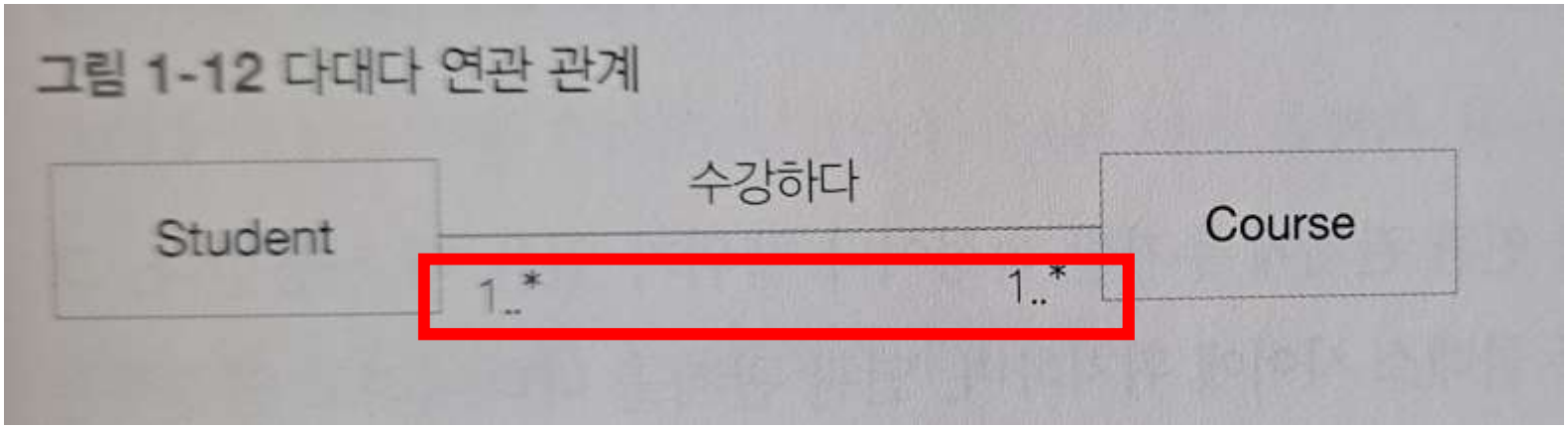
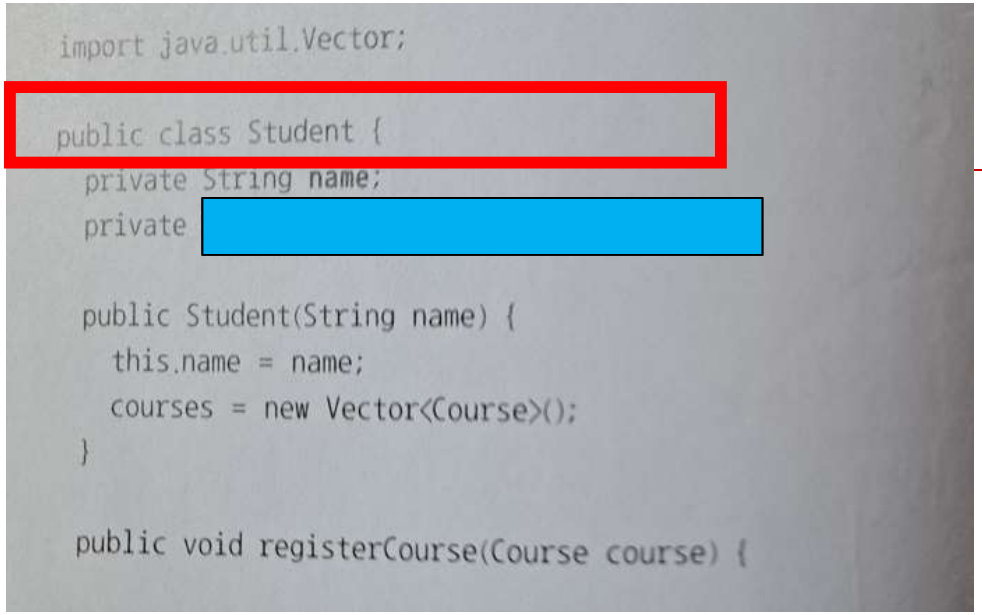
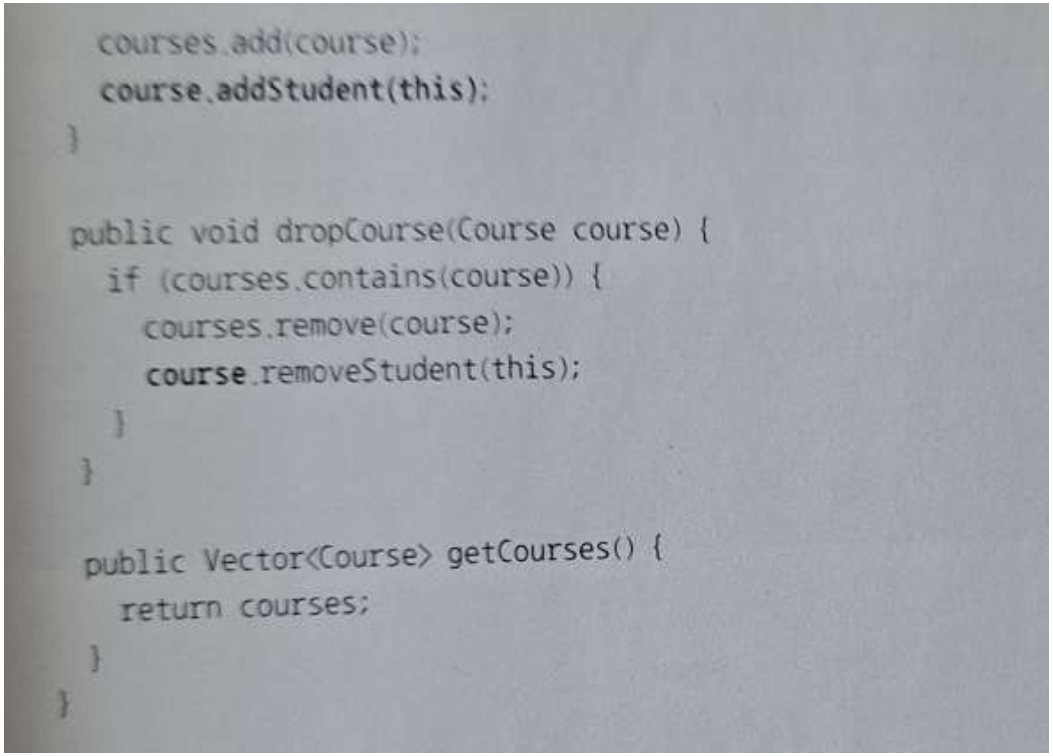
❖ 아래 그림 1-12를 코드(Student, Course Class)로 작성하라

