클래스 다이어그램

Contents

주요 내용

- 01 클래스 다이어그램의 구성 요소와 표현
- 02 클래스 사이의 관계
- 03 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현
- 04 클래스 다이어그램 모델링 연습

학습목표

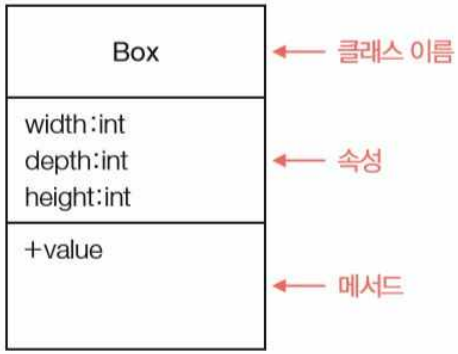
- 클래스의 개념과 구성 요소를 이해한다.
- 클래스 추출과 관계를 학습한다.
- 다양한 예제를 통해 클래스 다이어그램을 모델링하는 연습을 해본다.

1. 클래스 다이어그램의 구성요소와 표현

■ 클래스의 구성 요소

▪ 클래스는 클래스 이름, 속성, 메서드로 구성

- 클래스class
  - 공통의 속성, 메서드(오퍼레이션), 관계, 의미를 공유하는 객체 집합에 대한 기술
- 속성attribute
  - 클래스의 구조적 특성에 이름을 붙인 것
  - 구조적 특성에 해당하는 인스턴스가 보유할 수 있는 값의 범위를 기술
  - 영문 소문자로 시작함




```
// 클래스
Box {
    // 속성
    private int width;
    private int depth;
    private int height;

    // 메서드
    void value() {
    }
}
```

1. 클래스 다이어그램의 구성요소와 표현

■ 클래스의 구성 요소

- 메서드method
  - 오퍼레이션이라고도 함
  - 이름, 타입, 매개변수들과 연관된 행위를 호출할 때 제약사항이 요구되는데, 이 제약사항을 명세하는 클래스의 행위적 특징

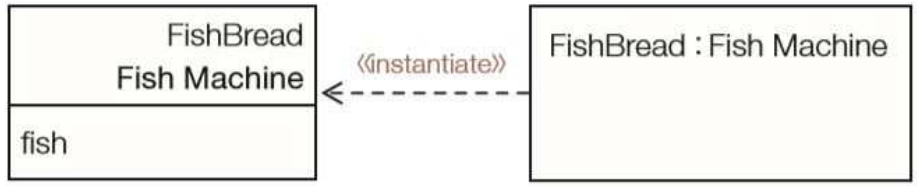
종류	부호	설명
public	+	자신의 속성이나 동작을 외부에 공개하는 접근 제어
private	−	상속된 파생 클래스만 액세스할 수 있는 접근 제어
protected	#	구조체의 멤버 함수만 접근할 수 있으며 외부에서 엑세스할 수 없는 접근 제어

1. 클래스 다이어그램의 구성요소와 표현

■ 객체와 클래스 사이의 관계와 표현

▪ 클래스

- 객체를 생성할 수 있는 구조와 정보를 갖는 구조
- 객체는 클래스의 인스턴스



1. 클래스 다이어그램의 구성요소와 표현

■ 객체와 클래스 사이의 관계와 표현

- Account 클래스와 Application 클래스로 구성된 객체 생성 다이어그램
- Application 클래스는 Account 클래스와 객체를 생성하여 실행하는 메인 메서드 포함

1. 클래스 다이어그램의 구성요소와 표현

■ 객체와 클래스 사이의 관계와 표현

```
class Account {
   int balance;
   int deposit(int amount) {
       balance = balance + amount;
       return balance;
   }
}

class Application {
   Account account1;
   public static void main(String args[]) {

       // 객체 생성
       account1 = new Account();
       account1.balance = 5000;
       account1.deposit(5000);
       System.out.println("Balance = " +account1.balance);
   }
}
```

1. 클래스 다이어그램의 구성요소와 표현

■ 객체와 클래스 사이의 관계와 표현

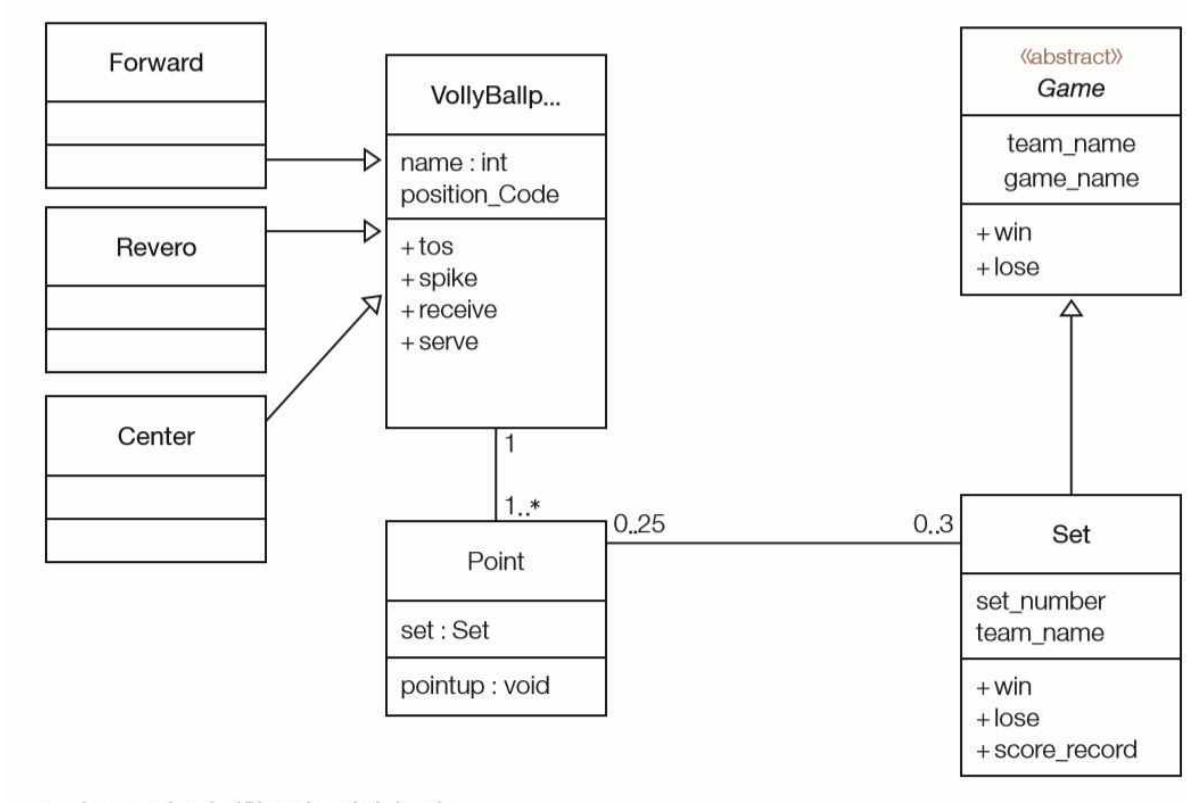
- 연관 관계는 의존 관계와 마찬가지로 객체를 생성하는 관계
- 그러나 연관 관계는 멤버 변수로 참조하고, 의존 관계는 메서드로 참조한다는 것이 다름

1. 클래스 다이어그램의 구성요소와 표현

■ 클래스 추출

▪ 명사는 클래스나 속성, 동사는 메서드

- 명세서를 통한 클래스 추출
  - 배구 선수, 점수, 세트, 경기, 리베로, 센터, 공격수
- 명세서를 통한 메서드 추출
  - 토스, 스파이크, 리시브, 이긴다, 진다, 올라간다



