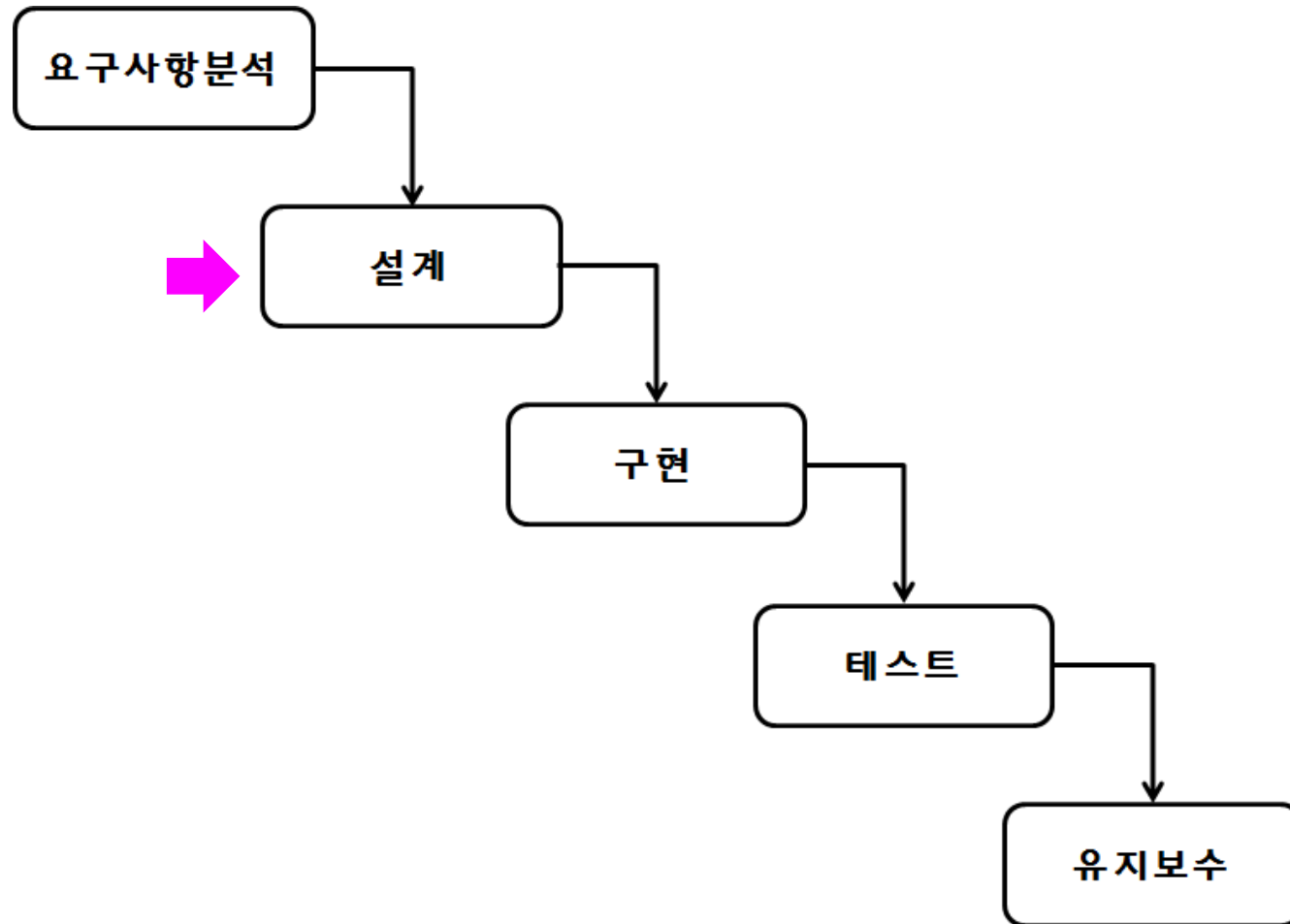


클래스 다이어그램 3




소프트웨어 개발 프로세스




[복습] 일반화(generalization) 관계와 실체화(realization) 관계

- 일반화(generalization) 관계와 실체화(realization) 관계
 - 일반화 관계 : 클래스 사이의 일반화(상속) 관계를 표현하면 '~이다(is-a)'로 해석
 - ※ 서브(하위)클래스는 슈퍼(상위)클래스의 한 종류이다.

