2. 객체지향 원리

학습목표

학습목표

- 추상화 이해하기
- 캡슐화 이해하기
- 일반화(상속) 관계 이해하기
- 다형성 이해하기

2.1 추상화

❖ 어떤 영역에서 필요로 하는 속성이나 행위를 추출하는 작업

- 관심 있는 부분에 더욱 집중할 수 있다

"추상화는 사물들의 공통된 특징, 즉 추상적 특징을 파악해 인식의 대상으로 삼는 행위다. 추상화가 가능한 개체들은 개체가 소유한 특성의 이름으로 하나의 집합class을 이룬다. 그러므로 추상화한다는 것은 여러 개체들을 집합으로 파악한다는 것과 같다. 추상적 특성은 집합을 구성하는 개체들을 '일반화'하는 것이므로 집합의 요소들에 보편적인 것이다."

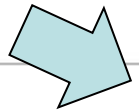
그림 2-1 추상화

추상화

❖ 추상화 개념을 사용하지 않는 경우에는 각각의 개체를 구분

코드 2-1

```
switch(자동차 종류)
  case 아우디:    // 아우디 엔진 오일을 교환하는 과정을 기술
  case 벤츠:      // 벤츠 엔진 오일을 교환하는 과정을 기술
end switch
```



코드 2-2

```
switch(자동차 종류)
  case 아우디:    // 아우디 엔진 오일을 교환하는 과정을 기술
  case 벤츠:      // 벤츠 엔진 오일을 교환하는 과정을 기술
  case BMW:       // BMW 엔진 오일을 교환하는 과정을 기술
end switch
```

추상화

❖ 자동차 개념 도입

코드 2-3

```
void changeEngineOil(Car c) {
  c.changeEngineOil();
}
```

2.2 캡슐화

- 요구사항의 변경
  - 소프트웨어 개발의 골치거리
  - 해결책은 요구사항 변경을 당연한 것으로 받아들이고 이에 대비하는 것이다.
  - 캡슐화를 통해 높은 응집도와 낮은 결합도를 갖는 설계

그림 2-3 정보 은닉

캡슐화

코드 2.4

```
private int top;
private int[] itemArray;
private int stackSize;
```

코드 2-6

```
public class StackClient {
  public static void main(String[] args) {
    ArrayListStack st = new ArrayListStack(10);
    st.push(20);
    System.out.print(st.peek());
  }
}
```

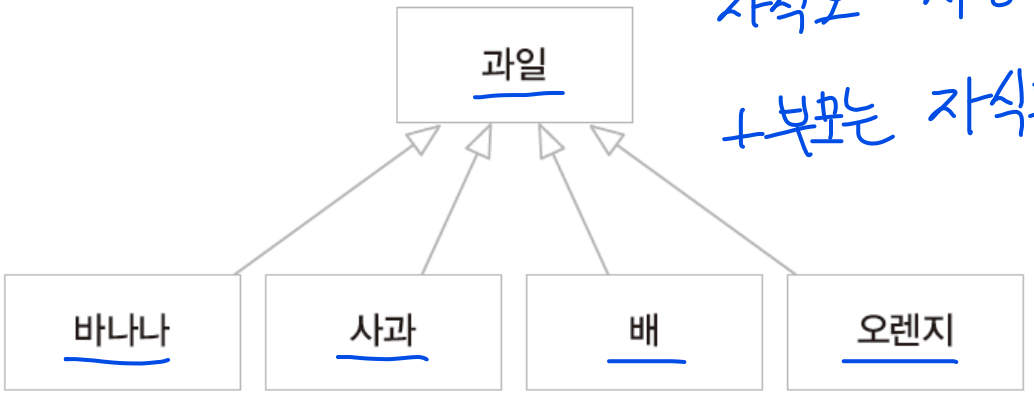
그림 2-4 정보 은닉의 장점

public 이 아닌 private 로 설정함으로써,
내부 메서드를 통해서만 변수에 접근할 수 있다 (good)
∴ 정보 은닉

2.3 일반화 관계

- 일반화(상속)을 속성이나 기능의 재사용 관점에서만 보는 것은 극히 제한된 관점
- 철학에서 일반화generalization는 “여러 개체들이 가진 공통된 특성을 부각시켜 하나의 개념이나 법칙으로 성립시키는 과정”

