UML과 소프트웨어개발

강사명: 손재현 -넥스트리소프트
-jhsohn@nextree.co.kr

과정 개요

❑ 교육 목표 & 특징

- UML2.x의 이해
- 유스케이스 작성
- 객체모델링 이해
- UML2.x의 다양한 다이어그램 이해 및 활용
- 모델링 도구 사용법 습득

강의 요구 기술

- **본 강의는 아래 기술에 대한 이해를 필요로 합니다.**
  - 객체지향 언어(Java) 기초
  - 개발프로세스 이해

교육 일정표

❑ 교육은 매 회 4 시간씩 총 5회에 걸쳐 진행합니다.

1 일차	2 일차	3 일차	4 일차	5 일차
- UML 개요 UML 소개 UML 다이어그램분류	- 유스케이스 유스케이스 개요 유스케이스 다이어그램 유스케이스 명세	- 분석모델 I 개념모델 구조 다이어그램 (클래스, 객체 ...)	- 분석모델II 컴포넌트 식별 인터페이스 식별 행위 다이어그램 (액티비티, 시퀀스 ...)	- 설계모델 컴포넌트 동적 설계 컴포넌트 내부 설계

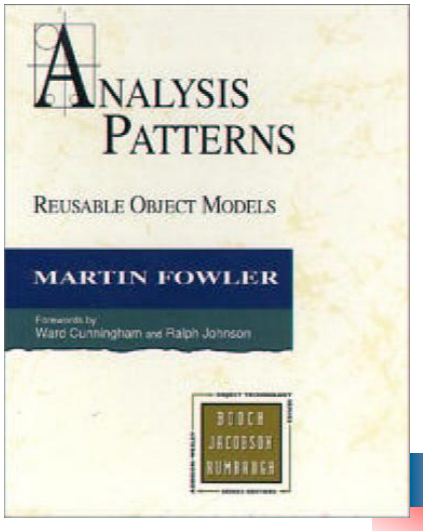
ONE STEP AHEAD

1. 개념모델
2. 구조 다이어그램

1. 개념 모델

ONE STEP AHEAD

- 개요
- 도메인 모델



개요 (1/2)

- **Martin Fowler의 저서 'Analysis Pattern – Reusable Object Model' [Addison-Wesley, 1996] 에서 처음 사용**
- **객체 개발의 중요한 원칙**
  - 소프트웨어의 문제를 반영한 구조의 디자인 수행
  - 모델은 분석과 설계의 마무리된 결과
  - 대부분의 개발자는 분석과 설계의 차이에 대한 인식이 없음
- **Mental Model**
  - 요구사항에 대한 지속적인 문제를 머리 속으로 모델링 하여 회고
  - 이해하고 있는 문제에 대한 Mental Model을 개념모델로 생성

개요 (2/2)

- **도메인 전문가*Domain Expert***
  - 개념모델링 수행 시 도메인 전문가를 포함하는 것이 필수적
  - 전문가의 지식은 좋은 분석 모델의 중심이 됨
- **분석 기술은 소프트웨어 기술에 독립적**
- **개념 모델은 소프트웨어 구현보다 소프트웨어 인터페이스에 관계가 있음**

모델링 원칙

- 모델은 옳고 그름이 아니고 보다 더 유용한가 아니면 덜 유용한가 이다.

- 개념모델은 구현(classes)이 아니고 인터페이스(types)와 연결 된다.

도메인 모델

❑ 정의

- 객체지향 분석 단계에서 도메인을 중요한 개념이나 객체로 분해
  - 도메인에서 개념적인*conceptual* 클래스나 실제 상황*real-situation* 객체의 시각적 표현
  - 개념*conceptual* 모델, 도메인 객체 모델, 분석 객체 모델
    - ➢ 도메인 모델은 데이터 모델이 아님.
- UP의 정의
  - 소프트웨어 객체가 아닌 실제 상황 개념적인 클래스의 표현
  - 소프트웨어 클래스를 기술하는 다이어그램들, 혹은 소프트웨어 아키텍처의 도메인 계층*layer*, 혹은 책임성이 있는 소프트웨어 객체들을 의미하지 않음.
  - 비즈니스 모델링 원칙에서 생성될 수 있는 산출물 중에 하나로 간주
    - ➢ "비즈니스 도메인에 중요한 '사물' 이나 제품을 설명하는데 초점을 맞춘" UP의 비즈니스 객체 모델BOM의 특수화된 형태
    - ➢ 하나의 도메인에 초점

도메인 모델

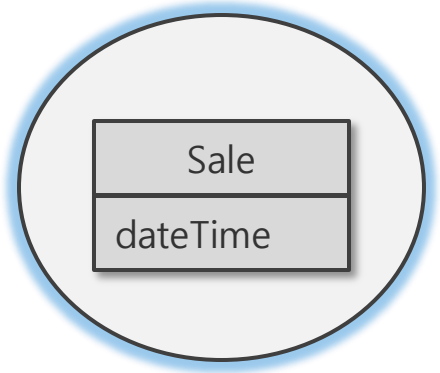
❑ UML 표기

- 오퍼레이션이 정의되지 않은 클래스 다이어그램으로 묘사
  - 개념적인 관점*conceptual perspective* 제공
  - 도메인 객체 혹은 개념 클래스
  - 개념 클래스 간의 연관*association*
  - 개념 클래스의 속성
- 시각적인 사전*visual dictionary*
  - UML을 사용한 정보는 평문으로 표현될 수 있음.
  - 도메인의 중요한 추상화, 도메인 어휘*vocabulary*, 정보 내용*information content*에 대한 시각적인 사전

도메인 모델

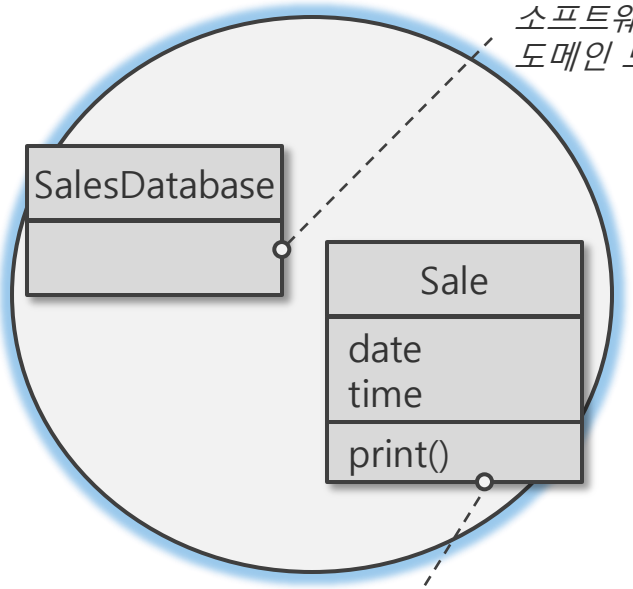
❑ 도메인 모델이 아닌 요소

- Java나 C# 클래스와 같은 소프트웨어 객체나 책임성을 가진 객체가 아닌 관심이 있는 실제 상황 도메인의 사물에 대한 시각화
- 소프트웨어 비즈니스 객체의 사진
  - 윈도우나 데이터베이스와 같은 소프트웨어 산출물
  - 책임성이나 메소드



