UML 기초

- UML이란?
- UML의 개념적 모델
- UML 구성요소

Computer & Software Engineering
Lee Jong Yeol
soyeum@kumoh.ac.kr

초기의 객체지향 방법론

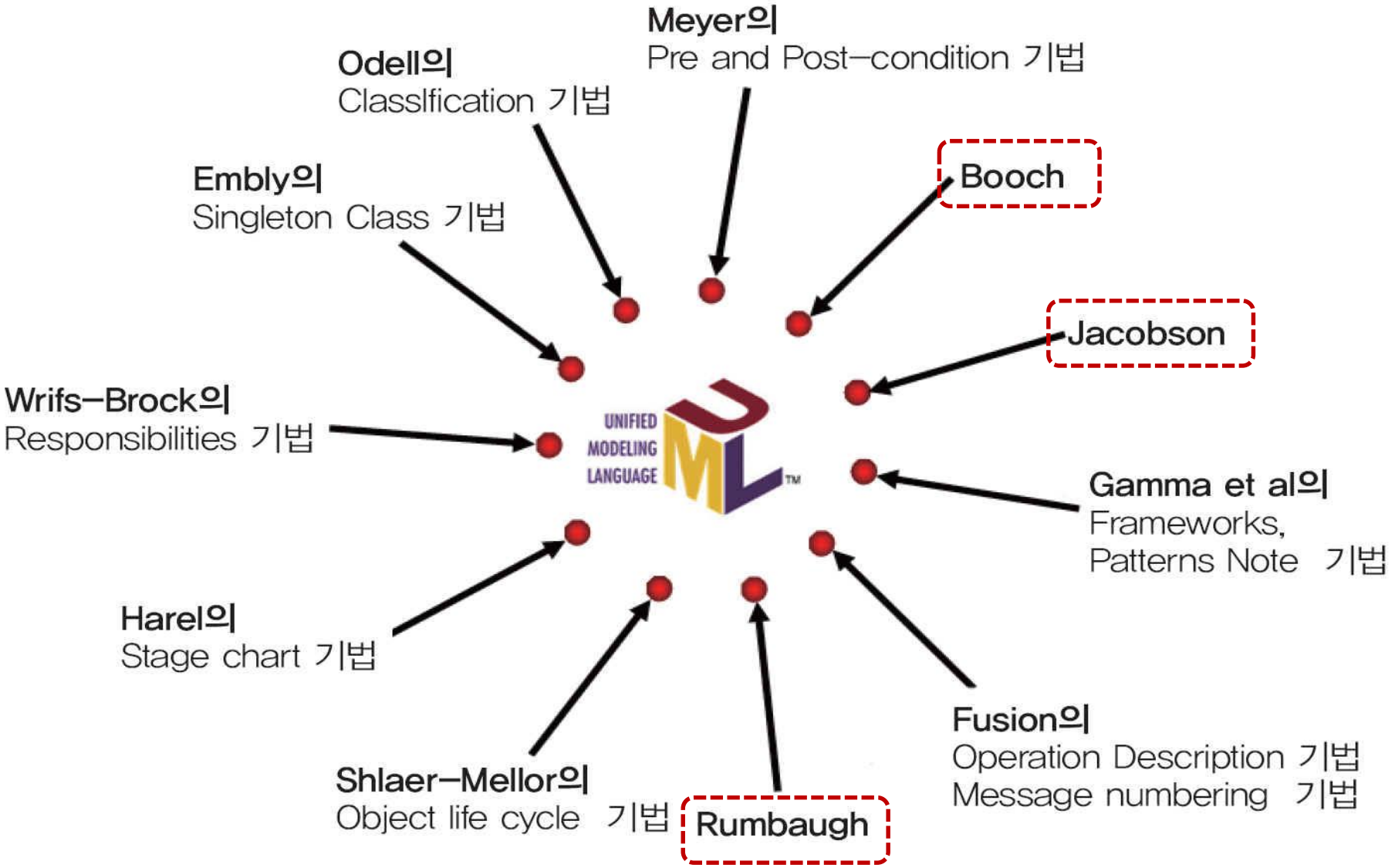
- Booch의 방법론
  - Rational Software Corporation의 Grady Booch가 제안
  - 설계 및 구현에 적합
- OOSE(Object-Oriented Software Engineering)
  - Objectory 사의 Ivar Jacobson이 제안
  - 요구 추출, 분석, 상위 레벨 설계에 적합
  - 유스케이스(use cases) 개념을 처음으로 도입
- OMT(Object Modeling Technique)
  - General Electric 사의 James Rumbaugh가 제안
  - 분석 단계에 적합, 대량의 자료를 다루는 정보처리 시스템에 강함
- Others
  - Fusion, Shlaer/Mellor, Coad/Yourdon

UML의 탄생

- 유력한 방법론의 통합 필요성 제기
  - 3가지 방법론이 서로 닮아가고 있었음
  - 객체지향 시장의 안정성 확보
  - 통합이 모두에게 유리

- UML의 목표
  - 시스템을 개발하는 모든 과정에 객체지향적 방법론을 적용하도록 함
  - 매우 크고 복잡한 시스템을 모델링 할 수 있도록 함
  - 인간과 컴퓨터 모두가 사용할 수 있도록 함

통합된 표기법


Meyer의
Pre and Post-condition 기법
Odell의
Classification 기법
Booch
Embly의
Singleton Class 기법
Jacobson
Wrifs-Brock의
Responsibilities 기법
UNIFIED
MODELING
LANGUAGE
Gamma et al의
Frameworks,
Patterns Note 기법
Harel의
Stage chart 기법
Fusion의
Operation Description 기법
Message numbering 기법
Shlaer-Mellor의
Object life cycle 기법
Rumbaugh

UML의 역사

- 1994년 10월: 통합 논의 시작
- 1995년 10월: 버전 0.8의 초안 발표
- 1996년 7월: UML 버전 0.9 발표, UML 컨소시움 결성
- 1997년 1월: UML 1.0의 표준 채택을 위해 OMG에 제출
- 1997년 11월 14일: OMG가 UML 1.1을 표준으로 채택
- 1998년 6월: OMG 개정팀에서 UML 1.2 발표
- 2000년-2003년: UML 1.3, 1.4, 1.5 발표
- 2005년 4월: UML 2.0 발표
- 2010년 5월: UML 2.3 발표
- 2011년 3월: UML 2.4 발표
- 2017년 11월 : UML 2.5 발표

http://www.uml.org/
http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/
http://www.omg.org/spec/UML/2.5.1/PDF/

UML이란?

- Well-defined Graphic Modeling Language
  - 모호하지 않고 간결하며 완전한 모델을 작성하는데 사용 가능
  - UML로 작성한 모델은 객체지향언어(Java, C++, VB, …)로 변환 가능
  - 시스템의 구조와 상세 내역에 대한 문서화 도구로 사용 가능

- 효과적으로 적용 가능한 영역
  - 기업 정보관리 시스템 분야
  - 금융 서비스 분야
  - 통신 및 운송 분야
  - 국방/방공 분야
  - 소매점 관리, 의료 기기, 과학 분야
  - 웹 기반의 분산 시스템 분야

UML의 3대 구성 요소

- Basic building blocks(기본 구조물)
  - things(事物) : p7~p44
  - relationships(관계) : p45~p55
  - diagrams(다이어그램) : p56~p57

- Rules(문법) : p58~p59
  - 기본 구조물을 조합하는 방법

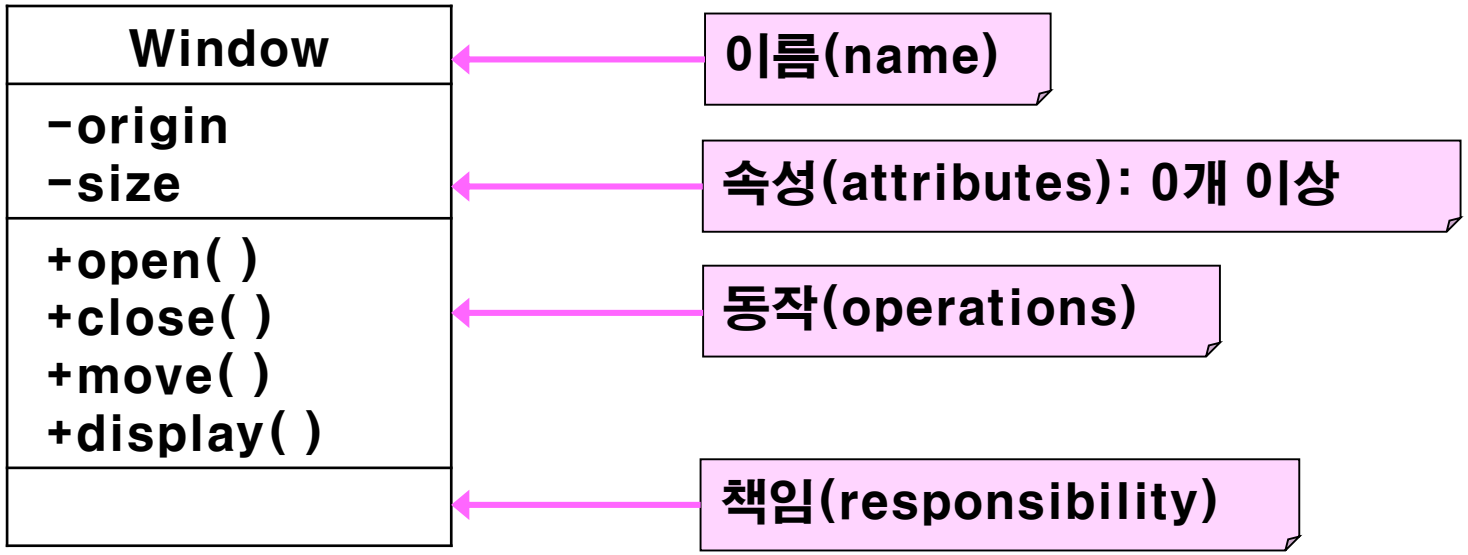
- Common mechanism(공통 메커니즘)
  - UML 전체에 일관되게 적용되는 메커니즘
  - 메커니즘의 종류
    - Specifications(설명) : p60
    - Adornments(장식) : p61~p63
    - Common divisions(공통 분할) : p64
    - Extensibility mechanism(확장 메커니즘) : p65~p71

Things

- UML로 작성한 모델에서 가장 중요한 구조물
  - Structural things or classifiers(구조 사물) : p8~p25
    - UML로 작성한 모델에서 명사(noun)에 해당하는 것
    - 물리적 or 개념적
    - Class, Interface, Collaboration, Use case, Active class, Component, Artifact, Node
  - Behavioral things(동작 사물) : p26~p41
    - 시간 및 공간의 변화에 따른 구조 사물의 동작을 표현하는 사물
    - Interaction, State machine, Activity
  - Grouping things(분류 사물) : p42
    - 시스템의 구성 요소를 grouping 하는데 사용하는 사물
    - Package
  - Annotational things(주석 사물) : p43~p44
    - 시스템의 구성 요소에 대한 주석을 붙이는데 사용하는 사물
    - Note

Class(1)

- Class(클래스)란?
  - 동일한 속성, 동작, 관계, 의미를 갖는 객체들의 집합
  - 시스템 분석 및 설계에서 가장 중요한 존재
  - 1개 이상의 인터페이스(interfaces)를 구현
  - 표기법: 직사각형. 필요에 따라 여러 칸으로 구분 가능



Class(2)

❑ 클래스 관련 문법(syntax)

