연관(association)

- 가시성(visibility) : 객체의 접근 가능성

- 연관을 맺은 두 객체가 서로를 알게 하고 접근하게 하는 방법

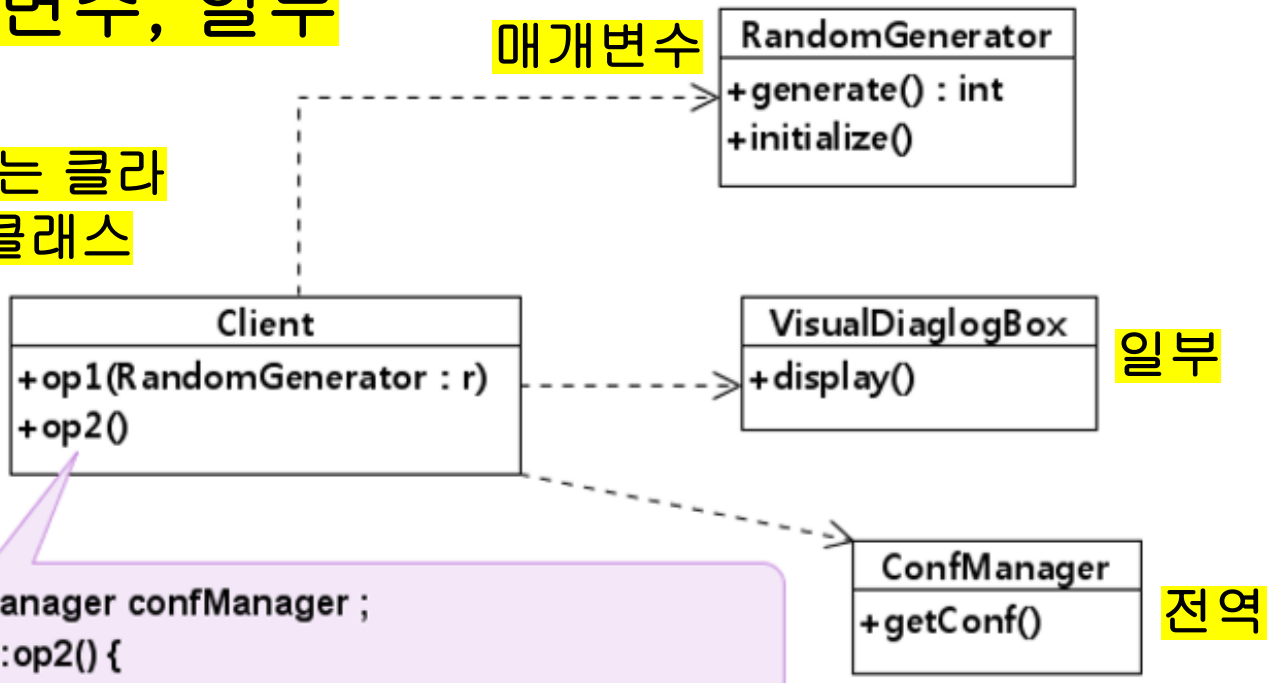
  - 연관된 객체(Course)를 **전역**으로 선언하여 클라이언트 객체(Student)가 접근할 수 있게 함
  - 연관된 객체(Course)를 클라이언트 객체(Student)의 메시지 호출 오퍼레이션의 **매개변수**로 만듦
  - 연관된 객체(Course)를 클라이언트 객체(Student)의 **일부로 만듦**

  의존(Dependency)라고 분류

  - 연관된 객체(Course)를 클라이언트 **객체(Student) 내에서 선언**

전역, 매개변수, 일부

"Client 클래스"는 클라이언트 객체의 클래스



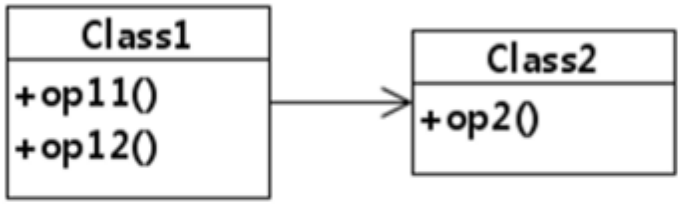

```
ConfManager confManager ;
Client::op2() {

  VisualDiaglogBox box = new VisualDialogBox() ;
  box.display() ;

  ...
  confManager.getConf() ;
}
```

