java uml class-diagram

[UML] 클래스 다이어그램 작성법

heejeong Kwon 04 Jul 2018

클래스 다이어그램 작성법

Goal

> - 클래스 다이어그램을 이해할 수 있다.
> - 클래스 다이어그램을 작성할 수 있다.

[들어가기 전]

- UML(Unified Modeling Language)이란
  - 시스템을 모델로 표현해주는 대표적인 모델링 언어
- UML 다이어그램의 종류
  i. 구조 다이어그램(Structure Diagram)
    - **클래스 다이어그램,** 객체 다이어그램, 복합체 구조 다이어그램, 배치 다이어그램, 컴포넌트 다이어그램, 패키지 다이어그램

  ii. 행위 다이어그램(Behavior Diagram)
    - 활동 다이어그램, 상태 머신 다이어그램, 유즈 케이스 다이어그램, 상호작용 다이어그램
- UML 작성 도구
  - http://staruml.io/
  - http://www.umlet.com/

클래스 다이어그램이란

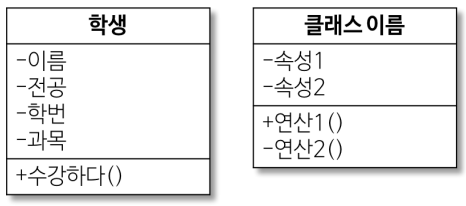
시간에 따라 변하지 않는 시스템의 정적인 면을 보여주는 대표적인 UML 구조 다이어그램

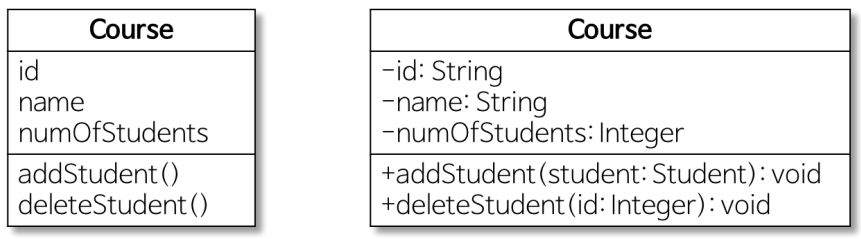
- 목적: 시스템을 구성하는 클래스들 사이의 관계를 표현한다.

클래스

- 클래스(Class)란
  i. 동일한 속성과 행위를 수행하는 객체의 집합

  ii. 객체를 생성하는 설계도
  - 즉, 클래스는 공통의 속성과 책임을 갖는 객체들의 집합이자 실제 객체를 생성하는 설계도이다.
- 클래스는 **“변화의 기본 단위”**
  - 디자인 패턴을 제대로 이해하려면 만들어진 프로그램을 흔들어보고 어떤 것이 변화되는지를 잘 살펴봐야 한다.
- UML 클래스의 표현
  - 가장 윗부분: 클래스 이름
  - 중간 부분: 속성(클래스의 특징)

- 마지막 부분: 연산(클래스가 수행하는 책임)
- 

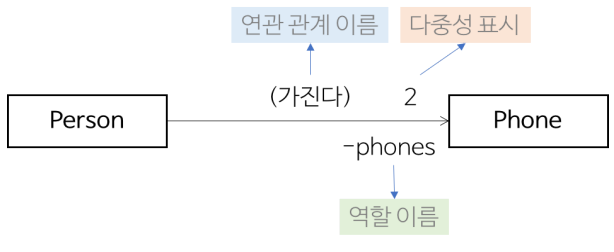
- 경우에 따라 속성 부분과 연산 부분은 생략할 수 있다.
- 속성과 연산의 가시화를 정의
  - UML에서는 접근제어자를 사용해 나타낸다.
접근 제어자	표시	설명
public	+	어떤 클래스의 객체에서든 접근 가능
private	-	이 클래스에서 생성된 객체들만 접근 가능
protected	#	이 클래스와 동일 패키지에 있거나 상속 관계에 있는 하위 클래스의 객체들만 접근 가능
package	~	동일 패키지에 있는 클래스의 객체들만 접근 가능
- 분석 단계와 설계 단계에서의 클래스 다이어그램
  - 