관심 도메인의
실세계 개념의 시각화

소프트웨어 클래스의 사진이 아님



소프트웨어 산출물 :
도메인 모델 요소가 아님

소프트웨어 클래스 :
도메인 모델 요소가 아님

도메인 모델

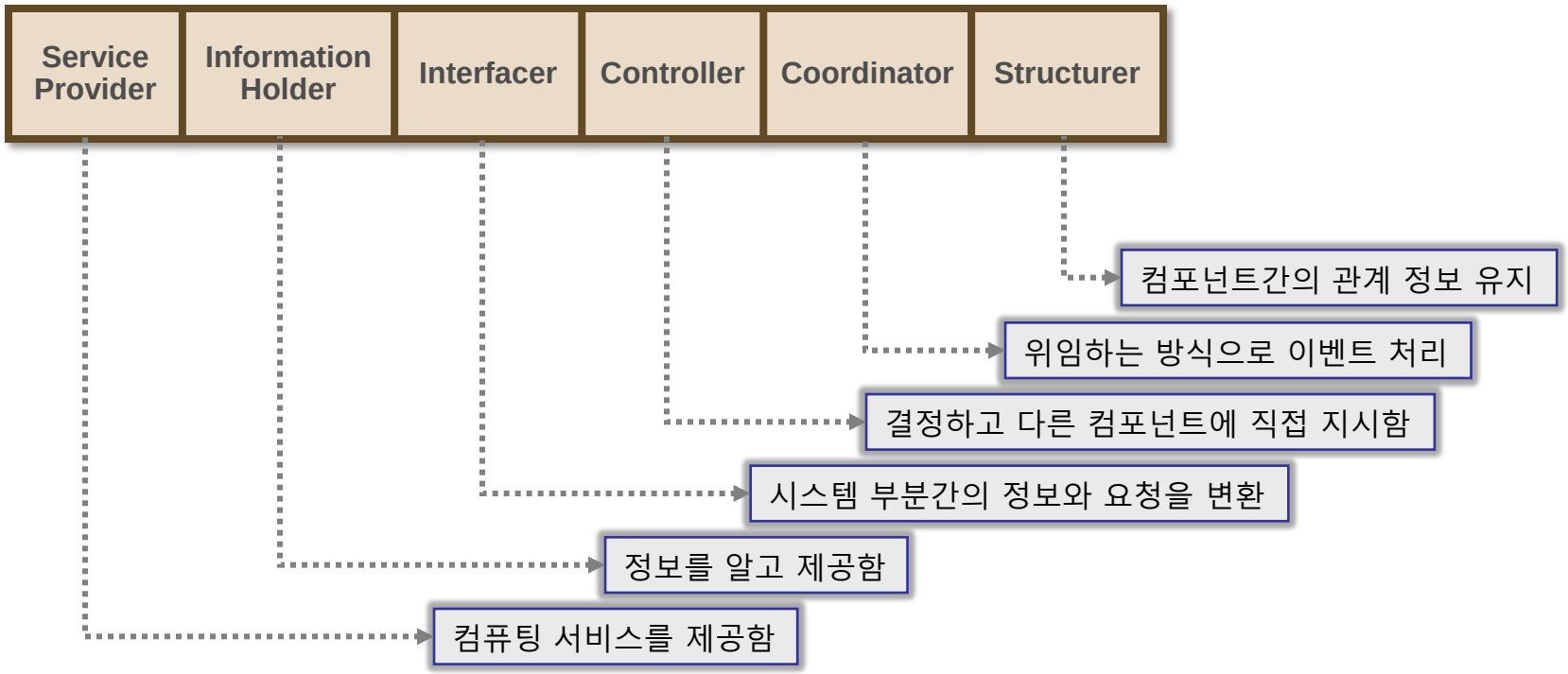
❑ 도메인 모델의 격리*isolation*

- 시스템의 다른 기능function과 분리
  - 도메인 개념을 소프트웨어 기술과 관련된 다른 개념과 혼동 회피
  - 도메인의 직관력 소실 회피
- 도메인 모델 설계 접근방법
  - 책임성 기반 설계*responsibility-driven design*
  - 계약에 의한 설계*design by contract*
  - 객체지향 설계*object-oriented design*
  - 모델 기반 설계*Model-driven design*

도메인 모델

❑ 책임성 기반 설계*responsibility-driven design*

- 역할에 의한 객체 분류



❑ 계약에 의한 설계*design by contract*

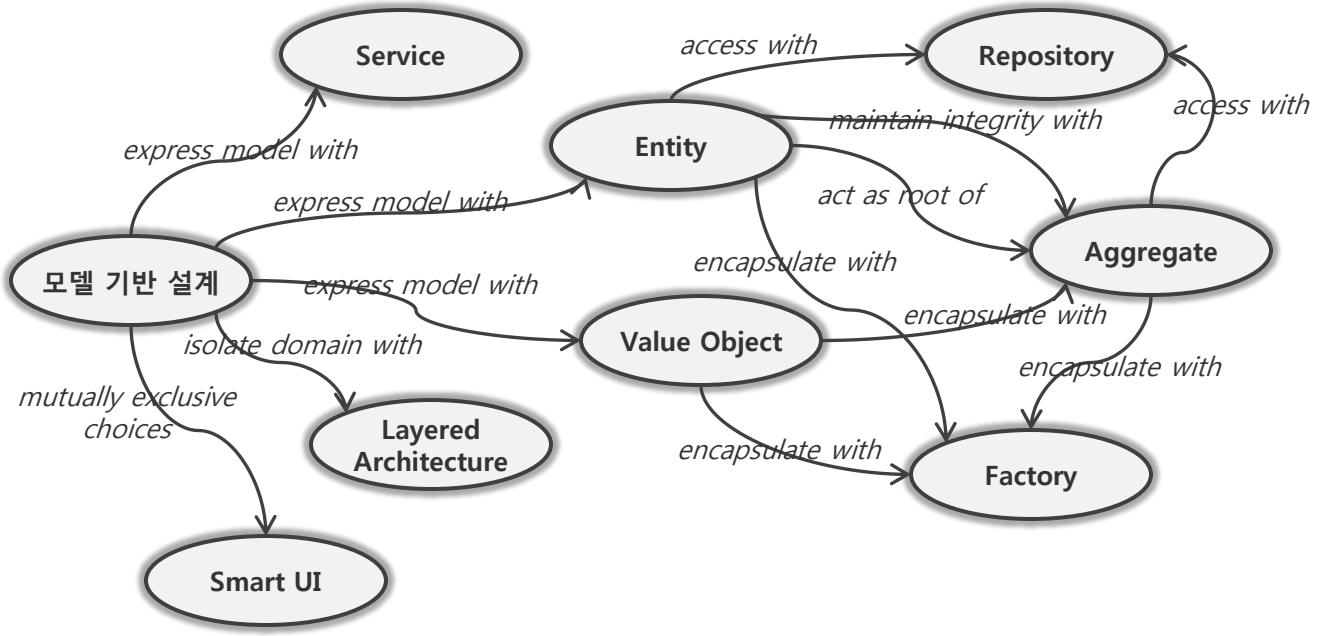
- 계약contract의 의미
  - 법적인 표현 형태
    - ➢ 계약 주체 당사자가 의무obligation를 수용하고, 권리right를 획득
  - 객체지향 관점
    - ➢ 객체의 책임성이 명확하고 애매모호하지 않게 수행됨
    - ➢ 객체는 측정 자극(권리)가 충족될 때에만if and only if 서비스(의무)를 실행하는 책임이 있음.

