



CSE053 Software Engineering

Lecture: Structural modeling

Software engineering laboratory
Yeungnam university



Learning objectives

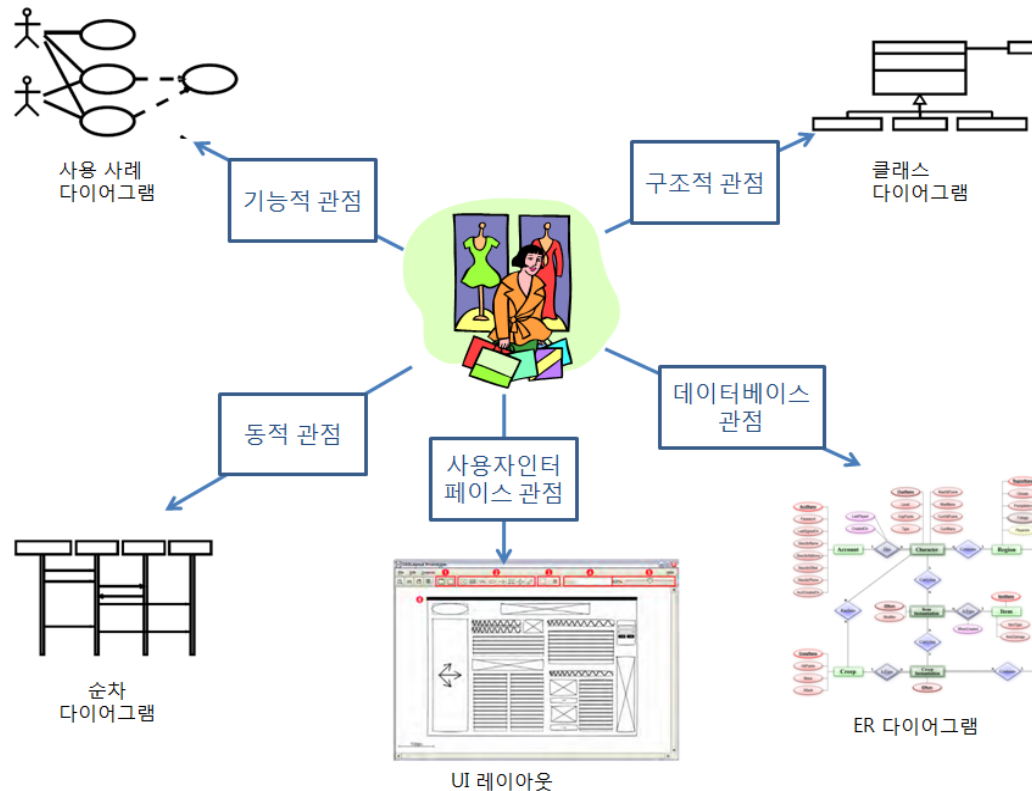
- 구조적 모델
- 클래스와 속성
- 관계와 오퍼레이션
- 클래스 다이어그램



Design 단계의 관점

■ 시스템을 구성하는 빌딩 블록을 보여 줄 수 있어야 함

- 건축물 설계도면의 단면도 평면도
- 관점에 따라 나타내는 대상이 다름

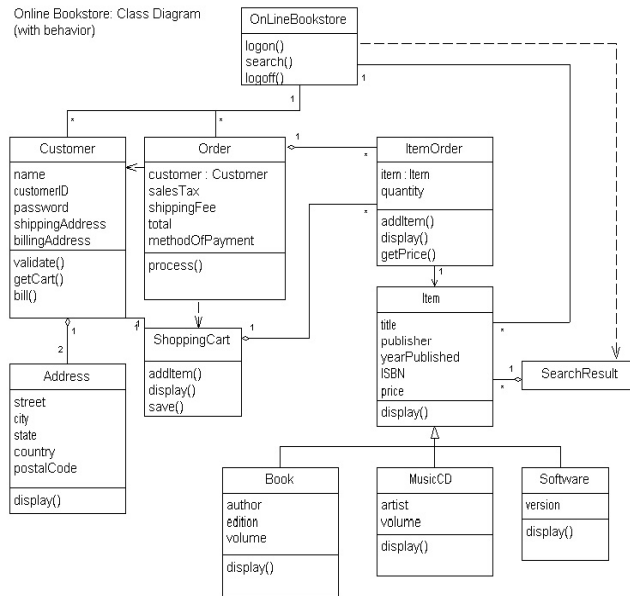


“구조적 관점”

■ 시스템 내부의 관점

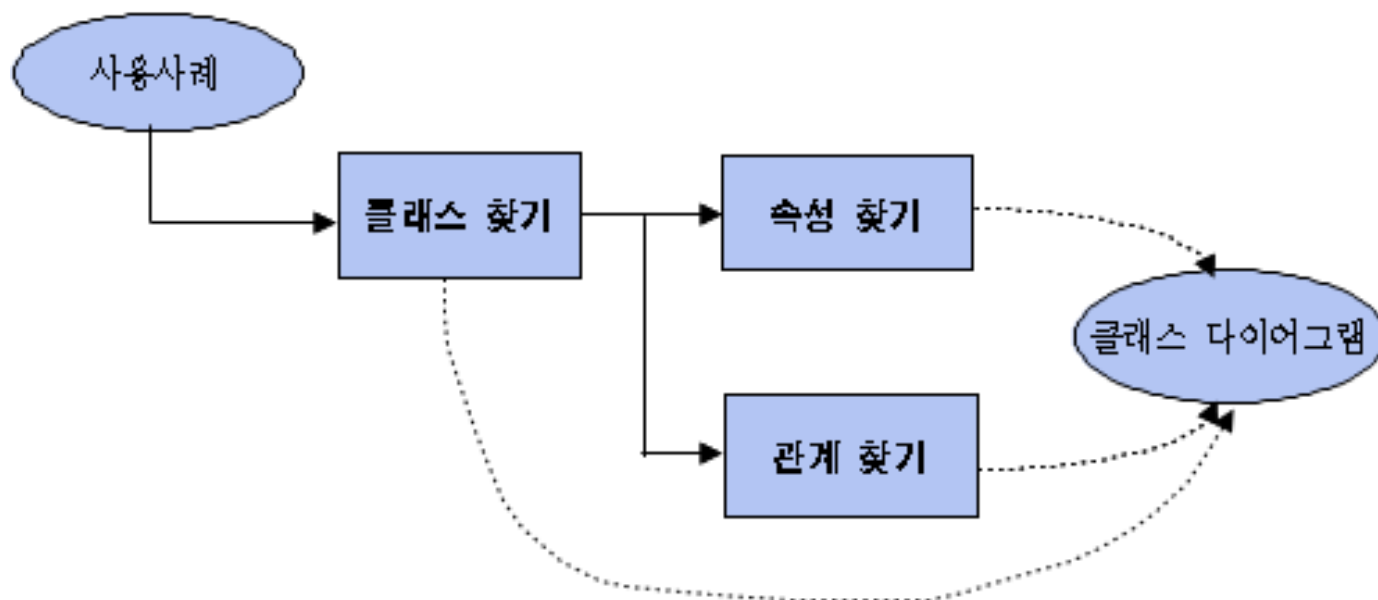
- 특히 구조라는 관점의 뷰(view)
- 어떤 구성요소가 있고 이들이 어떤 관계를 맺고 있는지 표현
- 시간이 흐르더라도 변하지 않는 정적 구조

■ 새로운 문제 도메인의 중요한 Class 발견



“구조적 관점” 단계의 작업

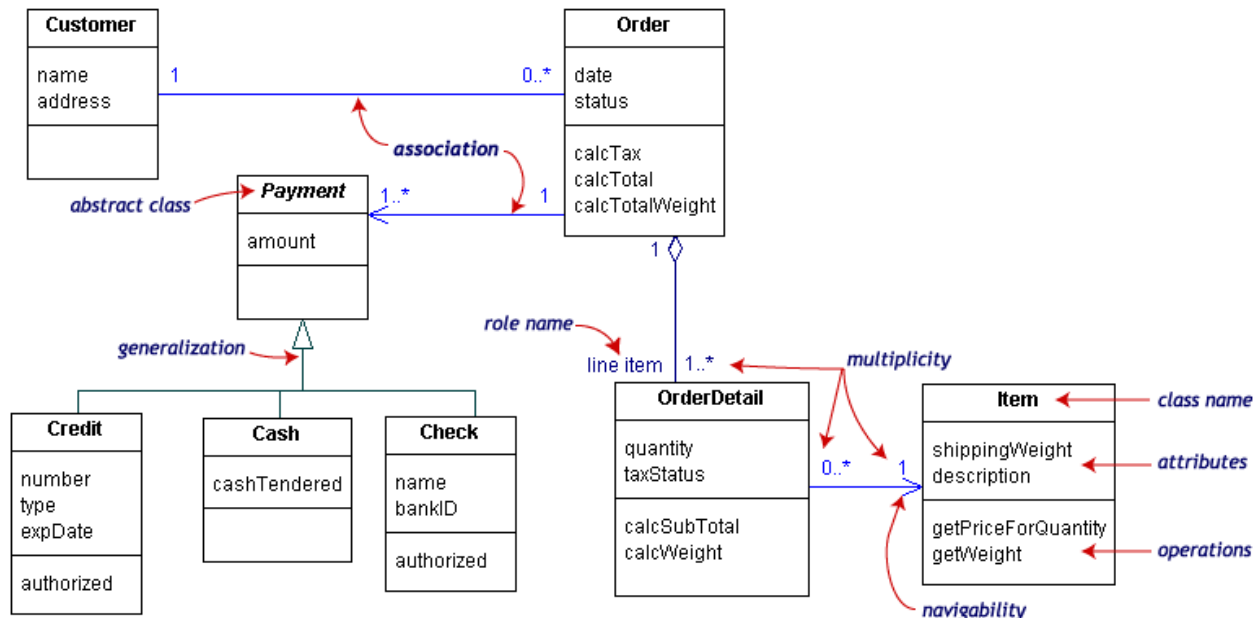
- 요구를 만족시키는 내부적 구조를 Class 관점에서 표현
 - Class(attribute, operation) 찾기
 - Relationship 찾기
 - Class diagram 그리기