    -분석 단계의 클래스- -설계 단계의 클래스-

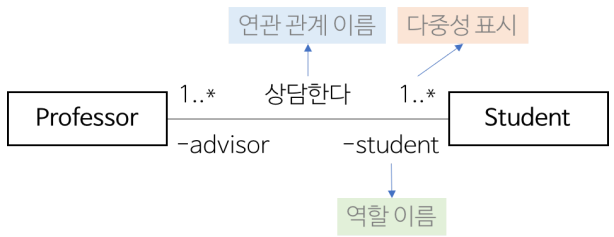
관계

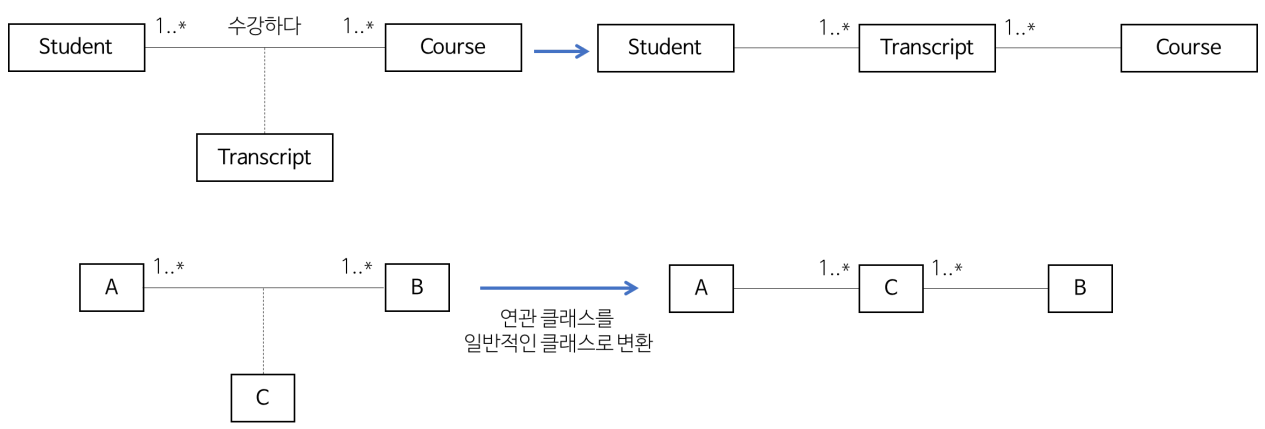
- UML에서 제공하는 클래스들 사이의 관계
관계		표시	설명
연관 관계 (association)		실선이나 화살표	클래스들이 **개념상 서로 연결**되었음을 나타낸다. 보통은 한 클래스가 다른 클래스에서 제공하는 기능을 사용하는 상황일 때 표시한다.
일반화 관계 (generalization)		속이 빈 화살표	객체지향 개념에서는 **상속 관계**라고 한다. 한 클래스가 다른 클래스를 포함하는 상위 개념일 때 이를 IS-A 관계라고 하며 UML에서는 일반화 관계로 모델링한다.
집합 관계	집약 관계 (aggregation)	속이 빈 다이아몬드	클래스들 사이의 전체 또는 부분 같은 관계를 나타낸다. 전체 객체의 라이프타임과 부분 객체의 라이프 타임은 **독립적**이다. 즉, 전체 객체가 없어져도 부분 객체는 없어지지 않는다.
	합성 관계 (composition)	속이 찬 다이아몬드	클래스들 사이의 전체 또는 부분 같은 관계를 나타낸다. 전체 객체의 라이프타임과 부분 객체의 라이프 타임은 **의존적**이다. 즉, 전체 객체가 없어지면 부분 객체도 없어진다.
의존 관계 (dependency)		점선 화살표	연관 관계와 같이 한 클래스가 다른 클래스에서 제공하는 기능을 사용할 때를 나타낸다. 차이점은 두 클래스의 관계가 한 메서드를 실행하는 동안과 같은, **매우 짧은 시간만 유지**된다는 점이다.
실체화 관계 (realization)		빈 삼각형과 점선	책임들의 집합인 **인터페이스**와 이 책임들을 실제로 실현한 **클래스들 사이의 관계**를 나타낸다.

