

# UML과 Class Diagram

이도연(영희)

# 목차

- ▶ 모델링이란?
- ▶ UML
- ▶ 다이어그램의 종류
- ▶ 클래스 다이어그램
- ▶ 클래스 다이어그램의 요소
- ▶ 클래스간의 관계

# 모델링이란?

- **모델링 (Modeling)**

- ✓ 모델을 만드는 모든 작업(추상화)으로써 품질이 좋은 소프트웨어를 개발 및 배치할 수 있게 하는 모든 활동
- ✓ 모델 구축을 통해 개발 대상 시스템에 대한 이해의 증진
- ✓ 모델링 언어 : 모델을 표현할 때 사용되는 언어 (UML)

- **모델 (Model)**

- ✓ 현실의 단순화 및 가시화를 통해 개발 시스템의 계획/구상을 표현
- ✓ 개발 고려 시스템의 총체적인 계획 및 상세 계획 표현
- ✓ 중요 영향 요소의 파악, 불 필요 요소의 생략 및 시스템 구축의 제약 조건 표현

- **모델링 목적**

- ✓ 시스템을 현재 또는 원하는 모습으로 가시화
- ✓ 시스템의 구조와 행동을 명세화
- ✓ 시스템을 구축하는 기본 형태를 제공
- ✓ 시스템 분석/설계의 문서화

# UML이란?

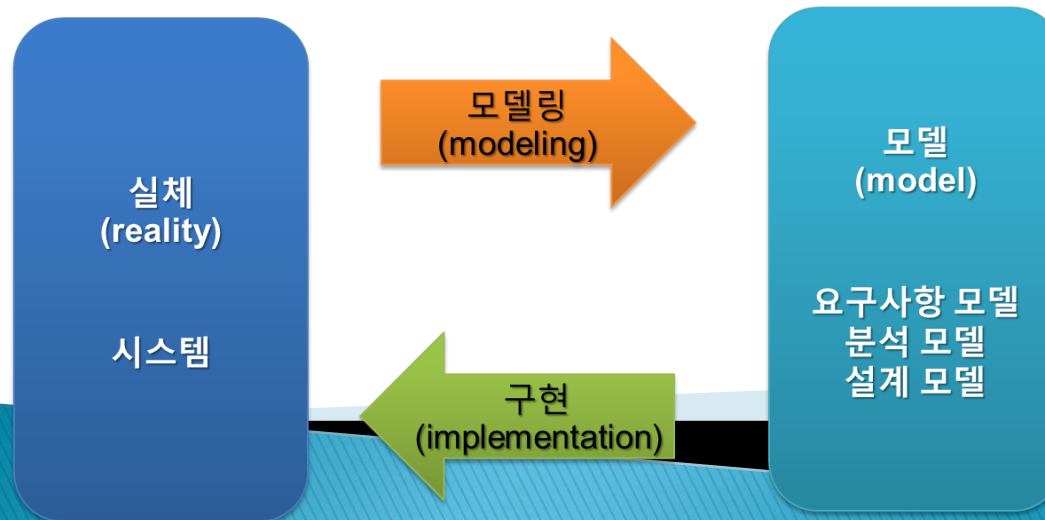
- ▶ UML(Unified Modeling Language) 정의
  - ✓ UML은 소프트웨어 청사진을 작성하는 표준언어로서, 소프트웨어 중심 시스템의 산출물을 **가시화**하고, **명세화**하고, **구축**하고, **문서화**하는데 사용될 수 있다.

# UML(Unified Modeling Language)

- ▶ **통합 모델링 언어**라는 뜻으로 즉, 객체 지향 소프트웨어 엔지니어링 분야의 표준화된 범용 모델링 언어를 말한다.
- 복잡한 소프트웨어 시스템 개발 모델링에 필요한 구성요소를 제시하고 이를 이용한 추상화 방법과 산출물들을 프로젝트 참여자들이 쉽게 이해할 수 있도록 소프트웨어 개발방법론(표현 및 기법)들이 통합된 객체지향개발 표준통합 모델링 언어
- UML은 시스템 개발자가 자신의 비전(vision)을 구축하고 반영하는데 있어서 표준적이고 이해하기 쉬운 방법으로 할 수 있도록 지원
- 자신의 설계 결과물을 다른 사람과 효과적으로 주고받으며 공유할 수 있는 메커니즘 제공

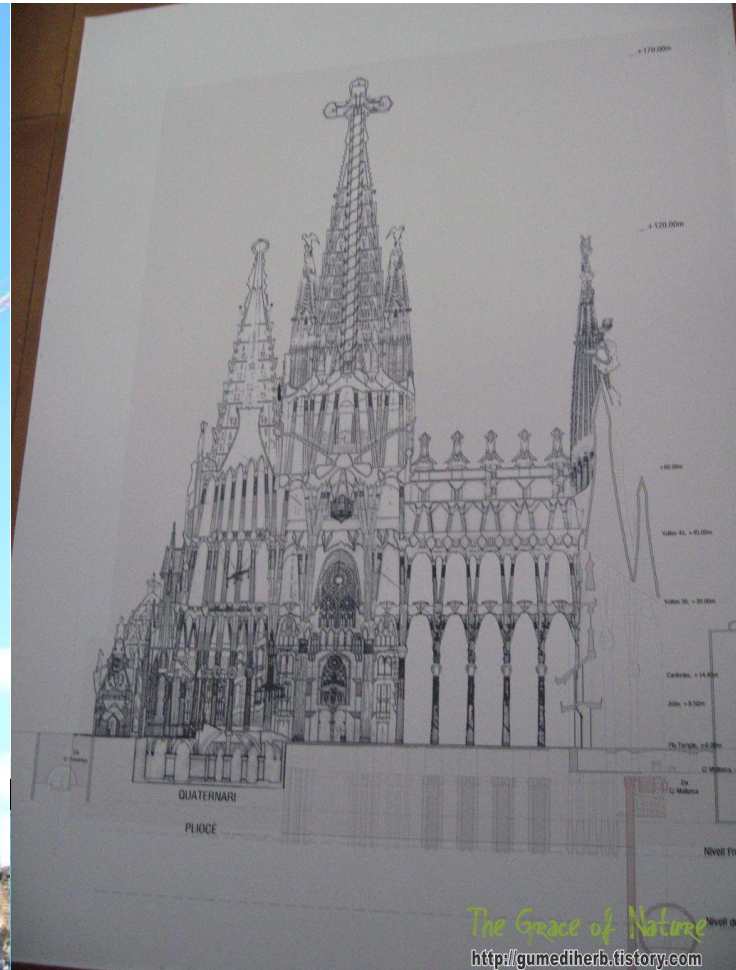
# Unified Modeling Language?

- Model
  - ◆ 현실의 단순화, 가시화를 통해 개발할 시스템에 대한 계획/구상에 대한 내용을 나타낸 것
- Modeling
  - ◆ 양질의 소프트웨어를 개발하기 위해 모델을 만드는 작업
- Modeling의 목적