주체	의무	권리
고객	2000원 지불 80g 이하의 물건 서울내 배송 주소 지정	4시간 내 전달되는 물건
배송 서비스 회사	4시간 내에 물건 배송	배송 주소는 서울 내 2000원 받음 모든 물건은 80g 이하

도메인 모델

❑ 모델 기반 설계*model-driven design*

- 분석은 이해되고 의미있는 방법으로 도메인에서 기본적인 개념을 포함
- 설계는 대상 배포 환경에 효과적으로 수행되고 어플리케이션이 해결하는 문제를 정확하게 푸는 프로젝트에 사용되는 프로그래밍 도구와 같이 구축될 수 있는 객체들의 집합을 지정



2. 구조다이어그램

- ❑ 클래스 다이어그램
- ❑ 객체 다이어그램

클래스 다이어그램

ONE STEP AHEAD

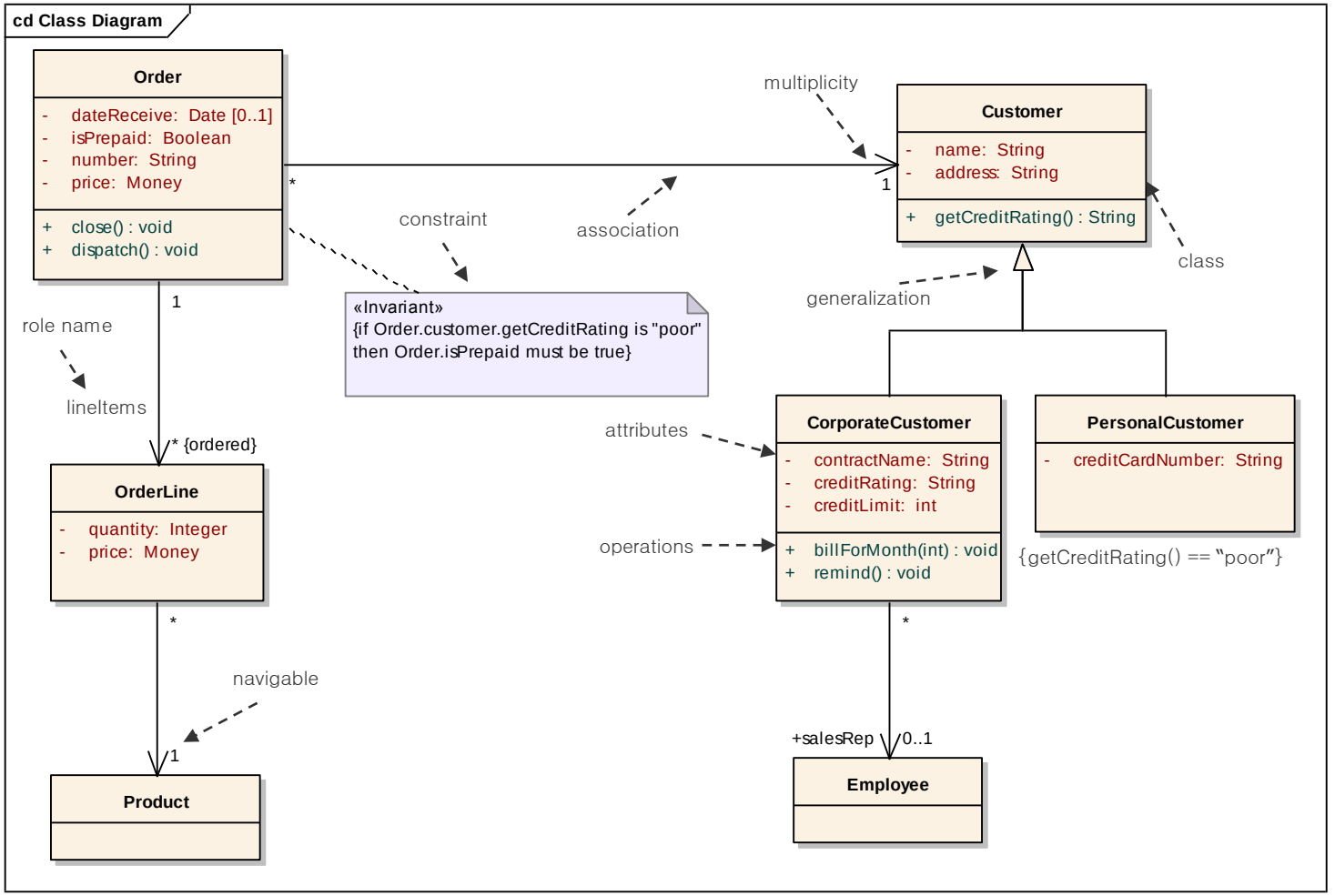
- 개요
- 클래스 다이어그램과 객체 다이어그램
- 클래스 란?
- 클래스 관계

❑ UML의 두 가지 정적 다이어그램

- 클래스 다이어그램
  - UML 다이어그램의 핵심, UML 도구의 코드 생성은 기본적으로 클래스 다이어그램을 기반
  - 클래스, 연관관계, 집합관계, 복합관계, 일반관계를 보여줌
  - 널리 사용되며 그 모델링 개념의 폭이 넓음
    - 기본 개념 : 모든 사람들에게 필요
    - 고급 개념 : 많이 사용 되지 않음
  - 객체 타입인 클래스를 표현하는 다이어그램
  - 클래스의 프로퍼티와 오퍼레이션 및 제약사항을 보여줌
    - 클래스의 특징(feature) = 클래스의 프로퍼티 + 오퍼레이션
  - 클래스 다이어그램에서 클래스는 클래스 개념을 지원하는 언어에서 직접 구현 될 수 있음
    - 예) Java, C++
- 객체 다이어그램
  - 특정 시점에서 객체들의 스냅샷
  - 클래스의 인스턴스와 인스턴스 사이의 링크를 보여줌
  - 클래스 다이어그램을 설명하는 간단한 예를 보여줌