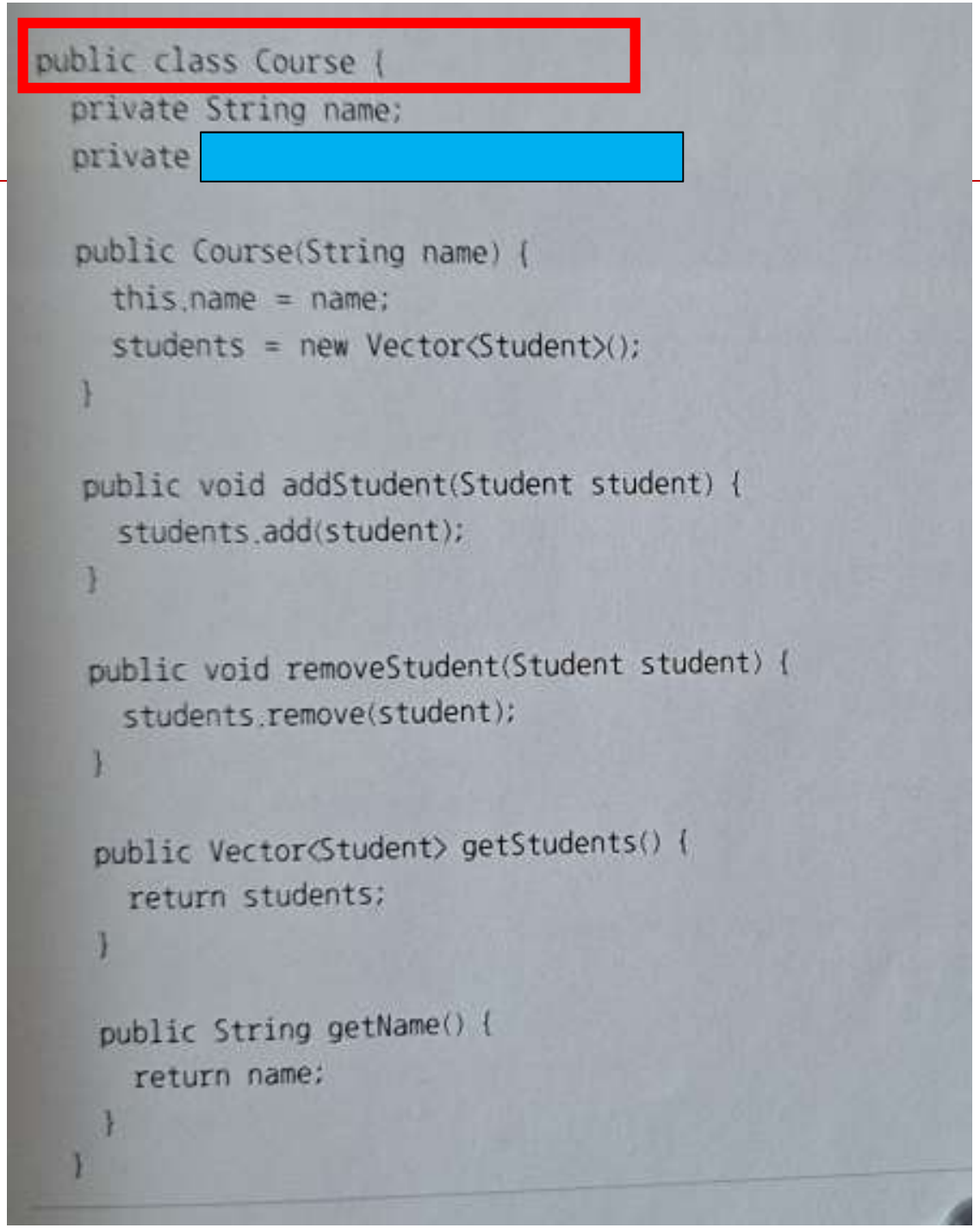




```
import java.util.Vector;

public class Student {
  private String name;
  private [redacted]

  public Student(String name) {
    this.name = name;
    courses = new Vector<Course>();
  }

  public void registerCourse(Course course) {
```



```
    courses.add(course);
    course.addStudent(this);
  }

  public void dropCourse(Course course) {
    if (courses.contains(course)) {
      courses.remove(course);
      course.removeStudent(this);
    }
  }

  public Vector<Course> getCourses() {
    return courses;
  }
}
```

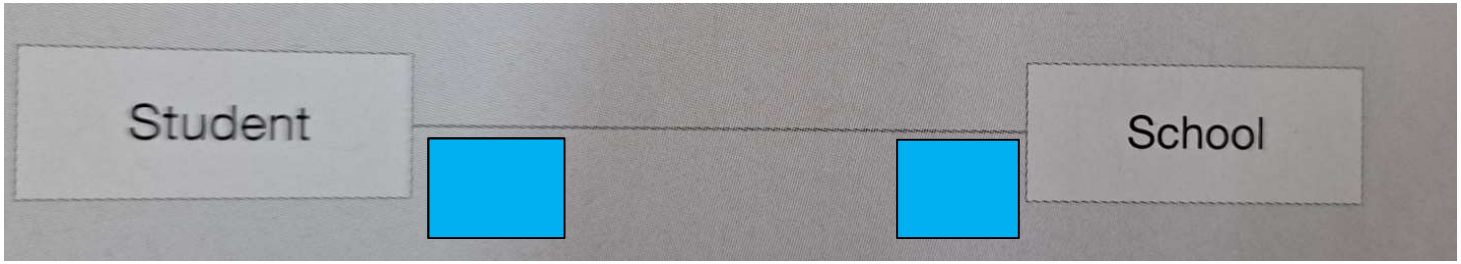


```
public class Course {
  private String name;
  private

  public Course(String name) {
    this.name = name;
    students = new Vector<Student>();
  }

  public void addStudent(Student student) {
    students.add(student);
  }

  public void removeStudent(Student student) {
    students.remove(student);
  }

  public Vector<Student> getStudents() {
    return students;
  }

  public String getName() {
    return name;
  }
}
```

