7주 2강

UML 기본 개념 1

이번 주차에는…

UML의 기본 개념1

- UML 이란
- UML 모델의 종류
- UML 작성법

1. UML 이란

- 정의
  - 프로그램 설계를 표현하기 위해 사용하는, 주로 그림으로 된 표기법을 의미
  - 객체지향 언어와 밀접한 관련이 있기에 객체지향 모델링 언어라고도 불린다.
  - 소프트웨어 시스템, 업무 모델링, 시스템의 산출물을 규정하고 시각화 하며 문서화하는 언어
  - 프로그램 언어는 아님
  - 모델링 언어 일뿐 방법론은 아님

- 사용하는 이유
  - 소프트웨어 시스템을 구축하기 전에 모델을 설계하는 것은 건물을 지을 때 청사진을 그리는 것과 마찬가지로 필수
  - 좋은 모델은 아키텍처를 건전하게 하고 프로젝트 팀의 의사 소통을 원활히 하는데 있음
  - 복잡한 시스템이 늘어나면서 시각적 모델링은 필수가 되고 있음

2. UML 모델의 종류(1)

- 클래스 다이어그램(Class Diagram)
  - 클래스 명세와 클래스 간의 관계를 표현
- 복잡 구조 다이어그램(Composite Structure Diagram)
  - 전체-부분 구조를 가진 클래스를 실행할 때의 구조를 표현
- 컴포넌트 다이어그램(Component Diagram)
  - 파일과 데이터베이스, 프로게스와 스레드 등의 소프트웨어 구조를 표현
- 디플로이먼트 다이어그램(Deployment Diagram)
  - 하드웨어와 네트워크 등 시스템의 물리 구조를 표현
- 객체 다이어그램(Object Diagram)
  - 인스턴스 간의 연관 관계를 표현
- 패키지 다이어그램(Package Diagram)
  - 패키지 간의 연관 관계를 표현

3. UML 모델의 종류(2)

- 액티비티 다이어그램(Activity Diagram)
  - 일련의 처리에 있어 제어의 흐름을 표현
- 시퀀스 다이어그램(Sequence Diagram)
  - 인스턴스 간의 상호 작용을 시계열로 표현
- 커뮤니케이션 다이어그램(Communication Diagram)
  - 인스턴스 간의 상호 작용을 주고 중심으로 표현
- 인터액션 오버뷰 다이어그램(Interaction Overview Diagram)
  - 조건에 따라 다르게 동작을 하는 스퀀스 다이어그램을 액티비티 다이어 그램 안에 포함하여 표현

4. UML 모델의 종류(3)

- 타이밍 다이어그램(Timing Diagram)
  - 인스턴스 간의 상태 전이와 상호 작용을 시간 제약으로 표현
- 유스케이스 다이어 그램(UseCase Diagram)
  - 시스템이 제공하는 기능을 이용자의 관계를 표현
- 스테이트 머신 다이어그램(State Machine Diagram)
  - 인스턴스의 상태 변화를 표현

5. UML 작성법

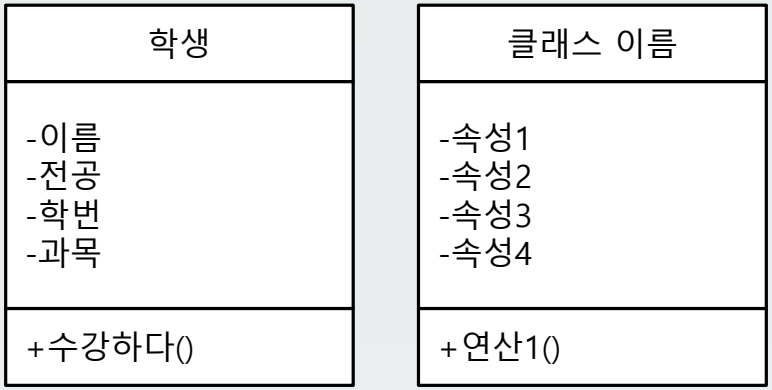
- UML(unified Modeling Language)이란
  - 시스템을 모델로 표현해주는 대표적인 모델링 언어
- UML 다이어그램의 종류
  - 구조 다이어그램(Structure Diagram)
    - 클래스 다이어그램, 객체 다이어그램, 복합체 다이어그램, 배치 다이어그램, 컴포넌트 다이어그램, 패키지 다이어그램
  - 행위 다이어그램(Behavior Diagram)
    - 활동 다이어그램, 상태 머신 다이어그램, 유즈 케이스 다이어그램, 상호 작용 다이어그램

