

# 프로젝트 방법론

---

## UML 다이어그램

빅데이터 분석가 과정

# 목차

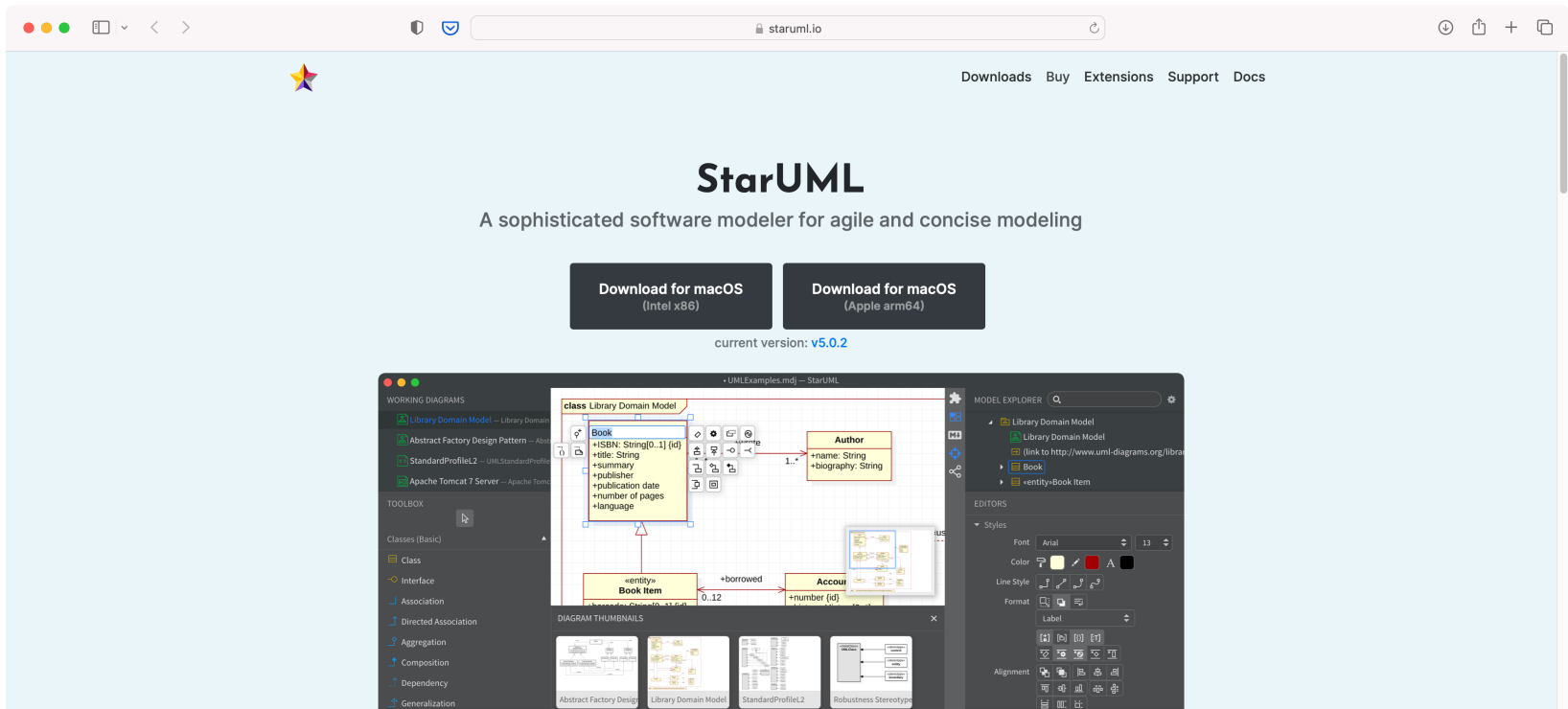
---

- UML 다이어그램
  - Usecase diagram
  - Sequence diagram
  - Class diagram

# UML 다이어그램 툴

## ■ StarUML

- <https://staruml.io>



## Key Features

# 무료 웹 UML 작성 툴

## ■ GitMind

- <https://gitmind.com/kr/>

The screenshot displays the GitMind website interface. At the top, there's a navigation bar with the GitMind logo, a search bar, and links for '템플릿' (Templates), '앱 다운로드' (Download App), and '나의 문서' (My Documents). On the right, there are buttons for '회원 가입' (Sign Up), '지원' (Support), and '로그인 / 가입' (Login / Sign Up).

The main content area is divided into a sidebar and a grid of diagram templates. The sidebar on the left contains a search bar and a list of categories: '모든 템플릿' (All Templates), 'Mind Map', 'Education', 'Product', 'Engineering', 'Sales', 'Operations', 'How to', 'Others', 'Flowchart', 'Analysis', 'Organization', 'UML' (highlighted), and 'Lane'.

The grid of templates includes:

- A large orange box with a plus sign and the text '새로운 플로우차트' (New Flowchart).
- 'Class Diagram-2': A complex class diagram with multiple classes and relationships.
- 'Use case diagram-3': A use case diagram for an 'XXX System' with four packages (Name-1 to Name-4) and multiple use cases.
- 'Use case diagram-2': A use case diagram for an 'XXX System' with multiple use cases and actors.
- 'Use case diagram-1': A use case diagram for an 'XXX System' with multiple use cases and actors.
- 'Class Diagram': A detailed class diagram with many classes and relationships.
- 'Order processing system': A use case diagram for an order processing system involving actors like 'Online User', 'credit card company', 'Deliveryman', and 'Warehouse clerk'.

