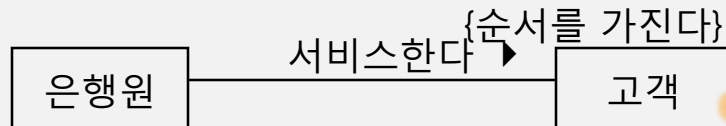
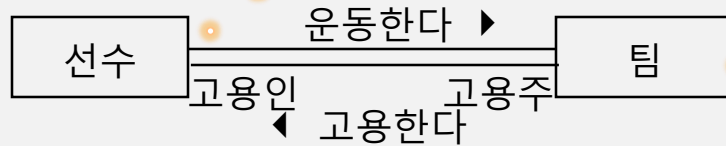
1..* → 1이상
2..7 → 2 이상 7까지
5,7 → 5 또는 7

예) 다섯 명의 선수를 갖는 팀



rolename: 각 클래스마다 역할을 표시할 수 있다.

•각 클래스마다 역할을 표시할 수 있다.



클래스 사이에 두 개의 연관을 나타낼 수 있다

연관에 제약(Constraints)을 둘 수 있다. 그림에서는 "서비스한다"에 {순서를 가진다}라는 제약이 가해져서 고객의 순서대로 은행원이 서비스 한다.

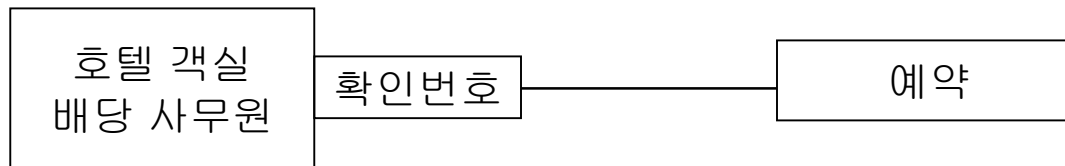
OR 제약
대학생이 진로를 정하는 상황을 모델링

•연관 클래스를 가질 수 있다.

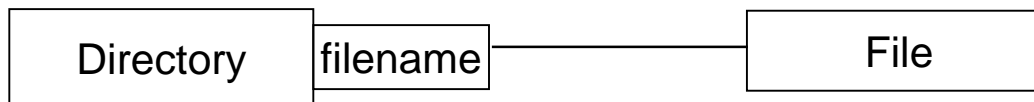
• Qualifier

- 일대 다(one-to-many)의 다중성 연관관계에서 한 객체가 특정한 객체를 가려내어야 하는 상황이 발생할 수 있다.
- 이 때, 식별정보 (Qualifier)를 지정할 수 있다.
- Qualifier는 일대 다(one-to-many)의 다중성 연관관계를 일대일(one-to-one) 다중성으로 줄이는 효과가 있다.

예) 호텔 객실 배당 사무원의 확인번호를 통해 예약을 찾는 예

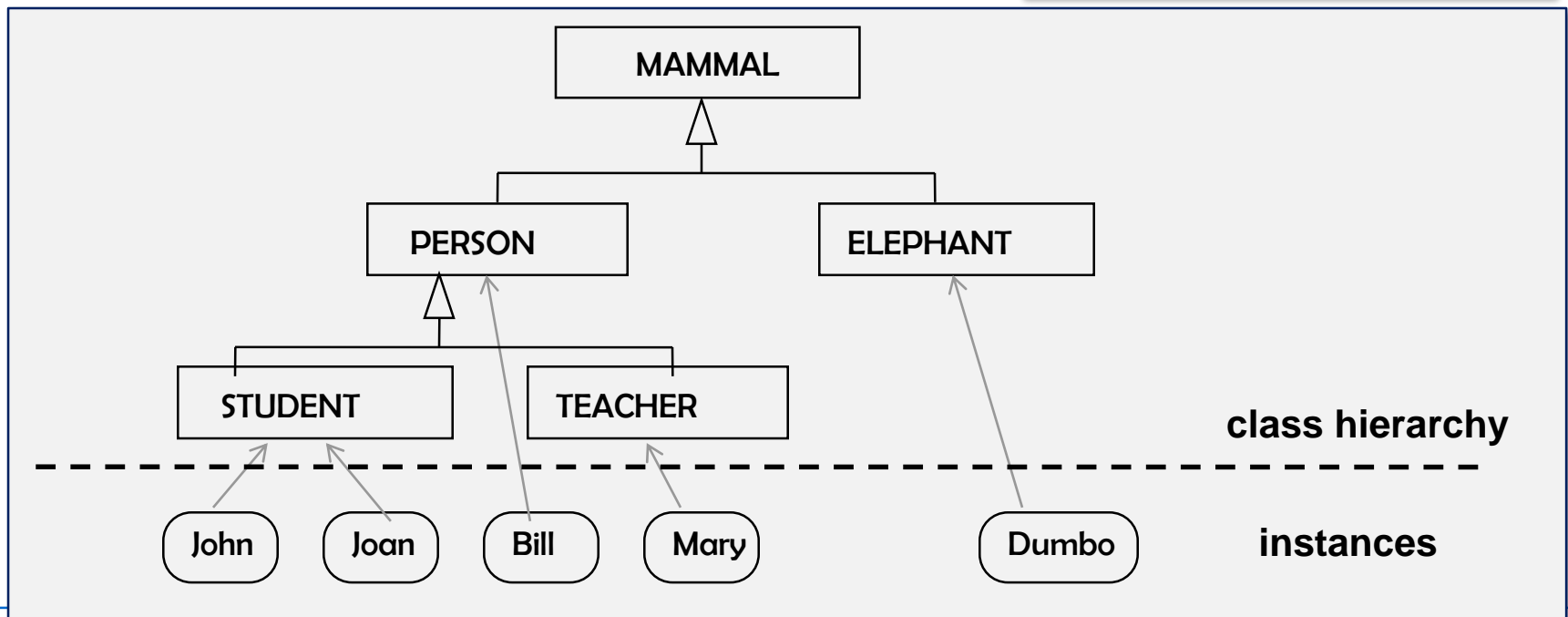
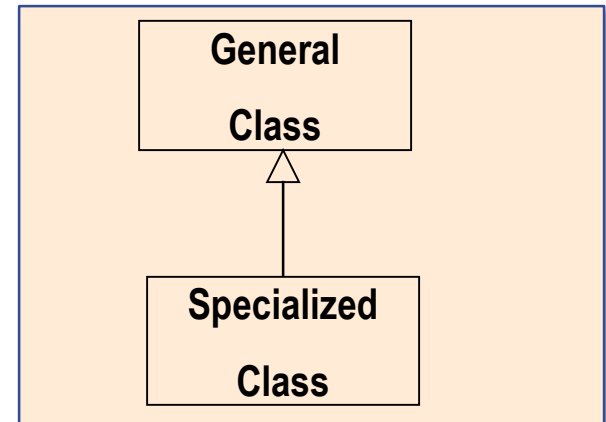


예) 파일과 디렉토리



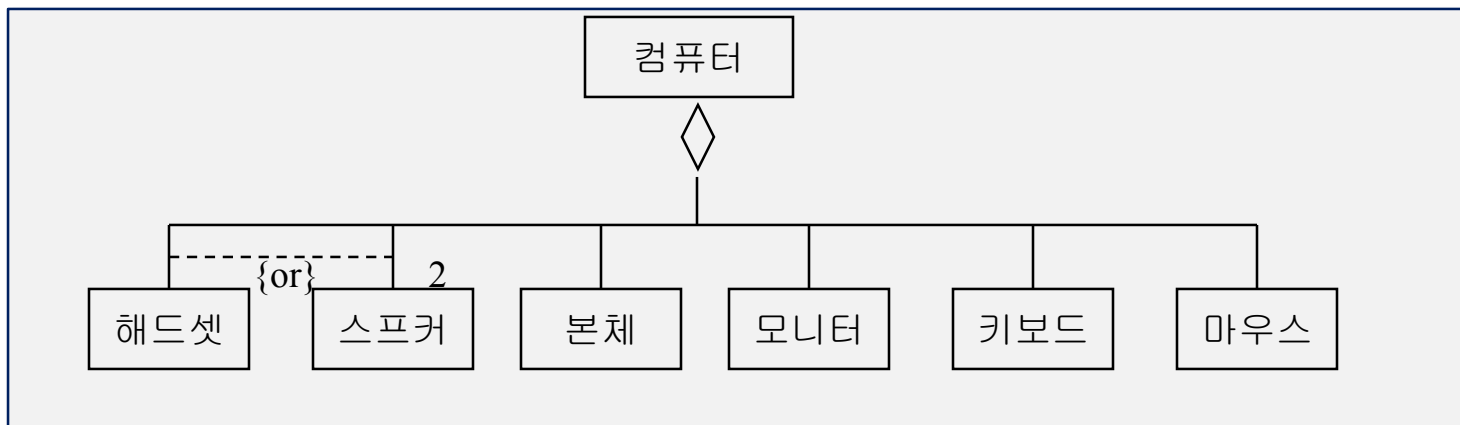
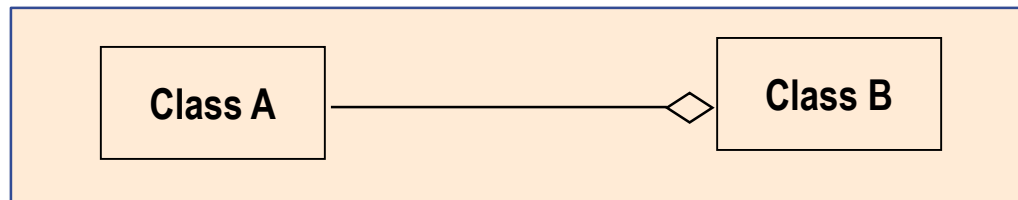
• Class 사이의 relationship- Inheritance

- Generalization
- “Is a kind of” Relation
- Root Class: super class를 가지지 않는 클래스
- Leaf Class: sub class를 가지지 않는 클래스