체크포인트

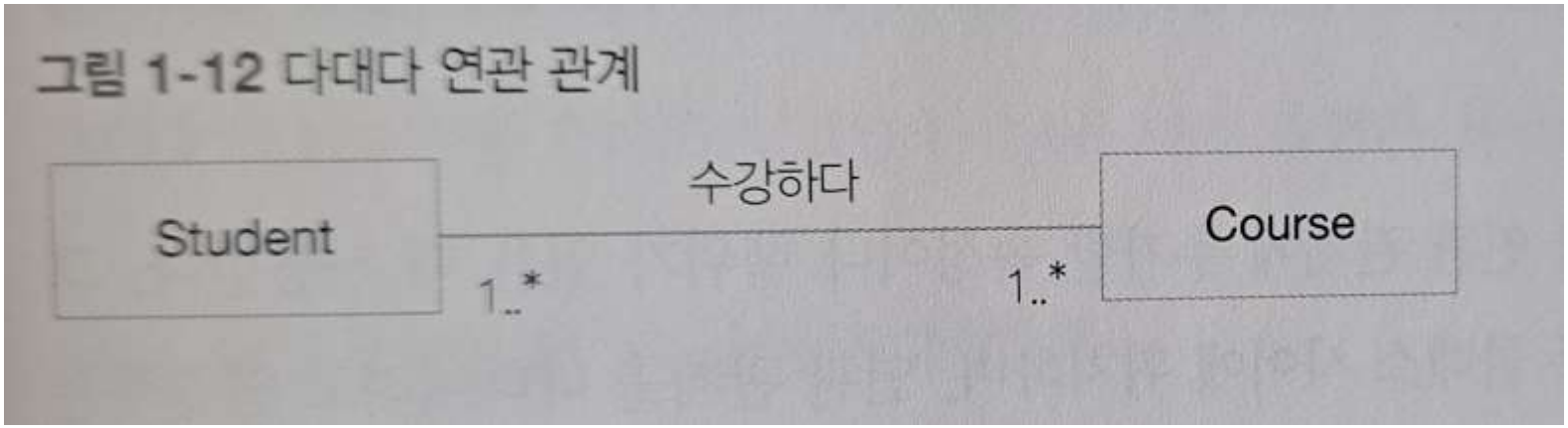
체크포인트_ 다음 설명에 맞는 클래스 다이어그램을 작성하라.

- 학생은 반드시 한 학교에 소속되어야 한다.
- 학교는 학생이 반드시 100명 이상 있어야 한다.



연관 클래스(association Class)

❖ 아래 1-12에서 성적 정보를 어디에 두어야 할까?



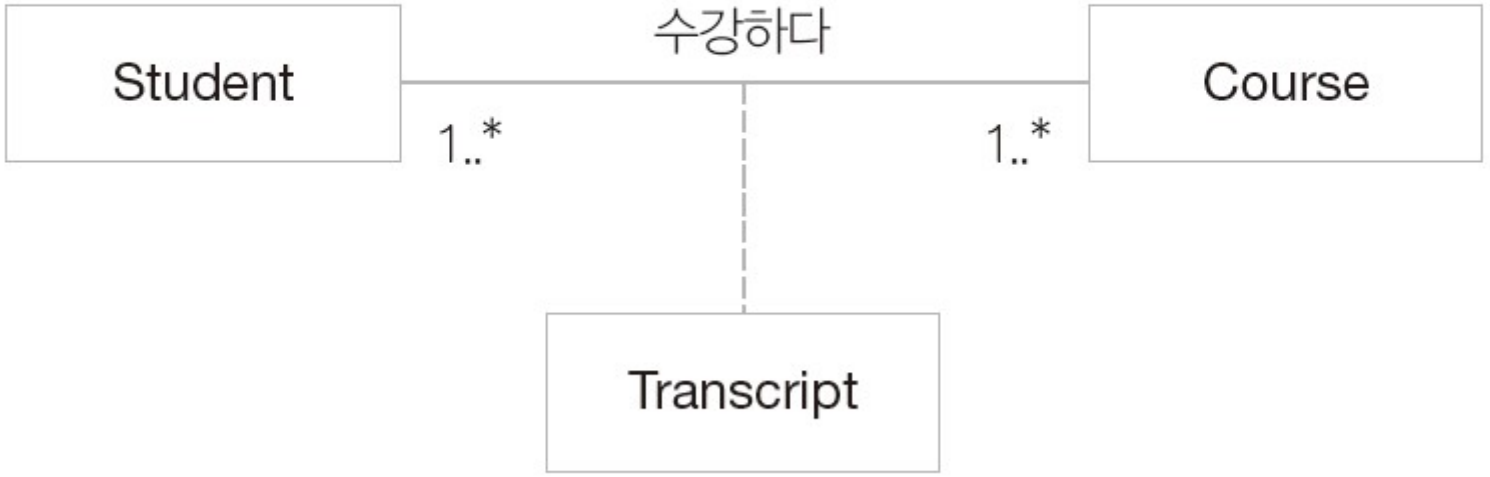
- Student Class에 두는것 X
- Course Class에 두는 것 X

- 홍길동은 2013년에 개설한 소프트웨어 공학에서 A+를 받았다.
- 홍길서는 2013년에 개설한 소프트웨어 공학에서 C를 받았다.
- 홍길동은 2013년에 개설한 디자인 패턴에서 A를 받았다.
- 홍길서는 2013년에 개설한 디자인 패턴에서 B를 받았다.
- 홍길남은 2013년에 개설한 데이터베이스에서 B+를 받았다.
- 홍길동은 2012년에 개설한 디자인 패턴에서 D를 받았다.