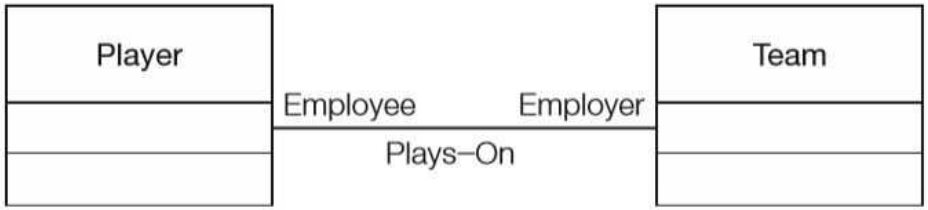
2. 클래스 사이의 관계

연관 관계

- 연관 관계는 클래스 간에 서로 개념적으로 연결된 선으로 표현

팀과 선수의 연관 관계

- 각 클래스에는 각각의 역할 role이 존재
- 선수와 팀의 역할은 각각 피고용인 Employee과 고용인 Employer

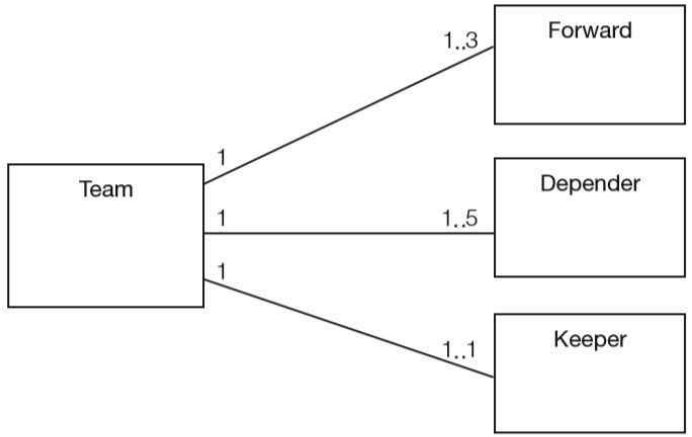


역할에 따른 관계 추가

2. 클래스 사이의 관계

■ 연관 관계

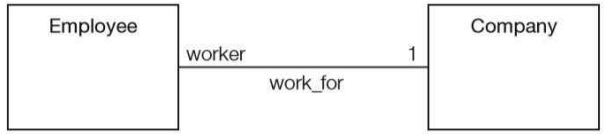
- 하나의 클래스는 여러 개의 클래스와 연관 관계를 가질 수 있음



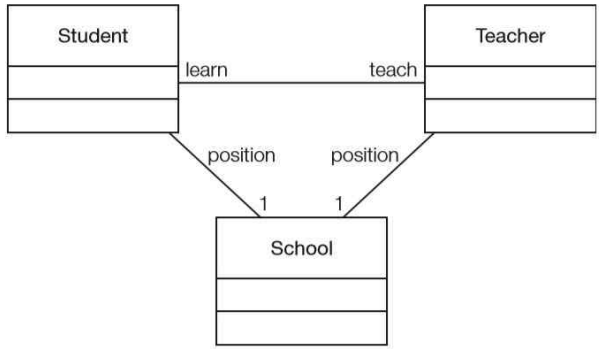
하나의 클래스와 여러 클래스의 연관 관계

- 그림 (a)
  - 사원과 회사가 서로 연관 관계로 연결될 수 있음
- 그림 (b)
  - 학생, 교사, 학교가 각각 다른 두 개체와 연관 관계에 놓임

(a) 사원과 회사



(b) 학생, 교사, 학교



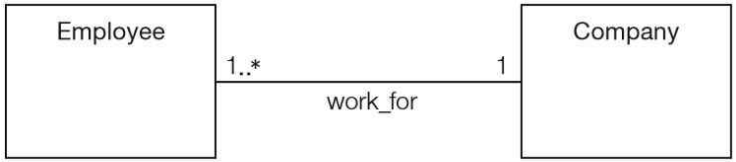
연관 관계의 예

2. 클래스 사이의 관계

■ 연관 관계의 다중성

- 다중성multiplicity
  - 클래스 사이에 연관 관계를 맺는 객체의 수가 1개 이상인 경우
  - 팀 쪽에 붙은 1은 기본값이므로 표현하지 않아도 무방

선수와 팀의 연관 관계 다중성



회사와 사원의 연관 관계 다중성

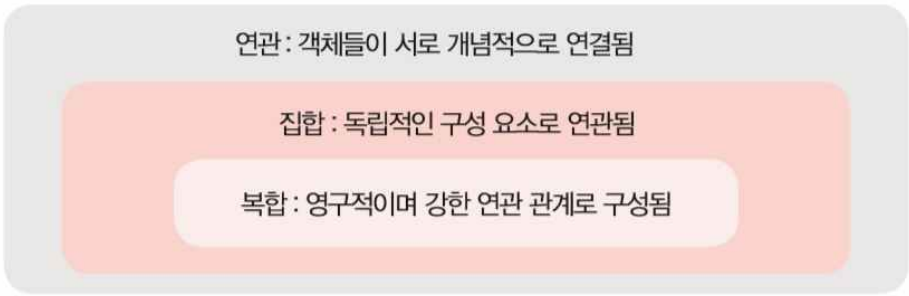
다중성 표현

표현	설명
1	한 객체와 연관된다. 표시하지 않아도 되는 기본값이다.
0..1	0개 또는 1개의 객체와 연관된다.
0..*	0개 또는 많은 수의 객체와 연관된다.
*	0..*와 동일하다.
1..*	1개 이상의 객체와 연관된다.
1..12	1개에서 12개까지의 객체와 연관된다.
1..2, 4, 8	1개에서 2개까지, 또는 4개 또는 8개의 객체와 연관된다.

2. 클래스 사이의 관계

■ 집합 관계의 복합 관계

- 집합 관계와 복합 관계는 연관관계에 포함



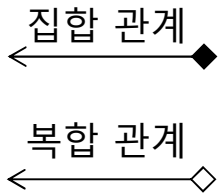
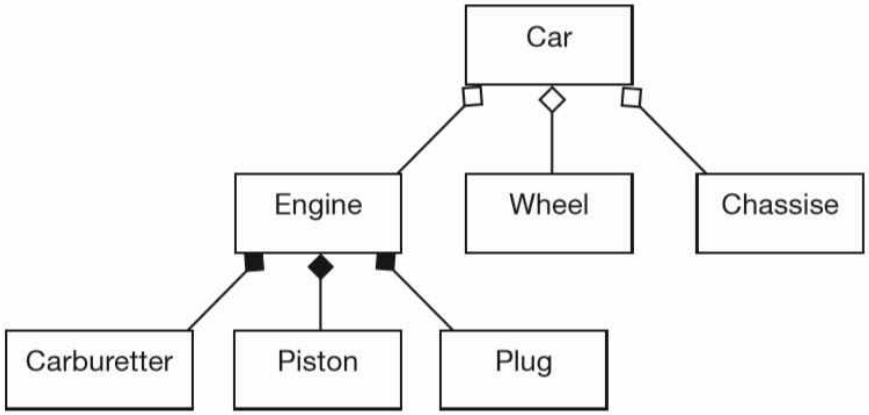


그림 4-12 연관 · 집합 · 복합 관계의 개념과 포함 관계

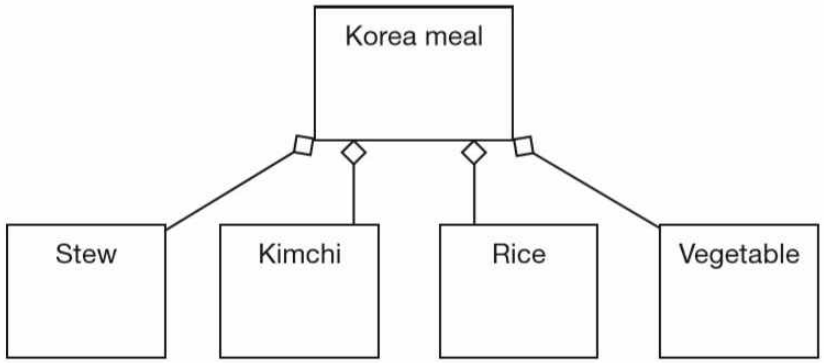
- 집합 관계는 하나의 객체가 독립적인 객체 여러 개로 구성되는 경우
- 복합 관계는 집합 관계보다 좀더 더 강한 관계로 구성되는 경우
- 복합 관계는 단독 사용이 불가능하며 반드시 슈퍼 클래스와 함께 사용



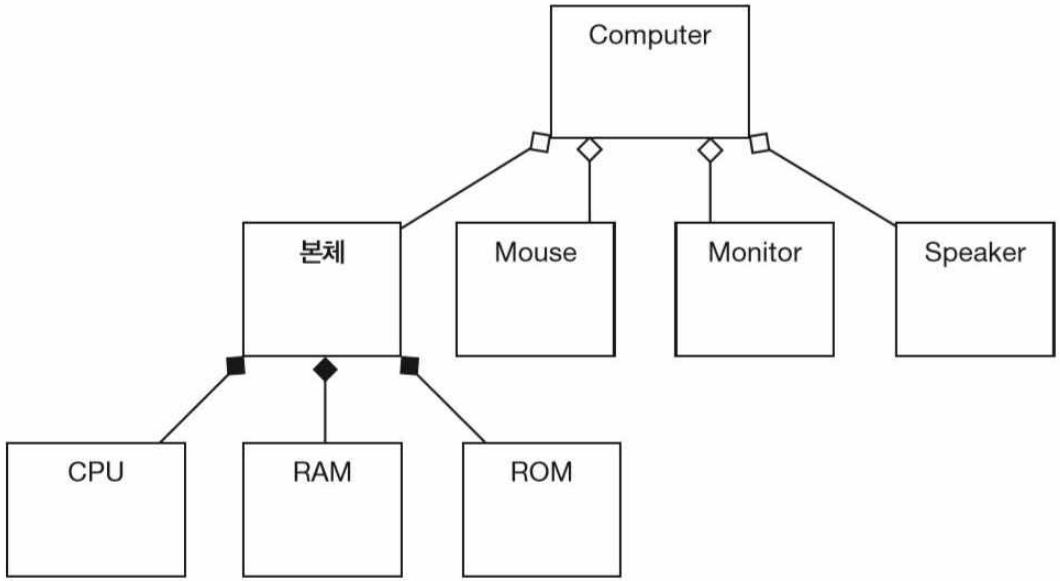
집합 관계(차와 엔진, 바퀴, 차체)와 복합 관계(엔진과 카뷰레터, 피스톤, 플러그)의 예 1