6. 클래스 다이어그램(1)

- 클래스 다이어그램이란
  - 시간에 따라 변화지 않는 시스템의 정적인 면을 보여주는 대표적인 UML 구조 다이어그램
- 목적
  - 시스템을 구성하는 클래스들 사이의 관계를 표현함
- 클래스(Class)란
  - 동일한 속성과 행위를 수행하는 객체의 집합
  - 객체를 생성하는 설계도
    - 즉, 클래스는 공통의 속성과 책임을 갖는 객체들의 집합이자 실제 객체를 생성하는 설계도
- 클래스는 “변화의 기본 단위”
  - 디자인 패턴을 제대로 이해하려면 만들어진 프로그램을 흔들어보고 어떤 것이 변화 되는지를 잘 살펴봐야 함

7. 클래스 다이어그램(2)

- UML 클래스의 표현
  - 가장 윗부분 : 클래스 이름
  - 중간 부분 : 속성(클래스의 특징)
  - 마지막 부분 : 연산(클래스가 수행하는 책임)



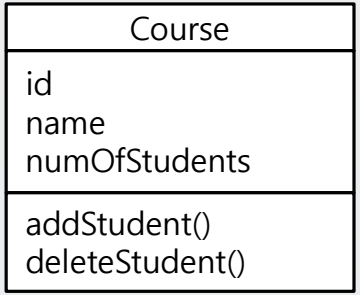
8. 클래스 다이어그램(3)

- 경우에 따라 속성 부분과 연산 부분은 생략 할 수 있음
- 속성과 연산의 가시화를 정의
  - UML에서는 접근 제어자를 사용해 나타냄

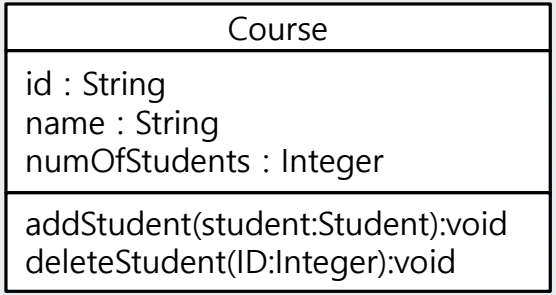
접근 제어자	표시	설명
public	+	어떤 클래스의 객체에서든 접근 가능
private	-	이 클래스에서 생성된 객체들만 접근 가능
protected	#	이 클래스와 동일 패키지에 있거나 상속 관계에 있는 하위 클래스의 객체들만 접근 가능
package	~	동일 패키지에 있는 클래스의 객체들만 접근 가능

9. 클래스 다이어그램(4)

- 분석 단계와 설계 단계에서의 클래스 다이어그램



분석 단계의 클래스



설계 단계의 클래스

10. 관계(1)