# UML

---

## ■ UML (Unified Modeling Language)

- 프로그램 설계를 표현하기 위해 사용하는 표기법
- 프로그램 언어가 아닌 기호와 도식을 이용하여 표현하는 방법

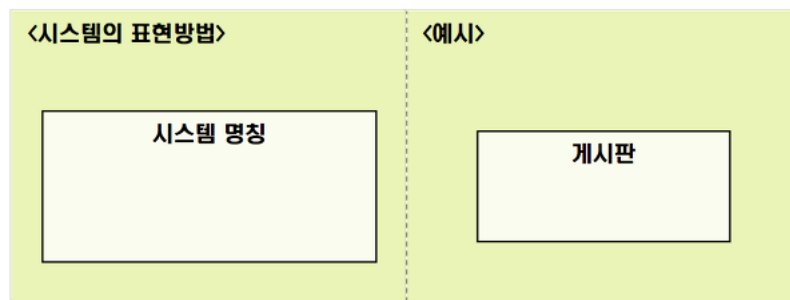
## ■ UML 다이어그램 종류

- 구조적 다이어그램
  - 클래스 다이어그램(Class Diagram): 클래스 명세와 클래스 간의 관계를 표현
  - 객체 다이어그램(Object Diagram): 인스턴스 간의 연관 관계 표현
  - 패키지 다이어그램(Package Diagram): 패키지 간의 연관 관계 표현
  - 컴포넌트 다이어그램(Component Diagram): 파일과 데이터베이스, 프로세스와 스레드의 소프트웨어 구조를 표현
- 행위 다이어그램
  - 유스케이스 다이어그램(Usecase Diagram): 시스템이 제공하는 기능과 이용자의 관계 표현
  - 액티비티 다이어그램(Activity Diagram): 일련의 처리에 있어 제어의 흐름을 표현
  - 시퀀스 다이어그램(Sequence Diagram): 인스턴스의 상호작용을 시계열로 표현

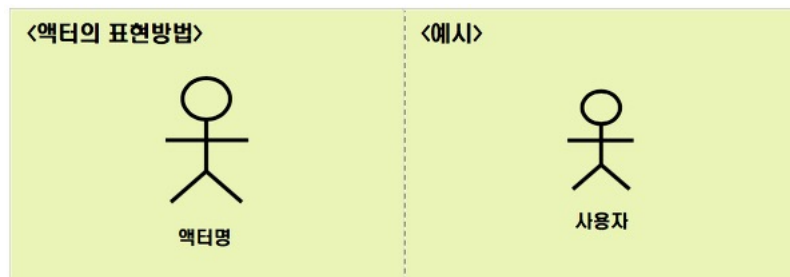
# Usecase Diagram

## ■ Usecase diagram 구성 요소

- 1) 시스템: 만들고자 하는 프로그램
  - 사각형 틀로 시스템 명칭을 안쪽 상단에 작성

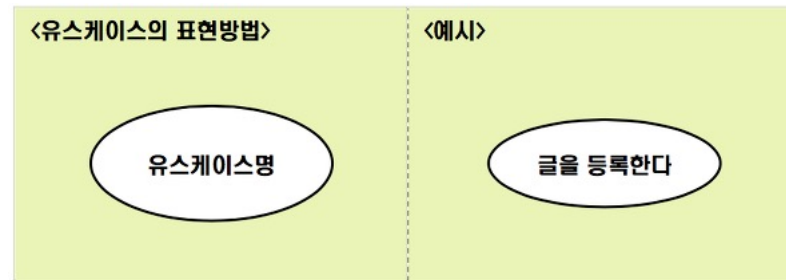


- 2) 액터(Actor)
  - 시스템의 외부에 있고 시스템과 상호작용을 하는 사람
  - 시스템에 정보를 제공하는 또 다른 시스템

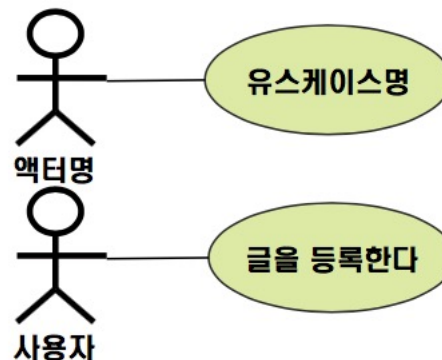


# Usecase Diagram

- 3) Usecase
  - 사용자 입장에서 바라본 시스템 기능
  - 시스템이 actor에게 제공해야 되는 기능으로 시스템의 요구사항을 나타냄
  - 타원으로 표시하고, 안쪽에 유스케이스명을 작성



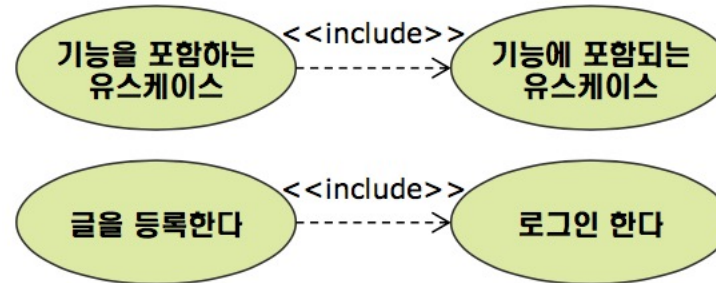
- 4) 관계 (relation): actor와 usecase 사이의 의미있는 관계를 나타냄
  - **연관관계(association)**
    - actor와 usecase간의 상호작용이 있음을 표시 (실선)