그림 2-5 일반화 관계



부모의 기능을
자식도 사용할 수 있다
+부모는 자식의 상위 개념

일반화 관계

코드 2-7

```
가격 총합 = 0
while(장바구니에 과일이 있다) {
  switch(과일 종류)
    case 사과:
      가격 총합 = 가격 총합 + 사과 가격
    case 배:
      가격 총합 = 가격 총합 + 배 가격
    case 바나나:
      가격 총합 = 가격 총합 + 바나나 가격
    case 오렌지:
      가격 총합 = 가격 총합 + 오렌지 가격
}
```

일반화 개념을 사용하지 않았을 때

키위가 추가된 경우

```
가격 총합 = 0
while(장바구니에 과일이 있다) {
        switch(과일 종류)
        case 사과:
                가격 총합 = 가격 총합 + 사과 가격
        case 배:
                가격 총합 = 가격 총합 + 배 가격
        case 바나나:
                가격 총합 = 가격 총합 + 바나나 가격
        case 오렌지:
                가격 총합 = 가격 총합 + 오렌지 가격
        case 키위:
                가격 총합 = 가격 총합 + 키위 가격
}
```

case를 계속 추가하기 번거롭다...

일반화 관계

코드 2-9

```
int computeTotalPrice(LinkedList<Fruit> f) {
  int total = 0;
  Iterator<Fruit> itr = f.iterator();
  while(itr.hasNext()) {
    Fruit curFruit = itr.next();
    total = total + curFruit.calculatePrice();
  }
  return total;
}
```

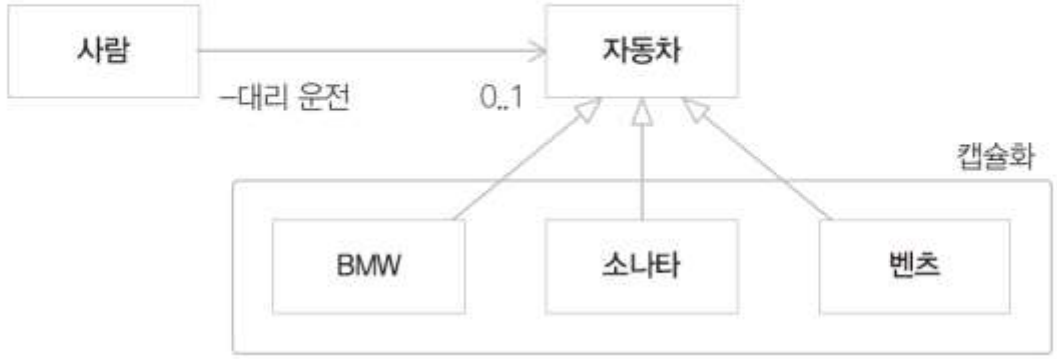
컬렉션에 저장된 요소들을 순차적으로 읽어오기 위해 사용한다

계속 추가할 필요 없음

일반화 관계

- 일반화는 클래스 자체를 캡슐화하여 변경에 대비할 수 있는 설계를 가능하게 한다
  - `새로운 클래스가 추가되더라도 클라이언트는 영향 받지 않음

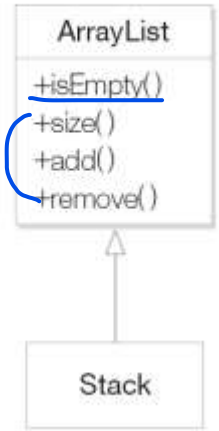
그림 2-6 일반화는 또 다른 캡슐화



일반화와 위임

가능 재사용은 일반화 보다 실체화가 좋다.

그림 2-7 ArrayList를 이용한 Stack의 구현



굳이 필요?

코드 2-10

```
class MyStack<String> extends ArrayList<String> {
  public void push(String element) {
    add(element);
  }

  public String pop() {
    return remove(size() - 1);
  }
}
```

아무 조건 없이 실행하게 해준다 (bad)

일반화와 위임

- 두 자식 클래스 사이에 'is a kind of 관계'가 성립되지 않을 때 상속을 사용하면 불필요한 속성이나 연산(빚이라 해도 될 것이다)도 물려받게 된다.