- UML에서 제공하는 클래스들 사이의 관계

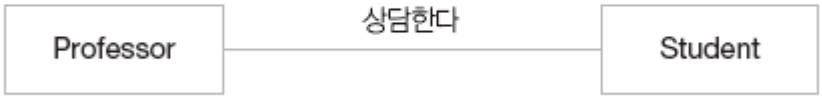
표 1-4 관계

관계	설명
연관 관계 (association)	클래스들이 개념상 서로 연결되었음을 나타낸다. 실선이나 화살표로 표시하며 보통은 한 클래스가 다른 클래스에서 제공하는 기능을 사용하는 상황일 때 표시한다.
일반화 관계 (generalization)	객체지향 개념에서는 상속 관계라고 한다. 한 클래스가 다른 클래스를 포함하는 상위 개념일 때 이를 IS-A 관계라고 하며 UML에서는 일반화 관계로 모델링한다. 속이 빈 화살표를 사용해 표시한다.
집합 관계 (composition, aggregation)	클래스들 사이의 전체 또는 부분 같은 관계를 나타낸다. 집약aggregation 관계와 합성composition 관계가 존재한다.
의존 관계 (dependency)	연관 관계와 같이 한 클래스가 다른 클래스에서 제공하는 기능을 사용할 때를 나타낸다. 차이점은 두 클래스의 관계가 한 메서드를 실행하는 동안과 같은, 매우 짧은 시간만 유지된다는 점이다. 점선 화살표를 사용해 표시한다.
실체화 관계 (realization)	책임들의 집합인 인터페이스와 이 책임들을 실제로 실현한 클래스들 사이의 관계를 나타낸다. 상속과 유사하게 빈 삼각형을 사용하며 머리에 있는 실선 대신 점선을 사용해 표시한다.

11. 연관관계

- 연관된 클래스 상에 실선을 그어 표시
- 두 클래스 상이의 관계가 명확한 경우에 이름을 사용하지 않아도 됨
- 한 클래스가 다른 클래스와 연관 관계 를 가지면 각 클래스의 객체는 해당 연관 관계에서 어떤 역할 을 수행하게 된다.
  - 두 클래스 사이의 연관 관계가 명확한 경우에는 연관 관계 이름 을 사용하지 않아도 된다.
  - 역할 이름 은 실제 프로그램을 구현할 때 연관된 클래스의 객체들이 서로를 참조할 수 있는 속성의 이름으로 활용할 수 있다.
  - 연관 관계는 방향성을 가질 수 있다. 양방향은 실선으로, 단방향은 화살표로 표시한다.

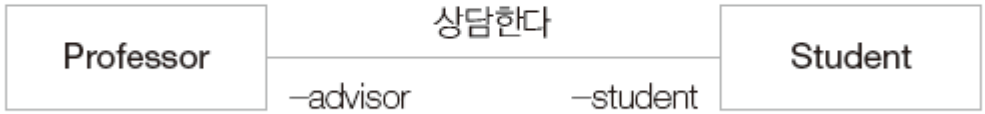
그림 1-8 Professor 클래스와 Student 클래스의 연관 관계



12. 연관 관계에서의 역할

- 연관 관계에서 각 클래스 객체의 역할은 클래스 바로 옆 연관 관계를 나타내는 선 가까이 기술
- 역할 이름은 연관된 클래스의 객체들이 서로를 참조할 수 있는 속성의 이름으로 활용

그림 1-9 연관 관계에서의 역할

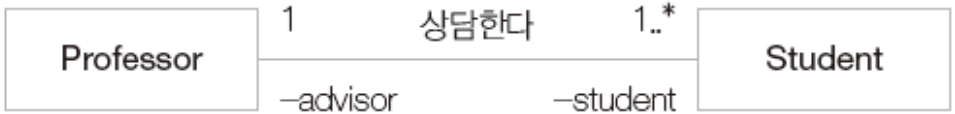


13. 다중성

표 1-5 다중성 표시

다중성 표기	의미
1	엄밀하게 1
*	0 또는 그 이상
0..*	0 또는 그 이상
1..*	1 이상
0..1	0 또는 1
2..5	2 또는 3 또는 4 또는 5
1, 2, 6	1 또는 2 또는 6
1, 3..5	1 또는 3 또는 4 또는 5

그림 1-10 다중성 예



14. 양방향, 단방향 연관관계

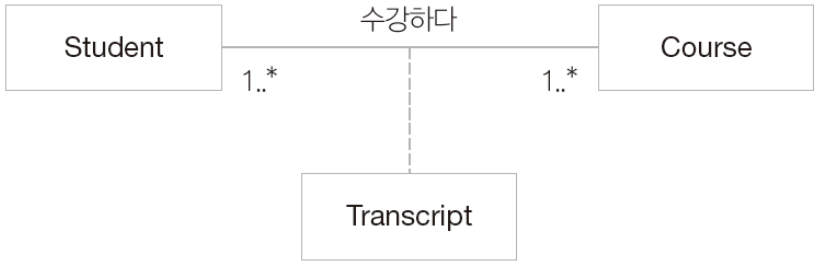
- 양방향 연관 관계
  - 두 클래스를 연경한 선에 화살표를 사용하지 않음
  - 서로의 존재를 인지
- 단방향 연관 관계
  - 한쪽으로만 방향성이 있는 관계
  - 한 쪽은 알지만 다른 쪽은 상대방의 존재를 모른다는 의미