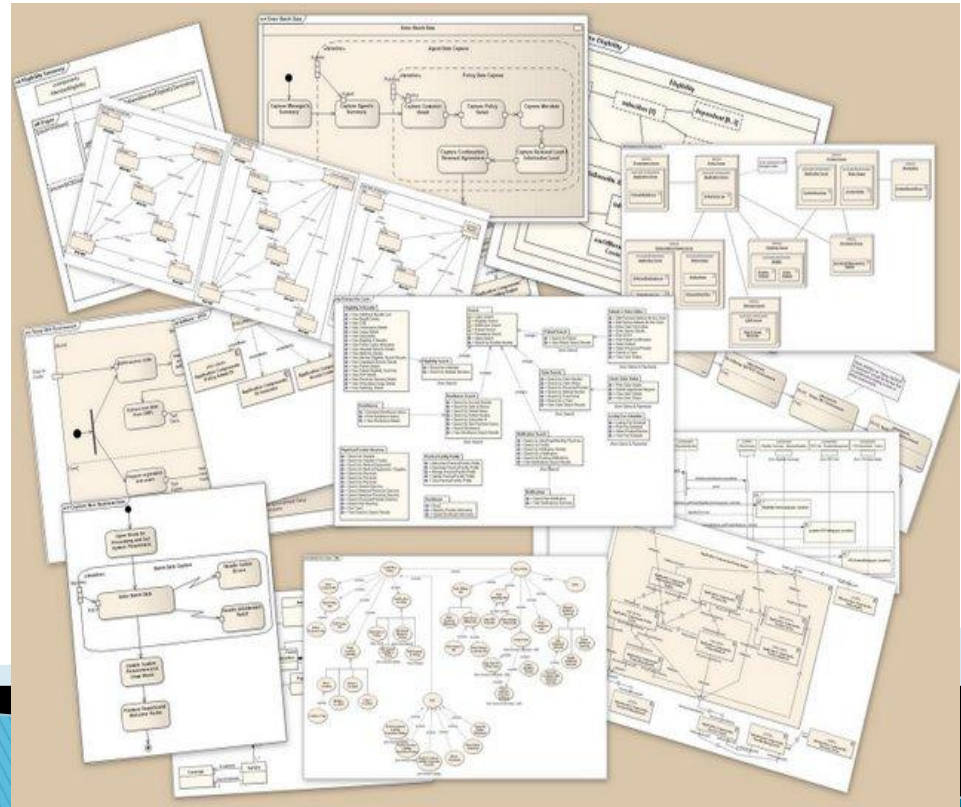
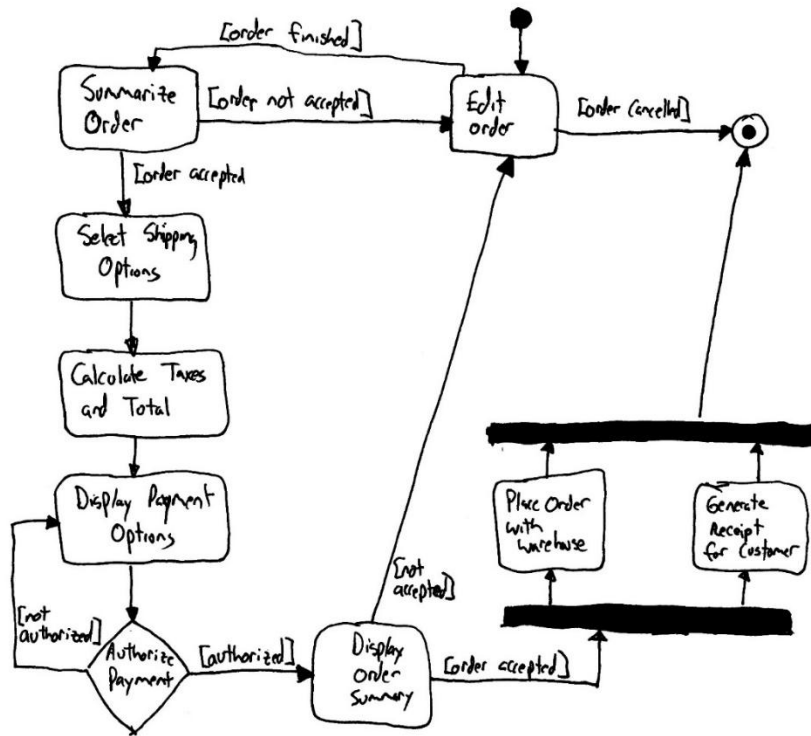


# Unified Modeling Language?





# Unified Modeling Language?





# Unified Modeling Language?

- UML을 왜 쓰는지?



# UML이란?

## ▶ UML을 이용한 모델링의 목적

- ✓ 사용자에게 즉시 사용가능하고 표현력이 강한 시각적 모델링 언어를 제공함으로써 사용자는 의미 있는 모델들을 개발하고 서로 교환할 수 있다.
- ✓ 핵심적이 개념을 확장할 수 있는 확장성과 특수화 방법을 제공한다.
- ✓ 특정 개발 프로세스와 언어에 종속되지 않는다.
- ✓ 모델링 언어를 이해하기 위한 공식적인 기초를 제공한다.
- ✓ 객체 지향 툴 시장의 성장을 장려한다.
- ✓ 콜라버레이션(Collaboration), 프레임워크(Framework), 패턴(Pattern)과 Component와 같은 고수준의 개발 개념을 제공한다.

# UML이란?

## ▶ UML을 이용한 모델링의 궁극적 목적

사용자, 분석가, 설계자, 개발자 등의 개발을 위한 의사소통

# UML이란?

## ▶ UML의 장점

### ▶ 생산성 향상

- ✓ 시스템의 모든 구성요소들이 독립적인 컴포넌트로 제작되므로 컴포넌트 별 제작, 통합, 오류수정, 유지보수 등에서 탁월한 생산성을 보임
- ✓ 프로젝트 신규 참여자들이 쉽게 프레임워크와 라이브러리 등을 이해하고 활용

### ▶ 재사용

- ✓ Application Framework의 재사용
- ✓ 사용자관리 컨트롤 등의 재사용 : 우편 수신처 지정, 게시와 문서관리의 사용 권한 지정, 결재선 지정, 회의 참석자 지정 등

### ▶ 개발 도구 사용

- ✓ 모델링 도구, 컴파일러, 형상관리 등의 도구를 함께 사용

### ▶ 팀 작업

- ✓ 30명이 넘는 개발팀원들이 하나의 파일로 이루어진 모델을 잘게 나누어 작업 함으로써 다른 팀원의 변경사항이 서로 즉각적으로 반영됨

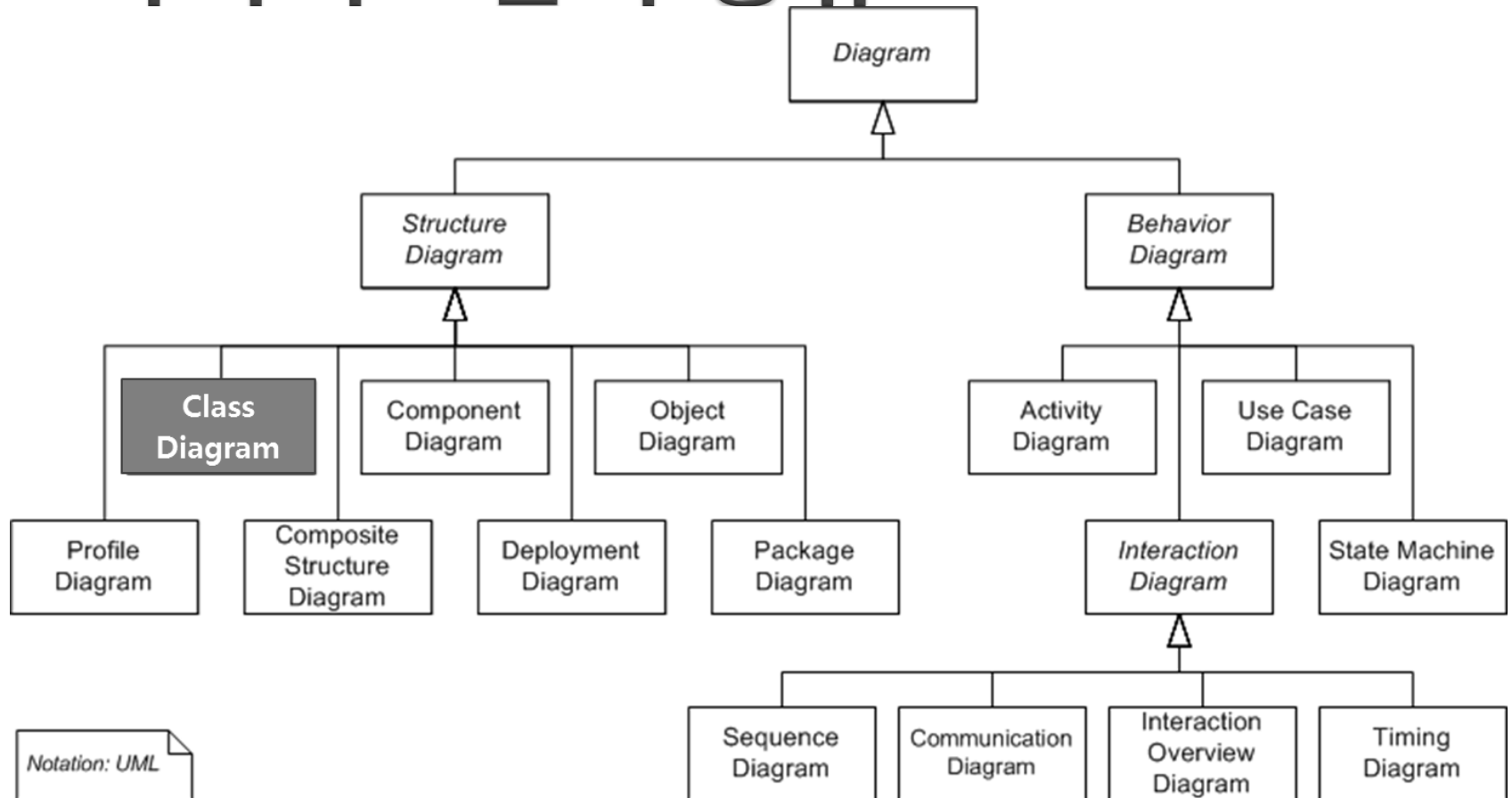


# UML이란?

## ▶ UML 필요 배경

- 시스템의 복잡성 → 표준적인 표기(notation) 법 즉, 모델링 언어 필요
  - 객체지향 분석/설계 개발 방법론의 표준 부재 (객체지향 개발방법론의 전쟁)
  - 객체지향 개발방법론의 거장의 협력 및 OMG의 표준화 활동