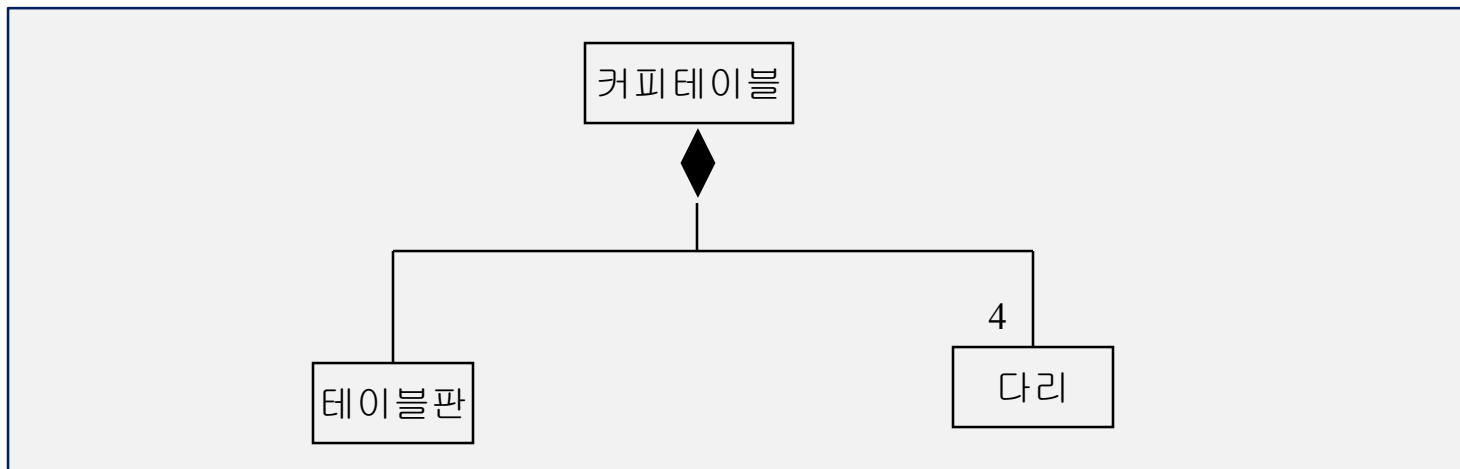
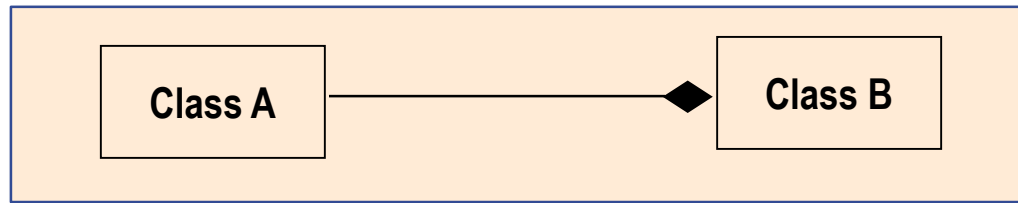
• Class 사이의 relationship- Aggregation

- 하나의 클래스가 여러 개의 클래스로 구성되어 있는 경우
- 부분-전체 (Part of). “has-a” “consists-of” relationship
- Notation: 빈 마름모



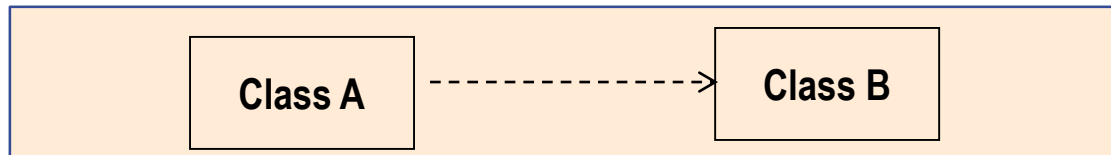
• Class 사이의 relationship- **Composition**

- 강한 집합연관으로써 각 sub 클래스가 오직 하나의 super 클래스에 대하여만 의미를 가질 때.
- Notation: 채워진 마름모

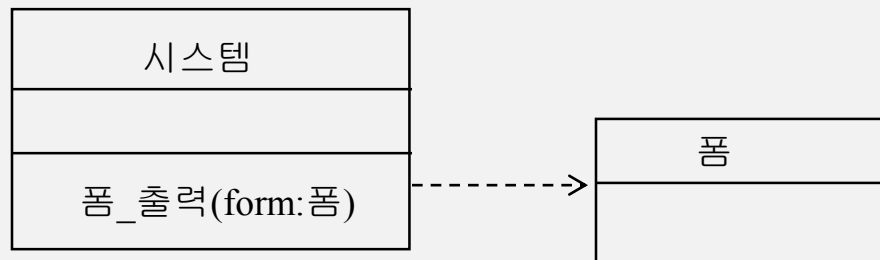


• Class 사이의 relationship- Dependency

- 한 클래스가 다른 클래스를 사용하는 관계를 말한다. 특히 한 클래스의 Operation이 다른 클래스를 사용하는 경우가 보일때이다.



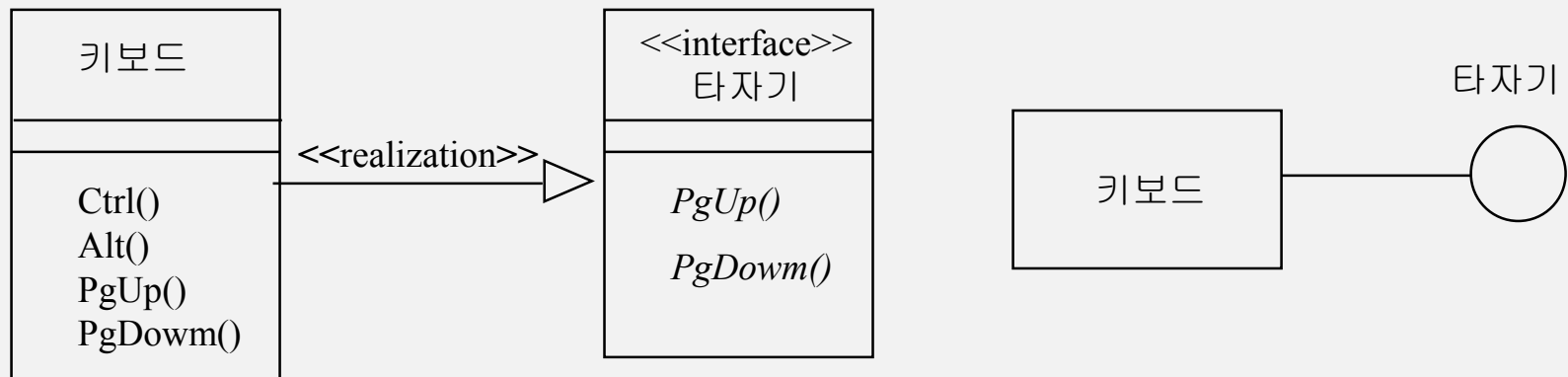
예를 들어 “시스템” 클래스와 “폼” 클래스가 있는데, “시스템”이 화면에 표시해 주는 서식은 전적으로 사용자가 선택한 “폼” 클래스에 따라 달라진다.



• Interface and Realization

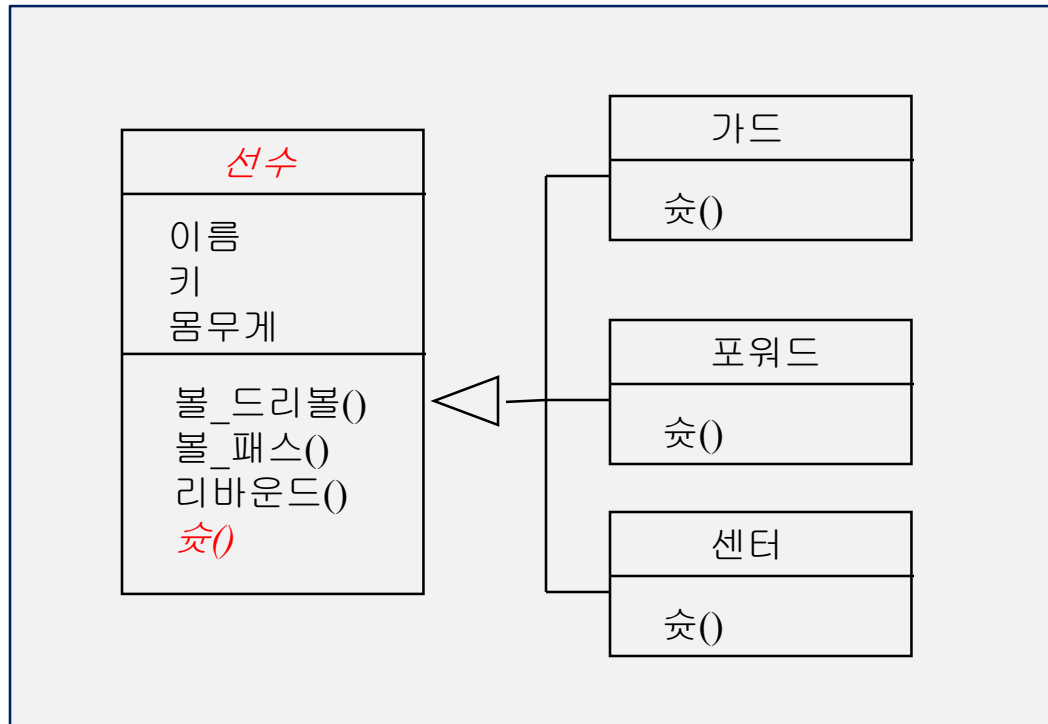
- 어떤 클래스들이 같은 signature를 가진 operation이 존재한다면, 이런 operation의 집합을 인터페이스(interface)라 한다.
- 인터페이스는 클래스의 일정한 행동(behavior)를 나타내는 operation의 집합으로, 다른 클래스에서 사용될 수 있다.