- 속성: 가시성 이름 [중첩도] : 자료형 = 초기값 {특성}
  - 가시성(visibility)
    : +(public), -(private), #(protected),~(package)
    ※ package: 같은 패키지 안에 있는 클래스만 접근 가능
  - 중첩도(multiplicity): 생성 가능한 객체의 개수 (2..*)
    ex> +origin[50] : Point = (0, 0) {readonly}

- 동작: 가시성 이름(매개변수) : 반환형 {특성}
  - 매개변수 -> direction 매개변수명 : 자료형 = 기본값
    ☞ direction: in, out, inout
    ex> +set(n:Name, s:String) : boolean {guarded}
  - Signature = 동작의 이름 + 매개변수 + 반환형
  - Method = 동작(operation)을 구현한 것

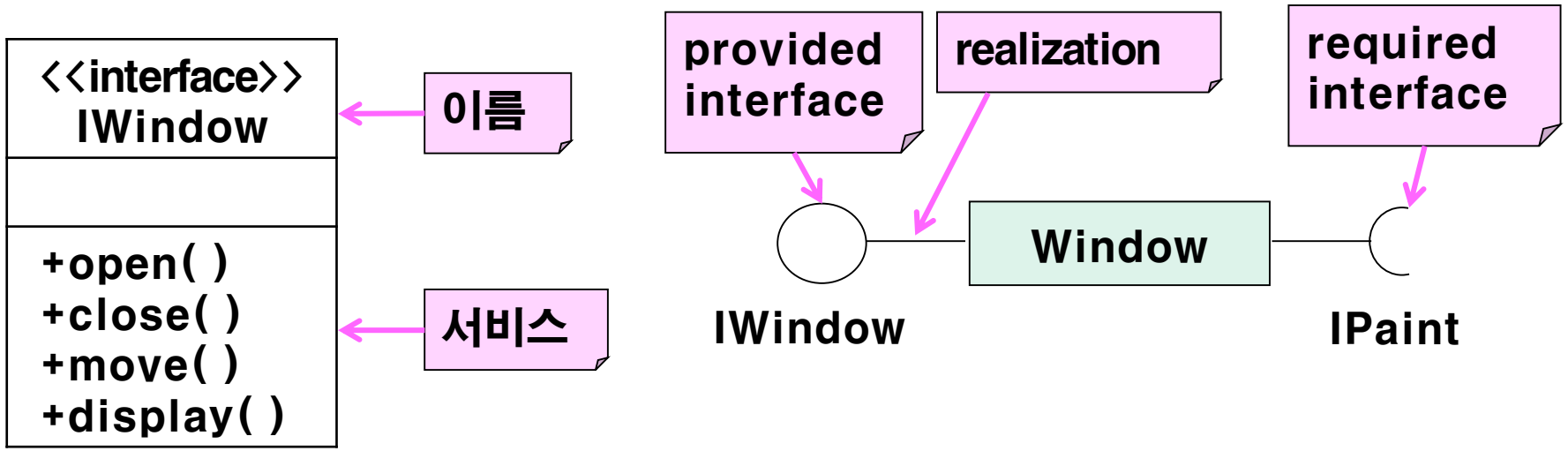
Class(3)

- Owner Scope
  - 정의: 객체 별로 고유한 값을 가질 수 있는가?
    - Instance: 객체 별로 자신만의 값을 가짐
    - Classifier: 모든 객체가 같은 값을 가짐
  - Classifier scope를 갖는 속성 또는 동작은 밑줄로 표시
    - static attributes in C++
    - Generating unique IDs

- 책임(responsibility) = 클래스가 담당해야 할 임무
  - 클래스에게 주어진 임무가 명확한가?
  - 클래스 별로 부여된 책임이 공평한가?

Interface

- Interface(인터페이스)란?
  - 클래스 또는 컴포넌트가 제공하는 서비스의 집합
    - 클래스 또는 컴포넌트의 동작 중에서 외부에 공개하는 것의 명세(signatures)를 기술
    - 통상적으로 인터페이스를 구현하는 클래스 또는 컴포넌트와 함께 나타남
  - 표기법: 직사각형 or 원

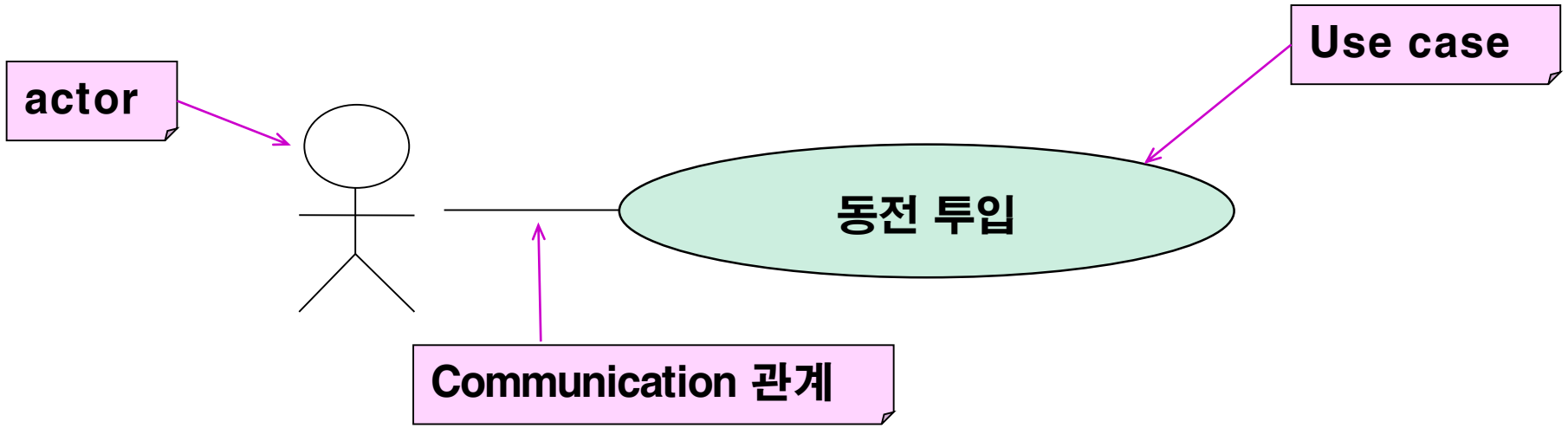


Collaboration

- Collaboration(협동체)란?
  - 주어진 목표를 달성하기 위해서 서로 협력하는 구성 요소들의 공동체를 표현
  - 구조적 차원과 행동적 차원의 양면성을 갖고 있음
  - 주로 아래의 용도로 사용
    - 유스케이스의 실현 방안 기술
    - 구조적으로 중요한 메커니즘의 모델링
    - 시스템을 구성하는 패턴(patterns)의 구현
  - 특정 클래스는 다수의 collaboration에 참여 가능
  - 표기법: 점선 타원

Use case(1)

- Use case(유즈 케이스)란?
  - 시스템이 Actor에게 제공하는 서비스를 표현한 것
    - ① 시스템이 수행하는 기능
    - ② 가시적인 결과가 존재
  - 시스템의 기능적 요구를 추출하는데 사용
  - 협동체(collaboration)에 의해 실현
  - 표기법: 실선 타원



Use case(2)

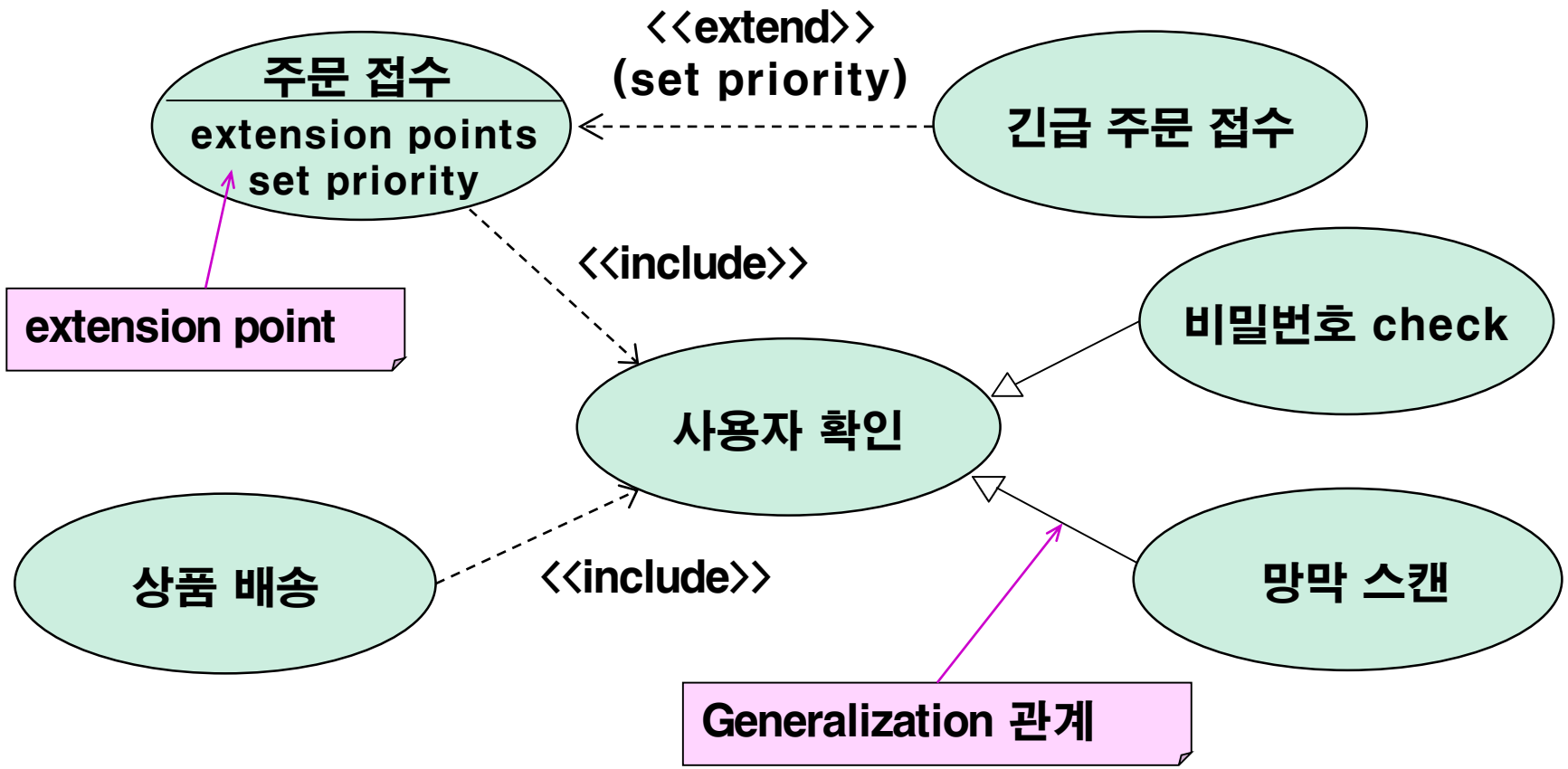
- Actor
  - 유즈 케이스와의 상호작용을 갖는 사람, 하드웨어, 다른 시스템
  - Actor는 개발 대상 시스템의 외부 환경에 속함

- 시나리오(Scenario)
  - 유즈 케이스가 실제로 사용되는 예
  - 여러 개의 시나리오로부터 유즈 케이스를 발견

- 유즈 케이스의 체계화 방법
  - Inclusion: 여러 유즈 케이스에 공동으로 사용되는 것
  - Extension: 특수 상황에서 선택적으로 사용되는 것
  - Generalization: n개의 세부적인 경우가 존재할 때