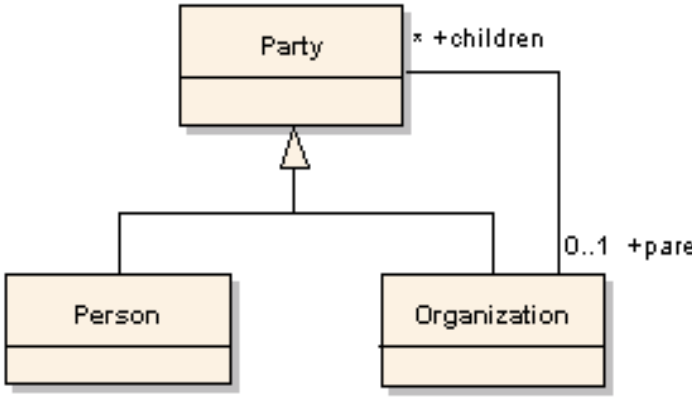
클래스 다이어그램

❑ 클래스 다이어그램의 예

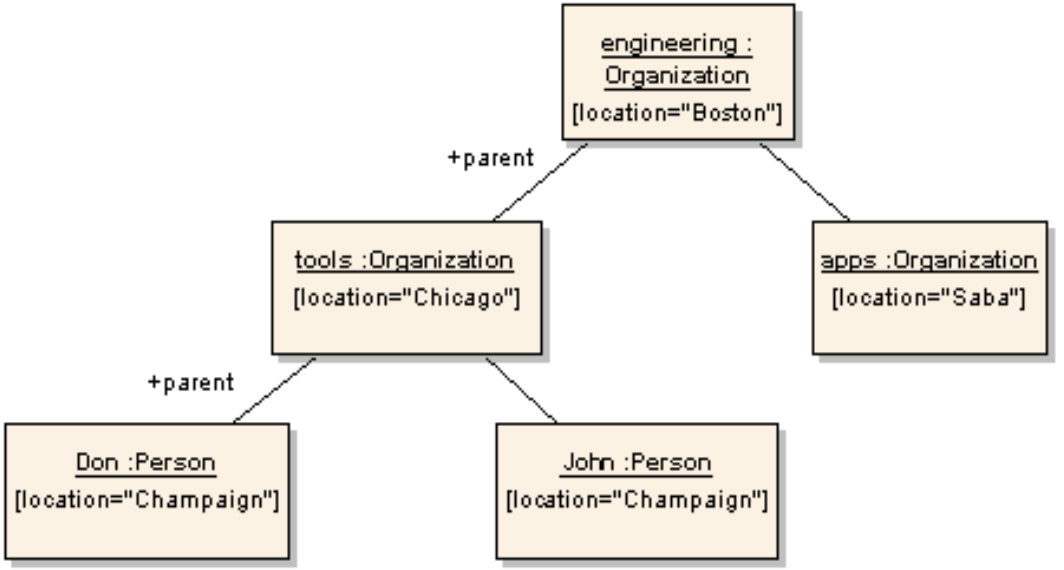


객체 다이어그램

❑ 객체 다이어그램의 예



파티 복합 구조
(Party Composition structure)
클래스 다이어그램

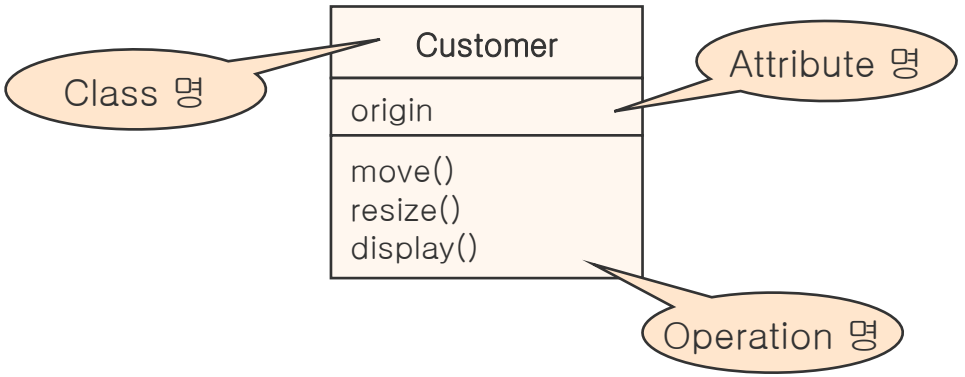


파티의 예제 인스턴스들을 보이는 객체 다이어그램

클래스 란?

❑ 클래스란 무엇인가?

- 사물들의 추상화
- 클래스는 가장 중요한 구성 요소
- 동일한 속성(attribute), 오퍼레이션(operation), 연관(relation), 그리고 의미를 공유하는 객체 집합을 표현
- 하나 또는 그 이상의 Interface를 구현



프로퍼티(Properties)(1/2)

❑ 프로퍼티 개요

- 프로퍼티는 클래스의 구조적인 측면을 나타내는 것
- 속성(attribute)이나 연관(association)으로 표현

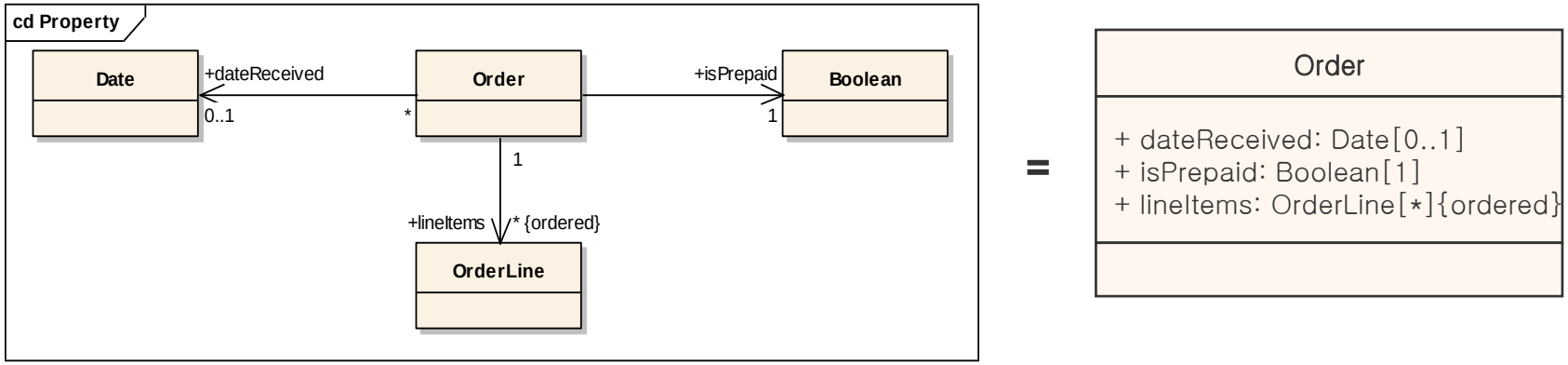
❑ 속성(Attributes)