예) 타자기의 행동을 실체화한 키보드 클래스

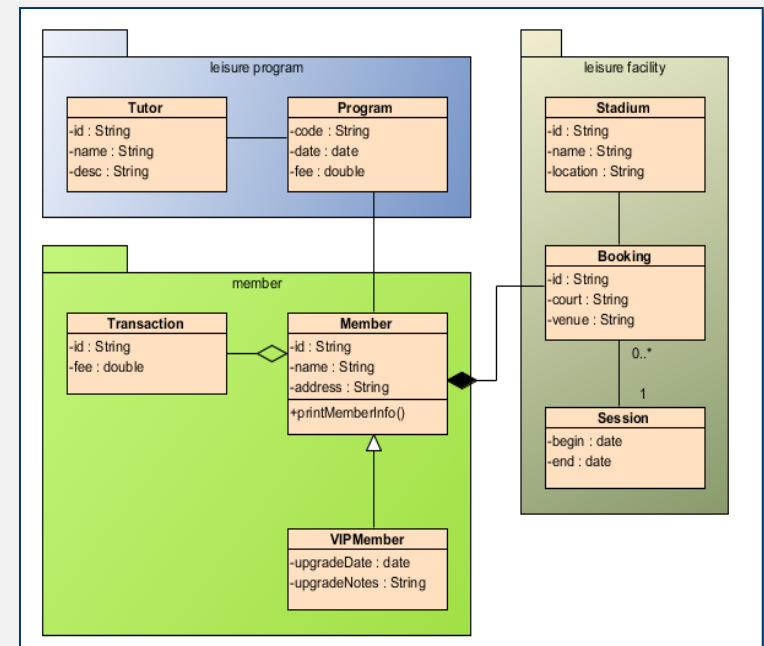
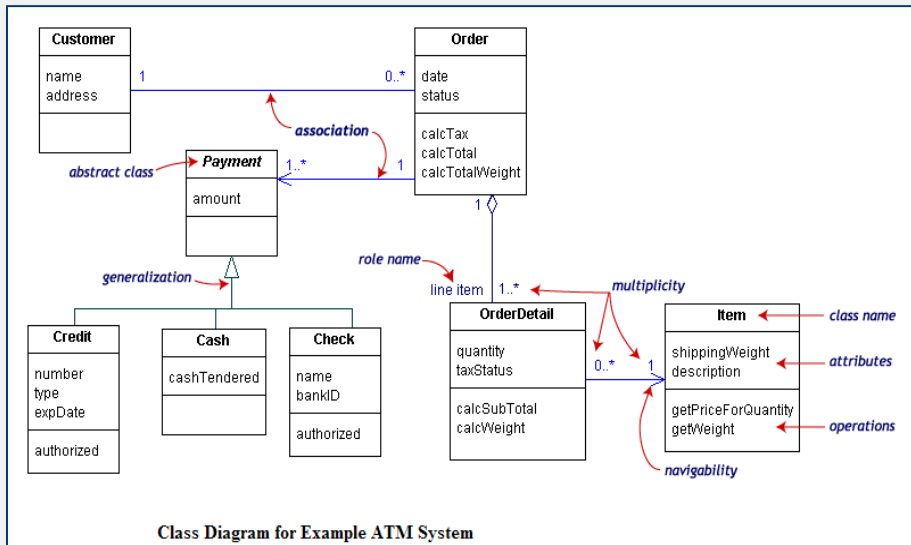


• Abstract Class

- 객체를 생성하지 않는 클래스
- Notation: 이태릭체 클래스명

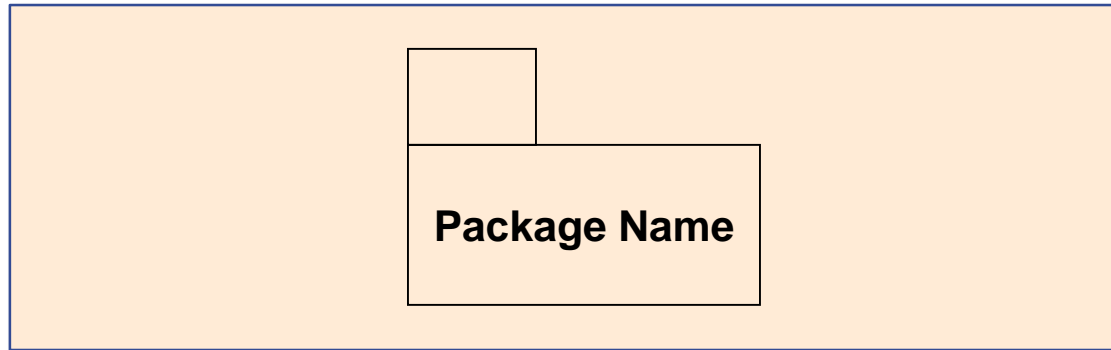


• Class Diagram: Example



Package Diagram

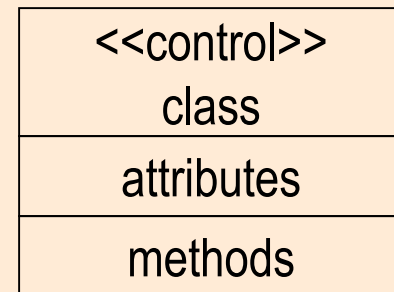
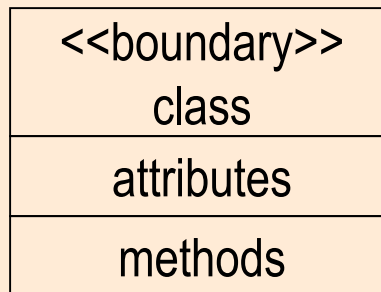
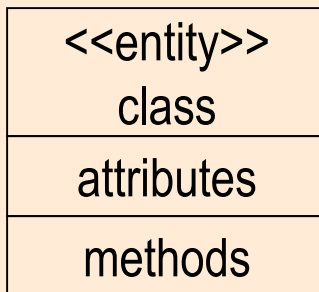
- 대부분의 시스템은 수많은 클래스로 이루어져 있다. 패키지는 연관된 클래스들의 집합이다



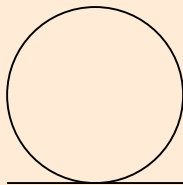
• 클래스 뷰

– Entity class / Boundary class / Control class

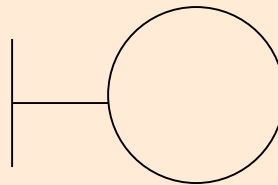
UML



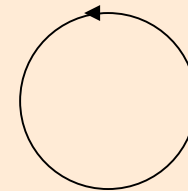
Rose 2000



entity class



boundary class



control class

MVC Architecture

• Entity Classes

– Entity class는 시스템의 중심이되는 필요한 내용을 모델링.

- 시스템 **내부적인 일**을 수행.

– Entity classes 추출

- Entity Class는 어떤 역할을 수행하기 위해 시스템에서 필요로 하는 클래스들.
- 역할을 표현하기 위해 사용된 명사나 명사구를 후보 클래스로 추출.
- 후보 클래스들에서 문제영역과 관련 없는 명사, 단지 언어적 표현, 중복된 내용을 표현한 명사 등은 제거

• Boundary Classes

- Boundary Class는 시스템 내부와 외부환경 사이의 커뮤니케이션을 다룸.
- 사용자 또는 다른 시스템과의 interface를 제공
- Boundary classes 추출
 - Flow of event를 기반으로 actor와 시스템과의 **user interface**를 정의.
 - 설계과정에서 선택된 GUI 메커니즘에 따라 정제됨.
 - 다른 시스템과의 통신을 지원하기 위해 추가.
 - 설계과정에서 선택된 통신 프로토콜에 따라 정제됨.

• Control Classes

- Control Class는 1개 이상의 Usecase에서 나타나는 연속적인 behavior를 modeling.
- Control classes 추출
 - Control Class가 연결고리(Sequencing) 이상의 역할을 하는 것은 바람직하지 않음.
 - Control Class의 사용은 매우 주관적.



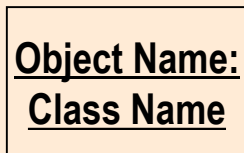
Sequence & Collaboration Diagrams

Sequence Diagram

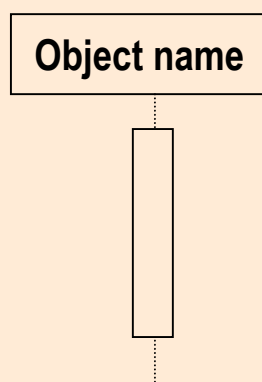
• 시간 경과에 따라 객체 상호간 교류 과정을 표현.