# Usecase Diagram

## – 포함 관계(include)

- 하나의 유스케이스가 다른 유스케이스의 실행을 전제로 할 때 형성



## – 확장 관계(extend)

- 기존 유스케이스에 부가적으로 추가된 기능을 표현

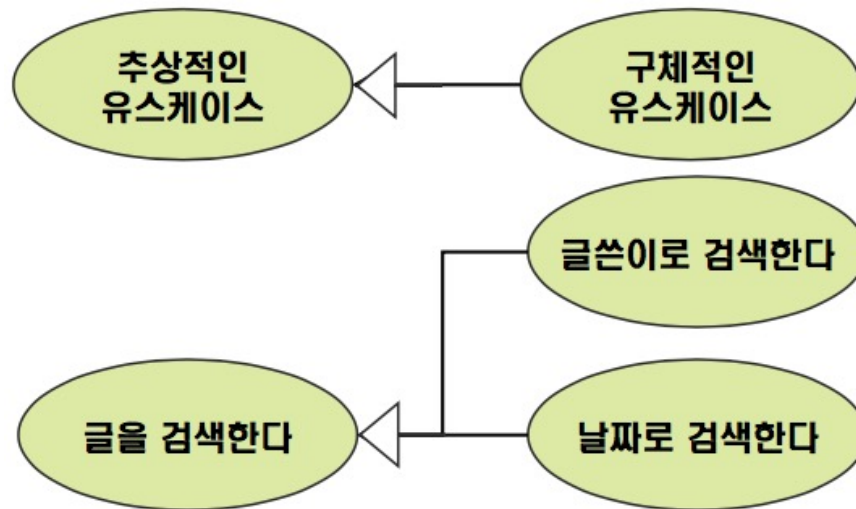




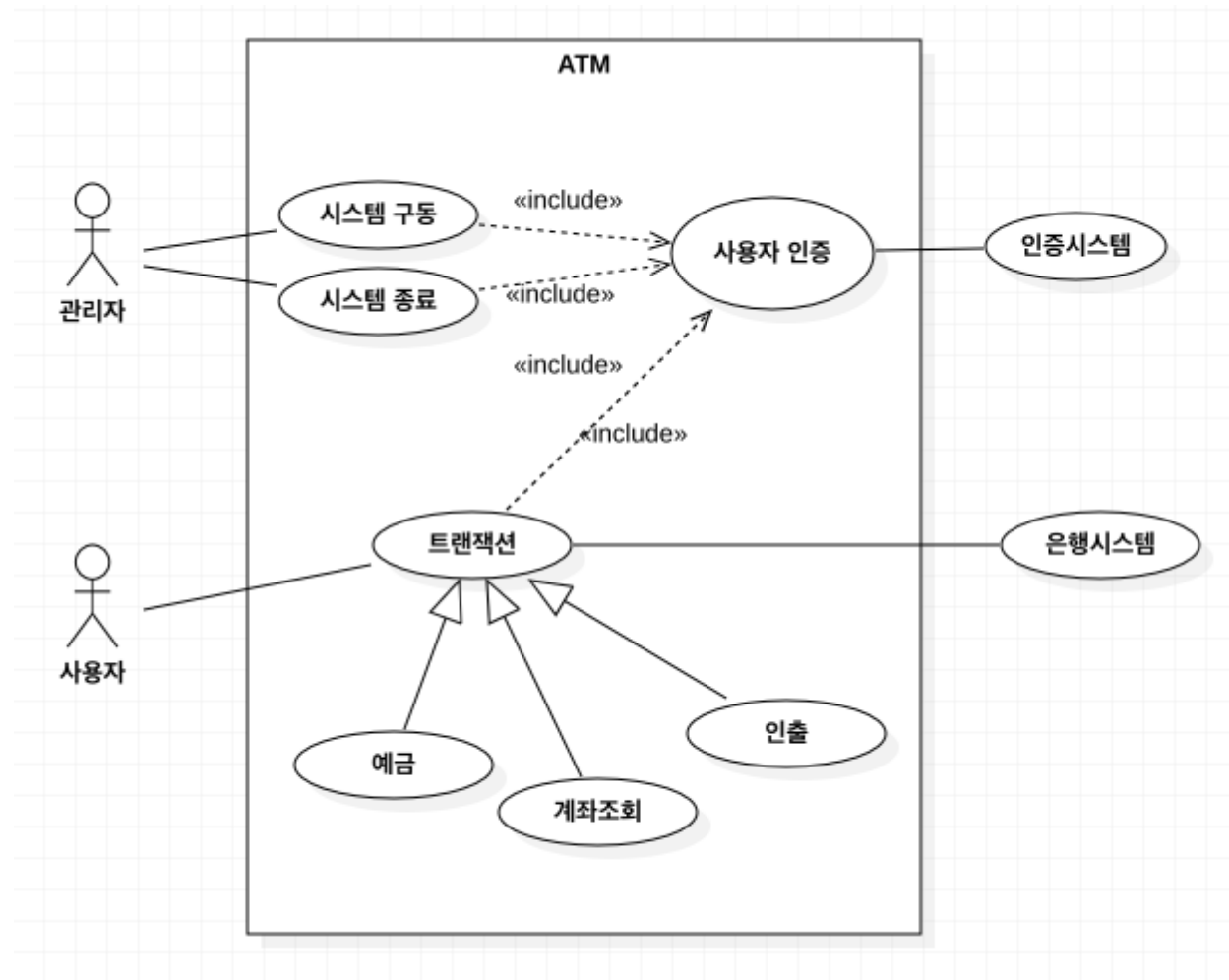
# Usecase Diagram

---

- 일반화 관계 (generalization)
  - 개별적인 것에서 공통 특징을 뽑아 이름을 붙인 것



# Usecase Diagram 예제



# 시퀀스 다이어그램

## ■ 시퀀스 다이어그램 (Sequence Diagram)

- 특정 행동이 어떠한 순서로 다른 객체와 상호 작용하는지 표현
- 메시지 순서에 초점을 맞춰 나타낸 것
- 어떤 작업이 객체 간에 발생하는지 시간 순서에 따라 쉽게 파악할 수 있음





## ■ 구성 요소

- 객체
  - 객체는 메시지를 보내고 받는 주체
  - ‘객체명:클래스명’으로 나타냄
- 객체 생명선
  - 객체의 생존 기간을 나타내며 X 표시는 객체가 소멸하는 시점
  - 위에서 아래로 내려가는 점선으로 나타냄
  - 생명선을 따라 나타나는 직사각형은 활성 구간
    - 객체의 메서드가 실행되고 있음을 나타냄
  - 구간의 길이는 메서드의 실행 시간을 나타냄
  - 활성 구간은 반드시 존재하는 것은 아니며 생략 가능