- 클래스 내부에 정의되어 프로퍼티를 설명
- visibility name : type multiplicity = default {property-string}
  ex) - name : String[1] = “Untitled” {readonly}
  - visibility: 가시성, public(+), private(-)
  - name: 속성의 이름
  - type: 속성의 종류
  - multiplicity: 다중성
  - default: 초기 값
  - {property-string}: 속성의 추가 특성을 표현

프로퍼티(Properties)(2/2)

❑ 연관(Associations)

- 연관관계를 이용한 프로퍼티 표현



- 연관관계는 소스 클래스와 타겟 클래스를 연결
- 프로퍼티의 이름은 역할이름으로 표현
- 연관관계의 양끝에 개수를 표현(참여하는 속성의 개수 표현)
- 값 객체와 같은 것들은 속성으로 표현, 참조 객체들은 연관으로 표현

다중성(Multiplicity)

❑ 다중성 개요

- 얼마나 많은 객체들이 존재하는지를 표현
- 속성에 대한 다중성 표현
  - attributeName: AttributeType [Multiplicity]

❑ 다중성의 의미

UML 다중성	의미
1	정확히 1
0..1	0 이거나, 혹은 1
*	무제한(0 포함)
1..*	적어도 하나 이상
2..6	2 부터 6 까지
2, 4	2 이거나, 혹은 4

❑ 다중성 예제

- name: String [1..2] = “Michael”

가시성(Visibility)

❑ 가시성 개요

- 클래스는 공용(public) 요소와 전용(private) 요소 보유
  - 공용(public) 요소는 다른 클래스에 의해 사용 가능
  - 전용(private) 요소는 소유 클래스에 의해서만 사용 가능
- 각각의 프로그래밍 언어는 자신만의 가시성 규칙 보유
  - 여러 프로그래밍 언어 사용자에게 혼동
- UML의 가시성 지시자(visibility indicator)
  - +(public), -(private), ~(package), #(protected)
- 가시성을 사용할 때는 사용중인 언어의 규칙을 적용

오퍼레이션(Operation)(1/2)

❑ 오퍼레이션 개요

- 클래스가 수행할 행위

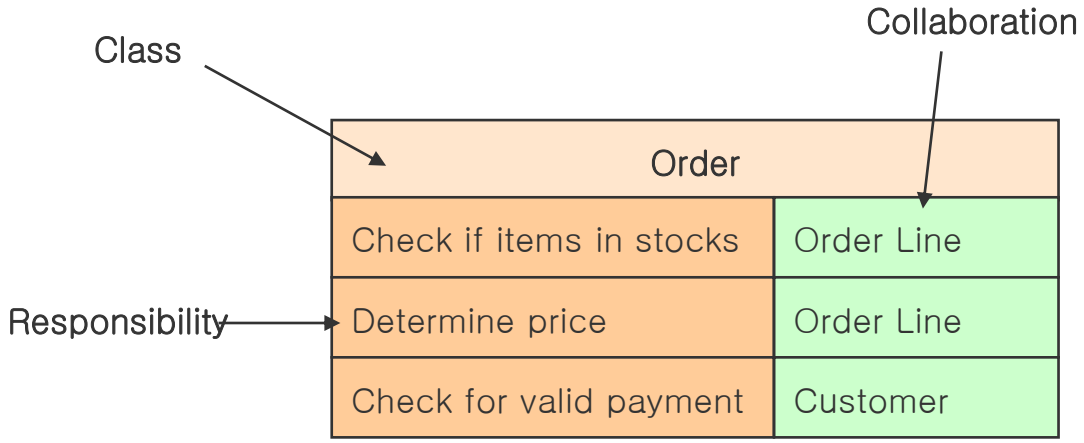
❑ 오퍼레이션 문법

- visibility name(parameter-list) : return- type{property-string}
  - visibility: 가시성, public(+) 또는 private(-)
  - name: 오퍼레이션의 이름(문자열)
  - parameter list: 파라메터 목록, 오퍼레이션을 위한 parameter list
  - return-type: 존재한다면 반환되는 값의 type
  - property-string : 연산에 적용되는 특성 값
- parameter list의 파라메터 표현
  - direction name : type = default value
    - name, type, default value는 속성)의 표현과 동일
    - direction: 파라메터 입력(in), 출력(out), 입출력(inout)을 표기

오퍼레이션(Operation)(2/2)

❑ CRC(Class Responsibility Collaboration) 카드

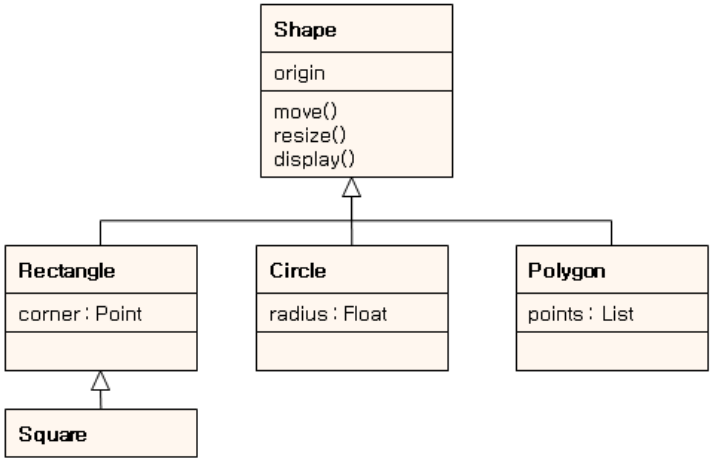
- 개념 모델상에서는 클래스에 오퍼레이션을 사용하지 않고, CRC 카드를 사용



일반화(Generalization)

❑ 일반화 개요

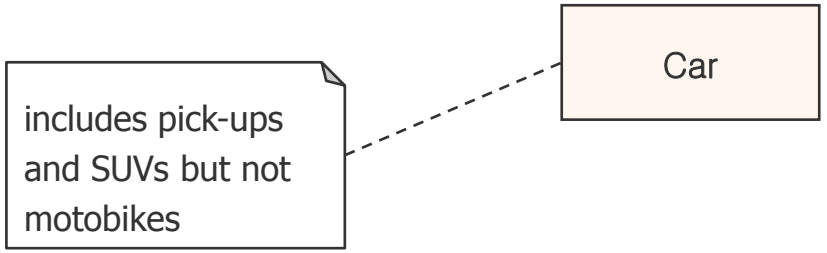
- "**is-a-kind-of**" 관계
- 일반화된 사물(supertype)과 보다 특수화된 사물(subtype)들 사이의 관계를 표현
- 하위타입의 인스턴스의 특징은 상위타입의 인스턴스가 가지는 모든 특징을 가짐
  - 속성, 연관, 오퍼레이션
- 하위타입의 인스턴스는 상위타입의 인스턴스를 대체(substitutability) 가능



노트(Notes)와 주석(Comments)

