

- 동적 모델링 : 클래스들의 상호작용이나 클래스의 상태 변화 등 시스템 내부의 동작을 모델링
- 동적 모델링의 세가지 다이어그램
  - 인터랙션 다이어그램 : 사용 사례를 실현시키기 위하여 내부 클래스들 이 어떻게 협동하는지 나타내는 그림  
==>시퀀스(순서) 다이어그램, 커뮤니케이션 다이어그램
  - 상태 다이어그램 : 복잡한 객체의 상태 변화를 나타낸 것
  - 액티비티 다이어그램 : 절차나 작업의 흐름을 나타낸 것

# 시퀀스 다이어그램의 요소

- 시스템의 동작을 정형화하고 객체들의 메시지 교환을 울타리 형태로 시각화하여 나타낸 것
- 참여객체(participating object)

<u>:Person</u>	Person 클래스의 이름 없는 인스턴스
<u>kim:</u>	이름 없는 클래스의 kim이라는 인스턴스
<u>kim:Person</u>	Person 클래스의 kim이라는 인스턴스
<u>:Person</u>	Person 클래스의 인스턴스의 모임
<<jsp>> <u>LoginPage:</u>	스테레오타입 객체

그림 5.30 ▶ 객체의 표현

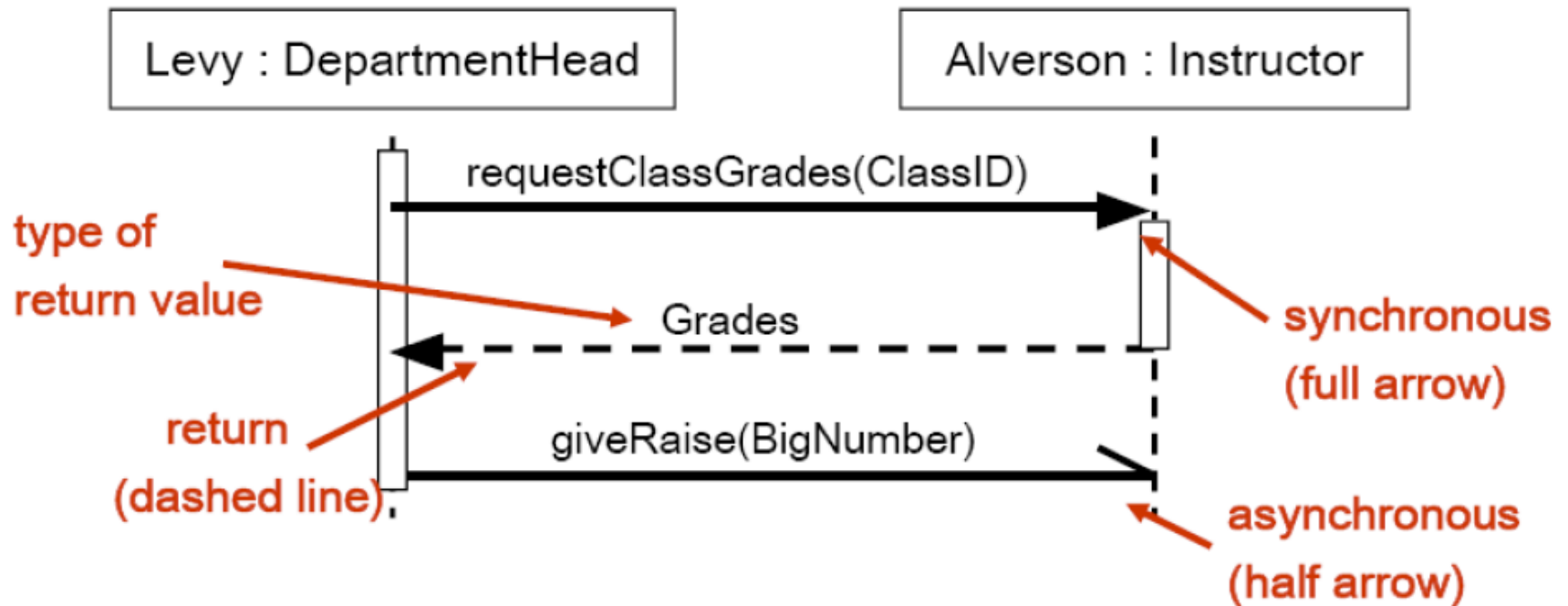
method name

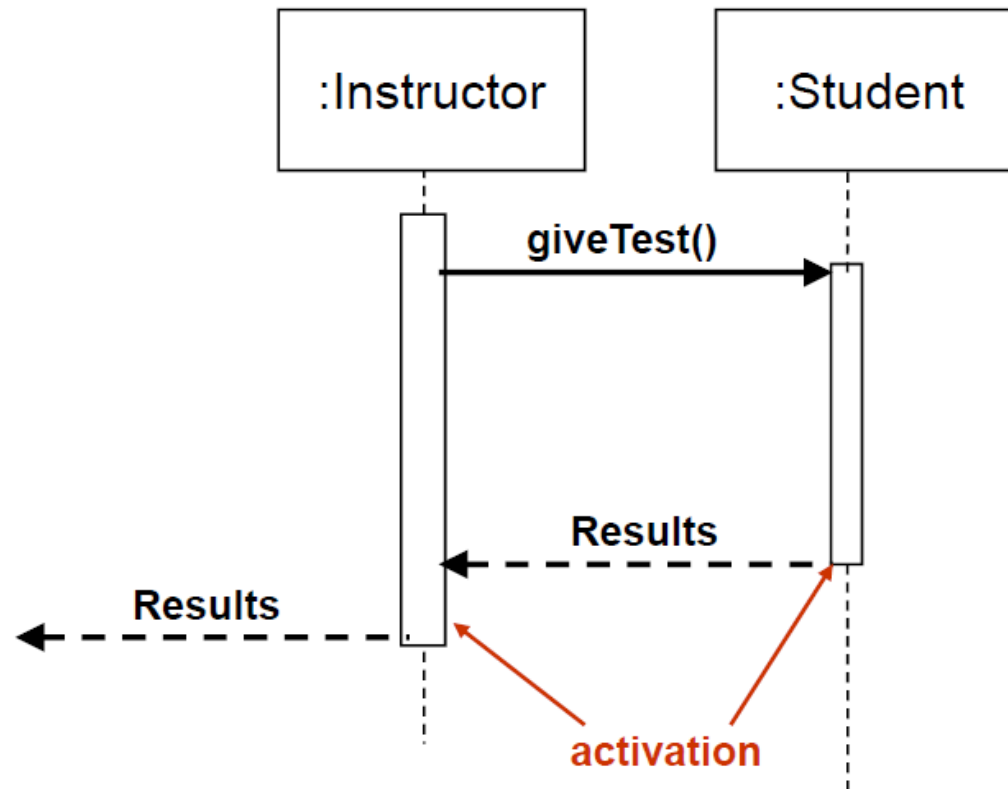
Alverson : Instructor

requestClassGrades(ClassID)

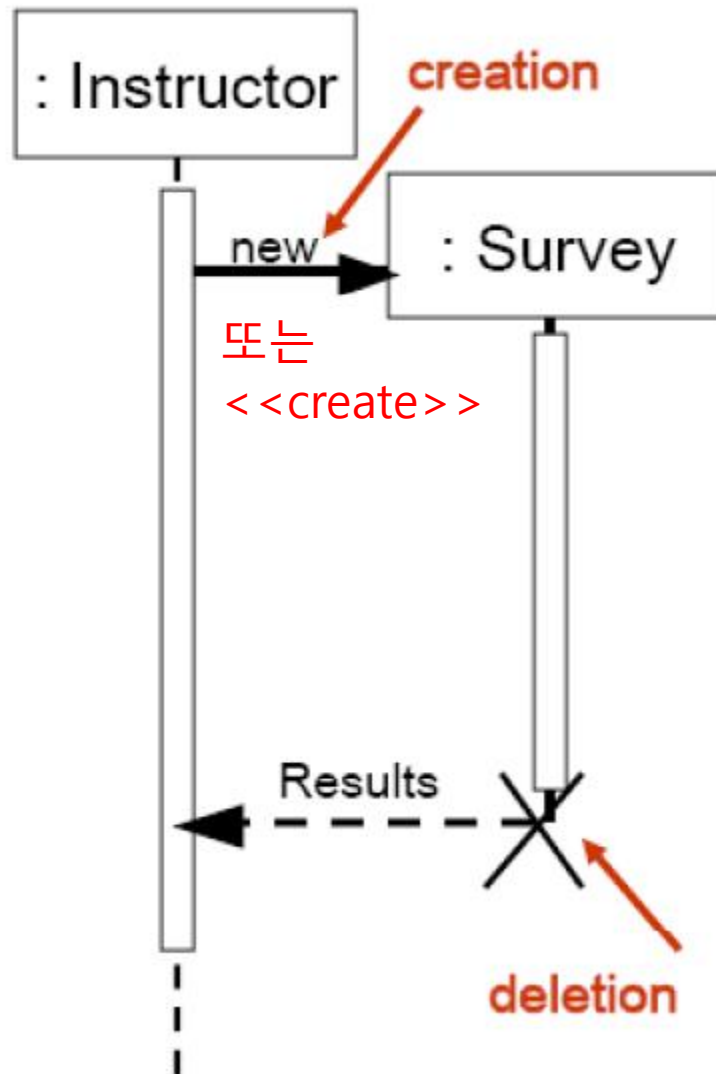
method argument types

생명선





객체의 코드가 실행되고 있거나 다른 객체의 메소드가 종료되기를 기다림.