2. 클래스 사이의 관계

- **집합 관계의 복합 관계**



(a) 식사-밥, 찌개, 김치, 나물 : 집합 관계



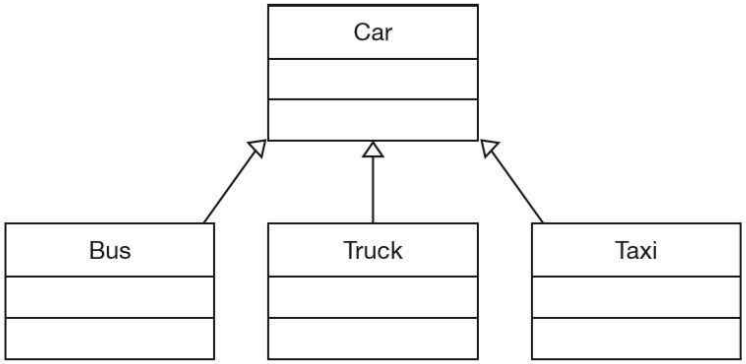
(b) 컴퓨터-모니터, 마우스, 키보드, 스피커 : 집합 관계 / 본체-CPU, ROM, RAM : 복합 관계

집합 관계와 복합 관계의 예 2

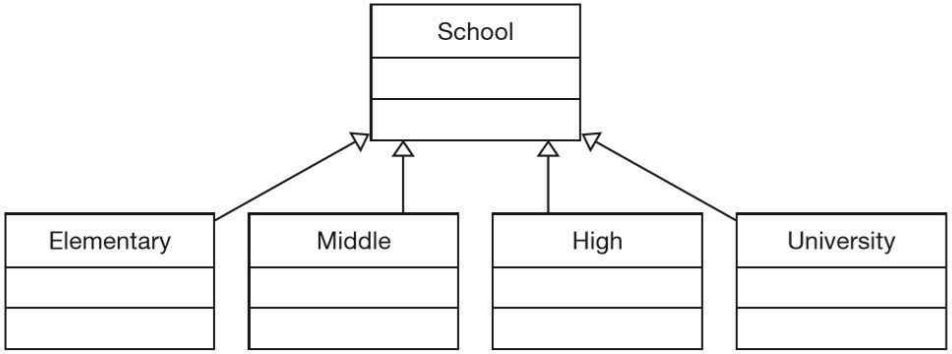
2. 클래스 사이의 관계

■ 일반화 관계

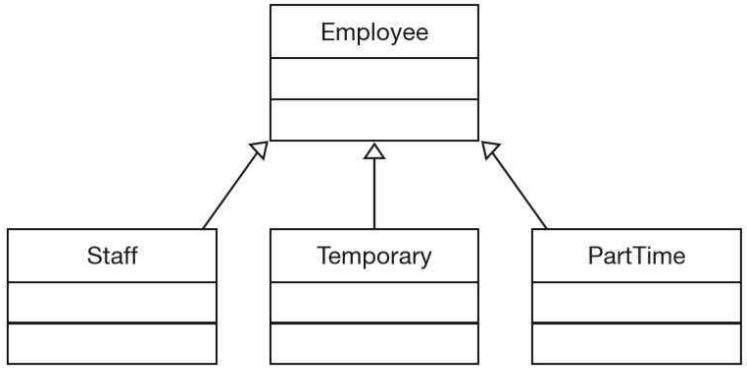
- 한 종류로 묶을 수 있는 관계 즉, a_kind_of의 관계를 의미



(a) 차와 버스, 트럭, 택시

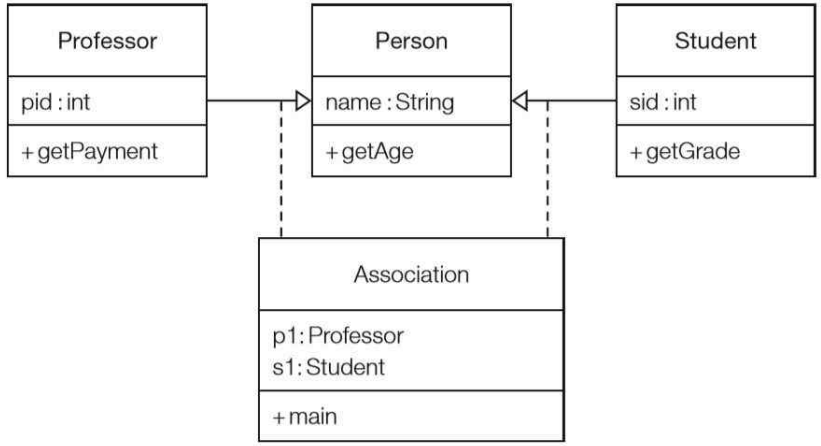


(b) 학교와 초등학교, 중학교, 고등학교, 대학교



(c) 사원과 정사원, 계약사원, 아르바이트생

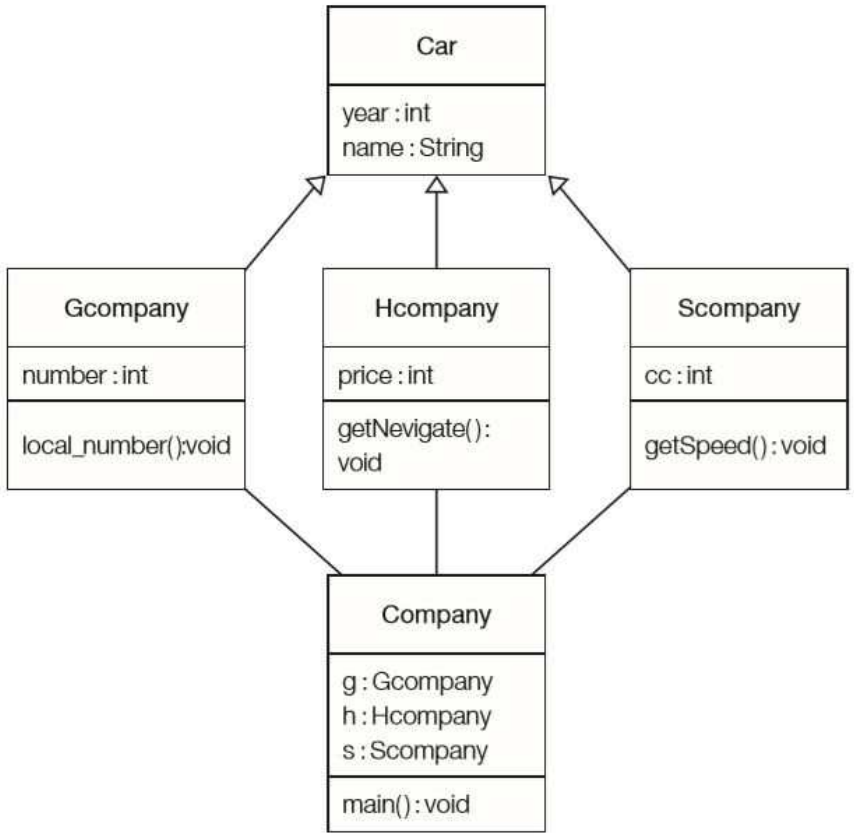
일반화 관계의 예



사람과 교수, 학생 간의 일반화 관계와 연관 관계의 예

2. 클래스 사이의 관계

■ 일반화 관계



자동차와 자동차 제품 간 일반화 관계

```java
class Person {
  String name;
  int getAge(int a) {
    if (a > 10000) {
      return 20+2020-(a/100);
    }
    else
      return 2020-a;
  }
}

class Student extends Person {
  int sid;
  int getGrade() {
    return sid-202000;
  }
}

// Person으로 상속받기 때문에 name, getAge(), pid, getPayment()로 구성
class Professor extends Person {
  int pid;
  int getPayment() {
    return pid*10000;
  }
}

class Association {
  public static void main(String[] args) {
    Student s1 = new Student();
    Professor p1 = new Professor();
    s1.name = "홍길동";
    s1.sid = 202001;
    System.out.println("Student name : " + s1.name + "Student ID : " + s1.sid);
    System.out.println("Student Age : " + s1.getAge(s1.sid) + " Student Grade :"
                       + s1.getGrade());
    p1.name = "한교수";
    p1.pid = 1970;
    System.out.println("Professor name : " + p1.name+"Professor ID : " + p1.pid);
    System.out.println("Professor Age : " + p1.getAge(p1.pid) + " Professor Payment :"
                       + p1.getPayment());
  }
}
```

