6주 3강

클래스의 관계와 설계 원칙

이번 주차에는…

클래스의 관계와 설계 원칙

- 클래스간의 관계와 설계 원칙
- 클래스 다이어그램 기초
- 클래스와 가시성
- 연관 관계와 다중도
- 일반화 관계와 전체/부분 관계

1. 연관관계

- 연관 관계association relationship
  - 서로 알고 지내는 관계로, 하나의 클래스가 또 다른 클래스를 인지하고 있음을 의미
  - 두 클래스는 서로 메시지를 주고받으며 이용하는 관계
  - 연관 관계: 클래스 사이에서 발생, 링크(link): 객체 사이에서의 이용 관계

그림 6-43 연관 관계의 예 : 사장과 직원 관계

2. 일반화 - 특수화 관계

- 일반화-특수화 관계(IS-A, generalization-specialization relationship)
  - 두 클래스 간의 상속 관계
  - 하위 클래스는 상위 클래스의 각 속성과 메서드를 모두 상속받아 사용 가능
  - 하위 클래스: 원래 가지고 있던 속성과 연산+물려받은 속성과 연산까지 모두 사용

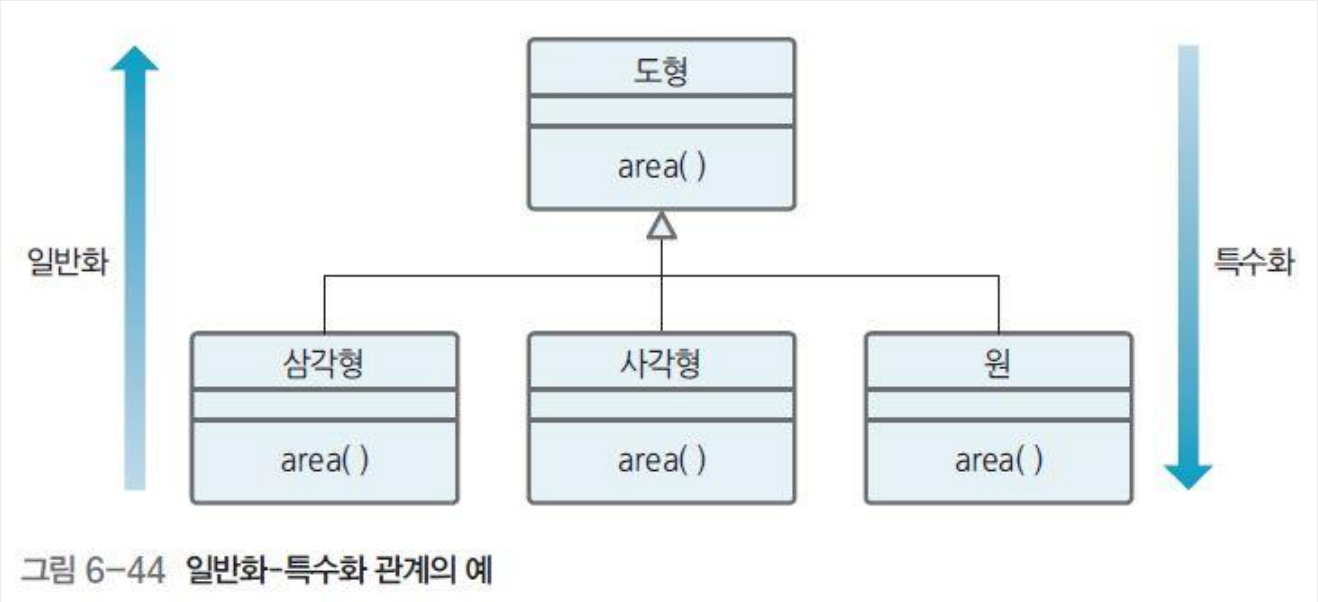


그림 6-44 일반화-특수화 관계의 예

3. 집합 관계

- 집합 관계aggregation relationship
  - 연관 관계를 더 구체적으로 나타낸 것
  - 거대한 객체 하나를 소규모 객체 여러 개로 구성할 때 발생
  - 전체와 부분 관계 성립
  - 집합 관계에 속한 부분 객체는 다른 곳에서도 공유 가능

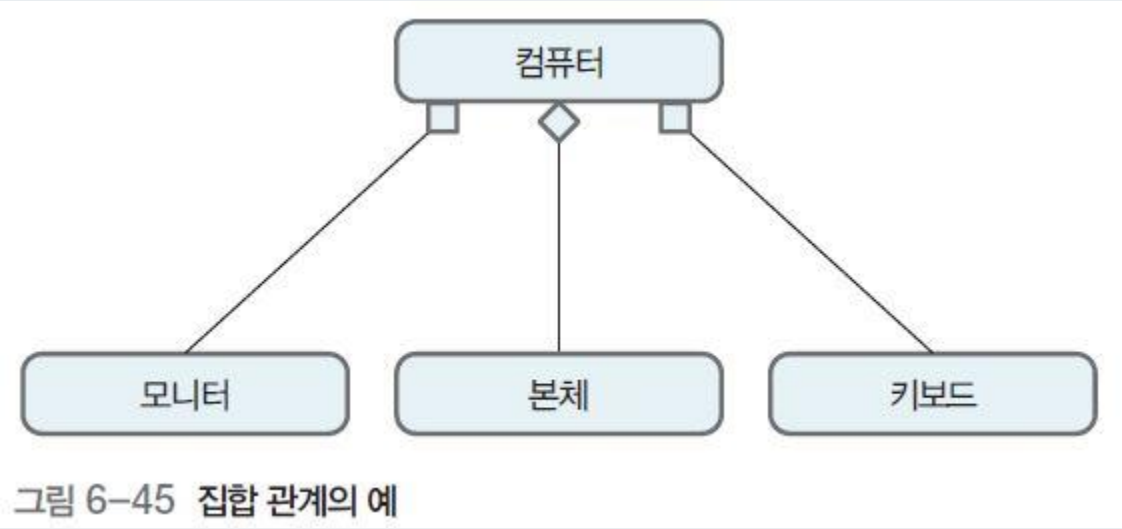


그림 6-45 집합 관계의 예

4. 포함 관계

- 포함 관계composition relationship
  - 전체 객체에 완전히 전속되어 독립된 객체로 존재할 수 없는 부분 객체가 존재하는 관계
  - 포함 관계의 부분 객체들은 전체 객체가 없어지면 같이 없어짐
    (예) 잉크가 떨어진 볼펜의 볼펜심만 따로 갈지 못해 통째로 버려야 하는 경우