코드 2-10을 이용해 만든 스택이 스택의 무결성 조건에 위배되도록 main 메서드를 작성하라.

```
public class Main {
  public static void main(String[] args) {
    MyStack<String> st = new MyStack<String>();

    st.push("insang1");
    st.push("insang2");
    st.set(0, "insang3"); // 허용되어서는 안됨
    System.out.println(st.pop());
    System.out.println(st.pop());
  }
}
```

스택의 규칙이 어긋남 (bad)

일반화와 위임

- 어떤 클래스의 일부 기능만을 사용하고 싶을 경우에는 위임을 사용하라
- 일반화를 위임으로 변환하는 프로세스
  1. 자식 클래스에 부모 클래스의 인스턴스를 참조하는 속성을 만든다. 이 속성 필드를 this로 초기화한다.
  2. 서브 클래스에 정의된 각 메서드에 1번에서 만든 위임 속성 필드를 참조하도록 변경한다.
  3. 서브 클래스에서 일반화 관계 선언을 제거하고 위임 속성 필드에 슈퍼 클래스의 객체를 생성해 대입한다.
  4. 서브 클래스에서 사용된 슈퍼 클래스의 메서드에도 위임 메서드를 추가한다.
  5. 컴파일하고 잘 동작하는지 확인한다.

단계 1

코드 2-11

```
public class MyStack<String> extends ArrayList<String> {
  private ArrayList<String> arList = this;

  public void push(String element) {
    add(element);
  }

  public String pop() {
    return remove(size() - 1);
  }
}
```

① 자식 클래스에

부모 클래스의 인스턴스 ②

③ ②를 참조

단계 2

코드 2-12

```
public class MyStack<String> extends ArrayList<String> {
  private ArrayList<String> arList = this;

  public void push(String element) {
    arList.add(element);
  }
  public String pop() {
    return arList.remove(arList.size() - 1);
  }
}
```

단계 1에서 만든 위임 속성 필드를 참조하게 함

단계 3

코드 2-13

(제거됨)

```
public class MyStack<String> {
  private ArrayList<String> arList = new ArrayList<String>();

  public void push(String element) {
    arList.add(element);
  }

  public String pop() {
    return arList.remove(arList.size() - 1);
  }
}
```

위임 속성 필드에 / this 대신 슈퍼클래스의 객체를 생성해 대입한다

단계 4

코드 2-14

```
public class MyStackDelegation<String> {
  private ArrayList<String> arList = new ArrayList<String>();

  public void push(String element) {
    arList.add(element);
  }

  public String pop() {
    return arList.remove(arList.size() - 1);
  }

  public boolean isEmpty() {
    return arList.isEmpty();
  }

  public int size() {
    return arList.size();
  }
}
```

