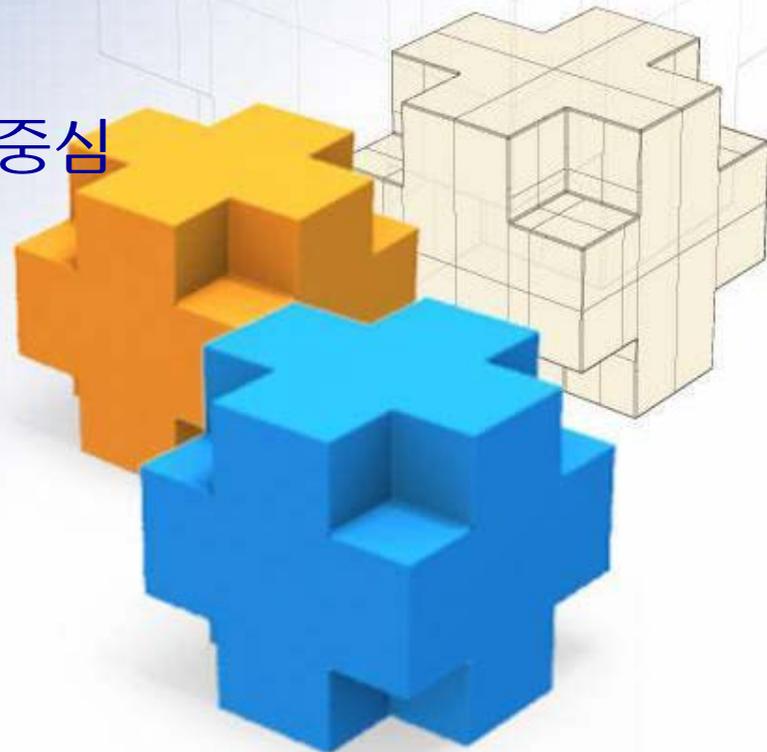


LECTURE

6

UML 클래스 다이어그램

-클래스 다이어그램은 UML의 중심



목 차

- UML이란?
- UML의 역사
- UML 클래스 다이어그램
- Exercise – 클래스 다이어그램 설계



UML

- 분석, 설계를 비주얼 화, 문서화 하기 위한 그래픽 언어
- Unified
 - 이전의 OO 방법들의 통합
- Modeling
 - 객체지향 분석 설계를 위한 비주얼 모델링
- Language
 - 모형화된 지식(의미)을 표현



UML은 이다.

- 시스템에 대한 지식을 찾고 표현하기 위한 언어
- 시스템을 개발하기 위한 탐구 도구
- 비주얼 모델링 도구
- 근거가 잘 정리된 가이드라인
- 분석, 설계 작업의 마일스톤
- 실용적 표준

UML은 이 아니다.

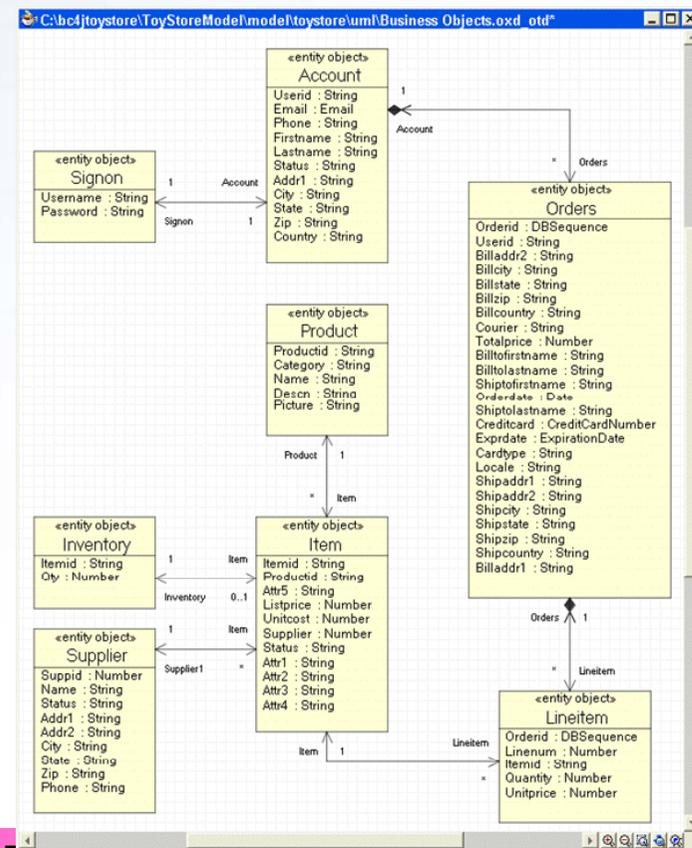
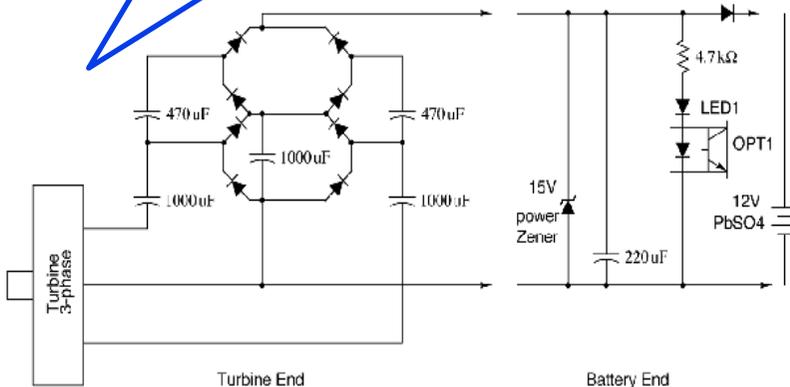
- 비주얼 프로그래밍 언어 환경
- 데이터베이스 표현 도구
- 개발 프로세스(SDLC)
- 모든 문제의 해결책
- 품질 보증 방안

UML



Every s/w engineer
WILL understand UML
diagrams.