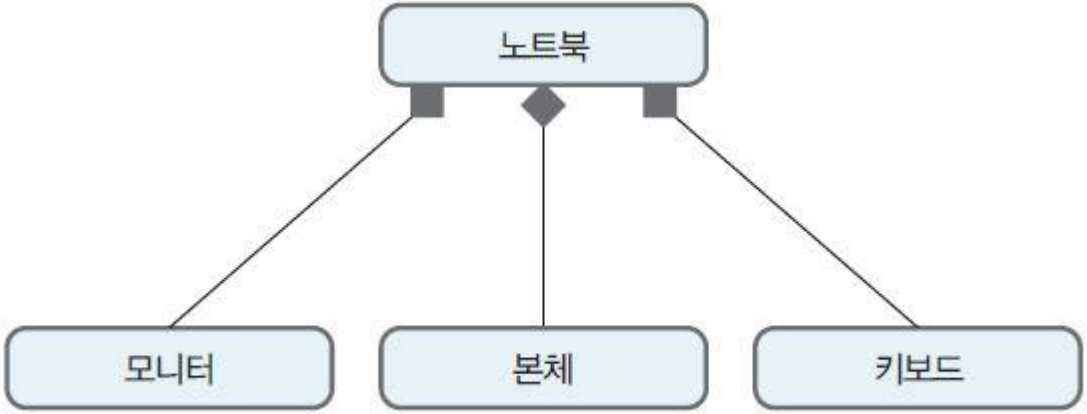


그림 6-46 포함 관계의 예

5. 클래스 설계의 원칙

- 단일 책임 원칙(SRP: Single-Responsibility Principle)
  - 클래스를 변경해야 하는 이유는 오직 하나여야 한다.
- 개방 폐쇄의 원칙(OCP: Open-Closed Principle)
  - 확장(상속)에는 열려 있어야 하고 변경에는 닫혀 있어야 한다.
- 리스코프 교체의 원칙(LSP: Liskov Substitution Principle)
  - 기반 클래스는 파생 클래스로 대체할 수 있어야 한다.
- 의존 관계 역전의 원칙(DIP: Dependency Inversion Principle)
  - 클라이언트는 구체 클래스가 아닌 추상 클래스(인터페이스)에 의존해야 한다.
- 인터페이스 분리의 원칙(ISP: Interface Segregation Principle)
  - 하나의 일반적인 인터페이스보다는 구 체적인 여러 개의 인터페이스가 낫다.

6. 단일 책임 원칙

- 클래스를 변경해야 하는 이유는 단 하나여야 한다
  - 좋은 설계: 모듈의 응집도를 높이고, 결합도를 낮추는 설계
  - 단일 책임 원칙: '클래스는 한 가지 책임(기능)만 갖도록 설계하자'
  - 만일 클래스에 두 개의 책임(기능)이 존재한다면: 두 개의 클래스로 분리하여 변경

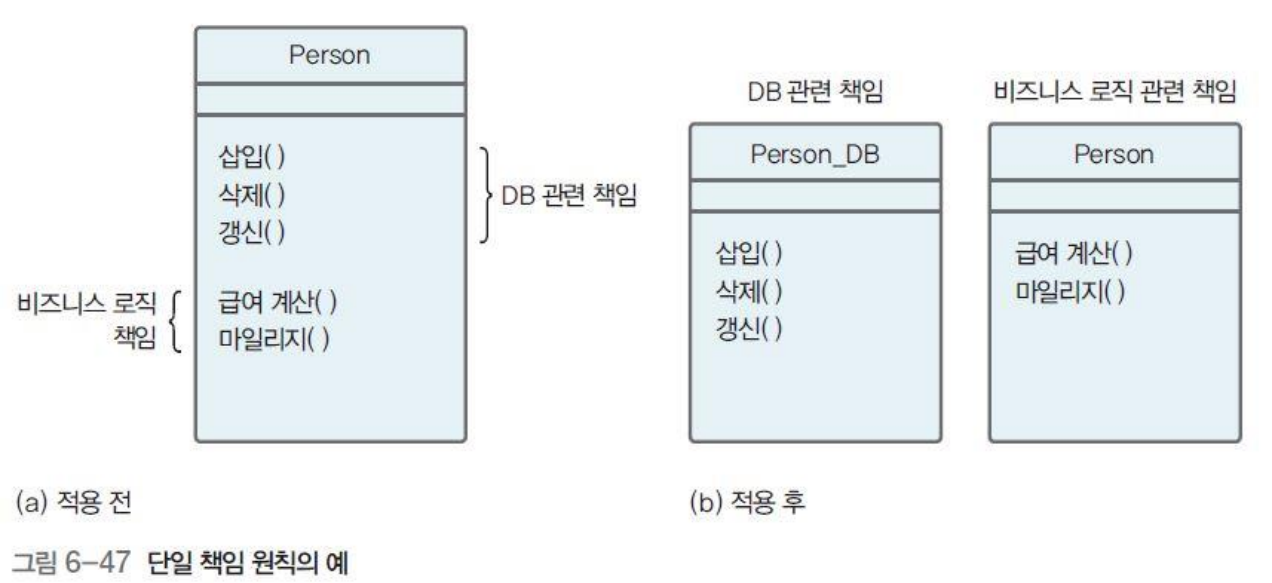


그림 6-47 단일 책임 원칙의 예

7. 개발 폐쇄의 원칙(1)

- 확장(상속)에는 열려 있어야 하고 변경에는 닫혀 있어야 한다
  - 개방 폐쇄의 원칙을 지키지 않고 설계하면 어떻게 될까?
    → 이는 하위 클래스의 특정 기능을 상위 클래스에서 미리 구현하는 것과 같다.
    → 이 기능이 필요 없는 다른 하위 클래스에서 강제로 상속받아야 하는 경우가 발생

자유롭게 상속하려면

추상 클래스와 구체 클래스를 분리

  - '변경에는 닫혀 있어야 한다'?
    → 특정 하위 클래스가 상위 클래스에 있는 공통의 기본 기능을 변경 및 간섭 금지
    → '자유로운 상속을 통한 확장과 재사용성'을 추구하기 위한 원칙

8. 개방 폐쇄의 원칙(2)

- 확장(상속)에는 열려 있어야 하고 변경에는 닫혀 있어야 한다
  - 개방 폐쇄의 원칙: '클래스를 확장은 쉽게, 변경은 어렵게 설계하자'는 것
    (예) '개방-확장은 쉽게': 휴대폰, '폐쇄-변경은 어렵게': 휴대폰 충전기
    → 휴대폰 기종은 바뀌어도 충전기 인테페이스만 같으면 충전기 계속 사용 가능

인터페이스는 자주 바뀌면 안 되고(폐쇄), 그 안은 얼마든지 확장할 수 있는 구조

가능한 방법?