2. 클래스 사이의 관계

■ 일반화 관계

```
01 class Car {
02   String name;
03   int year;
04   int getYear() {
05     return year;
06   }
07 }
08
09 class Gcompany extends Car {
10   int number;
11   String state;
12   String getLocal_number() {
13     if (number > 1000)
14       state = "true";
15     else
16       state = "false";
17     return state;
18   }
19 }
20
21 class Hcompany extends Car {
22   int price;
23   int getNevigate() {
24     return price-10000000;
25   }
26 }
27
28 class Scompany extends Car {
29   int cc;
30   int getSpeed() {
31     return cc-1300;
32   }
33 }
34
35 public class Company {
36   public static void main(String args[]) {
37     Gcompany g = new Gcompany();
38     Hcompany h = new Hcompany();
39     Scompany s = new Scompany();
40     g.name = "쏘렌토";
41     g.year = 2020;
42     g.number = 1234;
43     System.out.println("Car name: " + g.name + "Car number:" + g.number);
44     System.out.println("Car year: " + g.getYear() + "Car sold: " + g.getLocal_number());
45     h.name = "소나타";
46     h.price = 10000000;
47     System.out.println("Car name: " + h.name + "Car price: " + h.price);
48     System.out.println("Car year: " + h.getYear() + "Car number: " + h.getNevigate());
49     s.name = "르노";
50     s.cc = 2500;
51     System.out.println("Car name: " + s.name + "Car cc: " + s.cc);
52     System.out.println("Car year: " + s.getYear() + "Car speed: " + s.getSpeed());
53   }
54 }
```

2. 클래스 사이의 관계

■ 의존 관계

- 하나의 클래스가 또 다른 클래스를 사용해 영향을 미치는 관계
- 영향을 일으키는 쪽에 화살표

Class A ⟶ Class B

- 의존관계의 조건
  - 한 클래스의 메서드가 다른 클래스의 객체를 인자로 받아 메서드를 사용하는 경우
  - 한 클래스의 메서드 내부에서 다른 클래스의 객체를 생성하여 그 메서드를 사용하는 경우
  - 다른 클래스의 메서드가 또 다른 클래스의 객체를 반환하는 경우

① 텔레비전 → 리모컨
② 수업 → 교수
③ 전화기 → 버튼
④ 세탁기 → 손잡이
⑤ 자동차 → 기어

의존 관계 예

2. 클래스 사이의 관계

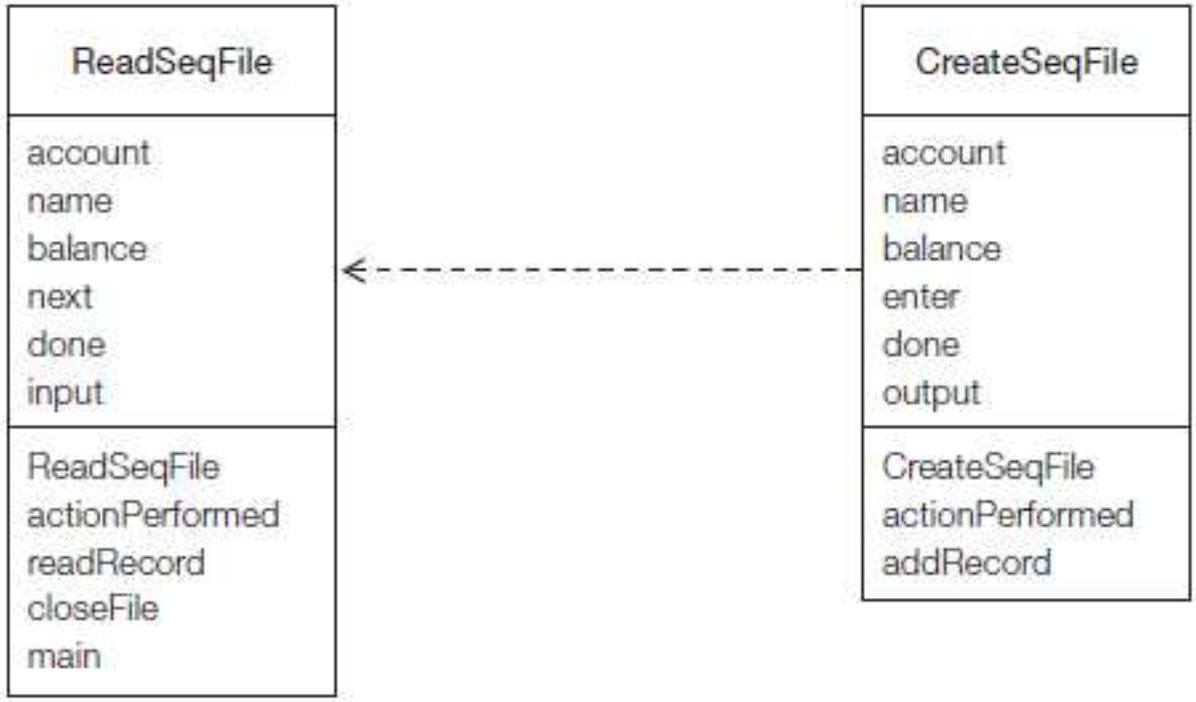
■ 의존 관계

메서드 호출의 의존 관계

```
01 public class MyApplet extends Applet {
02     init();
03     public void start();
04     public void paint(Graphics g) {
05         g.drawString("Hello Applet World", 10, 20);
06     }
07 }
```

2. 클래스 사이의 관계

- **의존 관계**



계좌를 저장하고 읽는 프로그램

2. 클래스 사이의 관계

■ 의존 관계

```
01 import java.io.*;
02 import java.awt.*;
03 import java.awt.event.*;
04
05 public class ReadSeqFile extends Frame implements ActionListener {
06    private TextField account, name, balance;
07    private Button next, done;
08    private DataInputStream input;    // 필터 입력 스트림 객체
09    public ReadSeqFile() {
10       super("고객 파일을 읽음");
11       try {
12          input = new DataInputStream(new FileInputStream("client.doc"));
13       }
14       catch (IOException e) {
15          System.err.println(e.toString());
16          System.exit(1);
17       }
18       setSize(250,130);
19       setLayout(new GridLayout(4, 2));
20       add(new Label("계좌 번호"));
21       account = new TextField();       // 계좌 번호 읽기
22       account.setEditable(false);     // 데이터 입력 금지시키기
23       add(account);
24       add(new Label("이름"));
25       name = new TextField(20);        // 이름 읽기
```

2. 클래스 사이의 관계

■ 의존 관계

```
26      name.setEditable(false);        // 데이터 입력 금지시키기
27      add(name);
28      add(new Label("잔고"));
29      balance = new TextField(20);    // 잔고 읽기
30      balance.setEditable(false);     // 데이터 입력 금지시키기
31      add(balance);
32
33      next = new Button("출력");       // 파일로부터 데이터를 읽는 버튼
34      next.addActionListener(this);
35      add(next);
36      done = new Button("완료");       // 프로그램을 종료하는 버튼
37
38      done.addActionListener(this);
39      add(done);
40      setVisible(true);
41    }
42    public void actionPerformed(ActionEvent e) {
43      if (e.getSource() == next)
44      readRecord();                   // 데이터를 한 레코드씩 읽는 메서드
45      else
46      closeFile();
47    }
48    public void readRecord() {
49      int accountNo;
50      double d;
51      String namedata;
52      try {
53        accountNo = input.readInt();   // 정수형인 계좌 번호 읽기
54        namedata = input.readUTF();    // 문자열인 이름 읽기
55        d = input.readDouble();        // 실수형인 잔고 읽기
56        /* 읽어들인 데이터를 관련된 텍스트 필드에 출력하기 */
57        account.setText(String.valueOf(accountNo));
58        name.setText(namedata);
59        balance.setText(String.valueOf(d));
60      }
61      catch(EOFException eof){closeFile();
62      }
63      catch (IOException io) {
64        System.err.println(io.toString());
65        System.exit(1);
66      }
67    }
68    private void closeFile() {
69      try {
70        input.close();
71        System.exit(0);
72      }
73      catch (IOException io) {
74        System.err.println(io.toString());
75        System.exit(1);
76      }
77    }
78    public static void main(String args[]) {
79      new ReadSeqFile();
80    }
81  }
```

2. 클래스 사이의 관계

■ 의존 관계

```
01 import java.io.*;
02 import java.awt.*;
03 import java.awt.event.*;
04
05 public class CreateSeqFile extends Frame implements ActionListener {
06    private TextField account, name, balance;
07    private Button enter, done;
08    private DataOutputStream output; // 필터 스트림 객체
09    public CreateSeqFile() {
10       super("고객 파일 생성");
11       try {
12          output = new DataOutputStream(new FileOutputStream("client.doc"));
13       }
14       catch (IOException e) {
15          System.err.println(e.toString());
16          System.exit(1);
17       }
18       setSize(250, 130);
19       setLayout(new GridLayout(4,2));
20       add(new Label("계좌 번호"));
21       account = new TextField();       // 계좌 번호 입력 필드
```