found message

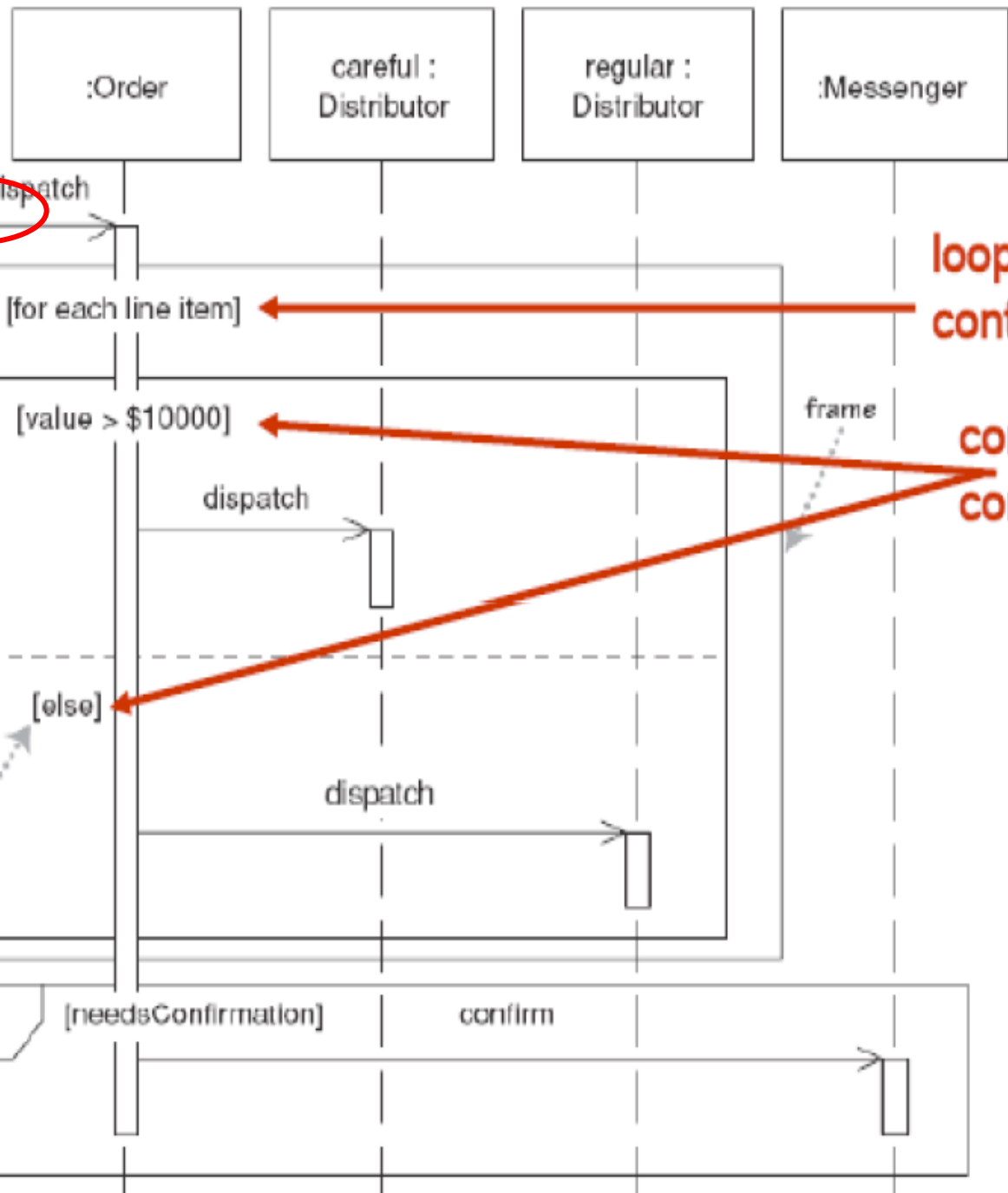
iteration frame

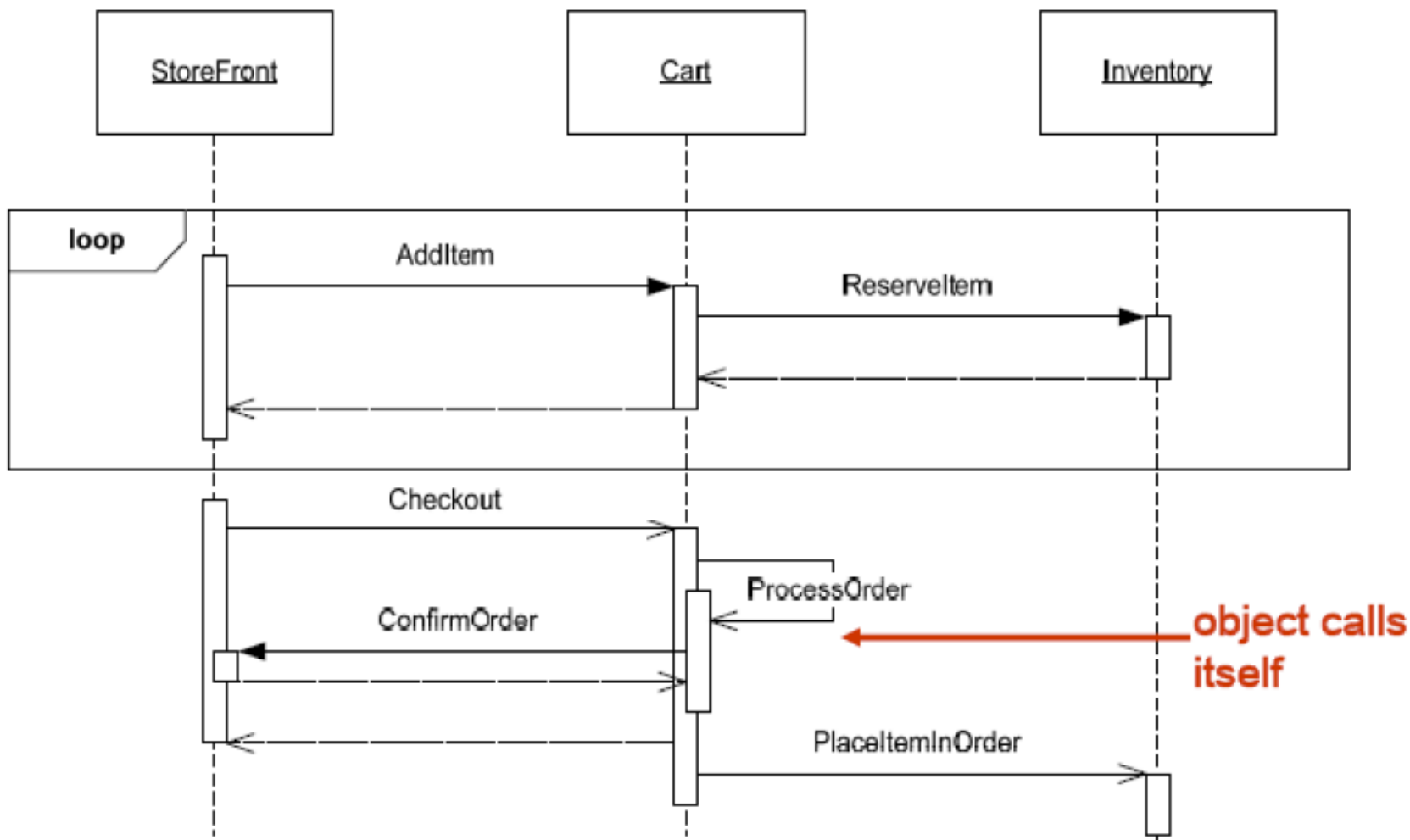
if/then/else  
frame

if/then  
frame

loop  
control

condition  
control

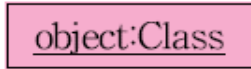
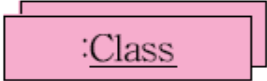



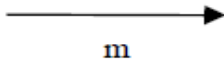
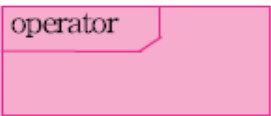






# 시퀀스 다이어그램의 요소

표 5.3 ▶ 시퀀스 다이어그램의 기본 요소

요소	표현방법	의미	연결
객체		- 특정 클래스의 객체 - 객체의 모임	라이프라인 사이에 활성막대와 연결됨
객체집합		- 라이프라인 위에 위치하며 콜론 앞은 객체의 이름, 뒤는 클래스의 이름	
라이프라인		객체가 시스템에 존재하나 아직 실행되 지는 않음을 의미.	객체를 활성막대와 연 결시키며 두 개의 이 웃 라이프라인을 연결
활성막대		시스템에 존재하는 메소드가 막대의 길 이만큼 실행됨을 의미. 점선은 라이프라 인임.	객체와 연결됨. 라이 프라인과 연결됨
객체 소멸		라이프라인 맨 위에 연결된 객체가 소 멸됨을 의미.	
메시지 호출		한 객체에서 다른 객체로 메시지를 보 냄을 의미. 즉 함수가 호출됨	상호작용하는 두 객체 를 연결함
프레임		시퀀스 다이어그램의 일부로 반복 또는 택일 구조의 묶여진 조각.	

# 시퀀스 다이어그램 작성

- **Step 1.** 참여하는 객체를 파악
- **Step 2.** 파악한 객체를 X축에 나열하고 라이프라인을 그음
- **Step 3.** 사용사례에 기술된 이벤트 순서에 따라 객체의 메시지 호출