# 다이어그램의 종류



UML은 구조 다이어그램 7개, 행위 다이어그램 7개로 총 14종류

# 클래스 다이어그램

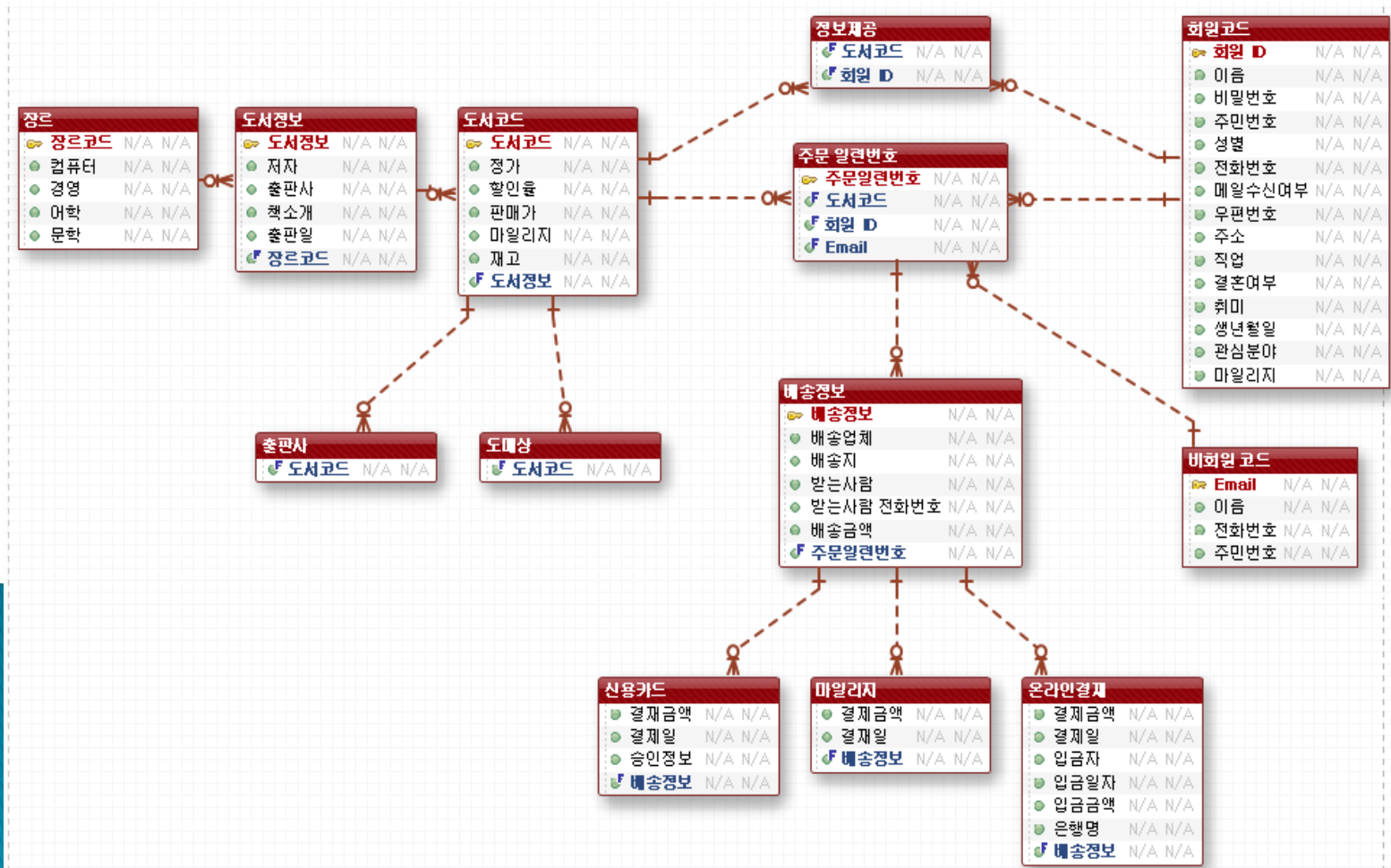
## ▶ 클래스 다이어그램

- ✓ 비슷한 속성과 공통적인 행동수단을 지닌 것들의 범주 / 그룹
- ✓ 속성(attribute)과 행동(behavior)으로 정의
- ✓ 주변 사물을 속성과 행동으로 생각하는 버릇
- ✓ 예) 세탁기 클래스

<b>Washing Machine</b>
<b>brand name</b> <b>model name</b> <b>serial name</b> <b>capacity</b>
<b>add clothes()</b> <b>add detergent()</b> <b>remove clothes()</b>