표기법  자바의 extends 키워드로 구현

- 실체화 관계 : 특정 클래스의 명세를 실현(구현)하는 관계를 표현한다.
 - ※ 구현하는 클래스와 인터페이스 관계를 말함

표기법  자바의 implements 키워드로 구현

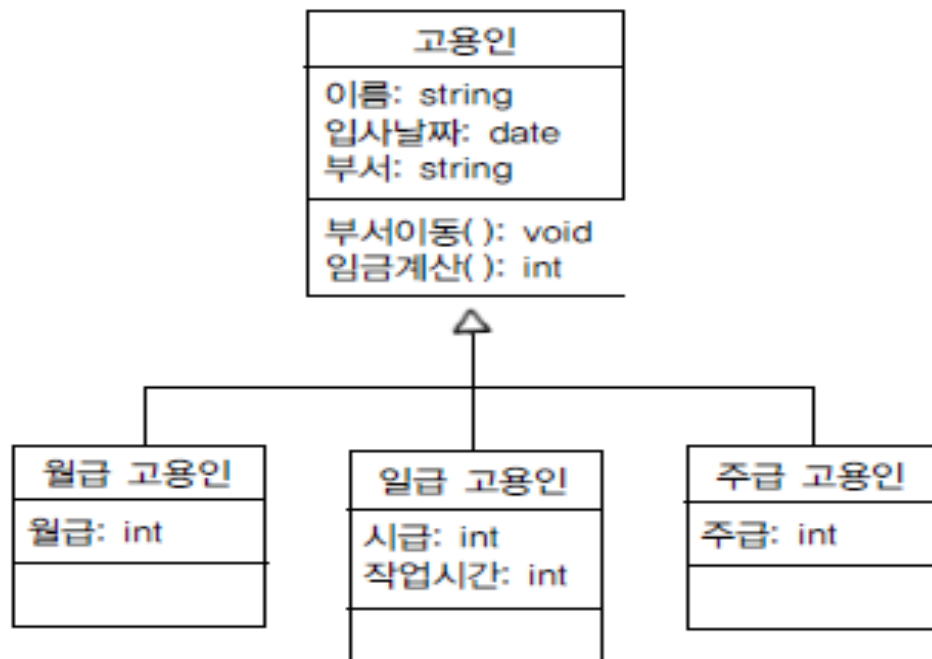
