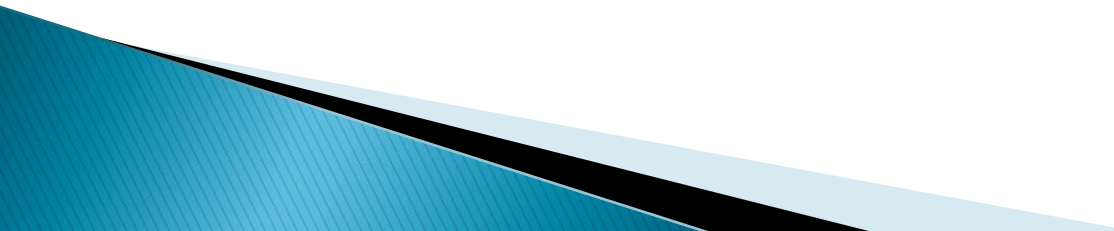


UML Class Diagram

목차

1. UML이란?, Class Diagram . . . 3p
 2. Class Diagram 작성법 . . . 3p
 3. 일반화(Generalization) . . . 6p
 4. 실체화(Realization), 의존(Dependency) . . . 7p
 5. 연관(Association, Directed Association) . . . 8p
 6. 집합(Aggregation), 합성(Composition) . . . 9p
- 

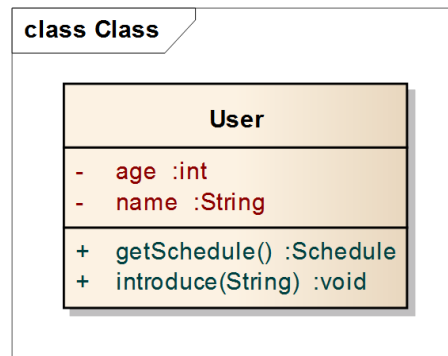
Class Diagram

▶ UML이란?

- Unified Modeling Language의 약자로 1997년 OMG(Object Management Group)에서 표준으로 채택한 **통합 모델링언어**
- 모델을 만드는 표준언어
- 전체 시스템의 구조 및 클래스의 의존성 파악을 위해 사용

▶ Class Diagram

- UML을 나타내는 구조 다이어그램 중 하나
- 3개의 구획(compartment)으로 나누어 **클래스 이름**, **속성**, **기능**을 표기
- 속성과 기능은 생략 가능
- ❖ **작성 툴 웹 사이트** : <https://app.diagrams.net/>



```
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 Diagram

▶ 세부사항 작성 순서

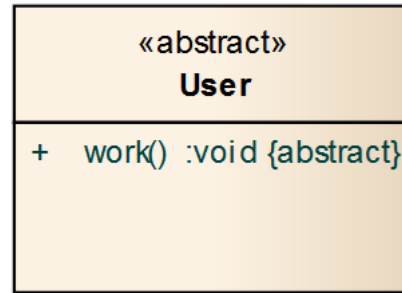
- 속성 : 접근제한자 필드명 : 데이터 타입
- 기능 : 접근제한자 함수명(매개변수 타입) : 리턴 타입

▶ 접근제한자(access modifier) 기호

- `-` : *private*
- `#` : *protected*
- `+` : *public*

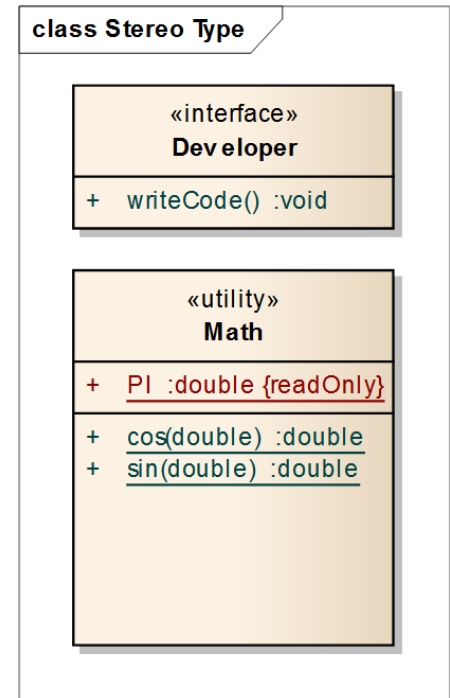
▶ 스테레오 타입(Stereo Type)

- 추가적인 확장요소를 나타낸다
- 길러멧(guillemet, « ») 사이에 작성
- «interface», «utility», «abstract», «enumeration»이 많이 사용된다
- ❖ {readOnly} : final, sealed 키워드(상속을 허용하지 않는다)
- ❖ 밑줄 : static(정적) 속성 및 기능



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

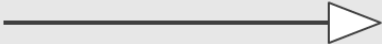






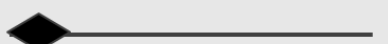