Class diagram

■ 개요

- “Class”라고 하는 객체지향 설계단위를 이용하여 시스템의 정적인 structure (구조)를 표현한 다이어그램
 - 분석, 설계, 구현 등 다양한 상황에서 그 사용목적에 맞게 상세화 정도를 조절하여 기술 가능
 - 프로그램의 구조를 잘 나타내고 있어 코딩 작업에 근간이 됨
- Attribute, Operation 정의, Interface 정의 등
- (1)Class와 (2)Relationship으로 구성

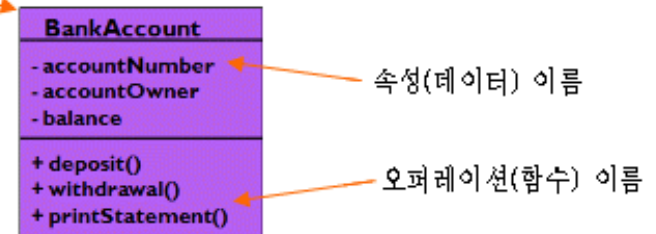


[1] Class

■ 구성요소

- Class name(클래스명), attribute(속성), operation(동작)

- Class에 대한 데이터와 행동양식을 정의 클래스 이름



- 클래스(class)

- 공통의 속성, 메서드(오퍼레이션), 관계, 의미를 공유하는 객체들의 집합

- 속성(attribute)

- 클래스의 구조적 특성에 이름을 붙인 것으로 특성에 해당하는 인스턴스가 보유할 수 있는 값의 범위를 기술

- 속성은 영문자 소문자로 시작

- 동작(operation)

- 이름, 타입, 매개변수들과 연관된 행위를 호출하는데 요구되는 제약사항들을 명세하는 클래스의 행위적 특징

[1] Class

■ 구성요소

- (Attribute 표기법)

→ 의미있는 명사형으로 표현

→ Visibility **Name**:**Type** = Default Value

MyDate	
- <u>day</u> :int	← static
- month:int	
- year:int=2007	

메서드의 종류	부호	내용
public	+	자신의 속성이나 동작을 외부에 공개하는 접근 제어
private	-	상속된 파생 클래스만이 액세스할 수 있는 접근 제어
protected	#	구조체의 멤버 함수만 접근할 수 있으며 외부에서 액세스할 수 없는 접근 제어

- (Operation 표기법)

→ 의미있는 동사형으로 표현


→ Visibility Name(Parameter-list):Return type – Expression[Property-String]

MyDate	
- <u>day</u> :int	
- month:int	
- year:int=2007	
+getDay()	
+setDay(d:int)	

[1] Class: 연산(operation) 기본 원칙

■ Operation 이름 정하기 가이드

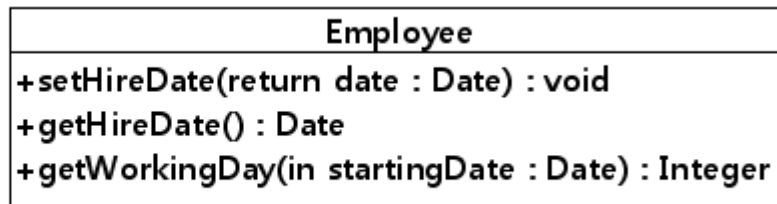
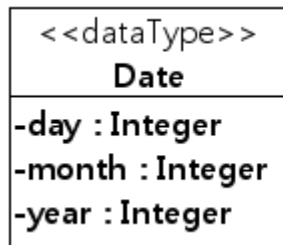
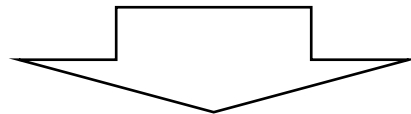
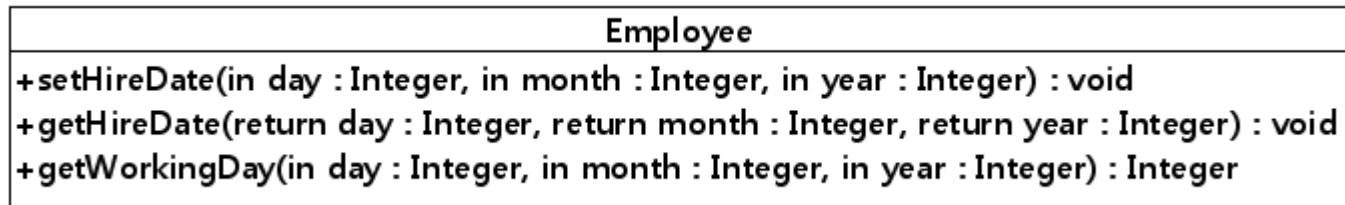
- 연산이 제공하는 유일한 기능의 결과를 뜻하는 용어가 연산의 이름으로 사용되어야 한다.
- 연산의 이름이 연산의 구현 방법을 뜻해서는 안 된다.

부적절한 이름		적절한 이름
Account::calculateBalance():Real		Account::getBalance():Real
User::Compare(id:String, pw:String):Boolean		User::checkValidity(id:String, pw:String):Boolean

- 연산이 제공하는 기능의 전체를 뜻하는 용어가 연산의 이름으로 사용되어야 한다.
- 연산이 정의된 클래스를 수행 주체로 하여 연산의 이름을 결정해야 한다.
 - 클래스는 연산의 수행 주체가 되어야 하므로 클래스를 주어로 하고 연산을 술어로 한 문장이 성립되어야 한다.

[1] Class: 연산(operation) 기본 원칙

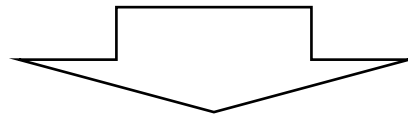
- 관련된 복 수개의 인자/반환 값들을 데이터 타입으로 정의할 수 있다.
 - 연산의 인자 또는 반환 값으로서 여러 개의 변수가 항상 함께 사용된다면 대응되는 복합 타입을 정의할 수 있다.
 - 복합 데이터 타입의 정의 예 1



[1] Class: 연산(operation) 기본 원칙

→ 복합 데이터 타입의 정의 예 2

Rectangle
+getLeftTop(return x : Integer, return y : Integer) : void +getRightBottom(return x : Integer, return y : Integer) : void +setLeftTop(in x : Integer, in y : Integer) : void +setRightBottom(in x : Integer, in y : Integer) : void +getDifference(in r : Rectangle, return x : Integer, return y : Integer) : void



<<dataType>> Point
-x : Integer -y : Integer

Rectangle
+getLeftTop() : Point +getRightBottom() : Point +setLeftTop(in p : Point) : void +setRightBottom(in p : Point) : void +getDifference(in r : Rectangle1) : Point

[2] Relationship

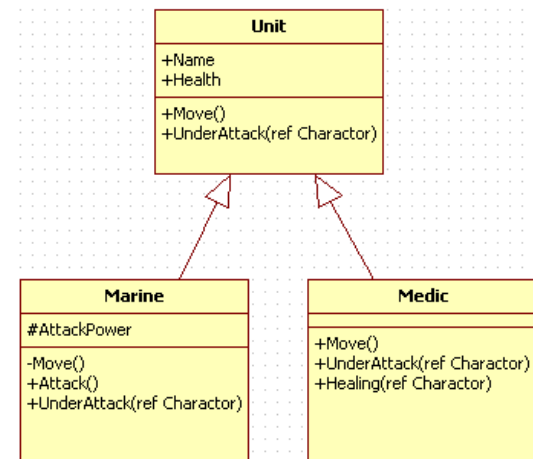
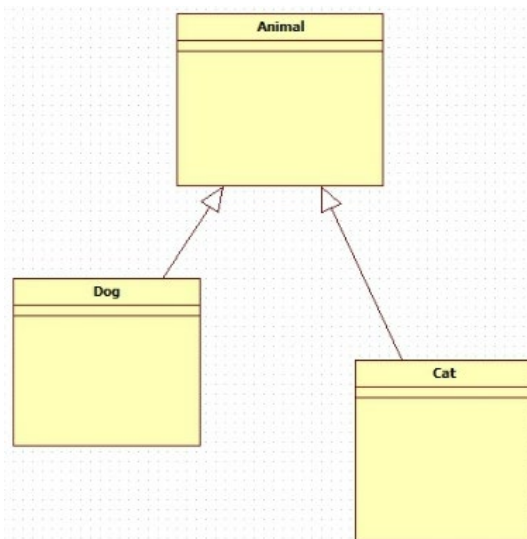
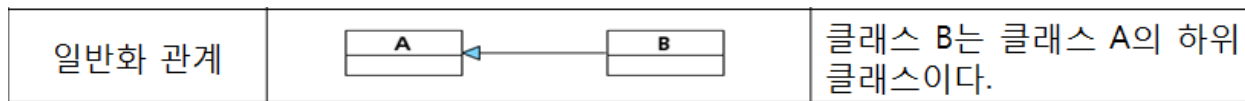
■ 종류