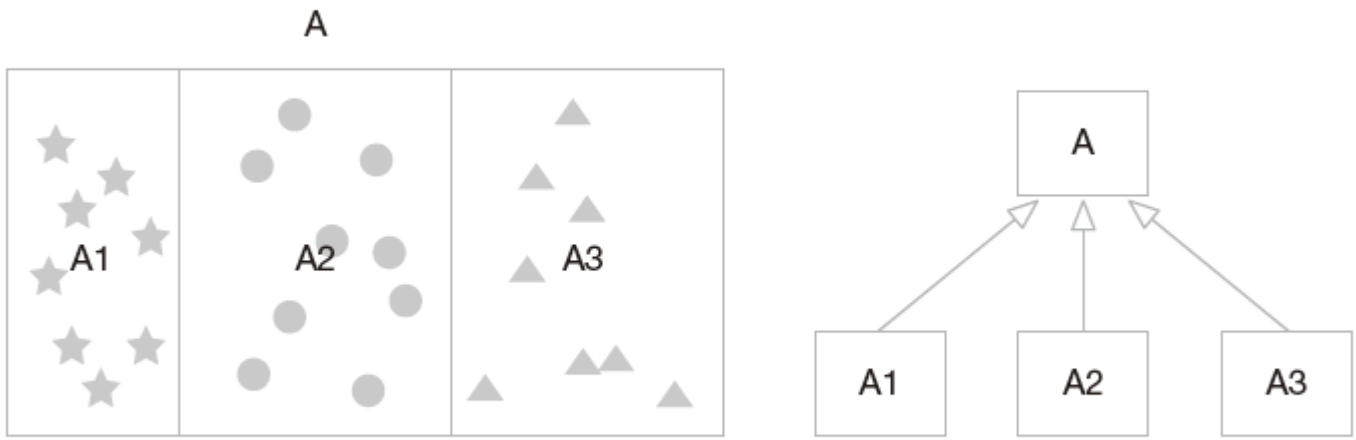
위임 메서드 추가

체크포인트

- Vector 클래스를 사용(위임)해 Stack 클래스를 구현하라.

집합과 일반화

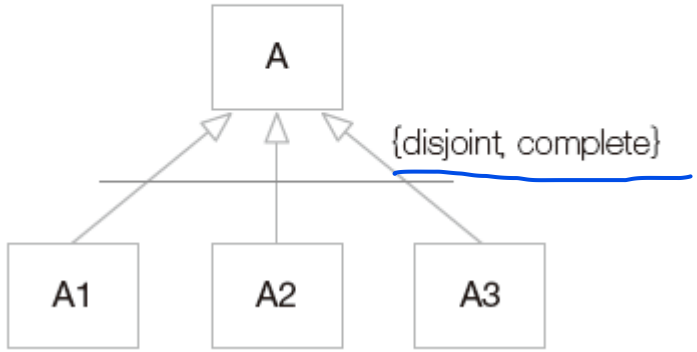
그림 2-8 집합과 일반화 관계



집합과 일반화

- 제약 {disjoint}는 자식 클래스 객체가 동시에 두 클래스에 속할 수 없다는 의미이고,
- {complete}는 자식 클래스의 객체에 해당하는 부모 클래스의 객체와 부모 클래스의 객체에 해당하는 자식 클래스의 객체가 하나만 존재한다는 의미다

그림 2-9 일반화 관계에서의 제약 조건



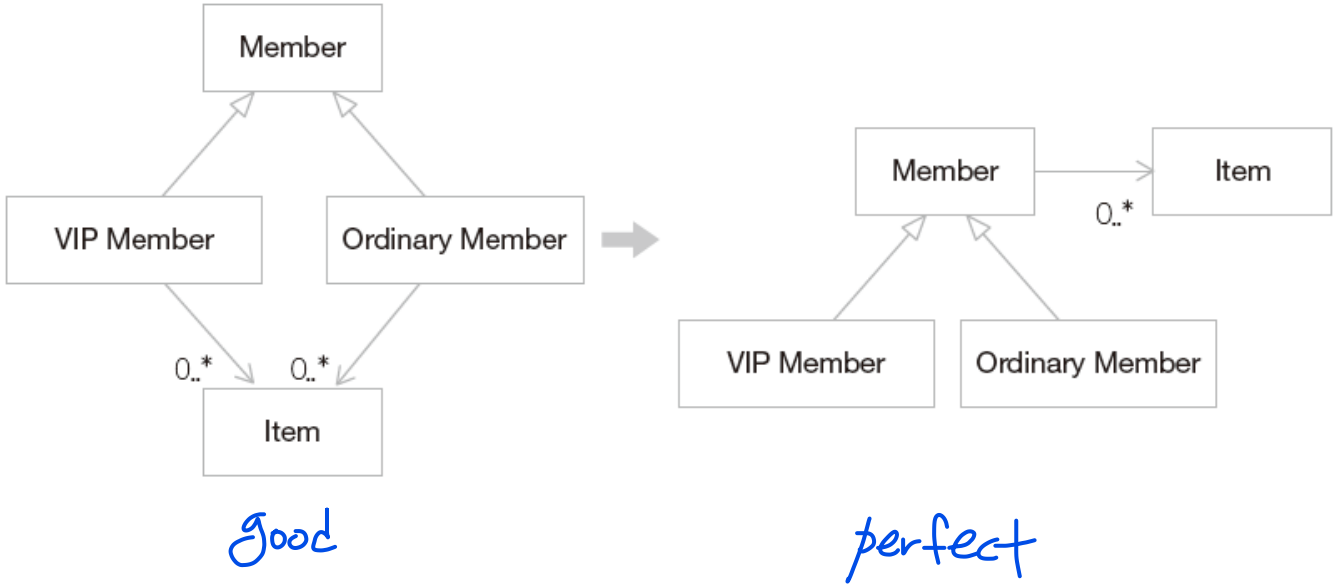
- $A = A1 \cup A2 \cup A3$
- $A1 \cap A2 \cap A3 = \varnothing$

