

소프트웨어 공학 (Software Engineering)

**모델링
(Modeling)**



In this chapter ... (1/3)

모델링 (Modeling)

- 왜 모델링 작업이 필요한가?
- 객체, 클래스, 캡슐화, 상속 다형성, 메시지 패싱이란 무엇인가?
- UML을 이용한 기초적인 모델링 방법은?
- 정적, 동적 모델링 방법과 차이는?





In this chapter ... (2/3)

모델링 (Modeling)

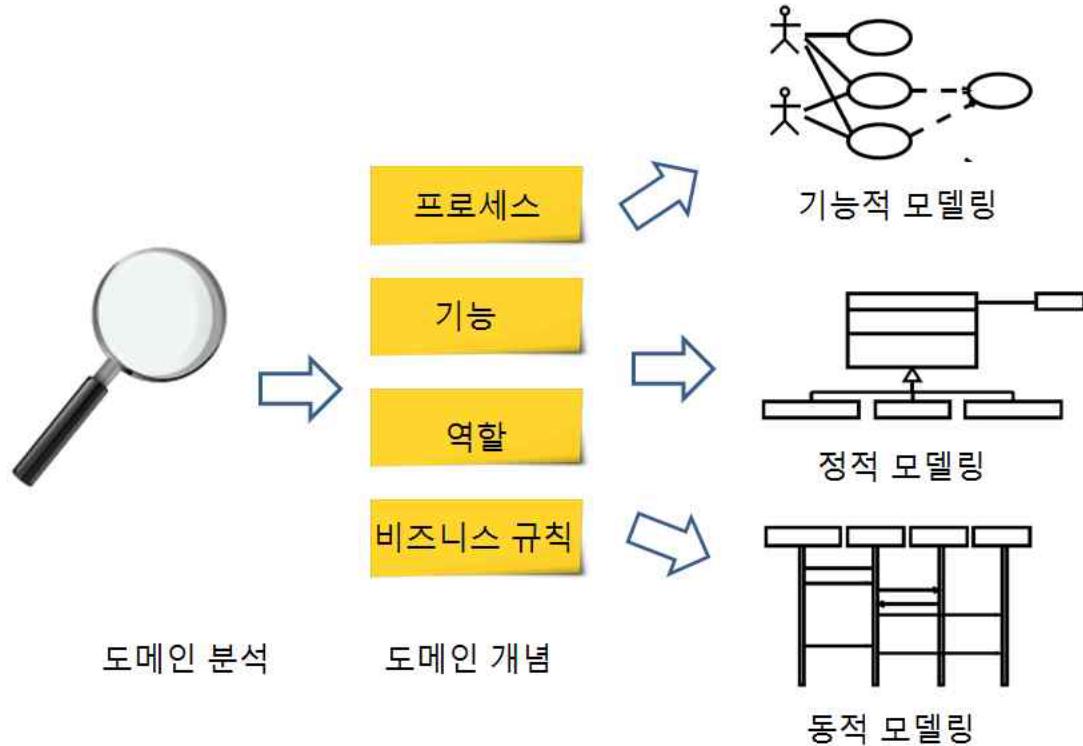


모델링

- 도메인 지식을 체계화하는 과정
- 중요한 도메인 개념과 특성, 관계를 파악하여 다이어그램으로 정형화
- 과거에는 자료 위주의 모델링, 현재는 객체지향 패러다임에서의 모델링
→ UML(Unified Modeling Language)를 사용하여 표현



모델링 과정의 도식화





In this chapter ... (3/3)

모델링 (Modeling)

모델링이 개발자에게 주는 도움

- 도메인 지식을 체계화하는 과정
- 응용문제를 이해하는 데 도움을 줌
- 개발팀원들 사이에 응용문제의 공통 개념으로 대화하게 하고 개선시킴
- 파악한 개념을 사용자와 고객에게 전달 할 때 도움을 줌
- 후속 작업 즉 설계, 구현, 테스팅, 유지보수에 개념적인 기준을 제공



In this chapter ...

모델링 (Modeling)



5.1 객체지향 개념



5.2 UML



5.3 정적 모델링



5.4 동적 모델링



5.5 모델링 도구



▣ 절차적인 방법은 조그만 변화에도 영향이 큼

- 최근에는 객체지향 설계, 언어가 매우 일반화된 추세임
- 함수와 자료를 함께 클래스로 묶음으로 여러 장점을 얻을 수 있음
(예제: 스크롤러 – 여러 환경에 적용되며, 독립적인 수정/관리가 가능)

▣ 객체지향의 장점

- 개발자가 설계를 작성하고 이해하기 쉬움
- 자료와 함수를 함께 추상화 함으로써 변화에 영향을 적게 받음
- 사용자 중심, 대화식 프로그램의 개발에 적합
- 프로그램을 뚜렷하게 구별되는 단위(object)로 분할 가능



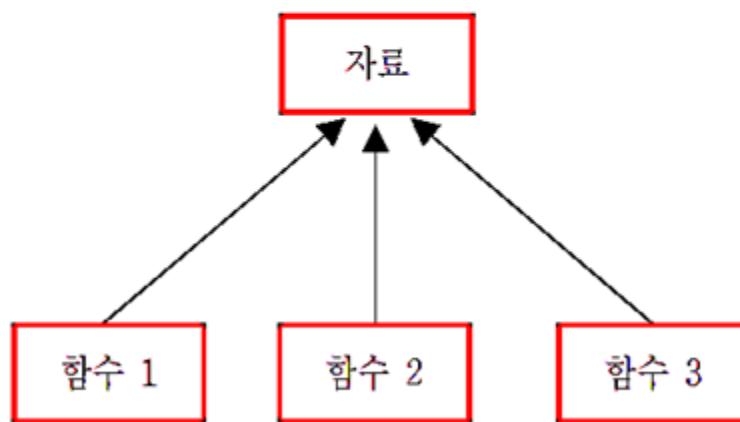
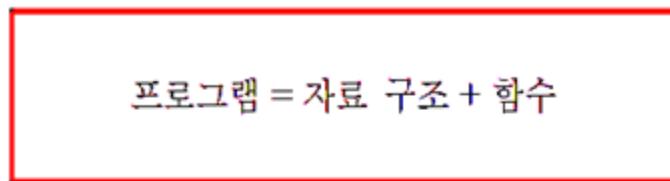


절차적 방법 vs 객체지향 방법

모델링 (Modeling)

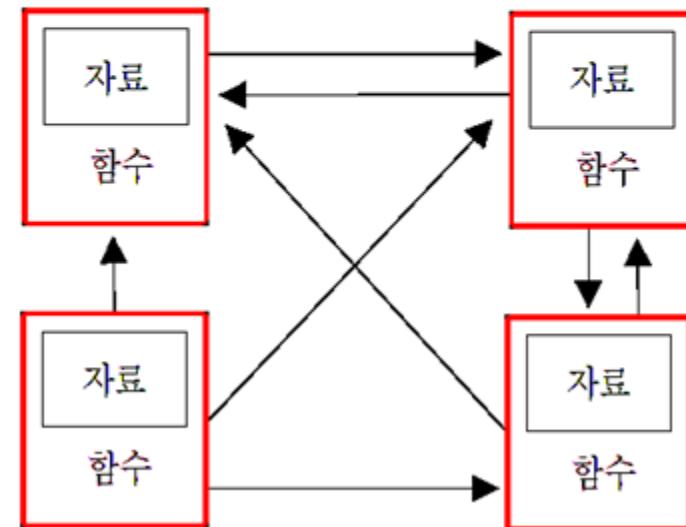
- 절차적 방법: 소프트웨어는 데이터 구조와 이를 이용한 함수들의 집합
- 객체지향 방법: 소프트웨어는 여러 객체의 모임이며, 객체는 데이터와 관련 함수를 모아 놓은 것

절차적 방법



객체 지향 방법

객체 = 자료 구조 + 함수
프로그램 = 객체 + 객체



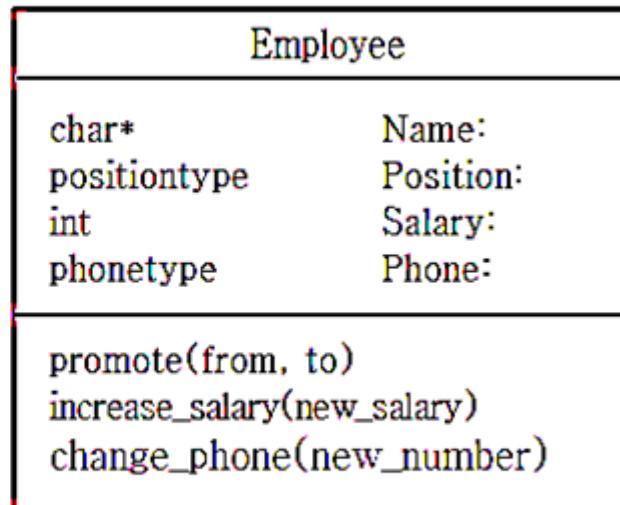


▣ **클래스**: 속성과 오퍼레이션을 캡슐화

▣ **객체**: 클래스의 인스턴스

▣ 클래스 개념을 사용한 그룹핑 예제

```
class Employee {  
public:  
    promote(from, to);  
    increase_salary(new_salary);  
    change_phone(new_number);  
    ...  
private:  
    char* name;  
    positiontype position;  
    int salary;  
    phonetype phone;  
    ...  
}
```





- **객체**: 속성과 오퍼레이션을 가진 애플리케이션의 독립된 존재
- **속성**: 객체의 특징을 결정

■ 객체의 구조

- 소프트웨어 모듈(객체) = 자료구조 + 함수

■ 객체는 상태(state), 능력(behavior), 정체성(identity)을 가짐

- 상태: 현재 속성들이 가진 값이 상태를 결정지음
- 능력: 연산(operation)을 수행할 수 있는 능력
- 정체성: 다른 객체로부터 구별되는 식별가능성



캡슐화 (Encapsulation)

모델링 (Modeling)



캡슐화의 정의

- 속성과 관련된 오퍼레이션을 클래스 안에 묶어서 하나로 취급하는 것
- 예제: 대학 학사관리 시스템
 - 데이터: 학번, 이름, 주소를 클래스에 포함시켜 캡슐화 함
 - 함수: 평점 계산, 주소 변경, 수강 신청 등을 클래스에 포함시켜 캡슐화 함



추상화의 수단: 자세한 속성, 오퍼레이션 등은 차후에 생각



정보은닉(information hiding)

- 캡슐 속의 (자세한) 항목에 대한 정보를 외부에 감추는 것
- 외부의 직접적 접근 불가, 일종의 블랙박스 역할을 함
- 객체지향 언어의 public, private 등의 문법으로 캡슐 내 접근을 제어할 수 있음





객체는 일반적으로 상호작용하여 동작함

- 객체끼리 메시지를 보내거나 메소드를 구동 시켜 상호작용할 수 있음
- 상호작용할 필요가 있는지 찾아내는 작업이 필요함



연관

- 하나 또는 그 이상의 클래스와의 관계를 일컬음
- 예제: 은행 시스템과 학사업무 시스템

<u>Own</u>		<u>Teach-Enrolled-by</u>		
Customer	Account	Student	Course	Professor
홍길동	자유저축1	홍길동	자료구조	이금희
홍길동	정기예금1	홍길동	객체지향 설계	박영희
김동국	자유저축2	김동국	객체지향 설계	박영희
이철수	자유저축3	이철수	소프트웨어공학	최은만
한국남	정기예금2	한국남	소프트웨어공학	최은만

- 원쪽: 고객과 계정과의 관계
- 오른쪽: 학생, 교과목, 교수와의 관계



▣ 가시성(visibility): 객체의 접근 가능성

(즉, 객체 A에서 객체 B를 알고 있어야 접근 가능함)

▣ 연관을 맺은 두 객체가 서로를 알게 하고 접근하게 하는 방법

- 방법 1) 연관된 객체(Course)를 전역으로 선언하여 클라이언트 객체(Student)가 접근할 수 있게 함
- 방법 2) 연관된 객체(Course)를 클라이언트 객체(Student)의 메시지 호출 오퍼레이션의 매개변수로 만듦
- 연관된 객체(Course)를 클라이언트 객체(Student)의 일부로 만듦
- 연관된 객체(Course)를 클라이언트 객체(Student)에서 선언함





집합(Set)

모델링 (Modeling)

- 주의: 수학의 집합과는 조금 개념이 다름
- 집합 관계는 전체(whole)와 부분(part) 개념 사이의 관계
- 격납(containment)의 의미를 가짐
- 예제: 디스크 \supset 트랙 \supset 섹터

```
class Disk {  
    private:  
        Track *tracks;  
        disk information  
        ...  
};  
class Track {  
    private:  
        Sector sectors[MAX];  
        ...  
};  
class Sector {  
    private:  
        ...  
};
```



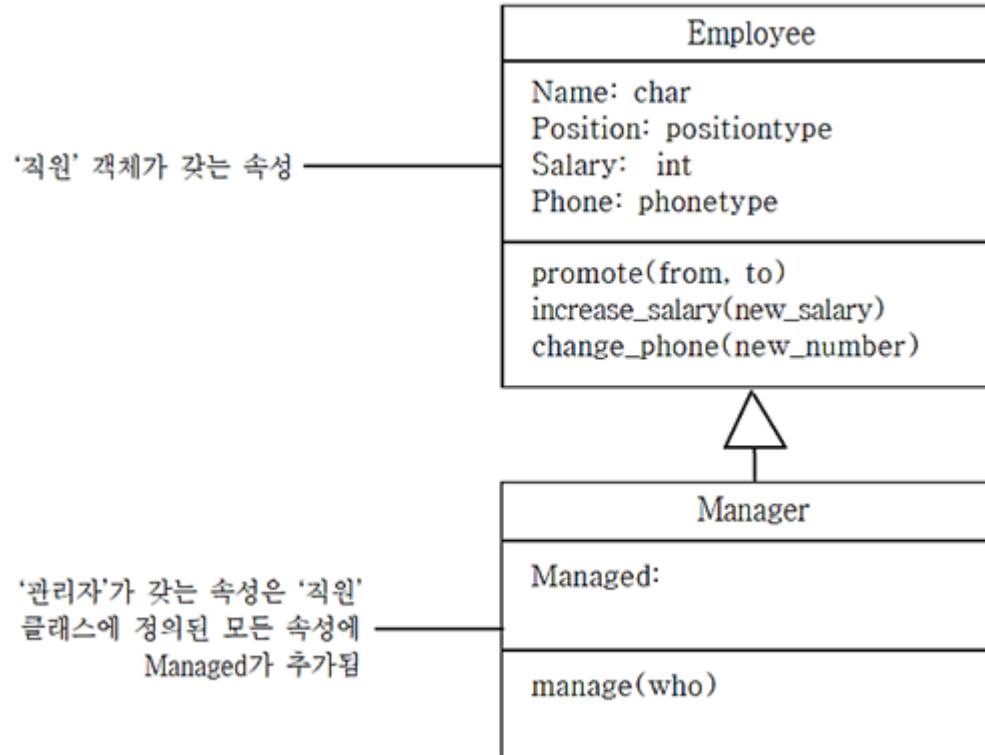
상속(Inheritance) (1/2)

모델링 (Modeling)