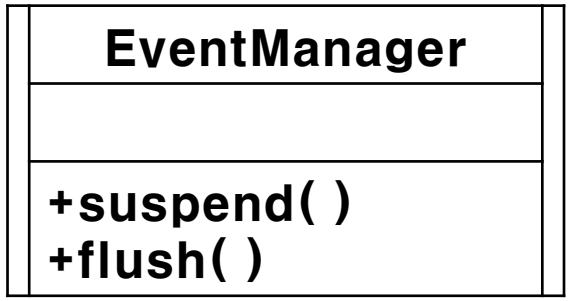
Use case(3)

❑ Generalization, Include, Extend 예시



Active Class(1)

- Active class(능동 클래스)란?
  - 객체가 1개 이상의 프로세스 or 쓰레드(threads)를 가질 수 있는 클래스
    - 새로운 프로세스 또는 쓰레드를 시작시킬 수 있음
    - 병렬처리를 할 수 있는 클래스
  - 표기법: 클래스와 동일. 단, 좌·우측은 이중선 사용



Active Class(2)

- Process & Thread
  - Process
    - 다른 프로세스와 병렬로 실행할 수 있는 제어 흐름의 단위
    - 각각의 프로세스는 별개의 CPU에서 실행 가능
  - Thread
    - 동일한 프로세스 안에서 다른 쓰레드와 병렬로 실행할 수 있는 제어흐름의 단위
    - 쓰레드는 반드시 같은 CPU에서 처리
- Sequential vs. Concurrent
  - sequential -> 모든 순간에 1개의 제어흐름만 존재
  - concurrent -> 어떤 순간에 2개 이상의 제어 흐름이 동시에 존재

Active Class(3)

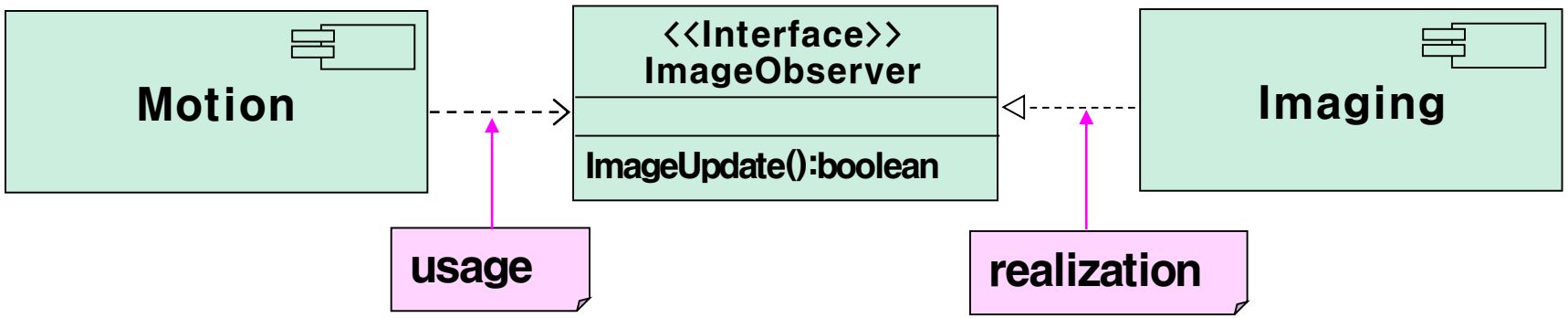
- 객체 사이의 메시지 전달 방법
  - Case1: passive object -> passive object
    - simple operation call -> no problem!
  - Case2: active object -> active object
    - inter-process communication -> no problem!
    - 다음의 2가지 방식이 가능
      - Case2.1: 동기식 호출 -> rendezvous
      - Case2.2: 비동기식 호출 -> mailbox
  - Case3: active object -> passive object
    - 다수의 능동적 객체가 동시에 특정 속성을 접근하는 경우에는 위험 발생 -> synchronization이 필요
  - Case4: passive object -> active object
    - case2와 동일

Active Class(4)

- Synchronization
  - 전통적인 상호 배제(mutual exclusion) 문제
  - Critical object에 대한 동시 접근을 어떤 방법으로 직렬화 (serialization) 시킬 것인가?
  - UML이 제공하는 3가지 접근 방법
    - Sequential: 특정 순간에 객체에 접근할 수 있는 process or thread를 1개가 되도록 외부에서 조정
    - Guarded: 객체의 guarded operation에 의해 처리
      -> deadlock의 위험성은 존재
    - Concurrent: 다수의 process or thread가 객체에 동시에 접근할 수 있도록 허용.
      읽기 전용 또는 disjoint한 속성인 경우에 가능

Component(1)

- Component(컴포넌트)란?
  - 지정된 인터페이스를 충족시키며, 교체가 가능한 모듈
    - ex> COM+ 컴포넌트, CORBA, Java Beans, …
    - 컴포넌트 내부에서 인터페이스를 어떻게 실현하고 있는가는 알 수 없으며, 포트를 통하여 메시지를 주고 받음
    - 포트(port): 컴포넌트가 외부와 소통하는 창구
    - 부품(part): 컴포넌트를 구현할 때 사용되는 구성품
  - 표기법: 우측 상단에 컴포넌트를 의미하는 icon



Component(2)

- 좋은 컴포넌트란?
  - 잘 정의된 인터페이스를 제공
  - 동일한 기능을 갖는 새로운 컴포넌트로 쉽게 교체 가능

- 컴포넌트와 인터페이스
  - 인터페이스: 클래스/컴포넌트가 제공하는 서비스 모음
  - COM+, CORBA, Enterprise Java Beans에서는 인터페이스를 기반으로 컴포넌트를 결합하여 사용
  - 컴포넌트와 인터페이스 사이의 관계는 문서화의 대상
    - Export Interface: 컴포넌트가 실현하는 인터페이스
    - Import Interface: 컴포넌트가 사용하는 인터페이스

Component(3)

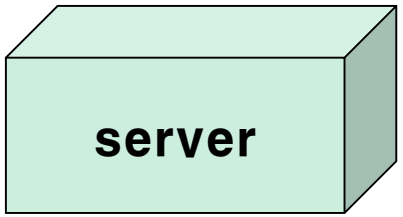
- 컴포넌트의 종류
  - Deployment Components : 실행 시스템의 구성요소
    - Executables(EXEs), Dynamic Web pages -> 산출물
    - Dynamic Libraries(DLLs), COM+, CORBA, EJB
    - Database Tables
  - Work Product Components: 개발과정에서 만들어진 산출물
    - source code files, data files
  - Execution Components
    : 시스템을 실행시킨 결과로 만들어진 컴포넌트
    - COM+ objects

Artifact

- Artifact(산출물)이란?
  - 물리적으로 존재하며, 교체 가능한 시스템의 구성 요소
  - 예시: 원시 파일, 실행 파일, 스크립트, 웹 페이지, …
  - 표기법: 직사각형에 <<artifact>> 표시

Node

- Node(노드)란?
  - 메인 메모리와 CPU를 갖춘 컴퓨팅 장비
    - 예시: PC, 서버, 네트워크 장비, 통신 케이블, …
    - 대부분의 컴포넌트는 임의의 노드에 탑재 가능하나, 특정 노드에만 탑재할 수 있는 컴포넌트도 있음
  - 표기법: 직육면체



Basic Structural Things의 변종

- 클래스 계보
  - Actors, Signals, Utilities, …
- 능동 클래스 계보
  - Processes, Threads
- 컴포넌트 계보
  - Applications, Documents, Files, Libraries, Pages, Tables, …

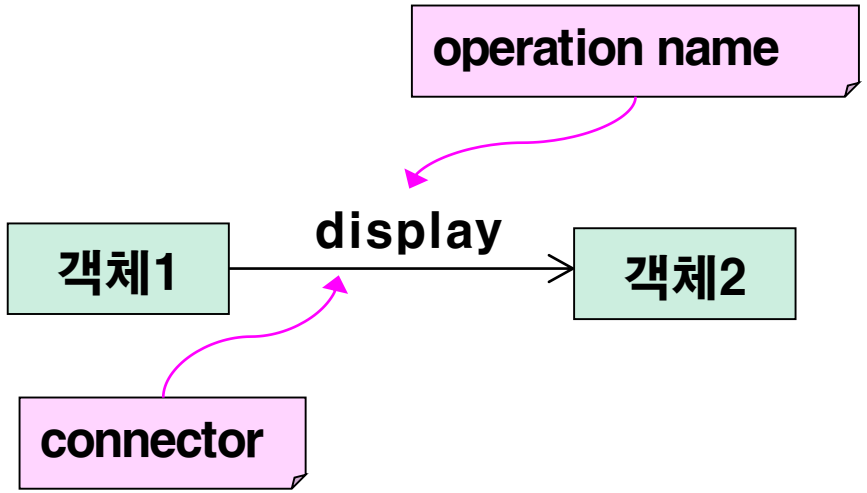
Interaction(1)

- Interaction(상호작용)이란?
  - 다수의 객체가 메시지 교환 방식을 통하여 어떤 목적을 달성해내는 과정을 기술한 것
    - 참여 객체: 일반적으로 다수의 객체이나, 개별 객체의 동작을 기술하는 것도 가능
    - 구성 요소: 메시지, actions, connectors
  - Link에 의해 연결된 객체 사이에 존재 가능
    - Link: association 관계, 객체 사이의 semantic connection
  - 어떤 용도로 사용하는가?
    - Collaboration의 동적인 측면을 표현
    - Operation 내부의 알고리즘을 표현
    - Class, component, node, use case의 개괄적인 동작을 표현

Interaction(2)

❑ 메시지(Message)란?

- 객체 사이에 주고 받는 정보를 기술
- 메시지의 종류
  - ① 함수 호출/반환
  - ② Send Signal
  - ③ 객체 생성
  - ④ 객체 소멸
  - ※ parameter 사용 가능!!
- 메시지 표기법 =>

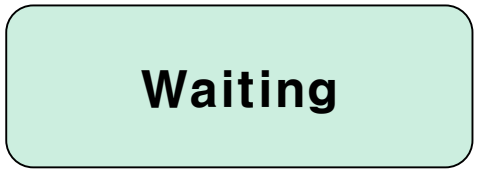


Interaction(3)

- Message Sequencing
  - 객체 사이에 주고 받는 메시지는 순서를 가짐
  - 메시지의 순서는 반드시 시작이 있으며, process 또는 thread가 존재하는 동안 지속
  - 메시지의 순서는 sequence number를 사용하여 표현
    - ex> 1: display(), 2: getNumber(), 3.1.2:…
  - 복잡한 형태의 순서는 반복, 분기, Guarded 메시지 등을 포함할 수 있음
  - 시간상 제약은 Timing mark를 사용하여 기술
    - 사건 발생 제약: {7am < a < 7pm}, {e-d <5sec.}
    - 경과시간 제약: {<1sec.}, {<10sec.}

State machine(1)

- State machine이란?
  - 객체가 내·외부에서 발생한 event에 대하여 반응하는 과정에서 거쳐가는 상태 변화와 이에 수반하여 수행하는 동작을 기술한 것
    - 대상 객체: 단일 객체 또는 서로 다른 클래스에 속하는 2개 이상의 객체
    - 구성 요소: 상태(states), 상태전이(transitions), 사건(events), 동작(activities)
  - 표기법: 둥근 직사각형으로 표시하고, 하위 상태를 가질 수 있음



State machine(2)

- Interaction vs. State Machine
  - 공통점: 잘 작성된 것은 그 자체가 훌륭한 알고리즘
  - 차이점
    - interaction: 다수의 객체가 주어진 목적을 달성하기 위해 상호 협력하는 과정을 기술
    - state machine: 개별 객체의 행위를 기술

- State machine을 표현하는 2가지 방법
  - 제어흐름을 강조 -> Activity diagram
  - 객체의 잠재적 상태와 상태전이를 강조 -> State diagram