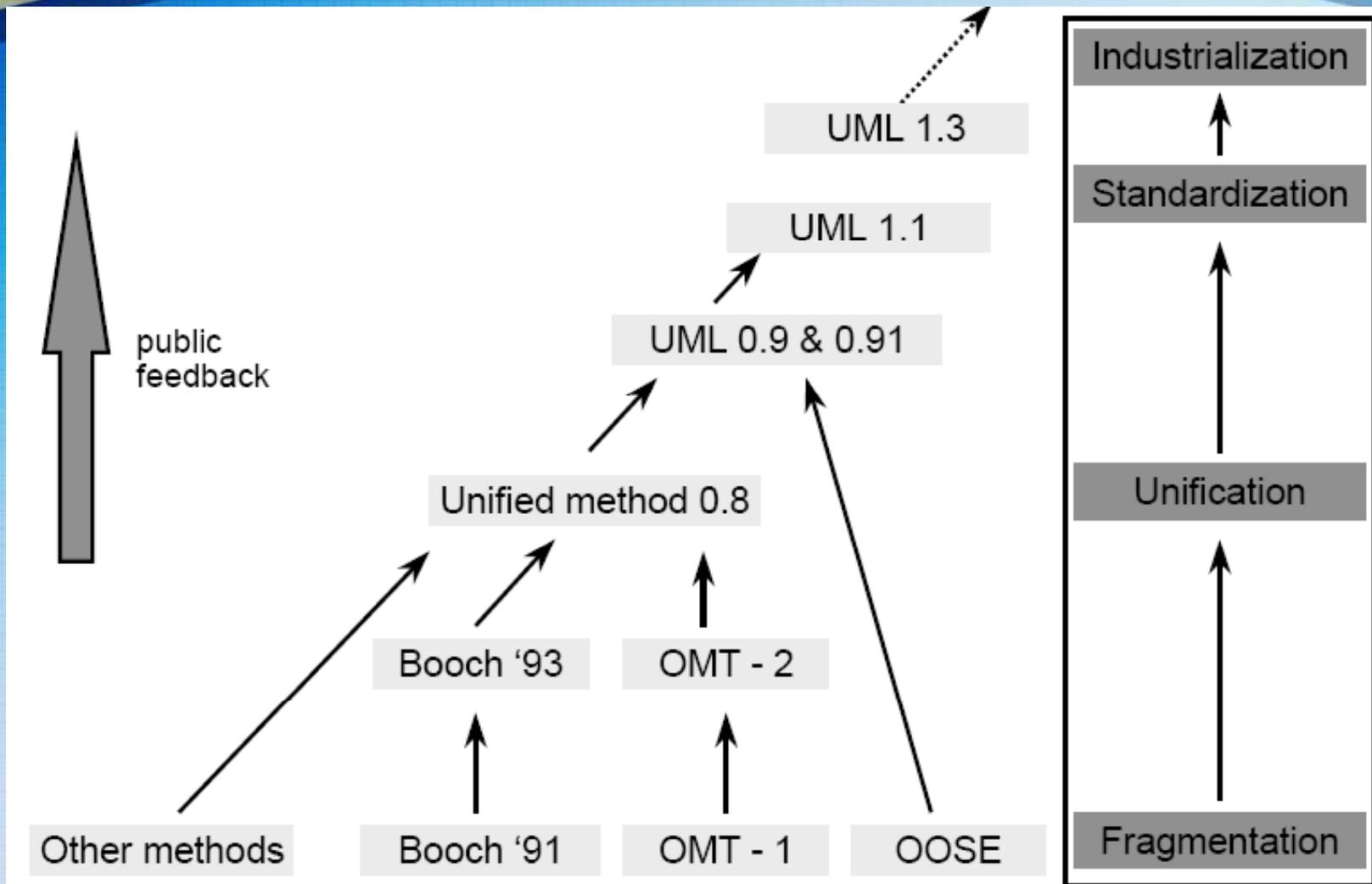
Every h/w engineer
understands curcuit
diagram.