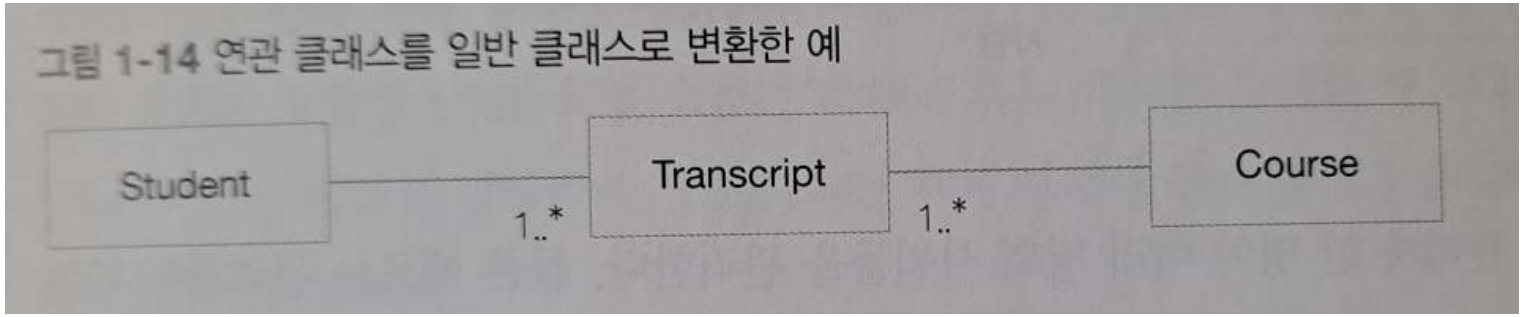
연관 클래스(association Class)

❖ 연관 관계에 추가할 속성이나 행위가 있을 때 사용

그림 1-13 연관 클래스



❖ 아래 그림 1-14를 코드로 작성하라 ==> page 48



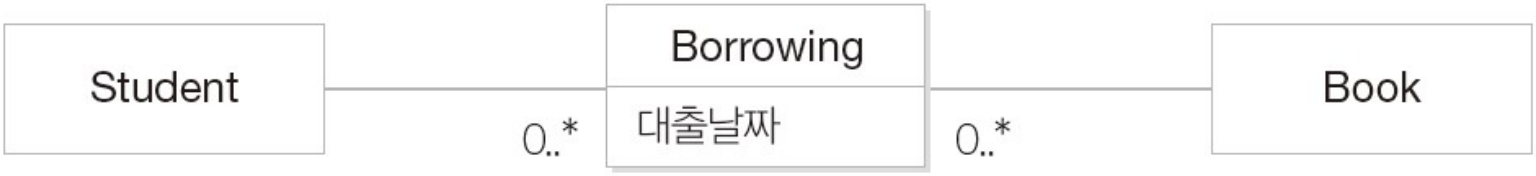
❖ 문제점

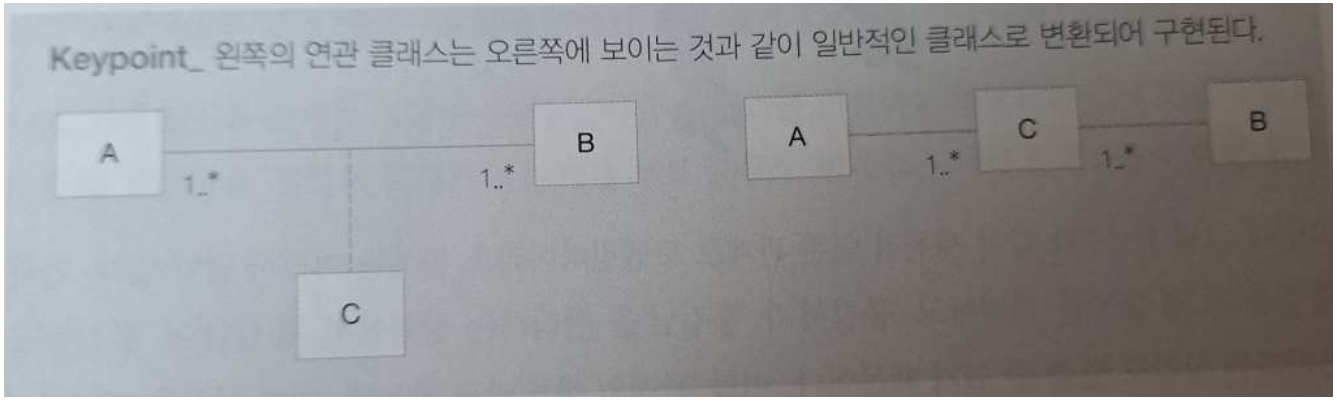
- 같은 학생이 같은 과목을 여러 번 수강할수도 있음
- 학생과 과목에 여러 개의 성적 정보가 있을수 있음
- ➔ history 개념 포함 (예. 날짜)