State machine(3)

- 어떤 용도로 사용하는가?
  - 객체 내외부의 비동기적 자극에 대해서 객체가 어떻게 반응하는가를 표현
  - 객체의 현재 행동이 지금까지 거쳐 온 과거에 따라 다르게 이루어져야 하는 경우, 이러한 객체의 행위를 기술
  - 시스템 외부의 actor가 보내는 signal에 대해 시스템이 어떻게 반응하는가를 개괄적으로 기술
  - use case가 작동하는 모습을 표현

States

- State(상태)란?
  - 정의: 객체가 특정 조건을 만족시키고 있는 상태 또는 객체가 처한 상황
    - 초기 상태: state machine이 작동을 시작하는 상태
    - 종결 상태: state machine이 작동을 완결한 상태
  - 구성요소
    - Name: 1개의 상태는 이름이 없어도 무방함
    - Entry/exit effects: 특정 상태에 들어갈 때/나올 때 수행하는 작업
    - Internal transitions(내부 전이): 상태 변경이 없는 전이
    - Substates(하위상태): 특정 상태의 내부에서 세분화된 상태
    - Deferred event: 처리의 연기를 위해 큐에 저장해 놓은 사건
      - 지연된 사건의 목록은 "defer"를 사용하여 작성 가능

State Transition(1)

- 상태 전이란?
  - 정의: 객체가 다음의 2가지 조건을 갖추어,
    현재 상태에서 다음 상태로 이동하는 것
    ①특정 사건이 발생 ②명시된 조건(guard condition)을 충족
    - 위의 2가지 조건을 갖추었을 때, 전이가 일어남(fire)
    - 전이가 일어나면 객체는 목표 상태에 도달
  - 구성 요소
    - Source state(현재 상태)
    - Event trigger(전이 유발 사건)
    - Guard condition (별도 명시 조건)
    - Effect(영향): 자신 또는 다른 객체에게 영향을 미치는 작업
      - inline computation, operation call,
        creation or destruction of other objects,
        sending signal
    - Target state(목표 상태)

State Transition(2)

- 특이한 상태 전이
  - triggerless transition: 현재 상태에서 필요한 작업을 완료하면, 사건 발생 없이 자동으로 목표 상태로 이동
  - self transition: 현재 상태와 목표 상태가 동일한 전이

- Guard condition
  - 상태전이를 유발시키는 사건이 발생한 후에 검사 시행
  - 조건을 만족하지 못하면 상태전이가 발생하지 않음

- Effect
  - Entry action: "entry/"로 표기
  - Exit action: "exit/"로 표기
  - Activity: "do/"로 표기

State Transition(3)

- SubStates & Internal transitions
  - Internal transition(내부 전이)
    : 어떤 상태 내부의 하위 상태 사이에 일어나는 상태 전이
  - Substates(하위 상태)
    - 정의: 다른 상태의 내부에 포함되어 있는 상태
    - 복잡한 행동을 표현할 때 유용하게 사용 가능
    - 임의의 레벨까지 만들 수 있음
  - Composite state: 순차적 또는 병렬적 하위 상태를 포함하는 상태
  - History States
    - composite state를 떠나기 전에 마지막으로 머물렀던 하위 상태를 기록할 수 있는 기능 장착
    - 표기법: 작은 원 내부에 기호 'H' 포함

Event(1)

- 기본 개념
  - 정의: 특정 시간 및 장소에서 발생한 의미 있는 사건
    - Things that happen
    - 상태 전이를 유발시킬 수 있는 자극의 발생
    - ex> Signal: 객체 사이에 주고 받는 비동기적 신호
  - 사건의 분류
    - 외부 사건 : 시스템과 actor 사이에서 발생한 사건
      - pushing of a button
      - interrupt from a collision sensor
    - 내부 사건: 시스템 내부의 객체 사이에서 발생한 사건
      - overflow exception

Event(2)

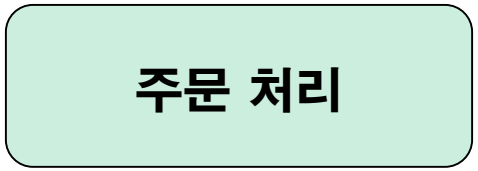
- UML에서 사용하는 사건
  - Signal event
    - 송신 객체가 보내고 수신 객체가 받는 신호
    - 비동기적으로 주고 받으며, 이름을 갖는다.
  - Call event
    - 오퍼레이션을 호출하는 것
    - 이벤트의 대부분을 차지
  - Time event
    - 일정 시간이 경과되면 발생하는 사건
  - Change event
    - 상태 변화 또는 특정 조건을 만족했을 때 발생하는 사건

Event(3)

- Signal Event vs. Call Event
  - 2개 이상의 객체가 참여
  - 모든 객체는 모든 객체에게 signal을 보내거나,
    다른 객체의 오퍼레이션을 호출할 수 있음
  - signal event는 비동기적, call event는 동기적으로 작동
    -> 오퍼레이션을 호출하면 수신 객체의 응답을 기다려야 함!
  - signal event는 multicasting 및 broadcasting 가능
  - signal 및 call event는 수신 객체에 기술
    - Call event -> 오퍼레이션으로 기술
    - Named signal -> 별도로 만든 칸에 기술

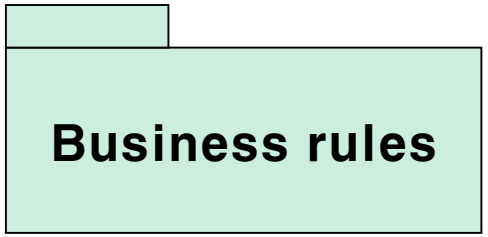
Activity

- Activity(동작)이란?
  - 어떤 기능을 실현하기 위한 작업의 기본 단위
  - 원자적 행위인 action으로 구성
    - action: activity의 각 단계에서 이루어지는 원자적 작업
  - 객체가 수행하는 작업의 제어구조에 초점
  - 표기법: 둥근 직사각형



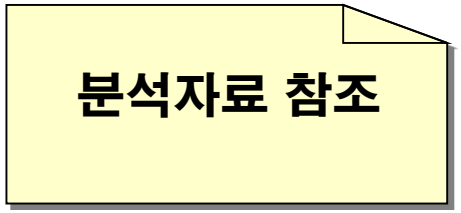
Package

- Package(패키지)란?
  - 구성 요소를 체계적으로 분류하여 그룹으로 묶어 주는 것
    - 패키지는 주로 연관도가 높은 구조 사물과 동작 사물을 포함
    - 패키지는 하위의 패키지를 포함할 수 있음
    - 관념적인 것으로 개발 단계에서 사용되며, 실행과는 무관
  - 패키지의 변종: Frameworks, Models, Subsystems
  - 표기법: 탭(tab) 폴더를 사용

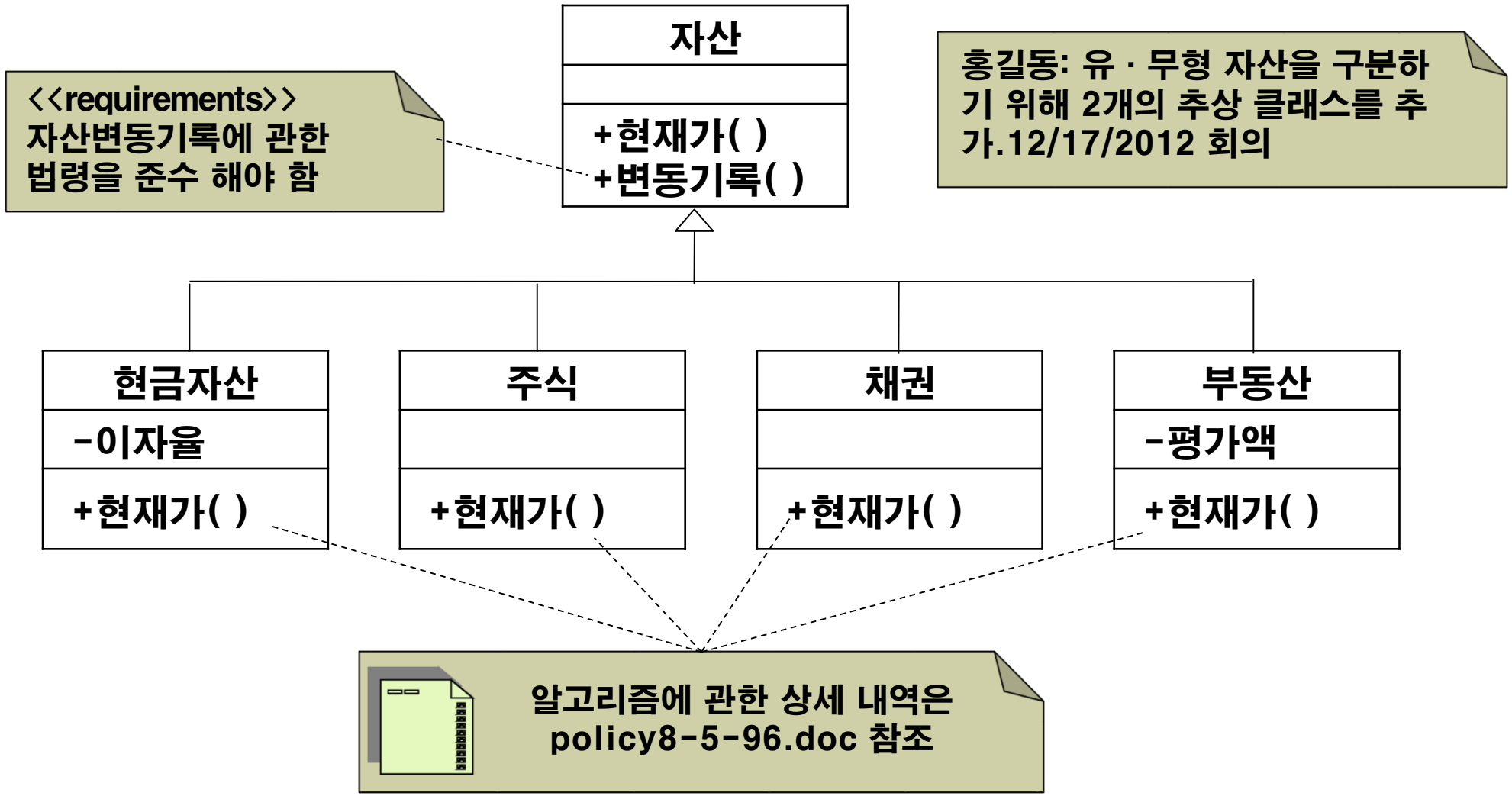


Note

- Note(노트)란?
  - UML 구성 요소에 대한 제약사항(constraints) 또는 간단한 설명(comments)을 붙이는 도구
    - 의무사항(requirements), 관찰 기록(observations), 검토내용(reviews), 세부 설명 등을 기술하는 용도로 사용
    - 분량이 많은 경우에는 상세 내역은 별도의 문서에 기록하고 주석에서는 해당 문서에 대한 링크(link)만 기록
  - 주석의 변종: 요구사항(Requirements)
  - 표기법: dog-eared rectangle



주석 사용 예시


자산
+현재가( )
+변동기록( )
<<requirements>>
자산변동기록에 관한
법령을 준수 해야 함
홍길동: 유 · 무형 자산을 구분하기 위해 2개의 추상 클래스를 추가.12/17/2012 회의
현금자산
-이자율
+현재가( )
주식
+현재가( )
채권
+현재가( )
부동산
-평가액
+현재가( )
알고리즘에 관한 상세 내역은
policy8-5-96.doc 참조

Relationships(1)