∴ A1, A2, A3은 겹치지 않는다

연관 관계의 일반화

- 일반화는 연관 관계를 단순하게 만들 수 있다

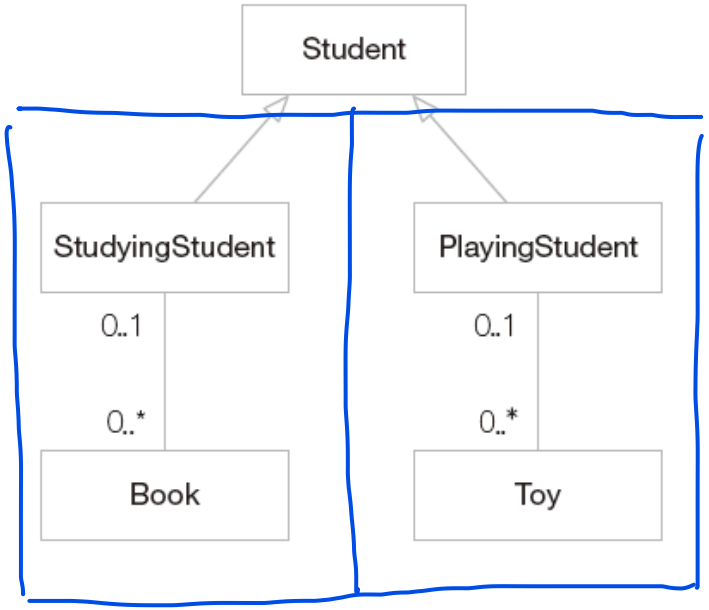
그림 2-10 집합론을 통한 연관 관계의 일반화



상호배타적 상태

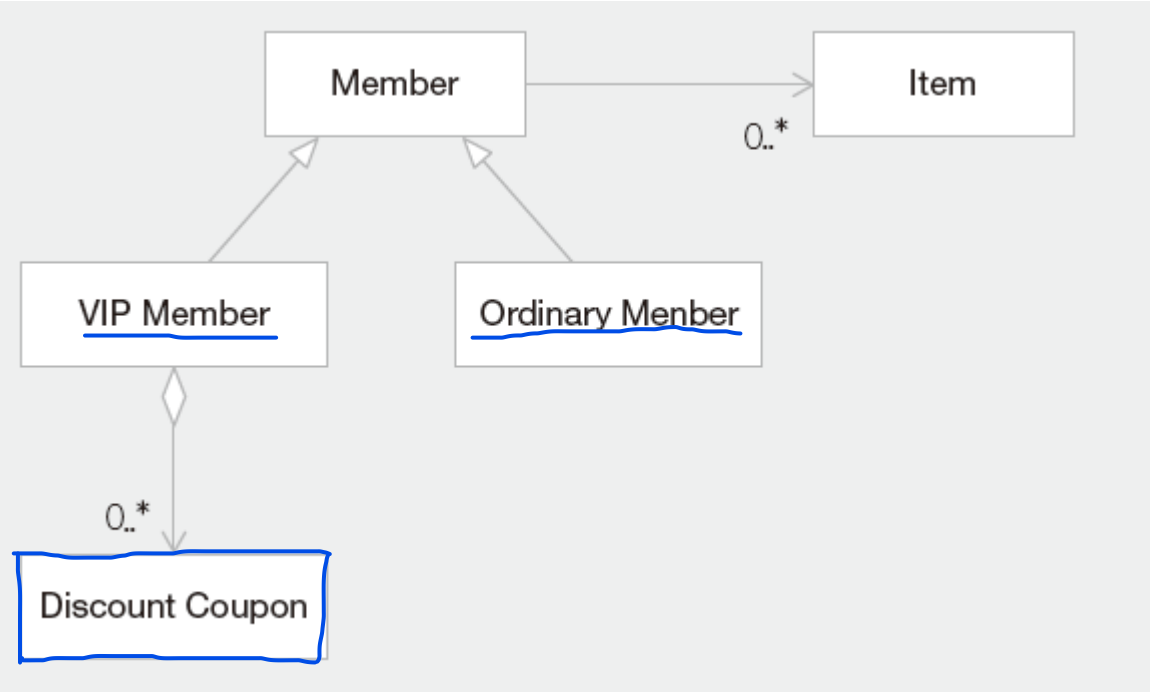
- 공부하고 있는 중에는 책만 볼 수 있다. 장난감을 가지고 공부 할 수 없다
- 노는 중에는 장난감만 갖고 놀 수 있다. 책을 가지고 놀지 않는다