```
2
3 public interface Developer {
4
5     public void writeCode();
6 }
7
8
```



```
3 public class Math {
4
5     public static final double PI = 3.14159;
6
7     public static double sin(double theta) {
8         // Sine 계산...
9         return 0;
10    }
11    public static double cos(double theta) {
12        // Cosine 계산...
13        return 0;
14    }
15 }
```

Class Diagram

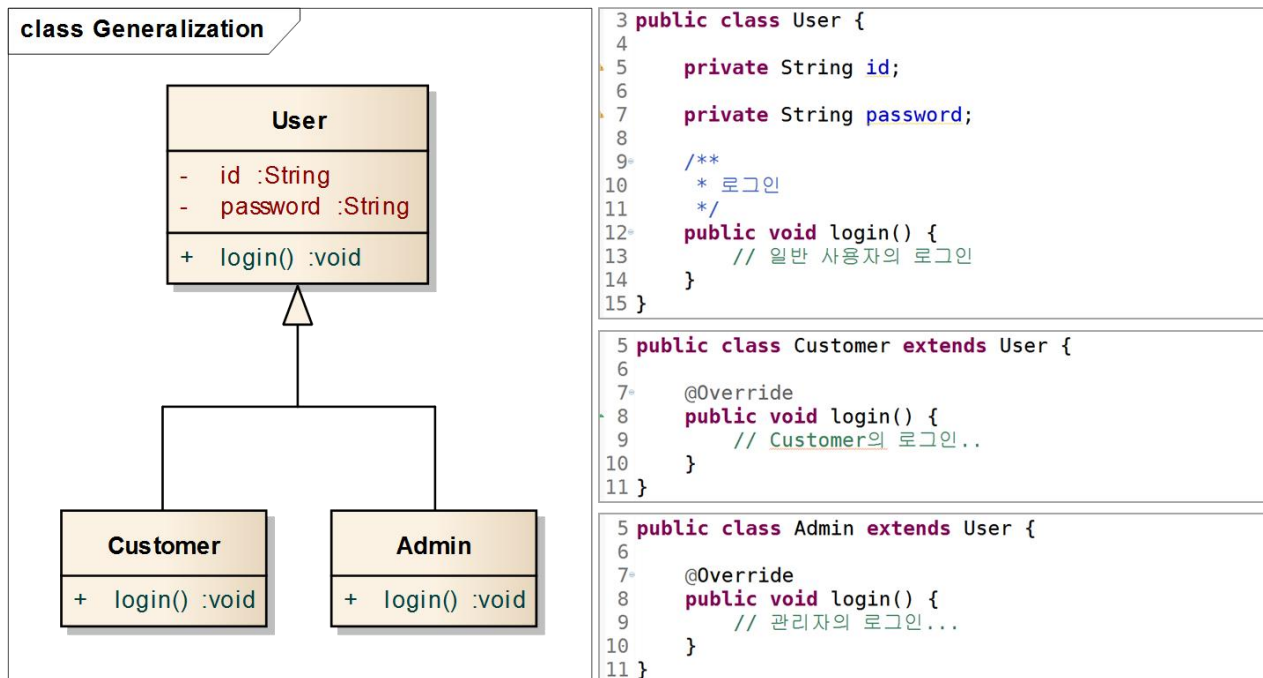
❖ 클래스간의 관계

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

Class Diagram

▶ 일반화(Generalization)

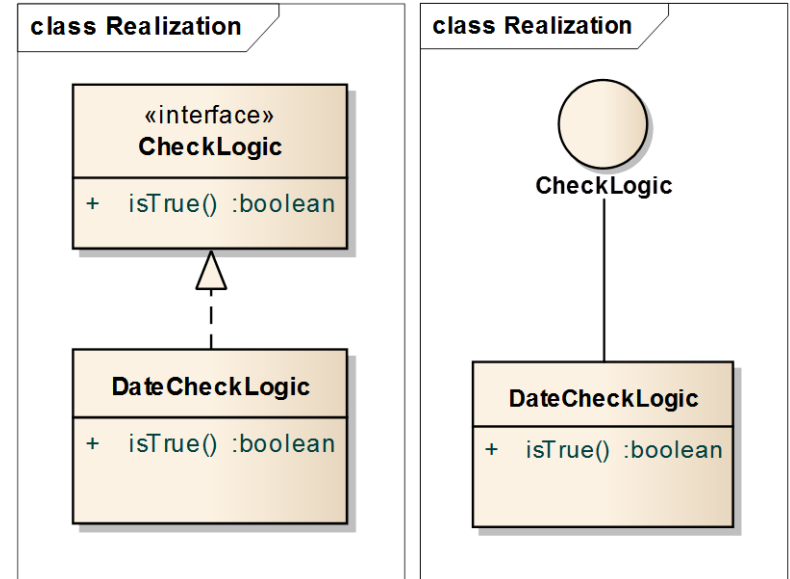
- 부모 클래스와 자식 클래스간의 상속 관계
- 자식 클래스에서 부모 클래스로 향한다



Class Diagram

▶ 실체화(Realization)