- Generalization(일반화 관계)
 - 일반적인 것(동물)에서 특화된 것(사자)과의 관계를 나타낸다.
 - 보통 Inheritance(상속)을 표현한다.
- Realization(실체화 관계)
 - interface와 그것을 구현한 것과의 관계를 나타낸다.
- Association(연관 관계)
 - 한 instance가 다른instance를 소유하거나 파라미터로 instance를 받아서 처리하는 관계를 나타낸다.
- Dependency(의존관계)
 - 한 객체가 다른객체를 소유하지는 않지만, 다른객체의 변경에 따라서 같이 변경을 해주어야 한다.

[2] Relationship

■ Generalization(일반화 관계)

- 여러 Class가 가진 공통적인 특징을 추출하여 하나의 Class로 일반화시키는 것을 의미
- is-A 관계 => 하나의 종류를 의미!
- OO에서는 Inheritance를 의미

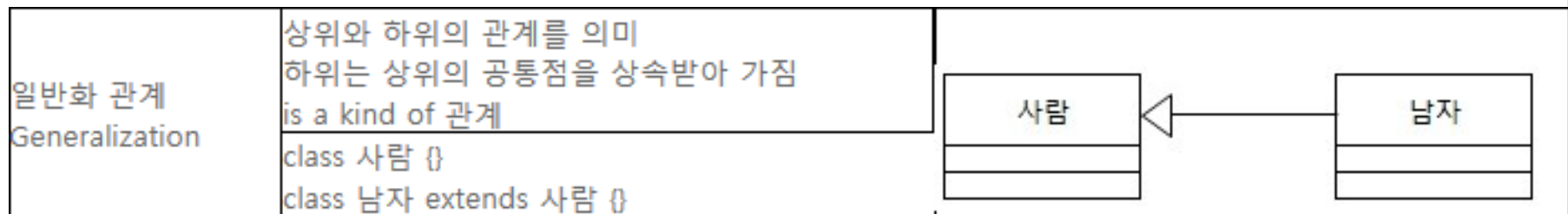
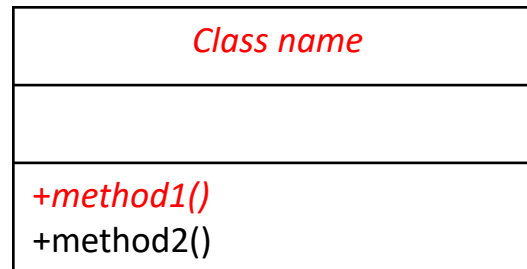


[2] Relationship

■ Generalization(일반화 관계)

- *참고사항

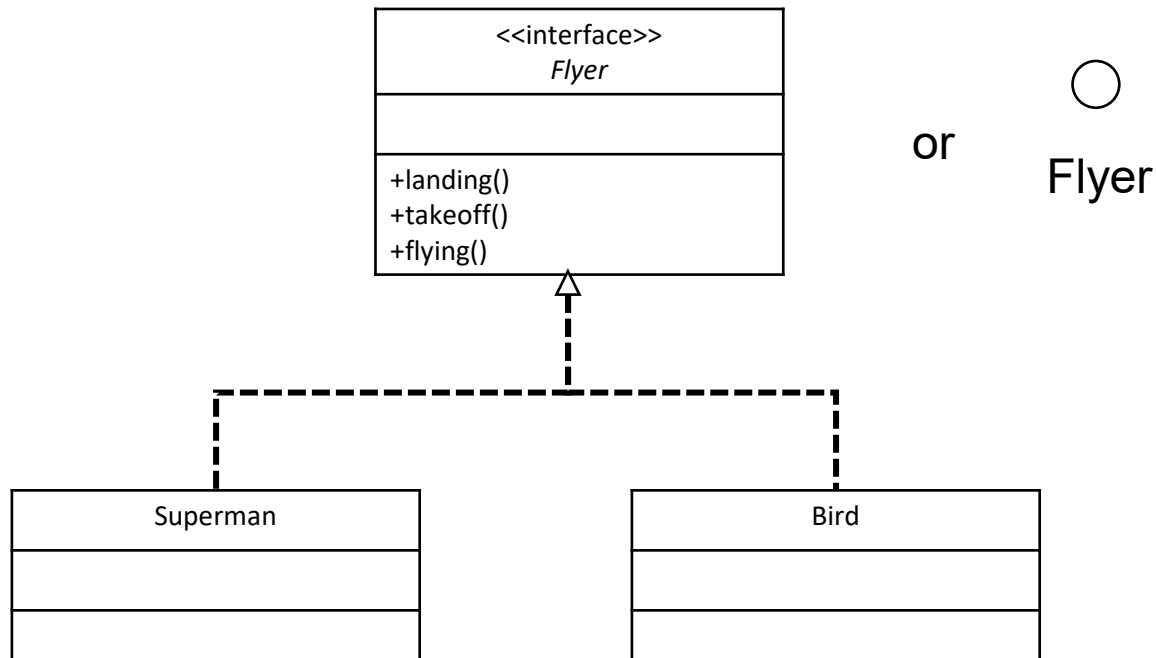
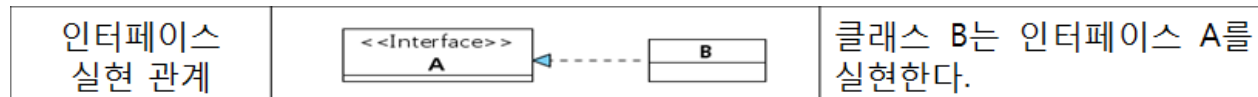
→ 추상형(Abstract)은 Class명이나 method명을 이탤릭체로 표시



[2] Relationship

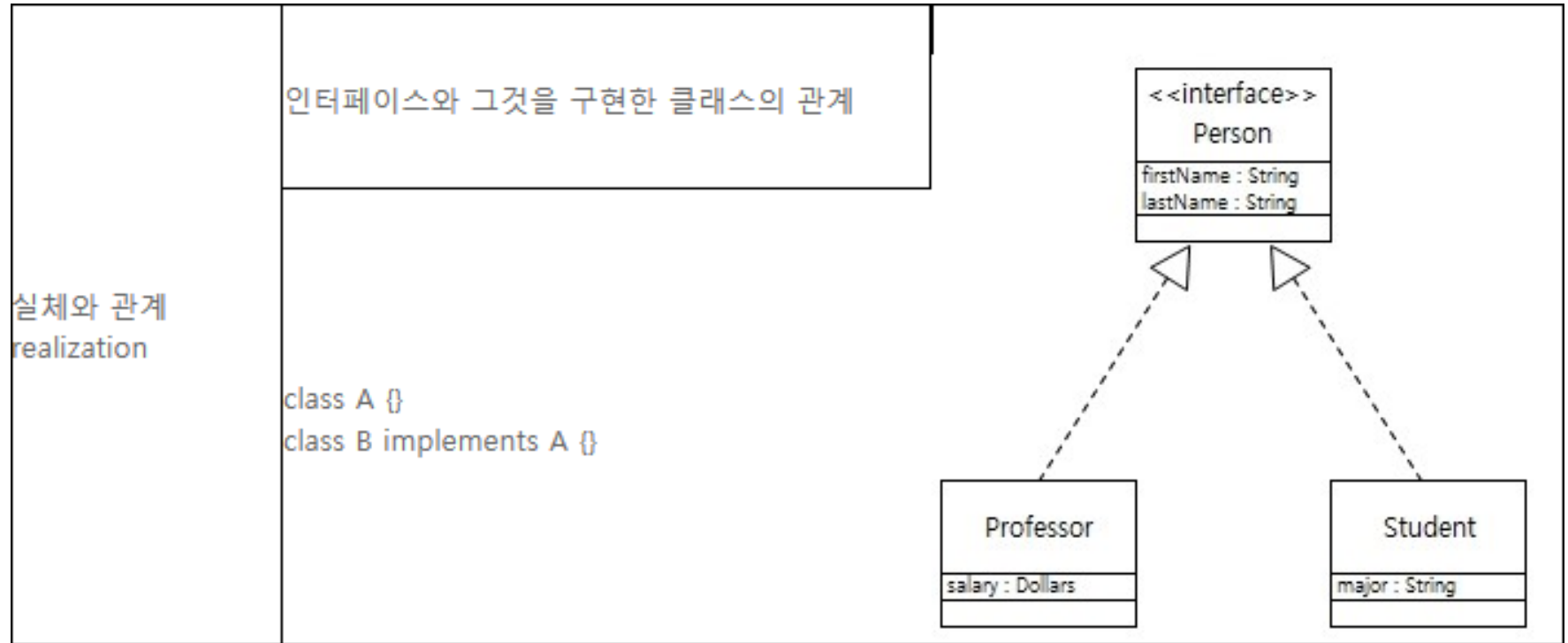
■ Realization(실체화 관계)