상속의 의미

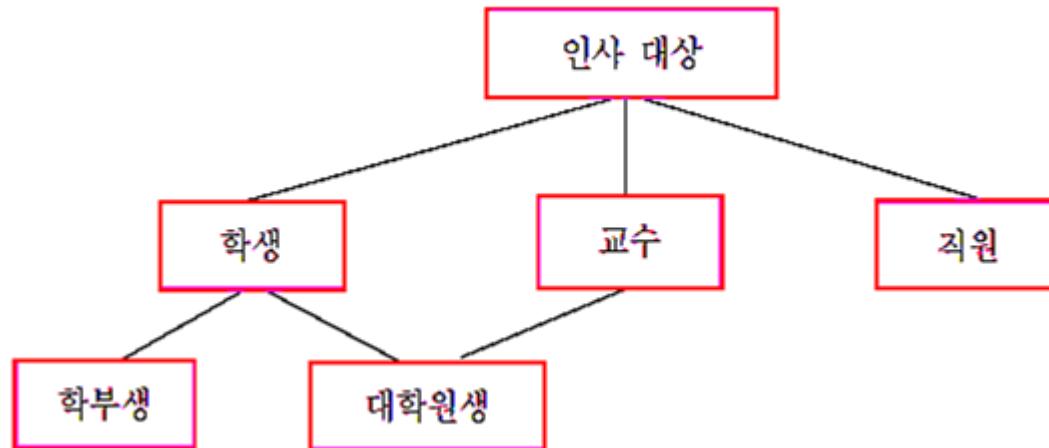
- 한 클래스가 다른 클래스의 일반화된 개념인 경우 성립
- 슈퍼클래스(superclass)를 상속하여 서브클래스(subclass)가 정의됨
- 예제: 직원 – 슈퍼클래스 , 관리자 – 서브클래스





복수 상속(Multiple Inheritance)

- 두 개 이상의 슈퍼클래스에서 상속을 받은 서브클래스
- 복수 상속의 예제: 학생이면서 강사인 대학원생





다형성(Polymorphism) (1/2)

모델링 (Modeling)

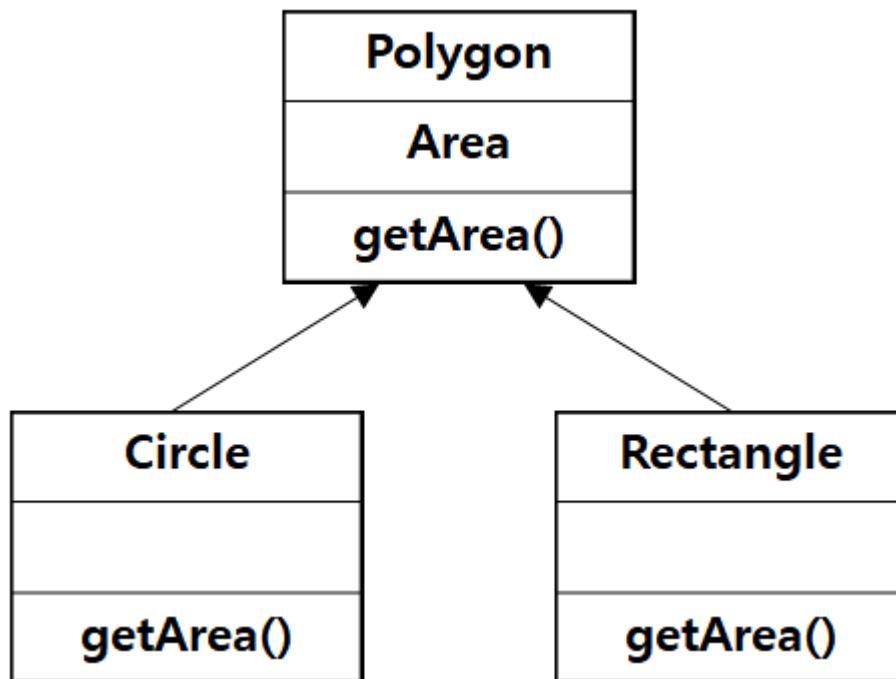
- ❶ 다형성의 정의:
여러 형태를 가지고 있다. (여려 형태를 받아들일 수 있다.)

- ❷ 같은 이름의 메소드를 다른 객체 or 다른 서브클래스에서 호출
 - 예제: `getArea()`에 대해, 도형의 모양에 따라 다른 오퍼레이션이 호출됨

- ❸ 메소드 이름이 같아도, 매개변수나 객체 클래스의 이름에 따라 다른 오퍼레이션이 적용됨



- 현재 코드를 변경하지 않고, 새로운 클래스를 쉽게 추가할 수 있음
- 다형성의 예제



- **getArea()의 호출**
 - `twoDShape.getArea();`



In this chapter ...

모델링 (Modeling)

 5.1 객체지향 개념

 5.2 UML

 5.3 정적 모델링

 5.4 동적 모델링

 5.5 모델링 도구

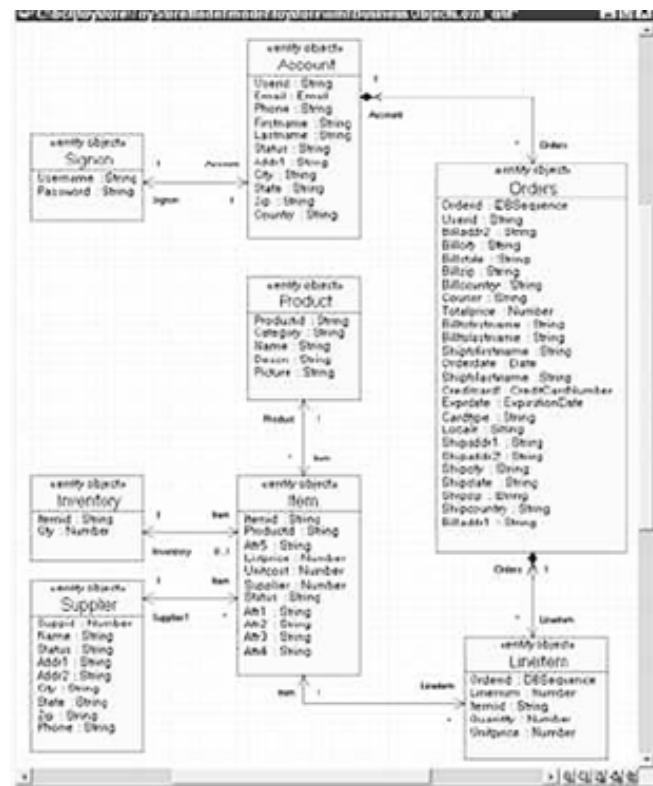
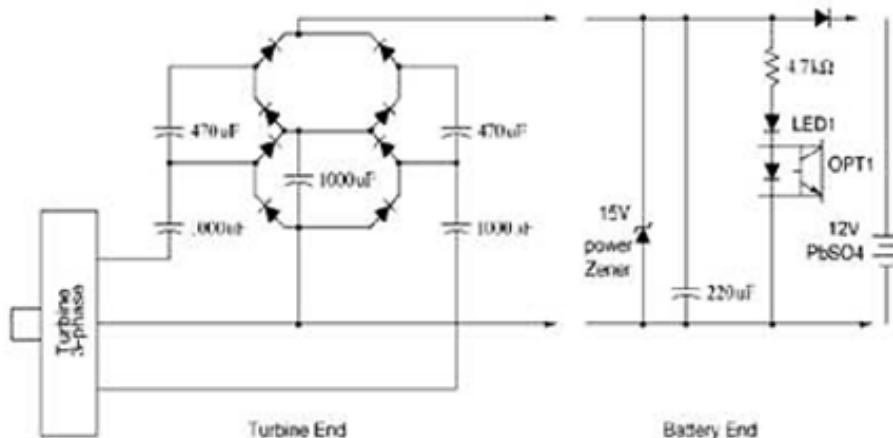


UML(Unified Modeling Language)

모델링 (Modeling)

객체지향 소프트웨어를 모델링 하는 (실질적) 표준 그래픽 언어

- 시스템의 여러 측면을 그림으로 모델링
- 하드웨어의 회로도 같은 의미
- 그림은 회로도 vs UML 사례를 나타냄



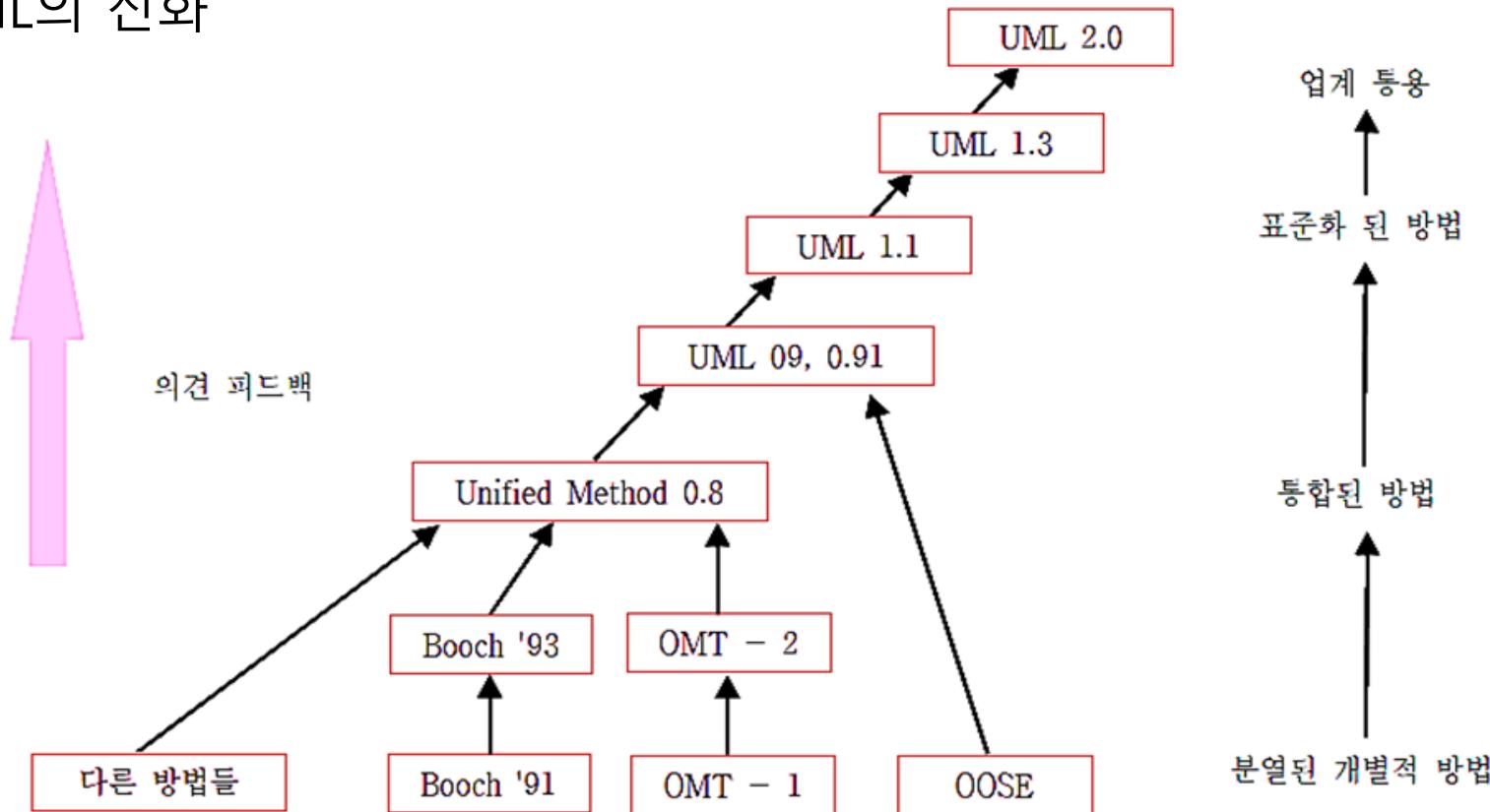


1988년부터 1992년까지 제안된 객체지향 분석 및 설계 방법

- Sally Shlaer와 Steve Mellor가 저술한 두 책에서 제안된 분석 및 설계한 이 방법은 재귀적 접근 방법(recursive design)으로 발전 [Shlaer, 1997]
- Peter Coad와 Ed Yourdon이 제안한 프로토타입 중심의 가벼운 방법[Coad, 1991a][Coad, 1991b][Coad, 1995]
- Smalltalk 그룹에서 제안한 의무 중심 (Responsibility-Driven) 설계 및 클래스-의무-협동 (Class-Responsibility-Collaboration) 방법[Beck, 1989]
- Rational Software에서 일한 Grady Booch의 방법[Booch, 1994, 1995]
- General Electric 연구소의 Jim Rumbaugh의 팀에서 제안한 객체 모델링 테크닉(Object Modeling Technique) 방법[Rumbaugh 1991, 1996]
- Jim Odell이 James Martin과 함께 정보 공학 방법론을 기초로 제안한 방법[Martin, 1994, 1996].
- Ivar Jacobson이 UseCase 개념을 소개한 방법[Jacobson, 1994, 1995].



- UML은 OMT(Object Modeling Technique)[Rumbaugh, 1991]와 Booch[Booch, 1994], OOSE(Object-Oriented Software Engineering)[Jackson, 1992] 방법의 통합으로 만들어진 표현
- UML의 진화





UML을 이용한 시스템 모델링은 크게 세 가지 관점으로 구성됨

- **기능적 관점**: 사용자 측면에서 본 시스템의 기능
→ 요구 분석 단계의 사용 사례 다이어그램
- **구조적 관점**: 시간 개념이 포함되지 않은 정적 모델로서, 구조적 측면을 나타냄
- **동적 관점**: 시간 개념이 포함되는 동적 모델로서, 상태, 순서 다이어그램으로 나타냄

정적 모델(Static Model)

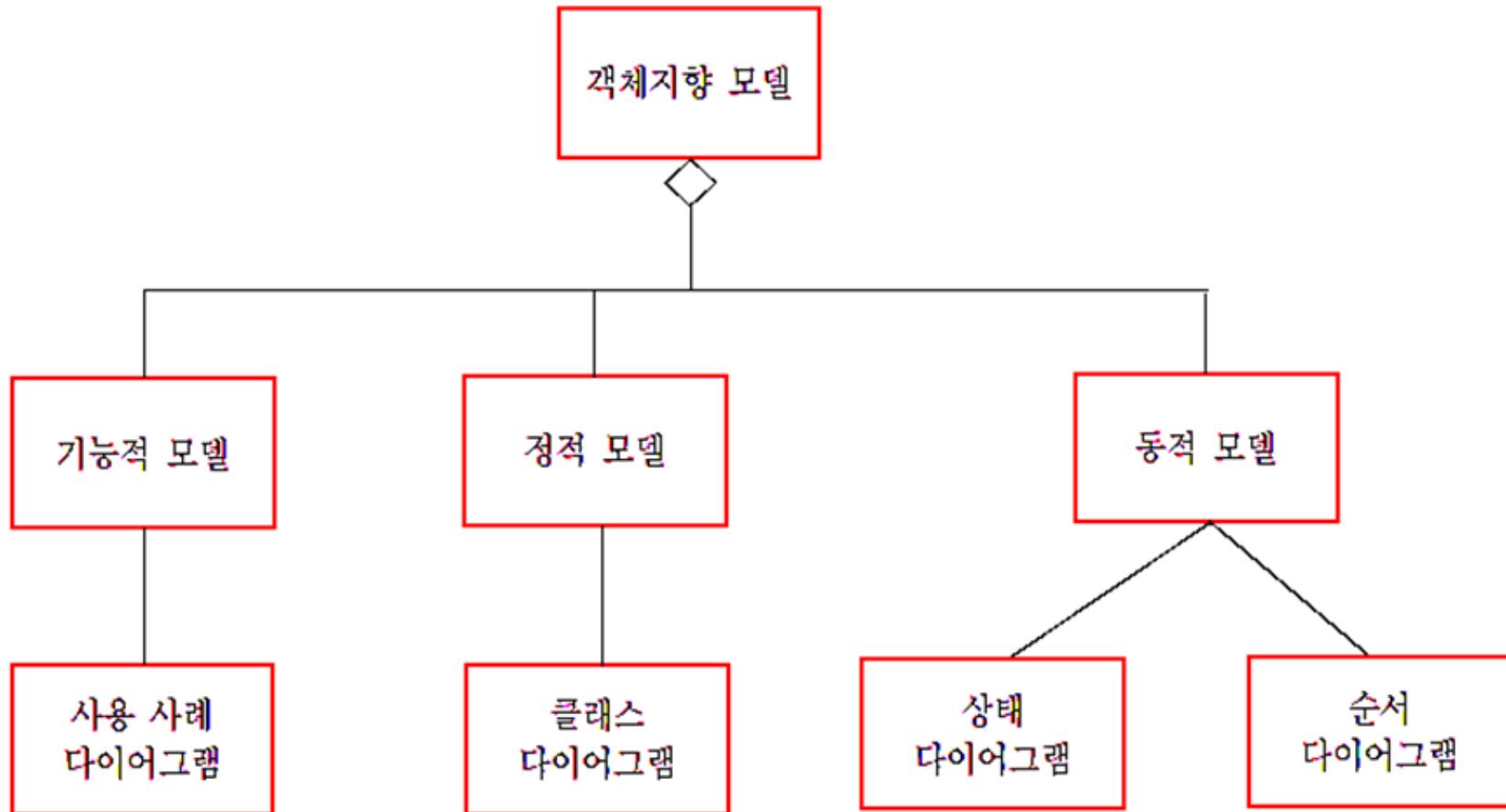
- 객체, 클래스, 속성, 연관, 패키지, 컴포넌트 등을 기초로 시스템의 구조를 나타냄
- 클래스 다이어그램, 패키지 다이어그램, 배치 다이어그램(하드웨어 배치를 나타냄)