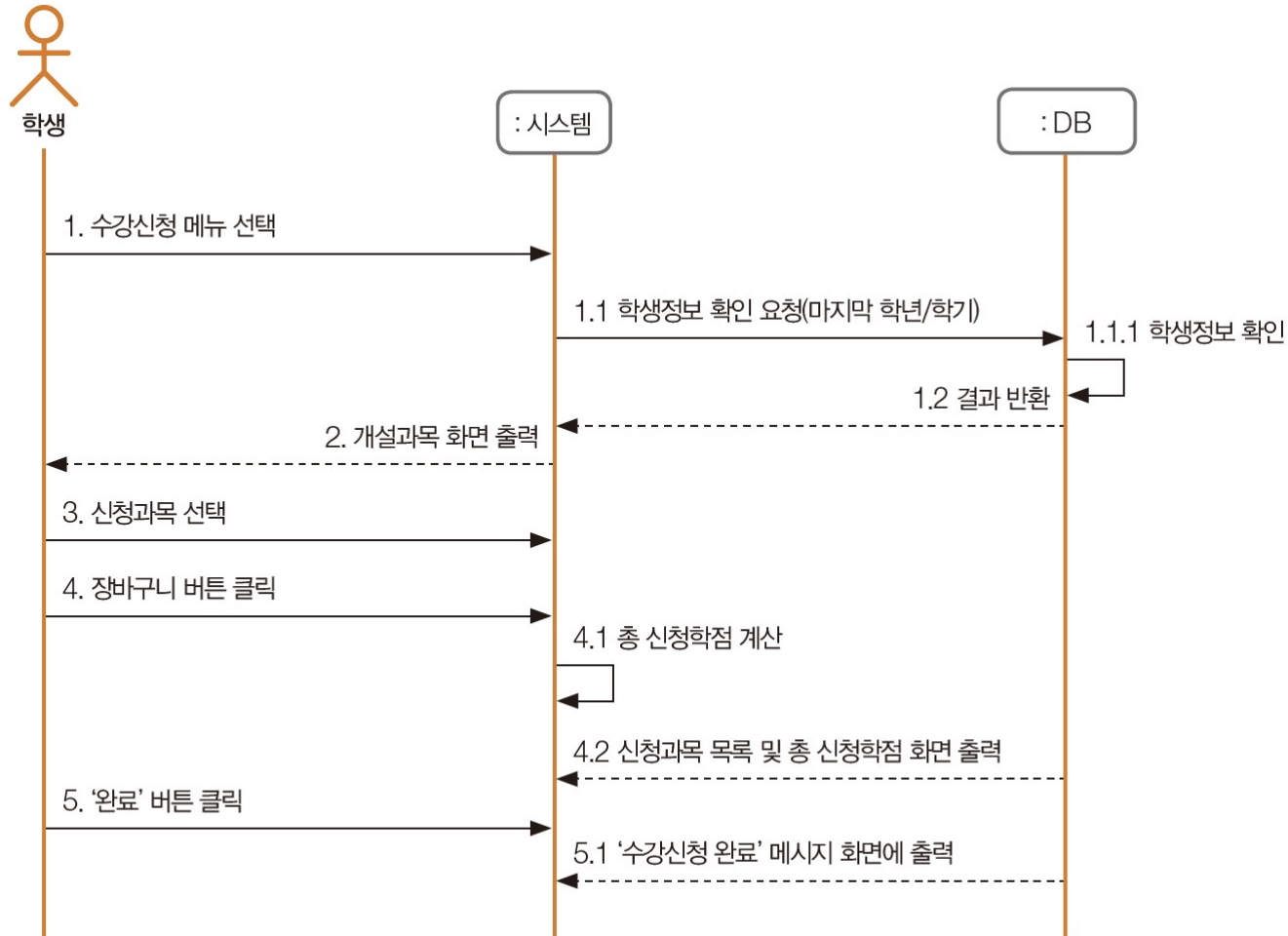


# 시퀀스 다이어그램

---

- 메시지
  - 객체와 객체의 상호작용을 표현하는 것으로 화살표를 이용
  - 화살표 위에는 수신 객체의 함수명을 명기
- 동기 메시지
  - 송신 객체가 수신 객체에 서비스를 요청(메서드 호출)하면 메서드 실행이 완료될 때까지 기다림
  - 실선과 속이 채워진 삼각형으로 나타냄 
- 비동기 메시지
  - 송신 객체가 수신 객체에 서비스를 요청(메서드 호출)할 때 메서드의 실행과 상관없이 다음 작업을 수행
  - 수신 객체로부터의 반환도 기다리지 않는다.
  - 일반 화살표 모양으로 나타냄 
- 재귀 메시지
  - 수신 객체가 자신의 메서드를 호출할 때 사용 
- 답신 메시지
  - 호출 메서드의 결과를 반환할 때 사용
  - 점선과 속이 채워진 삼각형으로 나타냄 

# 시퀀스 다이어그램 예제



수강 신청 sequence diagram

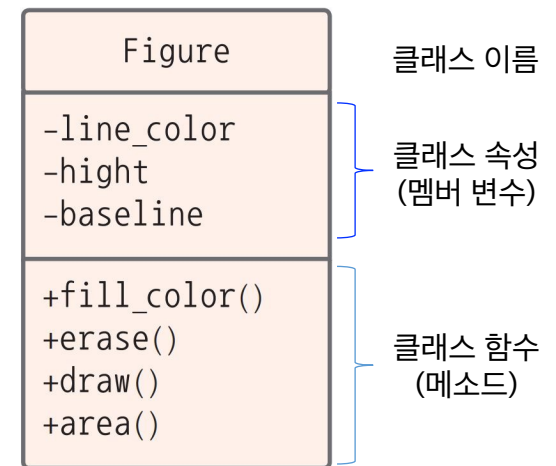
# 클래스 다이어그램

## ■ 클래스 다이어그램

- 소프트웨어의 기본 구성 단위인 시스템에서 사용하는 클래스를 정의
- 클래스들이 서로 어떻게 연결되어 있고 어떤 역할을 하는지 표현

## ■ 클래스

- 데이터(속성)와 메서드를 묶어 놓은 것
- 첫 번째 칸에는 클래스 이름
- 두 번째 칸에는 클래스의 속성
- 마지막 칸은 클래스가 제공하는 기능인 메서드를 나타냄