일부

연관된 객체(Class2 객체)를 클라이언트 객체 (Class1 객체) 내에서 선언

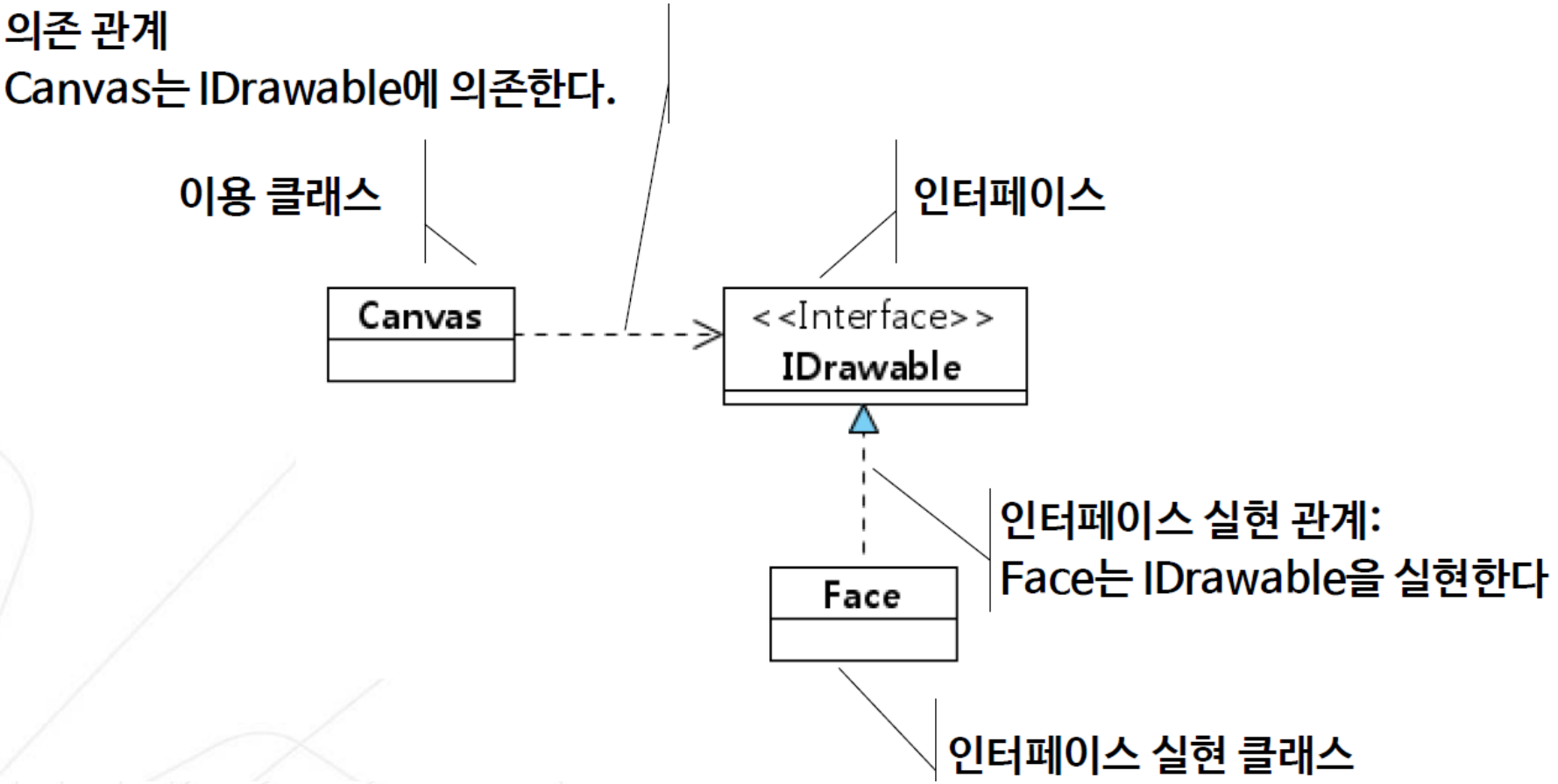



```
class Class1 {
  private Class2 aClass2 ;
  public void op11() {
    aClass2.op2() ;
  }
  public void op12() {
    ...
  }
}
```