- interface와 실제 구현된 class간의 관계
- 완전히 다른 class에 공통적인 기능(method)을 부여하는 것



[2] Relationship

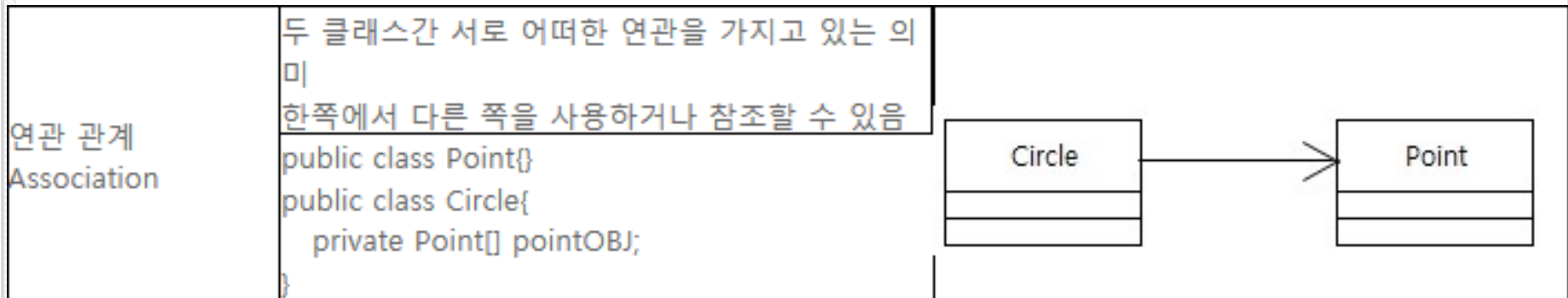
▪ Realization(실체화 관계)



[2] Relationship

■ Association(연관 관계)

- 한 Class가 다른 Class를 소유(사용)하거나, 참조하여 사용할때 (instance 단위)
- Unidirectional vs. Bidirectional
 - 단방향: 클래스간의 관계가 "->" 이렇게 구현이 되며, 화살표의 대상은 자신을 가리키는 클래스의 존재여부를 알지 못한다.
 - 양방향: 클래스간의 관계가 "-" 로 구현되며 서로 연관이 되어있다.



[2] Relationship

■ Association(연관 관계)

- 연관관계는 대상이 되는 객체의 Life Cycle에 따라서 2가지로 분류가 된다

→ ① Aggregation (Basic aggregation, 집합 연관 관계)

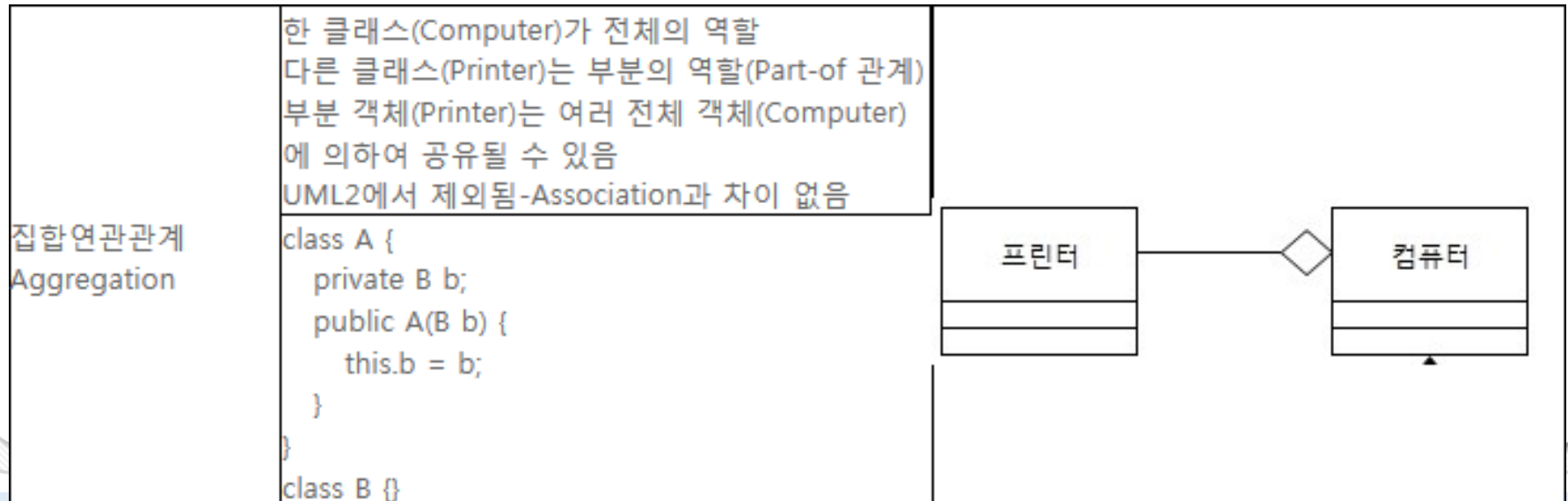
메인 클래스가 삭제될시 대상 클래스는 같이 삭제가 안됨 (라이프 사이클이 다름), 분리가 되도 독립적으로 동작됨 (sub class가 super class 외부에 위치)

약한 결합

has-A 관계, is a part of 관계

전체와 부분을 나타내는 관계이며, 전체와 부분은 서로 독립적인 관계

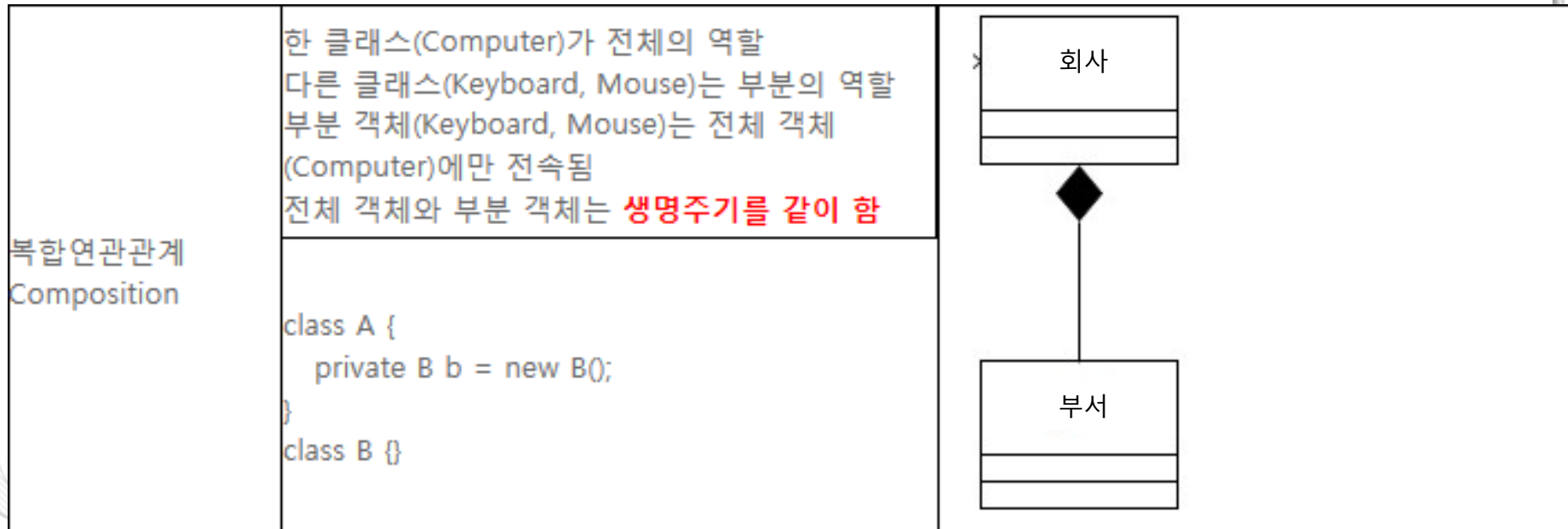
Ex) 썰매 - 루돌프



[2] Relationship

■ Association(연관 관계)