연관 클래스

- 사건 이력(event history) 표현
- 대출 이력

그림 1-15 이력의 표현





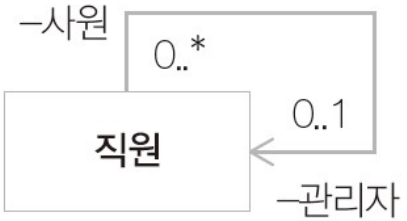
재귀적(reflexive) 연관 관계

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

그림 1-16 직원 역할을 클래스로 모델링

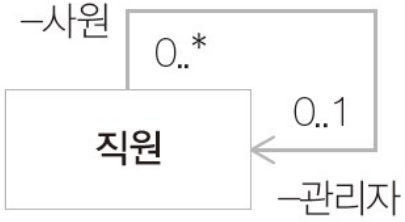
한 사람이 관리자일수도 있고
사원일수도 있는 문제점

그림 1-17 재귀적 연관 관계



제약 설정

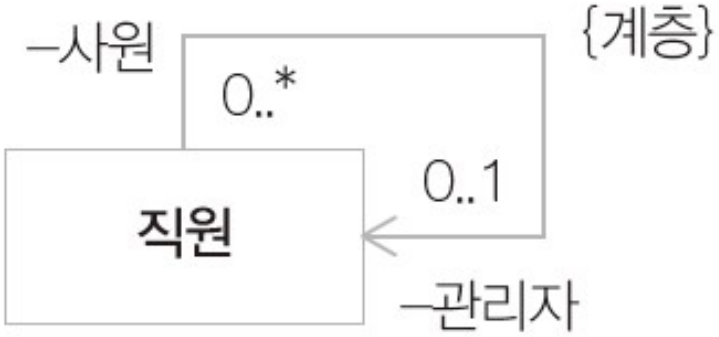
그림 1-17 재귀적 연관 관계



❖ 그림 1-17의 문제점

- 홍길동이 홍길서를 관리하고
- 홍길서가 홍길남을 관리하고
- 홍길남이 다시 홍길동을 관리하는 상황….
- ➔ 관계의 루프
- → 관계의 루프를 배재하는 그림 (그림 1-18)

그림 1-18 재귀적 연관 관계에서 제약 설정



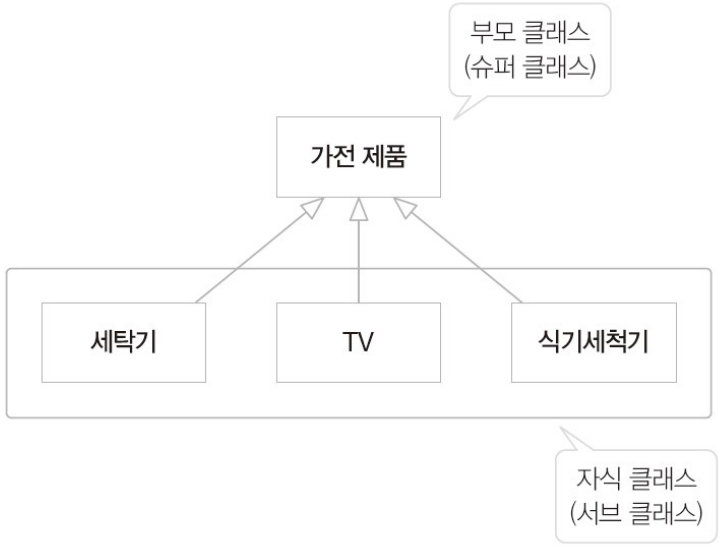
❖ {계층}

- 제약 설정의 예
- 객체 사이에 상하관계가 존재하고 사이클이 존재하지 않는다는 의미

일반화 [p.33]

- 일반화는 상속관계
  - 한 클래스가 다른 클래서를 포함하는 상위 관계일때
- 일반화는 “is a kind of” 관계
  - 세탁기 is a kind of 가전 제품
  - TV is a kind of 가전 제품
  - 식기세척기 is a kind of 가전 제품