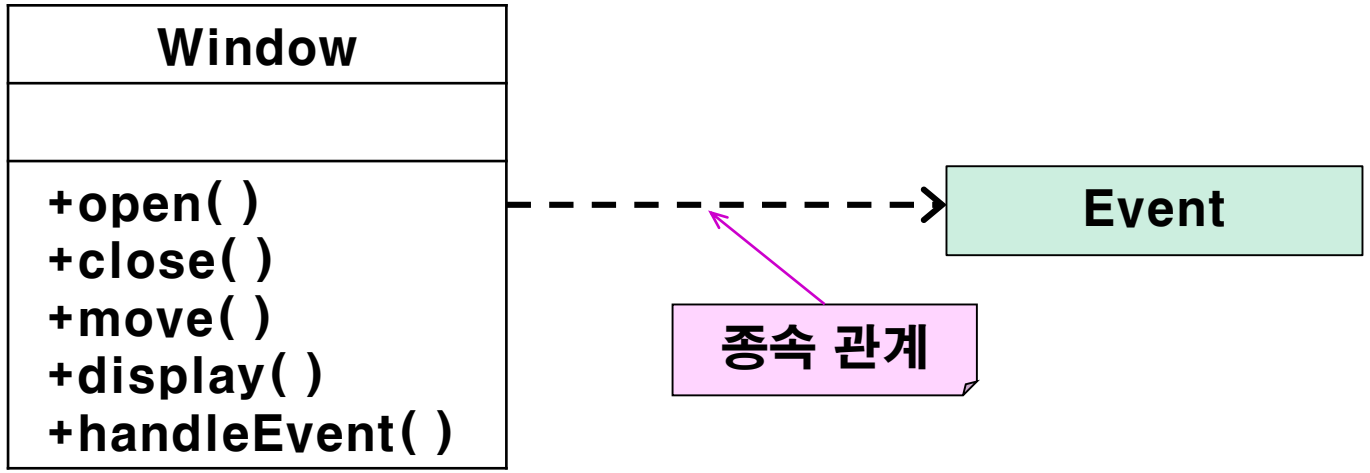
- Relationship(관계)란?
  - 정의: Things 사이의 존재하는 관계
  - 관계를 포착해야 하는 이유
    - 시스템 구성요소는 임무 수행을 위해 다른 구성요소와 다각도로 협력
      -> 단독으로 존재하는 구성요소는 거의 없음
    - 구성 요소 사이의 관계를 포착하여 표현함으로써
      모델의 완성도를 높일 수 있음
  - 기본적인 관계는 4종류가 있으며, 다양한 변종도 존재
    - Dependency(종속)
    - Association(연관)
    - Generalization(일반화, 상속)
    - Realization(실현)

Relationships(2)

- 관계의 발견
  - 쉽지 않은 작업 or 매우 어려운 작업
  - 많은 경험을 통하여 노하우를 쌓을 필요가 있음
  - 발견한 관계의 량에 따라 모델의 난이도가 달라짐
    - 과다 -> 이해하기 어려운 모델
    - 과소 -> 불완전한 모델

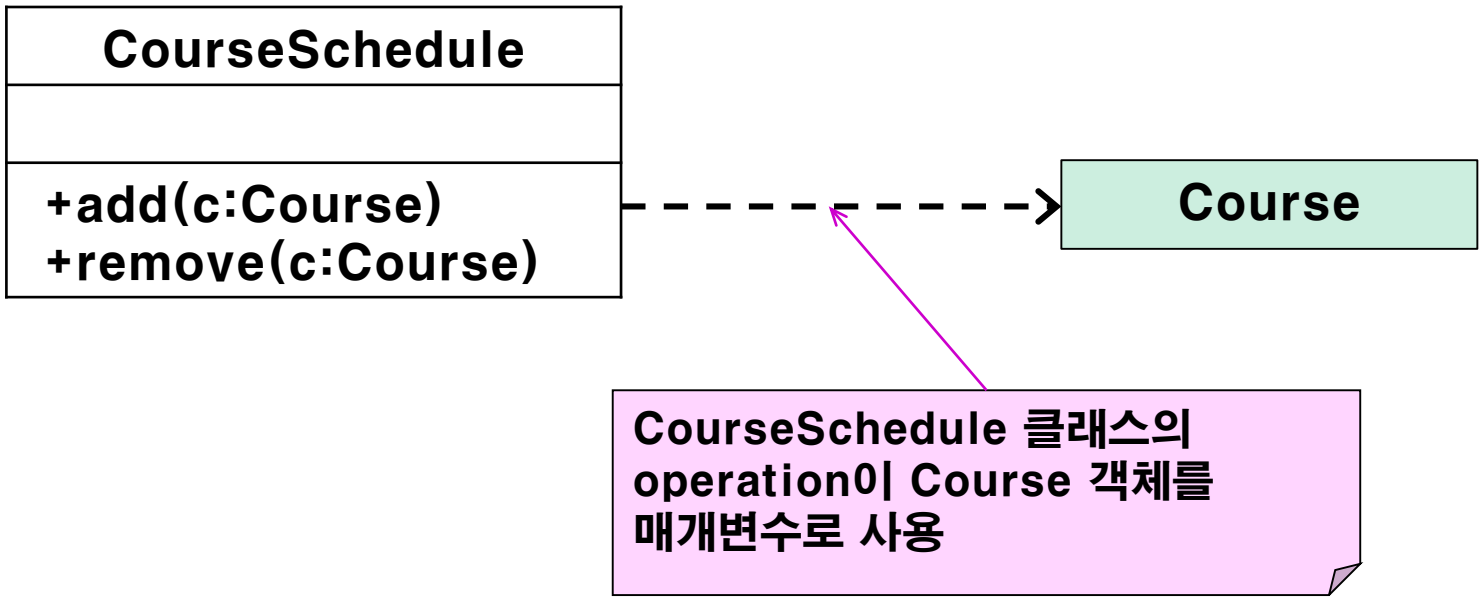
Dependency(1)

- 종속 관계란?
  - 구성 요소 A가 구성요소 B를 사용하는(using) 관계
    - B가 변경되면 A가 영향을 받는 경우, A는 B에 종속적
    - ex> 클래스 A가 클래스 B의 operation을 사용하는 경우
  - 변종: Refinement, Trace, Include, Extend, Bind
  - 표기법: 점선 화살표를 사용하며, 라벨 부착 가능



Dependency(2)

❑ 종속 관계 예시

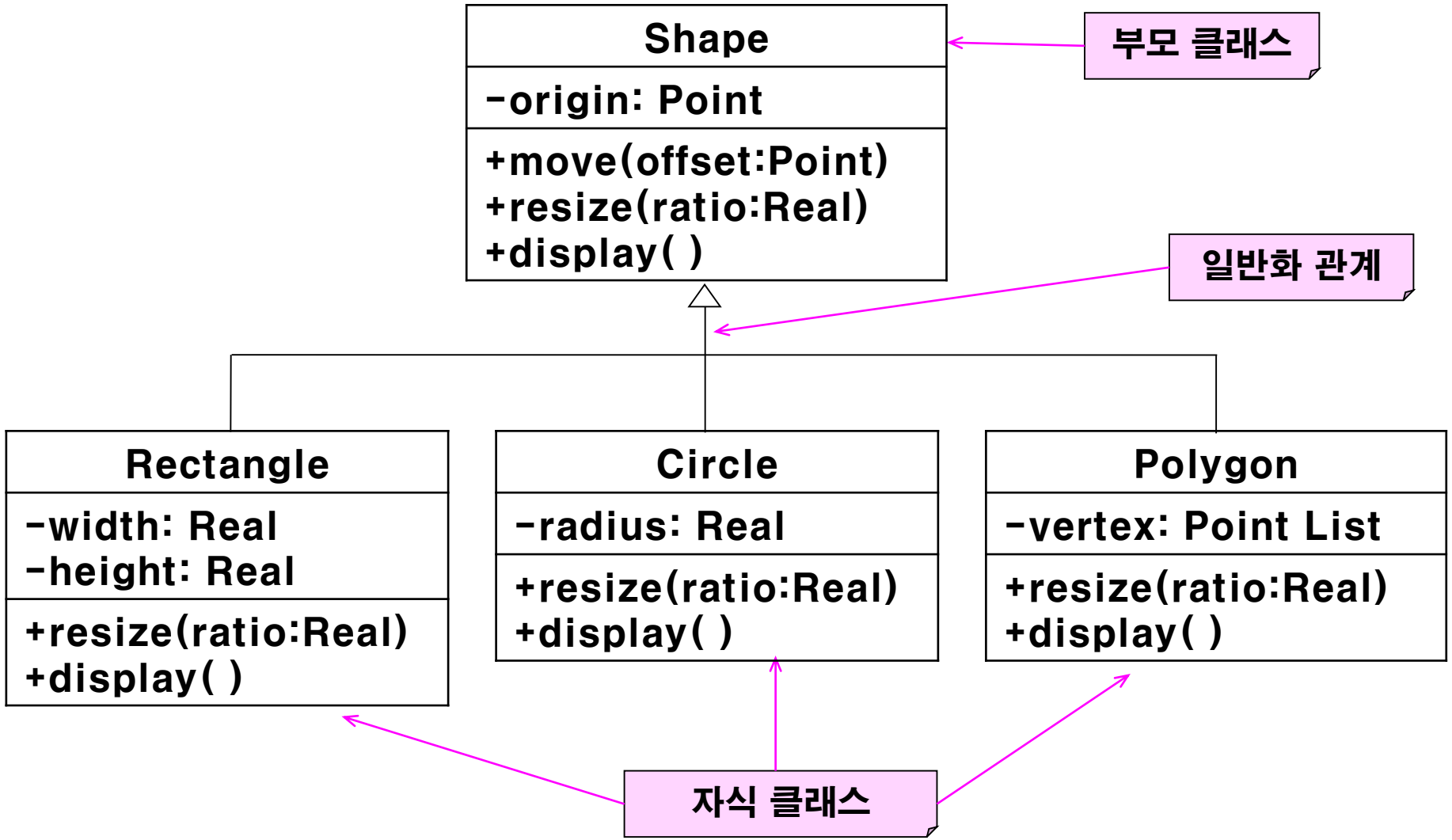


Generalization(1)

- 일반화 관계란?
  - 일반적인 구성요소(superclass/parent)와 이것의 종류인 하위 구성요소(subclass/child) 사이의 관계
    - "is-a-kind-of", 일반화/특수화 관계라고도 함
    - 클래스 또는 actor 사이의 상속 관계
  - 클래스는 n개의 부모를 가질 수 있음
  - 자식은 부모의 속성과 동작을 상속 받고, 자신만의 속성과 동작을 추가할 수 있음
  - 자식 객체는 부모 객체가 사용된 곳에 사용 가능
  - 자식은 부모의 동작을 무시(override)할 수 있으나, signature는 변경할 수 없음
  - 표기법: 빈 삼각형의 머리를 가진 실선 화살표