# 클래스 다이어그램

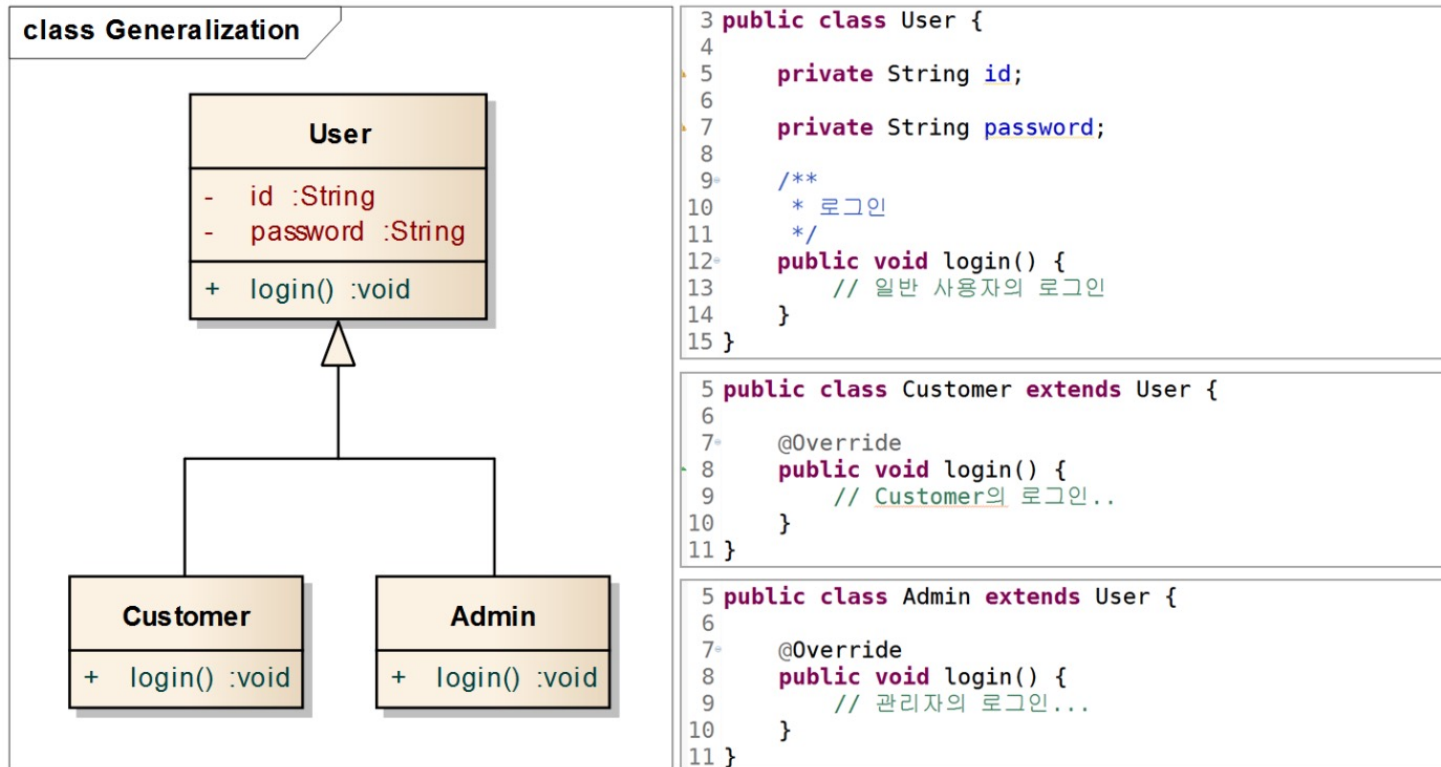
## ■ 클래스간의 관계

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

# 클래스 다이어그램

## ■ 일반화 (generalization)

- 부모 클래스와 자식 클래스의 상속 관계
- 부모 클래스의 내용을 자식 클래스에서 구체화



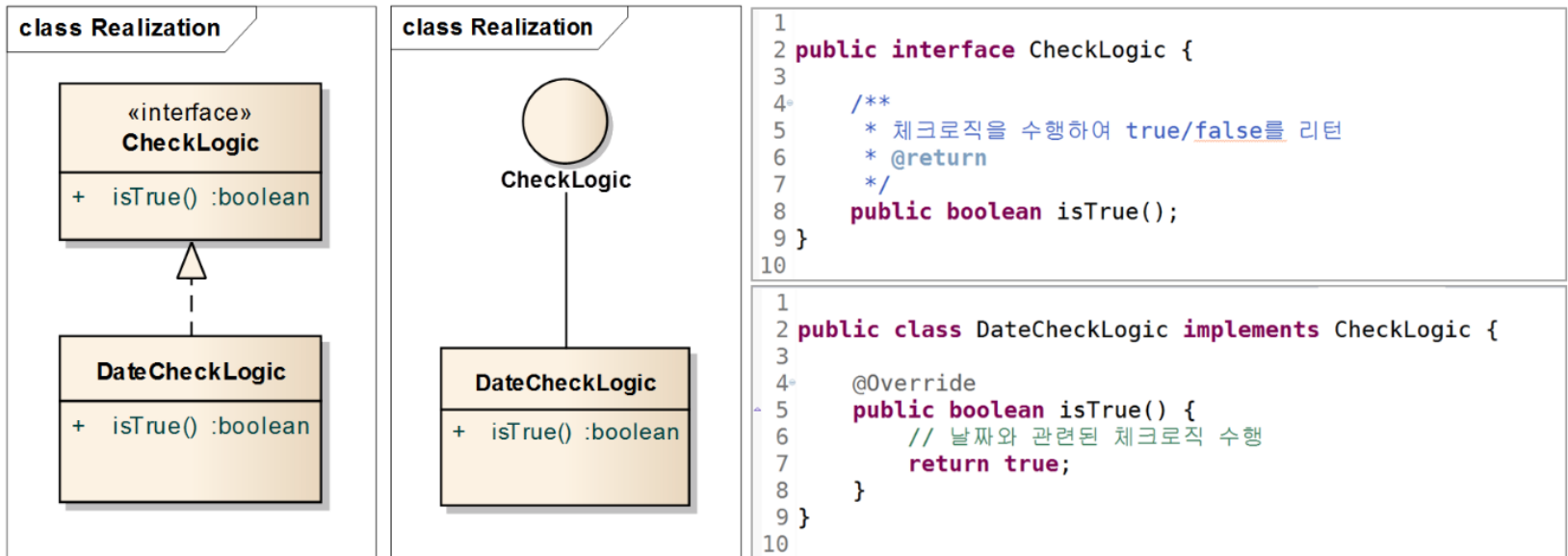
– Customer 클래스와 Admin 클래스는 User 클래스(부모 클래스)를 상속