연관 관계 vs 의존관계

	연관 관계	의존 관계
역할	메시지 전달의 통로	
관계의 발생 형태	상대 클래스의 속성으로 대응	해당 연산의 인자 클래스 해당 연산 내부 객체의 클래스 해당 연산에서 접근하는 전역 객체의 클래스
관계의 지속 범위	해당 객체의 생명주기	해당 연산 내부
방향성	양방향 가능	단방향

- 의존관계는 인터페이스 와 인터페이스이용자 간에도 사용된다

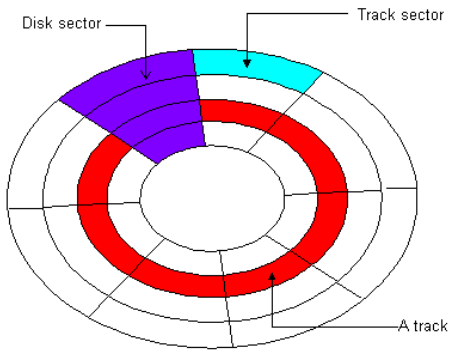


집합(aggregation)

- 집합 관계
  - ➔ 전체 개념(whole)과 부분 개념(part) 사이의 관계
  - 어떤 클래스가 다른 클래스의 모임으로 구성
  - 격납(containment)의 의미
  - 예> 디스크 ⊃ 트랙 ⊃ 섹터

```
class Disk {
    private:
        Track *tracks;
        disk information
        ...
};
class Track {
    private:
        Sector sectors[MAX];
        ...
};
class Sector {
    private:
        ...
};
```

집합 관계는
shared aggregation(줄여서 aggregation)과
composite aggregation(줄여서 composition)로 세분화 된다.



Composition은 전체와 부분이 같은 생명 주기를 가짐

```
public class Engine {
  ......
}

public class Car {
   Engine e = new Engine();
   ........
}
```

Aggregation은 전체와 부분이 동일한 생명 주기를 갖지는 않음.

```
class Person {
    public void learn(){
        Clock clock = new Clock();
    }
}
```

상속(inheritance)

- 상속의 의미
  - 한 클래스가 다른 클래스의 일반화(generalization)된 개념인 경우 성립
  - 슈퍼클래스(superclass), 서브 클래스(subclass)
  - 예> 직원 : 슈퍼클래스 , 관리자 : 서브클래스