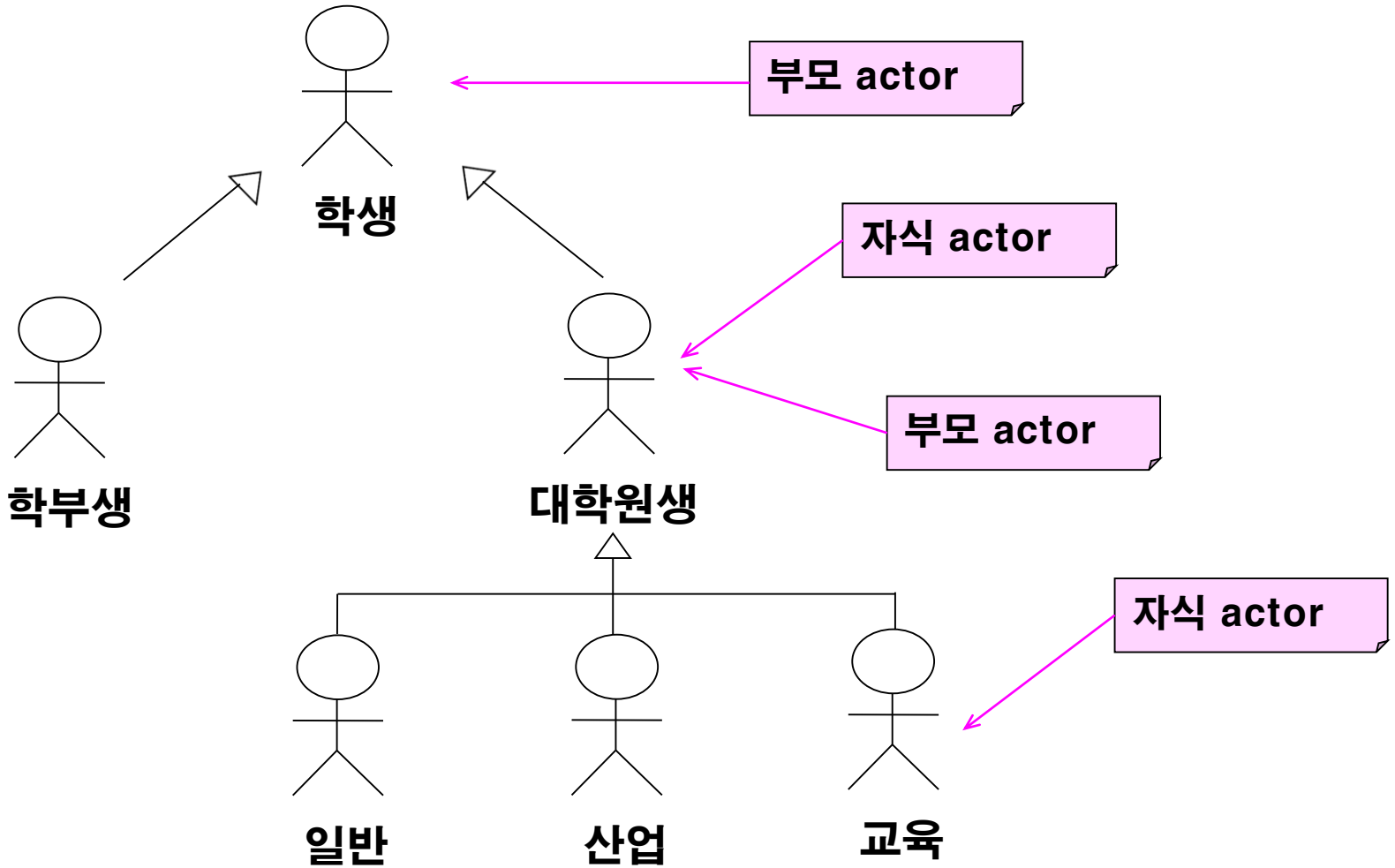
Generalization(2)

❑ 일반화 관계 예시(클래스)



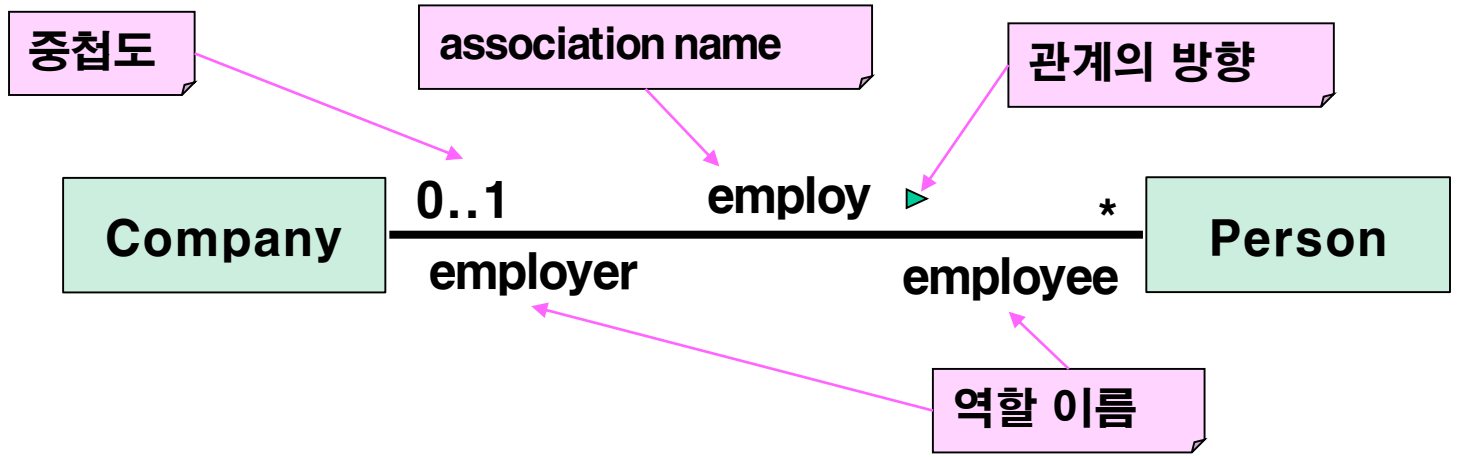
Generalization(3)

❑ 일반화 관계 예시(Actor)



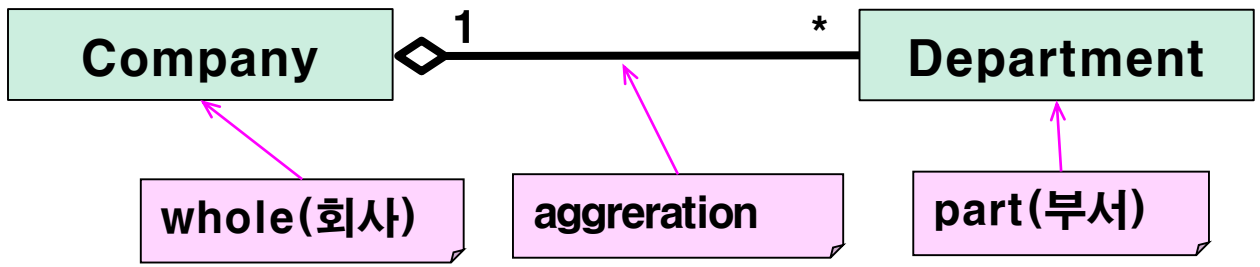
Association(1)

- Association(연관) 관계란?
  - 관련된 객체 사이의 연결 구조를 표현한 관계
    - 예시: teach, sit on, …
    - 일반적으로 이항 관계이나, n항 관계도 가능
  - 연관 관계를 갖는 클래스의 객체간에는 순회가 가능
  - 표기법: 실선으로 표시하며,
    필요한 때에는 추가로 장식(adornment)을 첨부할 수 있음



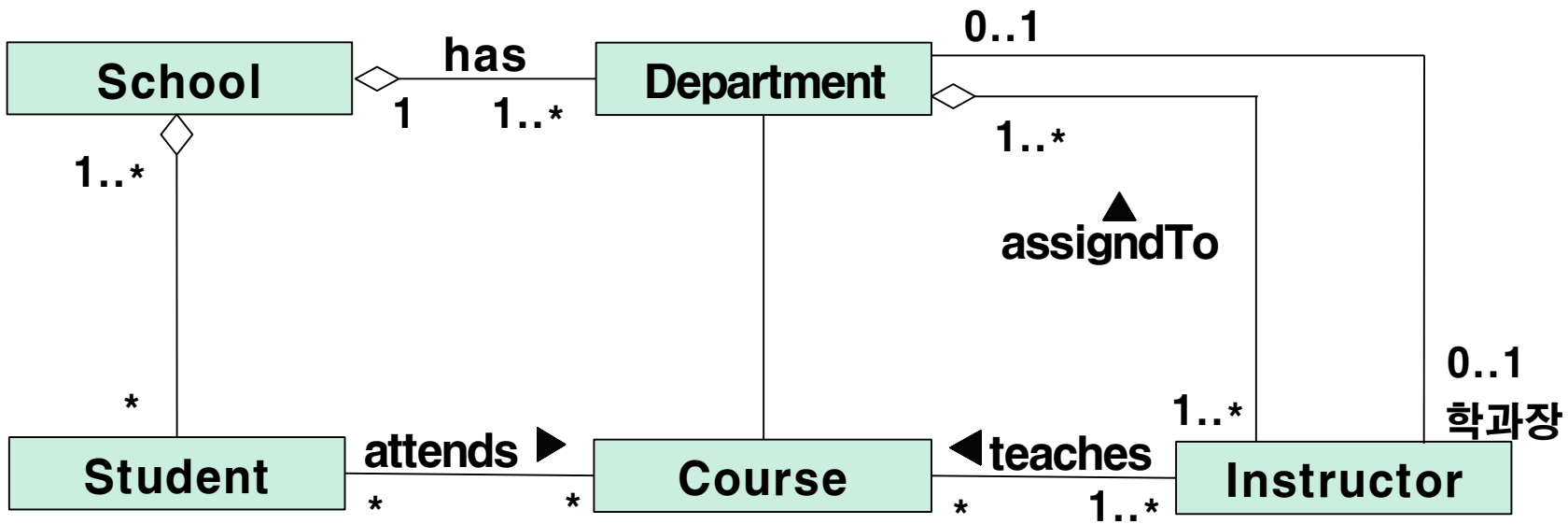
Association(2)

- 연관 관계에 첨부할 수 있는 장식
  - 연관 관계의 이름(name)
  - 관계의 방향이 애매한 때에는 방향(▶) 표시 가능
  - 참여 객체의 역할(role) 명시 가능
  - 중첩도(Multiplicity): 관계에 참여하는 객체의 수
    - 1, 0..1, 0..*, 1..*, 3, 3..7, ...
- Aggregation(집합)
  - 구성품 관계를 표현하는 연관 관계의 특수 형태
  - "has-a"관계라고도 부름



Association(3)

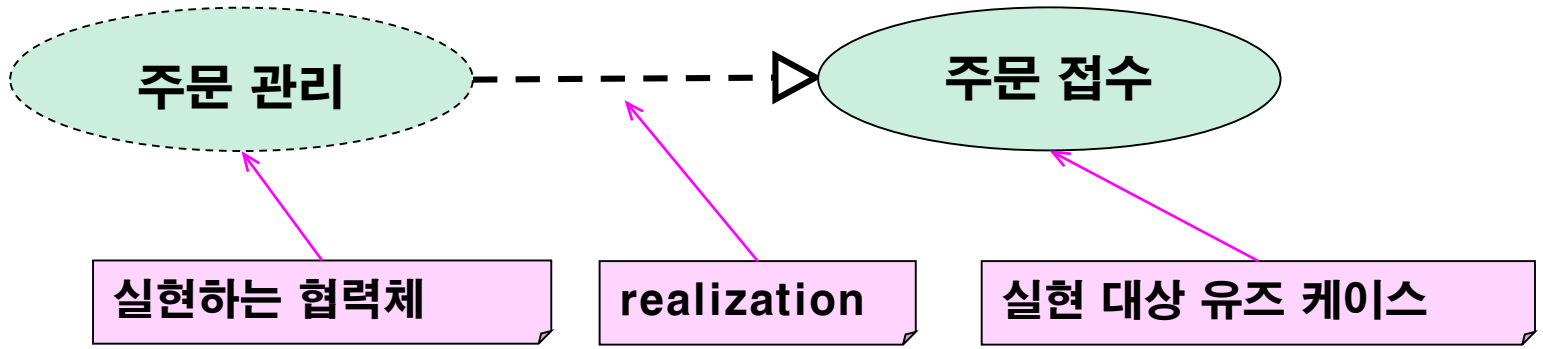
- Composition(합성)
  - 전체 객체와 부품 객체 사이에 강한 소유 관계
    - 부품 객체가 다른 전체 객체에 공유될 수 없음
    - 부품 객체는 전체 객체와 생존 기간이 일치
  - aggregation -> 약한 소유 관계
  - 연관 관계 예시



Realization

❑ 실현 관계란?