- Object(객체): 사각형. 가장 윗부분에 위치
- Message(메시지): 실선 화살표
- 시간: 수직선
- 각 객체로부터 아래로 뻗어 가는 점선은 객체의 lifeline(생명선)이라 함
- 생명선을 따라 좁다란 사각형 부분은 activation이라 한다. 즉 객체가 수행되고 있음. 길이는 실행 소요 기간을 나타냄.

lifeline:



activation:



message:

- Simple message: 한 객체에서 다른 객체로 제어흐름
—————→
- Synchronous message: 메시지 전송 후 수신 객체로부터 그 메시지를 받았다는 답변이 와야 자신의 작업을 계속할 수 있음
—————▶
- Asynchronous message: 메시지 전송 후 수신 객체로부터 답신을 받지 않고 작업
—————>

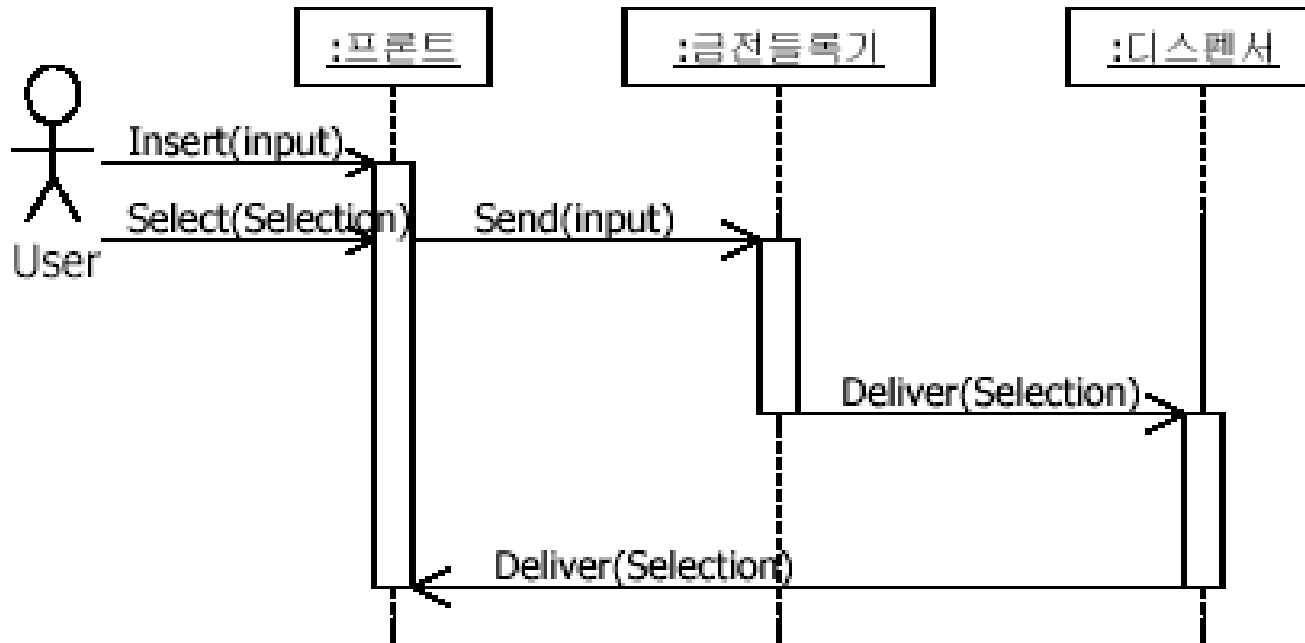
예1- Sequence diagram: 자판기에서 “음료수 사기” 시나리오1

• 객체

- 프론트: 음료수 자판기가 고객과 대화하는 유일한 인터페이스. 판매기 앞판에 있음.
- 금전등록기: 돈을 체크하고 등록함
- 디스펜서: 음료수를 따르고 내어줌

• 시나리오

1. 소비자가 자판기의 프론트 앞에 서서 투입구에 돈을 넣는다 (Insert)
2. 소비자가 마실 음료수를 고른다 (select)
3. 돈이 금전등록기에 들어간다 (send)
4. 등록기는 선택된 음료가 디스펜서에 들어있는지를 체크한다.
5. 선택된 음료수가 준비되어 있고, 등록기는 현금 잔고를 갱신한다.
6. 등록기는 디스펜서를 사용하여 음료수를 자동 판매기의 프론트로 보낸다 (deliver).

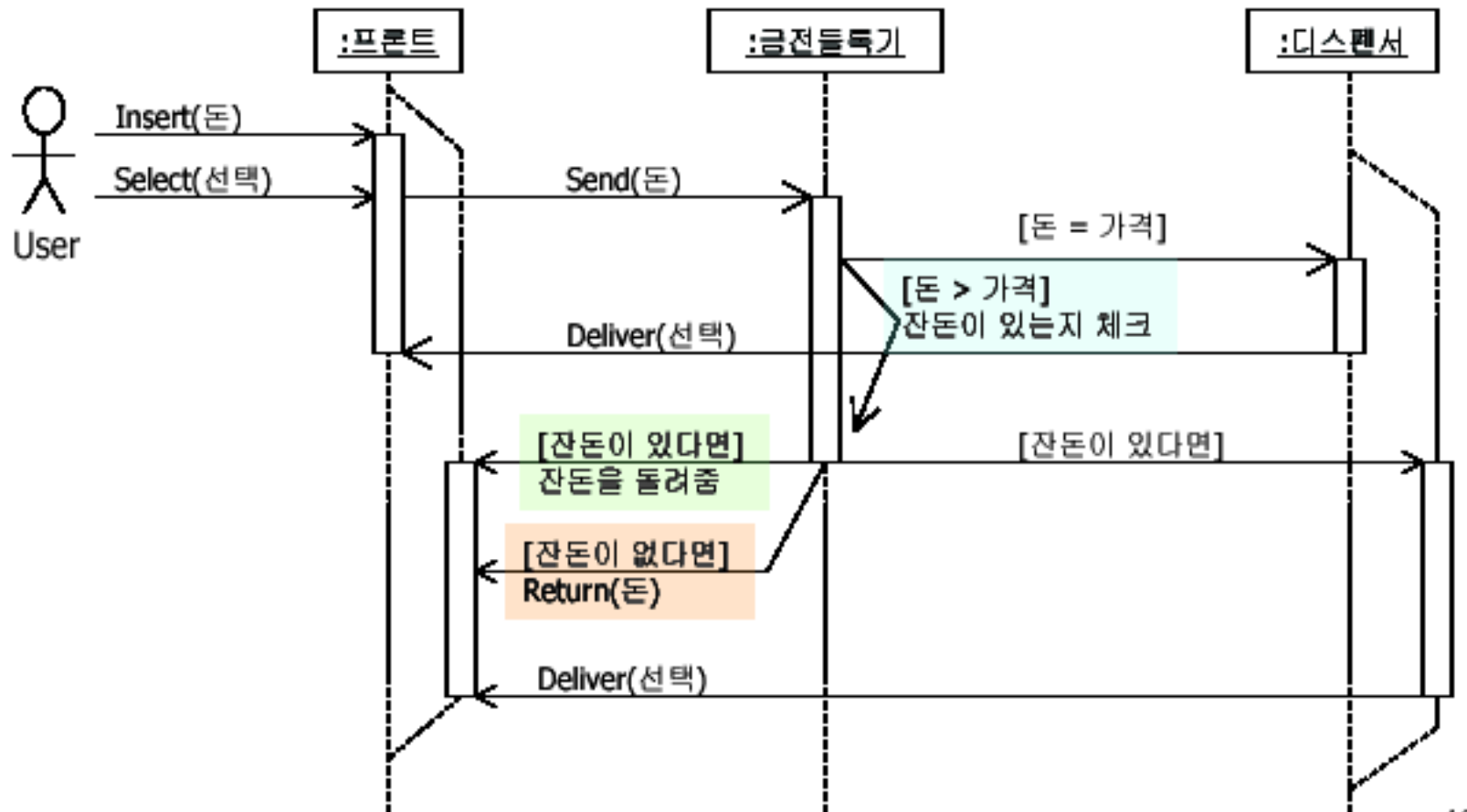


예2- Sequence diagram: 자판기에서 “음료수 사기” 시나리오2 (액수가 맞지 않은 경우)

• 시나리오

- 등록기는 소비자가 투입한 돈의 액수(input)가 음료수의 값(price)과 맞는지 체크한다.
- 만일 액수가 음료수 값보다 많으면, 등록기는 차액을 계산하고 그 만큼의 현금 잔고가 있는지 체크한다.
- 만일 차액 만큼의 현금이 잔고 (cash reserve)에 남아 있다면, 등록기는 거스름 돈(change)을 내어주고 나머지 동작은 그 전과 똑같이 진행한다.
- 만일 차액 만큼의 현금이 잔고에 남아 있지 않으면 등록기는 소비자가 투입한 돈을 그대로 돌려주고 “맞는 액수의 돈을 넣어 주세요”란 메시지를 표시한다.
- 만일 소비자가 투입한 돈의 액수가 음료수 값보다 적으면, 등록기는 아무것도 하지 않고 돈이 더 들어올 때까지 대기한다.

- “액수가 맞지 않는 경우”의 시나리오



3.60.4 Example

Simple sequence diagram with concurrent objects.