동적 모델(Dynamic Model)

- 변화, 흐름 등의 표현을 위해, 시스템의 내부 동작을 나타냄
- 시퀀스 다이어그램, 상태 다이어그램, 액티비티 다이어그램



UML의 세 가지 측면의 모델 도식화





UML 다이어그램과 모델링(1/2)

UML 다이어그램	설명	모델링 적용
사용사례 다이어그램	업무 프로세스를 나타내는 사용사례와 액터가 정점에 표시된 그래프로써 간선은 어떤 액터가 업무 프로세스와 상호작용하는지를 나타낸다.	사용사례 다이어그램은 현재 존재하는 애플리케이션이나 사용자가 개발 요구한 시스템의 업무 프로세스의 개관을 나타내는 데 사용된다. 또한 사용사례 다이어그램은 시스템 또는 서브시스템의 범위를 나타낸다.
클래스 다이어그램	정점은 클래스, 방향이 있는 간선에는 클래스의 관계를 나타내는 방향성 그래프. 정점에는 클래스가 가지고 있는 속성과 오퍼레이션의 정보가 표시되어 있다.	클래스 다이어그램은 도메인 모델을 나타내는 데 사용된다. 개발자가 도메인 개념과 이들 사이의 관계를 이해하고 전달하는 데 도움이 된다.
시퀀스 다이어그램	정점은 객체를 나타내며 방향성 있는 간선은 객체 사이에 오가는 메시지와 요구를 시간 순으로 나타낸다.	시퀀스 다이어그램은 개발팀이 현재의 업무 프로세스를 이해하고 분석하는 데 도움이 된다. 즉 시스템 안에 존재하는 어떤 객체들이 사용사례로 표시된 업무 프로세스에 대하여 개입하여 어떻게 처리하는지를 나타낸다.



UML 다이어그램과 모델링(2/2)

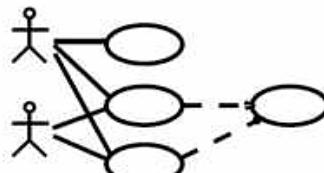
UML 다이어그램	설명	모델링 적용
상태 다이어그램	정점에는 시스템의 상태 방향이 있는 간선 에는 상태의 변환을 나타낸 방향성 있는 그래프.	상태 다이어그램은 상태 의존적이며 반응적인 시스 템 또는 서브시스템의 동작을 나타내는 데 사용된다.
액티비티 다이어그램	각 정점은 정보를 처리하는 작업을 나타 내며 방향이 있는 간선은 자료 및 제어 흐 름을 나타내는 방향성 있는 그래프. 제어 흐름은 순차, 병렬, 동기화를 나타낸다.	액티비티 다이어그램은 시스템 또는 서브시스템의 복잡한 작업 흐름을 나타내는 데 사용된다.
패키지 다이어그램	정점은 클래스의 묶음인 패키지, 방향이 있 는 간선은 패키지의 의존관계를 나타낸다.	패키지 다이어그램은 복잡한 클래스를 묶어서 서브시 스템으로 조직화하는 데 사용된다.
배치 다이어그램	정점은 분산 시스템의 물리적인 컴퓨팅 파 워와 그 위에 실행되는 컴포넌트를 나타내 며 간선은 네트워크 연결을 나타낸다.	배치 다이어그램은 분산 시스템의 각 컴퓨팅 노드, 컴포넌트, 커넥터 등 시스템의 물리적 자원 배치를 나타내는 데 사용된다.



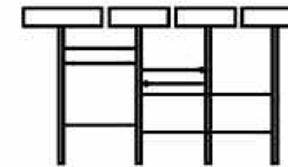
- ① 요구를 사용 사례로 정리하고 사용 사례 다이어그램을 작성
- ② 클래스 후보를 찾아내고 개념적인 객체 모형을 작성
- ③ 사용 사례를 기초하여 순서 다이어그램을 작성
- ④ 클래스의 속성, 오퍼레이션 및 클래스 사이의 관계를 찾아
객체 모형을 완성
- ⑤ 상태 다이어그램이나 액티비티 다이어그램 등 다른 다이어그램을
추가하여 UML 모델을 완성
- ⑥ 서브시스템을 파악하고 전체 시스템 구조를 설계
- ⑦ 적당한 객체를 찾아내거나 커스텀화 또는 객체를 새로 설계



UML 모델링 과정의 도식화

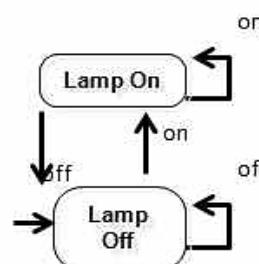


사용 사례
다이어그램

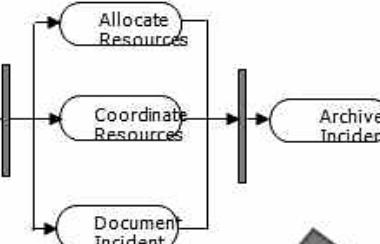


순서
다이어그램

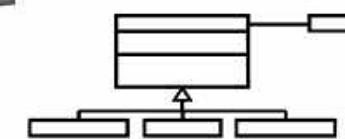
클래스 후보 선정 및
개념적 클래스 다이
어그램 작성



상태
다이어그램



액티비티
다이어그램



클래스
다이어그램



성공적인 모델링을 위한 기타 조건

- 복잡한 문제라면 도메인을 잘 아는 전문가와 같이 모델링 함
- 각 모델의 목적을 잘 이해하고 모델링을 위하여 어떤 정보가 필요한지 잘 알아 둠
- 한번 그린 모델로 만족하지 않고 계속 논의하고 향상시켜 나감
→ 이를 리팩토링이라 함
- 소그룹 회의를 열어 모델을 칠판에 그리고 토의함
- 디자인 패턴을 잘 숙지하고 필요하면 이를 이용함





In this chapter ...

모델링 (Modeling)

5.1 객체지향 개념

5.2 UML

5.3 정적 모델링

5.4 동적 모델링

5.5 모델링 도구



정적 모델: 시간 개념이 개입되지 않은 모델

- 클래스 다이어그램이 대표적 표현임
- 클래스와 클래스 사이의 관계를 나타내는 UML의 핵심 다이어그램임

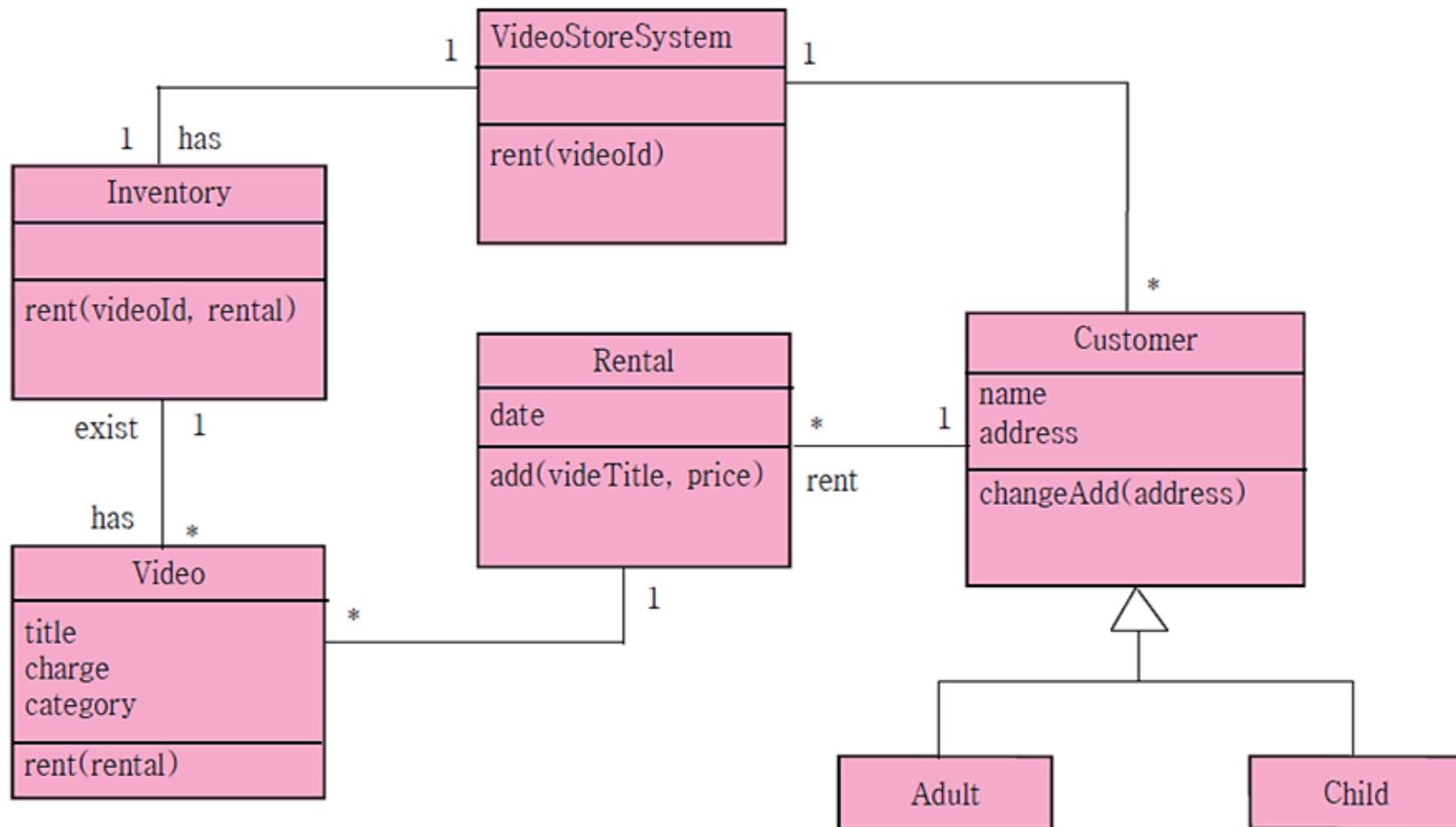
클래스 다이어그램의 표현 요소

요소	의미	표현방법	
클래스 속성	클래스는 타입이며 속성과 오퍼레이션은 클래스의 객체의 특징을 결정한다.	축약형	확장형
오퍼레이션		클래스 이름	클래스 이름 속성 리스트 오퍼레이션 리스트
상속	두 클래스의 일반화 및 상세화 관계	서브클래스	슈퍼클래스
집합	두 클래스 사이의 전체-부분 관계	부분	전체
연관 방향 다중도 역할	두 클래스 사이의 관계	클래스 1 [m] [역할1]	클래스 2 [레이블] [n] [역할2]
연관 클래스	연관을 나타내는 클래스	연관 클래스	



클래스 다이어그램의 예제

- 객체, 클래스, 속성, 오퍼레이션, 연관 등을 표현





클래스 심볼 – 세 부분으로 나누어 표현

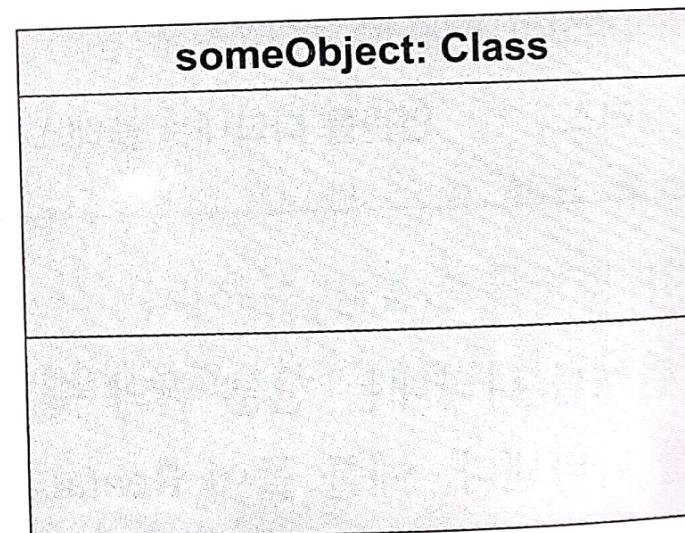
- 맨 위는 클래스 속성, 중간에는 클래스의 속성, 아래는 오퍼레이션

속성과 오퍼레이션의 외부 가시성(visibility)은 +, -, #로 표현

- +는 public으로 외부에서 접근 가능, -는 private로 외부에서 접근 불가
- #은 상속된 클래스에서만 제한적으로 사용 가능

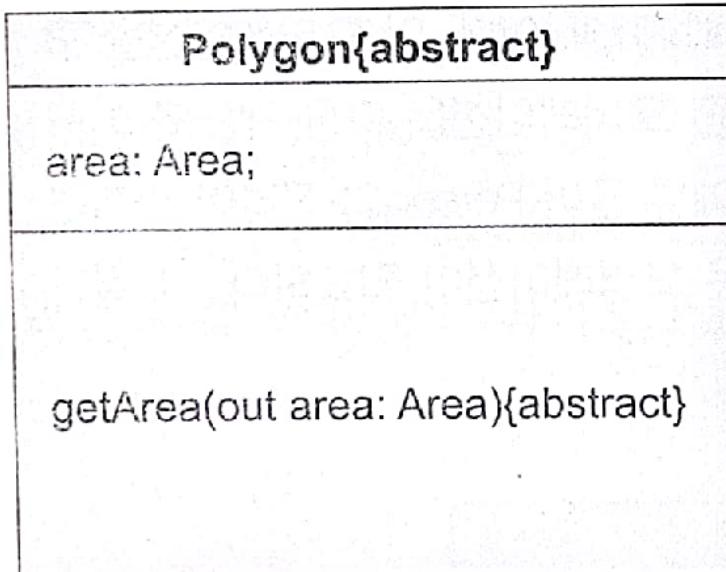
클래스와 객체의 표현 예제

Employee
+Name: char
-Position: positiontype
-Salary: int
#Phone: phonetype
+promote(from, to)
-increase_salary(new_salary)
#change_phone(new_number)