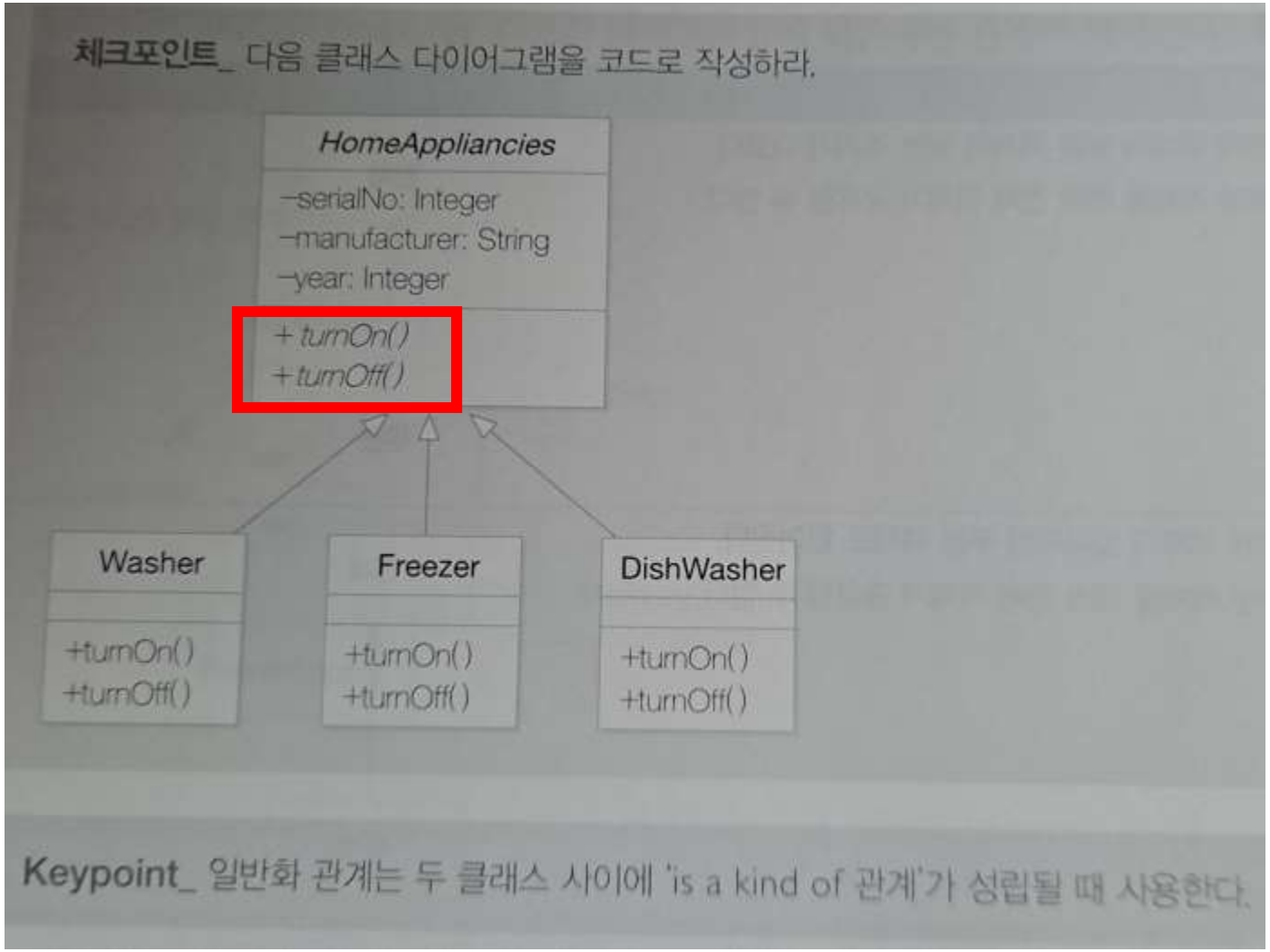
그림 1-19 일반화 관계

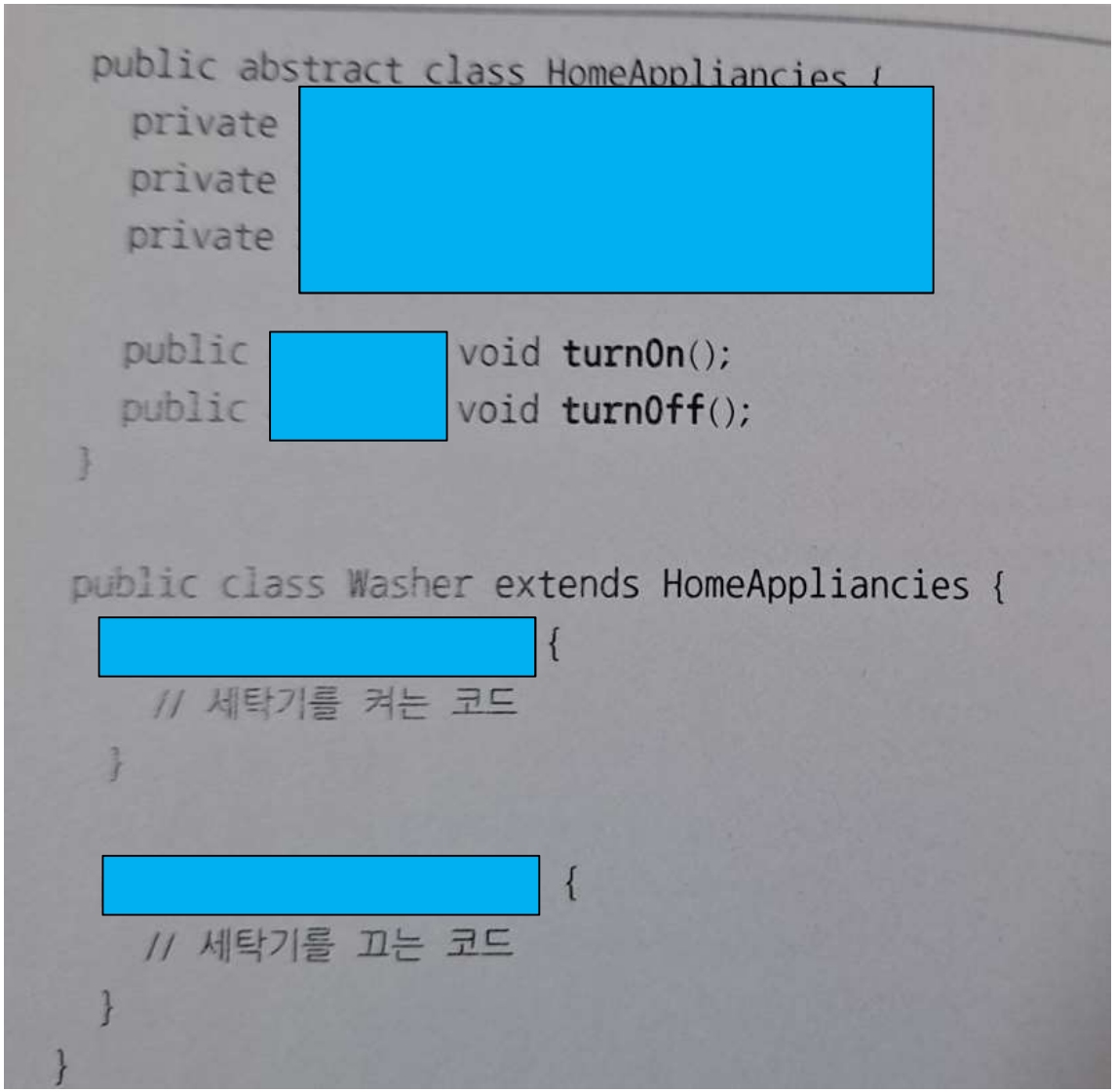


❖ abstract class / abstract method

- turnOn, turnOff기능은 세탁기, TV, 식기 세척기마다 해당 기능은 모두 있지만, 작동하는 방법은 다르다(다르게 구현된다)
- 상위 클래스에서 해당 기능은 abstract method로 정의
- 상위 클래서는 abstract class 가 됨

p.35

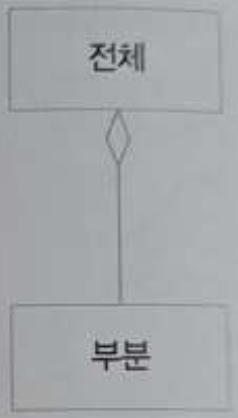
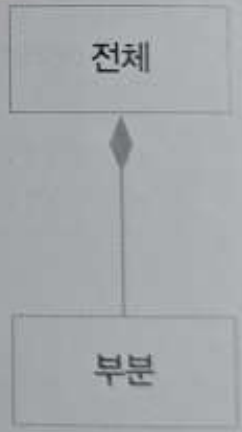