그림 1-11 단방향 연관 관계

15. 연관 클래스(1)

- 연관 관계에 추가할 속성이나 행위가 있을 때 사용
- 연관 클래스를 일반 클래스로 변환
  - 연관 클래스는 연관 관계가 있는 두 클래스 사이에 위치하며, 점선을 사용해 연결한다.
  - 이 연관 클래스를 일반 클래스로 변환하여 다대다에서 일대다 연관 관계로 변환한다.

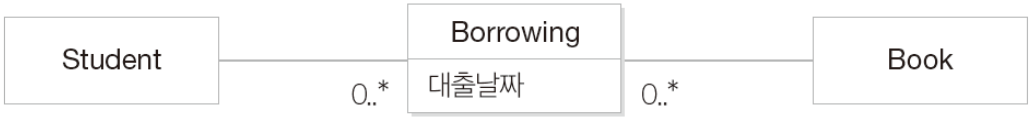
그림 1-13 연관 클래스



16. 연관 클래스(2)

- 사건 이력(event history) 표현
- 대출 이력

그림 1-15 이력의 표현



17. 재귀적 연관 관계

- 동일한 클래스에 속한 객체들 사이의 연관 관계
- 역할을 클래스로 할 때 문제가 발생

그림 1-16 직원 역할을 클래스로 모델링

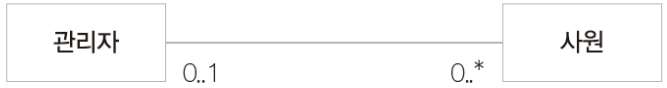
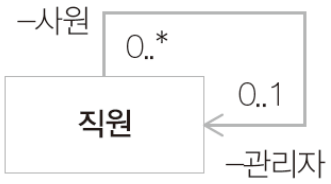
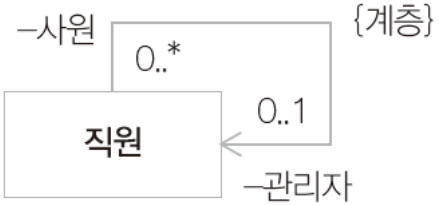


그림 1-17 재귀적 연관 관계



18. 제약 설정

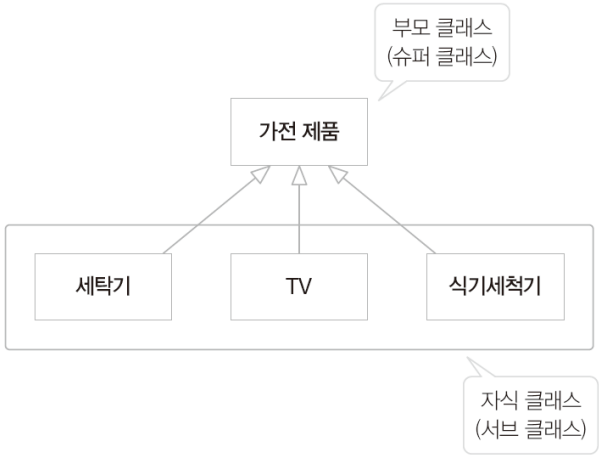
그림 1-18 재귀적 연관 관계에서 제약 설정



19. 일반화(1)

- 한 클래스가 다른 클래스를 포함하는 상위 개념일 때 두 클래스 사이에는 일반화 관계가 존재한다.
  - 객체지향 개념에서는 일반화 관계를 상속 관계(“is a kind of” 관계) 라고 한다.

그림 1-19 일반화 관계



20. 일반화(2)

- 부모 클래스
  - 추상적인 개념(실제로 존재하지 않는다.)
  - 삼각형 표시가 있는 쪽
  - 가전 제품
- 자식 클래스
  - 구체적인 개념
  - 삼각형 표시가 없는 쪽
  - 세탁기, TV, 식기 세척기
    - ✓ '세탁기' is a king of '가전 제품'
    - ✓ 'TV' is a king of '가전 제품'
    - ✓ '식기 세척기' is a king of '가전 제품'
- 부모 클래스는 자식 클래스의 공통 속성이나 연산을 제공하는 틀이다.
- 예를 들어, 가전 제품 클래스에 제조번호, 제조년도, 제조회사와 같은 공통 속성과 turnOn, turnOff와 같은 공통 연산을 두고 이를 상속받아 세탁기, TV, 식기 세척기와 같은 자식 클래스에서 사용하면 된다.