# Class Diagram

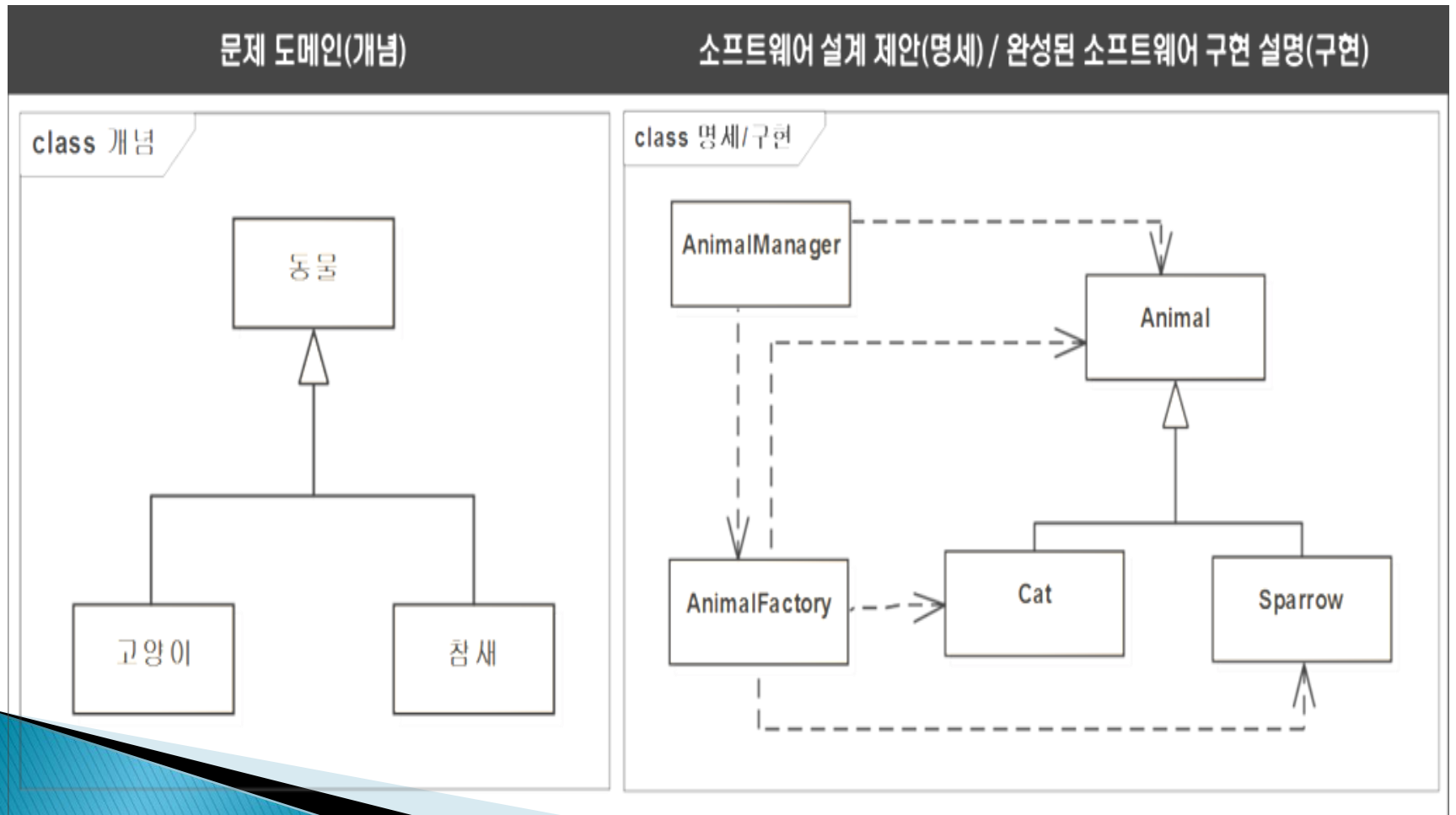
- ER Diagram





# 목적 별 클래스 다이어그램

## ▶ 목적 별 클래스 다이어그램

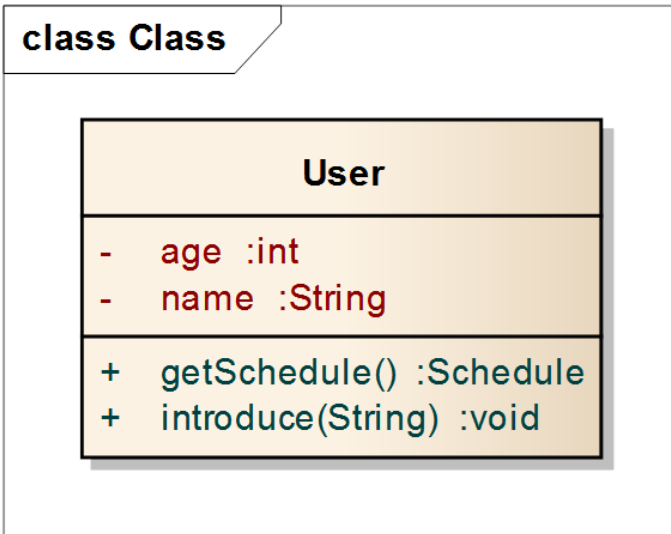


# 클래스 다이어그램의 요소(Element)

- ▶ **Class**
- ▶ **Stereo Type**
- ▶ **Abstract Class / Method**

# 클래스 다이어그램의 요소(Element)

## ▶ Class

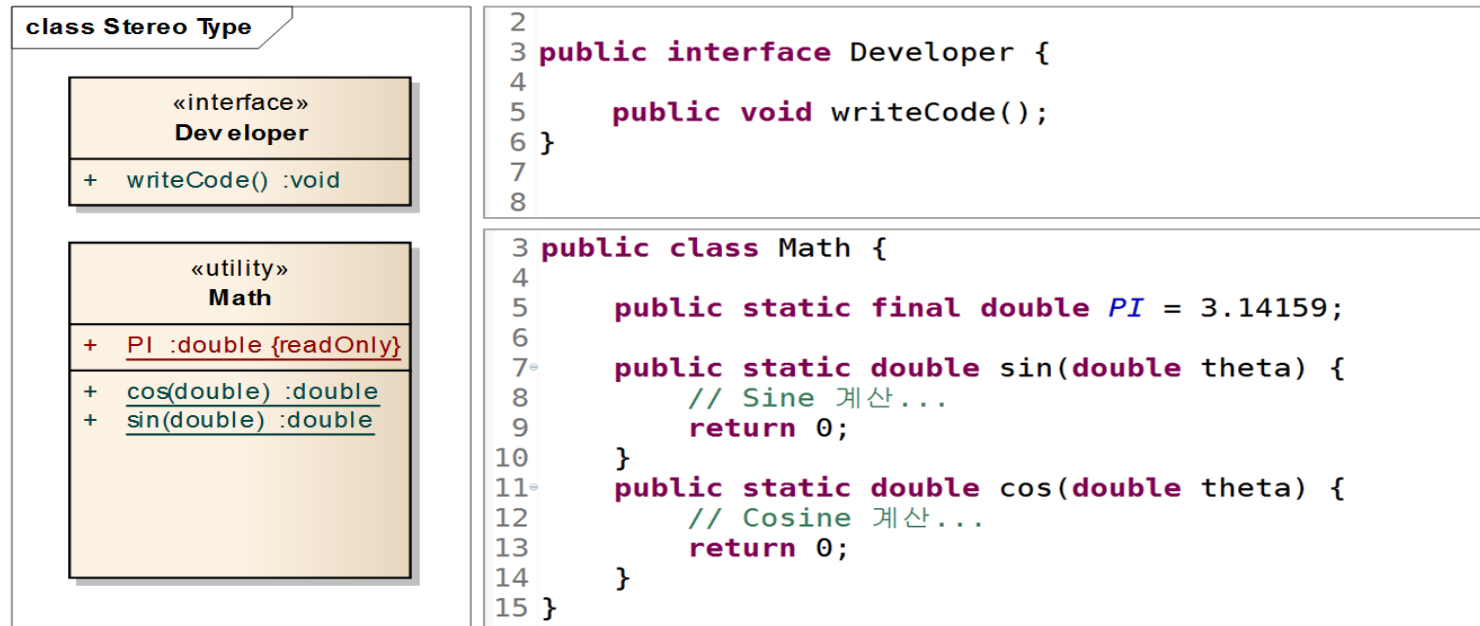


```
4
5 public class User {
6     private int age;
7     private String name;
8
9     public Schedule getSchedule() {
10         // 스케줄을 본다.
11         return null;
12     }
13     public void introduce(String introduce) {
14         // 자기소개를 한다.
15     }
16 }
17
```

- ▶ 클래스는 보통 3개의 compartment(구획)으로 나누어 클래스의 이름, 속성, 기능을 표기합니다. 속성과 기능은 옵션으로 생략이 가능하지만 이름은 필수로 명시해야 합니다.

# 클래스 다이어그램의 요소(Element)

## ▶ Stereo Type (스테레오 타입)

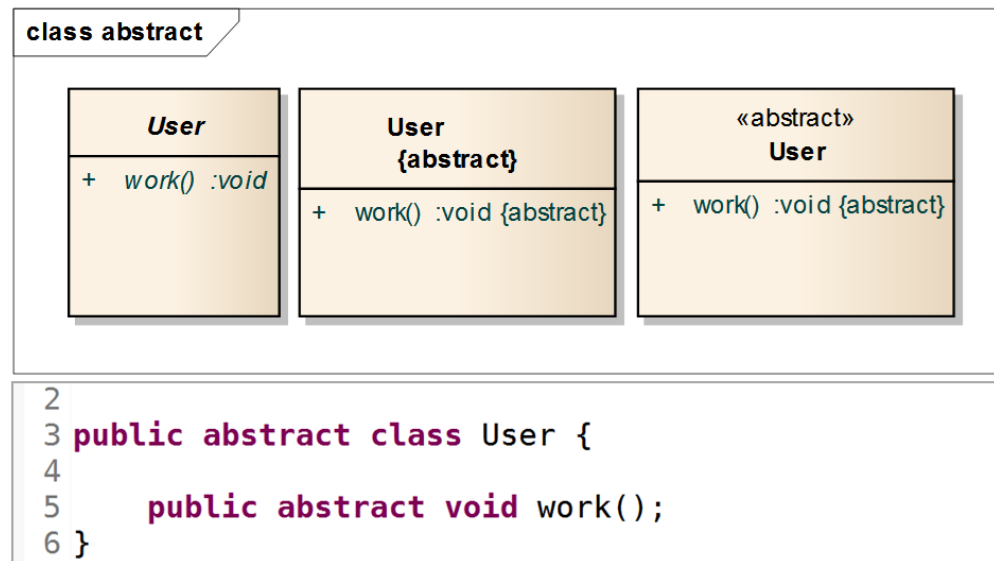


- ▶ 스테레오 타입이란 UML에서 제공하는 기본 요소 외에 추가적인 확장요소를 나타내는 것으로 쌍 꺾쇠와 비슷하게 생긴 길러멧(guillemet, « ») 사이에 적습니다.



# 클래스 다이어그램의 요소(Element)


## ▶ Abstract Class/Method



- ▶ 추상클래스란 1개 이상의 메서드가 구현체가 없고 명세만 존재하는 클래스를 말합니다.

# 클래스간의 관계

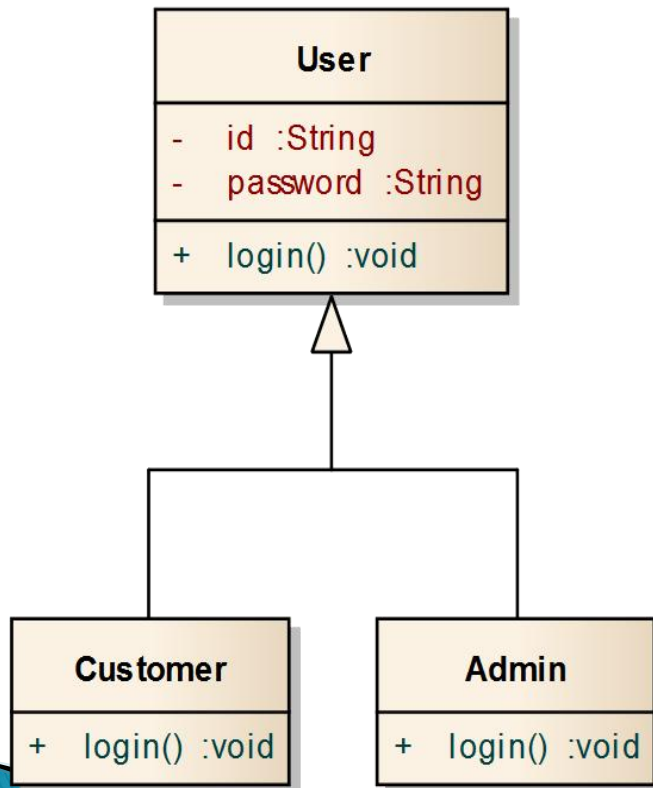
- ▶ 클래스 다이어그램의 주 목적은 클래스간의 관계를 한눈에 쉽게 보고 의존 관계를 파악하는 것에 있습니다. 그렇기 때문에 클래스 다이어그램에서 가장 중요한 것이 클래스간의 관계입니다.