[복습] 일반화 관계 자바코드

- '고용인' 클래스의 일반화 관계 및 자바코드

표기법



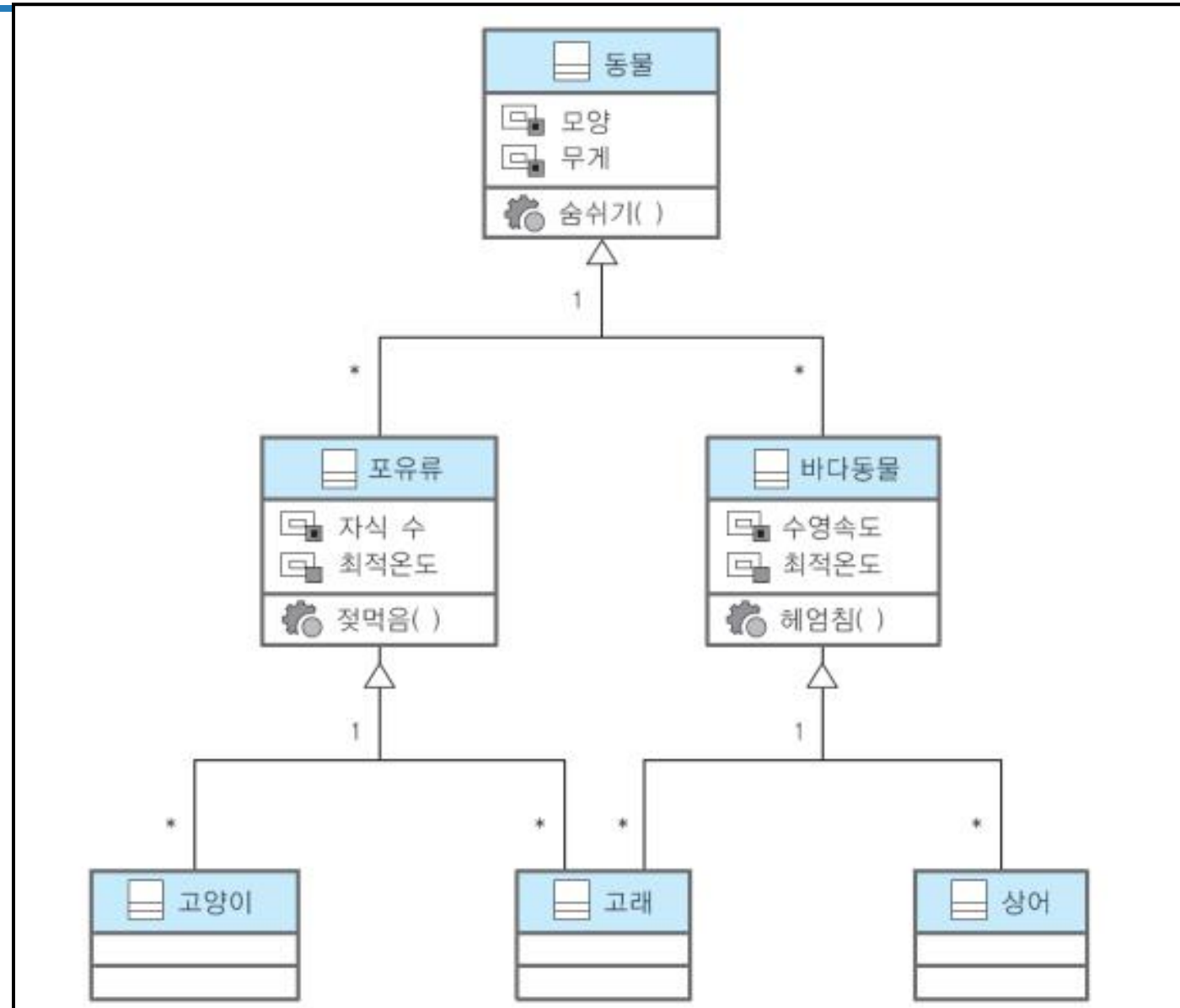
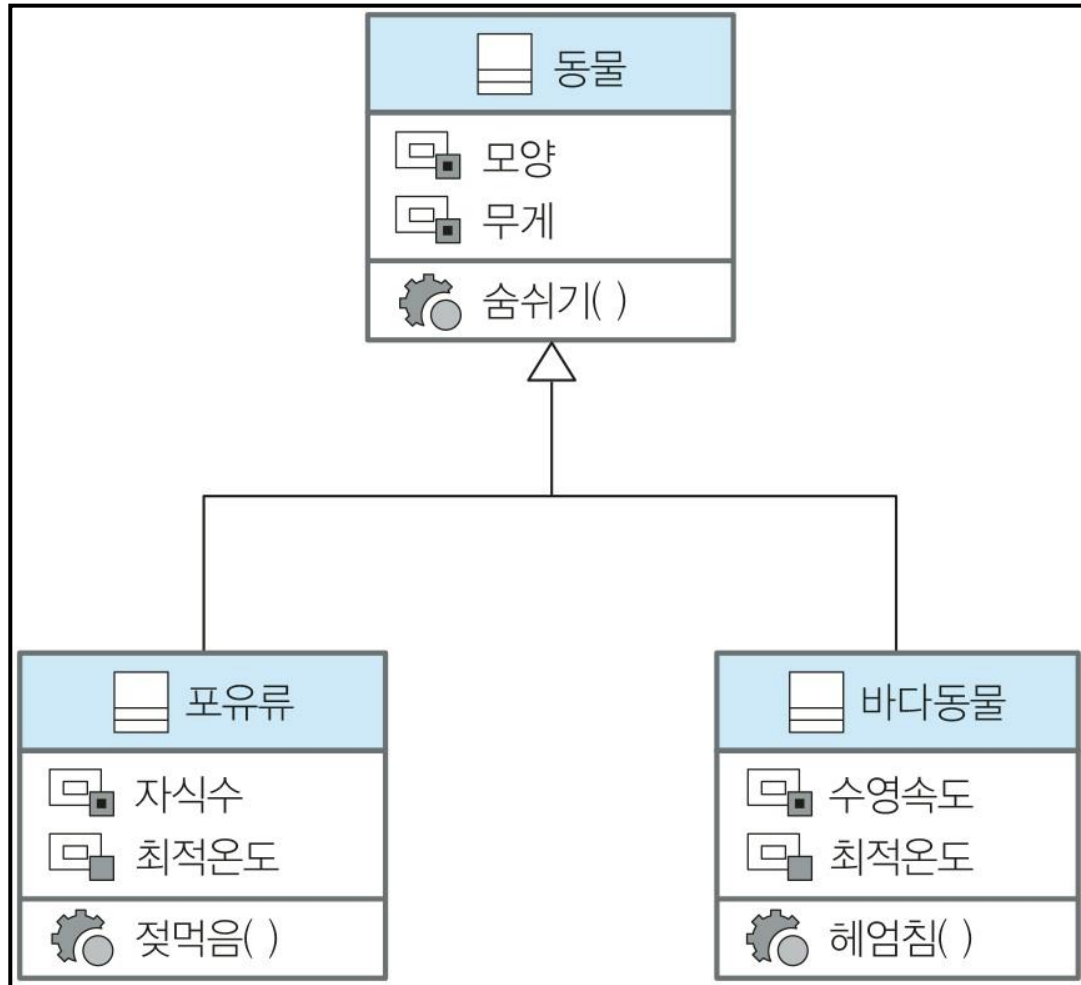
자바의 extends 키워드로 구현