- 연관관계는 대상이 되는 객체의 Life Cycle에 따라서 2가지로 분류가 된다
 - ② Composition (Composition aggregation, 합성 연관 관계)
 - 메인 클래스가 삭제될시 대상 클래스도 같이 삭제가 됨(라이프 사이클이 동일)
 - 분리가 되면 의미가 없어짐
 - 강한 결합
 - has-A 관계



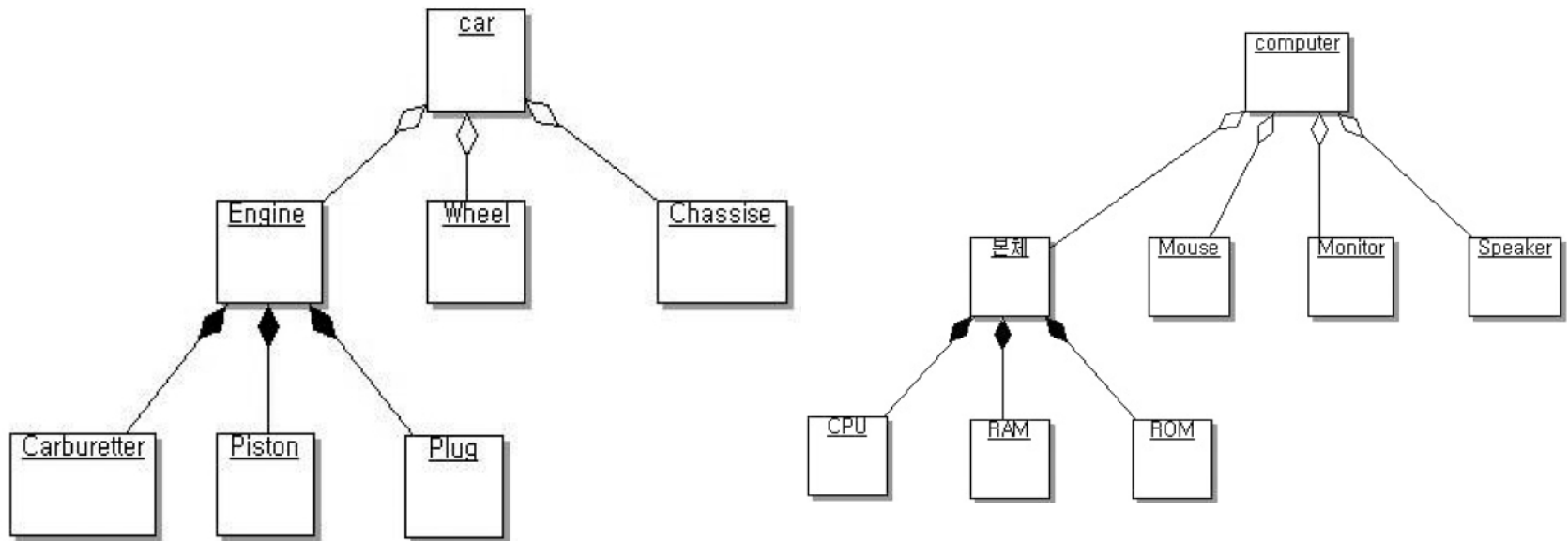
[2] Relationship

■ Association(연관 관계)

연관: 객체들이 서로 개념적으로 연결됨을 의미

집합: 독립적인 구성요소로 연관됨을 의미

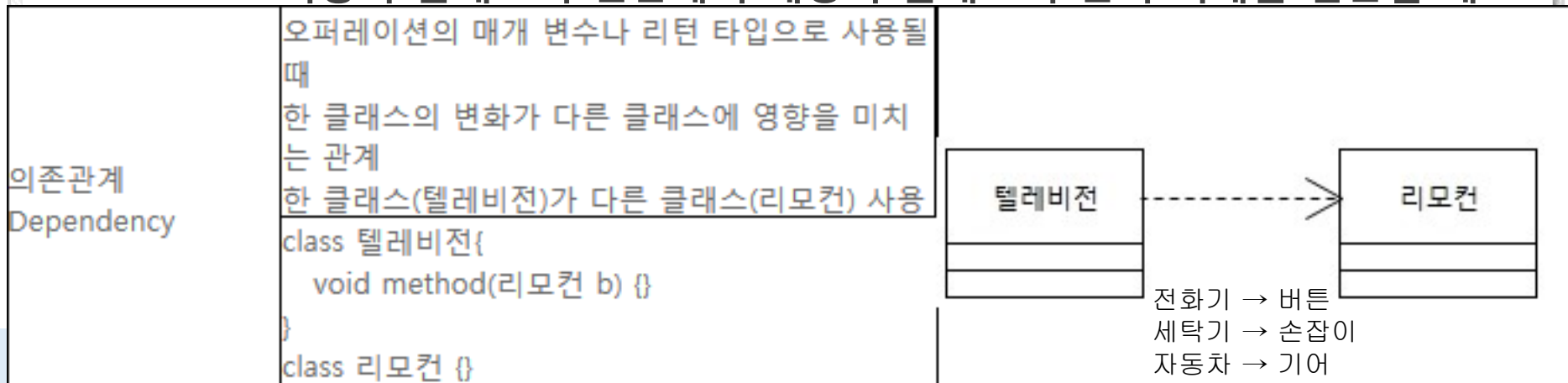
복합: 영구적이며 강한 연관관계로 구성됨



[2] Relationship

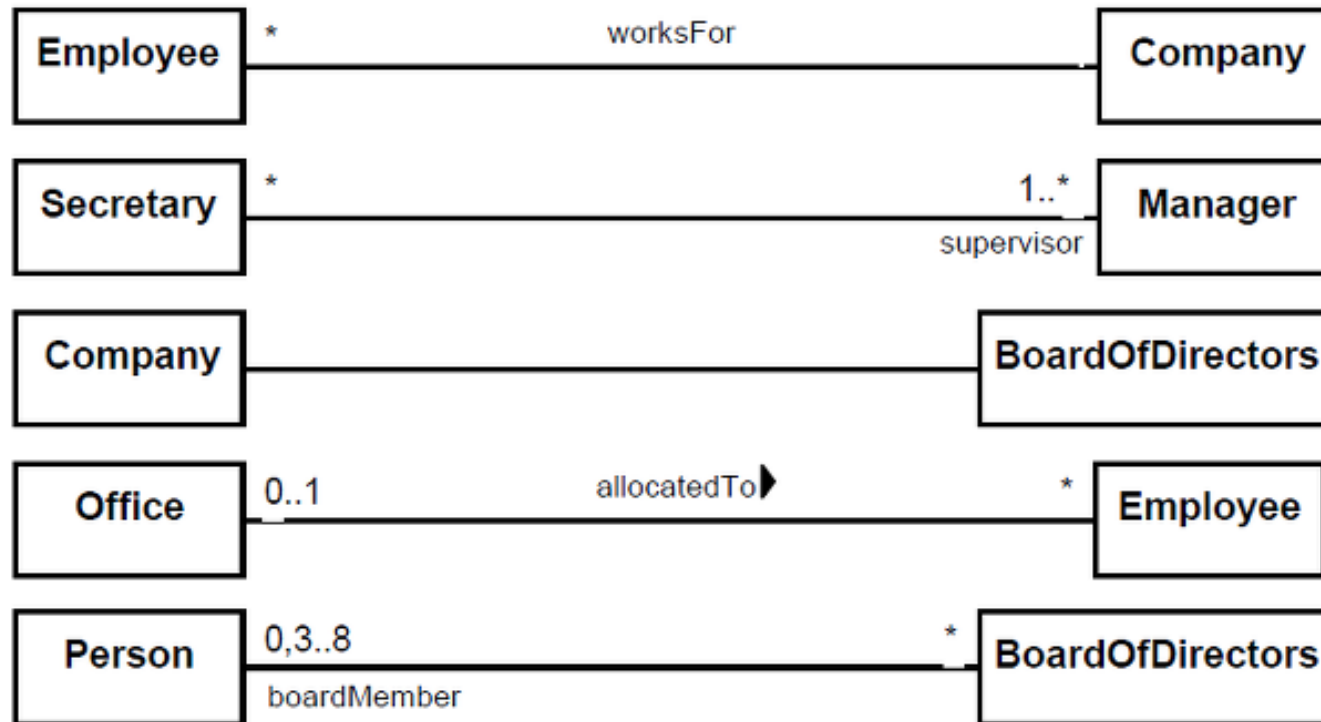
▪ Dependency(의존관계)

- 하나의 모델 요소가 다른 모델 요소를 사용하는 관계 (단순사용관계)
- 클래스 관계 중 가장 약한 결합이며 일시적인 사용을 표현
- 점선으로 된 화살표 모양
- 화살표 방향은 사용하는 쪽에서 사용되는 쪽
- Ex) 프로그래머 -----> 컴퓨터
 - 이용자 클래스의 연산의 인자 타입으로 제공자가 사용될 때 (메소드 파라미터)
 - 이용자 클래스의 연산에서 제공자 클래스의 객체를 생성할 때
 - 이용자 클래스의 연산에서 제공자 클래스의 전역 객체를 접근할 때



[2] Relationship

▪ Multiplicity(다중성)의 이해



1. 여러명(*는 $0 \sim \infty$)의 직원은 회사에서 일을 한다(Many to one).
2. 여러명의 비서는 최소 1명 이상(1..*은 $1 \sim \infty$)의 매니저를 상관으로 가진다. 매니저는 비서가 있을 수도 있고 없을 수도 있다($0 \sim \infty$).
3. 하나의 회사는 하나의 이사회를 가진다.
4. 직원에게 할당된 사무실은 없거나 1개($0 \sim 1$)가 있다. 여러명의 직원(*)이 한 사무실을 사용할 수도 있다.
5. 이사의회의 멤버는 없거나, 있다면 3명이상 8명 이하여야 한다. 멤버는 여러 이사회에 속할 수 있다.