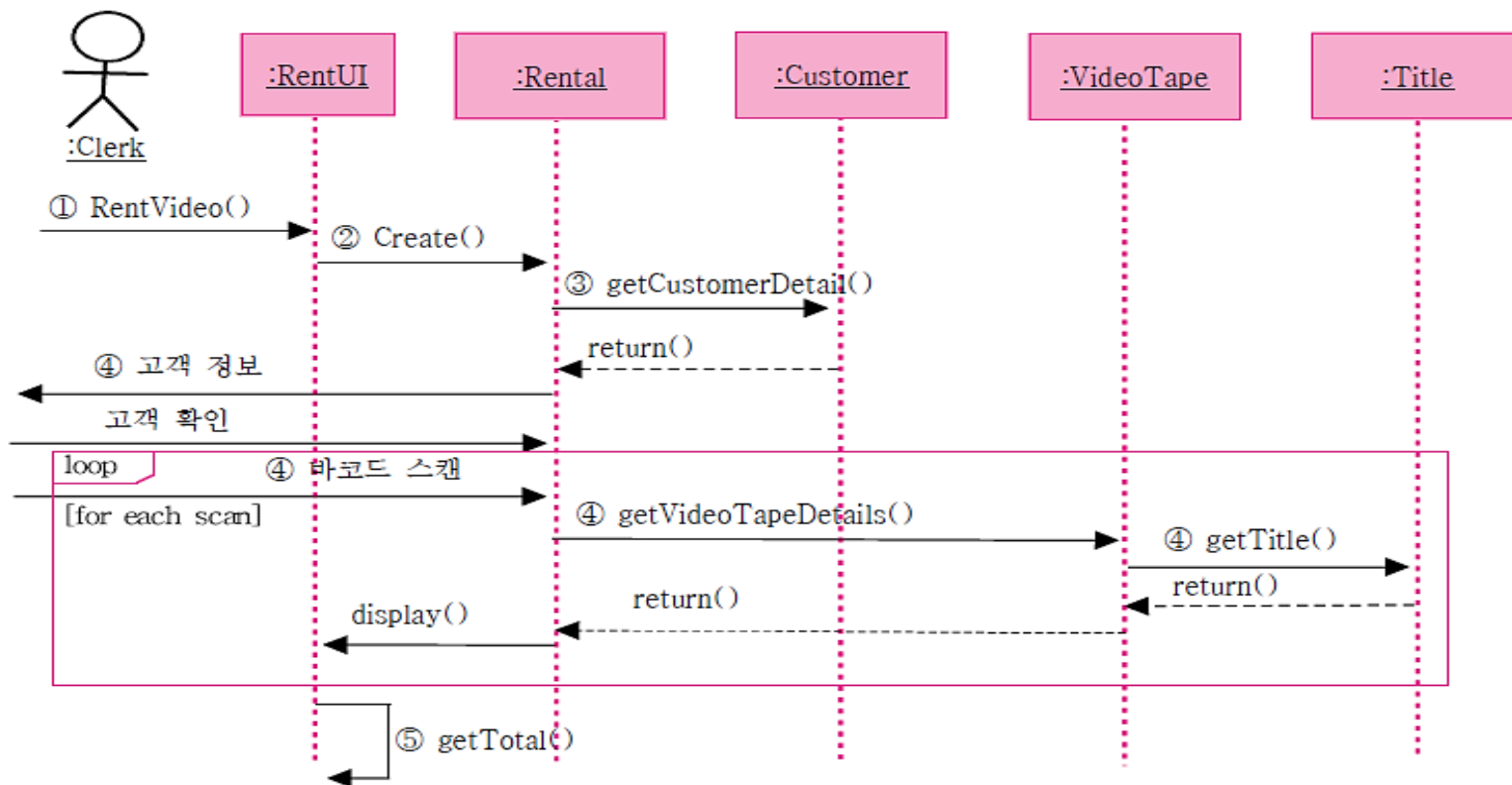
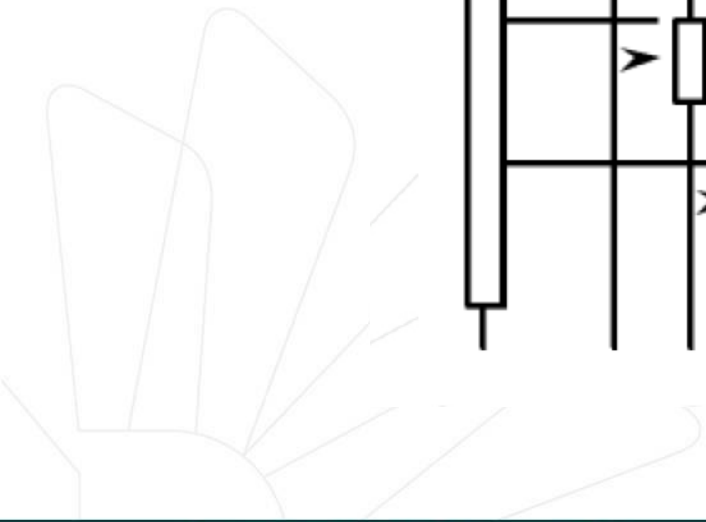
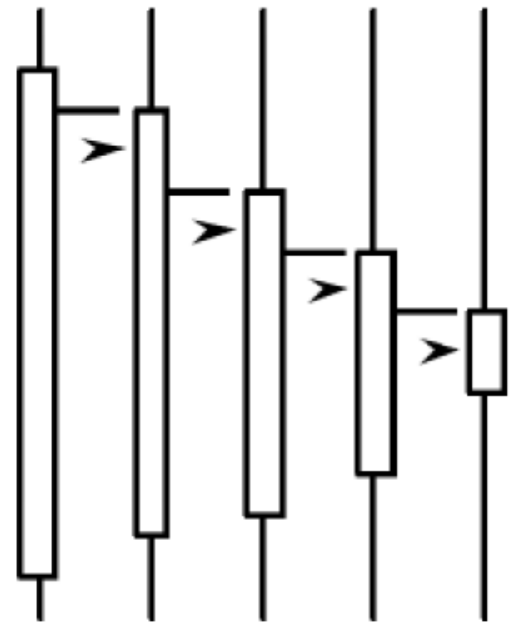
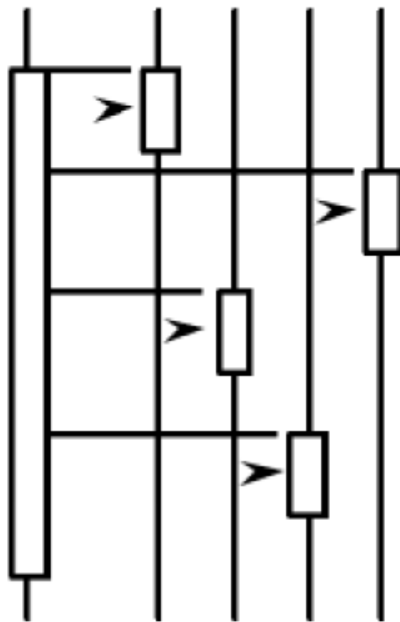


그림 5.32 ▶ 시퀀스 다이어그램(비디오 대여 사용 사례)