체크포인트_ 다음 클래스 다이어그램을 코드로 작성하라.
HomeAppliancies
-serialNo: Integer
-manufacturer: String
-year: Integer
+turnOn()
+turnOff()
Washer
+turnOn()
+turnOff()
Freezer
+turnOn()
+turnOff()
DishWasher
+turnOn()
+turnOff()
Keypoint_ 일반화 관계는 두 클래스 사이에 'is a kind of 관계'가 성립될 때 사용한다.


public abstract class HomeAppliancies {
private
private
private
public
void turnOn();
public
void turnOff();
}
public class Washer extends HomeAppliancies {
{
// 세탁기를 켜는 코드
}
{
// 세탁기를 끄는 코드
}
}

집합 관계 (p. 35)

- 전체와 부분간의 관계
- 집약(aggregation):
  - 전체를 나타내는 객체와 부분을 나타내는 개체의 라이프 타임이 독립적
  - 부분을나타내는 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

표 1-6 집합 관계

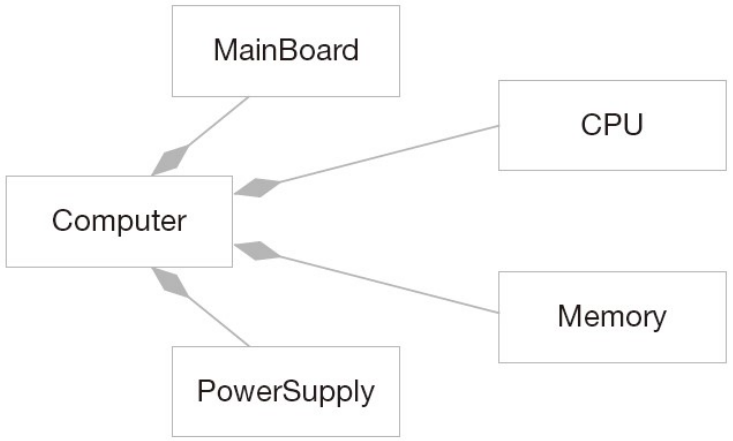
집합 관계	특징	클래스 다이어그램
집약	전체 객체와 부분 객체의 생명 주기가 다르다. 부분 객체를 여러 전체 객체가 공유할 수 있다.	
합성	전체 객체가 없어지면 부분 객체도 없어진다. 부분 객체를 여러 전체 객체가 공유할 수 없다.	

❖ Composition 관계의 사례

코드 1-2

```
public class Computer {
  private MainBoard mb;
  private CPU c;
  private Memory m;
  private PowerSupply ps;
  public Computer() {
    this.mb = new MainBoard();
    this.c = new CPU();
    this.m = new Memory();
    this.ps = new PowerSupply();
  }
}
```

그림 1-20 합성 관계



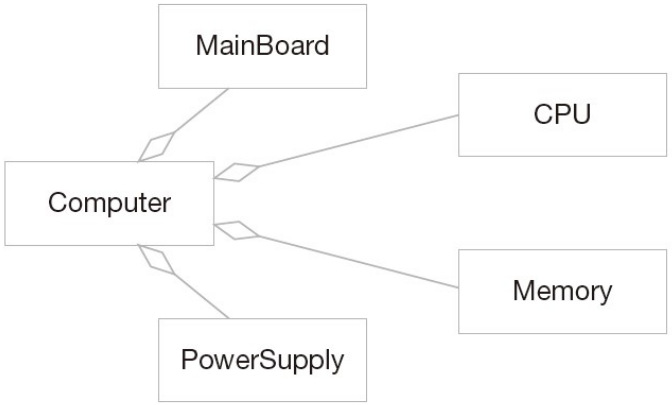
집약/합성 관계

❖ Aggregation 관계

코드 1-3

```
public class Computer {
  private MainBoard mb;
  private CPU c;
  private Memory m;
  private PowerSupply ps;

  public Computer(MainBoard mb, CPU c, Memory m, PowerSupply ps) {
    this.mb = mb;
    this.c = c;
    this.m = m;
    this.ps = ps;
  }
}
```

그림 1-21 집약 관계



체크포인트 (p. 38) ==> p. 53

체크포인트_ 동아리와 학생의 관계에서 다음 사실을 모두 클래스 다이어그램으로 표현하라.

- 학생은 한 동아리에만 가입할 수 있다.
- 한 동아리에는 여러 명의 학생들이 있다.
- 동아리가 없어지면 동아리에서 활동했던 학생들의 정보도 없어진다.

의존 관계 (p. 38)

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

- 그림 1-22.
  - 출근할때 차를 사용하는 Person
  - 출근할때마다 같은 차를 사용함을 가정

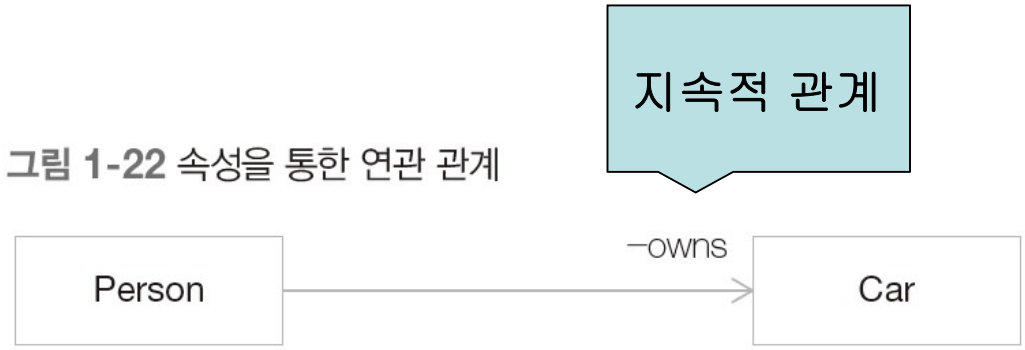


그림 1-22 속성을 통한 연관 관계

❖ 그림 1-22를 코드로 작성하라

```
public class Person {
  private Car owns; // 이 속성으로 연관 관계가 설정된다

  public void setCar(Car car) {
    this.owns = car;
  }

  public Car getCar() {
    return this.owns;
  }
}

public class Car {
  // private Person person; // 단방향 연관 관계이므로 필요 없다
  ...
}
```