[2] Relationship

■ Summary

Generalization (일반화)	
Realization (실체화)	
Dependency (의존)	
Association (연관)	
Directed Association (직접연관)	
Aggregation (집합, 집합연관)	
	
Composition (합성, 복합연관)	
	

Class diagram을 작성하는 목적

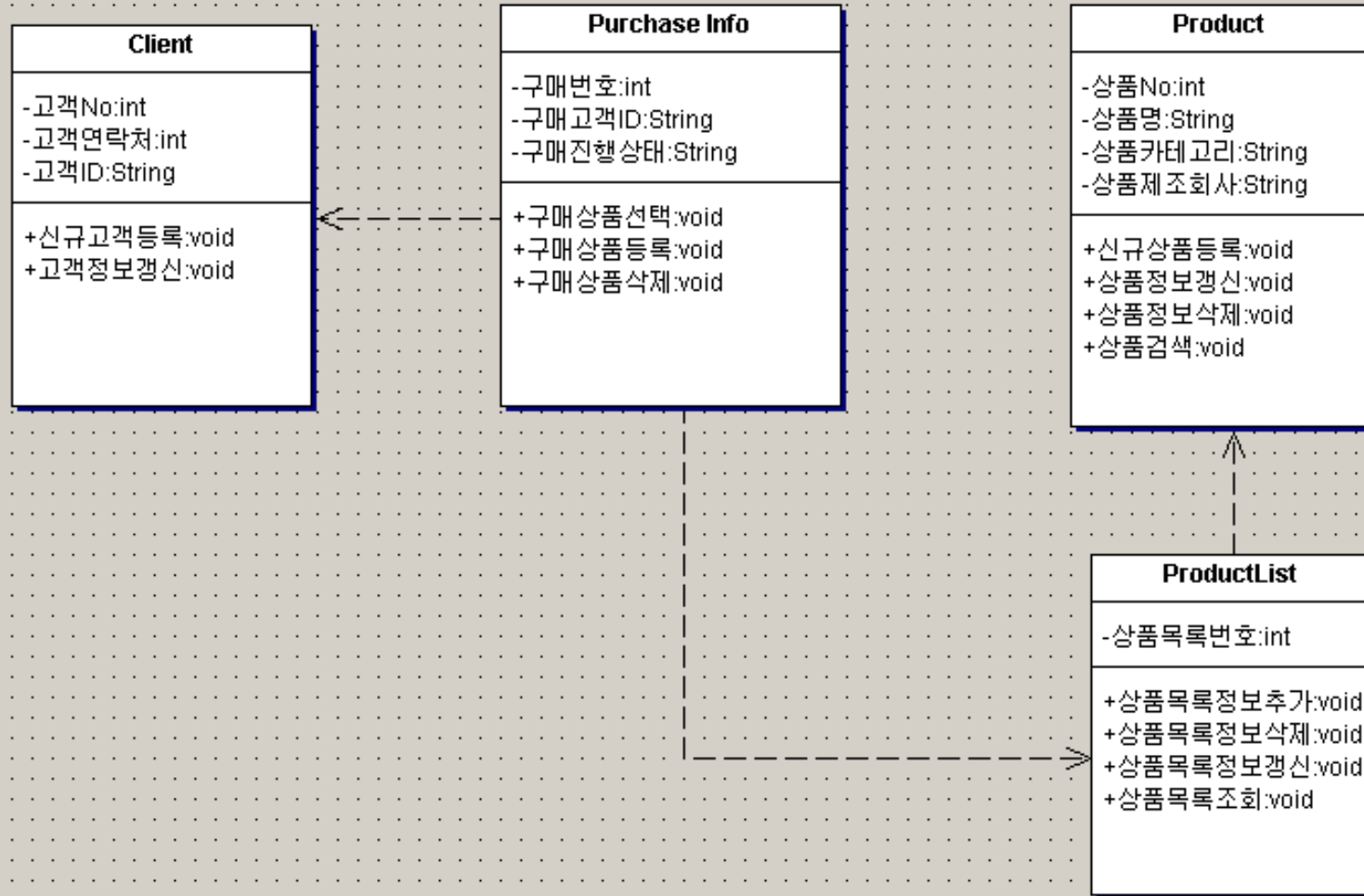
- 시스템을 구성하는 요소를 문서화
- 클래스 사이의 연관, 일반화, 집합 관계를 표시
- 클래스의 기능, 특히 속성과 오퍼레이션을 나타냄
- 문제 영역의 클래스 명세로부터 구현을 위한 자세한 설계까지 시스템의 클래스 구조를 나타냄
- 시스템의 클래스들이 클래스 라이브러리와 어떻게 협력하는지를 나타냄
- 클래스들의 인터페이스를 나타냄
- 시스템 안에 어떤 객체가 존재할 수 있는지를 나타냄

실습 예제 [1]

■ 다음 Use case description에서 Class diagram을 작성하세요.

- Use case name : 상품선택
- 행위자 : 고객
- Use case 개요 및 설명 : 고객은 쇼핑몰에서 상품을 선택한다.
- 사전조건 : 고객은 고객 확인을 받은 상태이다.
- 이벤트 흐름
 - 정상흐름
 - ① 고객은 쇼핑몰에서 컴퓨터를 구입하기 위해 컴퓨터 상품을 선택한다.
 - ② 시스템은 컴퓨터에 관련된 목록을 보여준다.
 - ③ 고객은 목록 중에서 키보드 상품을 선택한다.
 - ④ 시스템은 키보드 상품들을 보여준다.
 - ⑤ 고객은 마음에 드는 하나의 상품을 선택한다.
 - ⑥ 시스템은 사용자의 장바구니에 상품을 등록하고 다른 키보드 상품을 보여준다.
 - ⑦ 고객은 구매하기 위해 구매 버튼을 누른다.
 - 대치흐름
 - 해당사항 없음
 - 예외흐름
 - 해당사항 없음

실습 예제 (1)



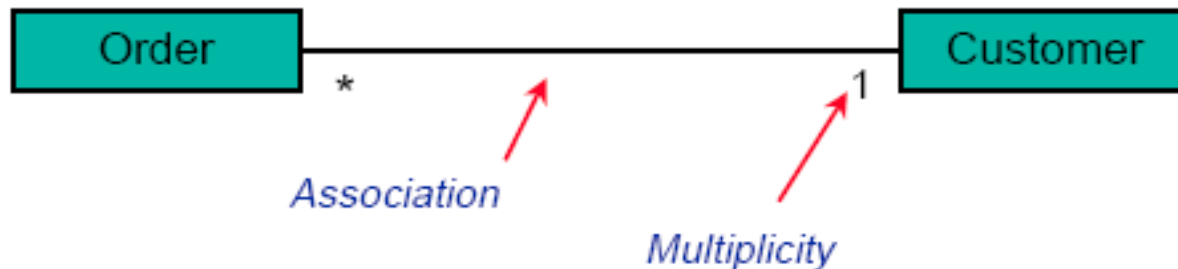
실습 예제 [2]

■ 다음 요구사항을 검토하여 Class diagram을 작성하세요.

- Order scenario:

- We have customers who order our products.
(우리에게는 우리 제품들을 주문하는 고객들 있다.)
- We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders.
(기업고객들과 개인고객들을 구분하고 있다. 기업고객들은 매달 대금청구서를 발송하고 있는 반면 개인고객은 주문 전에 주문 이전에 선납을 해야 하기 때문이다.)
- We want our orders to be lined up product by product.
(우리는 주문이 제품별로 정렬되기를 바란다)
- Each line should contain the amount and the price of each product.
(각 주문라인은 각 제품의 가격, 수량을 포함되어야 한다)

Example: Order - Associations



Order:

We have customers who may order several products.