그림 2-11 일반화 관계를 이용한 상호 배타적 관계 모델링



특수화

- VIP 멤버에게만 할인 쿠폰이 발행된다

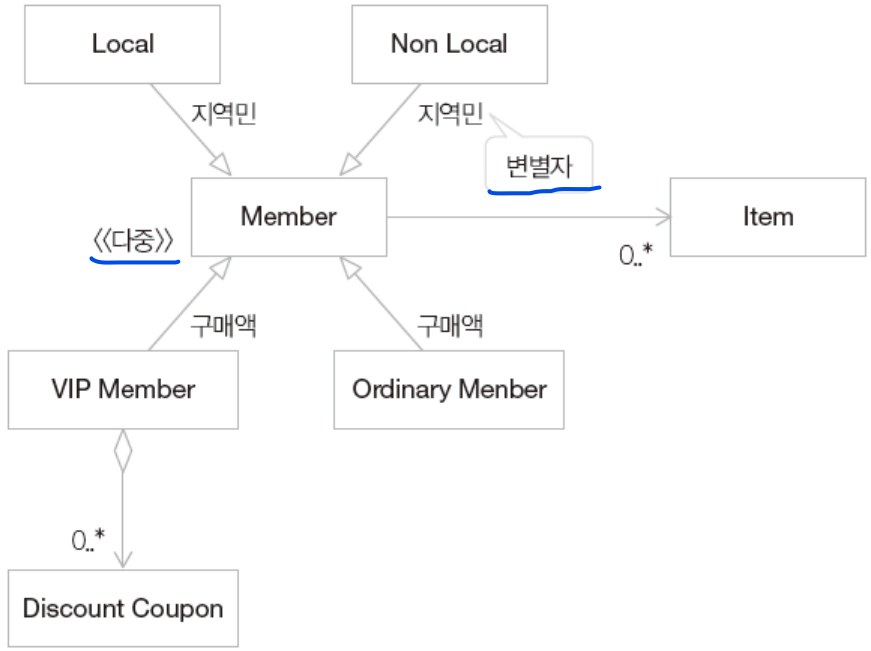


변별자와 다중 분류

- 변별자: 인스턴스 분류 기준
- 다중: 한 인스턴스가 동시에 여러 클래스에 속함 (다중분류)

(ex. 지역민이 아니지만 VIP 이다)

그림 2-12 변별자와 다중 분류



변별자와 다중 분류

- 지역주민이든 아니든 VIP 멤버에게는 할인 쿠폰이 지급된다
- How? 일반 회원이지만 지역 주민에게는 경품을 제공하도록 시스템에 새로운 요구사항을 추가