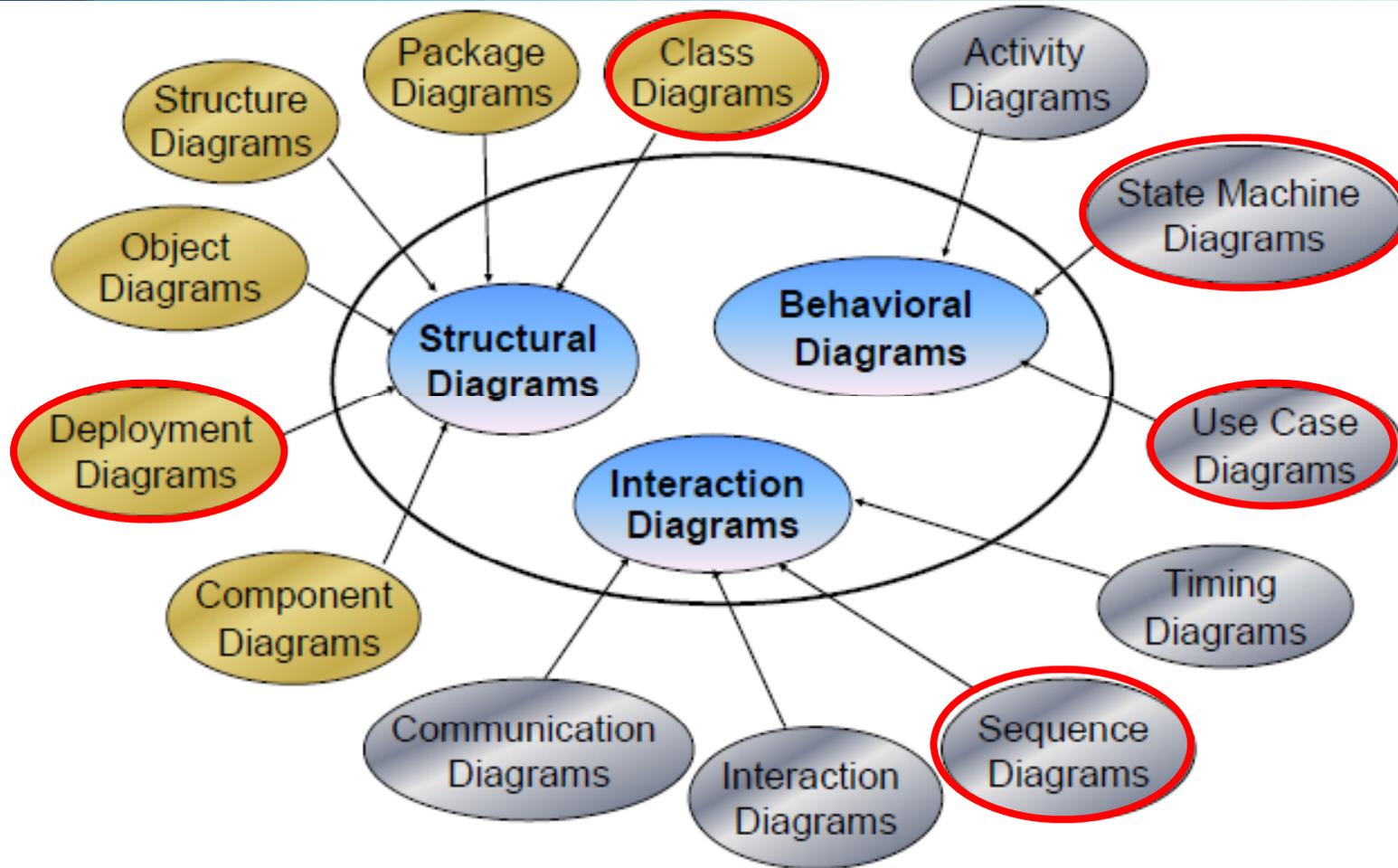
UML의 역사

연도	방법 (버전)	비고
'89~'94	춘추전국	20개 이상의 객체지향 분석 설계 방법 (OMT, Booch, Fusion, Jacobson, Yourdon)
1994	UM 0.8	OMT+Booch 방법
1996. 6	UML 0.9	OMT+Booch+OOSE (Jacobson), Feedback
1997. 1	UML 1.0	여러 업체의 참여, OMG에 제출
1997. 9	UML 1.1	OMG 릴리스 (새로 참여한 업체의 의견 반영)
2001. 4	UML 1.4	오류 수정
2003.	UML 2.0	Superstructure (notation, semantic) Diagram interchange
2007. 9	sysML	시스템 엔지니어링으로 확장

UML의 진화



UML 2.0 다이어그램 체계



UML 클래스 다이어그램

□ UML 클래스 다이어그램

객체지향 시스템에 존재하는 클래스, 클래스 안의 필드, 메소드,
서로 협력하거나 상속하는 클래스 사이의 연결 관계를 나타내
는 그림

나타내지 않는 것

- 클래스가 서로 어떻게 상호작용 하는지
- 자세한 알고리즘
- 특정한 동작이 어떻게 구현되는지

클래스 나타내기

□ 박스 위에 클래스 이름

- 추상 클래스는 이탤릭체
- 인터페이스 클래스는 <<interface>> 추가

□ 속성

- 객체가 가지는 모든 필드를 포함

□ 오퍼레이션/메소드

- 아주 흔한 메소드(get/set)는 생략
- 상속된 메소드도 포함할 필요 없음

Rectangle
- width: int
- height: int
/ area: double
+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double

Student
-name:String
-id:int
<u>-totalStudents:int</u>
#getID()int
+getName():String
~getEmailAdress()String
+getTotalStudents()int

클래스 속성

□ 속성(필드, 인스턴스 변수)

○ **visibility** name: **type**[count] = *default value*

○ visibility: + public
 # protected
 - private
 ~ package(디폴트)
 / derived

○ Underline static variable

○ 파생된 속성: 저장되지 않고 다른 속성값으로부터 계산됨

Rectangle
- width: int
- height: int
/ area: double
+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double

Student
-name:String
-id:int
<u>-totalStudents:int</u>
#getID()int
+getName():String
~getEmailAdress()String
<u>+getTotalStudents()int</u>

클래스 오퍼레이션/메소드

□ 오퍼레이션/메소드

○ **visibility** name(parameters) : *return_type*

○ visibility: + public
protected
- private
~ package(디폴트)

○ Underline static method

○ 파라메타 타입 (name: type)

○ 생성자나 리턴 타입이 void인 경우는 ***return_type*** 생략

Rectangle
- width: int - height: int / area: double
+ Rectangle(width: int, height: int) + distance(r: Rectangle): double

Student
-name:String -id:int <u>-totalStudents:int</u>
#getID():int +getName():String ~getEmailAdress():String <u>+getTotalStudents():int</u>

클래스 사이의 관계

□ 일반화(generalization): 상속(isa) 관계

- 클래스 사이의 상속
- 인터페이스 구현

□ 연관(association): 사용(usage) 관계(3 종류)

- 의존
- 집합(aggregation): 어떤 클래스가 다른 클래스의 모임으로 구성
- 합성(composition): 포함된 클래스가 컨테이너 클래스가 없이는 존재할 수 없는 집합관계의 변형

일반화 관계

□ 일반화(상속)

- 부모를 향한 화살표로 표시되는 하향 계층 관계
- 선/화살표는 부모 클래스의 종류에 따라 다름

❖ 클래스:

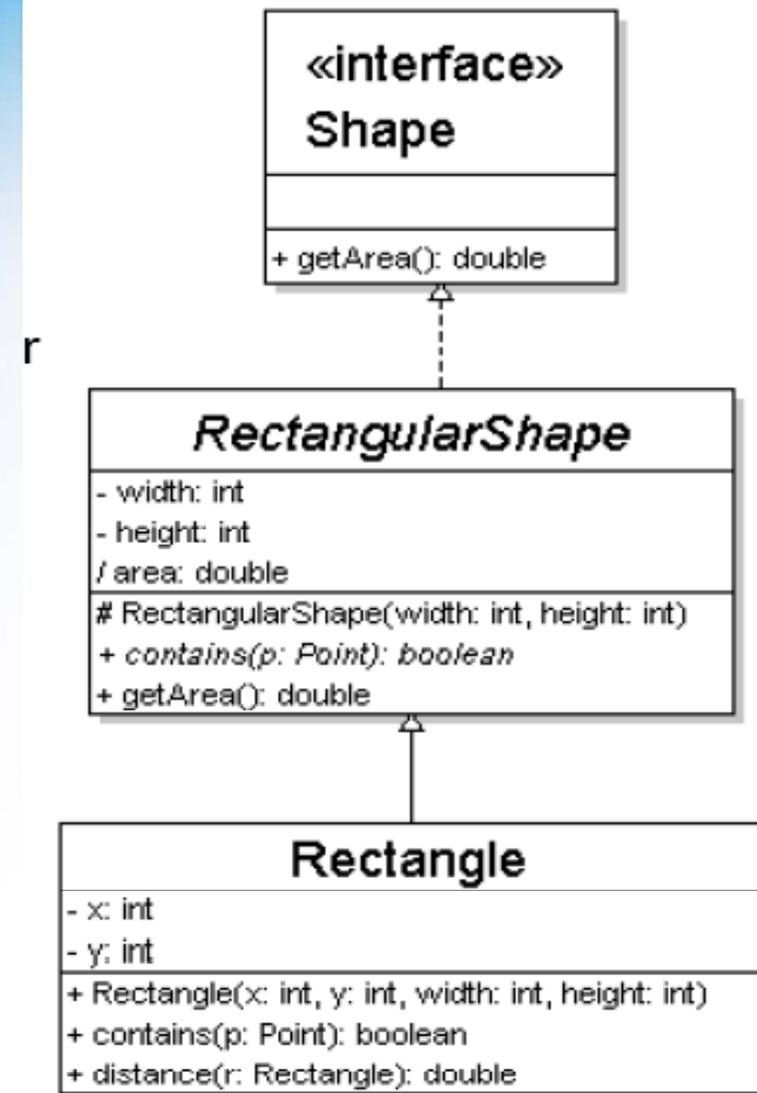
실선/검은 헤드 화살표

❖ 추상 클래스:

실선/흰 헤드 화살표

❖ 인터페이스:

점선/흰 헤드 화살표



연관 관계

연관(association): 어떤 클래스의 인스턴스가 작업을 수행하기 위하여 다른 클래스를 알아야 하는 함

1. 다중도(multiplicity)

- * ⇔ 0, 1, or more
- 1 ⇔ 정확히 1개
- 2..4 ⇔ 2개 내지 4개
- 3..* ⇔ 3개 이상

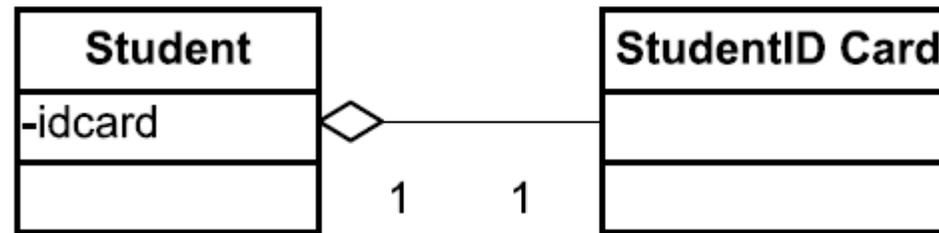
2. 이름 - 객체들의 관계 이름

3. 방향성(navigability) - 질의의 방향, 객체 사이의 선으로 표시하며 양쪽 방향인 경우는 화살표시 없음

연관 관계의 다중도

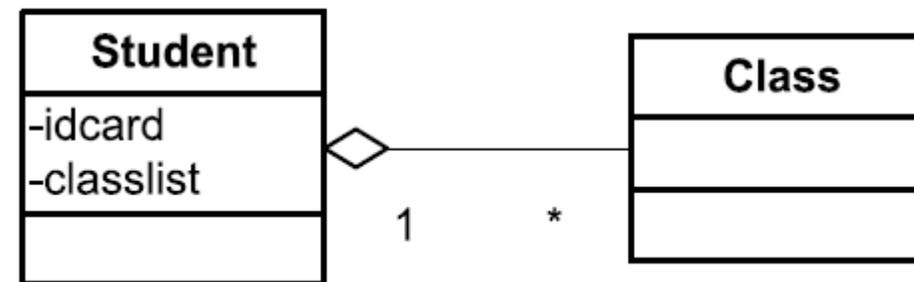
□ 1 대 1

- 학생 1명이 학생증(id card) 한 개만을 가진다.



□ 1 대 다

- 학생 1명이 여러 클래스를 수강할 수 있다.



Example

□ square-list 라는 클래스를 추가하고 square 클래스와 연관시키시오.

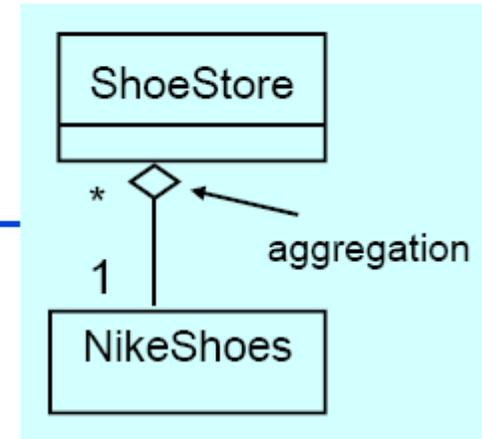
□ 연관 양 끝에 다중도를 표시하라.

- * ⇨ 0, 1, or more
- 1 ⇨ 정확히 1개
- 2..4 ⇨ 2개 내지 4개
- 3.. * ⇨ 3개 이상

연관의 타입

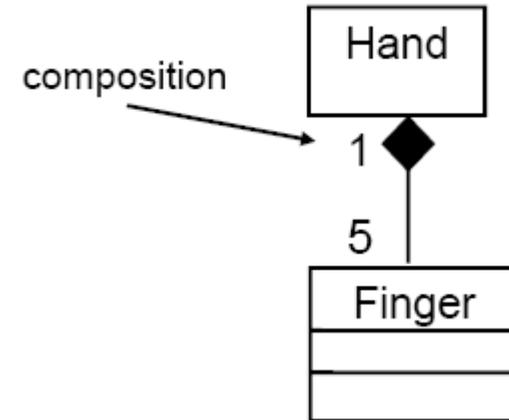
❑ 집합(aggregation): “contains”

- 포함하고 있는 클래스 쪽에 하얀 다이아몬드 표시



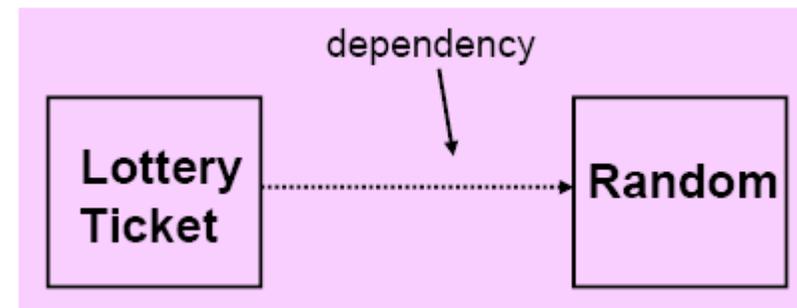
❑ 합성(composition): “이 목적을 위하여만 포함됨”