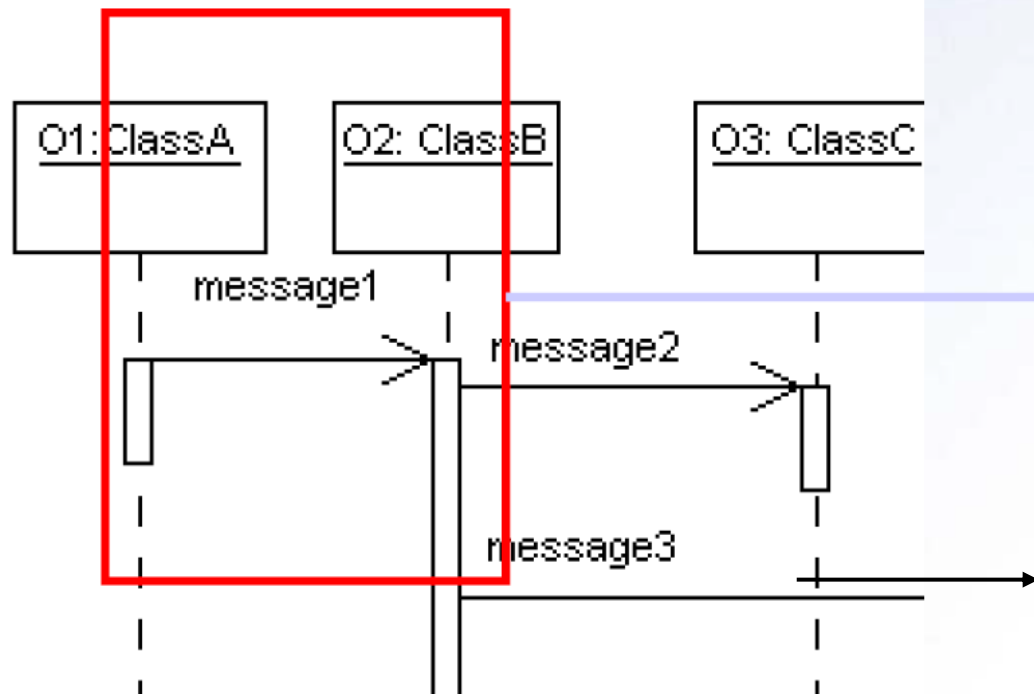
- 시스템의 제어 흐름의 형태는?
  - 중앙 집중형
  - 분산형



# 순서 다이어그램 과 코딩

- 순서 다이어그램을 코딩하는 방법
  - 메시지는 메소드의 호출로 코딩. 객체의 생성은 생성자(constructor)를 호출
  - 메시지를 받는 객체의 클래스 안에 메소드 구현
  - 분기구조(alternative frame)는 if-then-else 문장과 같은 조건문으로 구현
  - 반복구조(loop frame)는 while 문
  - 병렬구조 (parallel frame)는 thread로 구현

- 메시지가 호출 당하는 클래스 안에 메소드 구현



ClassB

```
{
  ClassC o3;
  ClassD o4;
  method1(...)
  {
    o3.method2(..);
    o4.method3(..);
  }
}
```

# Communication Diagram (협업 다이어그램)

- 위임 과 전달 을 네트워크 형태로 표현 (메시지 순서 있음)

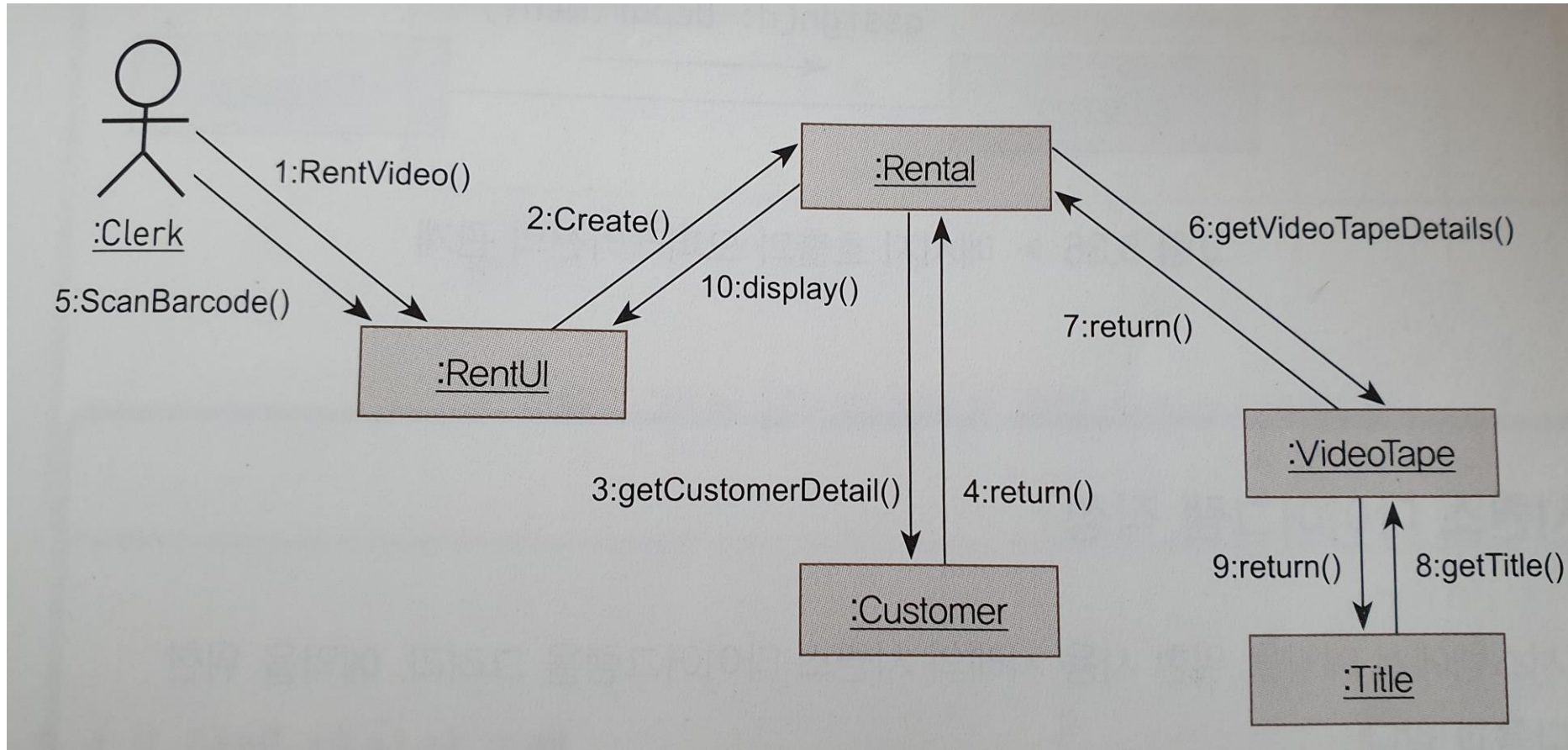



그림 5.35

# 상태 다이어그램

## ● 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

표 5.4 ▶ 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 변화 됨
레이블	e[exp] / a1; a2	이벤트(e)가 발생하고 <b>가드조건(exp)</b> 이 참이면 트랜지션이 일어나고 액션 a1, a2가 실행됨