그림 2-12 변별자와 다중 분류

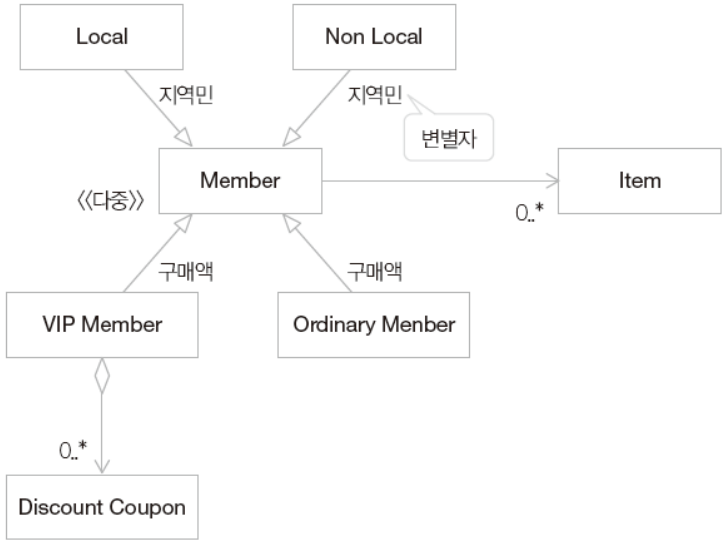
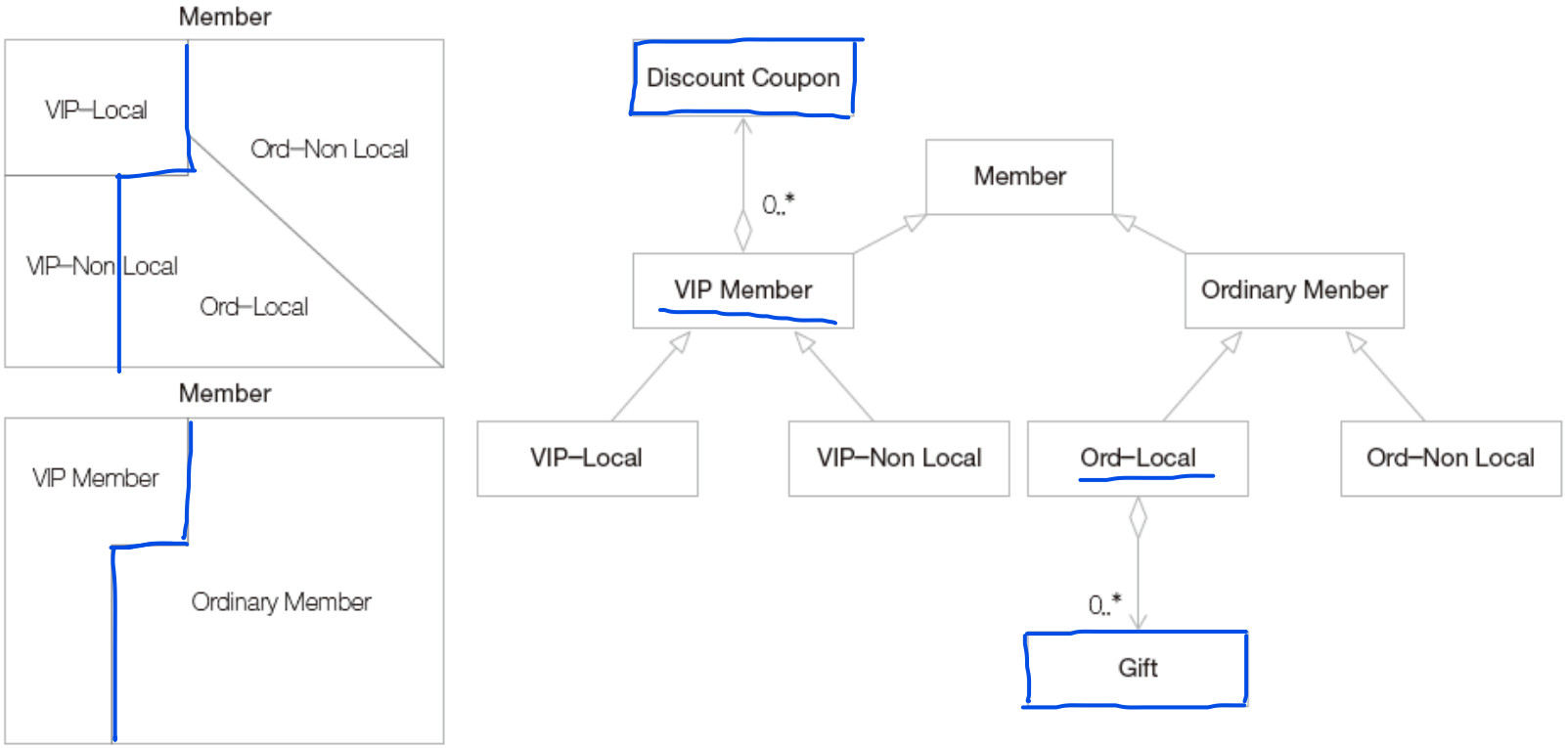
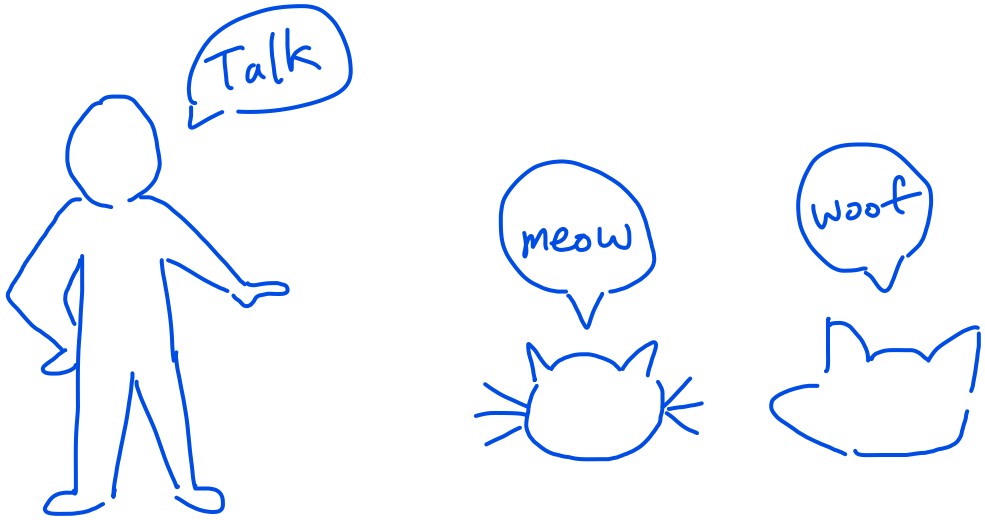