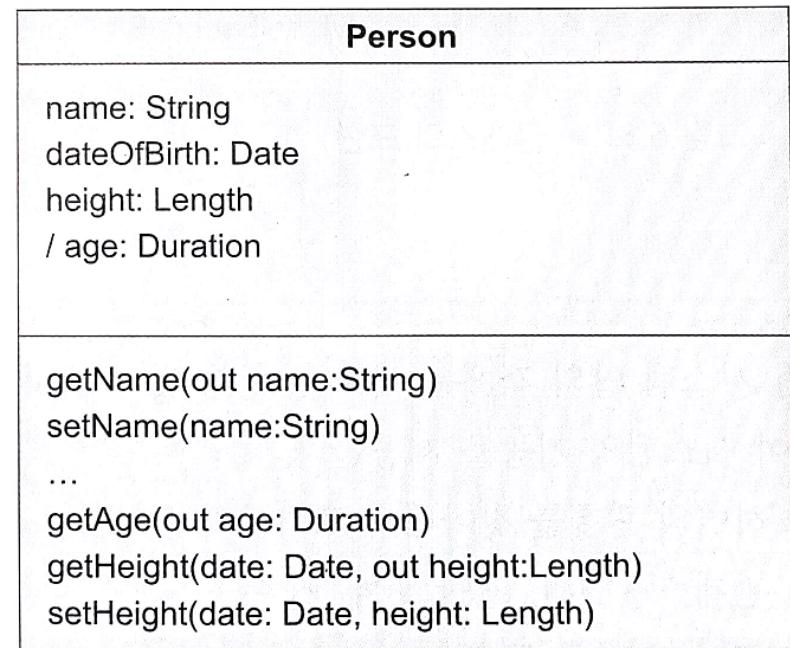




- 인스턴스가 없는 추상 클래스나 구현이 없는 추상 오ペ레이션은 정의 후에 <abstract>라고 표시함



- 예제: Person 클래스의 표현

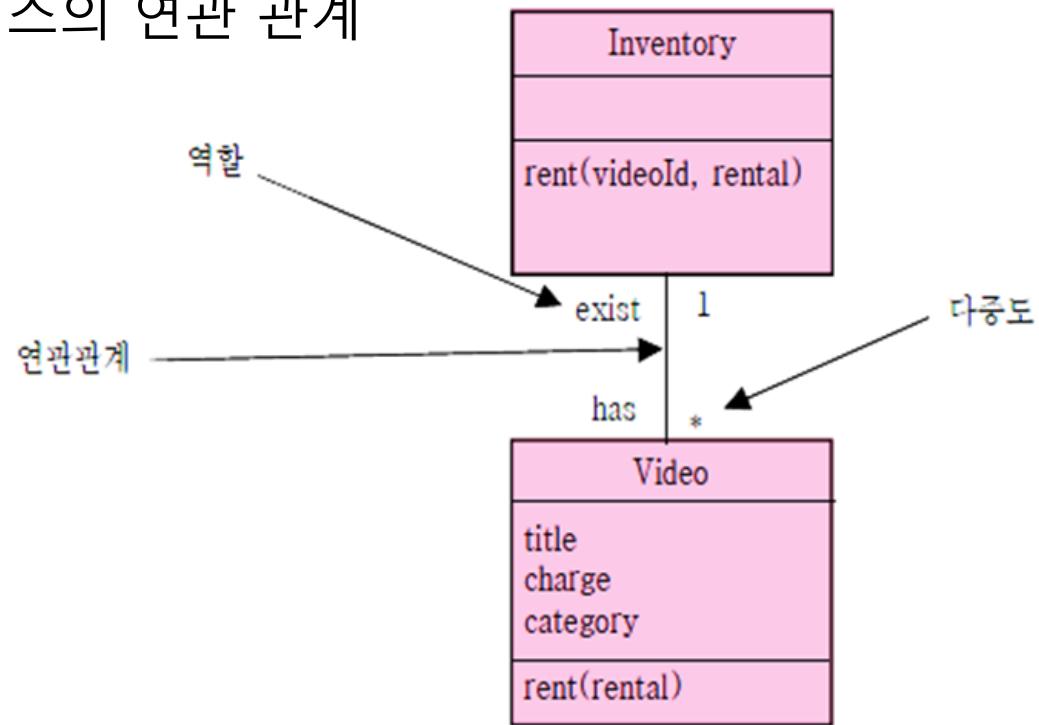




관계의 표현 – 연관(Association)

모델링 (Modeling)

- 연관은 클래스(객체) 사이의 관계/연결을 나타냄
- 연관의 표현: 클래스를 선분으로 연결
 - 선분의 끝에는 역할(role)을 표시함
 - 선분 끝의 숫자는 다중도(multiplicity)를 나타냄
(예: Customer는 여러 번의 대출이 가능하므로, 각각 1과 *로 표현)
- 예제: Inventory와 Video 클래스의 연관 관계

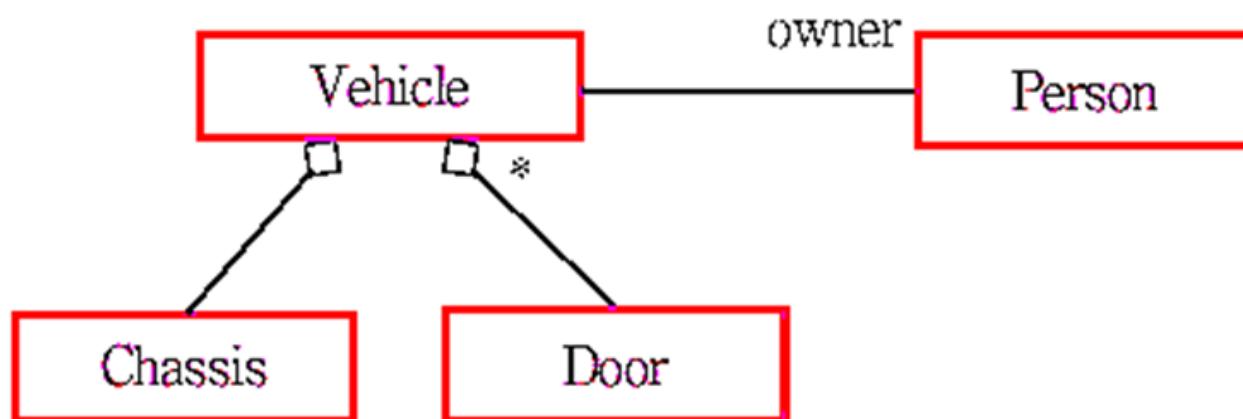




관계의 표현 – 집합(Set)

모델링 (Modeling)

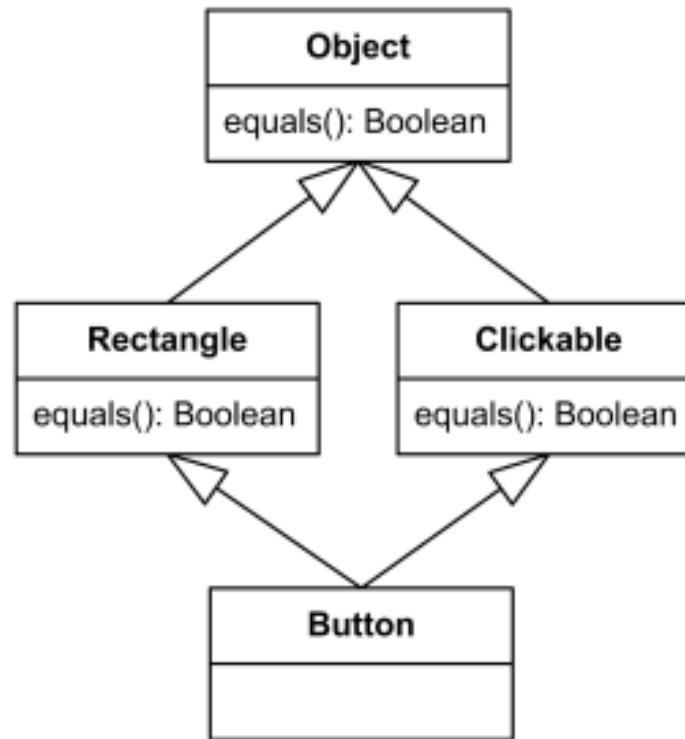
- 집합 관계는 어떤 클래스가 다른 클래스의 모임(부분)으로 구성
→ UML에서는 전체 부분 관계로 표현
- 전체 부분 관계의 표현: 전체 개념의 클래스에 다이아몬드 표시
- 예제: 자동차와 부품과의 전체 부분 관계





- 일반화된 개념의 클래스와 더 구체적인 개념의 클래스 사이의 관계
- 상속의 표현: 속이 빈 삼각형 화살표로 나타냄

상속의 예제

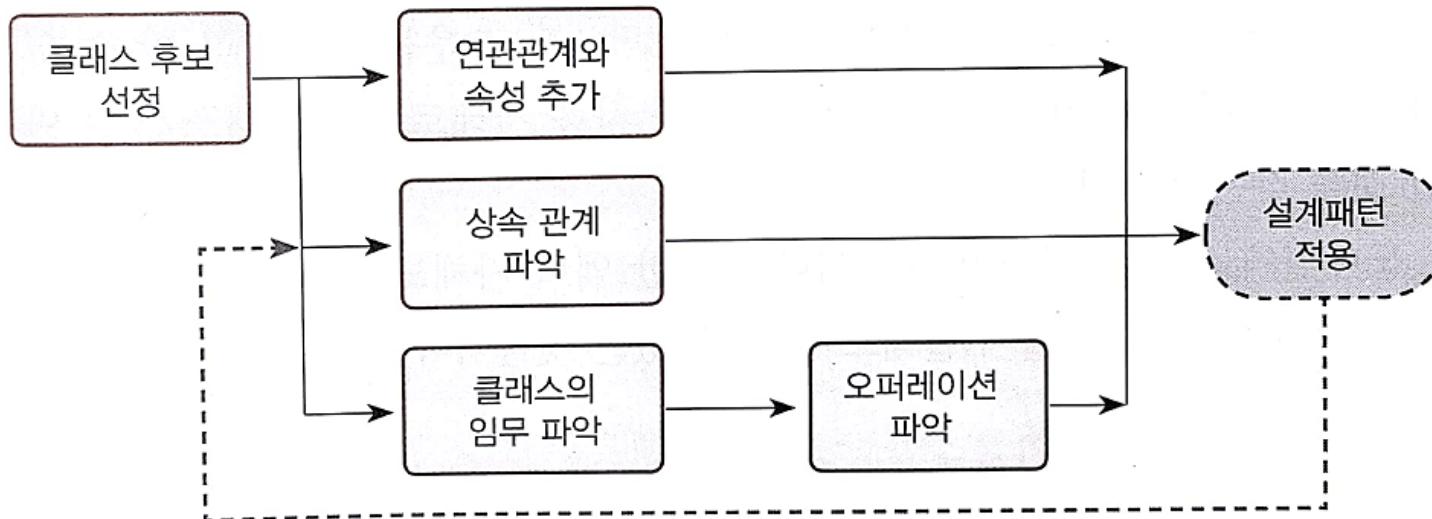




클래스 다이어그램의 작성 과정

모델링 (Modeling)

- **Step 1** 클래스가 될 만한 후보를 파악한다.
- **Step 2** 가장 중요한 클래스를 시작으로 연관, 상속, 속성을 추가한다.
- **Step 3** 클래스의 주요 임무(responsibility)를 찾아내어
오퍼레이션으로 추가한다.
→ [전체 과정을 되풀이하며 모델을 완성해 나간다.]





클래스를 찾는 작업은 도메인 개념, 즉 사용 사례로부터 클래스가 될 만한 것을 찾는 일

클래스가 될 수 있는 요소

- 구조, 외부 시스템, 디바이스
- 역할, 운용 절차
- 장소, 조직
- 완성된 시스템에 의하여 조작되어야 할 정보

역할에 따른 클래스의 유형

- 엔티티 클래스(entity class)
- 경계 클래스(boundary class)
- 제어 클래스(control class)

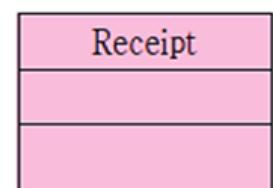
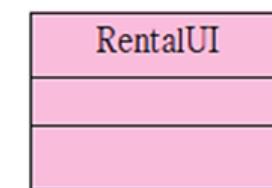
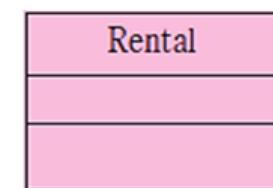
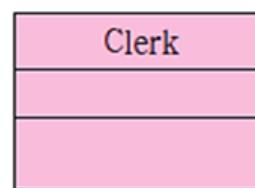
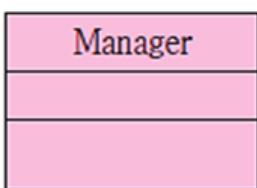
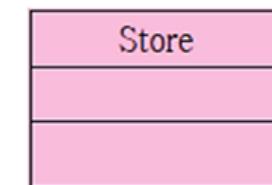
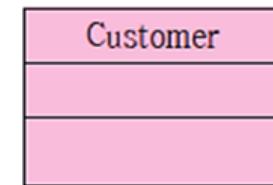
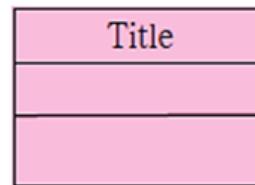
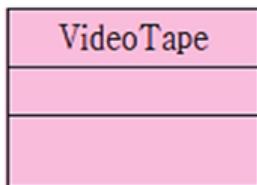


엔티티 클래스 찾기

- 사용 사례를 이해하기 위하여 사용자와 개발자가 명확히 규정한 용어
- 사용 사례에서 반복되어 나오는 용어 (예: Video Tape)
- 시스템이 계속 주적하여야 하는 실세계의 엔티티 (예: Rental, Title)
- 자료저장소 또는 단말(예: Scanner)
- 자주 사용하는 응용 도메인의 용어(예: Customer)



비디오 대여점의 엔티티 후보 (RentalUI는 경계 클래스임)





경계 클래스 찾기

- 사용자가 자료를 시스템에 입력하기 위하여 필요한 양식과 윈도우를 찾음
(예: RentalUI, ReportRental)
 - 시스템이 사용자에게 반응하는 메시지나 알림을 찾음(예: PendingRentalNotice)
 - 인터페이스가 어떻게 보이는지는 경계 객체에 모형화 하지 않음
 - 인터페이스를 나타내는 사용자 언어는 구현 기술과 관련 없는 용어 사용
- ➔ 주로 입출력, 인터페이스와 관련됨