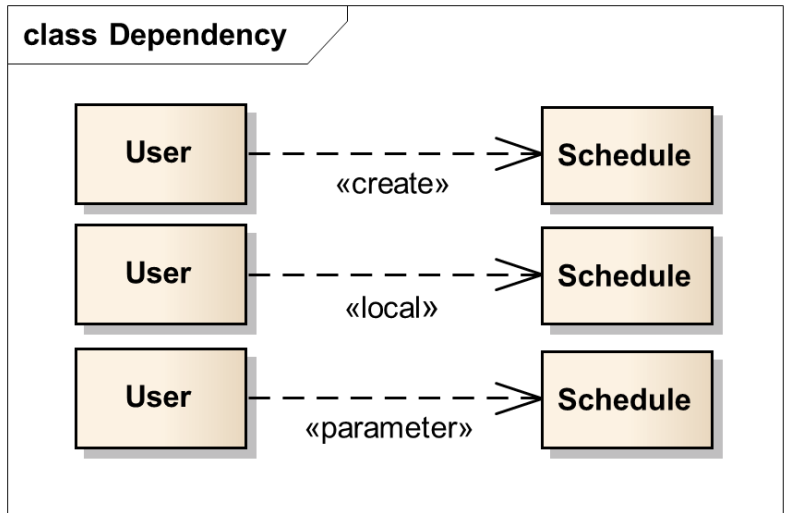
- Interface를 override하여 실제 기능을 구현
- 구현 클래스에서 인터페이스로 향한다



▶ 의존(Dependency)

- 클래스 다이어그램에서 가장 많이 사용하는 관계 표현
- A 클래스가 B 클래스를 참조, 일시적(Local) 사용
- 참조하는 클래스에서 참조되는 클래스로 향한다

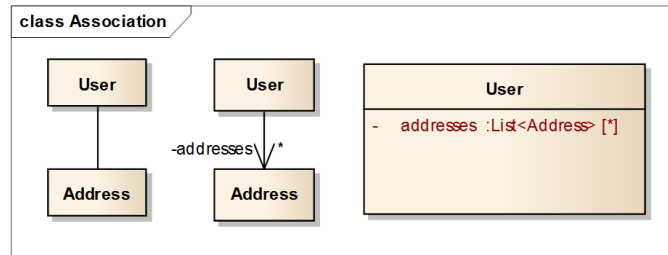
```
2
3 public class User {
4
5     public Schedule crateSchedule() {
6         // 객체 생성 및 리턴
7         return new Schedule();
8     }
9
10    public void useSchedule(Schedule schedule) {
11        // 객체를 매개변수로 받아 사용
12        // use schedule...
13        Schedule schedule2014 = schedule.getScheduleByYear(2014);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



Class Diagram

▶ 연관(Association)