9. 개방 폐쇄의 원칙(3)

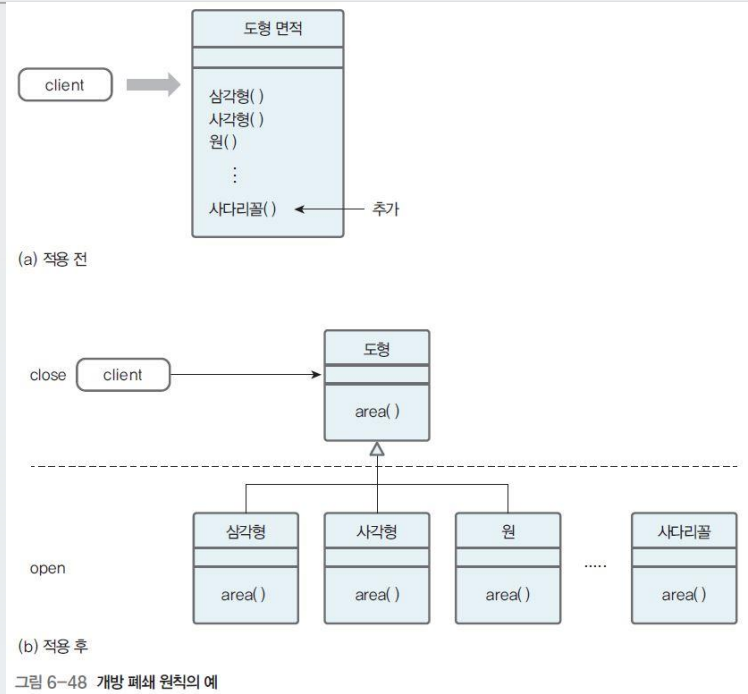


그림 6-48 개방 폐쇄 원칙의 예

- 개방 폐쇄의 원칙
  - 변경이 필요한 경우 기존 코드 변경 없이 상속과 확장을 통해 변경
  - 변경 시 영향 축소를 위해 중간에 추상 클래스-인터페이스 같은 완충 장치를 두는 것

10. 리스코프 교체의 원칙

- 기반 클래스는 파생 클래스로 대체할 수 있어야 한다
  - 기반 클래스가 들어갈 자리에 파생 클래스를 넣어도 문제없이 잘 작동함을 의미
    (예) 문서편집기: 구버전에서 작업한 내용을 신버전에서도 계속 사용할 수 있도록 설계
  - 서브 타입(상속받은 하위 클래스-신버전)은 어디에서나 자신의 기반타입(상위 클래스-구버전)으로 교체할 수 있어야 함을 의미
- 클래스 설계에서 이 원칙이 중요한 이유?
  - 파생 클래스(신버전)에서 기반 클래스의 모든 메서드(구버전)를 지원하지 않으면 상속의 기본인 IS-A 관계가 성립이 안 되기 때문

11. 의존 관계 역전의 원칙

- 클라이언트는 구체 클래스가 아닌 추상 클래스(인터페이스)에 의존해야 한다
  - 상속개념을 적용할 때 구체 클래스에서 상속을 받는 구조로 설계하지 말라는 것

이유?

구체 클래스는 추상 클래스(인터페이스)보다 변하기 쉽기 때문

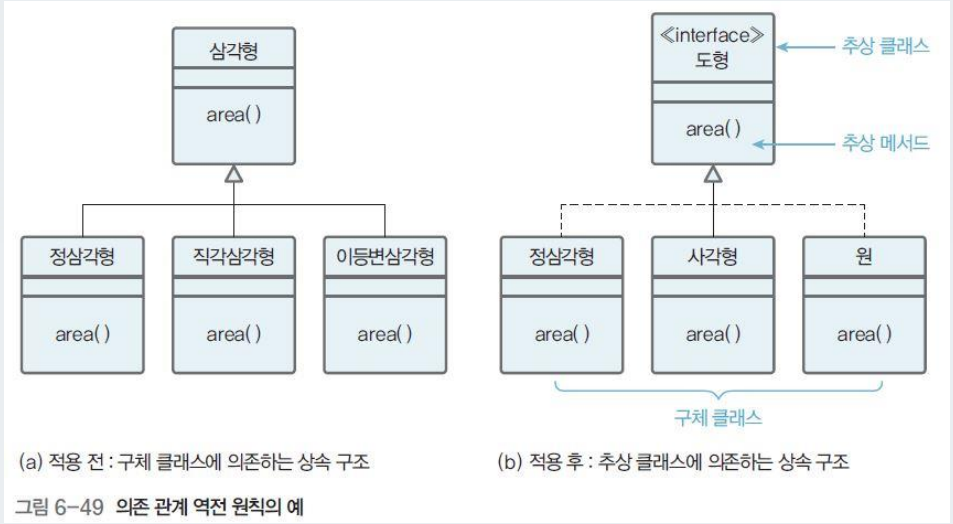


그림 6-49 의존 관계 역전 원칙의 예

12. 인터페이스 분리의 원칙(1)

- 하나의 일반적인 인터페이스보다는 구체적인 여러 개의 인터페이스가 낫다
  - '클라이언트는 자신이 사용하지 않는 메서드와 의존 관계를 맺으면 안 된다'