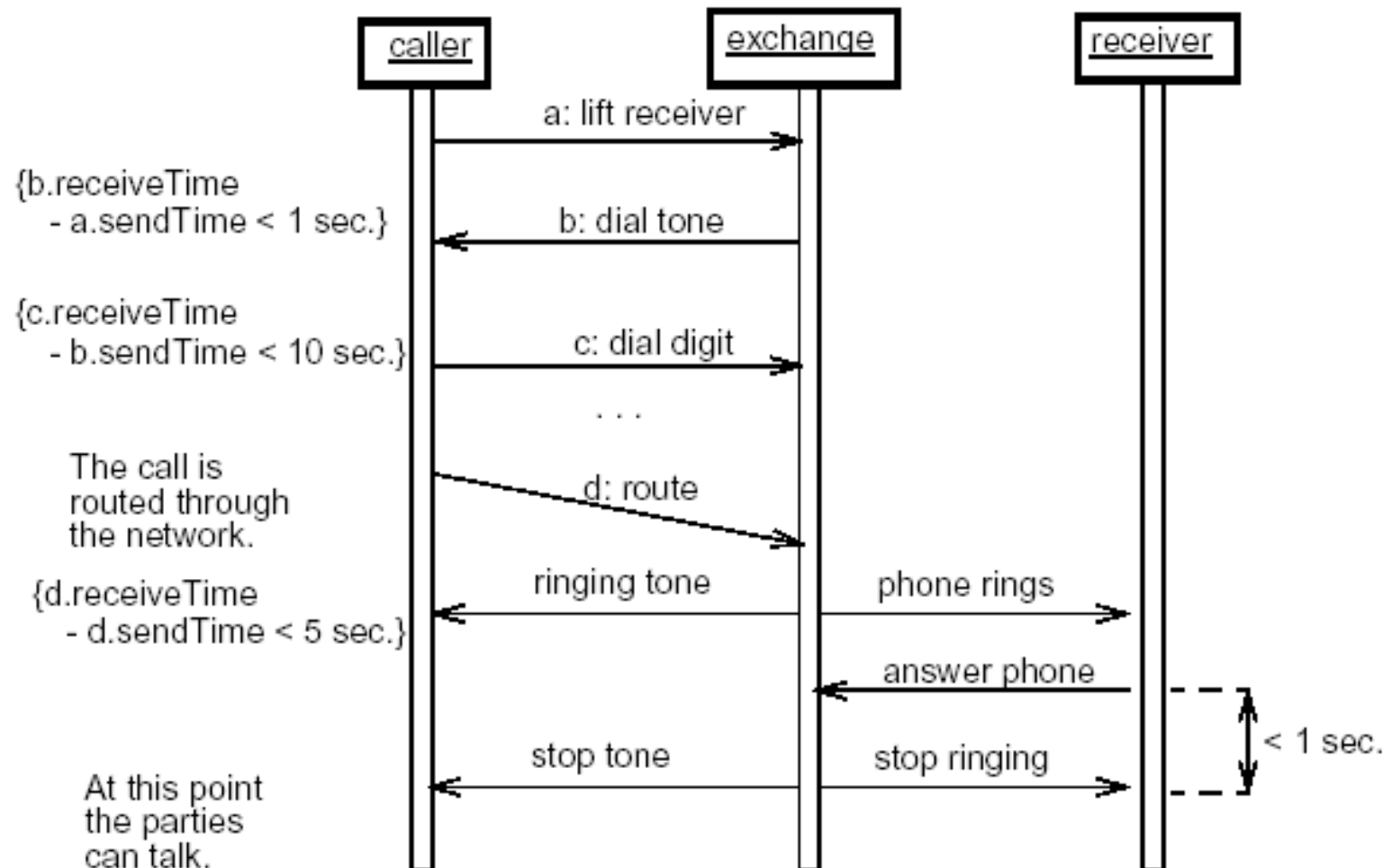


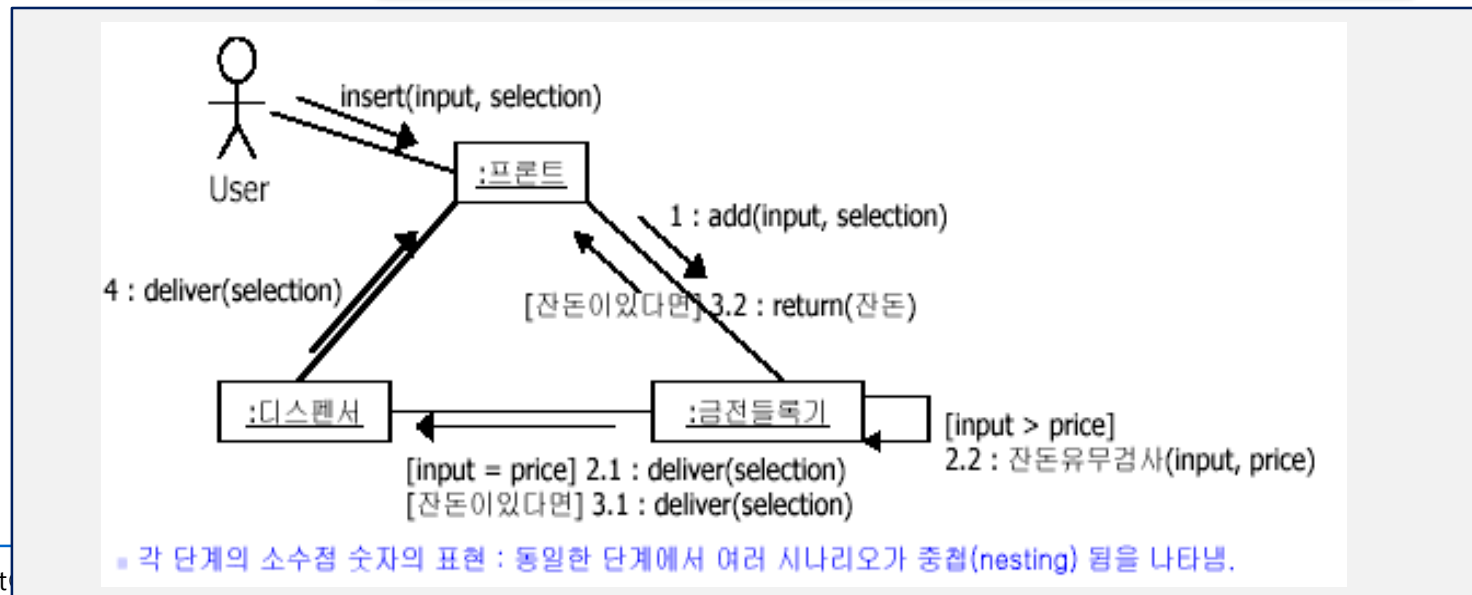
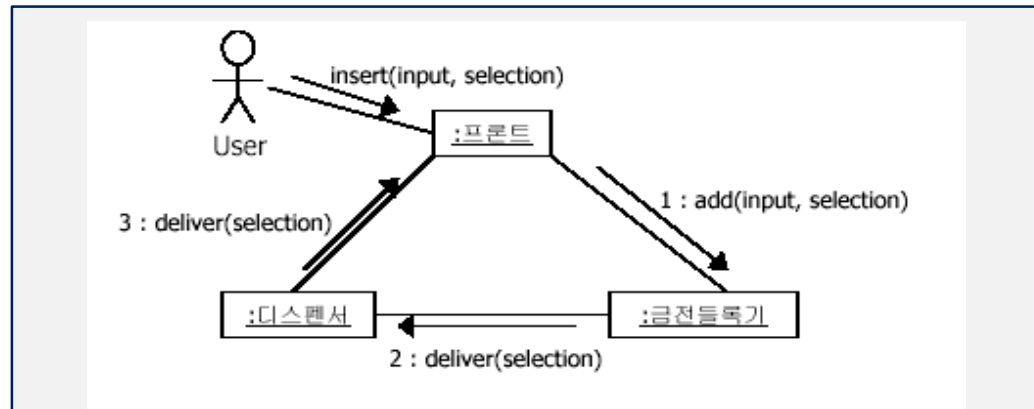
Figure 3-55 Simple Sequence Diagram with Concurrent Objects (denoted by boxes with thick borders).

Collaboration Diagram

- 객체간의 상호관계를 보여준다. (시퀀스 다이어그램은 시간 순으로 시나리오를, 콜래보레이션 다이어그램은 클래스들의 관계 파악 용이)

예1- 자판기에서 “음료수 사기” 시나리오1 ▶

예2- 자판기에서 “음료수 사기” 시나리오2
(액수가 맞지 않는 경우) ▼



※ 각 단계의 소수점 숫자의 표현 : 동일한 단계에서 여러 시나리오가 중첩(nesting) 됨을 나타냄.

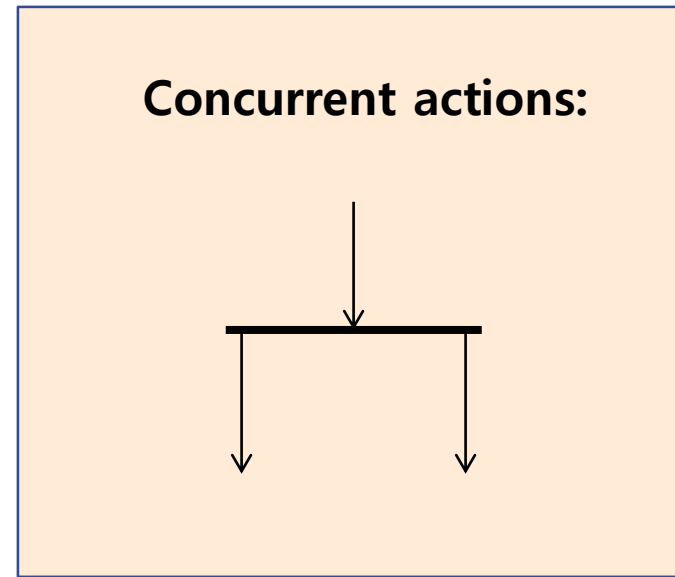
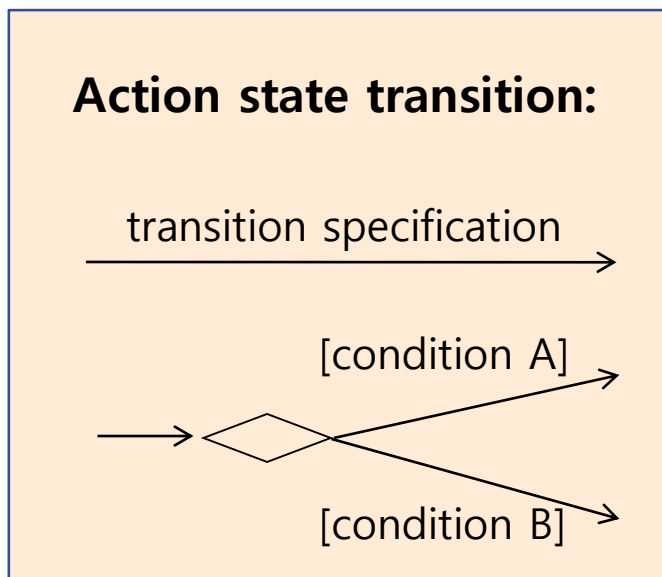
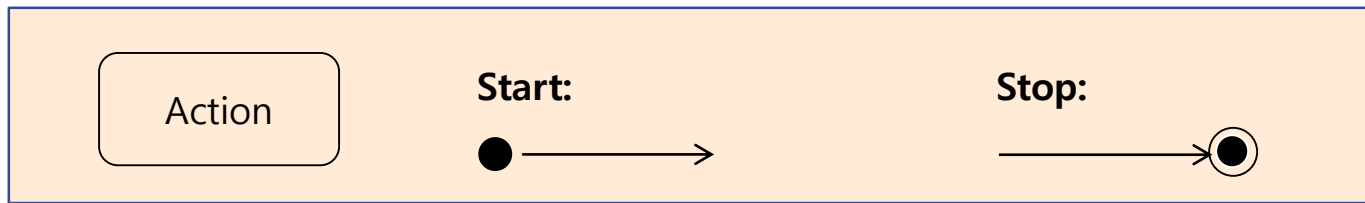