# 상태 다이어그램

- 예> 비디오 대여 시스템의 비디오테이프 객체의 상태 변화 모델링

**Action:**

**Entry** - 상태에 진입할 때 액션이 구동됨

**Exit** - 상태에서 빠져나가기 바로 전에 액션이 실행됨

**Event** - 이벤트에 대한 반응으로 액션이 실행됨

**Do** - 상태 안에서 액션이 수행됨

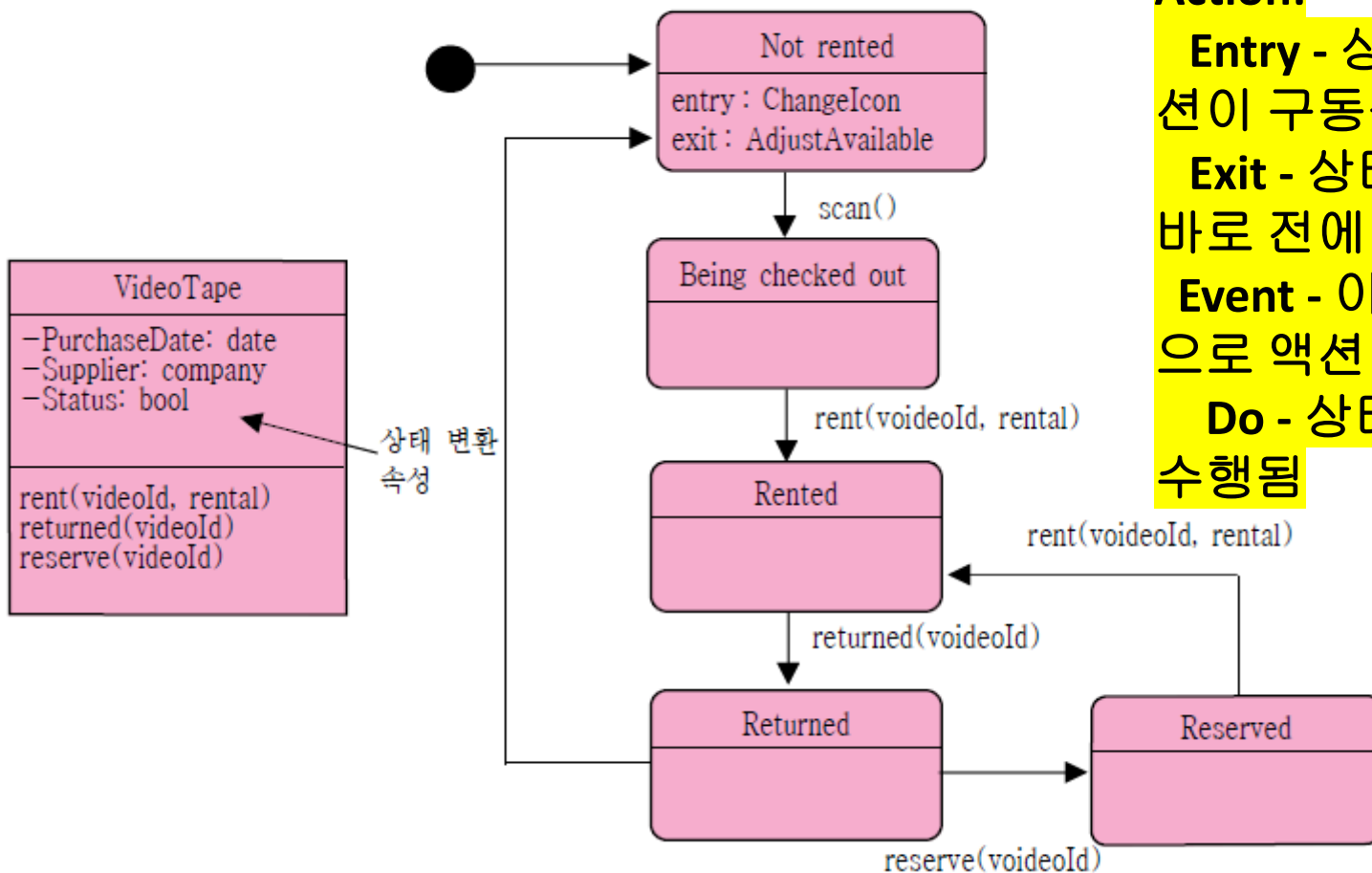
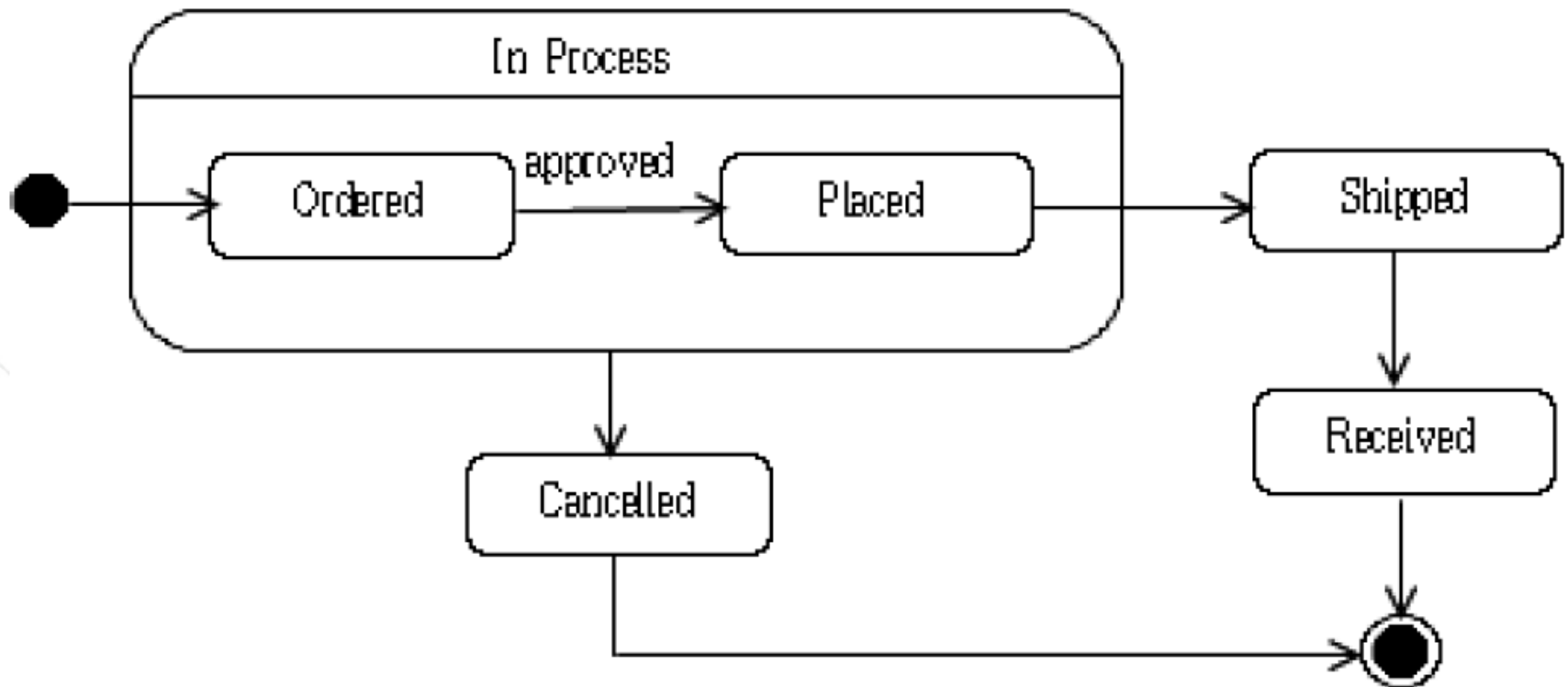