※ 클래스 분리 아님

각 클라이언트에 맞는 인터페이스만 분리

- 인터페이스를 분리하면?
  → 시스템 확장 시 변화의 폭을 필요한 기능에 대한 변경이나 확장으로 제한 가능
  → 많은 메서드로 인한 가독성 및 높은 결합도의 문제 해결

13. 인터페이스 분리의 원칙(2)

- 하나의 일반적인 인터페이스보다는 구체적인 여러 개의 인터페이스가 낫다

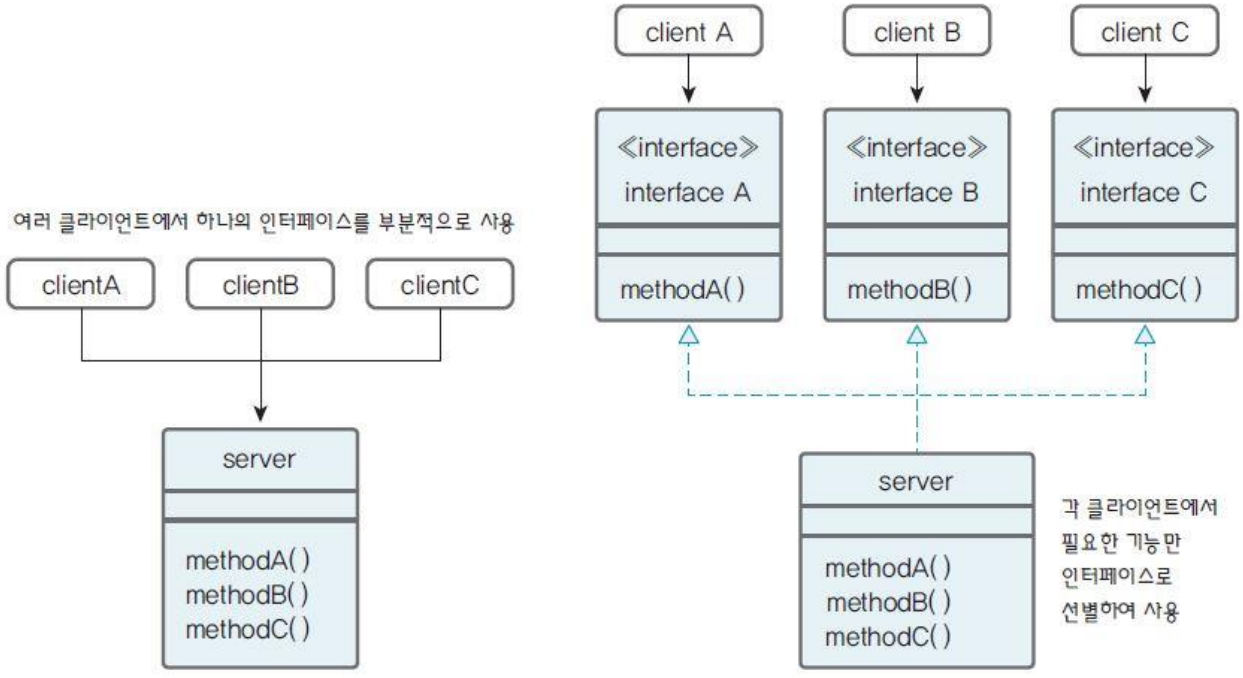


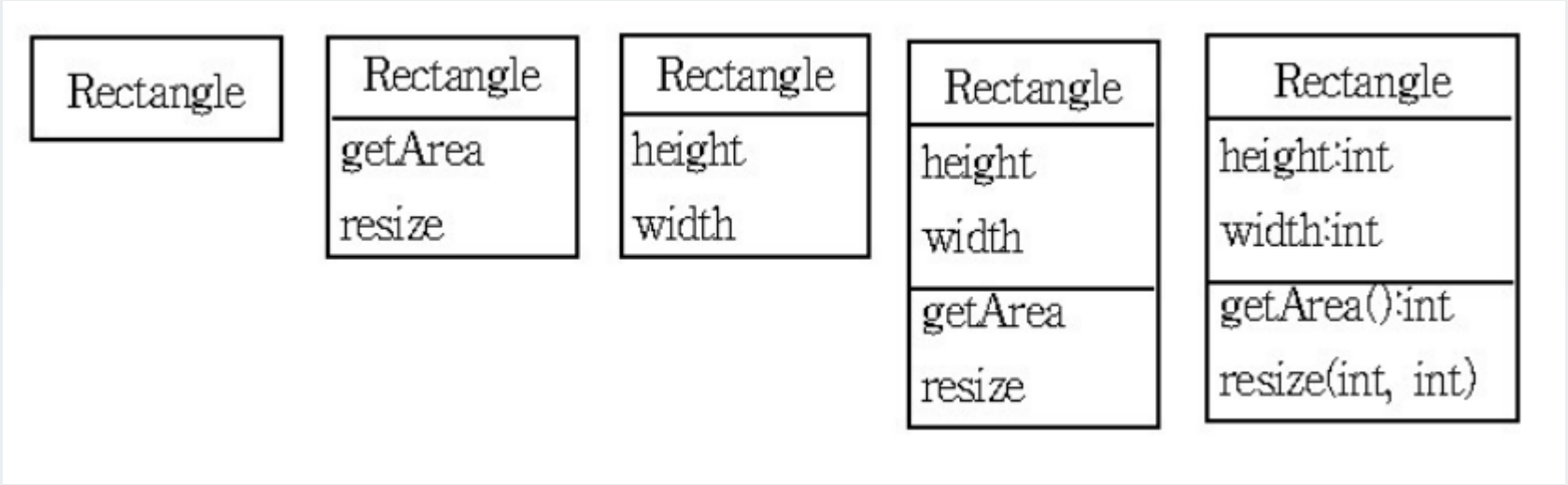
그림 6-50 인터페이스 분리의 원칙의 예

14. 클래스 다이어그램의 기초

- 클래스 다이어그램의 구성요소
  - 클래스 : 자료 타입 그 자체를 나타냄
  - 연관관계 : 클래스 인스턴스 사이의 관계를 나타냄
  - 속성 : 클래스와 그 인스턴스 안에 존재하는 단순 자료
  - 오퍼레이션 : 클래스와 그 인스턴스에서 수행될 함수
  - 일반화 : 클래스를 상속 구조로 구루핑