- 계약을 기술한 객체와 이를 실천하는 객체와의 관계
  - 클래스 vs. 인터페이스
  - 컴포넌트 vs. 인터페이스
  - 협력체(Collaboration) vs. 유즈 케이스
- 표기법: 빈 삼각형의 머리를 가진 점선 화살표



Diagram(다이어그램)

- 다이어그램이란?
  - 대상 시스템의 구성 요소와 이들 사이의 관계를 표현
    - 연결 그래프(connected graph)를 사용
    - vertex=things, arcs= relationship
  - 대상 시스템을 다양한 관점에서 가시화 시키는 도구
    - 다이어그램 = 시스템을 특정 방향에서 투영(projection)한 결과
    - UML은 13 종류의 다이어그램을 제공
  - 특정 구성 요소가 모든 다이어그램에 나타날 수도 있으나, 통상적으로 일부 다이어그램에만 나타남.
  - 이론적으로는 Things와 Relationships의 어떠한 조합도 가능하나, 자주 나타나는 조합은 제한적임

Diagram의 종류

- **Structural Diagrams**
  - Class Diagram
  - Object Diagram
  - Component Diagram
  - Composite Structure Diagram
  - Deployment Diagram
  - Package Diagram
- **Behavioral Diagrams**
  - Use case Diagram
  - Sequence Diagram
  - Communication Diagram
  - State Diagram
  - Activity Diagram
  - Timing Diagram
  - Interaction Overview Diagram

※ 목록에 열거한 다이어그램은 공통적으로 사용되는 것이며,
UML에서는 이외에도 다양한 다이어그램을 사용함!
※ Artifact Diagram: 구조 다이어그램의 일종으로 취급
※ Interaction Diagram = Sequence + Communication

Rules(1)

- UML 규칙이란?
  - 적법한 UML 모델을 작성하기 위해 지켜야 할 규칙
    - 적법한(well-formed) 모델: 자체적으로 모순이 없고, 다른 모델과 조화를 이루는 모델
  - UML은 아래 항목에 관한 규칙을 갖고 있음
    - Name: 사물(Things), 관계, 다이어그램의 명명 규칙
    - Scope: 이름을 사용할 수 있는 범위에 관한 규칙
    - Visibility: 특정 구성 요소를 외부에서 볼 수 있는가?
    - Integrity: 특정 구성 요소를 다른 것과 적절하고 모순 없이 결합하려면 어떻게 해야 하는가?
    - Execution: 동적 모델을 실행시켰을 때 어떤 결과가 예상되는가?

Rules(2)

- 불완전한 모델의 작성
  - 불완전한 모델(less-than-well-formed model)
    - 상세한 내역을 생략하거나,
    - 모든 규칙을 충족시키지 않거나(incomplete),
    - 다른 부분과 상충될 수 있는(inconsistent) 모델
  - 시스템 개발 과정에서는 어떤 목적을 위해서 불완전한 모델을 작성하여 사용하는 경우도 있음
    - 개발 과정에서 모델을 완전한 상태로 유지하는 것은 매우 어렵고, 상당한 시간을 소요
    - 완전한 모델을 작성하는데 필요한 정보가 불충분
  - UML은 개발자에게 가능한 한 완전한 모델을 작성하도록 권유하나, 강요하지는 않는다!

Specification

❑ Specification(설명)이란?

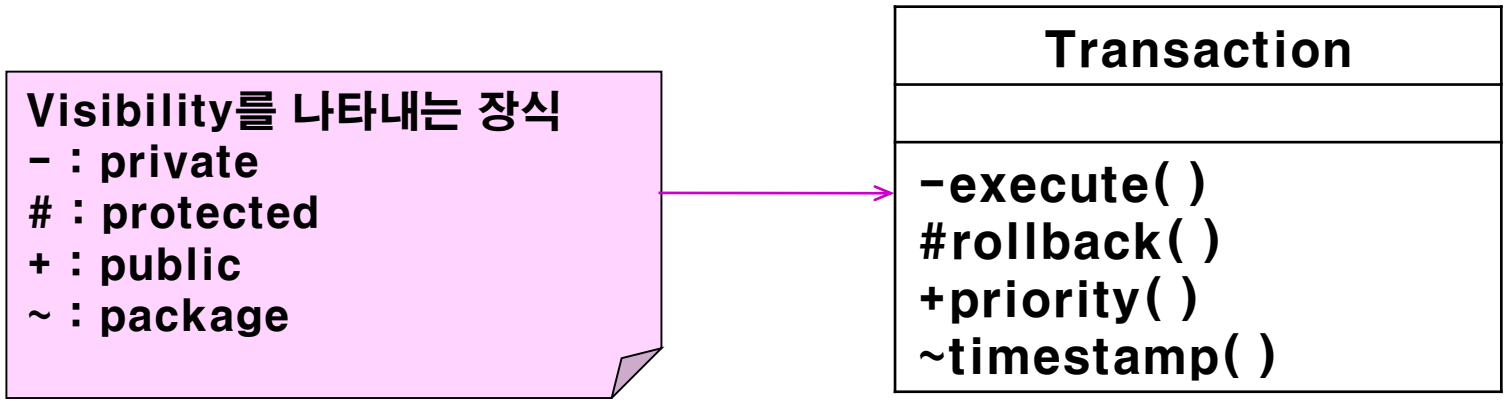
- 그래픽 기호로 표현된 UML 구성요소에 대한 텍스트 형태의 설명
  - 문법 또는 의미에 대한 상세한 설명
- 예시) 클래스 아이콘 <-> 아이콘으로 표현된 클래스에 대하여 속성, 동작, 임무 등에 대한 상세 내역

❑ 사용 용도

- UML 모델에 대한 전반적인 배경 설명
- 시스템의 구성 요소에 관한 세부 내역을 기술
- 다이어그램의 의미를 상세하게 기술

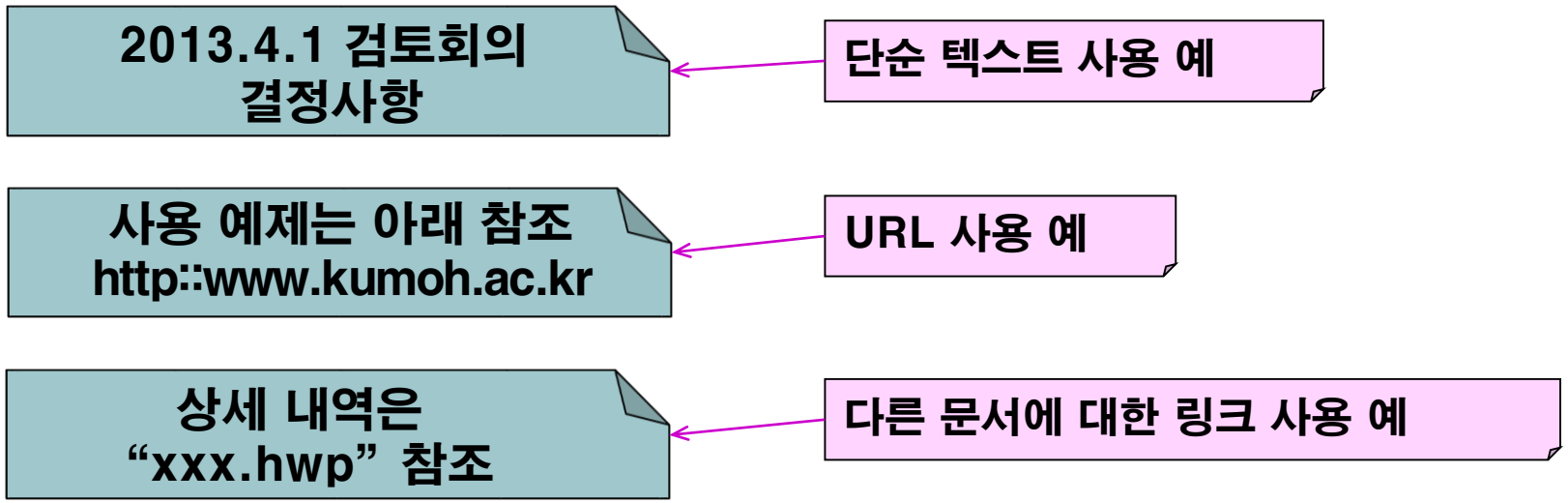
Adornment(1)

- Adornment(장식)이란?
  - UML의 구성 요소에 대한 기본 기호 외에 부가적인 의미를 표현하는데 사용되는 텍스트 또는 기호
  - 장식을 붙이는 방법
    - Note 사용
    - Extra Compartment 사용
    - 그래픽 심볼을 추가하는 방식



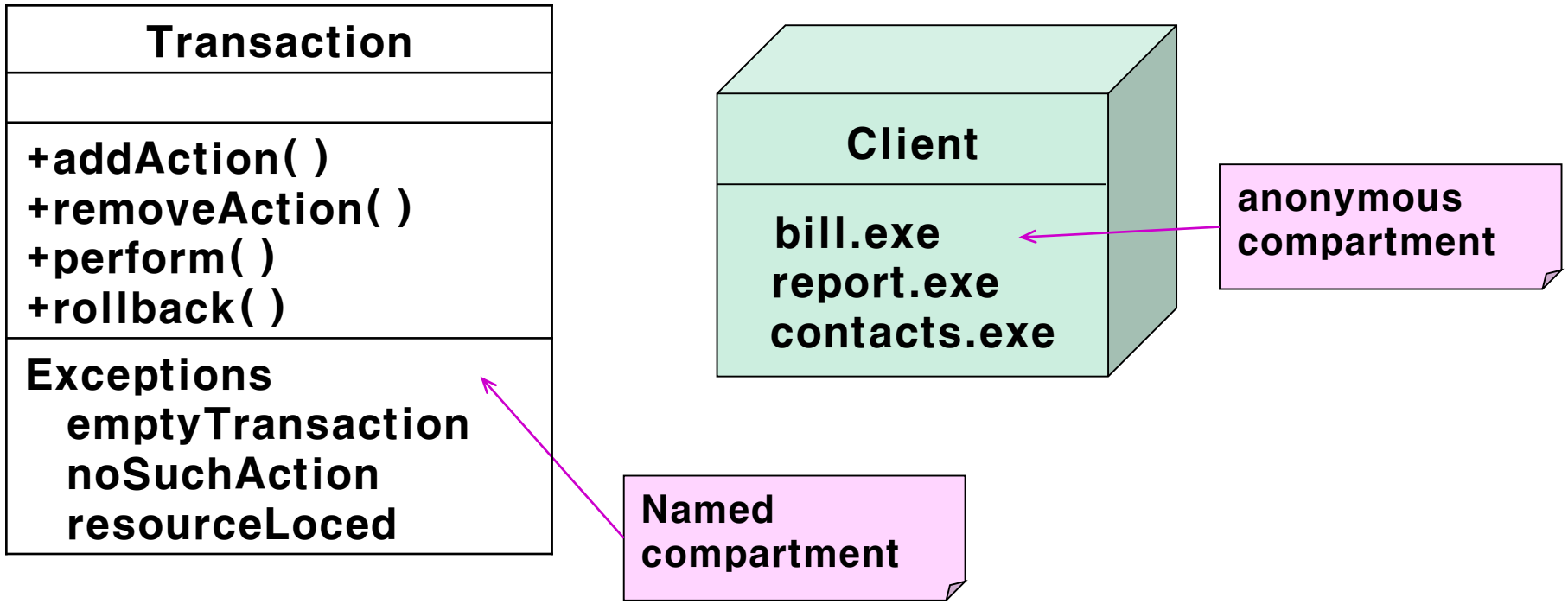
Adornment(2)