- 집합보다 더 끈끈한 관계
- 부분은 전체가 살고 죽느냐에 좌우 됨
- 포함하고 있는 클래스 쪽에 검은 다이아몬드로 표시

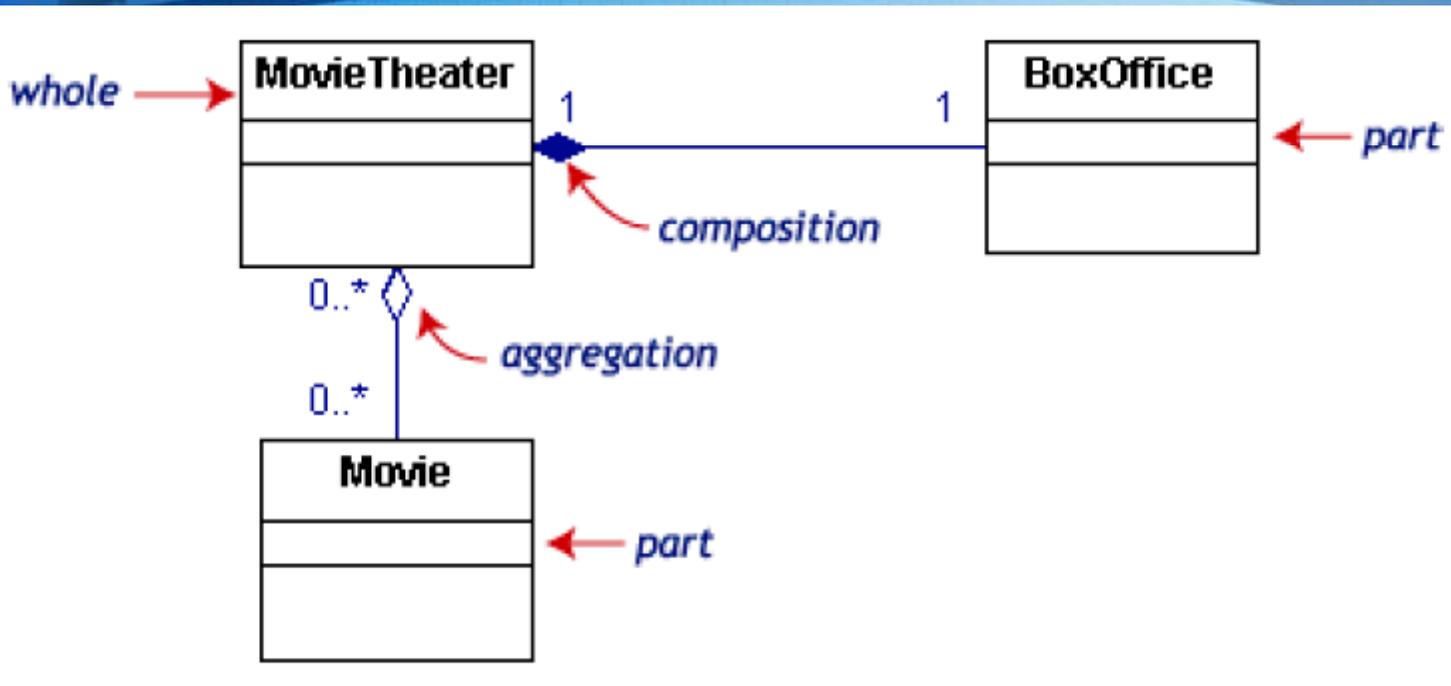


❑ 의존(dependency): “일시적 사용”