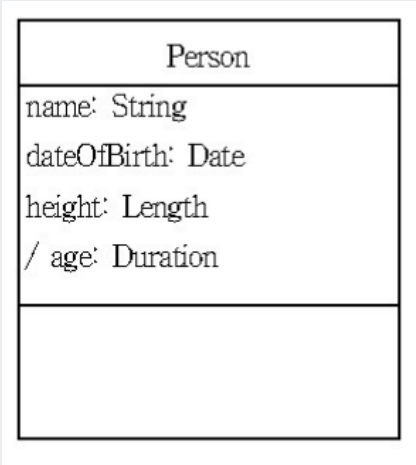
15. 클래스와 가시성

- 클래스는 박스로 표현하며 그 안에 이름을 적는다
  - 다이어그램은 속성과 오퍼레이션을 나타낼 수 있다.
  - 오퍼레이션의 원형은 다음과 같이 표현한다.
    - operationName(parameterName parameterType, …):returnType



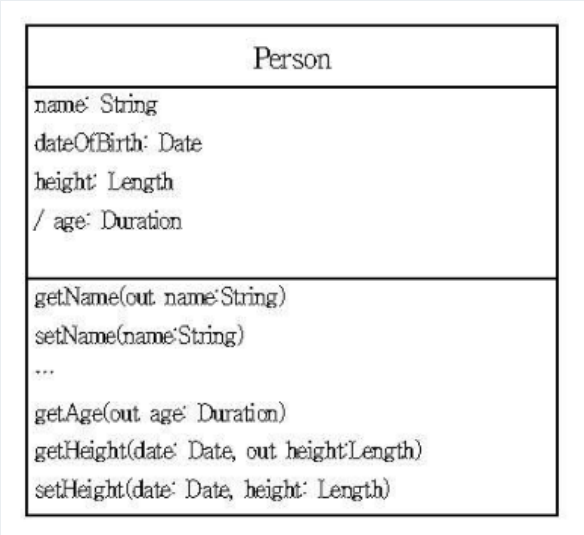
16. 속성

- 객체의 상태 또는 성질을 나타냄
  - 객체에 대한 정보를 나타냄
  - 속성은 변수와 동의어는 아님
  - 추상적으로 정의한 성질
  - 객체 외부에서 값을 가져갈 수도 있고 변경할 수 도 있음
  - 읽기 전용도 있음



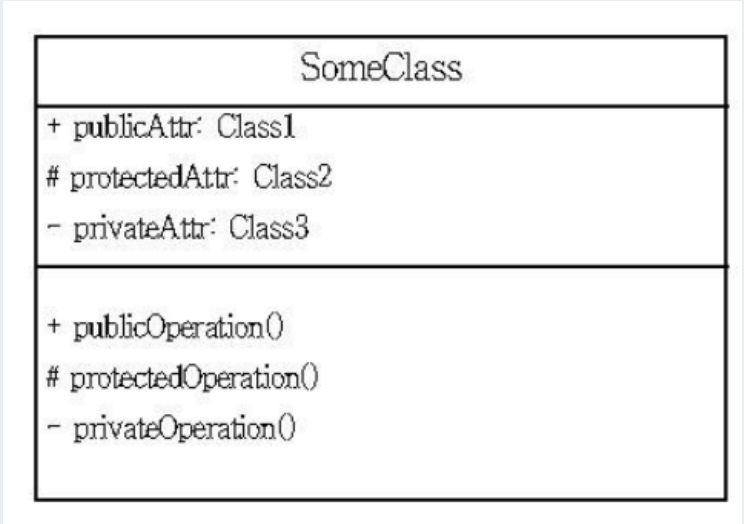
17. 오퍼레이션

- 박스 맨 아래 오퍼레이션 원형으로 표시
  - 대부분 속성에 대한 get(), set()
  - 읽기 전용은 get() 오퍼레이션 만 사용
  - 일부 오퍼레이션은 매개변수 필요
  - 일부 오퍼레이션은 다른 객체에 메시지 호출할 필요가 있음



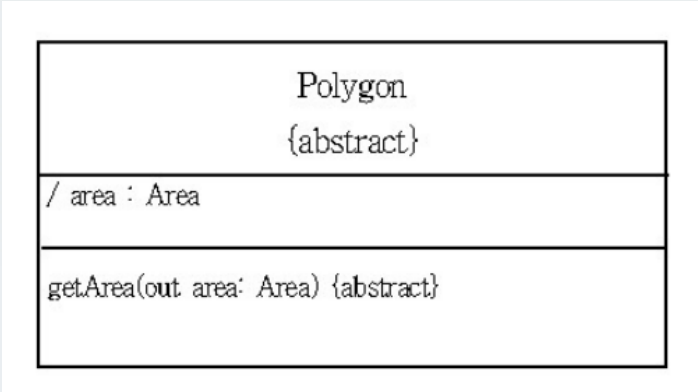
18. 가시성

- 속성과 오퍼레이션 앞에 visibility 표시
  - Public : '+'
  - Protected : '#'
  - Private : '-'



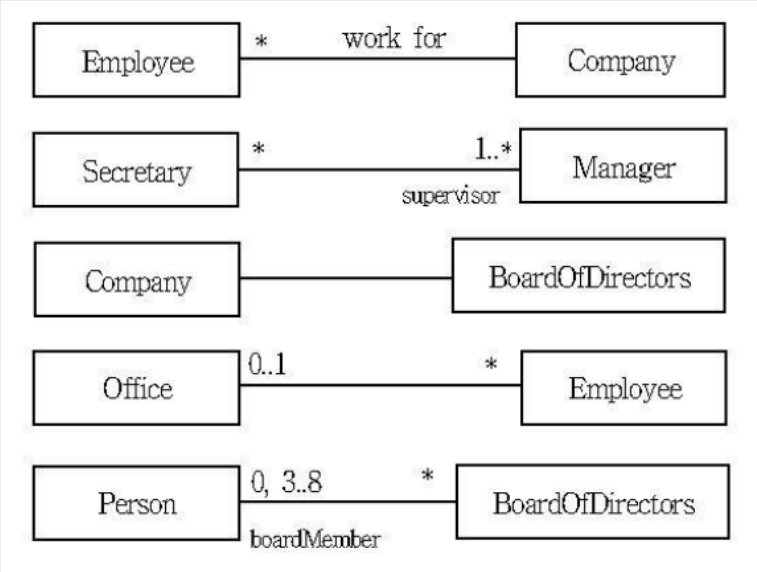
19. 추상 클래스

- 추상 오퍼레이션
  - 구현이 없는 오퍼레이션
- 추상 클래스
  - 인스턴스가 없는 클래스



20. 연관관계와 다중도

- 두 클래스가 서로 어떻게 관련이 있는지를 나타냄
  - 연관 관계의 양 끝에는 다중도를 표시
  - 연관 관계 특성을 드러내기 위해 이름을 붙임