제어 클래스 찾기

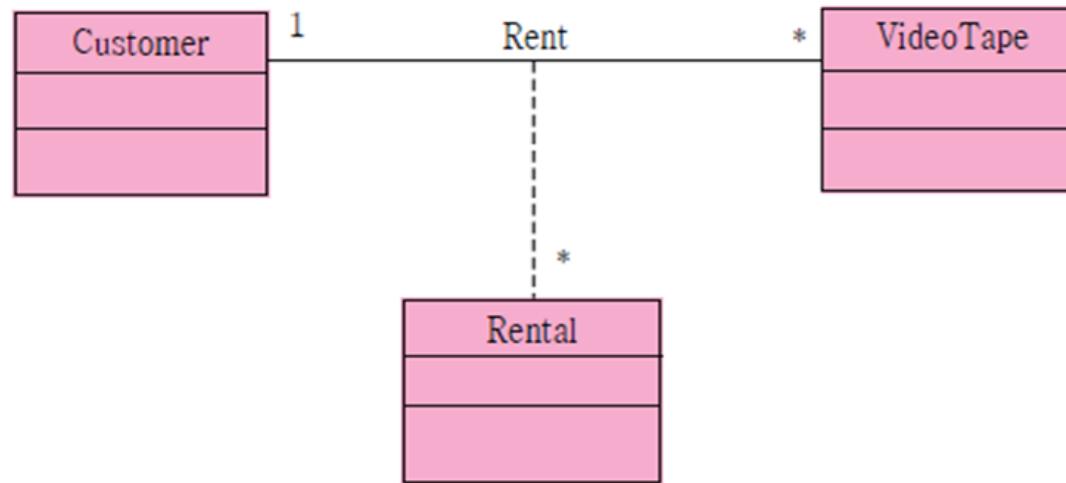
- 경계 클래스와 엔티티 클래스 사이에 중간 역할
- 경계 클래스로부터 정보를 받아 (적절한 처리/제어를 거쳐) 엔티티 클래스에게 전달
- 제어 클래스는 비즈니스 로직을 담고 있는 클래스로도 볼 수 있음
- 예를 들면, 양식의 순서, Undo, 히스토리 저장 큐 관리 등의 동작을 나타냄
- 사용 사례가 복잡하여 소규모의 이벤트로 분할해야 한다면, 하나 이상의 사용 사례 당 한 개의 제어 클래스를 찾음
- 사용 사례에서 액터 하나 당 하나의 제어 클래스를 찾음



연관 찾기 (1/2)

모델링 (Modeling)

- 연관은 두 개 이상의 클래스 사이에 어떤 관계가 있을 때 나타냄
- 예제: Customer와 VideoTape 클래스 사이의 연관 관계(Rent)

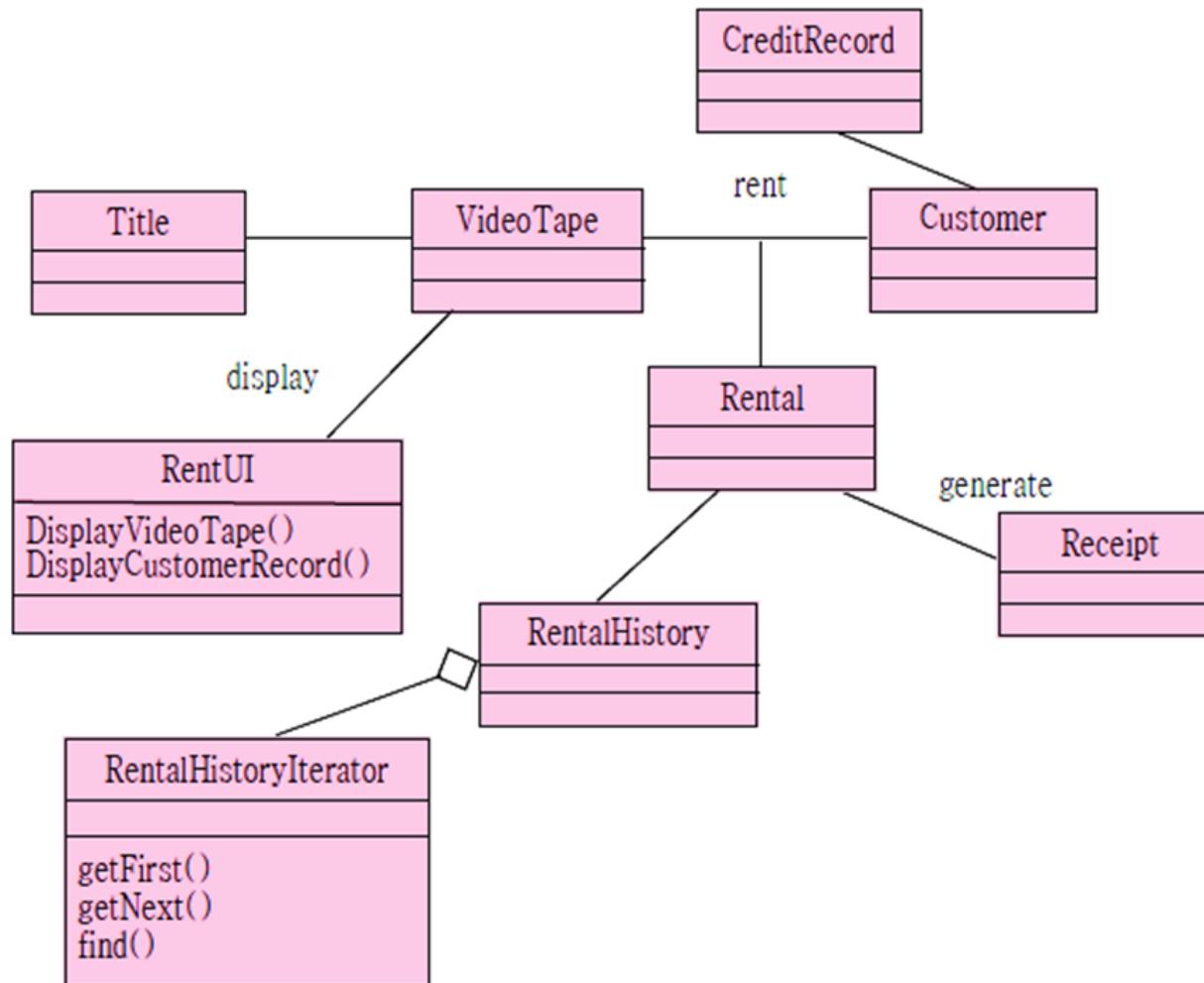


연관의 속성

- 이름**: 두 클래스 사이의 연관 관계를 나타냄
- 역할**: 연관 관계의 양쪽 끝에 있는 클래스의 기능을 나타냄
- 다중도**: 연관 관계를 구성하는 인스턴스의 개수



비디오 대여 시스템에 대한 클래스들의 관계





속성 추가

모델링 (Modeling)

속성: 개별 객체들이 가지는 특성

Receipt
-Date : date -Title : string -Total : float

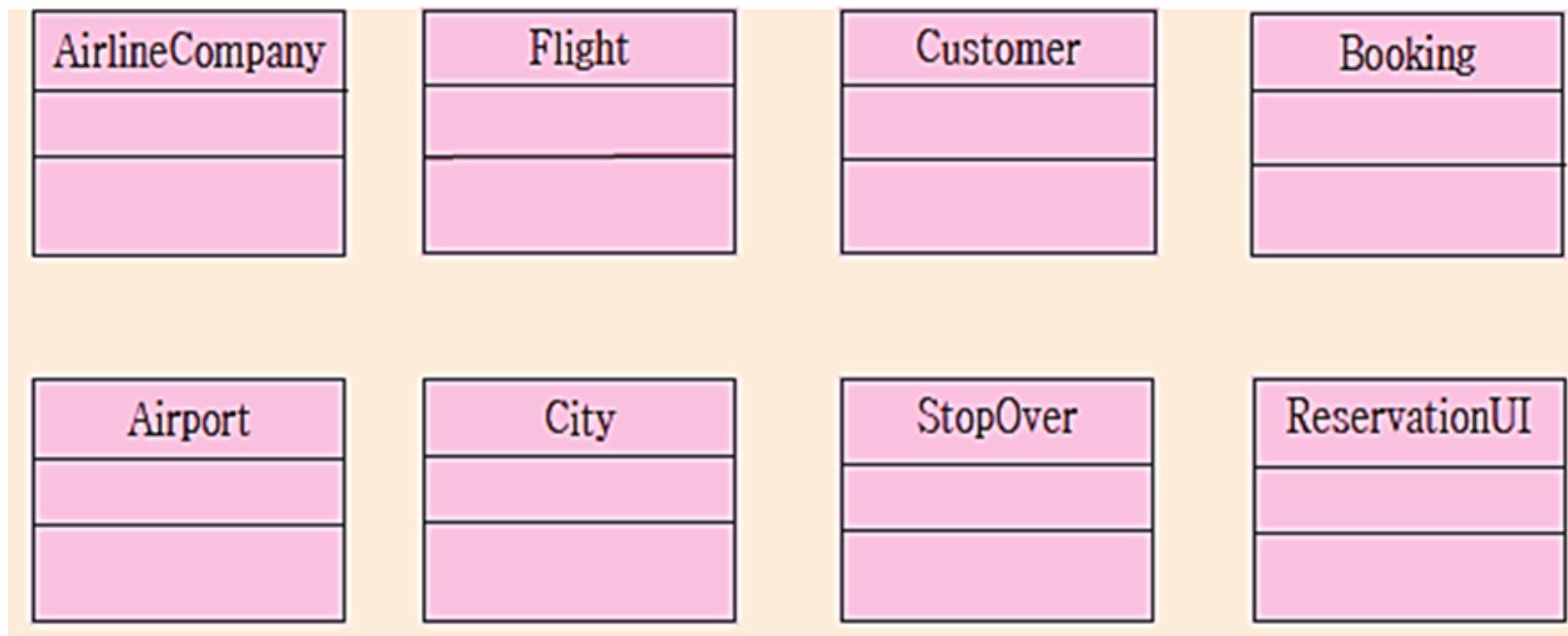
Customer
-Name : string -Address : string -Phone : string -Age : int

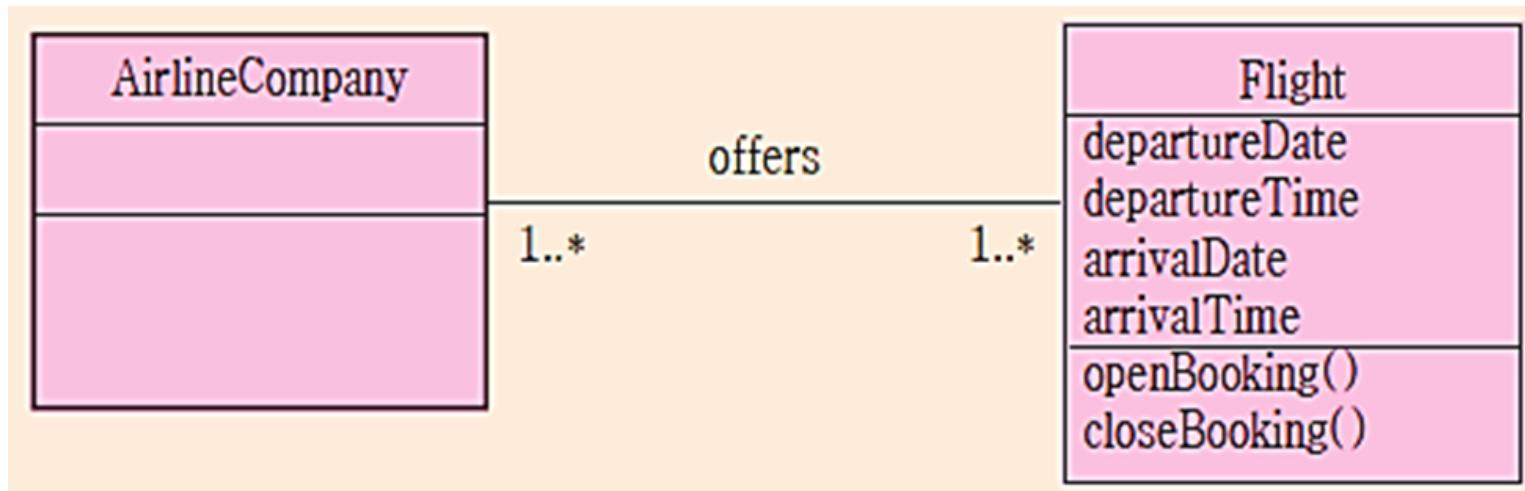
Rental
-Date : date -Duration : int -Status : enum

VideoTape
-PurchaseDate : date -Supplier : company

속성의 요소

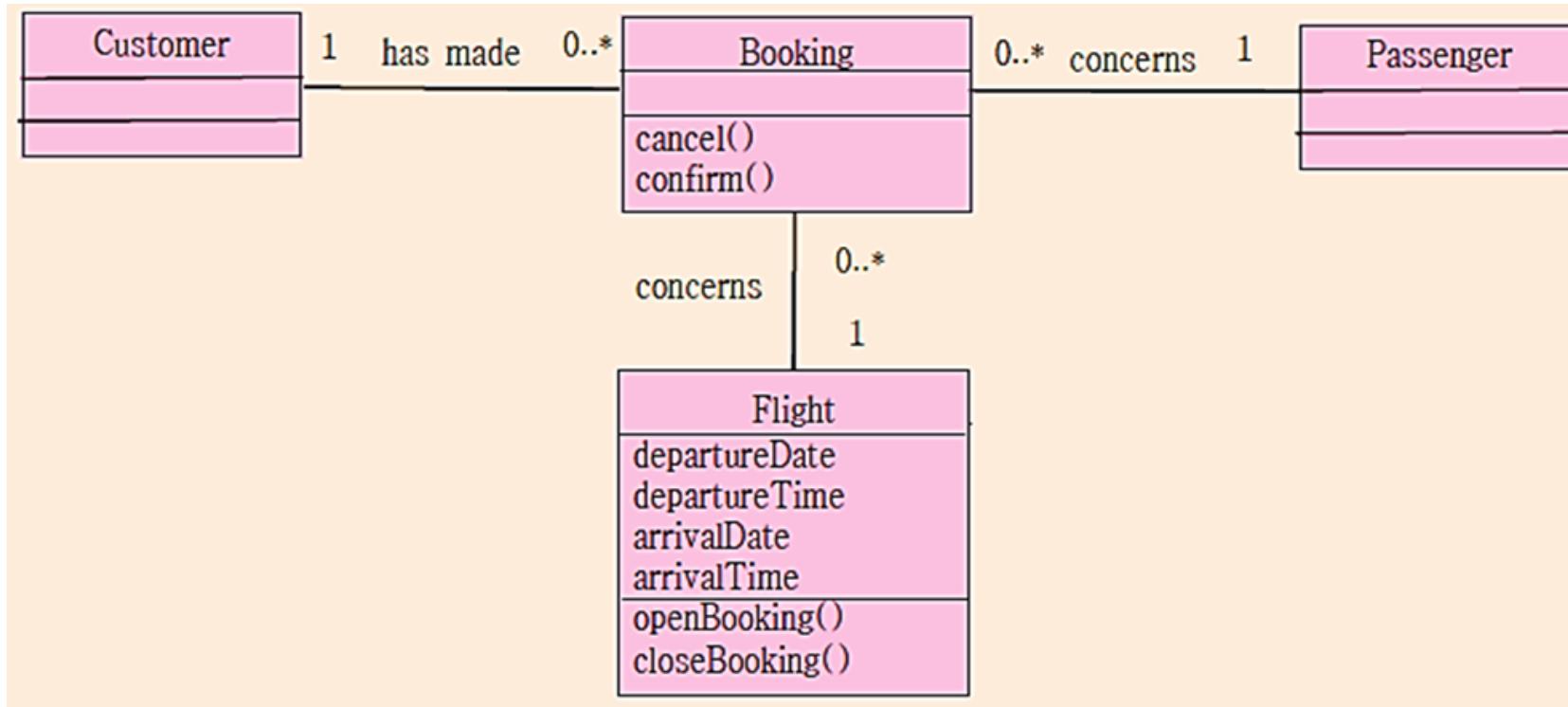
- 이름:** 객체 안에서 구별할 수 있는 속성의 이름.
(예를 들어 VideoTape은 PurchaseDate와 Supplier 속성을 가짐. PurchaseDate은 테이프를 구매한 날짜이며 Supplier는 비디오 공급업체를 나타냄)
- 간단한 설명:** 구현하는 프로그래머를 위하여 간단히 설명을 첨가
- 속성값의 타입:** 예를 들어 Name 속성은 스트링. 또한 Status는 열거형으로 rentable, rented, returned라는 값을 가질 수 있음