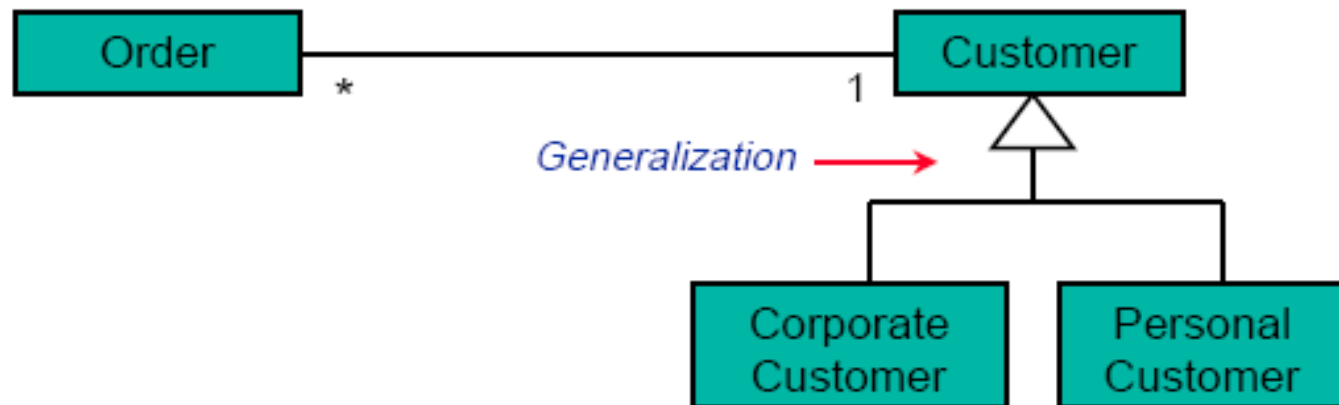
We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders with a credit card.

We want our orders to be lined up product by product.

Each line should contain the amount and the price of each product.

실습 예제 (2)

Example: Order - Generalization



Order:

We have customers who order our products.

We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders with a credit card.

We want our orders to be lined up product by product.

Each line should contain the amount and the price of each product.

실습 예제 [2]

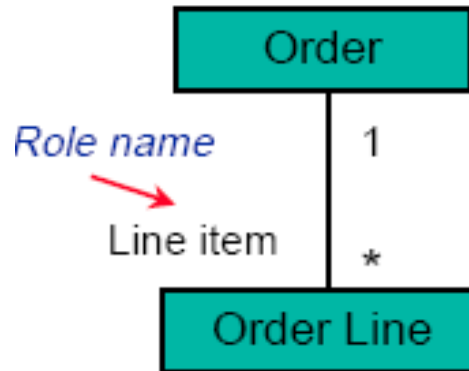
Order:

We want our orders to be **lined up product by product**.

→ 각 Order에 대해 제품별 주문라인 (Order line)

즉, Order는 다수의 Order line과 관련.

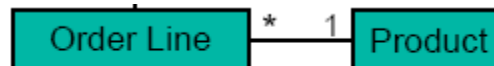
Order line 하나의 Order와 관련



Each line should contain the amount and the price of each product

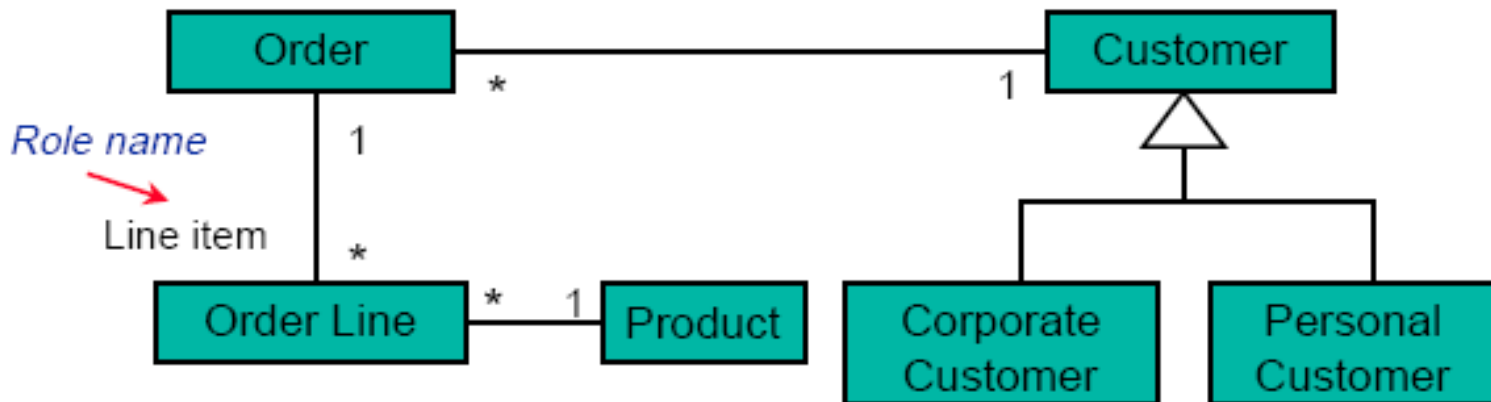
→ 하나의 Order line에 한 종류씩의 product 를 포함하며
각 product의 수량 및 가격을 포함.

→ 하나의 product 는 다수의 Order line에 포함 될 수 있다.



실습 예제 (2)

Example: Order - More Associations



Order:

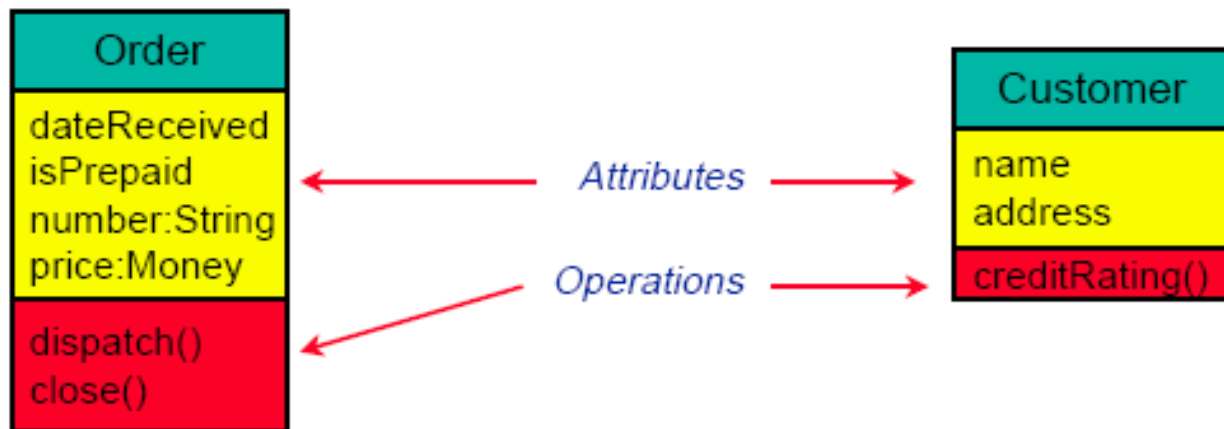
We have customers who order our products.

We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders with a credit card.

⇒ We want our orders to be lined up product by product.

Each line should contain the amount and the price of each product.

Example: Order- Attributes & Operations



Order:

We have customers who order our products.

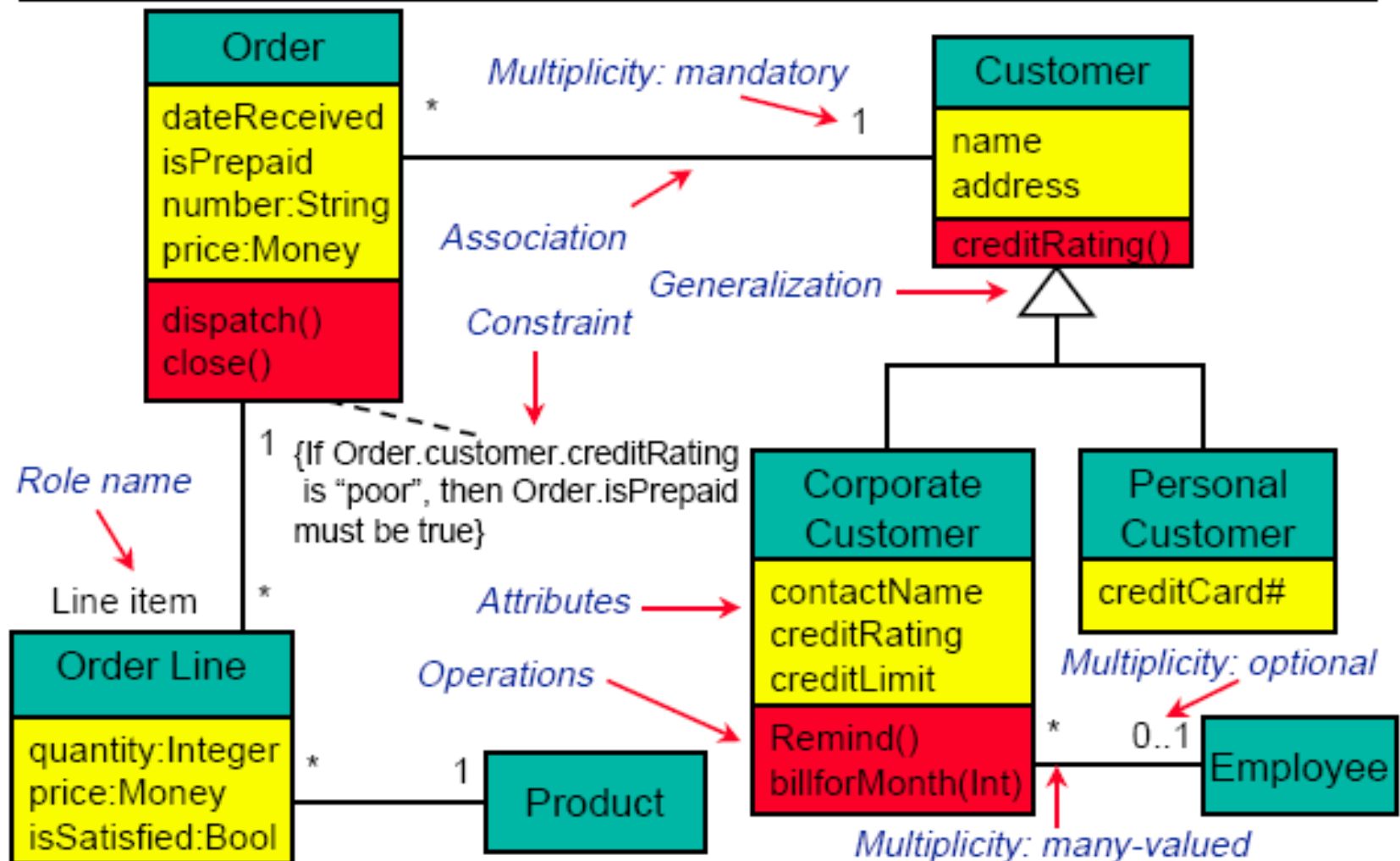
We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders with a credit card.