21. 연관관계 검토(1)

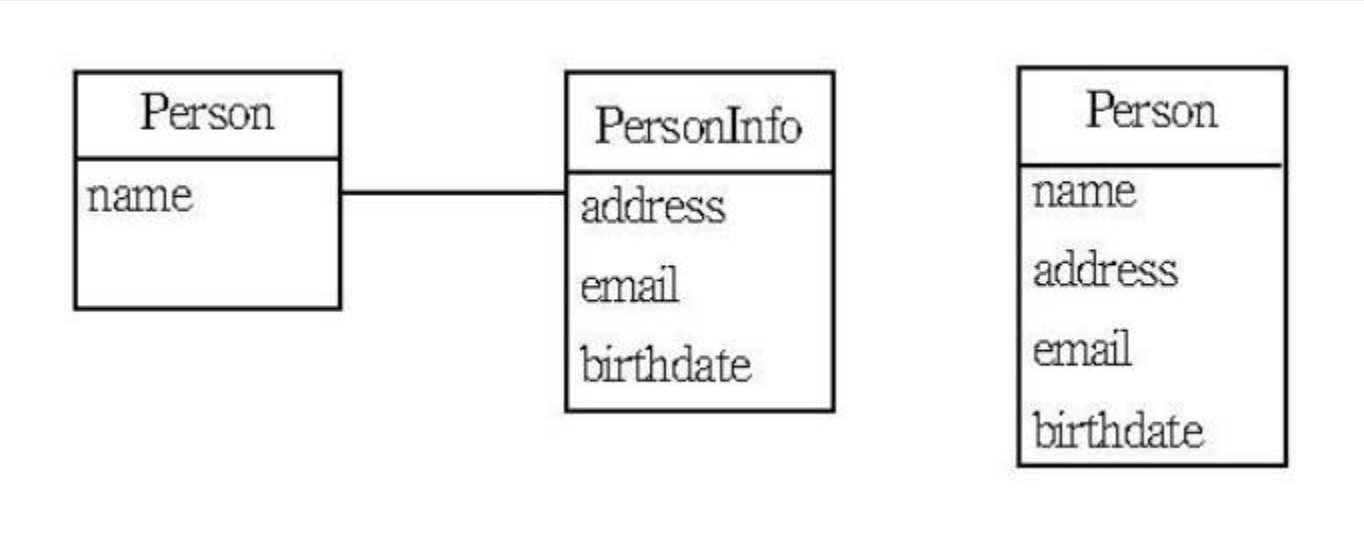
- One-to-one
  - 한 회사에 오직 한 개가 책임자 위원회가 있음
  - 책임자 위원회는 한 회사를 위한 위원회임
  - 회사는 항상 위원회를 가지고 있어야 함
  - 위원회는 어떤 회사의 소속 임

22. 연관관계 검토(2)

- Many-to-many
  - 비서 한명이 여러 명의 관리자를 위해 일함
  - 관리자 한명이 여러 명의 비서를 둘 수 있음
  - 비서는 인력풀에서 일함
  - 관리자들은 비서의 그룹을 가질 수 있음
  - 어떤 관리자는 비서가 없을 수도 있음
  - 관리자가 없는 비서는 일시적으로 가능한가?

23. 연관관계 검토(3)

- 불필요한 1 대 1 관계는 피함

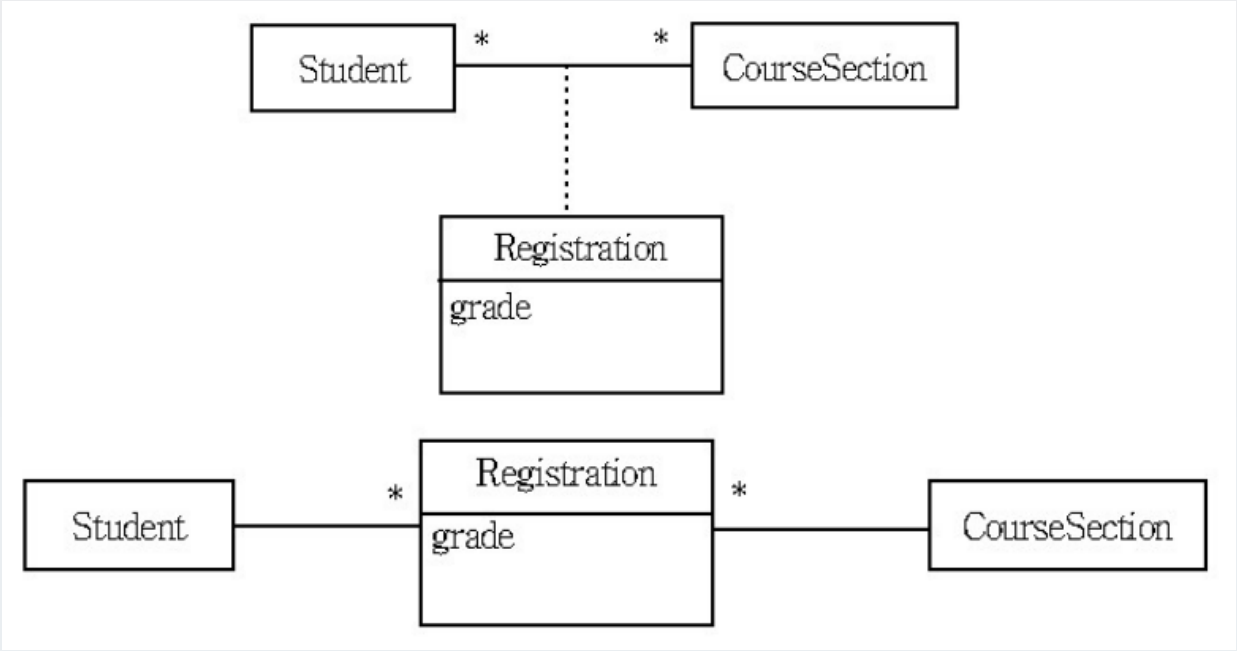


24. 복잡한 예

- 예약은 항상 단 하나의 탑승객에 대한것
  - 탑승객은 없는 예약은 없음
  - 예약에 탑승객이 여러 명인 경우는 없음
- 탑승객이 다수의 예약을 할 수 있음
  - 탑승객의 예약이 하나도 없을 수 있음
  - 탑승객이 다수의 예약이 있을 수 있음