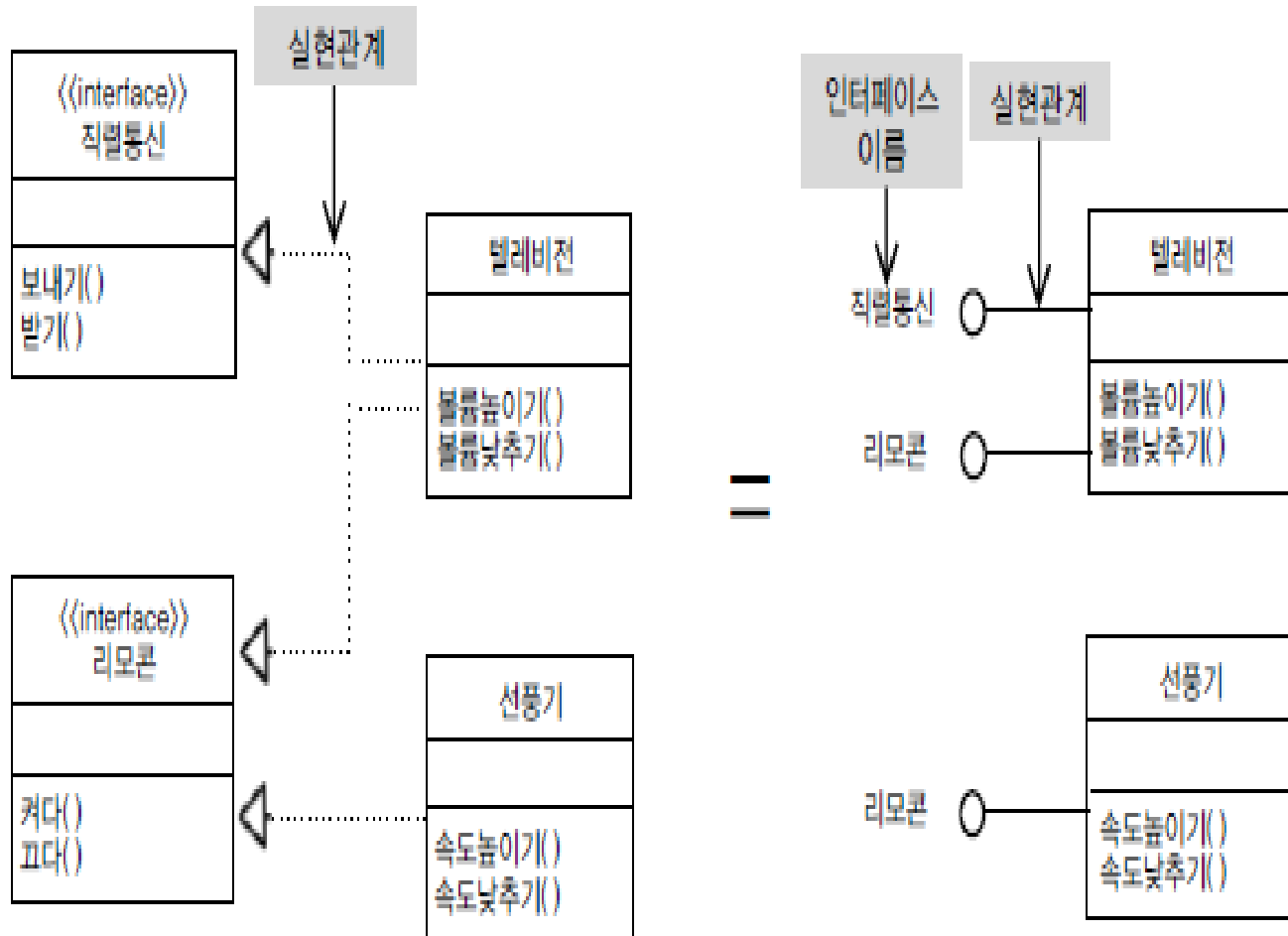
```
class 고용인 {
    String 이름;
    Date 입사날짜;
    String 부서;
    void 부서이동( );
    int 임금계산( );
}
```

```
class 월급고용인 extends 고용인 {
    int 월급;
}
class 일급고용인 extends 고용인 {
    int 시급;
    int 작업시간;
}
class 주급고용인 extends 고용인 {
    int 주급;
}
```

[복습] 일반화 관계 자바코드



[복습] 실체화(실현) 관계 구현하는 클래스와 인터페이스 관계를 말함



자바의 `implements` 키워드로 구현

직렬통신	리모콘
<pre> public interface SerialCom { void send(); void receive(); } </pre>	<pre> public interface RemoteControl { void turnOn(); void turnOff(); } </pre>
텔레비전	선풍기
<pre> public class TV implements SerialCom, RemoteControl { public void send() {.....}; public void receive() {.....}; public void turnOn() {.....}; public void turnOff() {.....}; public void volumeUp() {.....}; public void volumeDown() {.....}; } </pre>	<pre> public class Fan implements RemoteControl { public void turnOn() {.....}; public void turnOff() {.....}; public void speedUp() {.....}; public void speedDown() {.....}; } </pre>