- 점선으로 표시



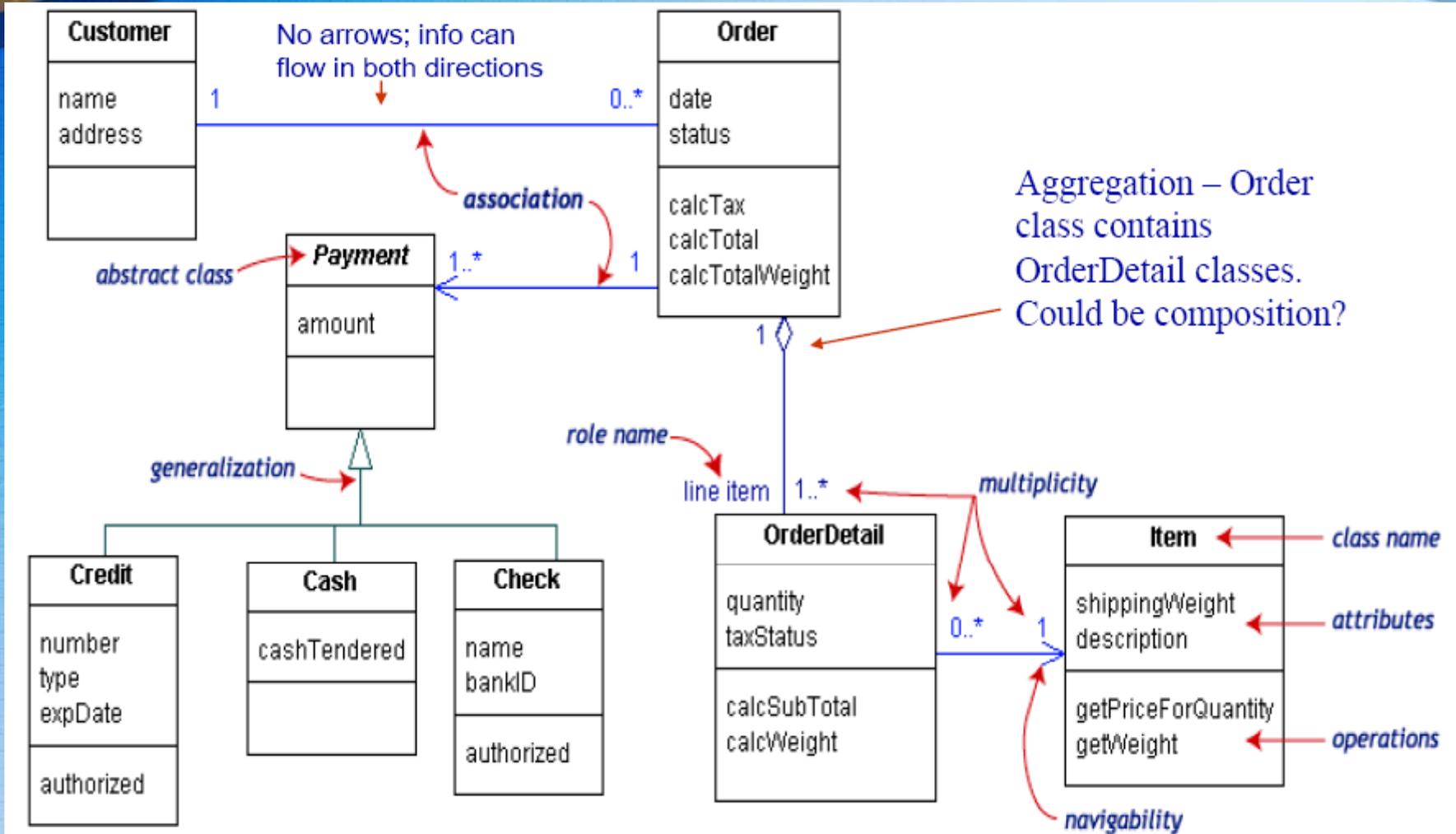
합성/집합 관계의 예



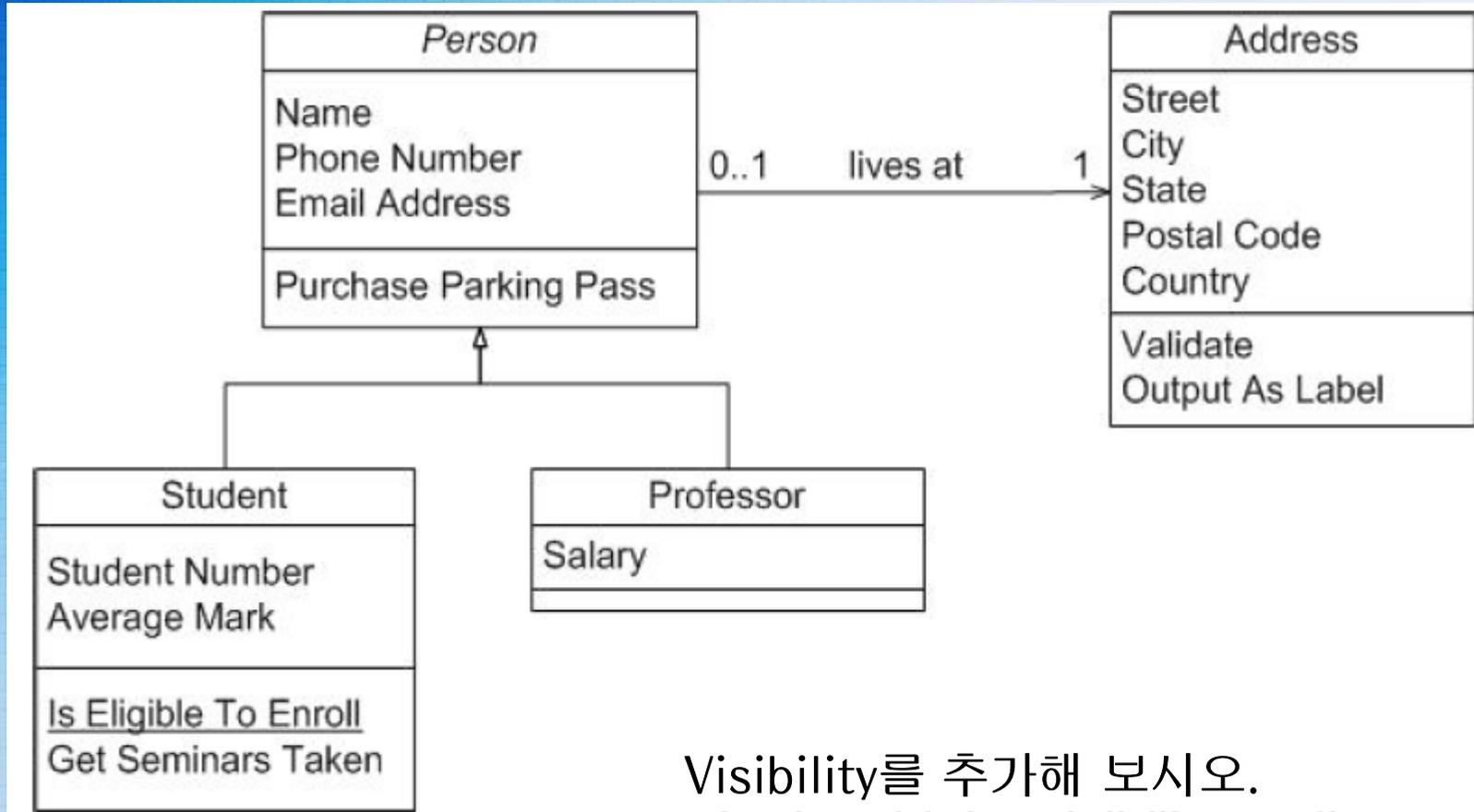
○ 영화관이 없어지면

- 매표소도 없어짐 ⇒ 합성
- 그러나 영화는 아직 존재 ⇒ 집합

클래스 다이어그램 예

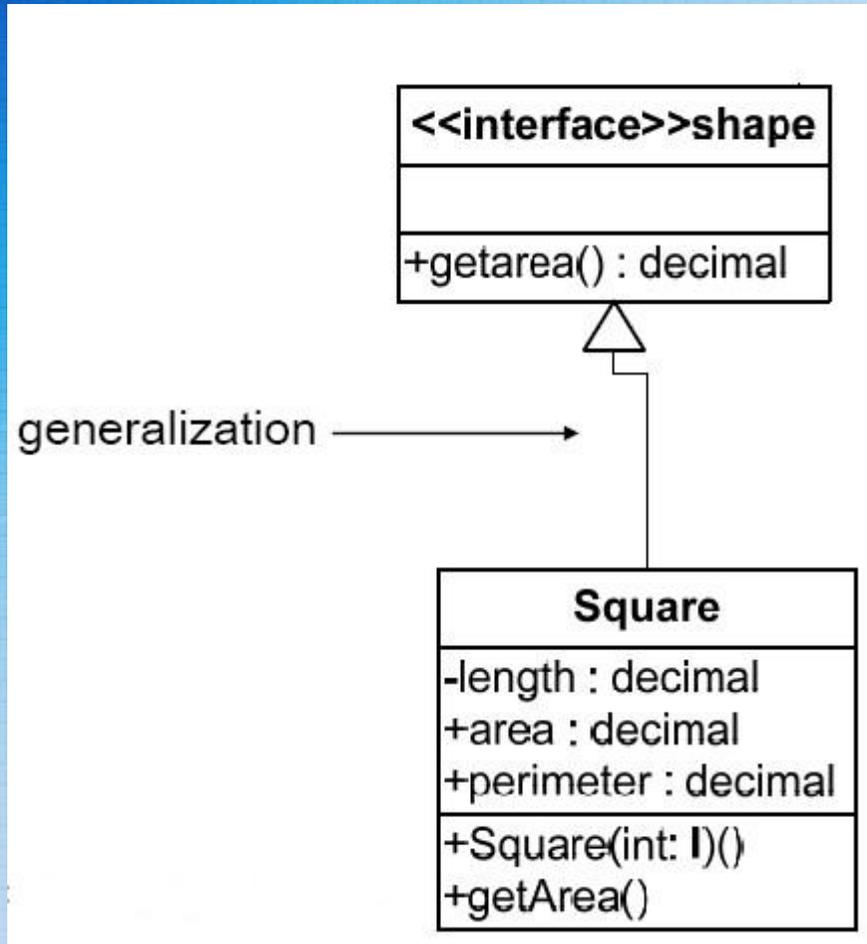


클래스 다이어그램 예 #2



Visibility를 추가해 보시오.