Activity Diagram

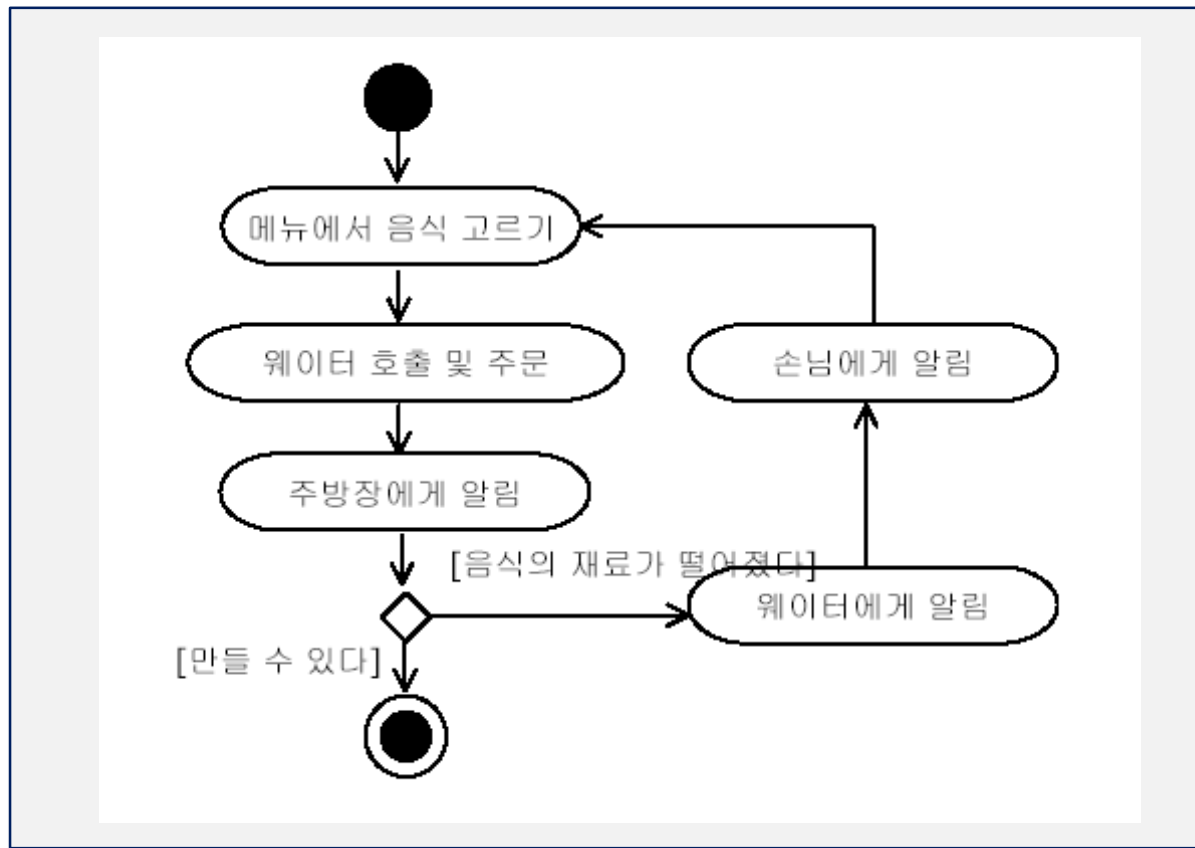


Activity Diagram

- 업무 과정(Business Process)의 활동 흐름을 표현하거나, 오퍼레이션(Operation)의 알고리즘을 나타내는데 사용함

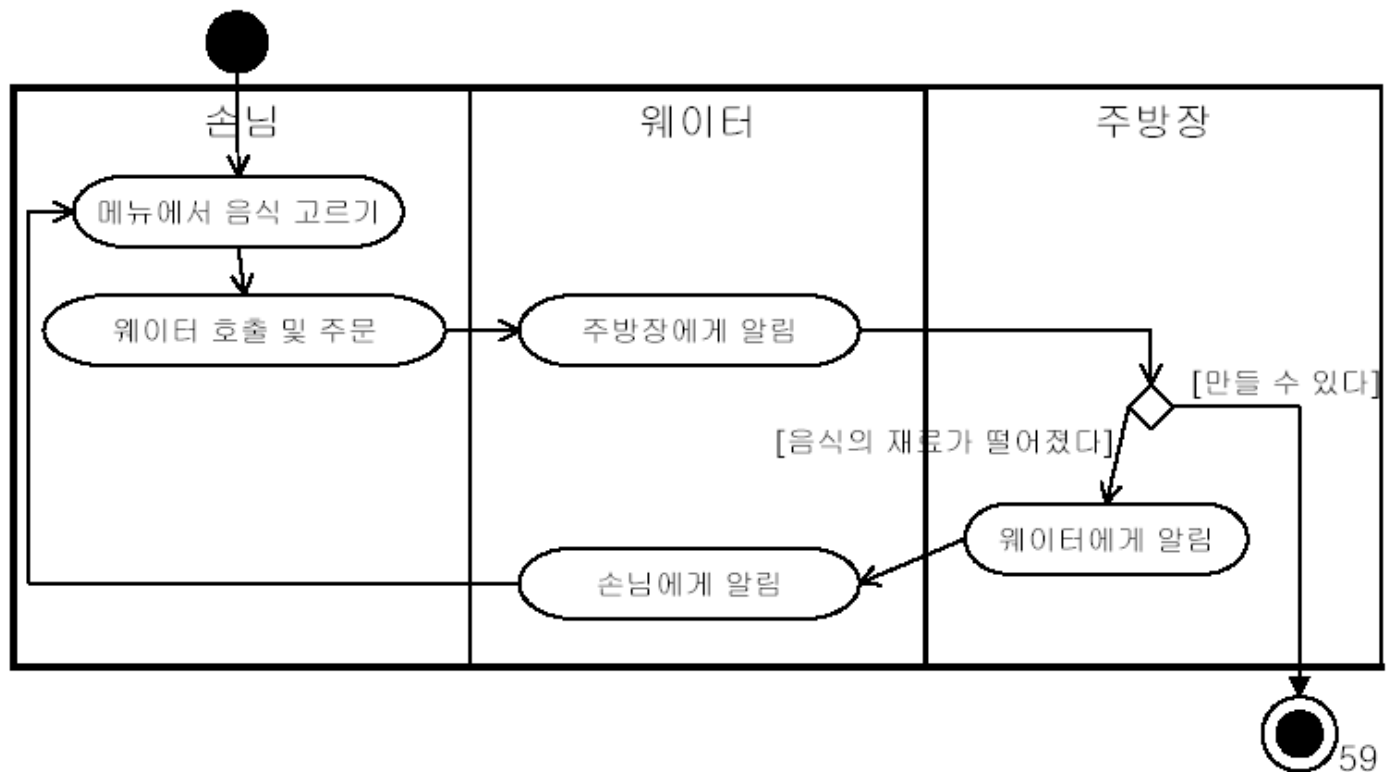


- 예- Activity Diagram - 음식점에서 주문하는 과정 (process)



• Swimlane

- Activity Diagram(활동 다이어그램)에 **역할(role)**을 표시함으로써 각 활동의 책임이 누구에게 있는지 나타낼 수 있다.





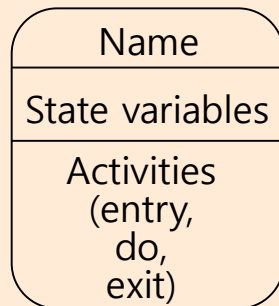
State Chart Diagram



State Chart

- 사건이나 시간에 따라 시스템 객체의 상태 변화를 표현
- 단일 객체의 상태를 나타냄
- 시스템의 변화를 잡아내기 위하여 사용함

State:

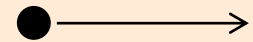


Entry - 시스템이 상태로 들어갈 때 일어남
Exit - 시스템이 상태에서 빠져나올 때
Do - 시스템이 상태 안에 있는 동안

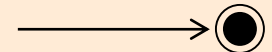
State transition:

Event[조건]/action →

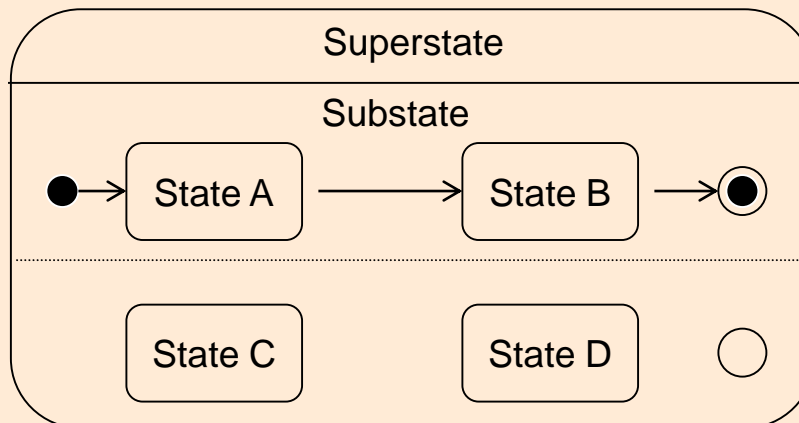
Start state:



Stop state:

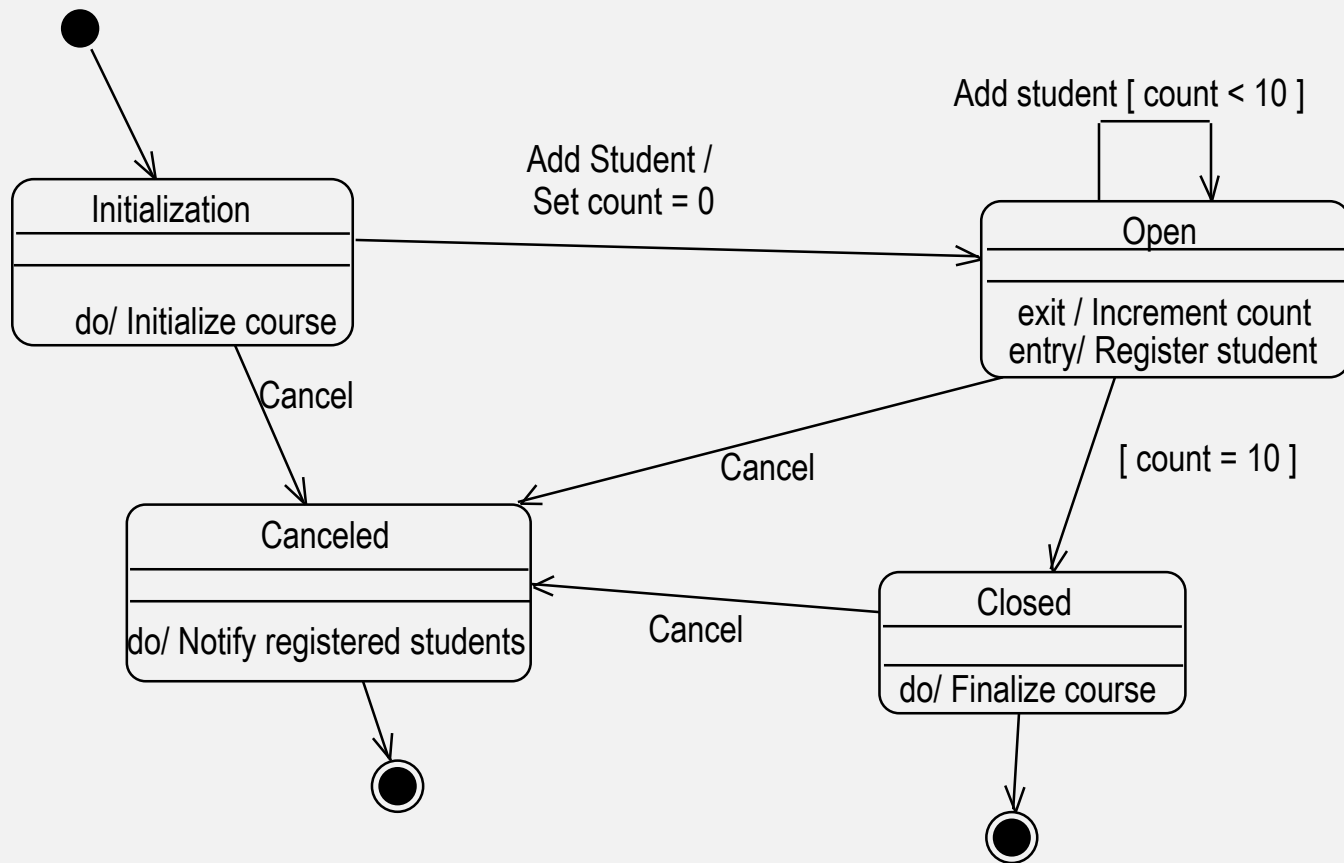


Substate:

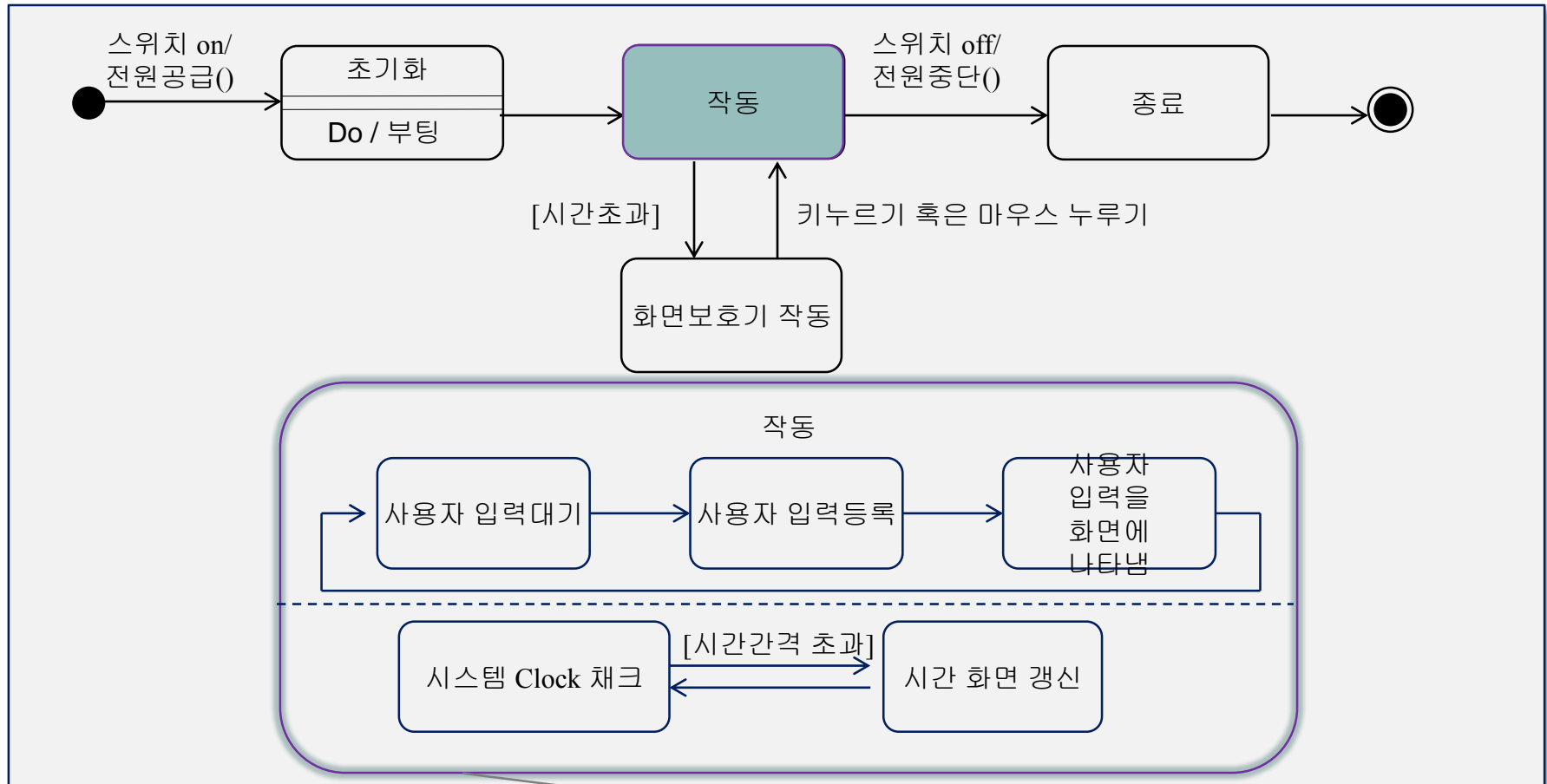


Concurrent State

• 예)