그림 5.37 ▶ Video의 상태 다이어그램



- 복합 상태: 계층적 분할

Order 상태



# 상태 다이어그램

- 상태 다이어그램을 모델링하기에 적합한 속성의 조건
  - 속성의 값으로 가질 수 있는 종류가 적어야 함
  - 속성의 값에 따라 허용되는 오퍼레이션이 제한되어야 함

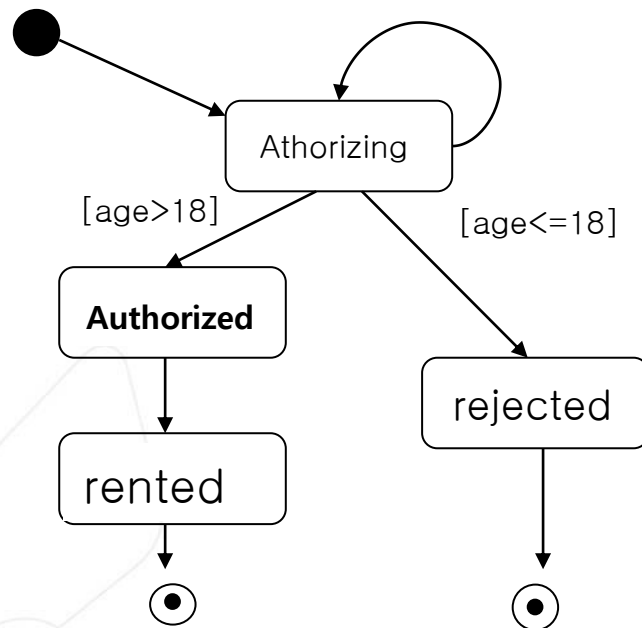


그림 5.33 상태 다이어그램의 예

# 상태 다이어그램의 구현

- 상태 다이어그램을 클래스로 매핑
- 상태정보를 저장하기 위한 속성 추가
- 이벤트는 메소드로 상태 변화나 이벤트의 액션은 메소드 안에 탑재

자판기 제어 객체

```
class VendingMachineControl
{
    int _state;
    float _amount, _price;
    static final int WaitingCoin = 1;
    static final int WaitingSelection = 2;
    static final int DispensingSoftDrink = 3;
    static final int DispensingChange = 4;
    static final int EjectingCoins = 5;
```

상태 천이는 메소드 로.

가드(guard)는 메소드 안의 조건 체크