❑ 노트 개요

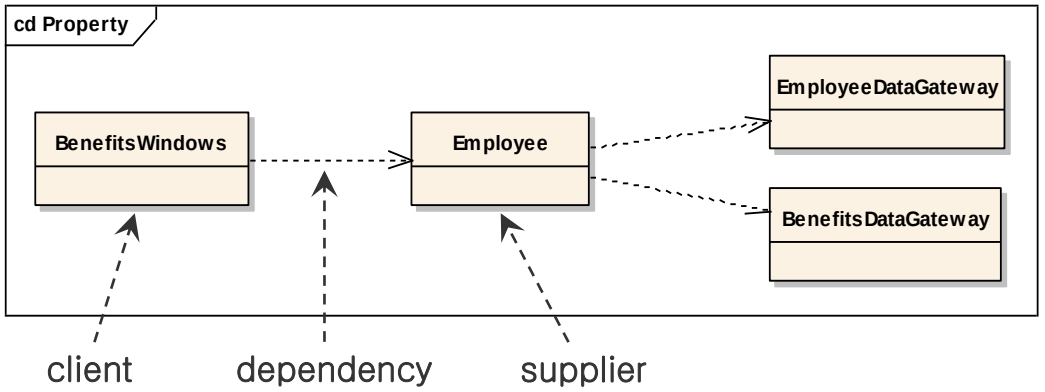
- 노트는 다이어그램을 설명하기 위한 주석
- 다이어그램과 연결되거나 그 자체로 사용될 수 있음



의존성(dependency)

❑ 의존성 개요

- 한 요소(supplier)의 변화가 다른 요소(client)에 영향을 미칠 경우 의존성이 존재
  - 어떤 클래스가 다른 클래스로 메시지를 전송할 경우
  - 어떤 클래스가 다른 클래스를 데이터의 일부로 소유하고 있을 경우
  - 어떤 클래스가 다른 클래스를 오퍼레이션의 파라메터로 언급하고 있을 경우
- 시스템의 대형화로 의존성 제어가 더욱 중요
- 의존성을 통제하지 못할 경우, 특정 요소의 변화에 대한 파급효과는 커짐
- UML에서 모든 요소들의 의존성을 표현할 수 있음
- 기본적으로 의존성은 이행성(transitive)이 없음
- 의존성을 최소화 하는 것이 매우 중요, 특히 패키지 레벨의 순환 의존성을 제거해야 함



의존성(dependency)

실습) 전자 상거래 (EC는) 개발한다.

고객은 시스템을 통해 다양한 컴퓨터 제품을 검색하고

하드웨어 및 소프트웨어를 주문한다.

고객은 신용 카드 또는 온라인 송금을 통해 지불할 수 있습니다.

일단 고객이 자신의 주문에 대한 지불하고 시스템은 온라인 또는 오프라인으로 주문한 제품을 제공합니다.

소프트웨어를 온라인으로 다운로드 받을 수 있고, 나머지는 택배 회사에 의해 배달된다.

이 시스템은 모든 판매 및 거래를 추적합니다.

키워드(Keyword)

❑ 키워드 개요

- 기존의 모든 UML 심볼의 의미를 기억하기 매우 곤란하여 키워드를 사용
- UML에 정의되지 않은 심볼이지만 의미가 비슷한 경우 기존 UML 심볼을 사용하고 차이를 키워드로 명시
- UML 인터페이스(interface)
  - 키워드의 대표적인 예
  - 메소드 몸체가 없고 오직 공용 오퍼레이션만을 가진 클래스
  - 클래스 아이콘에 <<interface>> 키워드를 사용하여 표시

❑ 키워드 표기법

- 이중 꺽쇠(<<~>>) 내에 텍스트 표시
  - <<interface>>
- 중괄호({~}) 내에 텍스트 표시
  - {abstract}
- 이중 꺽쇠 내에 표기할 내용과 중괄호 내에 표기할 내용 구분은 불명확

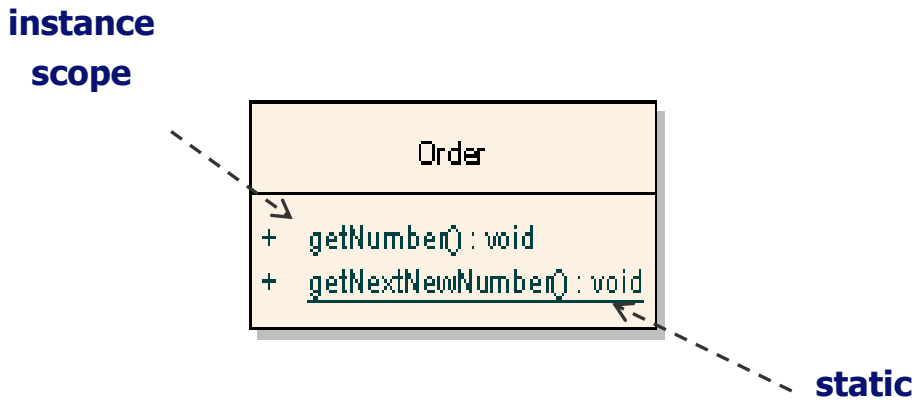
❑ 스테레오타입(stereotype)

- UML 1에서 이중 꺽쇠(<<~>>)로 표현된 키워드

정적 오퍼레이션과 속성(Static Operations and Attributes)

❑ 정적 오퍼레이션과 속성 개요

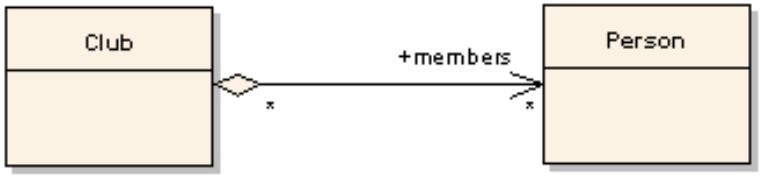
- 인스턴스가 아니라 클래스에 적용되는 오퍼레이션 또는 속성
- 클래스 다이어그램 상에 밑줄을 그어 표현



집합(Aggregation)과 복합(Composition)

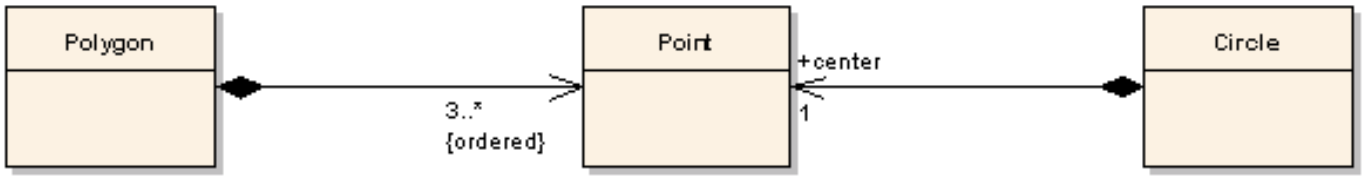
❑ 집합

- part-of 관계
- 연관관계(association)와의 차이가 모호
  - UML은 집합을 포함하지만 특별한 의미를 부여하지는 않음



❑ 복합

- 비 공유 규칙(“no sharing” rule)
  - 인스턴스는 오직 하나의 소유 클래스만을 가짐
    - ➢ 클래스 다이어그램 상에는 여러 개의 소유 클래스 표현 가능
  - 포함하는 클래스 쪽의 다중성은 0..1 또는 1
  - 포함하는 인스턴스 삭제 시 자동적으로 포함되는 인스턴스 삭제
- 용도 : 값 객체(value object), 강한 배타적인 소유관계 표현