클래스 다이어그램의 관계

<클래스 다이어그램은 개발자에게 가장 중요한 다이어그램>

■ 관계

- 클래스 및 **클래스 간의 관계**를 통해 시스템의 전체적인 모습을 그려준다.
- 클래스가 하나로만 이루어지는 시스템을 상상하는 것은 어렵다. 객체 지향 시스템도 여러 개의 클래스가 서로 긴밀한 관계를 맺어 기능을 수행한다.

- 이 관계를 통해 메시지를 주고받으며 기능 제공

- 클래스 선정 -> 속성 정의 -> 메소드 추출 -> 관계 설정 순서로 작성

↓
유스케이스에서 추출

↓
변수로 구현

※ 클래스 계층의 설계도 쉬운 일이 아니다. 상세한 이해 및 경험과 노하우 필요
따라서 소프트웨어 공학이란 분야를 무조건 어렵다고 생각하는 경향 존재하는 것도 사실
하지만 주먹구구식 소프트웨어 개발을 지양하고, 원칙에 입각한 개발을 지향하는 실질적
인 내용을 다루는 학문으로 개발자에게 필수적인 소양

클래스 다이어그램의 관계

■ 클래스 다이어그램 관계의 종류

- 일반화(generalization) 관계 : 상속, 서로 비슷한 속성 및 오퍼레이션을 수행하는 클래스들을 묶어서 처리
 - ☞ 속성 : 클래스 안의 데이터를 정의, 대부분 변수로 구현
 - ☞ 오퍼레이션 : 클래스 안에 포함된 함수
- 실체화(realization) 관계 : 인터페이스와 이를 실제 구현하는 클래스 사이의 관계

☞ 클래스가 서로 연결되어(연관되어) 있음을 나타내는 관계

- 연관(association) 관계 : 두 객체가 생성과 동시에 지속적인 연관을 맺는 경우 (지속적인 관계)
- 의존(dependency) 관계 : 두 객체가 필요에 따라서 일시적으로 연관을 맺는 경우 (일시적인 관계)

☞ 특별한 형태의 연관 관계, 전체와 부분과의 관계를 명확하게 명시하고자할 때 사용


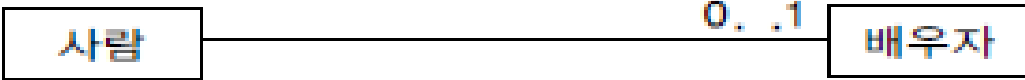
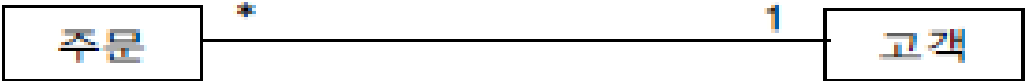
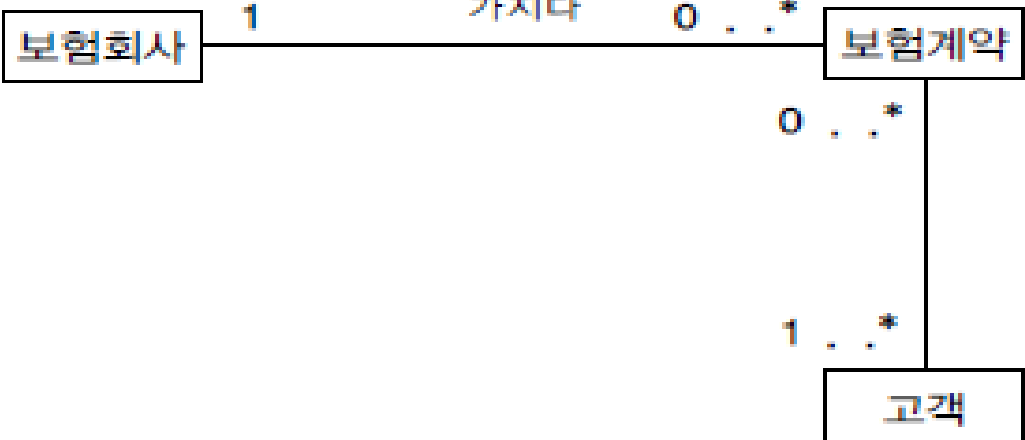
- 집합(aggregation) 관계 : 전체와 부분 간의 관계를 나타내는 관계 (약한 포함 관계)
(전체가 없어져도 독립적으로 존재, 예: 데스크탑)
- 합성(composition) 관계 : 전체와 부분 간의 관계를 나타내는 관계 (강한 포함 관계)
(전체가 없어지면 같이 없어지는 존재, 예 : 노트북)