2. 클래스 사이의 관계

■ 의존 관계

```
22      add(account);
23      add(new Label("이름"));
24      name = new TextField(20);        // 이름 입력 필드
25      add(name);
26      add(new Label("잔고"));
27      balance = new TextField(20);     // 잔고 입력 필드
28      add(balance);
29      enter = new Button("입력");        // 입력된 데이터를 저장하는 버튼
30      enter.addActionListener(this); // 이벤트와 연결
31      add(enter);
32      done = new Button("종료");         // 입력을 종료하는 버튼
33      done.addActionListener(this);  // 이벤트와 연결
34      add(done);
35      setVisible(true);
36    }
37    public void addRecord() {
38      int accountNo = 0;
39      Double d;
40    if(!account.getText().equals("")) {              // 계좌 번호의 입력 체크
41        try {
42          accountNo = Integer.parseInt(account.getText());
43          if(accountNo > 0) {
44            output.writeInt(accountNo);          // 필터 스트림을 통해 정수 저장
45            output.writeUTF(name.getText());     // 문자열 저장
46            d = new Double(balance.getText());     // 실수 객체 생성
47            output.writeDouble(d.doubleValue()); // 실수 저장
48          }
49
50          account.setText("");                       // 텍스트 필드 삭제
51          name.setText("");
52          balance.setText("");
53        }
54        catch (NumberFormatException nfe) {
55          System.err.println("정수를 입력해야 합니다.");
56        }
57        catch(IOException io) {
58          System.err.println(io.toString());
59          System.exit(1);
60        }
```

```
61      }
62    }
63    public void actionPerformed(ActionEvent e) {
64      addRecord();                                  // 입력된 데이터를 파일에 저장
65      if(e.getSource()==done) {
66      try {
67        output.close();                             // 파일 닫기
68      }
69
70      catch (IOException io) {
71        System.err.println(io.toString());
72      }
73      System.exit(0);                               // 프로그램 종료
74      }
75    }
76  }
```

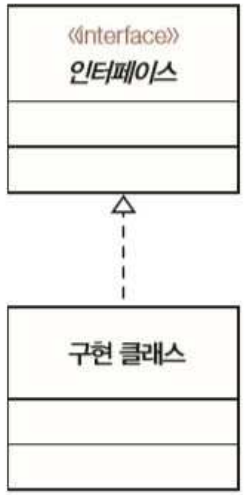
2. 클래스 사이의 관계

■ 실체화 관계

- 추상 클래스나 인터페이스를 상속받아 자식 클래스가 추상 메서드를 구현할 때 사용
- 클래스 이름을 이탤릭체로 표시하며 <<abstract>>로 표기

추상 클래스와 인터페이스

추상 클래스와 인터페이스

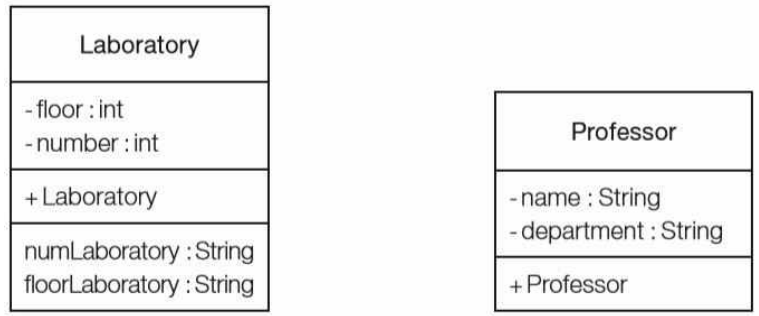


실체화 관계

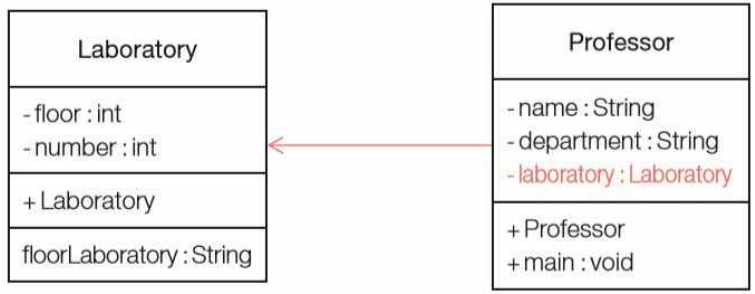
3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

■ 단방향 연관 관계

- Laboratory연구실 클래스
  - 층수와 방 번호를 속성으로 함
  - 층수와 방 번호를 출력하는 멤버변수와 매개변수를 갖는 생성자를 가짐
- Professor교수 클래스
  - 이름과 학과를 속성으로 함,
  - 매개변수를 갖는 생성자를 가짐



연구실 클래스와 교수 클래스

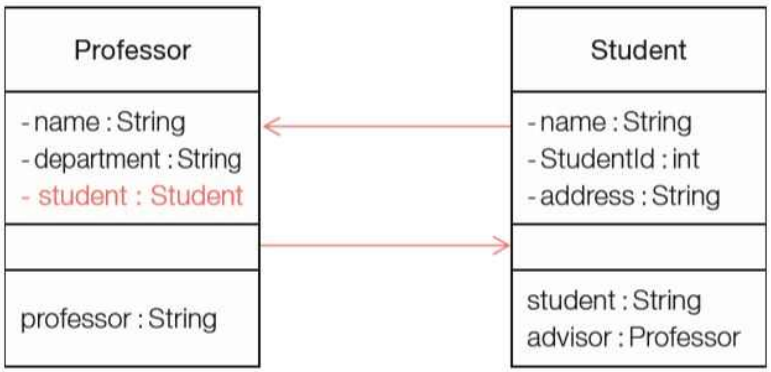


연구실 클래스와 교수 클래스의 단방향 연관 관계

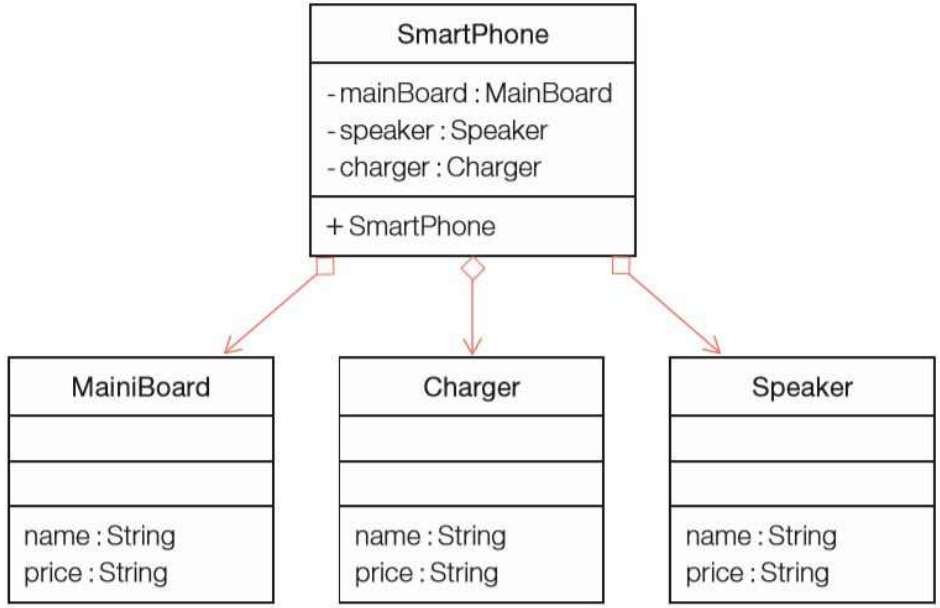
3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

■ 양방향 연관 관계

- Professor교수 클래스
  - 이름과 학과를 속성으로 함,
  - 교수명을 반환하는 메서드를 가짐
- Student학생 클래스
  - 이름, 학번, 주소를 속성으로 함
  - 학생 이름을 반환하는 메서드, 지도교수명을 출력하는 메서드를 가짐