1. 연관 관계

- 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.
- 화살표
  - 단방향 연관 관계
    - 한 쪽은 알지만 다른 쪽은 상대방의 존재를 모른다.
    - 

- 실선
  - 양방향 연관 관계
    - 두 클래스의 객체들이 서로의 존재를 인식한다.
    -



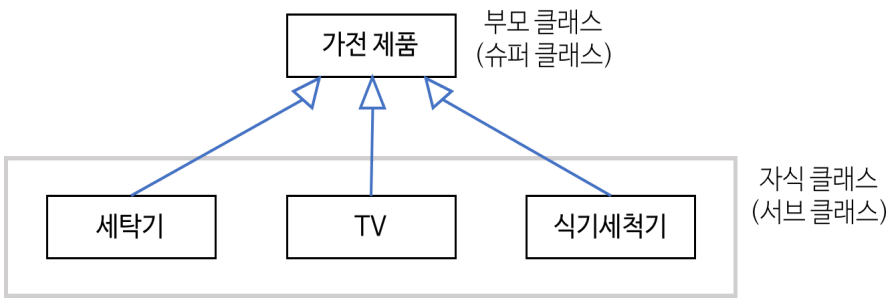
    - 일반적으로 **다대다** 연관 관계는 양방향 연관 관계로 표현되는 것이 적절하다.
    - 하지만 양방향 연관 관계를 구현하는 것은 복잡하기 때문에 보통 다대다 연관 관계를 **일대다 단방향 연관 관계** 로 변환해 구현한다. -> 연관 클래스
- 연관 클래스
    - 연관 관계에 추가할 속성이나 행위가 있을 때 사용
    - 연관 클래스를 일반 클래스로 변환
        - 연관 클래스는 연관 관계가 있는 두 클래스 사이에 위치하며, 점선을 사용해 연결한다.
        - 이 연관 클래스를 일반 클래스로 변환하여 **다대다에서 일대다** 연관 관계로 변환한다.
        - 

- 다중성 표시 방법
    -

다중성 표기	의미
1	엄밀하게 1
* 혹은 0..*	0 또는 그 이상
1..*	1 또는 그 이상
0..1	0 또는 1
1, 2, 6	1 또는 2 또는 6

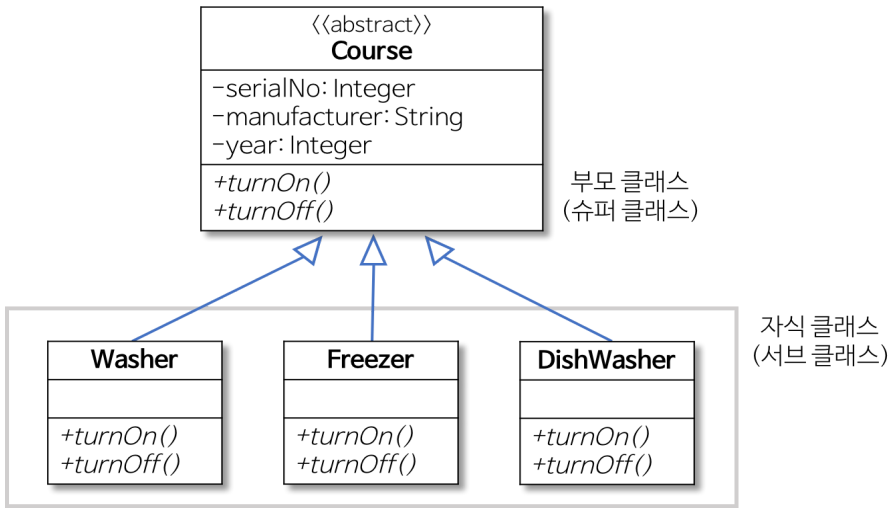
    - 선에 아무런 숫자가 없으면 **일대일** 관계

2. 일반화 관계

- 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.
    - 

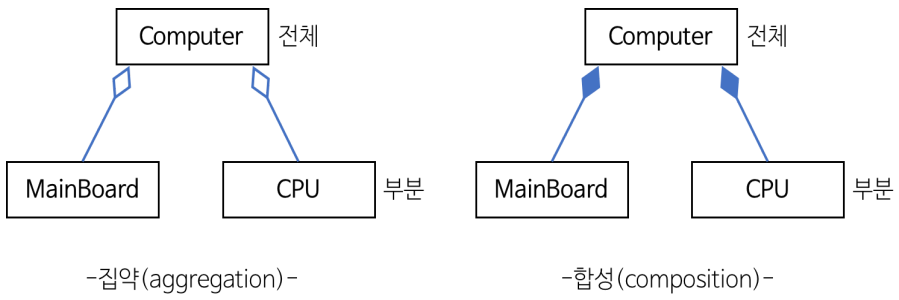
        - 부모 클래스
            - 추상적인 개념(실제로 존재하지 않는다.)
            - 삼각형 표시가 있는 쪽
            - 가전 제품
        - 자식 클래스
            - 구체적인 개념
            - 삼각형 표시가 없는 쪽
            - 세탁기, TV, 식기 세척기
                - ‘세탁기’ is a king of ‘가전 제품’
                - ‘TV’ is a king of ‘가전 제품’
                - ‘식기 세척기’ 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.
    - 추상 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)
    -