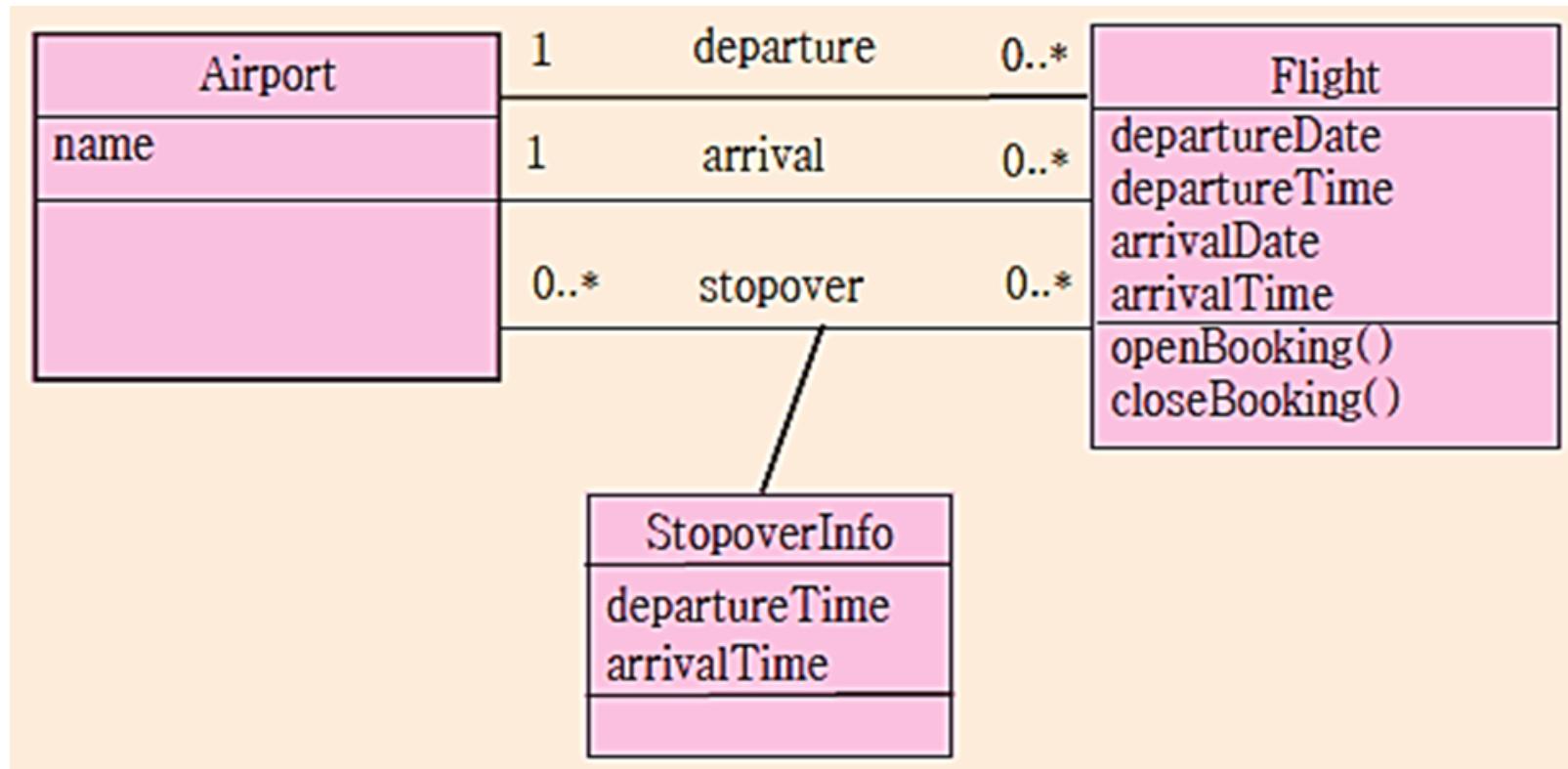
 클래스 찾기 (클래스 후보 도출)

 관계 찾기 – 항공편과 항공사의 관계



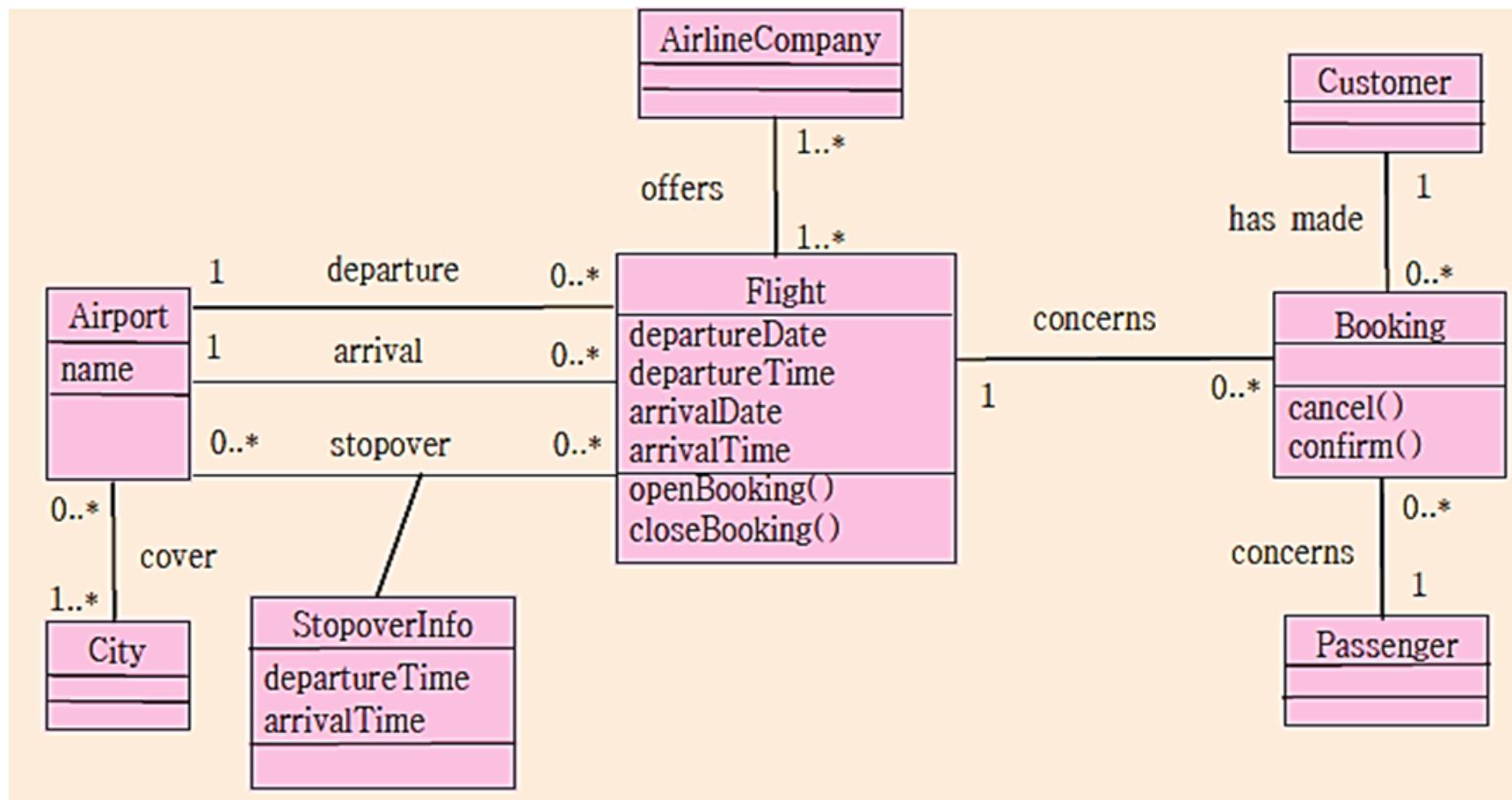
관계 찾기 – 항공편과 예약, 고객, 탑승자 관계



 관계 찾기 – 항공편과 공항의 관계



항공권 예매 시스템의 모델링 결과





In this chapter ...

모델링 (Modeling)

5.1 객체지향 개념

5.2 UML

5.3 정적 모델링

5.4 동적 모델링

5.5 모델링 도구



- (classes) 클래스들의 상호작용이나 클래스의 상태 변화 등 시스템 내부의 동작을 모델링함
- (diagram) 동적 모델링의 다이어그램
 - 인터랙션 다이어그램**: 사용 사례를 실현시키기 위하여 내부 클래스들이 어떻게 협동하는지 나타내는 그림 ([시퀀스 다이어그램](#), [커뮤니케이션 다이어그램](#))
 - 상태 다이어그램**: 복잡한 객체의 상태 변화를 나타낸 것으로, 이벤트 발생에 의해 객체가 어떻게 변화하는지 나타냄
 - 액티비티 다이어그램**: 절차나 작업의 흐름을 나타낸 것으로, 순차적/병렬적 작업인지, 어떤 작업과 동기화되어야 하는지를 나타냄



- 시스템의 동작을 정형화하고, 객체들의 메시지 교환을 울타리 형태로 시각화 하여 나타냄
- 시퀀스 다이어그램에서 객체의 표현

:Person

Person 클래스의 이름 없는 인스턴스

kim:

이름 없는 클래스의 kim이라는 인스턴스

kim:Person

Person 클래스의 kim이라는 인스턴스

:Person

Person 클래스의 인스턴스의 모임

<<jsp>>

LoginPage:

스테레오타입 객체



시퀀스 다이어그램의 요소 (2/3)

모델링 (Modeling)



시퀀스 다이어그램의 기본 요소

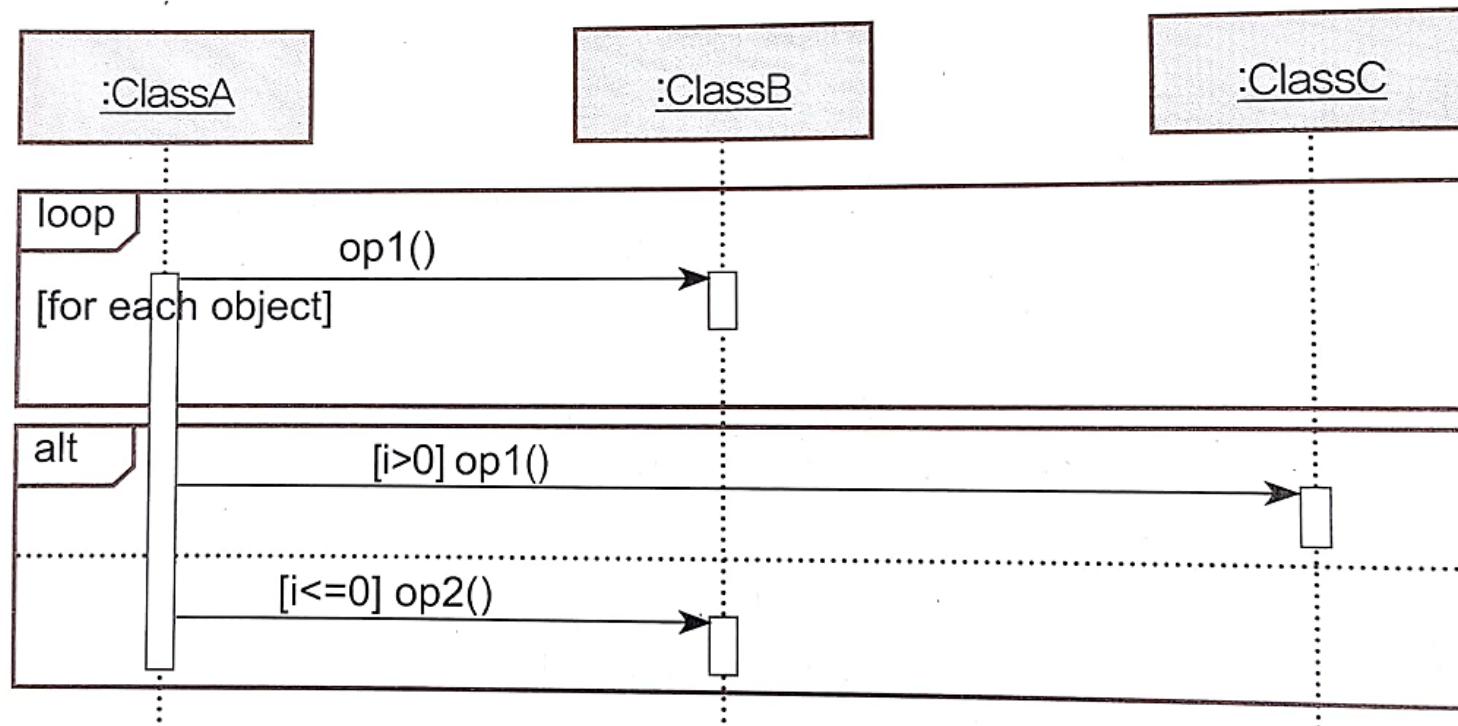
요소	표현방법	의미	연결
객체		<ul style="list-style-type: none"> - 특정 클래스의 객체 - 객체의 모임 	라이프라인 사이에 활성막대와 연결됨
객체집합		<ul style="list-style-type: none"> - 라이프라인 위에 위치하며 콜론 앞은 객체의 이름, 뒤는 클래스의 이름 	
라이프라인		객체가 시스템에 존재하나 아직 실행되지는 않음을 의미.	객체를 활성막대와 연결시키며 두 개의 이웃 라이프라인을 연결
활성막대		시스템에 존재하는 메소드가 막대의 길이만큼 실행될을 의미. 점선은 라이프라인임.	객체와 연결됨. 라이프라인과 연결됨
객체 소멸		라이프라인 맨 위에 연결된 객체가 소멸됨을 의미.	
메시지 호출		한 객체에서 다른 객체로 메시지를 보냄을 의미. 즉 함수가 호출됨	상호작용하는 두 객체를 연결함
프레임		시퀀스 다이어그램의 일부로 반복 또는 택일 구조의 묶여진 조작.	