UML Square

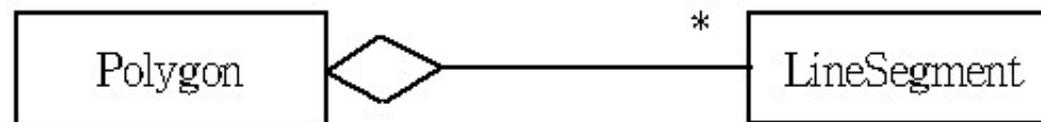


□ Visibility에 오류가 있는 것은?

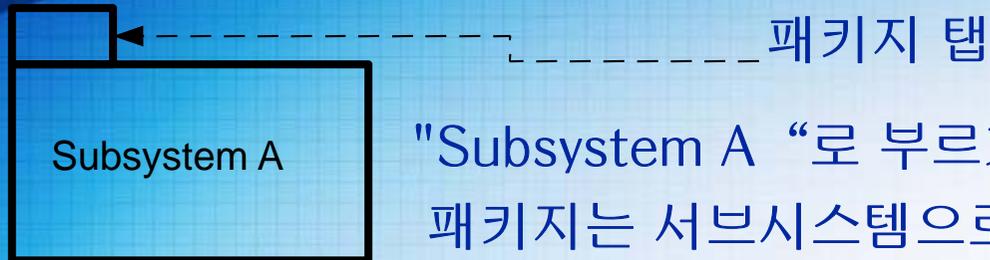
□ 생성자 타입의 오류?

전파(propagation) 현상

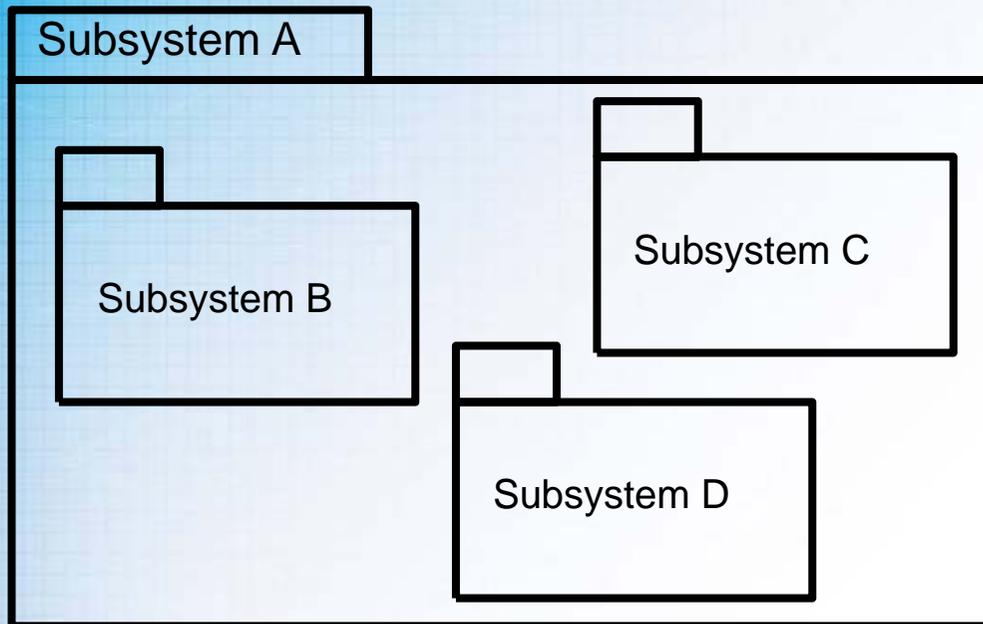
- 전체 개념의 오퍼레이션이 부분 개념의 오퍼레이션에 의하여 구현되는 현상
- 동시에 부품의 속성이 전체 개념에 전파되는 현상
- 전파(propagation)과 전체부분 개념의 관계는 상속과 일반화 관계와 유사
 - 중요한 차이는
 - 상속은 묵시적인 메커니즘
 - 전파는 필요할 때 프로그램