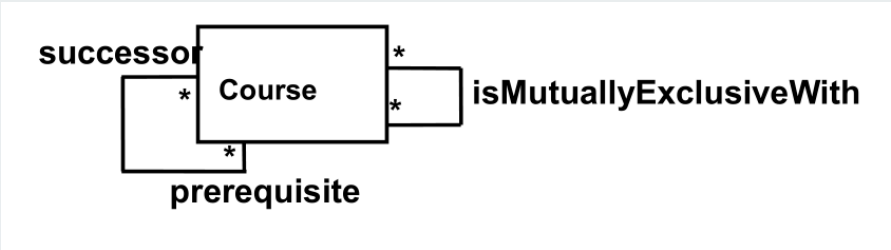
25. 연관 클래스

- 두개의 연관 클래스에 관한 속성을 어느 한쪽에 위치 시킬 수 없을 때
- 다음과 같은 경우



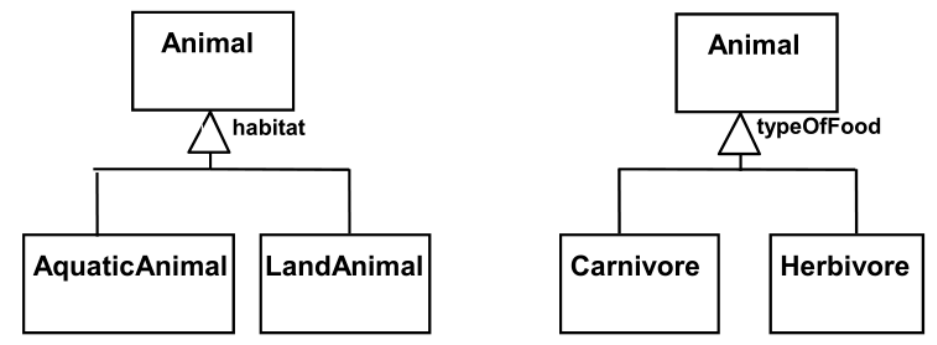
26. 재귀 연관 관계

- 클래스 자신과 연관 관계를 맺음



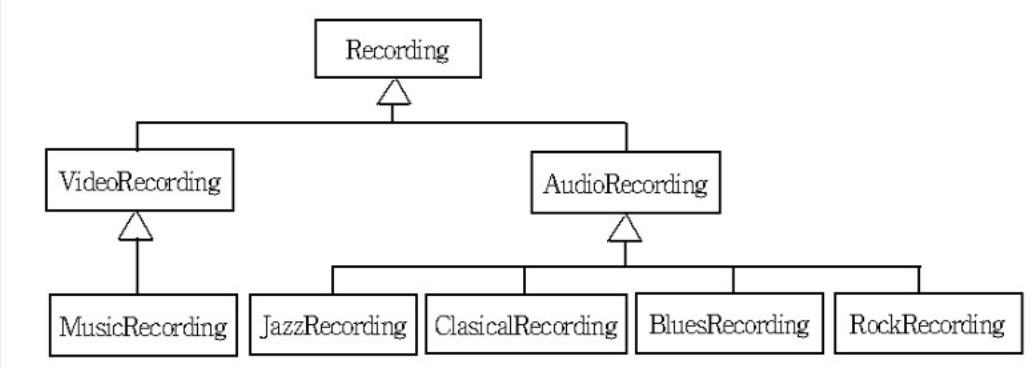
27. 일반화 관계와 전체/부분 관계

- 일반화 관계
  - 두가지 이상의 서브 클래스로 구체화 시키는 것
  - Discriminator : 상세화 할 때 기준을 나타내는 레이블

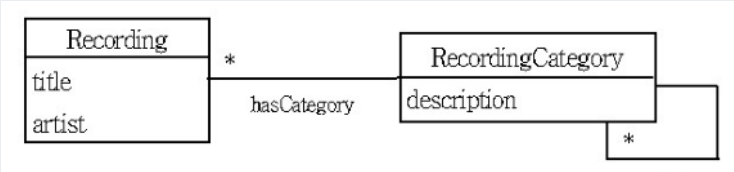


28. 불필요한 상속 피하기

- 인스턴스가 되어야 할 것을 무리하게 클래스 구조로 상속한 경우

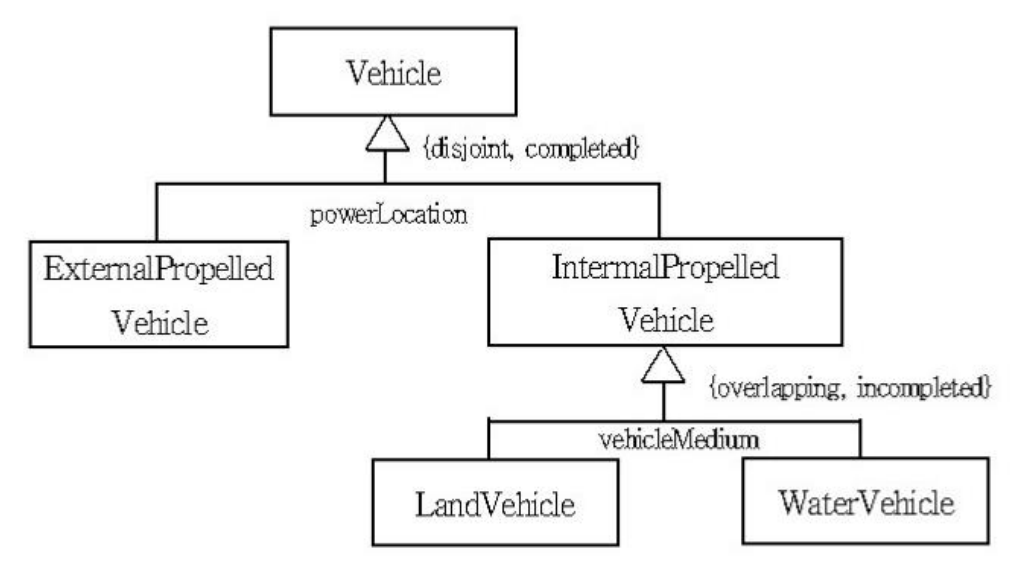


- 개선 후 클래스 다이어그램



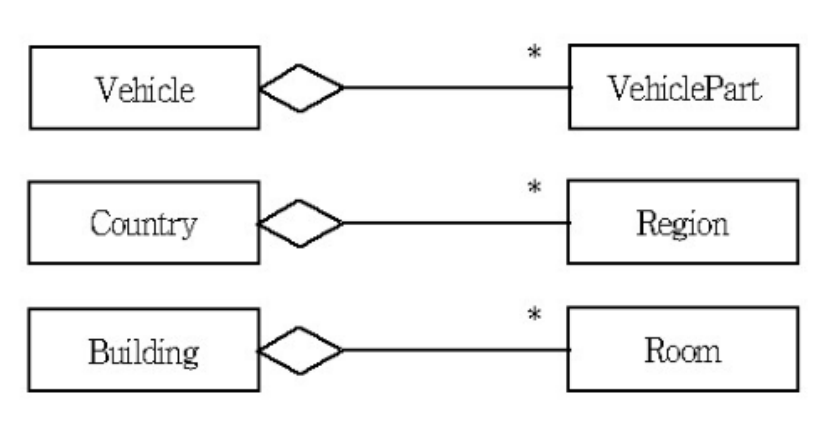
29. 구별자

- 구루핑 하는 기준