연관(association) 관계

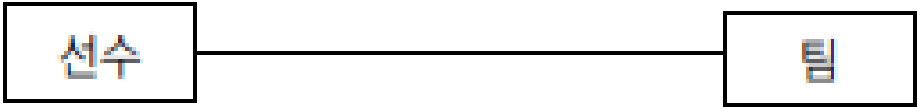
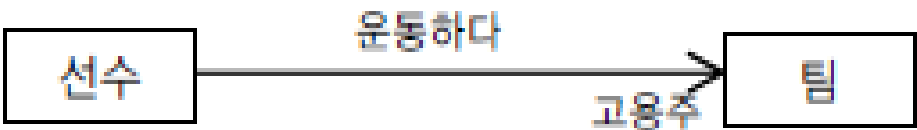
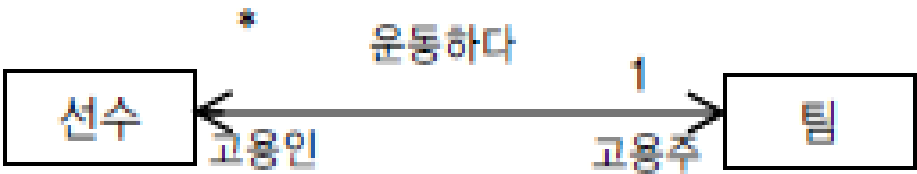
- 연관(association) 관계 (지속적인 관계)
 - 클래스가 서로 연결되어(연관되어) 있음을 나타내는 관계
 - 두 클래스(객체)가 생성과 동시에 지속적인 연관을 맺는 경우 (멤버 변수로 표현되어 클래스 구조에 영향)
 - 구조적으로 연관을 맺고 있음, 시간이 흘러도 유지되는 지속적인 관계

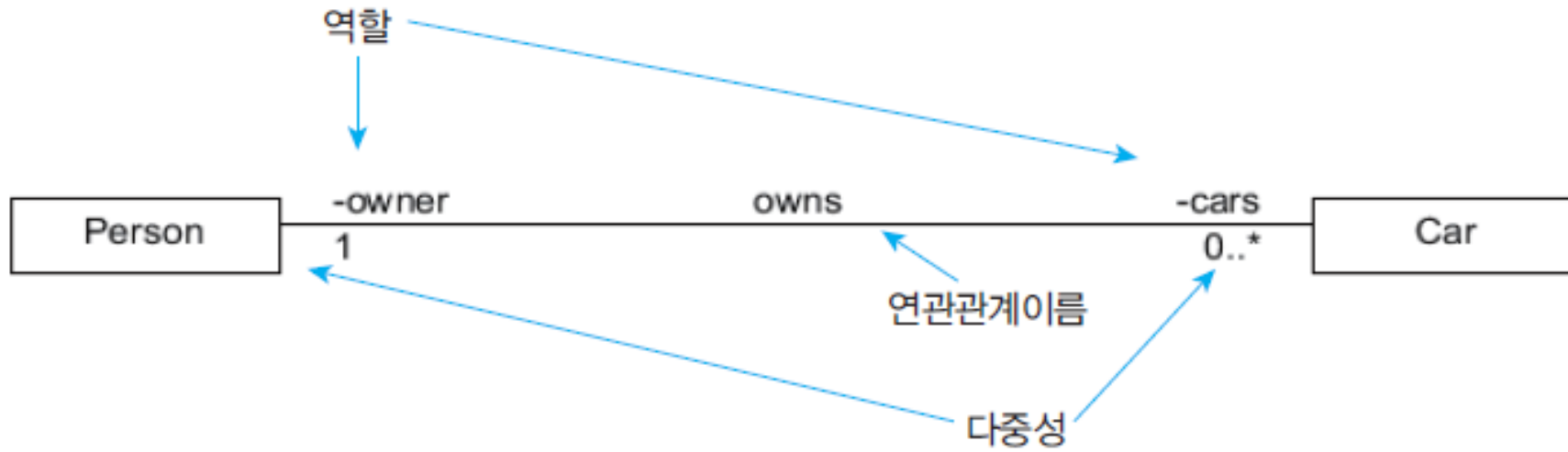
다중성 표시

다중성 표기	의미
1	엄밀하게 1
*	0 또는 그 이상
0..*	0 또는 그 이상
1..*	1 이상
0..1	0 또는 1
2..5	2 또는 3 또는 4 또는 5
1,2,6	1 또는 2 또는 6
1, 3..5	1 또는 3 또는 4 또는 5