시퀀스 다이어그램의 요소 (3/3)

모델링 (Modeling)

시퀀스 다이어그램에서 반복(loop)과 조건(alt)의 예제

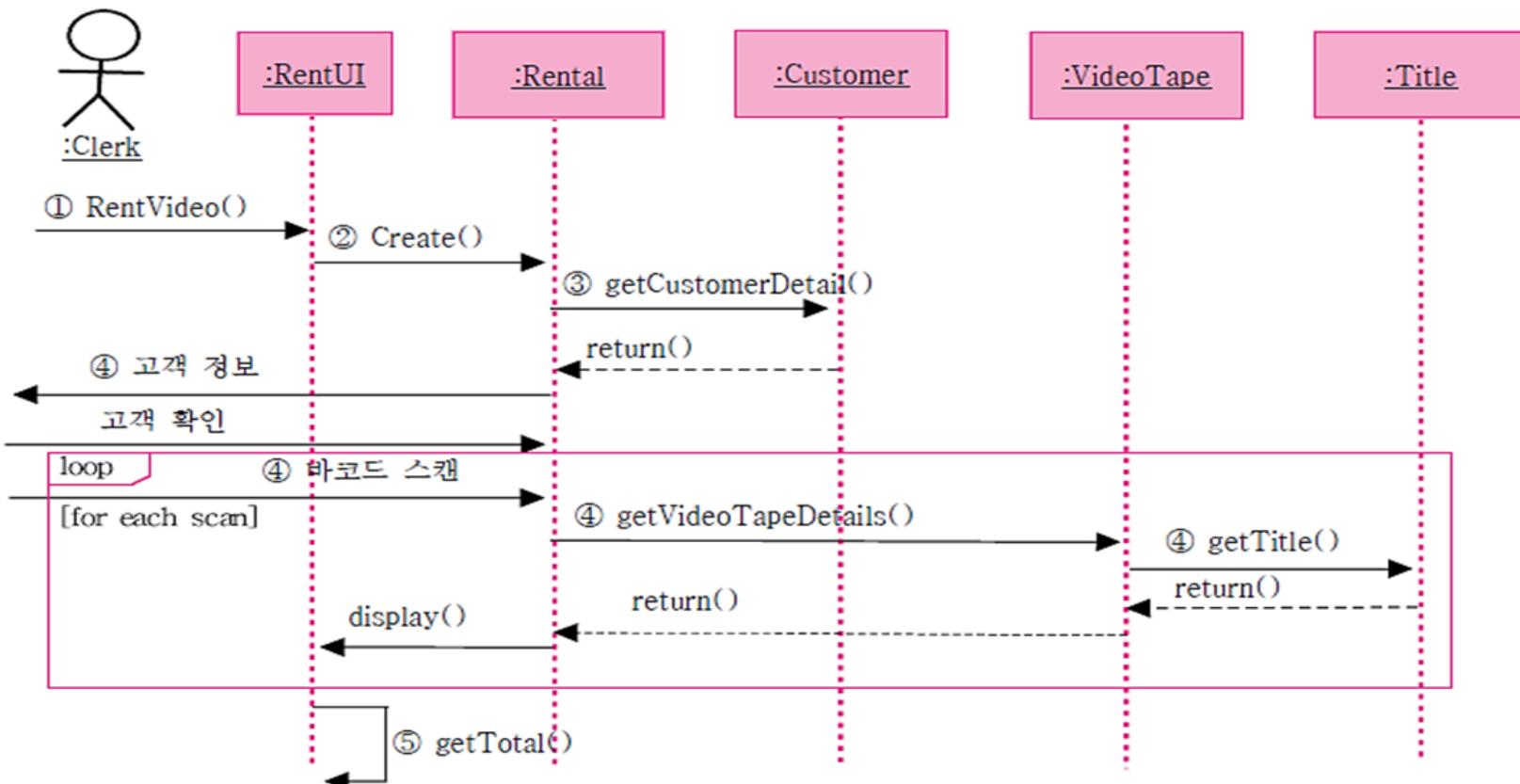




시퀀스 다이어그램 작성

모델링 (Modeling)

- ▣ Step 1 참여하는 객체를 파악
- ▣ Step 2 파악한 객체를 X축에 나열하고 라이프라인을 그음
- ▣ Step 3 사용사례에 기술된 이벤트 순서에 따라 객체의 메시지 호출
- ▣ 비디오 대여 사용 사례

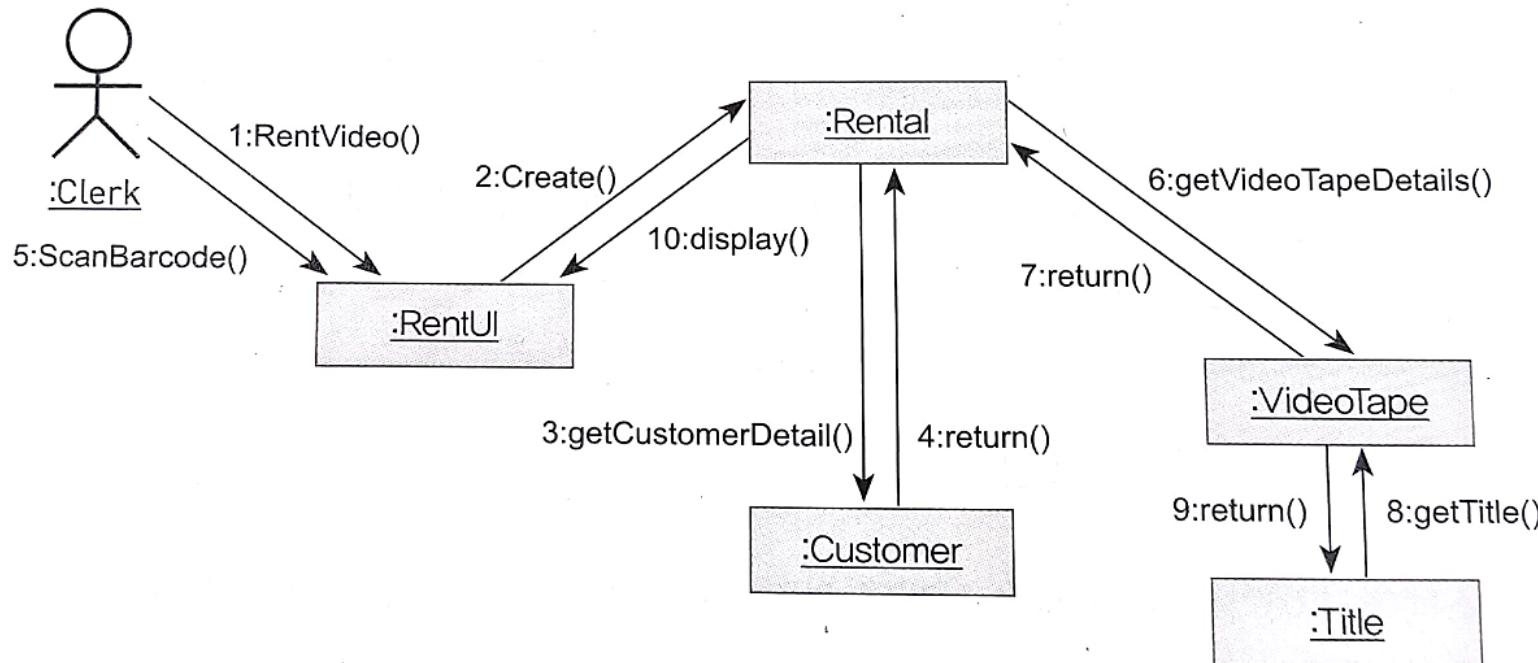




객체들이 정적으로 어떻게 연결되어 있는지를 보여줌

- 시퀀스 다이어그램과 같이 객체는 박스로 화살표는 메시지 전달을 나타냄
- 다른 점은 메시지 호출의 순서가 숫자로 표시된다는 점

비디오 대여 사용 사례





상태 다이어그램 (1/4)

모델링 (Modeling)

객체가 가질 수 있는 가능한 상태 표현

- **이벤트**: 서브시스템 또는 객체나 컴포넌트에 대하여 요청이나 관심이 일어난 것
- **상태**: 이벤트의 발생으로 들어가거나 빠져 나오게 되는 서브시스템 또는 객체의 조건을 추상적으로 이름 붙여 놓은 것

상태 다이어그램의 기본 요소

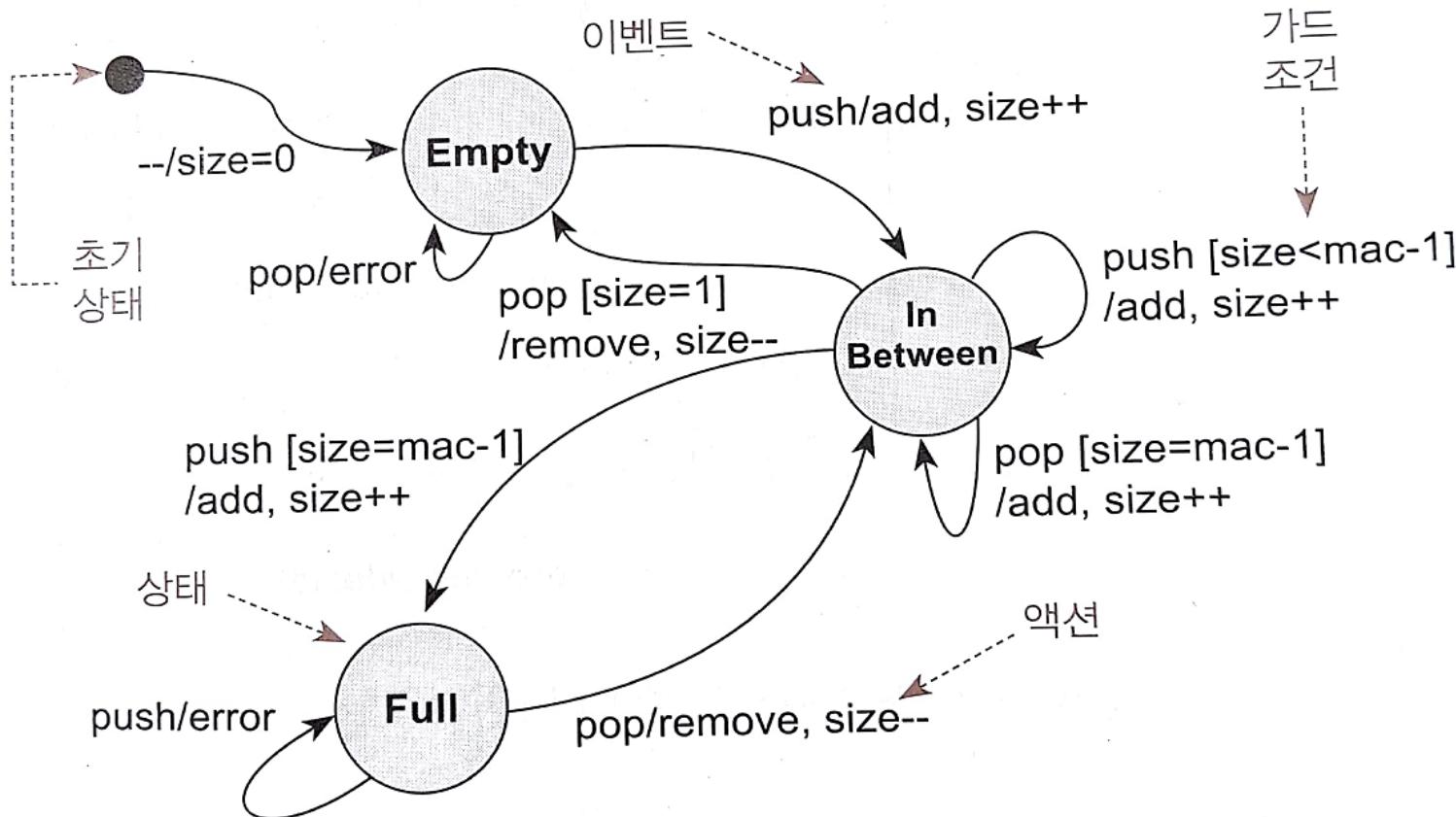
요소	표현방법	연결
상태	□	단순 상태: 다른 상태를 품고 있지 않은 단순한 상태. 서브시스템이나 객체의 조건이나 상황이다.
시작 상태	●	시스템이 시작되었을 때 머무르는 가상의 상태
종료 상태	○	시스템이 종료되었음을 나타내는 상태
트랜지션	→	하나의 상태에서 다른 상태로 이벤트에 의하여 변화됨
레이블	e[exp] / a1; a2	이벤트(e)가 발생하고 가드조건(exp)이 참이면 트랜지션이 일어나고 액션 a1, a2가 실행됨



상태 다이어그램 (2/4)

모델링 (Modeling)

스택의 상태 다이어그램 예제

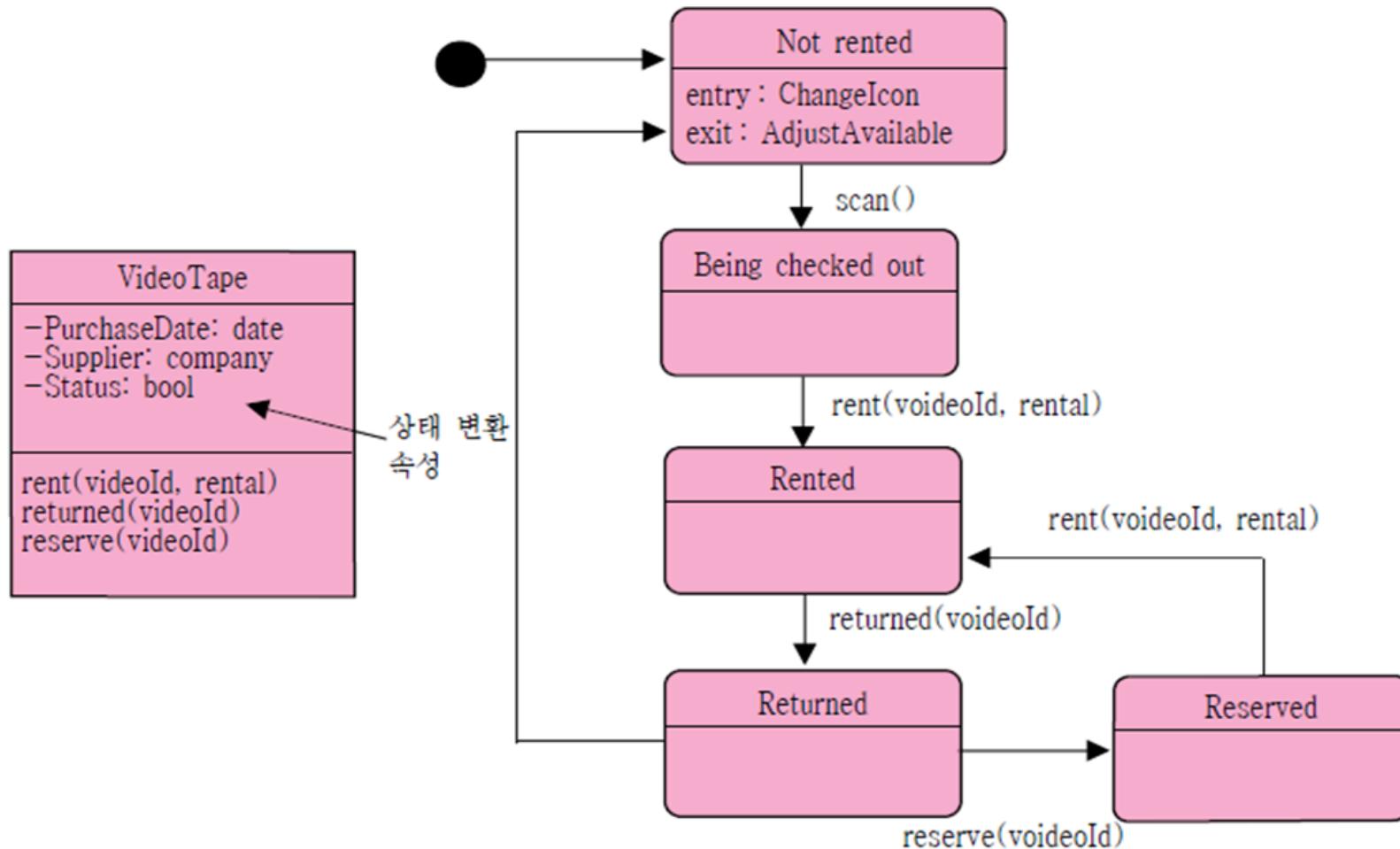




상태 다이어그램 (3/4)

모델링 (Modeling)