21. 일반화(3)

- 추상 클래스
  - 추상 메서드를 하나 이상 가지는 클래스
  - 추상 메서드
    - 부모 클래스에서 구현되지 않은 빈 껍데기만 있는 연산
    - 예를 들어, 위의 예에서 turnOn과 turnOff는 자식 클래스마다 다르기 때문에 부모 클래스인 가전 제품에서 해당 연산에 대한 정의를 하지 않고 빈 껍데기만 있는 연산(추상 메서드)를 제공한다.
  - 추상 클래스는 다른 일반적인 클래스와는 달리 객체를 생성할 수 없다.
  - UML에서의 추상 클래스와 추상 메서드 표현
    - 이탤릭체
    - 스테레오 타입('≪', '≫' 기호 안에 원하는 이름을 넣음)

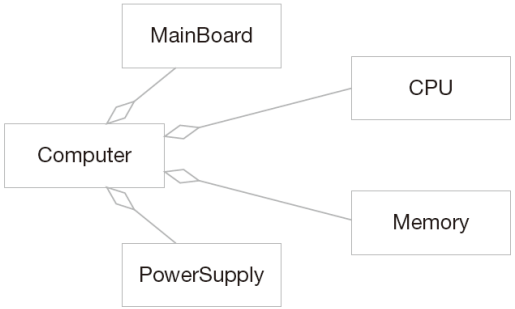
22. 집합 관계

- 전체와 부분 간의 관계
- 집약(aggregation):
  - 전체를 나타내는 객체와 부분을 나타내는 개체의 라이프 타임이 독립적
  - 부분을 나타내는 객체를 다른 객체와 공유 가능
  - 빈 마름보로 표시
- 합성(composition)
  - 전체를 나타내는 객체에 부분을 나타내는 개체의 라이프 타임이 종속적
  - 전체 객체가 사라지면 부분 객체도 사라짐
  - 채워진 마름보로 표시

23. 집약 관계

- 한 객체가 다른 객체를 포함하는 것
  - '부분'을 나타내는 객체를 다른 객체와 공유할 수 있다.
- '전체'를 가리키는 클래스 방향에 빈 마름모로 표시
- 전체 객체의 라이프타임과 부분 객체의 라이프 타임은 독립적이다.
  - 전체 객체가 메모리에서 사라진다 해도 부분 객체는 사라지지 않는다.
- • 예시
  - 생성자에서 참조값을 인자로 받아 필드를 세팅한다.

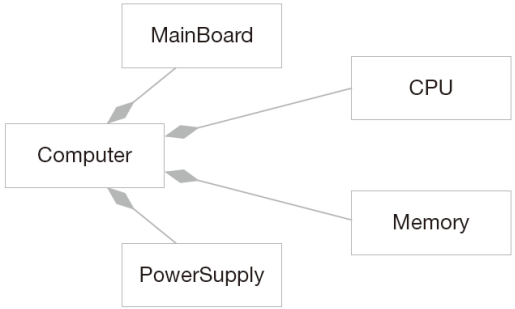
그림 1-21 집약 관계



24. 합성 관계

- 부분 객체가 전체 객체에 속하는 관계
  - '부분'을 나타내는 객체를 다른 객체와 공유할 수 없다.
- '전체'를 가리키는 클래스 방향에 채워진 마름모로 표시
- 전체 객체의 라이프타임과 부분 객체의 라이프 타임은 의존적이다.
  - 전체 객체가 없어지면 부분 객체도 없어진다.
- 예시
  - 생성자에서 필드에 대한 객체를 생성한다.