- Notes(노트) 사용
  - UML 구성 요소에 제약사항 또는 주석을 붙일 때 사용하는 기호
  - 노트에 포함할 수 있는 내용
    : 단순 텍스트, URL, 다른 문서에 대한 링크



Adornment(3)

- Extra Compartment(별도 칸) 사용
  - 클래스, 컴포넌트, 노드 등은 별도 칸을 사용하여 장식할 수 있음
  - 별도 칸에는 이름을 붙일 수도 있음



Common Division

- Common Division (공통 분할)이란?
  - 현실 세계에 있는 개체는 대응하는 종류로 양분되는 경우가 자주 나타남
    - 추상적인 존재 vs. 구체적 instance
      - 클래스 vs. 객체
      - use case vs. use case instance
      - component vs. component instance
      - node vs. node instance
    - 인터페이스 vs. 인터페이스의 구현
      - 인터페이스 vs. 이를 구현한 클래스 또는 컴포넌트
      - 유즈 케이스 vs. 유즈 케이스를 실현한 collaboration
      - operation vs. method
    - 개체의 종류를 나타내는 type vs. 개체의 역할(role)
      - Person 타입 vs. customer 객체
  - UML은 대응되는 양쪽을 모두 표현할 수 있는 방법을 제공

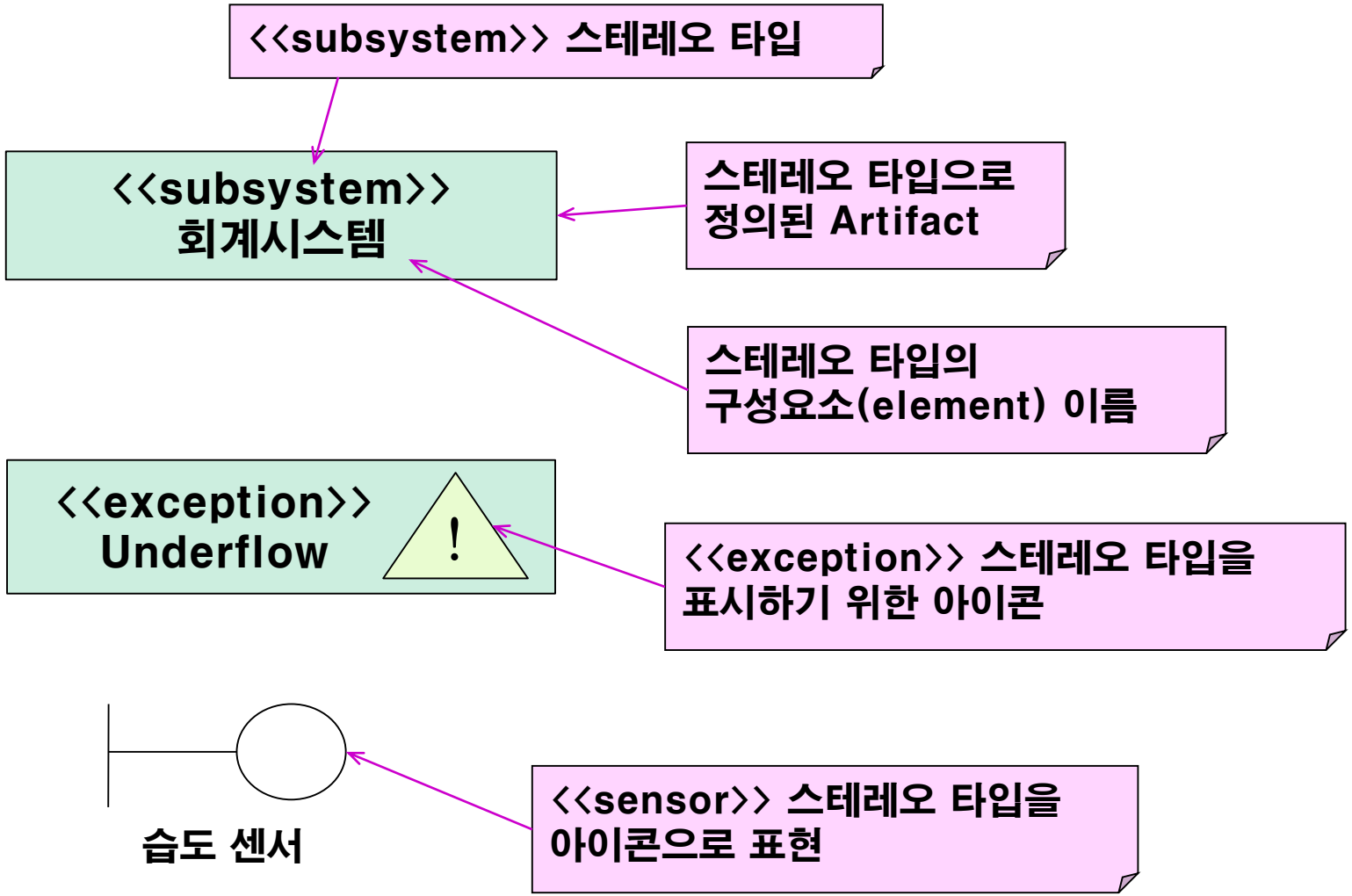
Extensibility Mechanism

- Extensibility Mechanism(확장 메커니즘)이란?
  - UML은 개방된 언어로서 개발자가 수행하는 프로젝트의 특성에 맞게 확장할 수 있는 방법을 제공
  - UML이 제공하는 3개의 확장 메커니즘
    - Stereotypes
    - Tagged values
    - Constraints

Stereotype

- Stereotype이란?
  - UML에서 사용 가능한 구조물을 확장하는 방법
    - 기존의 구조물 (building blocks)로부터 개발대상 시스템에 필요한 새로운 구조물을 만드는 수단
    - 새로운 타입을 만들어 주는 메타 타입(meta type)
    - 각각의 스테레오 타입은 자신만의 tagged values, 제약사항, 기호를 가질 수 있음
  - 표기법: <<name>>
    - <<exception>>, <<subsystem>>
    - 고유한 아이콘을 이름 옆에 표시할 수 있음

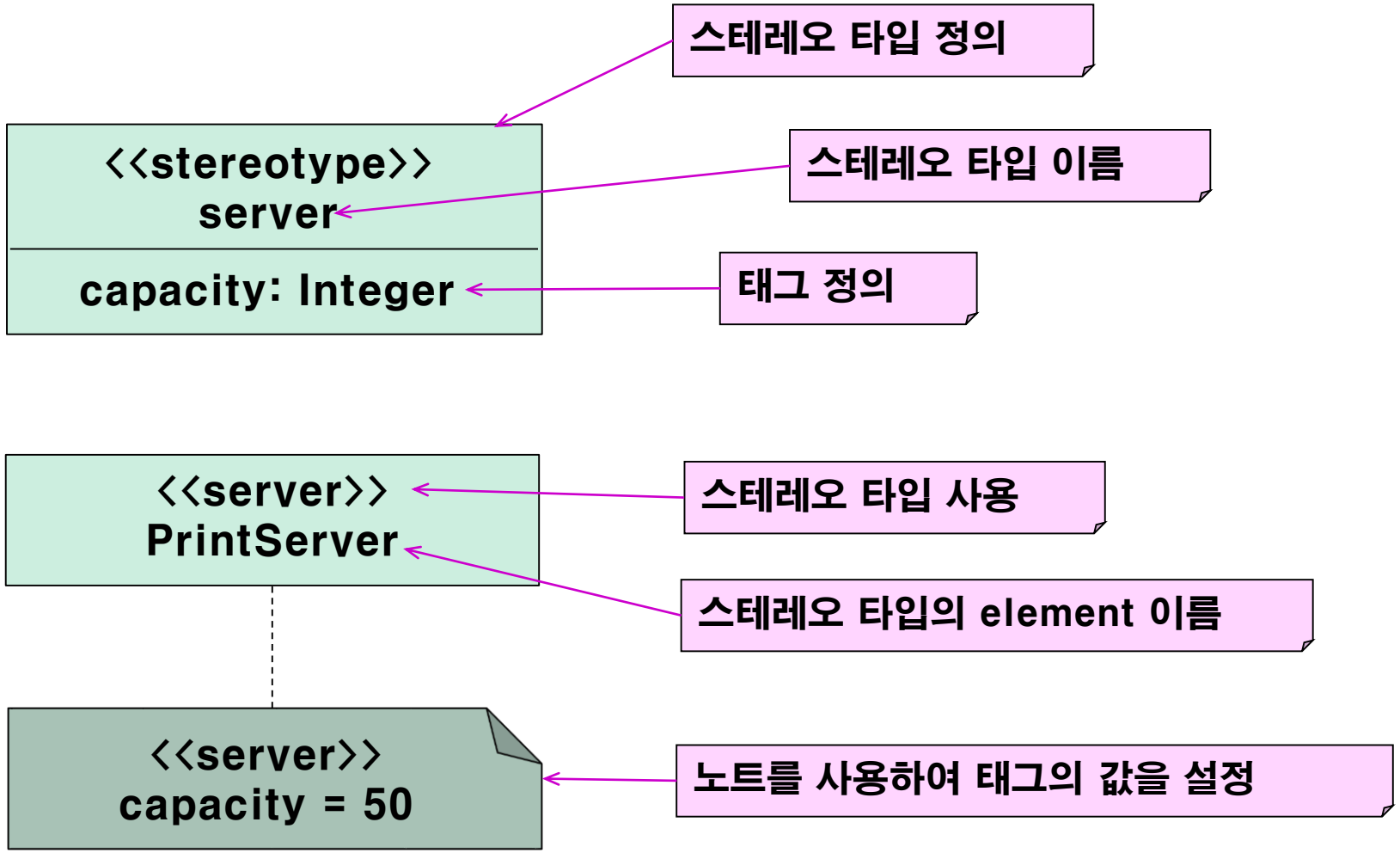
Stereotype 예시



Tagged Value

- Tagged Value(태그 값)이란?
  - UML 구성 요소에 새로운 속성(property)을 추가하는 방법
    - UML 구성 요소는 각각의 고유한 속성을 가짐
      - 클래스의 속성: name, attributes, operations, …
      - 연관 관계의 속성: name, role, direction, …
    - 기본 속성 외에 새로운 속성을 추가할 필요가 있을 때 사용
    - 클래스의 attributes는 클래스가 갖는 속성 중의 하나이나, 태그는 새로운 속성 자체를 정의하는 수단
  - 표기법: name = value
    - version=3.2, author= “Kim”
    - 태그의 값은 노트 안에 기술

Tagged Value 예시



Constraint

- Constraint(제약사항)이란?
  - UML 구성 요소에 새로운 규칙을 추가하거나 기존의 규칙을 확장하는 방법
    - ex> 실행 시에 충족시켜야 할 조건
    - 정확한 기술을 위해서
      UML의 OCL(Object Constraints Language) 사용 가능
  - 표기법
    - 제1방법: 중괄호를 사용하여 기술
      - ex> { > 10M/Sec line }
    - 제2방법: Note 안에 제약사항을 기술

Constraint 예시