연관(association) 관계 표기법 보기	해석
 <pre>classDiagram class 사람 class 자동차 사람 "1..*" -- "0..*" 자동차 : 소유하다 소유자</pre>	<ul style="list-style-type: none">· 한 사람의 소유자는 여러 대의 자동차를 소유할 수도 있고 전혀 소유하지 않을 수도 있다.· 한 자동차는 최소 한 사람 이상의 소유자와 관련된다.
 <pre>classDiagram class 사람 class 배우자 사람 -- "0..1" 배우자</pre>	<ul style="list-style-type: none">· 한 사람은 배우자가 없을 수도 있고 많아야 한 명의 배우자와 관련된다.· 한 배우자는 한 사람과 관련된다.
 <pre>classDiagram class 주문 class 고객 주문 "*" -- "1" 고객</pre>	<ul style="list-style-type: none">· 한 주문은 반드시 한 고객과 관련된다.· 한 고객은 여러 주문과 관련되며 주문과 관련되지 않을 수도 있다.
 <pre>classDiagram class 보험회사 class 보험계약 class 고객 보험회사 "1" -- "0..*" 보험계약 : 가지다 보험계약 "0..*" -- "1..*" 고객</pre>	<ul style="list-style-type: none">· 한 보험회사는 여러 보험계약을 가질 수 있지만 전혀 보험계약을 갖지 않을 수도 있다. 한 보험계약은 단지 한 보험회사와 관련된다.· 한 명의 고객은 여러 보험계약을 가질 수 있지만 전혀 보험계약을 갖지 않을 수도 있다. 한 보험계약은 한 명 이상의 고객과 관련된다.· 한 보험 계약은 한 보험회사와 한 명 이상의 고객과 관련될 수 있다.

연관(association) 관계에 대한 자바 코드 (연관관계는 방향성을 가질 수 있다)

연관관계	자바 코드
<div><pre>classDiagram class Player class Team Player --- Team</pre></div> <p>양방향 연관 관계</p>	<pre>class Player { Team t; } class Team { Player p; }</pre>
<div><pre>classDiagram class Player class Team Player --> Team : 운동하다, 고용주</pre></div> <p>단방향 연관 관계</p>	<pre>class Player { Team employer; } class Team { }</pre>
<div><pre>classDiagram class Player class Team Player "1" -- "*" Team : 운동하다 note for Player "고용인" note for Team "고용주"</pre></div> <p>양방향 연관 관계</p>	<pre>class Player Team employer; } class Team { Collection<Player> employee; }</pre>



```

class Person {
    private Car[] cars;
}

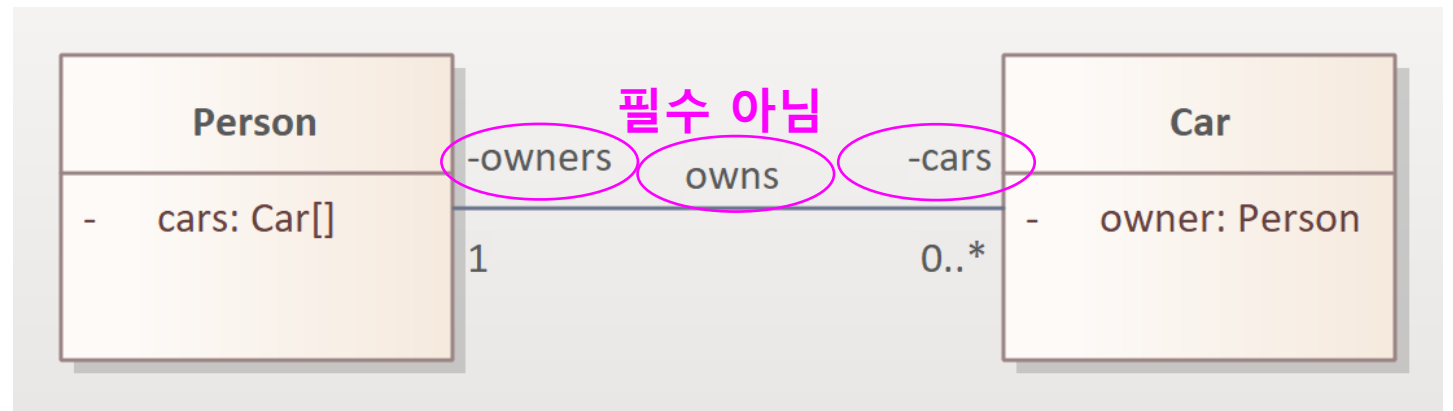
```

```

class Car {
    private Person owner;
}

```

association 선택하여 선언



실습

■ 만들어진 클래스

association 선택하여 선언결

Attributes Operations Receptions Parts / Properties Interaction Points		
Name	Type	Scope
<i>New Attribute...</i>		
cars	Car[]	Private