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

# 클래스간의 관계

## ▶ Generalization (일반화)

class Generalization



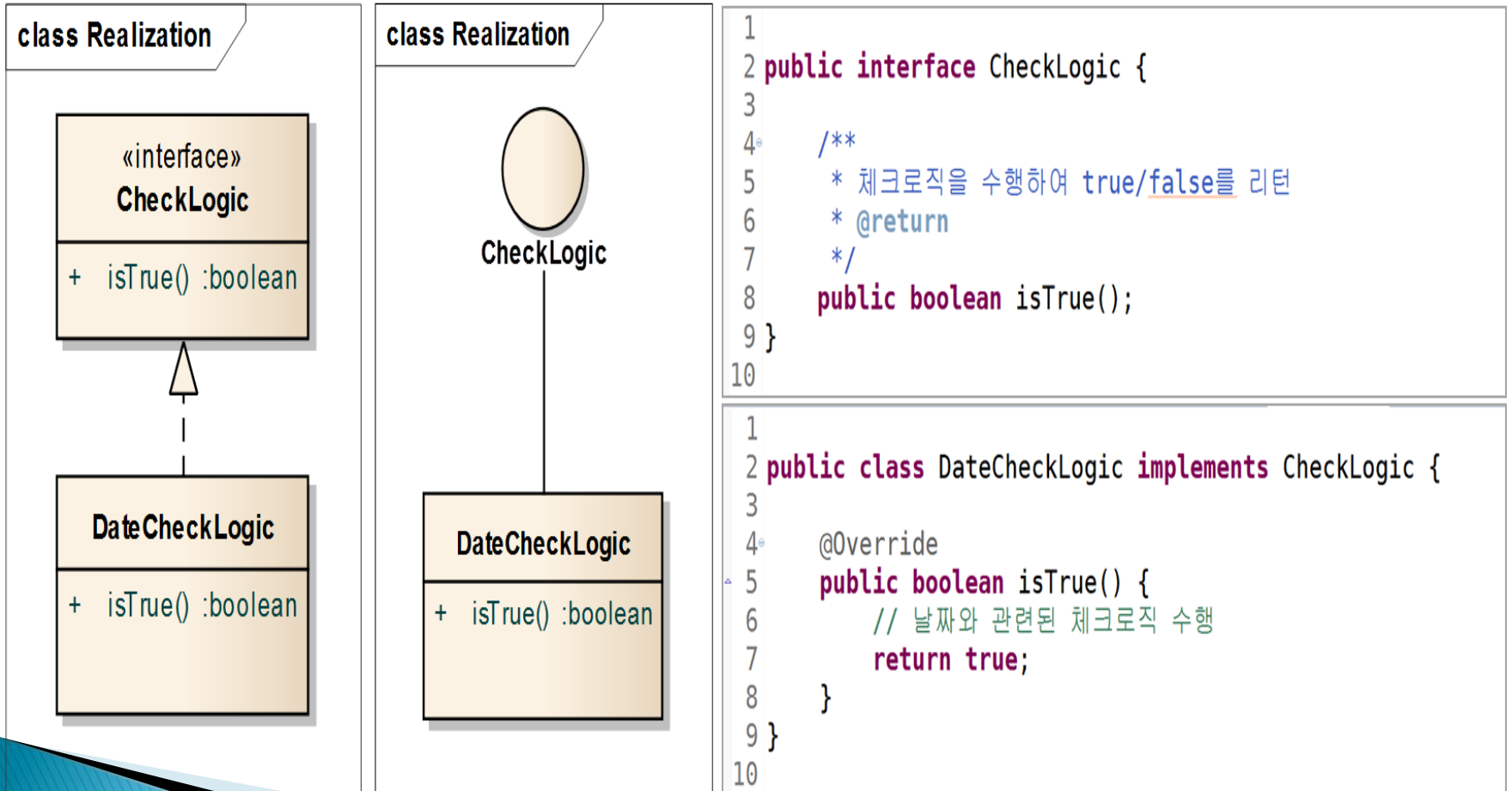
```
3 public class User {
4
5     private String id;
6
7     private String password;
8
9     /**
10      * 로그인
11      */
12     public void login() {
13         // 일반 사용자의 로그인
14     }
15 }
```

```
5 public class Customer extends User {
6
7     @Override
8     public void login() {
9         // Customer의 로그인..
10     }
11 }
```

```
5 public class Admin extends User {
6
7     @Override
8     public void login() {
9         // 관리자의 로그인...
10     }
11 }
```

# 클래스간의 관계

## ▶ Realization (실체화)

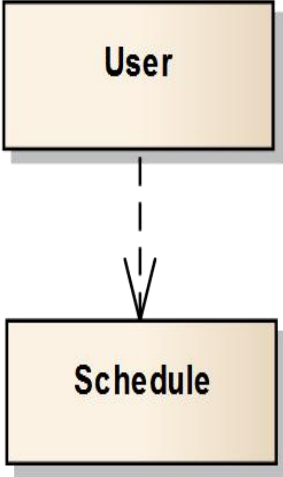




# 클래스간의 관계

## ▶ Dependency (의존)

class Dependency

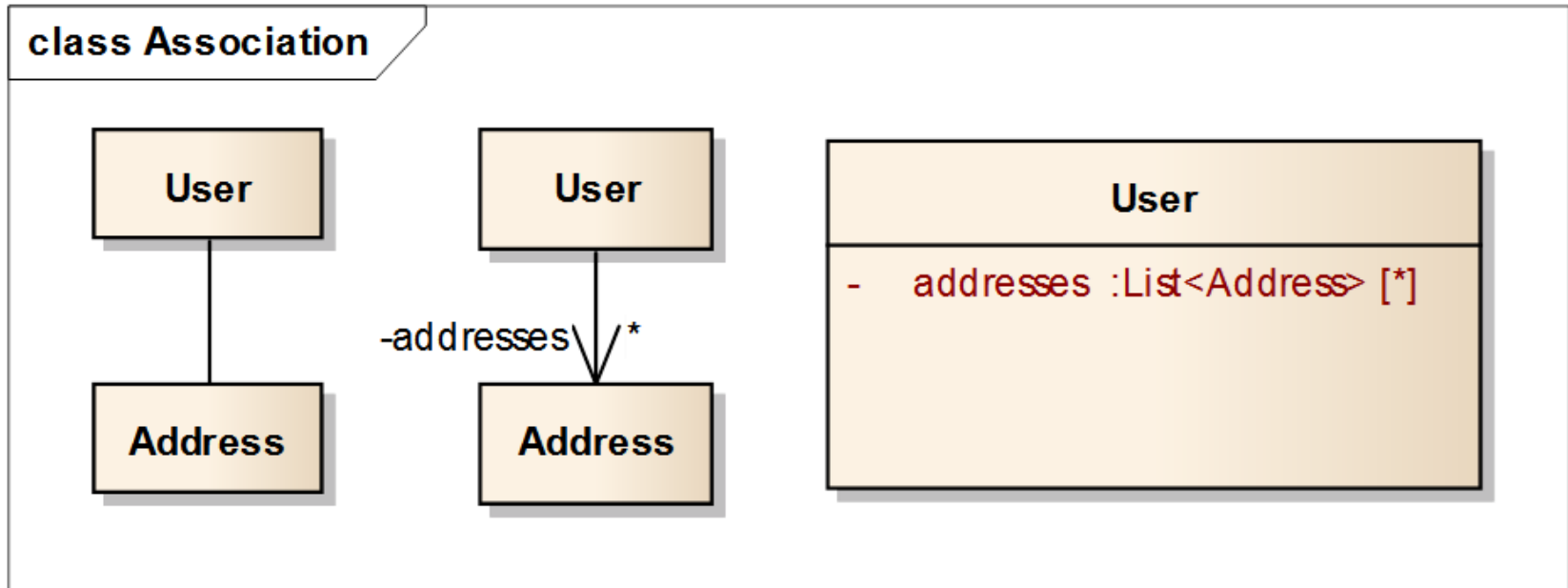


```
graph TD; User --> Schedule;
```

```
2
3 public class User {
4
5     public Schedule crateSchedule() {
6         // 객체 생성 및 리턴
7         return new Schedule();
8     }
9
10    public void useSchedule(Schedule schedule) {
11        // 객체를 매개변수로 받아 사용
12        // use schedule...
13        Schedule schedule2014 = schedule.getScheduleByYear(2014);
14    }
15 }
16
```