교수 클래스와 학생 클래스의 양방향 연관 관계



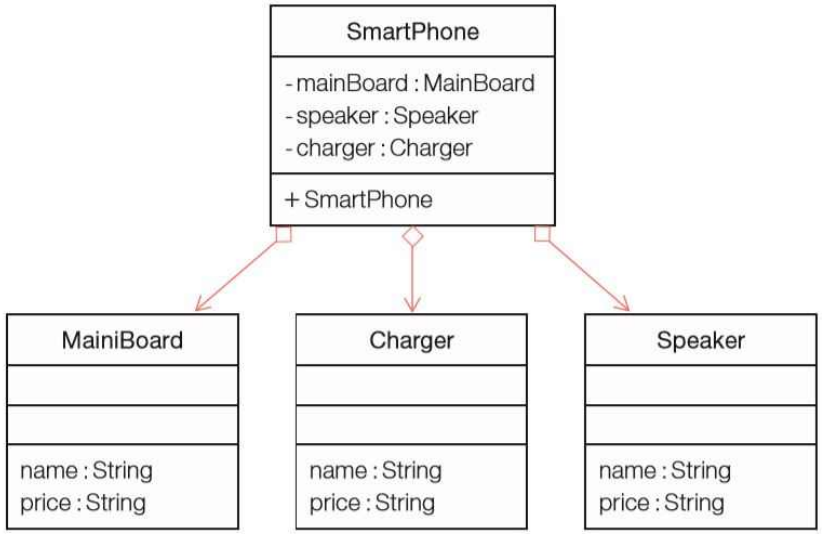
스마트폰 클래스의 복합 관계

3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

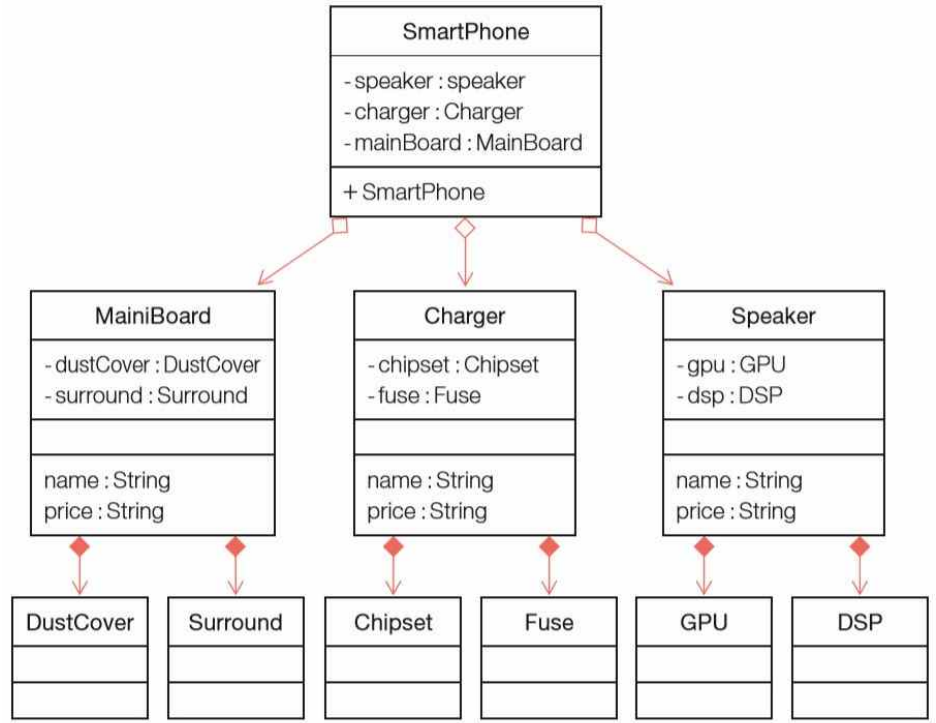
■ 집합 관계와 복합 관계

▪ 스마트폰

- 메인보드, 충전기, 스피커로 구성, 이들은 집합 관계
- 메인보드, 충전기, 스피커를 이루는 부품과 이들은 복합 관계



스마트폰 클래스의 복합 관계



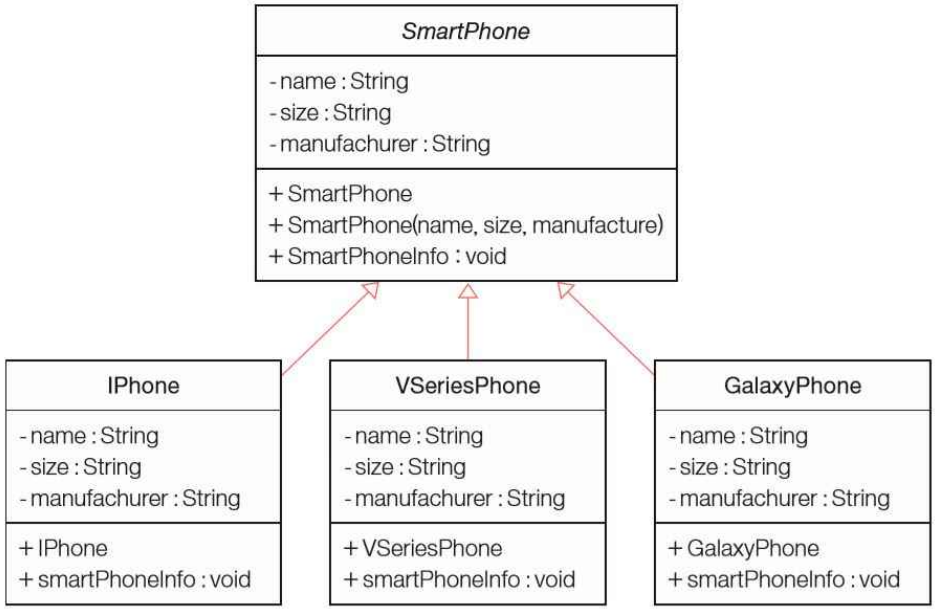
스마트폰 클래스의 복합 관계와 집합 관계

3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

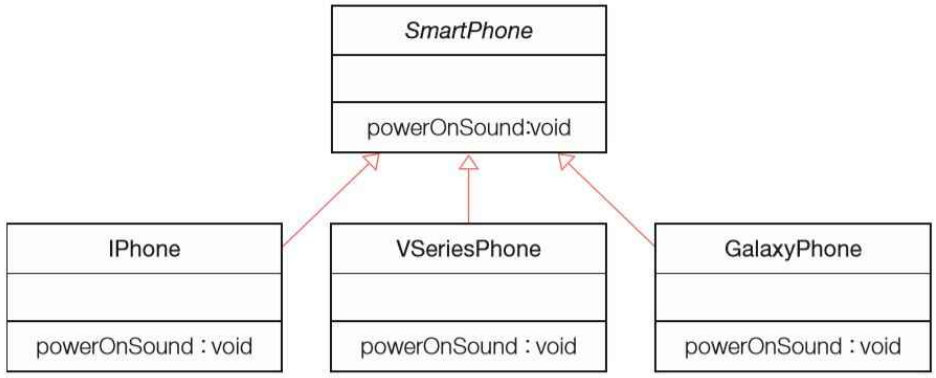
■ 일반화 관계

▪ 스마트폰

- 아이폰, V시리즈폰, 갤럭시폰은 모두 스마트폰의 한 종류
- 스마트폰과 각 클래스의 관계는 일반화 관계



스마트폰 클래스의 일반화 관계

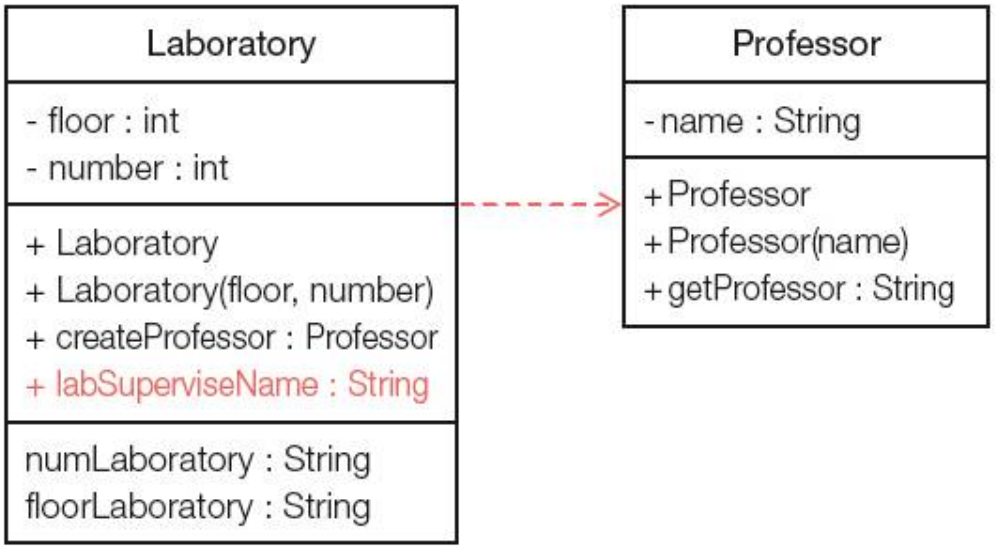


스마트폰 클래스의 일반화 관계의 예

3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

■ 의존 관계

- 연구실
  - 연구실 관리자라는 메서드를 가짐
  - 교수 객체를 이용하여 멤버 함수를 이용
- 교수
  - 교수명을 할당 받기 위해 객체를 생성
  - get 메서드를 이용해 교수를 등록
  - 연구실의 정보를 받기 위해 연구실 객체를 생성하여 해당 값을 할당