그림 1-20 합성 관계

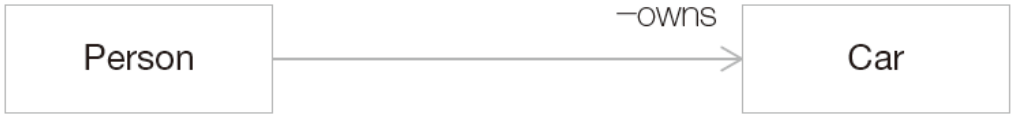


25. 의존 관계(1)

- 한 클래스에서 다른 클래스를 사용하는 경우
  - 클래스의 속성에서 참조
  - 연산의 인자로 참조
  - 메소드의 지역 개체로 참조

지속적 관계

그림 1-22 속성을 통한 연관 관계



26. 의존 관계(2)

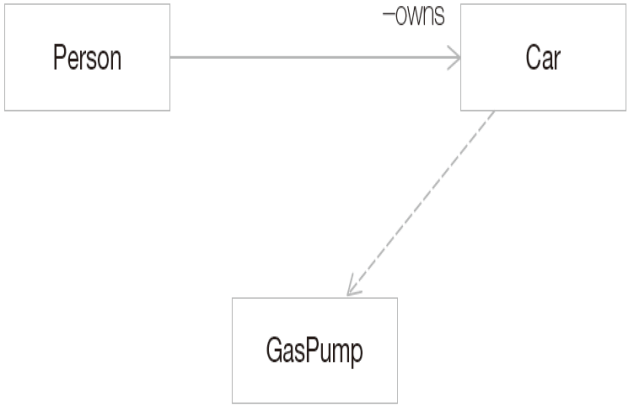
- 연산의 인자나 메소드의 지역 개체로 참조
  - 찰나적 관계

그림 1-23 의존 관계

```
public class Car {
    ...

    public void fillGas(GasPump p) {
        p.getGas(amount);
        ...
    }
}
```

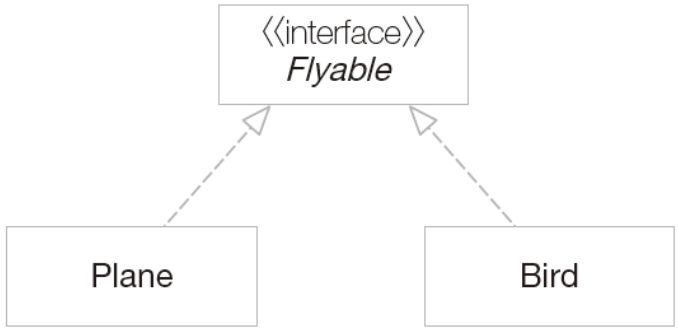
그림 1-24 의존 관계와 연관 관계



27. 인터페이스와 실체화 관계

- 인터페이스란 책임이다.
  - 리모콘의 책임은 가전 기기를 켜거나 끄거나 볼륨을 높이거나 낮춘다
- 어떤 공통되는 능력이 있는 것들을 대표하는 관점

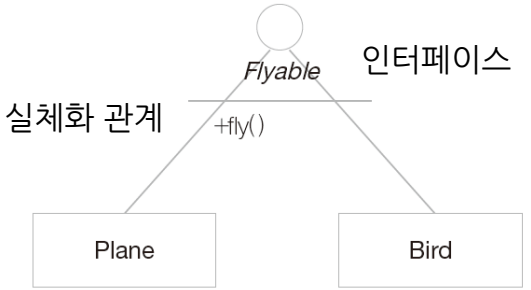
그림 1-25 인터페이스와 실체화 관계



28. 인터페이스와 실체화 관계

- 인터페이스란
  - 책임이다.
  - 어떤 객체의 책임이란 객체가 해야 하는 일 또는 객체가 할 수 있는 일
  - 즉, 객체가 외부에 제공하는 서비스나 기능은 객체가 수행하는 책임으로 본다.
  - 어떤 공통되는 능력이 있는 것들을 대표하는 관점
- UML에서의 인터페이스 표현
  - 인터페이스: 클래스에 사용하는 사각형을 그대로 사용하고 인터페이스 이름 위에 스테레오 타입으로 interface 표시(≪interface≫)
  - 인터페이스 관계: 빈 삼각형과 점선을 사용

그림 1-26 실체화 관계의 또다른 표현



다음 시간

UML 관련 툴 및 사용법