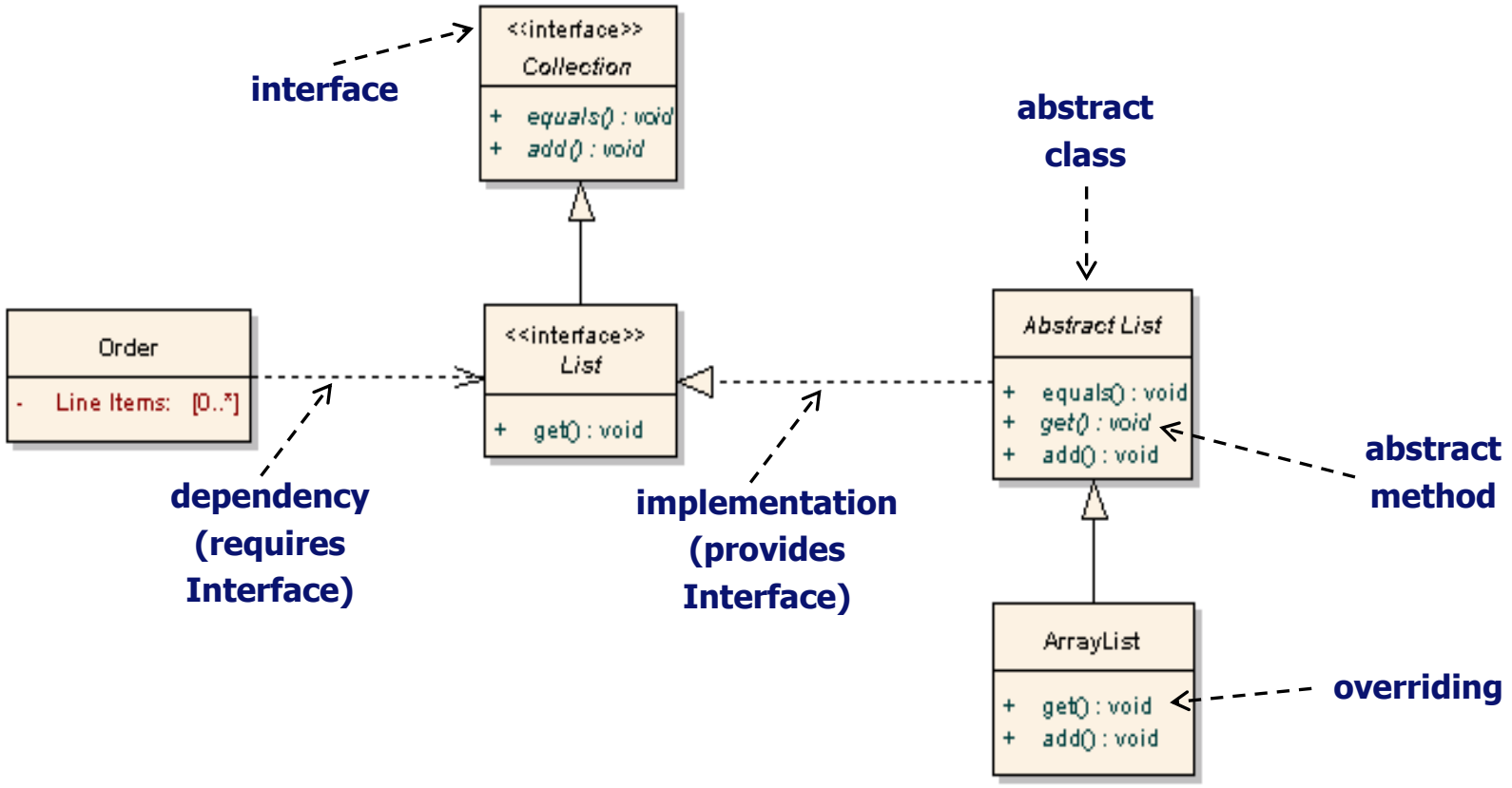
인터페이스(Interface)와 추상(Abstract) 클래스(1/3)

- **추상 클래스**
  - 직접 인스턴스를 생성할 수 없는 클래스
  - 추상 오퍼레이션(abstract operation)
    - 구현이 없이 순수한 정의(pure declaration)만을 가진 오퍼레이션
    - 추상 클래스는 하나 이상의 추상 오퍼레이션 보유
  - 추상 클래스와 추상 오퍼레이션은 이탤릭 체로 표현
- **인터페이스**
  - 어떤 구현도 가지지 않는 클래스
    - 모든 특성이 추상(abstract)
    - 키워드 <<interface>>를 사용하여 표현
- **클래스와 인터페이스 간의 관계**
  - 인터페이스 제공(provides interface)
    - 클래스가 인터페이스를 치환(substitutable) 가능
    - 인터페이스를 구현하거나 인터페이스의 서브타입을 구현
  - 인터페이스 요구(requires interface)
    - 작업을 수행하기 위해 인터페이스의 인스턴스를 요구
    - 인터페이스에 대한 의존(dependency) 관계

인터페이스(Interface)와 추상(Abstract) 클래스(2/3)

❑ 인터페이스의 장점

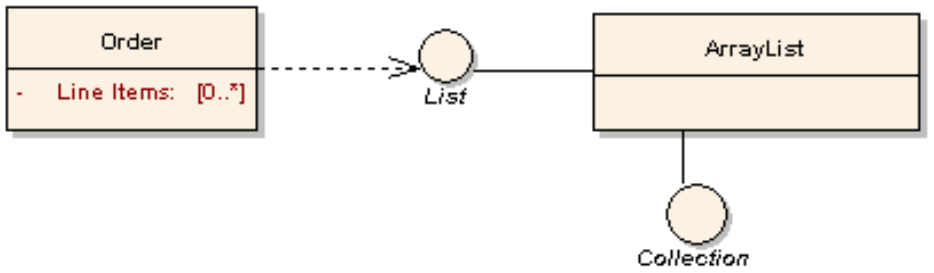
- 구현의 변경이 용이
- 구현이 아닌 인터페이스에 대해 프로그래밍
  - 변경에 의해 영향 받는 부분을 최소화



인터페이스(Interface)와 추상(Abstract) 클래스(3/3)