We want our orders to be lined up product by product.

Each line should contain the amount and the price of each product.

실습 예제 (2)

Example: Order - Full Class Diagram



실습 예제 (3)

▪ Class diagram 작성 요령 (반복적인 과정)

- 개념적 모델링 - 간단히 중요한 클래스의 존재와 관계만 표현
- 명세적 모델링 - 구현에 필요한 자료구조, UI, 데이터베이스, 통신에 필요한 클래스 포함

▪ 클래스 찾기

- “인터넷 서점 시스템은 고객이 시스템에 로그인하여 상호작용하면서 서점에서 판매하는 책을 찾아보고 구매한다. 구매자가 구매이력을 확인하기 위하여 시스템은 구매기록을 보관하고 보안을 제공할 필요가 있다. 고객은 구매한 책을 신용카드나 온라인 송금 등 여러 가지 방법으로 결제할 수 있다.
- 고객은 구매하고 싶은 책을 저자 색인으로 찾을 수 있고 책을 구매하고 읽은 사람들의 리뷰를 볼 수 있다. 시스템은 고객이 구매한 이력을 바탕으로 고객의 취향을 찾아내어 보관한다.”

Class 식별

■ 클래스로 적당하지 않는 것 배제

- 중복 클래스 - 고객, 구매자
- 관계 없는 클래스 - 시스템 구축 비용
- 불확실한 클래스 - 보안

■ 인터넷 서점의 클래스 후보

Order	Book	Customer	Review	Author
Account	OrderHistory	CreditCard	Payment	On-line
ShoppingCart	Stock	Catalog	Shipping	Preference

Association 찾기

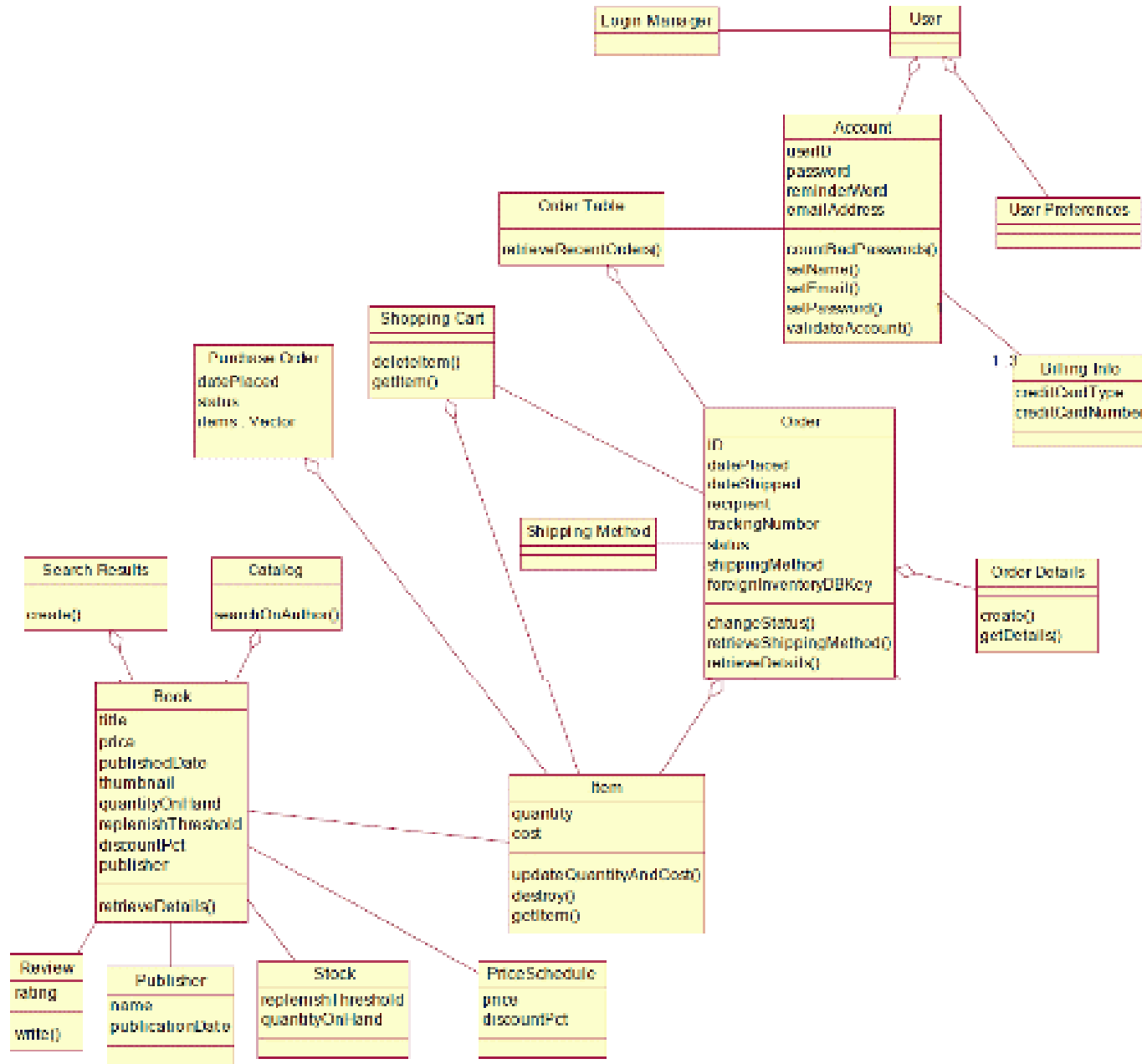
■ 문제 정의에 있는 것

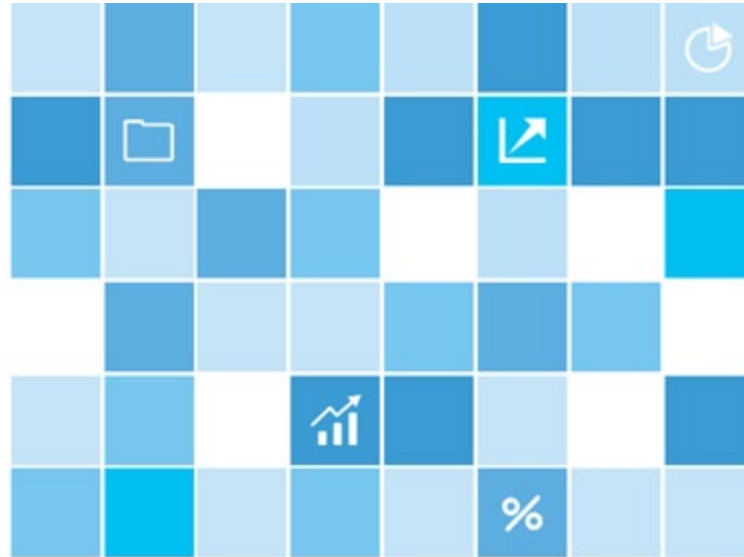
- 고객은 책을 구매한다.
- 사용자는 로그인 한다.
- 사용자는 구매이력을 확인한다.
- 주문을 위하여 결제한다.
- 결제에는 신용카드와 온라인 송금이 있다.
- 책을 저자 색인으로 찾는다.
- 고객은 취향을 가진다.

■ 문제 정의에 없는 것

- 사용자는 계정을 갖는다.
- 주문은 주문 상세로 구성된다.
- 검색결과는 책으로 구성된다.
- 출판사는 책을 출간한다.

속성 찾기 및 클래스 다이어그램 그리기





Thank You