• 예) GUI State diagram



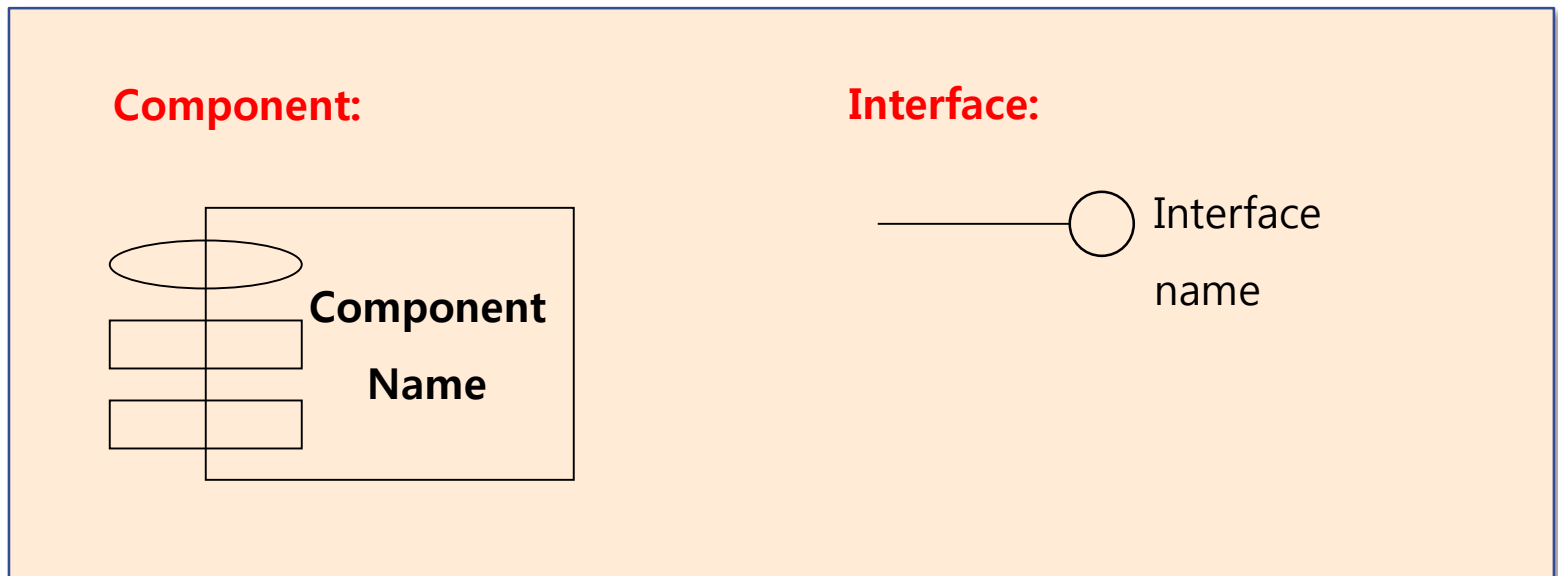
워드를 치는 동안 시계는 매 분마다 바뀐 시간 표시

-
-
-
-
-
-
-
-
-
-

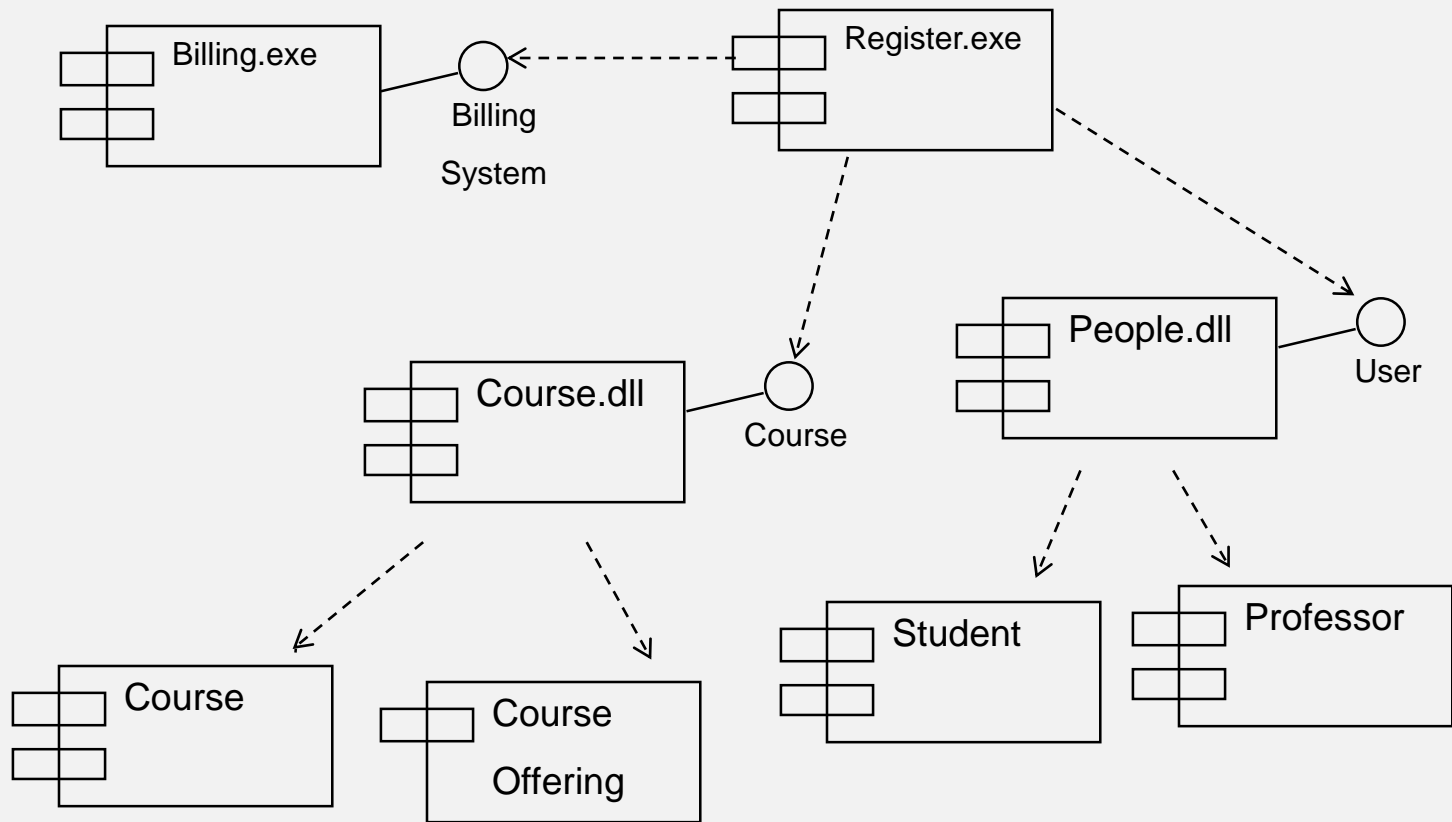
Component & Deployment Diagrams

Component Diagram

- 컴포넌트는 개발환경 내에서 실제적인 소프트웨어 모듈에 대한 구성으로 패키지와 관련된 구성요소이다. 패키지와 1대1관계가 아닐 수 있다.
- 컴포넌트
 - 소스코드 컴포넌트(.h, .cpp, .dat)
 - 런타임 컴포넌트(.dll)
 - 실행 파일 컴포넌트(.exe)

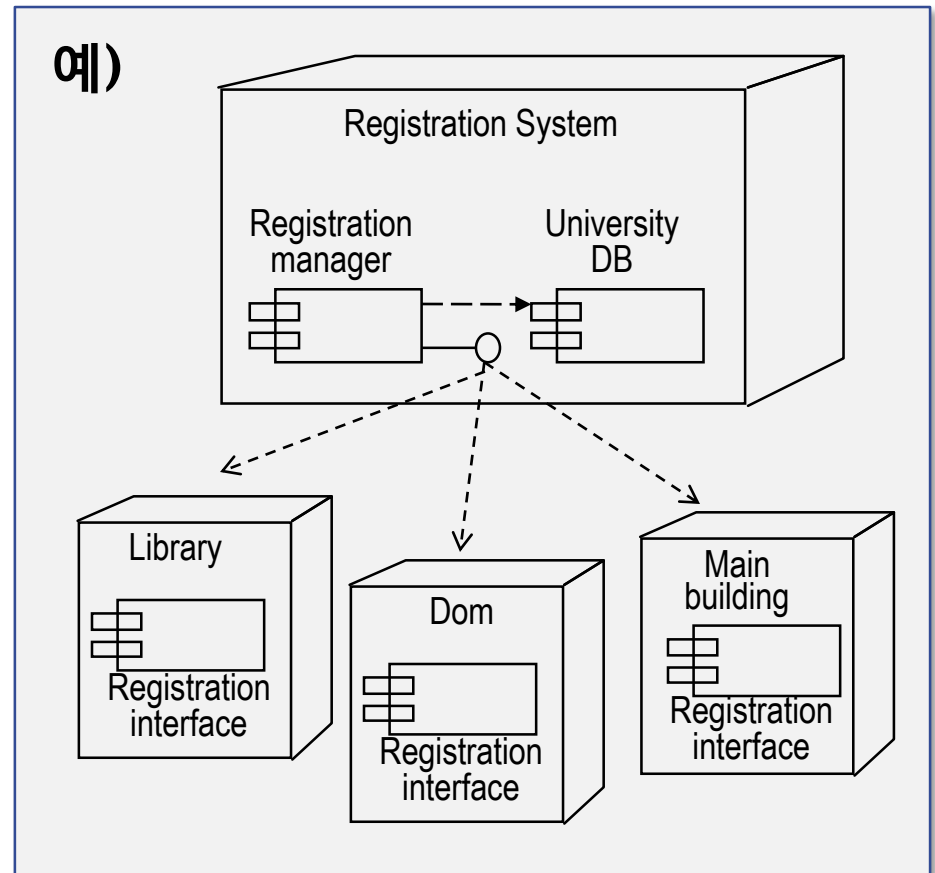
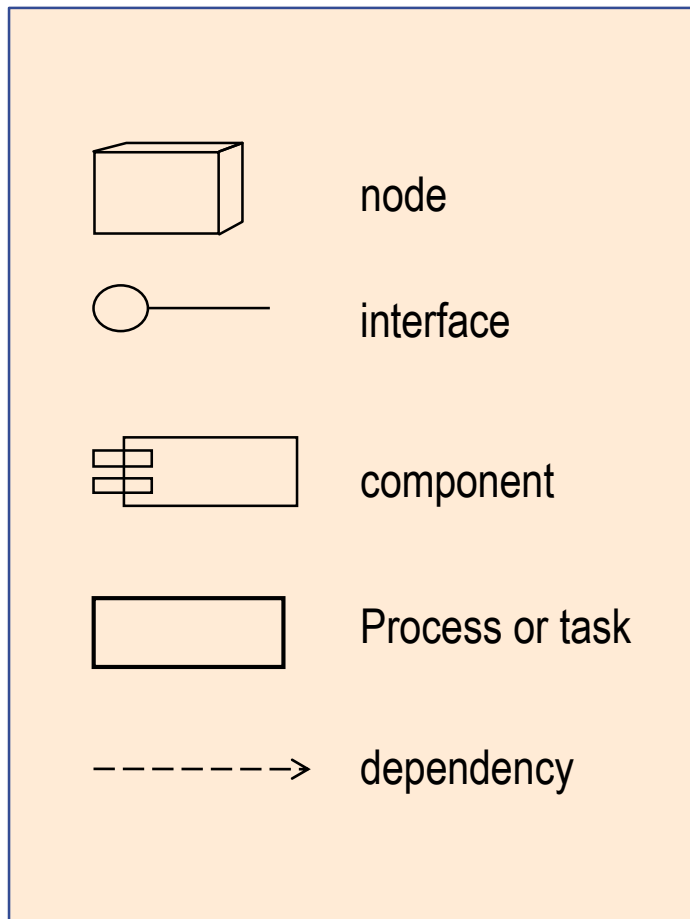


- 예) Component Diagram



Deployment Diagram

- 전체 시스템 구성 요소들의 실제 하드웨어적인 배치와 연결 상태를 표현.



UML 확장

- 스테레오타입(Stereotypes)을 사용하여 사용자 정의 타입이나 아이템을 생성하여 UML을 확장
- 스테레오타입은 관계(연관(associations), 상속(inheritance)), 클래스, 그리고 컴포넌트를 확장하기 위하여 사용
- 예:
 - 클래스 스테레오타입: boundary, control, entity, utility, exception
 - 상속 스테레오타입 : uses, extends
 - 컴포넌트 스테레오타입 : subsystem