3. 집합 관계

- UML 연관 관계의 특별 경우로 **전체와 부분의 관계** 를 명확하게 명시하고자할 때 사용한다.
-



1. 집약 관계(aggregation)
    - 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.

```
public class Computer {
 private MainBoard mb;
 private CPU c;
 // 생성자
 public Computer(MainBoard mb, CPU c) {
 this.mb = mb;
 this.c = c;
 }
}
```

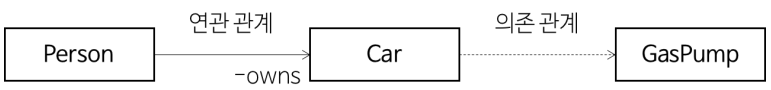
2. 합성 관계(composition)
    - 부분 객체가 전체 객체에 속하는 관계
        - ‘부분’을 나타내는 객체를 다른 객체와 공유할 수 없다.
    - ‘전체’를 가리키는 클래스 방향에 채워진 마름모로 표시
    - 전체 객체의 라이프타임과 부분 객체의 **라이프 타임은 의존적이다.**
        - 전체 객체가 없어지면 부분 객체도 없어진다.
    - 예시
        - 생성자에서 필드에 대한 객체를 생성한다.

```
public class Computer {
 private MainBoard mb;
 private CPU c;
```

```
  // 생성자
  public Computer() {
  this.mb = new MainBoard();
  this.c = new CPU();
  }
}
```

4. 의존 관계

- 일반적으로 한 클래스가 다른 클래스를 사용하는 경우
  i. 클래스의 속성("멤버 변수")에서 참조할 때
  ii. 연산의 "인자"(참조값)로 사용될 때
  iii. 메서드 내부의 "지역 객체"로 참조될 때
  - 1번: 연관 관계 / 2,3번: 의존 관계
- 연관 관계와 의존 관계의 차이
-



  - 연관 관계
    - 오랜 시간 동안 같이할 객체와의 관계
    - 예를 들어, 자동차(Car)와 소유한 사람(Person)의 관계

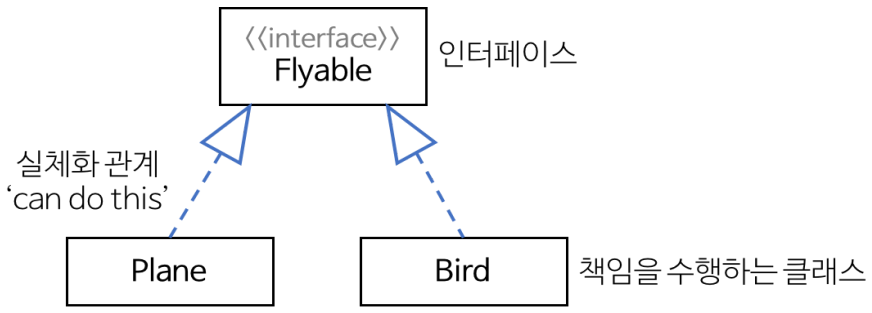
```
public class Person {
  // 클래스의 속성("멤버 변수")에서 참조
  private Car owns;
  // getter, setter
  public void setCar(Car car) {
    this.owns = car;
  }
  public Car getCar() {
    return this.owns;
  }
}
```

  - 의존 관계
    - 짧은 시간 동안 이용하는 관계
    - UML에서는 점선으로 나타낸다.
    - 예를 들어, 자동차(Car)와 주유기(GasPump)의 관계

```
public class Car {
  ...
  // 연산의 "인자"(참조값)로 사용
  public void fillGas(GasPump p) {
    p.getGas(amount);
  }
}
```

5. 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
  -



- 객체지향 개념에서는 실체화 관계를 **"can do this" 관계** 라고 한다.
  - 일반화 관계(상속 관계): "is a kind of" 관계

References

> - JAVA 객체지향 디자인 패턴, 한빛미디어

HeeJeong Kwon
computer programmer

[Java] OOP(객체지향 프로그래밍)의 특징
Goal OOP(객체지향 프로그래밍)의 4가지 특징을 이해한다. 추상화를 이해할 수 있다...

[Agile] 짝 프로그래밍(Pair Programming)이란
Goal Pair Programming의 정의 Pair Programming의 효과 ...