# 클래스간의 관계

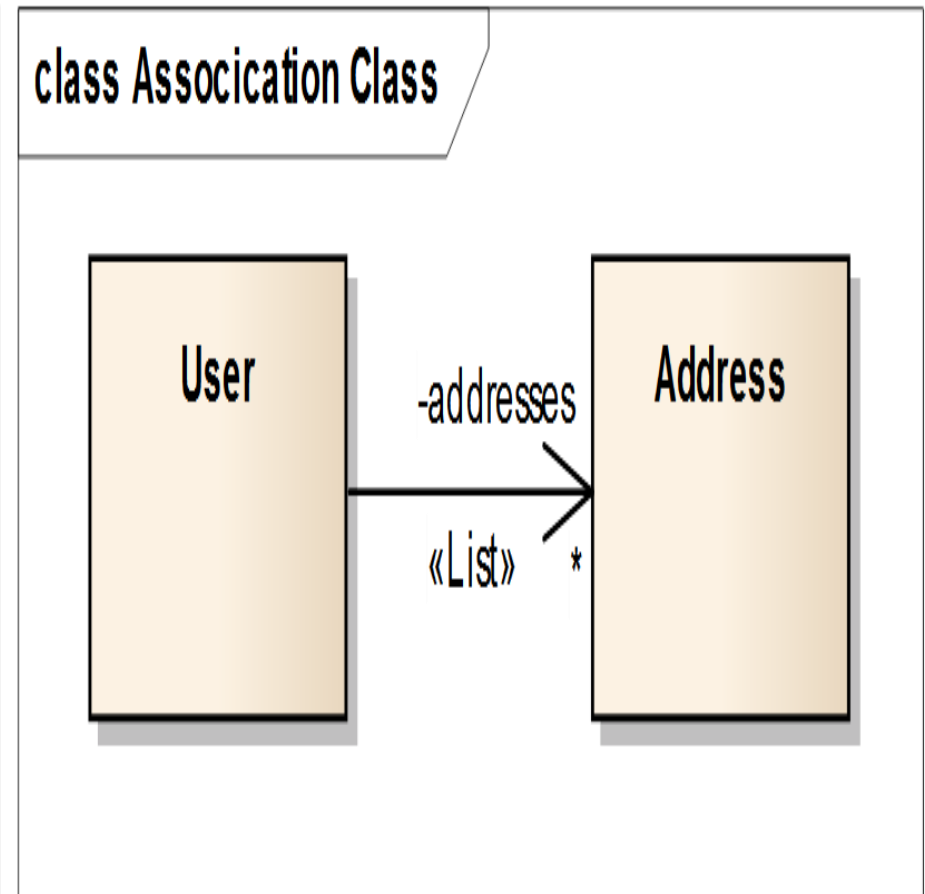
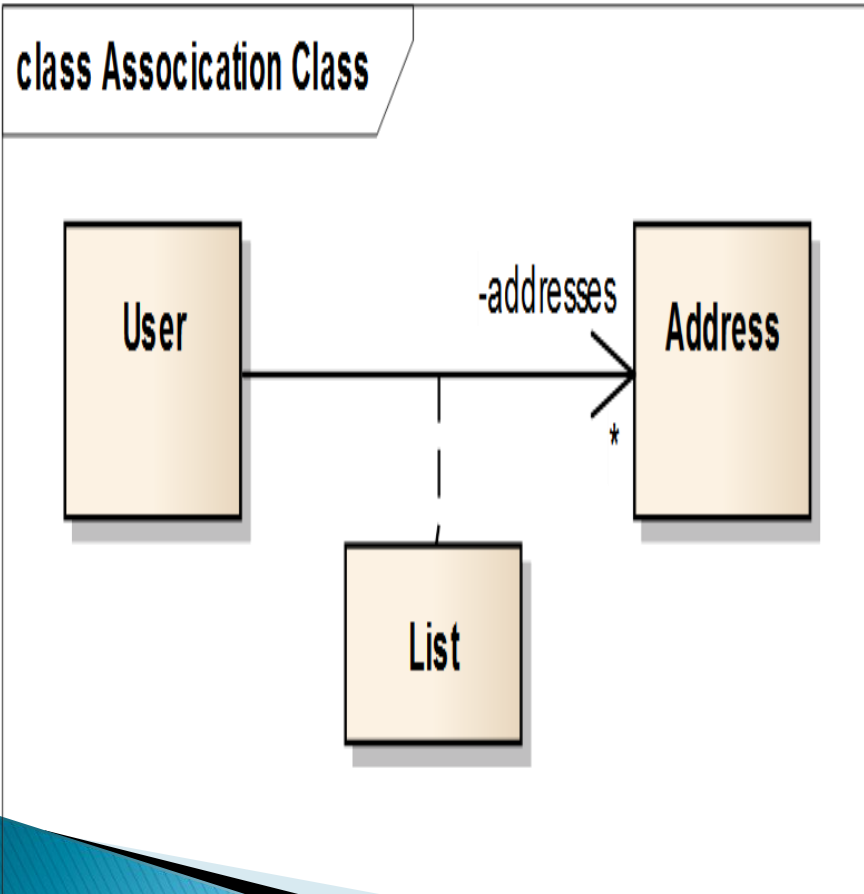
- ▶ Association (연관), Directed Association(방향성 있는 연관)



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

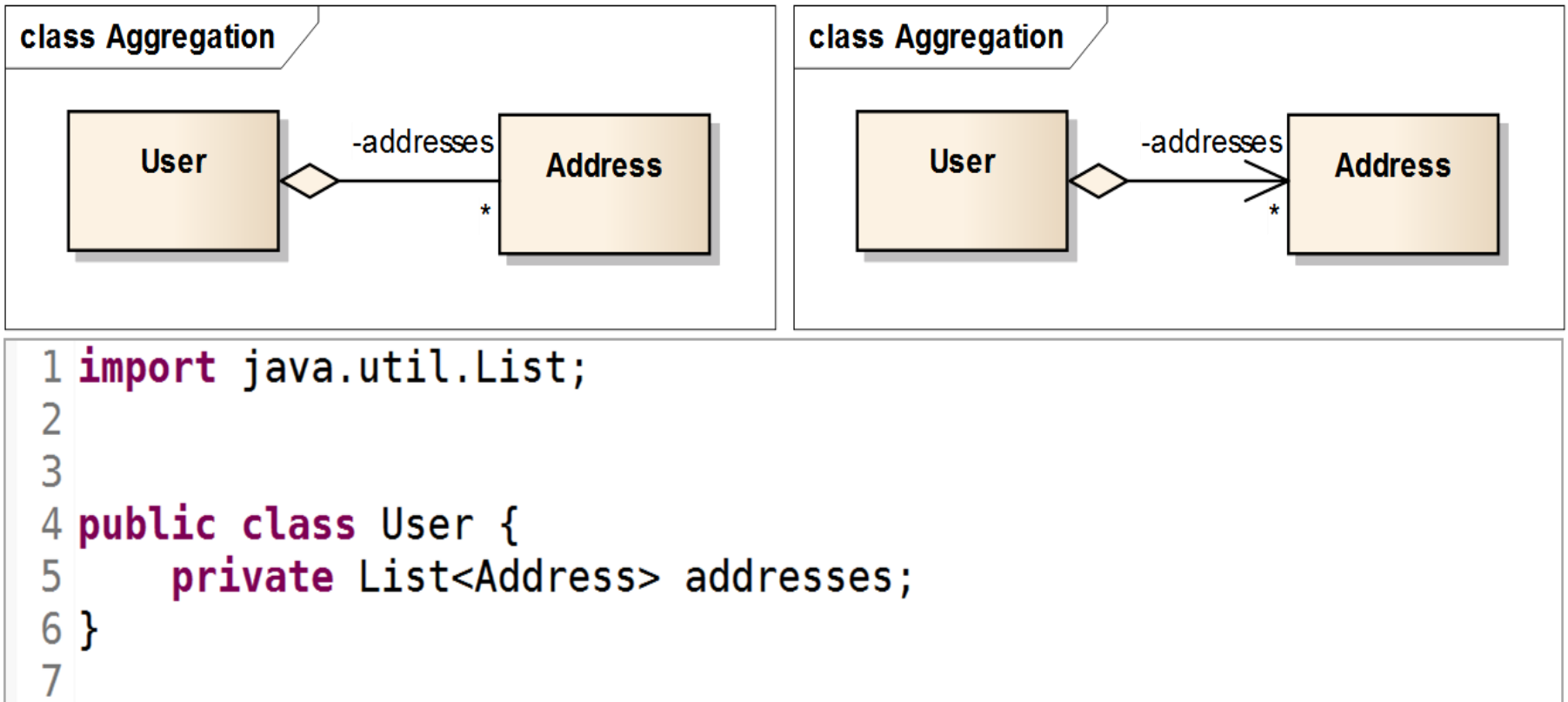
# 클래스간의 관계

- ▶ Association (연관), Directed Association(방향성 있는 연관)