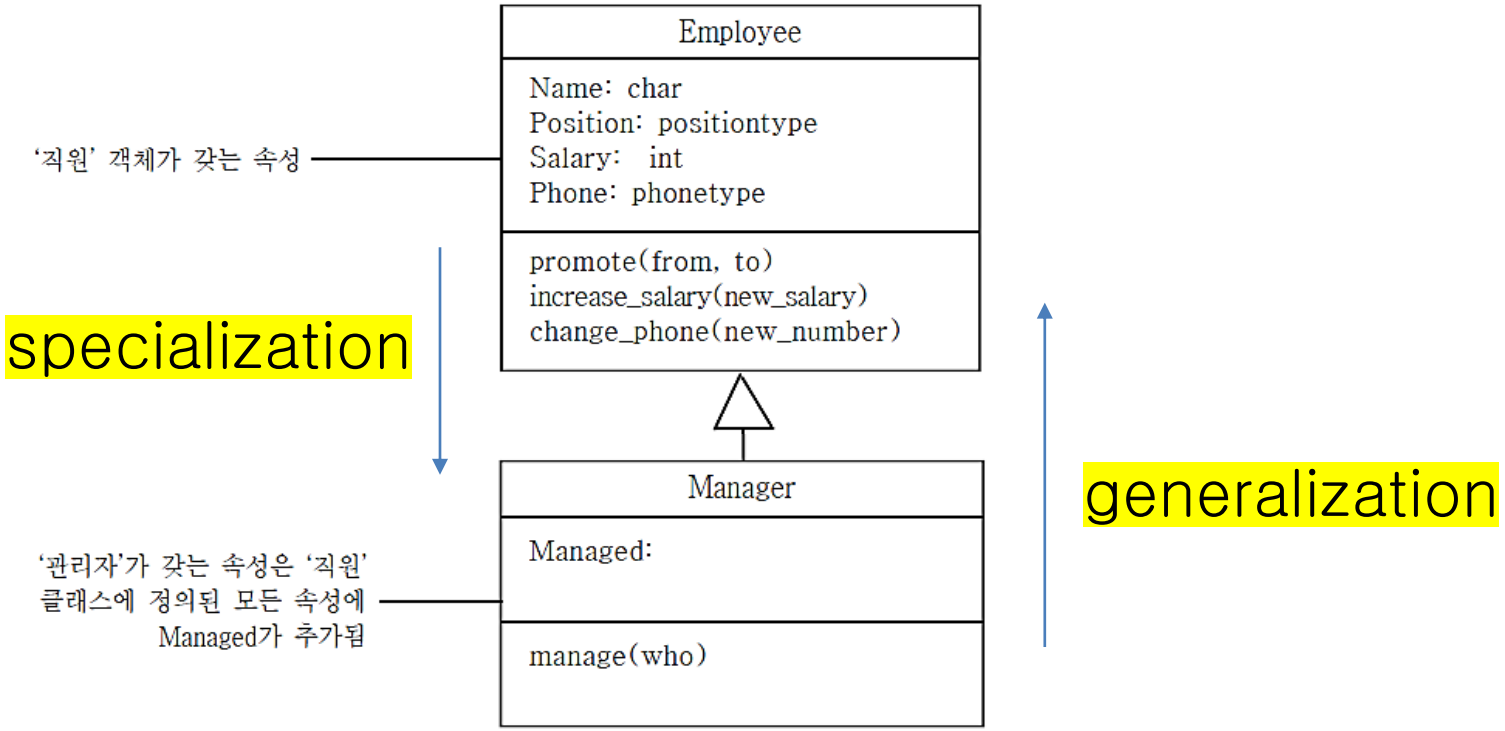
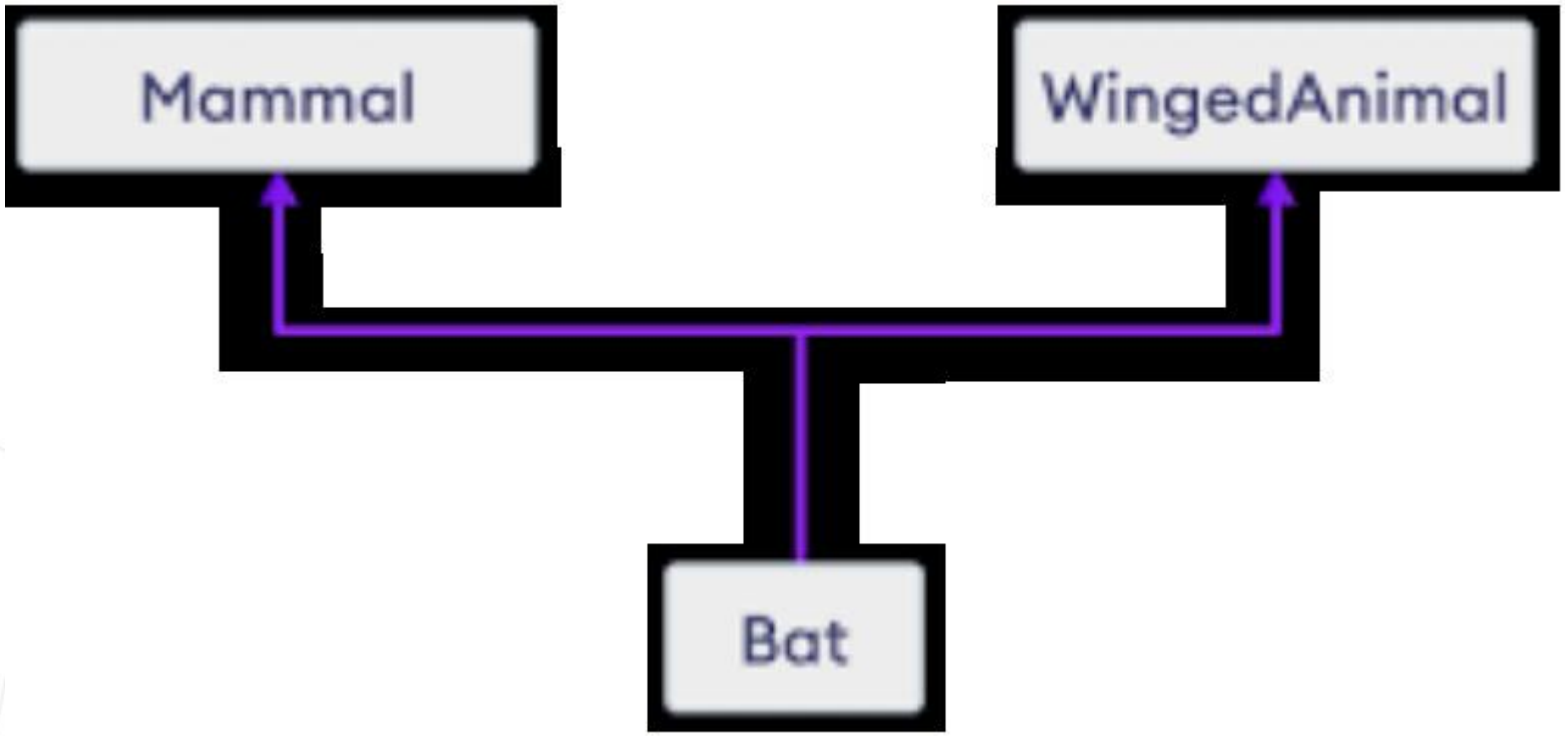