# 클래스 다이어그램

## ■ 실체화 (Realization)

- 인터페이스(interface)의 메소드를 오버라이딩하여 실제 기능 구현
- 빈 화살표와 점선으로 표현

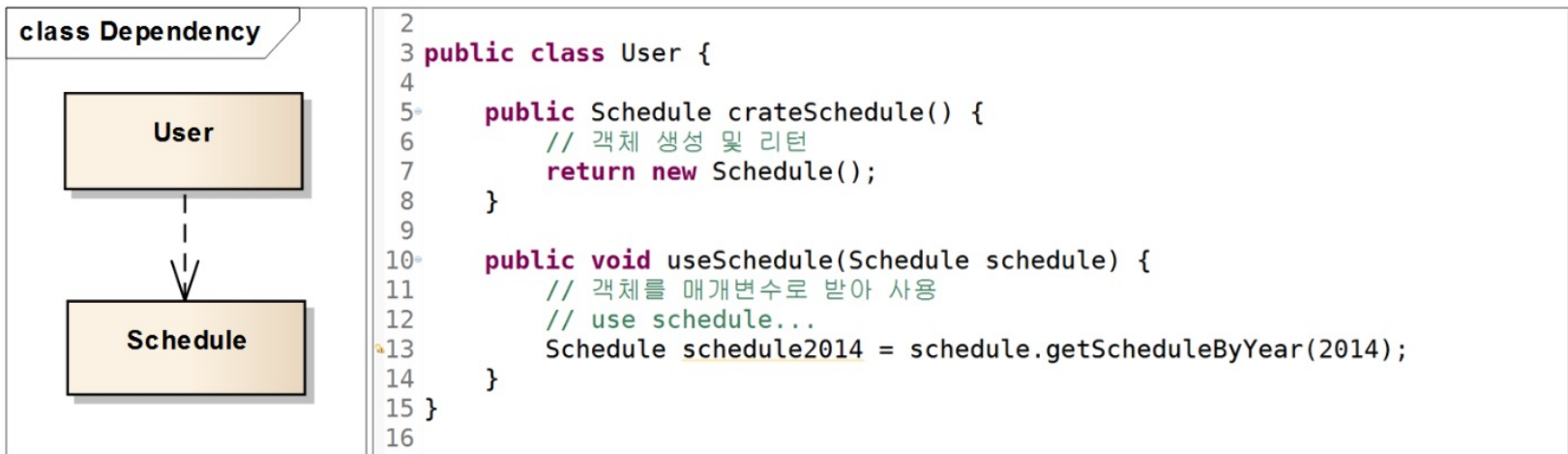


– DateCheckLogic 클래스에서 인터페이스 CheckLogic의 메소드의 기능을 구현

# 클래스 다이어그램

## ■ 의존 (Dependency)

- 어떤 클래스가 다른 클래스를 참조
- 클래스 다이어그램에서 가장 많이 사용되는 관계

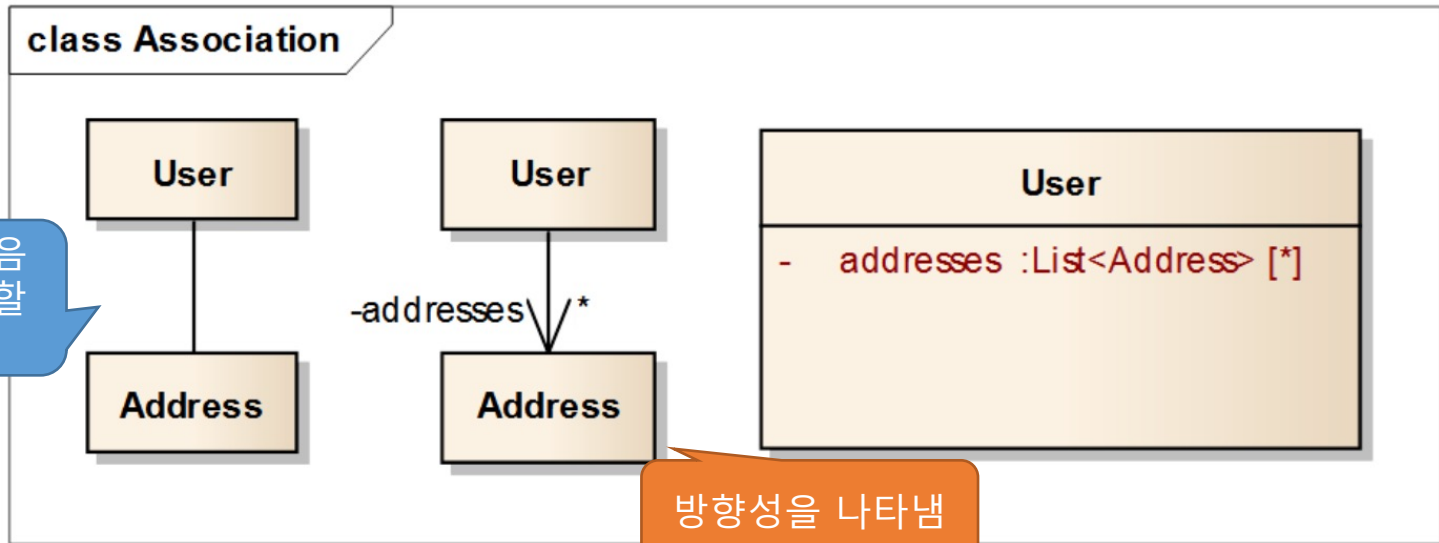


– User 클래스 내부에서 Schedule 클래스의 객체를 참조

# 클래스 다이어그램

## ■ 연관 (Association)

- 하나의 클래스에서 다른 객체의 참조를 가짐
  - User 클래스의 멤버 변수로 Address 객체를 참조함
  - \*: 인스턴스 개수 범위 (0..1)