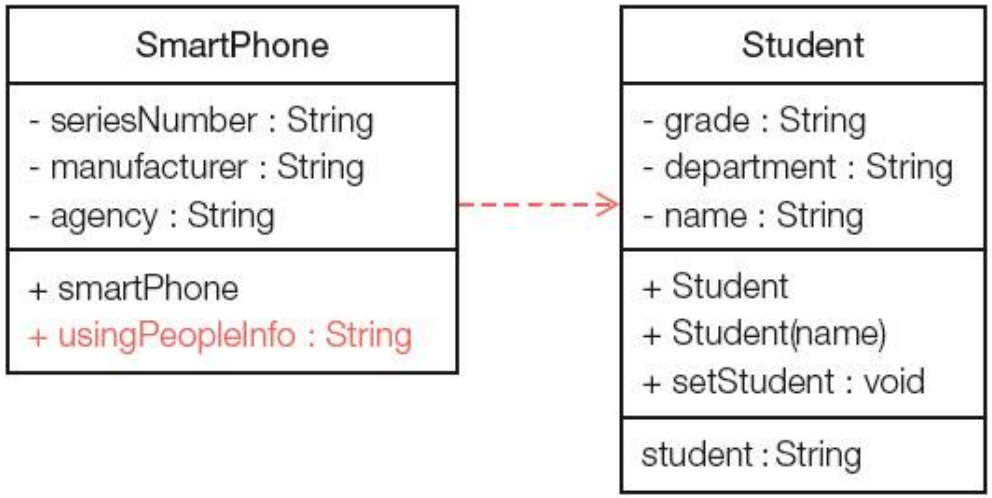


연구실 클래스와 교수 클래스의 의존 관계

3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

■ 의존 관계

- 스마트폰
  - 사용자 정보를 반환하는 메서드
  - 학생 객체를 이용하여 학생 클래스의 이름을 반환하는 메서드를 이용
- 학생
  - 학생 클래스에서는 스마트폰 객체를 생성
  - 스마트폰의 사용자 정보를 반환하는 멤버 변수를 이용
  - 매개변수 값으로 학생 객체를 생성하여 할당

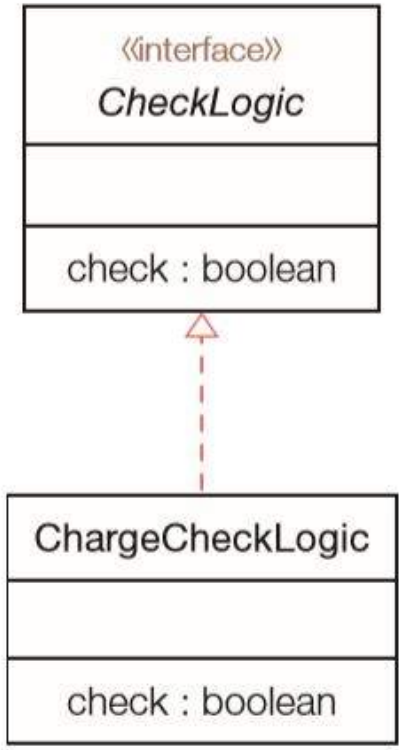


스마트폰 클래스와 학생 클래스의 의존 관계

3. 클래스 다이어그램의 단계별 모델링 : 다양한 관계 구현

■ 실체화 관계

- 충전기의 인터페이스를 실체화하여 충전 상태를 체크



충전기 인터페이스의 실체화 관계

3. 클래스 다이어그램의 모델링 연습

■ 날씨 관련 시스템

▪ 클래스 추출

- 에어컨은 온도가 30도 이상일 경우 작동하고 20도 이하일 경우 정지
- 제습기는 습도가 70% 이상이면 작동하고 30% 이하이면 정지
- 강수량이 200mm 이상일 경우 알림 경보가 울림

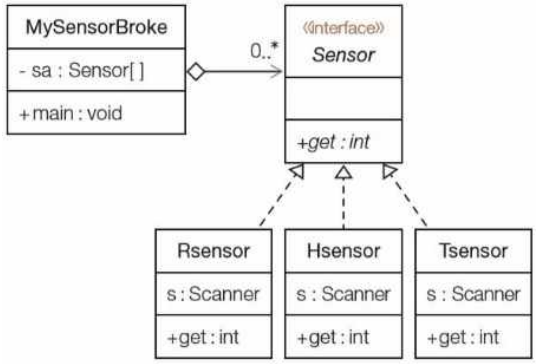
클래스 추출	메서드 추출
MySensorBroker(main) Alarm Dehumid Aircon Warning Broker Rsensor Hsensor Tsensor Frame Graphics	날씨 정보 입력(get, put) 경보 알림(Warning paint setColor fillOval) 기계 작동(cast)

추상화	객체 추출	인터페이스 추출
Actor	Broker	Sensor

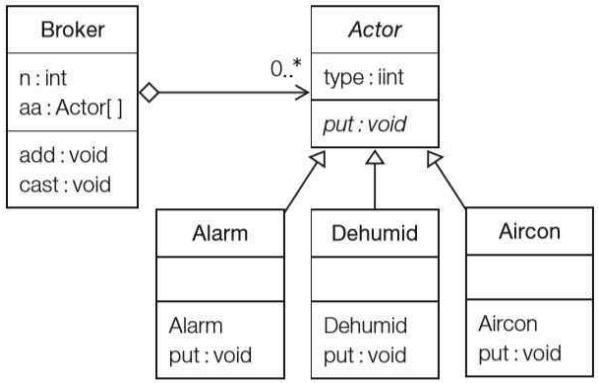
3. 클래스 다이어그램의 모델링 연습

날씨 관련 시스템

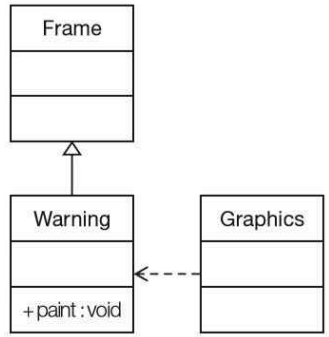
- 각 센서 클래스를 재정의하여 사용
  - Sensor 인터페이스의 객체를 생성하여 MySensorBroker 메인 클래스에서 프로그램을 실행



  - 추상 클래스의 메인 클래스는 Broker로 Actor를 객체 생성하여 프로그램을 실행

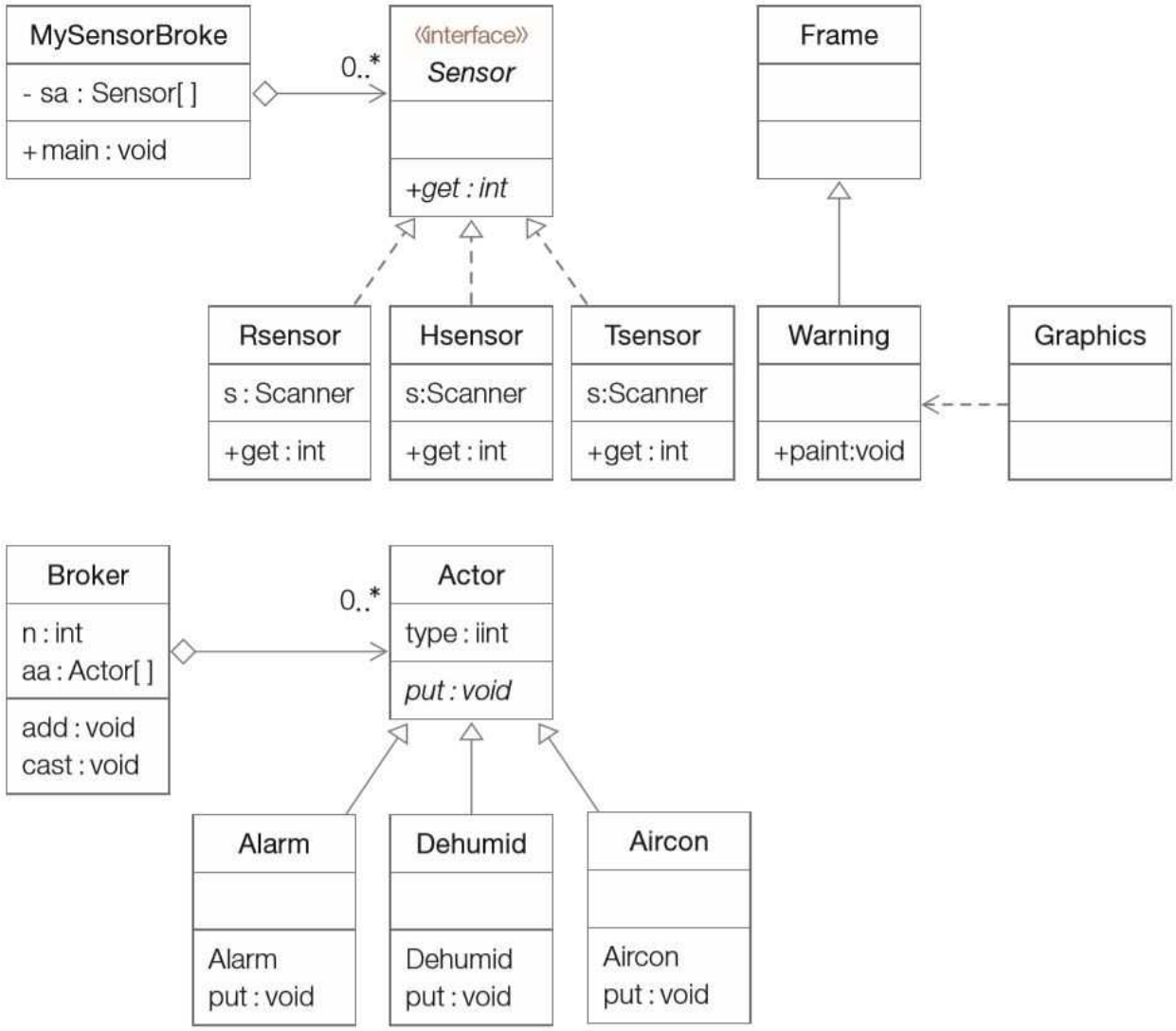


  - GUI 프로그래밍 환경을 구성하는 Frame 클래스와 Graphics 클래스를 상속받아 Warning 에서 paint() 메서드를 통하여 프로그램을 실행