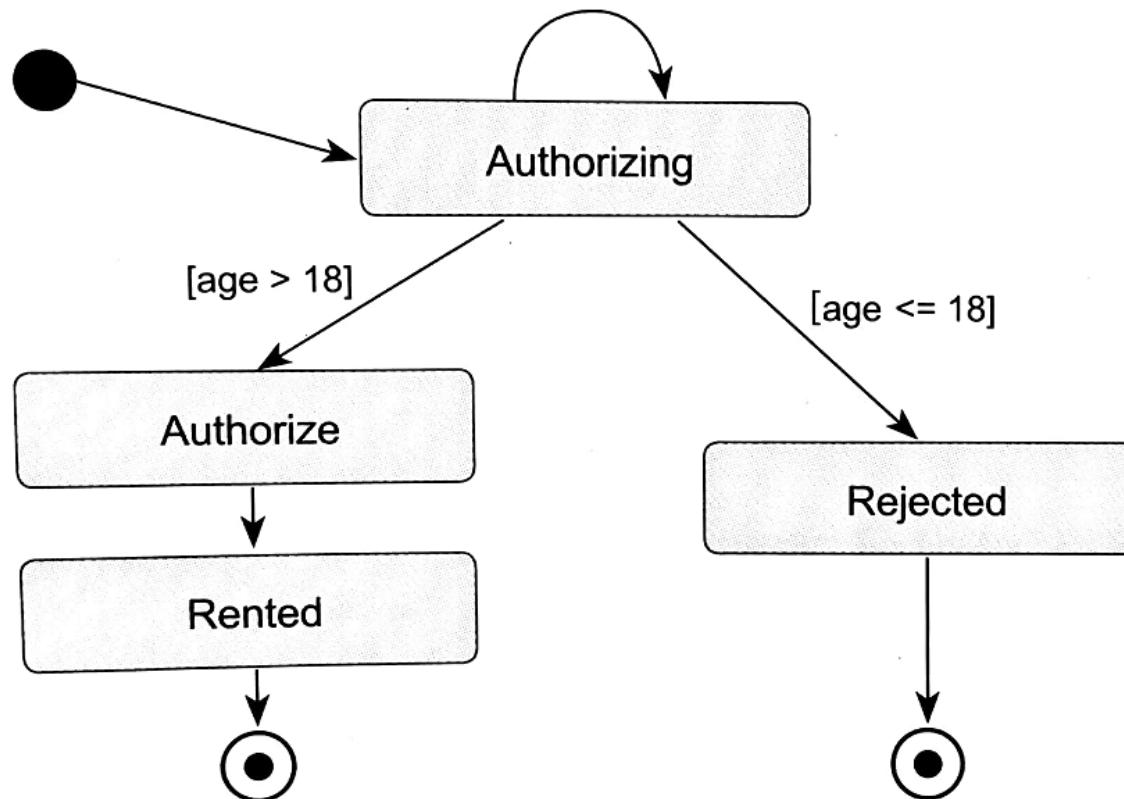
비디오 대여 시스템의 상태 다이어그램 예제





상태 다이어그램을 모델링하기에 적합한 속성의 조건

- 속성의 값으로 가질 수 있는 종류가 적어야 함
- 속성의 값에 따라 허용되는 오퍼레이션이 제한되어야 함





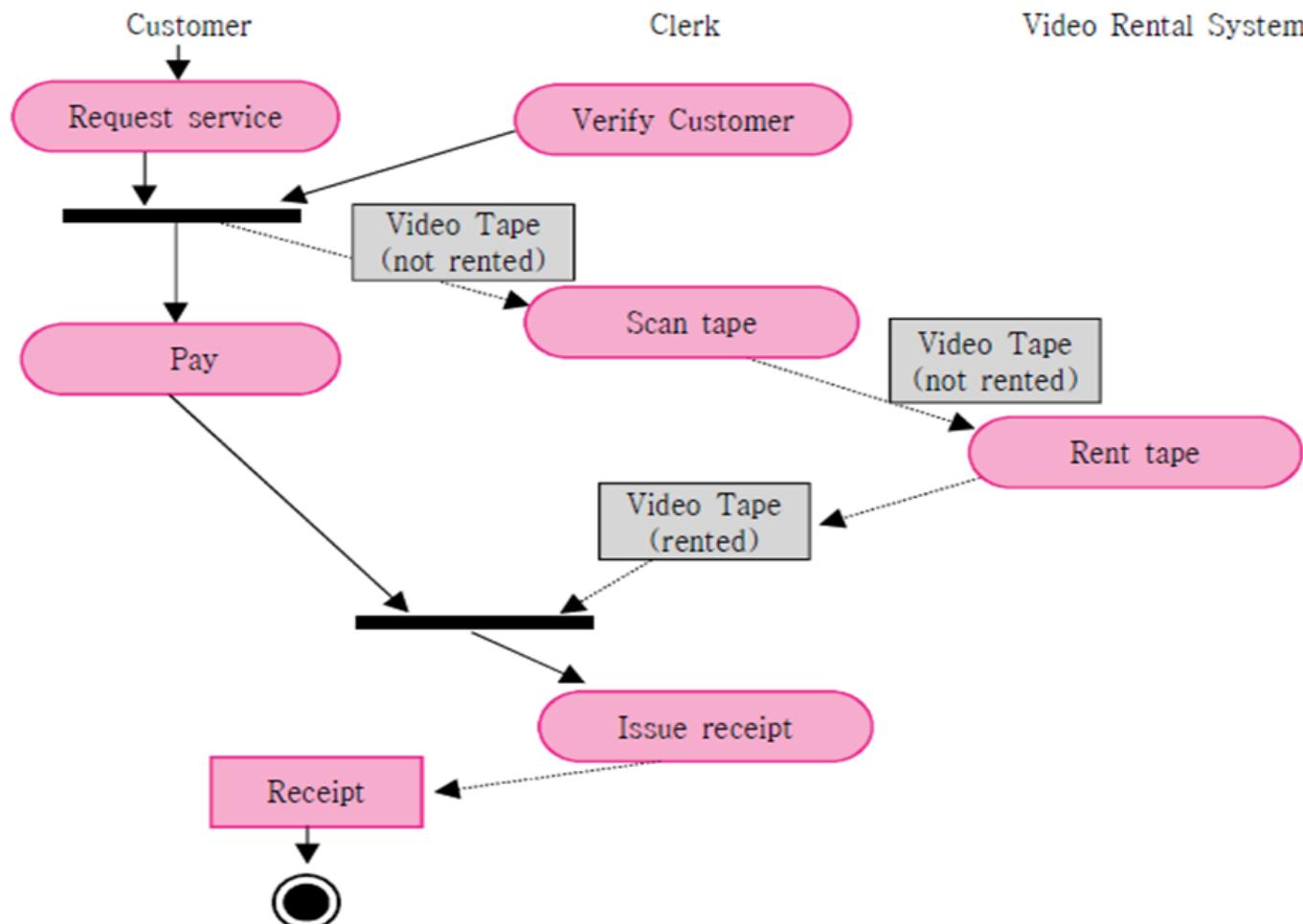
- 시스템의 동적인 부분을 모델링하는 목적으로 사용
- 액티비티 사이의 제어흐름을 보여 주는 일종의 흐름도
- 액티비티 다이어그램의 사용
 - 시스템 수준에서 시스템과 상호작용하는 각 액터의 관점에서 모델링
 - 메소드 수준에서, 복잡한 오퍼레이션의 수행흐름을 표현

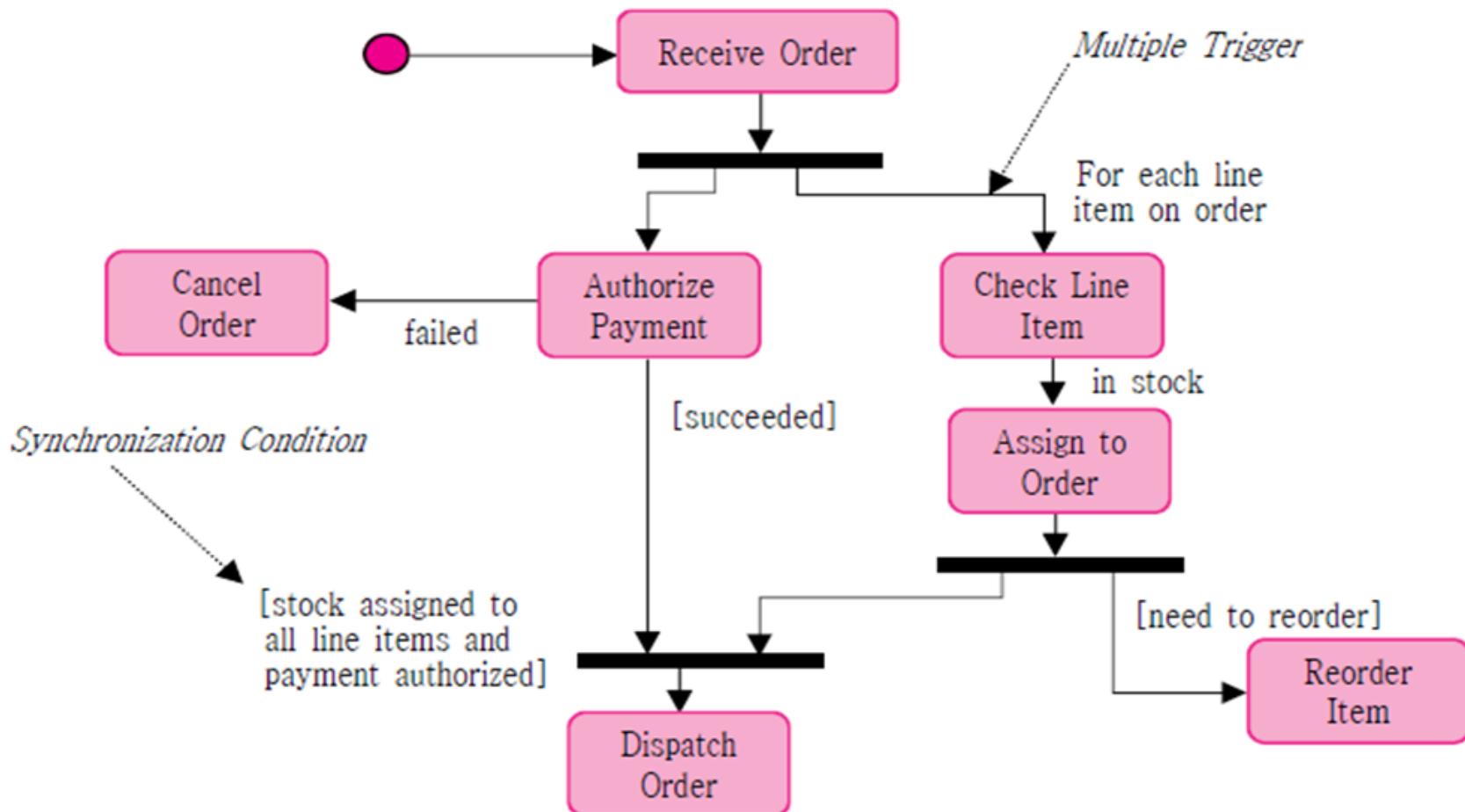


액티비티 다이어그램 (2/3)

모델링 (Modeling)

- 시스템 수준의 액티비티 다이어그램
(예제: 비디오 대여 시스템의 비즈니스 프로세스)



 메소드 수준의 액티비티 다이어그램 (예제: 주문 처리 시스템)



In this chapter ...

모델링 (Modeling)

5.1 객체지향 개념

5.2 UML

5.3 정적 모델링

5.4 동적 모델링

5.5 모델링 도구

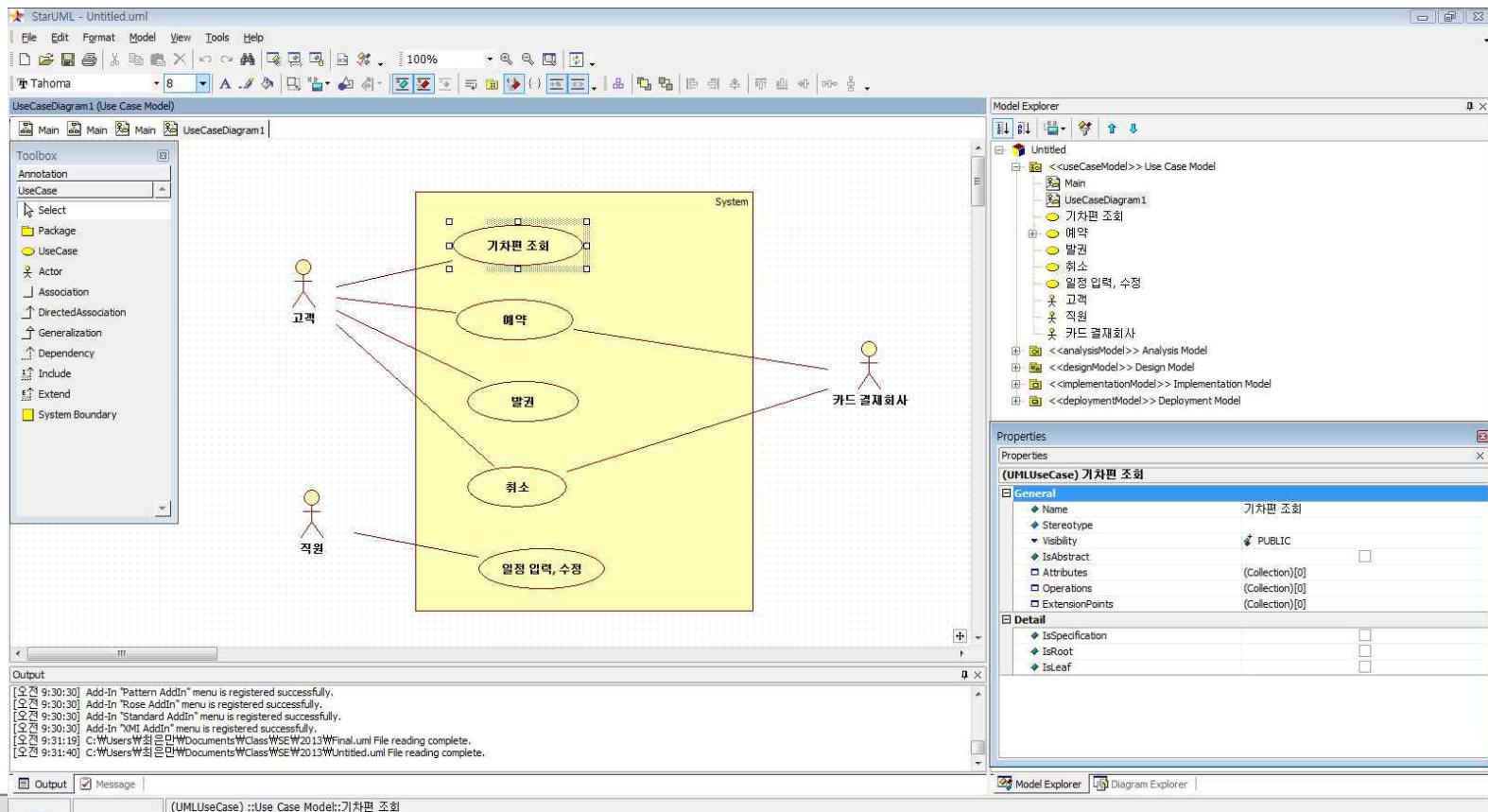


많이 알려진 모델링 도구

- IBM의 Rational Modeler
- 마이크로소프트의 Visio, ArgoUML, StarUML, NetBeans UML 플러그인



StarUML – 오픈소스로 많이 사용되고 있음





프로젝트 관리

- 다이어그램과 코드를 모델별로 그루핑하여 프로젝트 별로 보관하고 불러냄
- 서브시스템이나 패키지별로 그루핑
- 프로젝트의 내용은 주로 하나의 파일로 관리하지만 여러 팀으로 나누어 작업하기 위하여 유닛별로 쪼개어 관리



코드 및 문서 생성

- 사용자가 템플릿을 정의하고 템플릿을 이용하여 문서를 일괄 생성
- StarUML의 경우 텍스트 문서만이 아니라 엑셀, 프레젠테이션 파일도 만듬



모델 검사

- 연관의 이름이 고유한지, 순환적 상속이 없는지, 동일한 속성이 없는지 등의 규칙을 가지고 모델을 자동 검증



In this chapter ...

모델링 (Modeling)

5.1 객체지향 개념

5.2 UML

5.3 정적 모델링

5.4 동적 모델링

5.5 모델링 도구



Homework #3

계획 (Planning)