그림 5.5 ▶ 상속의 예

상속(inheritance)

- 복수상속 (multiple inheritance)
  - 두 개 이상의 수퍼 클래스에서 상속 받음



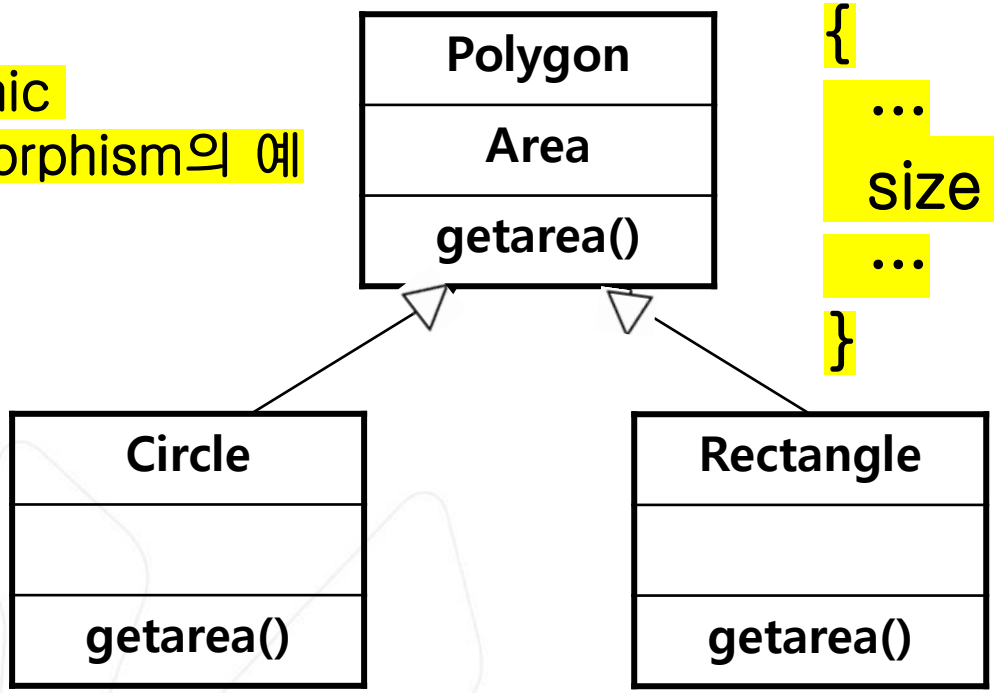
다형성(polymorphism)

- 다형성의 정의
  - 여러 형태를 가지고 있다 (=여러 형태를 받아들일 수 있다)
- 같은 이름의 메시지를 다른 클래스 객체들에서 호출
  - 예> getarea() 를 도형의 모양이 달라도 호출
- 메소드 : 특정한 클래스를 위하여 오퍼레이션을 구현
- 하나 이상의 메소드를 가진 오퍼레이션
  - 매개변수/객체가 속한 클래스의 이름으로 구분

다형성(polymorphism)

- 현재 코드를 변경하지 않고 새로운 클래스를 쉽게 추가할 수 있음
  - 다형성의 예

Dynamic Polymorphism의 예




```
A::op1(Polygon twoDShape)
{
  …
  size = twoDShape.getarea();
  …
}
```

- getArea()의 호출
  - twoDShape.getArea();

```
class A {
    go(int dist) {..}
}

class B extends A {
    go(int dist, int orient) {..}
}

B  bb = new B();
bb.go(20, 0);
bb.go(20);
```