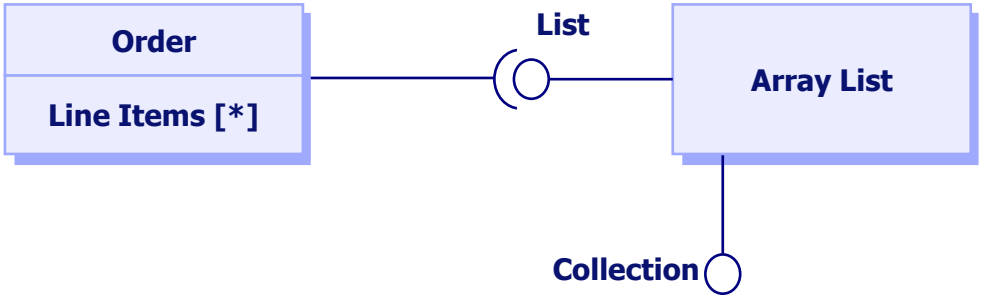
❑ 인터페이스 - UML 1 표기법

- 인터페이스를 롤리팝(lollipop)으로 표현
- 의존 관계 사용



❑ 인터페이스 - UML 2 표기법

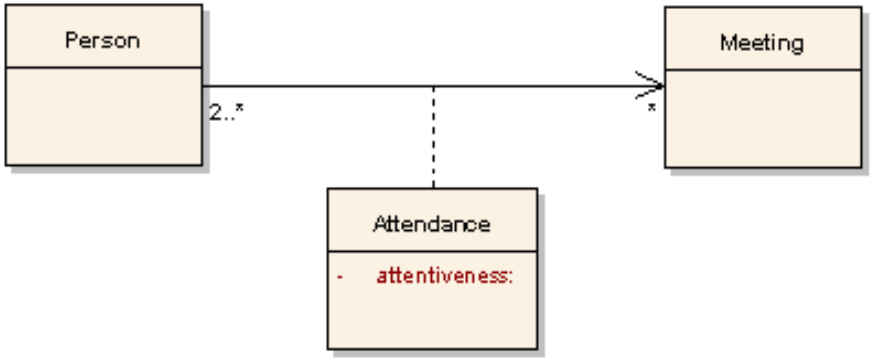
- 의존관계를 소켓(socket) 표기법으로 대체



연관 클래스(Association Class)

❑ 연관 클래스 개요

- 연관관계에 속성, 오퍼레이션, 다른 특성 추가 가능
- 참여 객체간에 제약사항 부가
  - 참여하는 두 객체 간에 오직 하나의 연관 클래스 인스턴스 만이 존재 가능



연관 클래스

연관 클래스를 완전한 클래스로 만들기

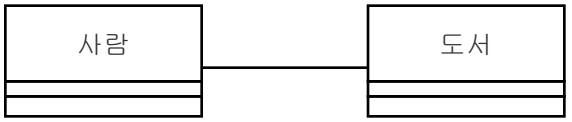
연관 관계(1/3)

❑ 연관 관계

- 한 클래스가 다른 클래스를 인지하는 것
- 연관은 매우 광범위한 의미를 갖는다.
- 연관 관계의 의미를 명확하게 표현해야 한다.

❑ 모호한 연관 관계

- 예)
  - 사람이 도서를 대출한다. 사람은 학생 또는 교수를 나타낸다.
  - 사람이 도서를 관리한다. 사서가 도서의 등록/삭제 등을 한다.

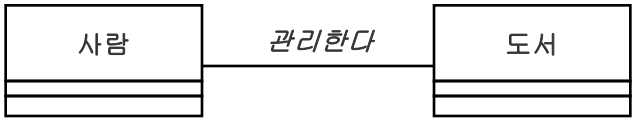
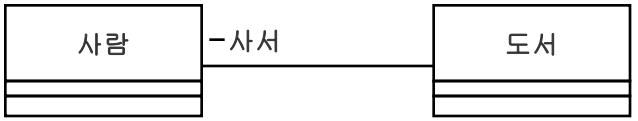


모호한 연관 관계

연관 관계(2/3)

❑ 연관의 이름과 역할

- 연관의 이름은 실선 위에 표시 – 동사 또는 동사구
- 역할의 이름은 연관 관계를 속성으로 표현할 때 상대 객체에 대한 이름으로 사용

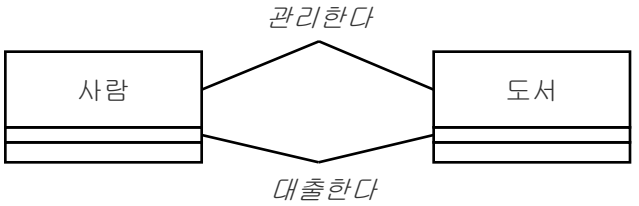
(1) 연관 이름의 사용	**(2)** 역할의 사용
	

class 사람 {
private 도서 the_도서;
}
	class 도서 {
	private 사람 사서;
	}

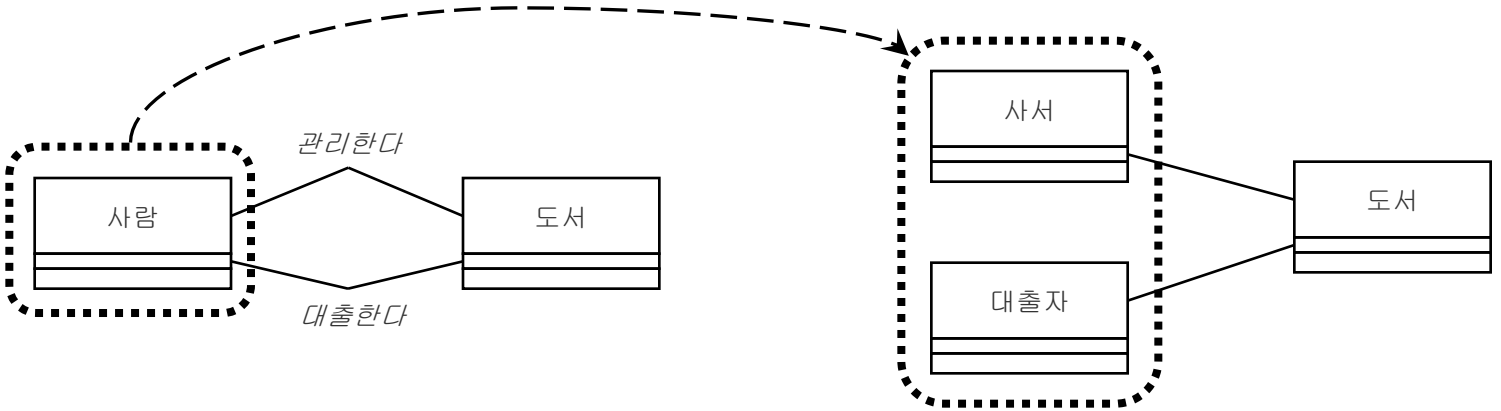
연관 관계(3/3)

❑ 복수 연관

- 동일한 두 클래스 사이에 두 개 이상의 연관 관계가 맺어지는 것
- 두 클래스가 명확하게 다른 의미의 관계를 맺는 경우 사용



- 복수 연관 관계의 신중한 사용



복수 연관 사용의 대안