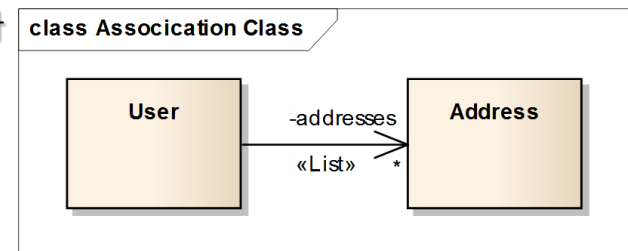
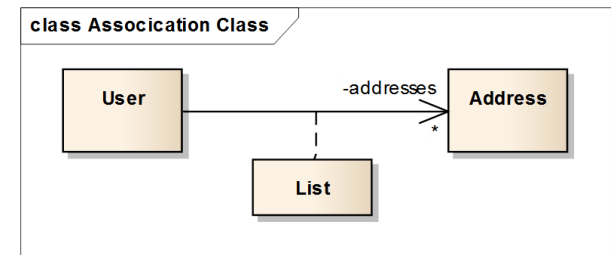
- A 클래스와 B 클래스가 참조, **지속적(Member) 사용**
- A가 B를 참조 또는 B가 A를 참조 또는 A와 B가 서로를 참조
- 방향성이 없다



▶ 방향성이 있는 연관(Directed Association)

- A 클래스가 B 클래스를 참조, **지속적(Member) 사용**
- 참조하는 클래스에서 참조되는 클래스로 향한다
- ❖ * (Multiplicity, 개수) : 대상 클래스가 지닐 수 있는 인스턴스의 수
- ❖ 정수(min...max) 표기 : 객체를 최소 min 개에서 최대 max 개를 가진다
- ❖ * 표기 : 0...*와 같은 의미로 객체 수에 제한이 없으며 객체가 없을 수 있다

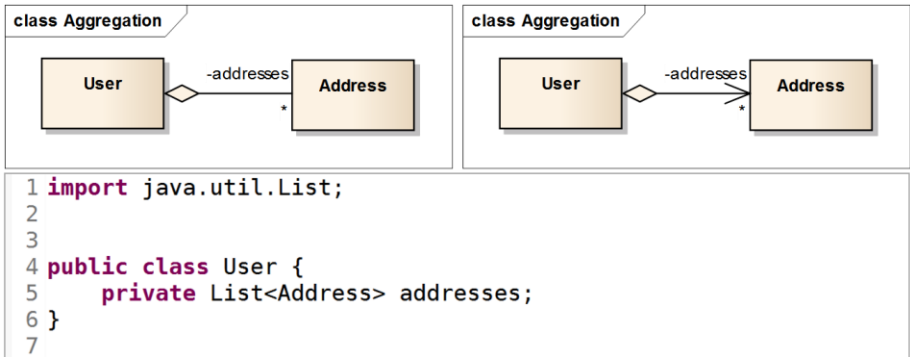
```
1 import java.util.List;
2
3
4 public class User {
5     private List<Address> addresses;
6 }
7
```



Class Diagram

▶ 집합(Aggregation)

- 전체(whole)와 부분(part) 관계
- **부분(part)**이 전체(whole)에 대하여 **독립적**
- 전체(whole)가 부분(part)을 빌려서 사용하는 것과 비슷한 개념
- 전체(whole) 쪽에 다이아몬드 표기를 하고 부분(part)로 향하게 실선이 이어진다
- 부분(part) 방향으로 화살표를 표기해도 되고 하지 않아도 상관없다
- ❖ **연관(Association)과 코드에서는 차이를 구분하기 힘들다**



▶ 합성(Composition, Composite Aggregation)

- 전체(whole)와 부분(part) 관계
- 집합(Aggregation)보다 더욱 강한 집합을 의미
- **부분(part)**이 전체(whole)에 **종속**
- 전체(whole)가 부분(part)을 소유
- 전체(whole) 쪽에 내부가 채워진 다이아몬드 표기를 하고 부분(part)로 향하게 실선이 이어진다
- 부분(part) 방향으로 화살표를 표기해도 되고 하지 않아도 상관없다
- **집합(Aggregation)과의 차이점**
 - 전체(whole) 인스턴스가 부분(part) 인스턴스를 생성(**동적할당**)
 - 전체(whole) 인스턴스가 소멸되면 부분(part) 인스턴스도 **함께 소멸**
 - 전체(whole) 인스턴스가 복사되면 부분(part) 인스턴스도 **함께 복사(깊은 복사)**
 - 부분(part) 인스턴스는 공유되지 않는다(**private**)