  - 동력의 위치에 따른 구별 - 내연, 외연
  - 운송 매체 - 육상, 해상



30. 전체 부분 관계 - Aggregation

- Aggregation - 전체부분관계를 나타내는 특수한 형태
  - 전체에 해당하는 클래스는 aggregate 또는 assembly라 부름
  - 다이어몬드 심볼은 isPartOf 라는 관계로 생각할 수 있음



31. Aggregate를 사용하는 경우

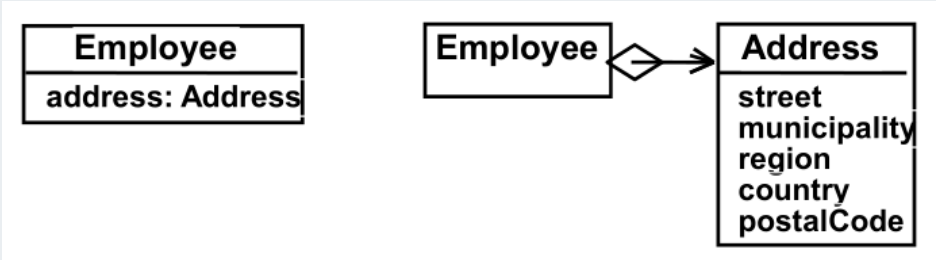
- 일반적인 경우 다음의 경우가 성립되면 aggregate를 사용
  - ‘is a part of’ 관계가 성립 할 때
  - ‘is composed of’ 관계가 성립 될 때

- 집합 개념을 관리하거나 소유하고 있다면 부분 개념도 소유

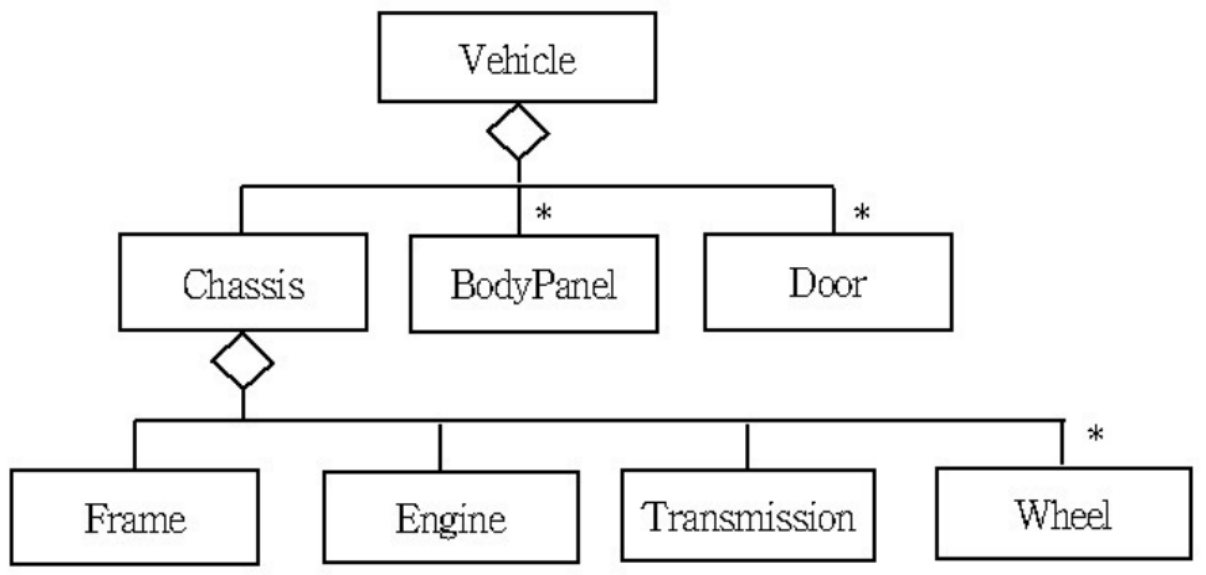
32. 합성 관계(Composition)

- 합성은 aggregation이 강한 관계
  - 집합이 소멸되면 부분도 따라서 소멸

  - 주소를 표현하는 두가지 방법



33. Aggregation의 구조



다음 시간

소프트웨어 구현(1)