❖ 연산의 인자나 메소드의 지역 개체로 참조

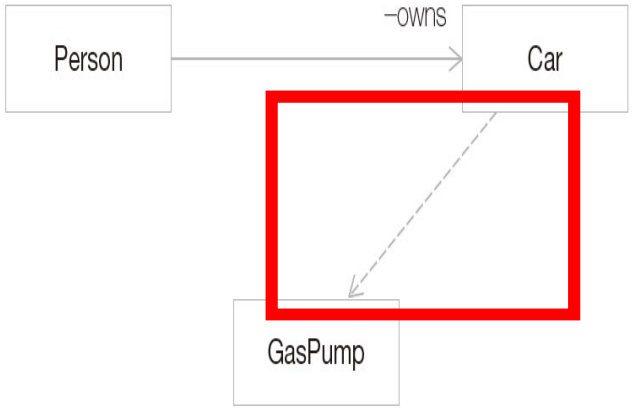
- 찰나적 관계
- 자동차와 주유기 관계
- 주유 서비스를 받을 때마다 사용하는 주유기가 매번 달라짐
- 인자나 지역객체로 사용

❖ 의존관계 : 점선으로 표시

그림 1-23 의존 관계

```
public class Car {
    ...

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

그림 1-24 의존 관계와 연관 관계



인터페이스와 실체화 관계 (p. 40)

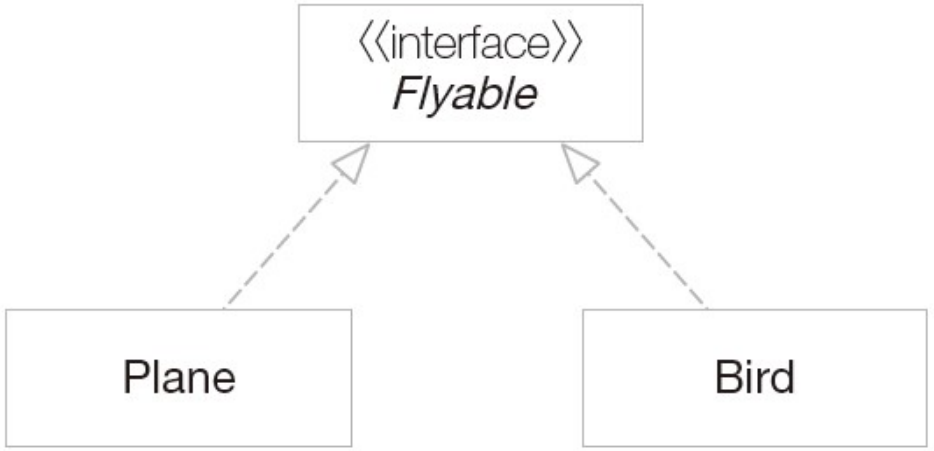
- 인터페이스란 책임(responsibility) 이다.
  - 리모콘의 책임은 가전 기기를 켜거나 끄거나 볼륨을 높이거나 낮춘다
  - 책임 : 꼭 갖고 있어야할 기능

표 1-7 인터페이스와 책임

연산	의미
turn_on()	켜다
turn_off()	끄다

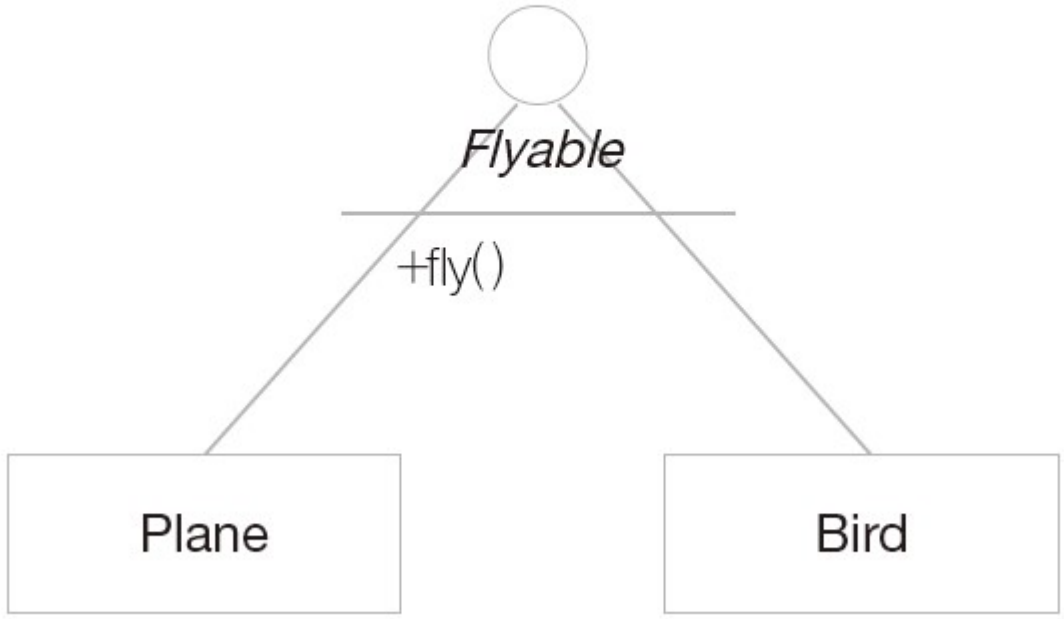
- **인터페이스 : 어떤 공통되는 능력이 있는 것들을 대표하는 관점**
- **표현**
  - **점선 및 닫힌 삼각형 : implementation 관계**
  - **스테레오타입 ‹‹ ››**
  - **키워드**
  - **이탤릭체**

그림 1-25 인터페이스와 실체화 관계



실체화 관계

그림 1-26 실체화 관계의 또다른 표현



Keypoint

- 일반화 관계 : is a kind of 관계
- 실체화 관계 : can do this의 관계