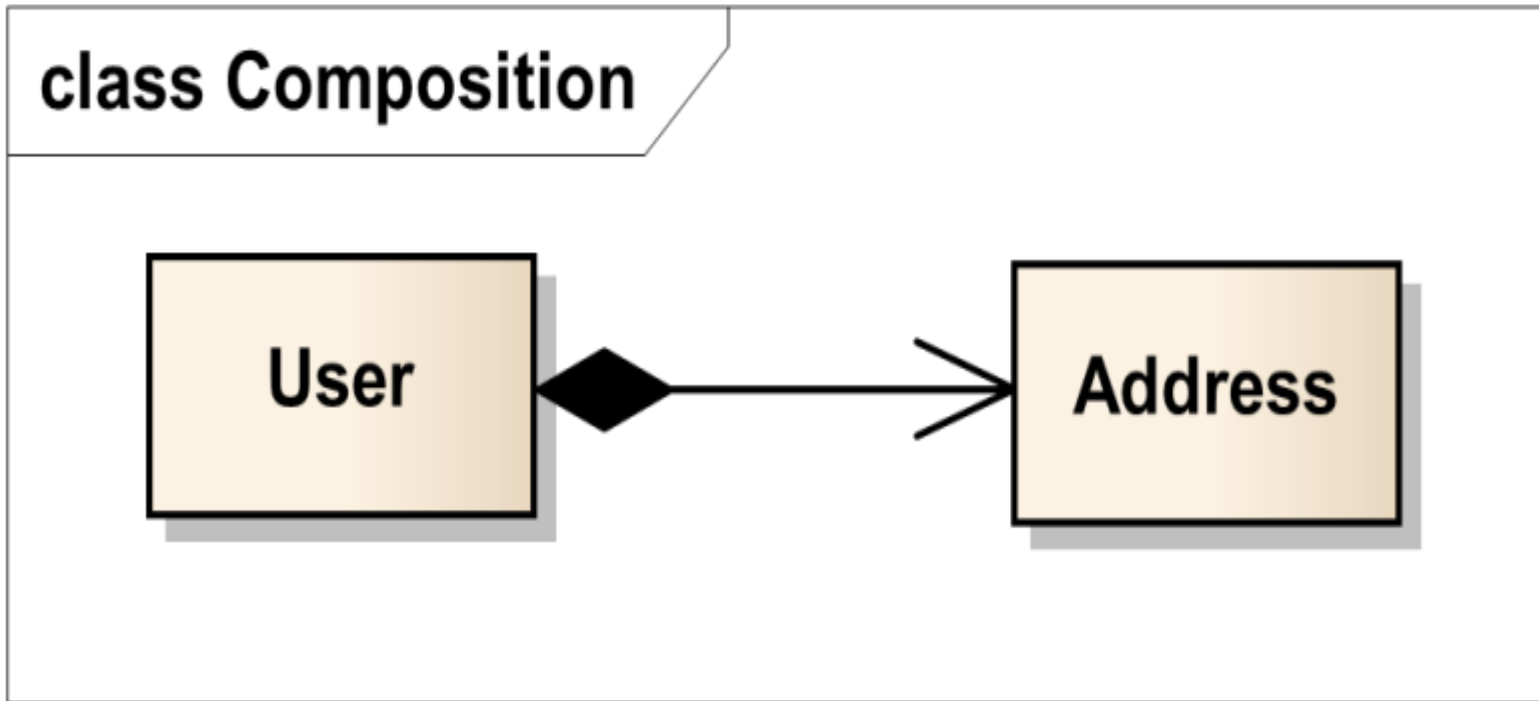
# 클래스간의 관계

## ▶ Aggregation (Shared Aggregation, 집합)



# 클래스간의 관계

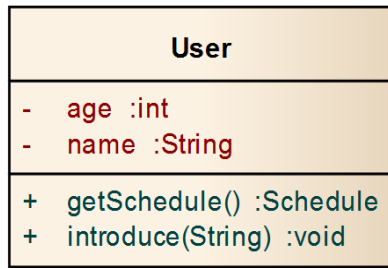
- ▶ Composition (Composite Aggregation, 합성)



# Class Diagram

## 요소

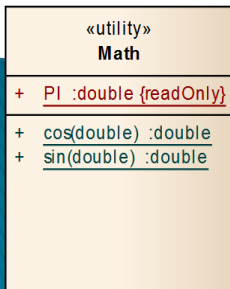
### class Class



```
4
5 public class User {
6     private int age;
7     private String name;
8
9     public Schedule getSchedule() {
10         // 스케줄을 본다.
11         return null;
12     }
13     public void introduce(String introduce) {
14         // 자기소개를 한다.
15     }
16 }
17
```

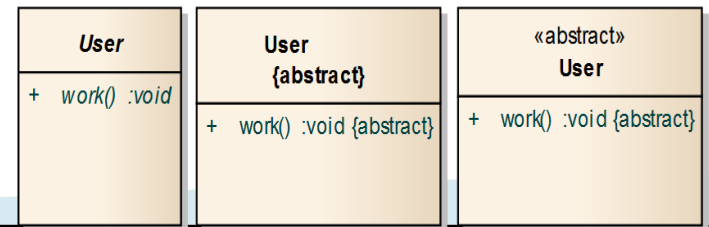


### class Stereo Type



```
2
3 public interface Developer {
4
5     public void writeCode();
6 }
7
8
9
10
11 public class Math {
12
13     public static final double PI = 3.14159;
14
15     public static double sin(double theta) {
16         // Sine 계산...
17         return 0;
18     }
19
20     public static double cos(double theta) {
21         // Cosine 계산...
22         return 0;
23     }
24 }
25
```

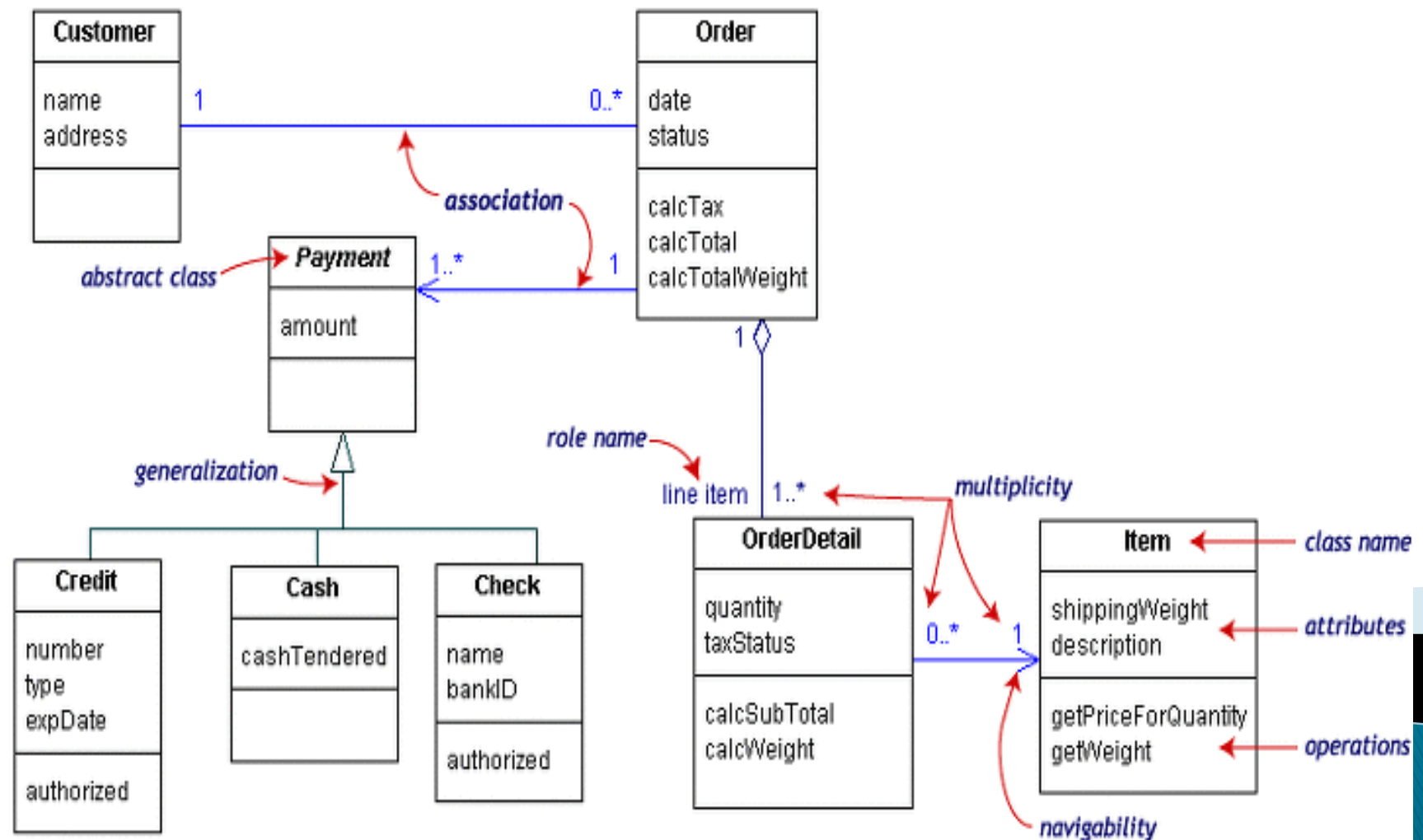
### class abstract



```
2
3 public abstract class User {
4
5     public abstract void work();
6 }
```