패키지 다이어그램

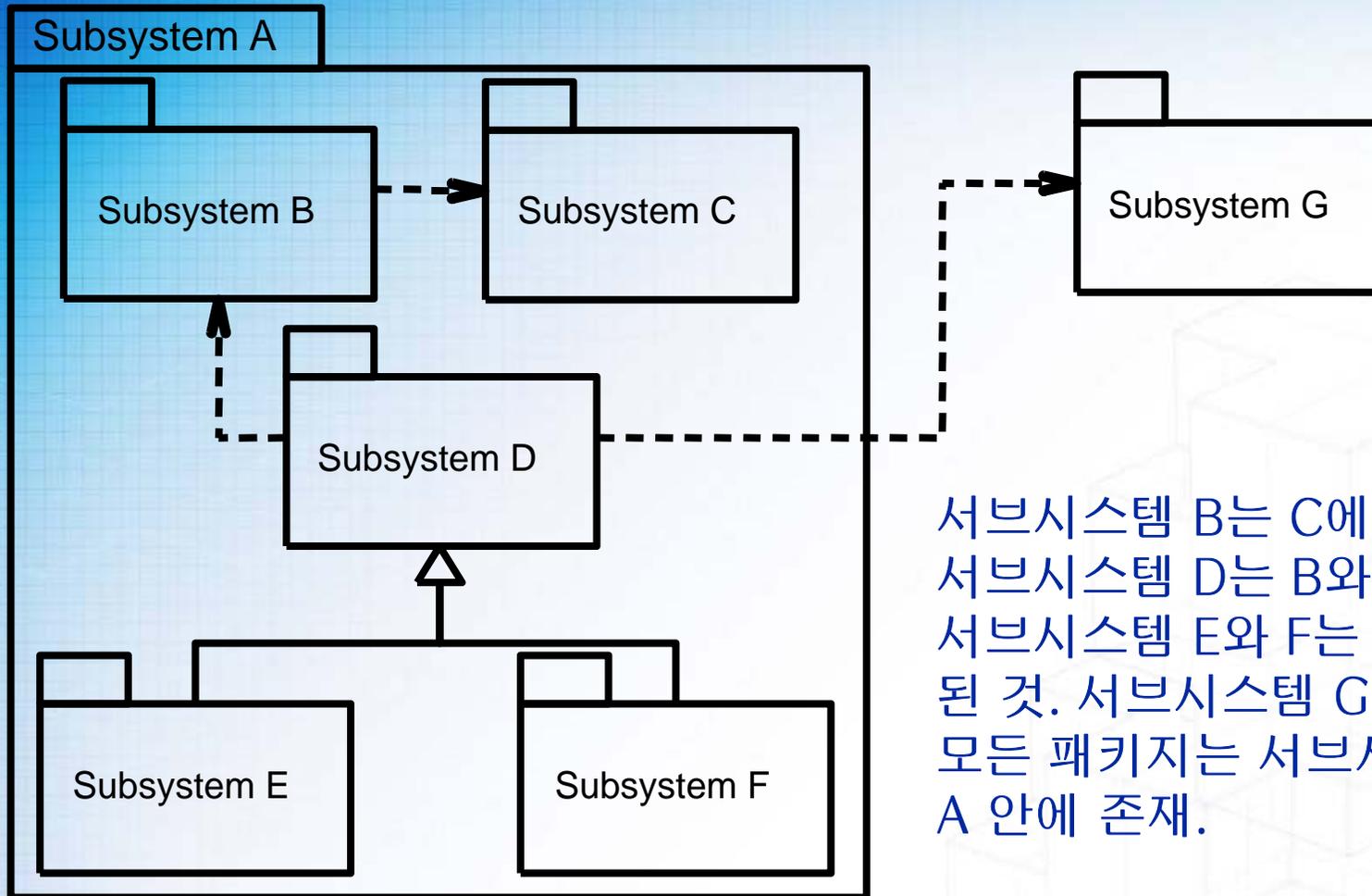


"Subsystem A"로 부르기로 한 패키지의 외부관점
패키지는 서브시스템으로 부름



“서브 시스템 A는 세 개의 다른 패키지 “서브시스템 B, C, D” 를 그룹핑, 확장된 패키지의 이름은 패키지 탭에 표기.

패키지 사이의 관계



서브시스템 B는 C에 의존한다.
서브시스템 D는 B와 G에 의존
서브시스템 E와 F는 D의 상세화
된 것. 서브시스템 G를 제외한
모든 패키지는 서브시스템
A 안에 존재.

설계 연습 문제 #1

□ 항공권 예약 문제

- 예약의 기록은 항상 탑승객 한 명 단위로 이루어짐. 탑승객이 없는 예약은 없음
- 예약에 탑승객이 여러 명인 경우는 없음
- 탑승객이 다수의 예약을 할 수 있음
- 탑승객이 예약이 하나도 없을 수 있음

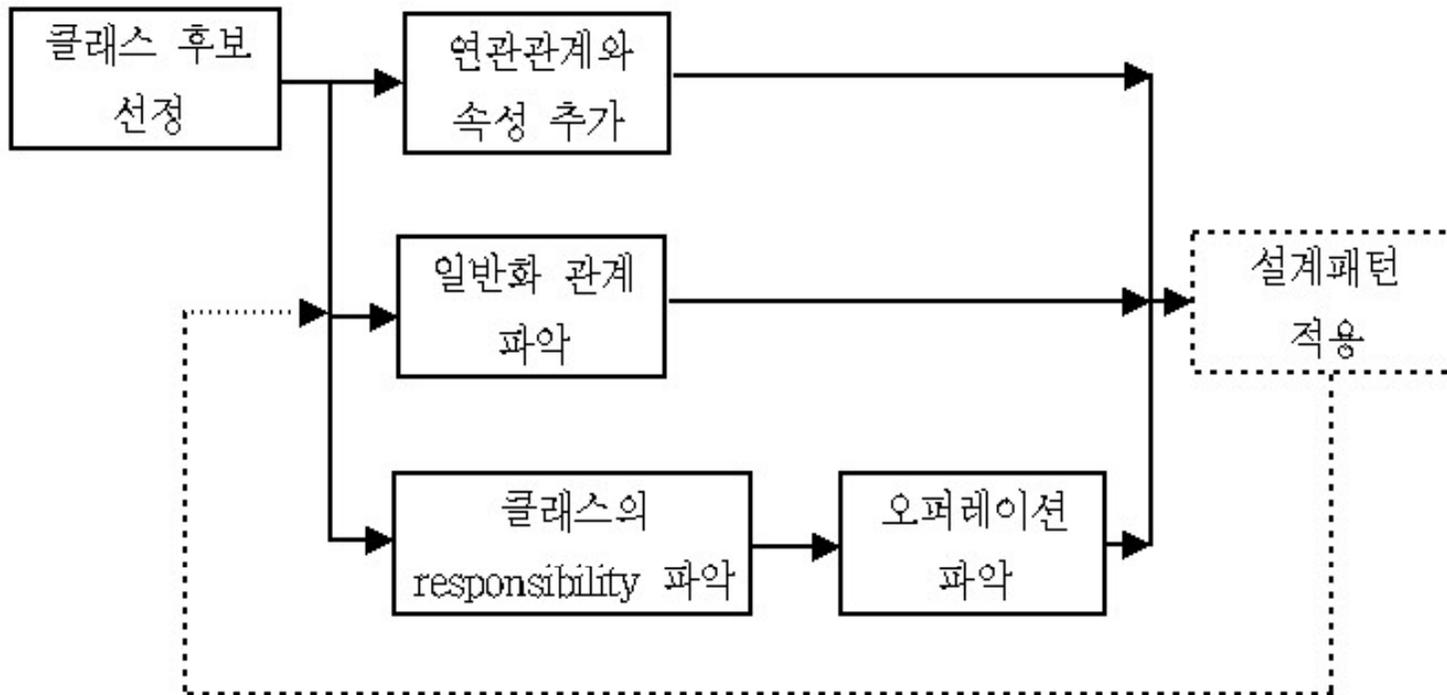
설계 연습 문제 #2

- 자동판매기를 객체지향으로 개발하기로 하였다. 자판기 안에 있는 여러 컴포넌트들의 관계를 나타내는 UML 클래스 다이어그램을 완성하시오.
 - 자판기에는 동전을 일정 시간 넣지 않으면 자동으로 동전을 내뱉기 위하여 클락이 내장되어 있다.
 - 음료수 선택을 위한 버튼
 - 동전 슬롯
 - Shelf 센서와 배출구

클래스 다이어그램 작성 과정

□ 반복, 점증적 방법

- 초벌로 작성 후 계속 추가, 삭제



참고 문헌

- Practical UML: A hands on introduction for developers
<http://dn.codegear.com/article/31863>

- UML에 관련된 많은 서적과 웹 튜토리얼이 있음
“UML Distilled” , by Martin Fowler.