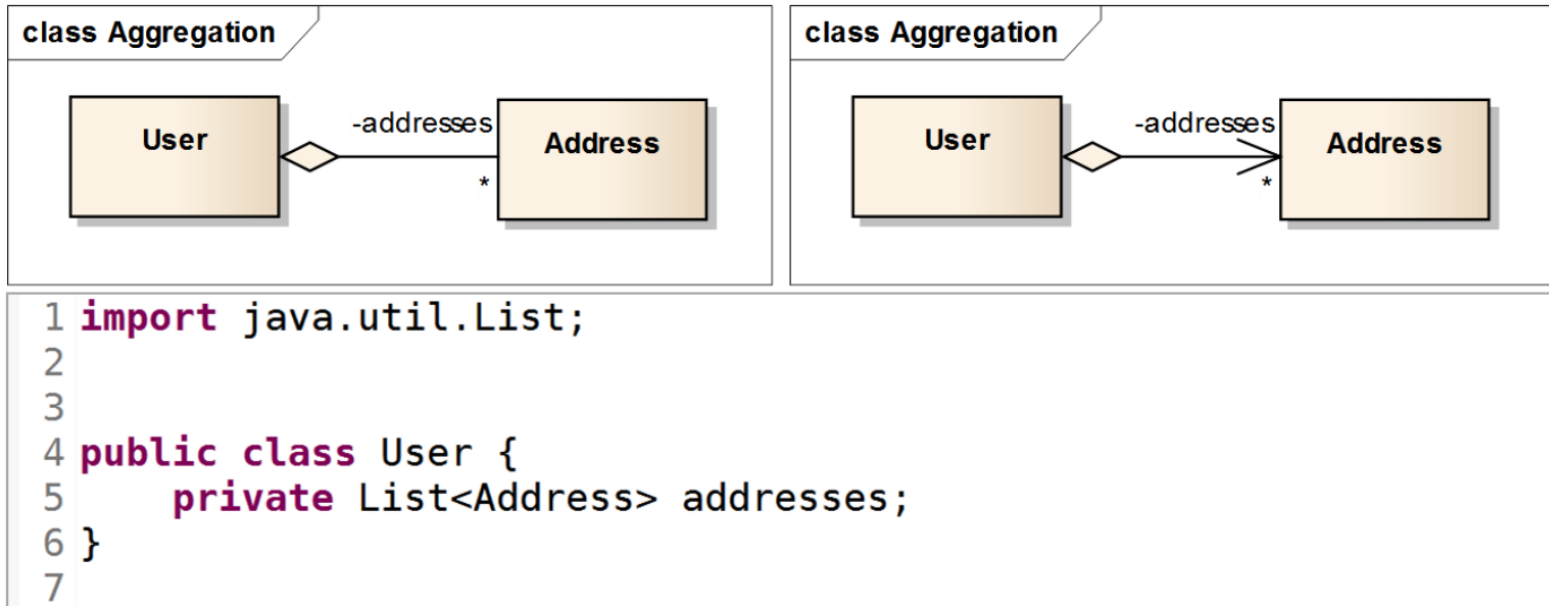


```
1 import java.util.List;
2
3
4 public class User {
5     private List<Address> addresses;
6 }
7
```

# 클래스 다이어그램

## ■ 집합 (Aggregation)

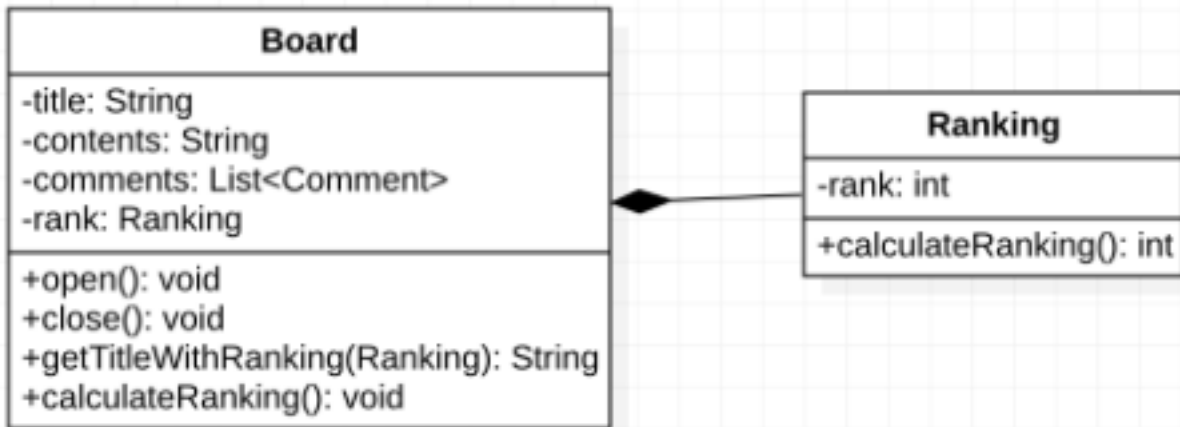
- whole(전체)와 part(부분)의 관계를 나타냄: Association과 차이점이 모호
- **Association의 집합 관계를 나타냄** (Collection, Array를 이용하는 관계)
- part가 whole에 대해 독립적 (whole이 part를 빌려씀)
- 실선으로 연결, whole쪽에 비어 있는 다이아몬드를 배치
  - 화살표 사용은 옵션



# 클래스 다이어그램

## ■ 합성 (Composition)

- Aggregation과 비슷하게 전체와 부분의 집합 관계를 나타냄
- Aggregation 보다 더 강한 집합을 의미
  - part가 whole에 종속적 (whole이 part를 소유)
- 다이어몬드 내부가 채워져 있음



- Ranking 클래스는 Board 클래스 내부에서만 사용
- Board 클래스가 사용되지 않으면, Ranking 클래스도 참조가 불가능한 경우



# Questions?