3. 클래스 다이어그램의 모델링 연습

- **날씨 관련 시스템**



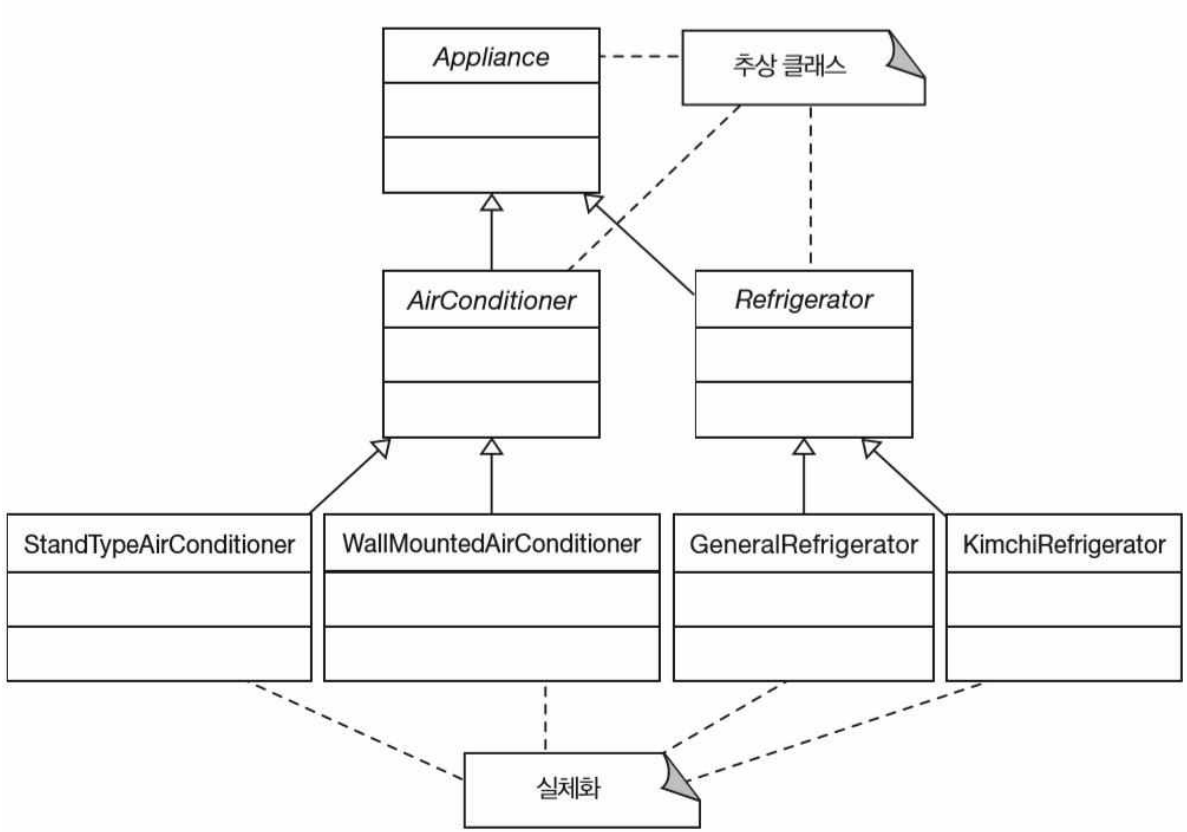
날씨에 관한 클래스 다이어그램

3. 클래스 다이어그램의 모델링 연습

■ 가전제품

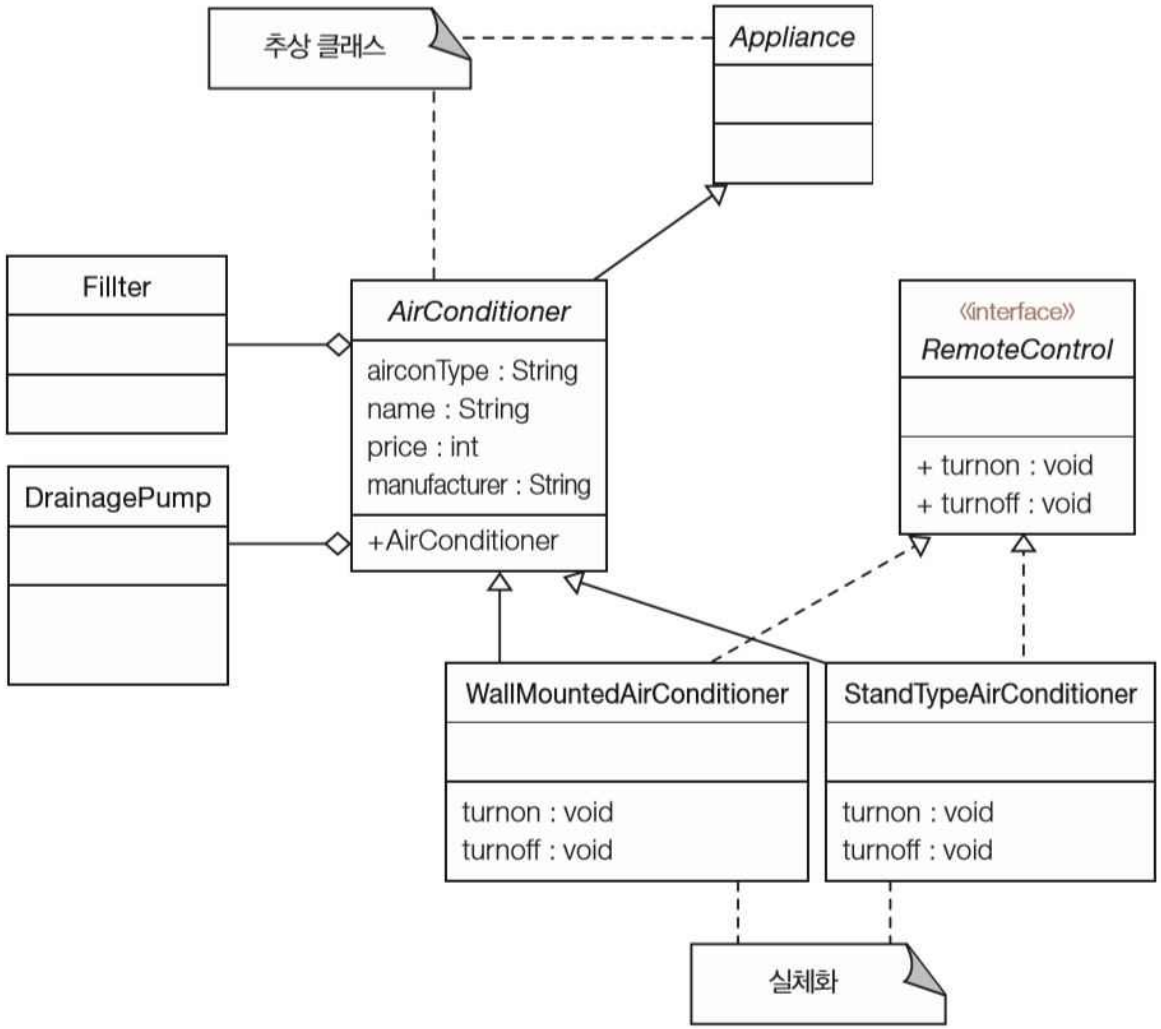
▪ 가전제품Appliance 클래스

- 에어컨과 냉장고의 공통 내용을 포함하는 상위 클래스
- 에어컨AirConditioner과 냉장고Refrigerator를 하위 클래스를 두 개씩 갖는 추상 클래스



3. 클래스 다이어그램의 모델링 연습

- **가전제품**



가전제품에 관한 클래스 다이어그램