그림 2-13 집합과 일반화 관계



다형성

- 서로 다른 클래스의 객체가 같은 메시지를 받았을 때 각자의 방식으로 동작하는 능력’
- 코드 2-15, 2-16, 2-17



상속 규칙

❖ 피터 코드의 상속 규칙

- 자식 클래스와 부모 클래스 사이는 '역할 수행is role played by' 관계가 아니어야 한다.
- 한 클래스의 인스턴스는 다른 서브 클래스의 객체로 변환할 필요가 절대 없어야 한다.
- 자식 클래스가 부모 클래스의 책임을 무시하거나 재정의하지 않고 확장만 수행해야 한다.
- 자식 클래스가 단지 일부 기능을 재사용할 목적으로 유틸리티 역할을 수행하는 클래스를 상속하지 않아야 한다.
- 자식 클래스가 '역할role', '트랜잭션transaction', '디바이스device' 등을 특수화specialization해야 한다.

상속 규칙

그림 2-14 상속으로 표현한 역할 수행 관계

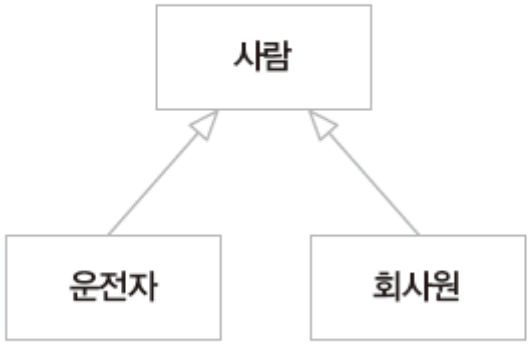
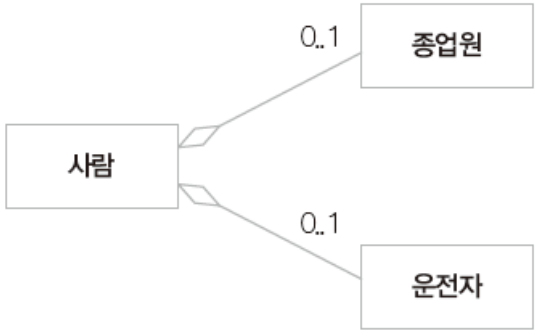


그림 2-15 집약 관계를 이용한 역할 수행 표현



택시 운전사라면?

bad