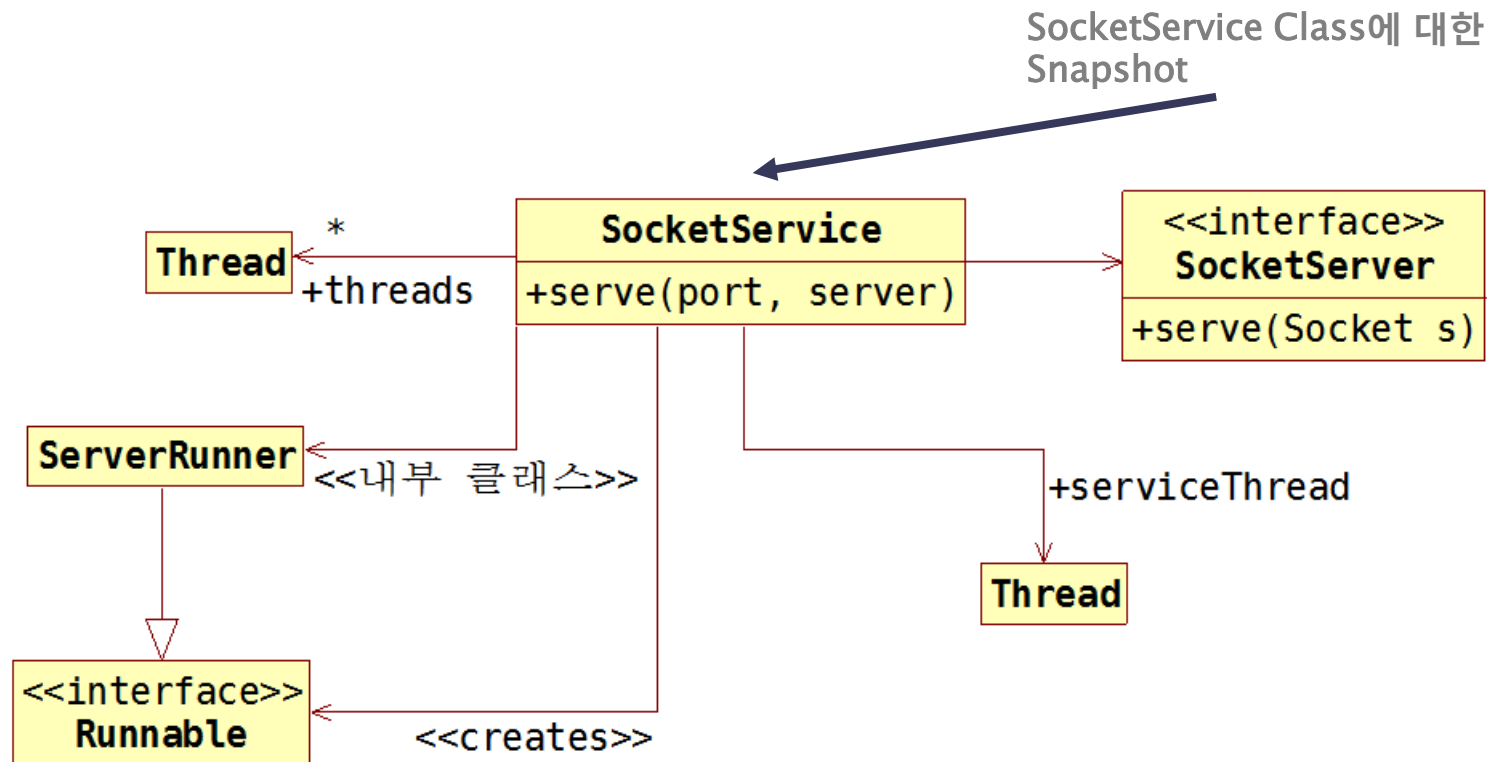


# Class Diagram



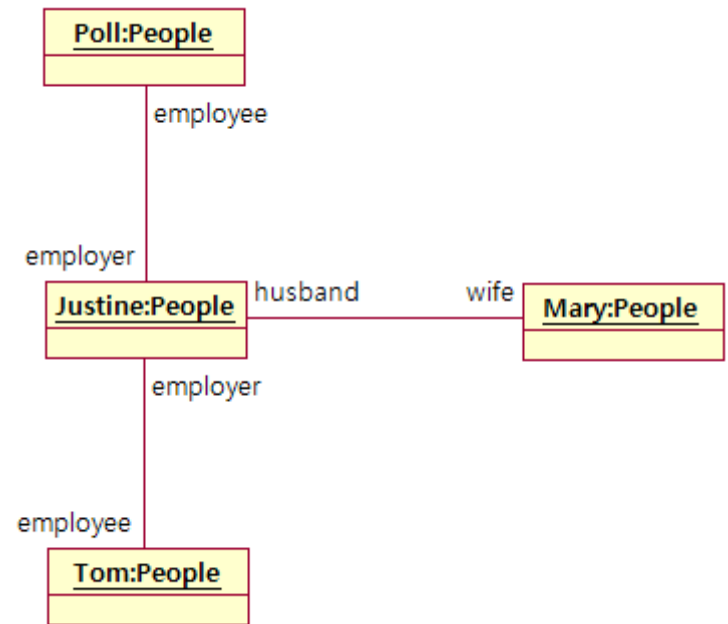
# Object Diagram

- 정의
  - 시스템의 실행 중 어느 시간의 객체와 관계를 표시
  - 특정 시점의 시스템 상태를 Snapshot 으로 찍어두는 개념



# Object Diagram

- 관계
  - ◆ Object와 Object의 관계를 표시
- 요소
  - ◆ Object
    - Object에는 밑줄을 표시
    - Class 표기와 비슷하지만 속성만 나타낼 수 있고, Method 는 나타낼 수 없음
  - ◆ Link
    - Object 간 접속을 의미



# Wrap up

## UML

UML이란 모델을 다이어그램으로 그리기 위해 사용하는 시각적인 표기법

커뮤니케이션과 유지보수에 용이하기 때문에 사용

다이어그램을 그려야 할 때와 그리지 말아야 할 때를 잘 판단해야 한다

### Class Diagram

Class와 Class 사이의 관계를 표현한 Diagram

모든 변수와 Method를 기록할 필요는 없고  
중요한 Method를 기록

시스템 설계 시 주로 사용

연관 관계를 세부적으로 표현할 필요는 없다.  
(association 만으로 충분)

### Object Diagram

Object와 Object 사이의 관계를 표현한 Diagram

정적인 구조로부터 동적으로 만드는 경우 유용

시스템 분석 용도로 사용

Class Diagram에서 유추할 수 없을 때 사용

# Reference

- 참고

- ◆ <http://www.nexttree.co.kr/p6753/>
- ◆ Java 프로그래머를 위한 UML 실전에서는 이것만 쓴다! (R.C.Martin 저)
- ◆ 아무도 가르쳐 주지 않았던 소프트웨어 설계 테크닉 (Watanabe kouze 저)
- ◆ 그 외 많은 Google 검색 자료.