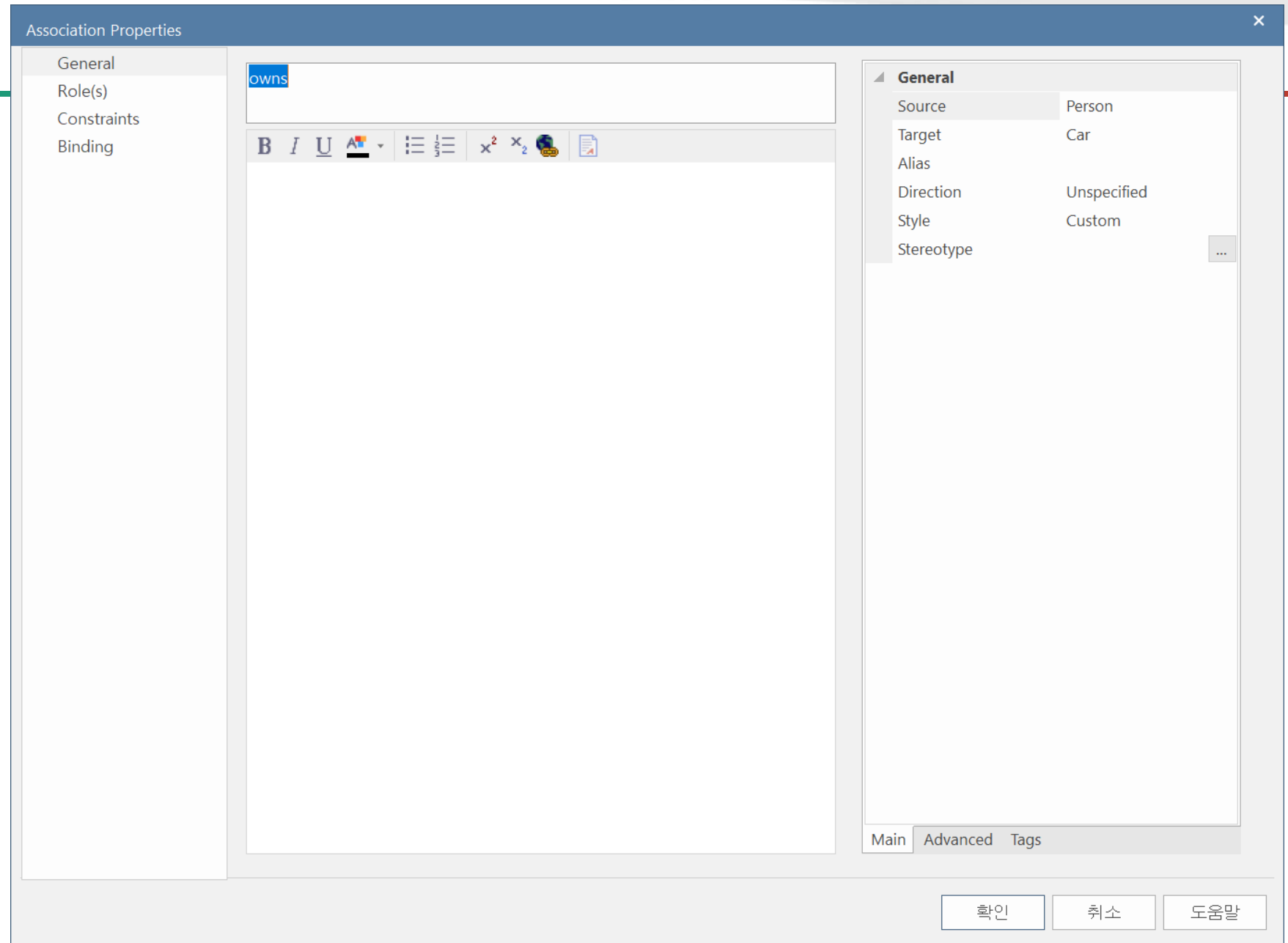
```
class Person {
    private Car[] cars;
}
```

Attributes Operations Receptions Parts / Properties Interaction Points		
Name	Type	Scope
<i>New Attribute...</i>		
owner	Person	Private

```
class Car {
    private Person owner;
}
```

실습

- 관계(선)을 두 번 클릭



General

Role(s)

Constraints

Binding

SOURCE Person

owners

Multiplicity

Multiplicity

1

Ordered

False

Allow Duplicates

False

Detail

Stereotype

Alias

Access

Private

Navigability

Unspecified

Aggregation

none

Scope

instance

Constraints

Qualifiers

Advanced

TARGET Car

cars

Multiplicity

Multiplicity

0..*

Ordered

False

Allow Duplicates

False

Detail

Stereotype

Alias

Access

Private

Navigability

Unspecified

Aggregation

none

Scope

instance

Constraints

Qualifiers

Advanced

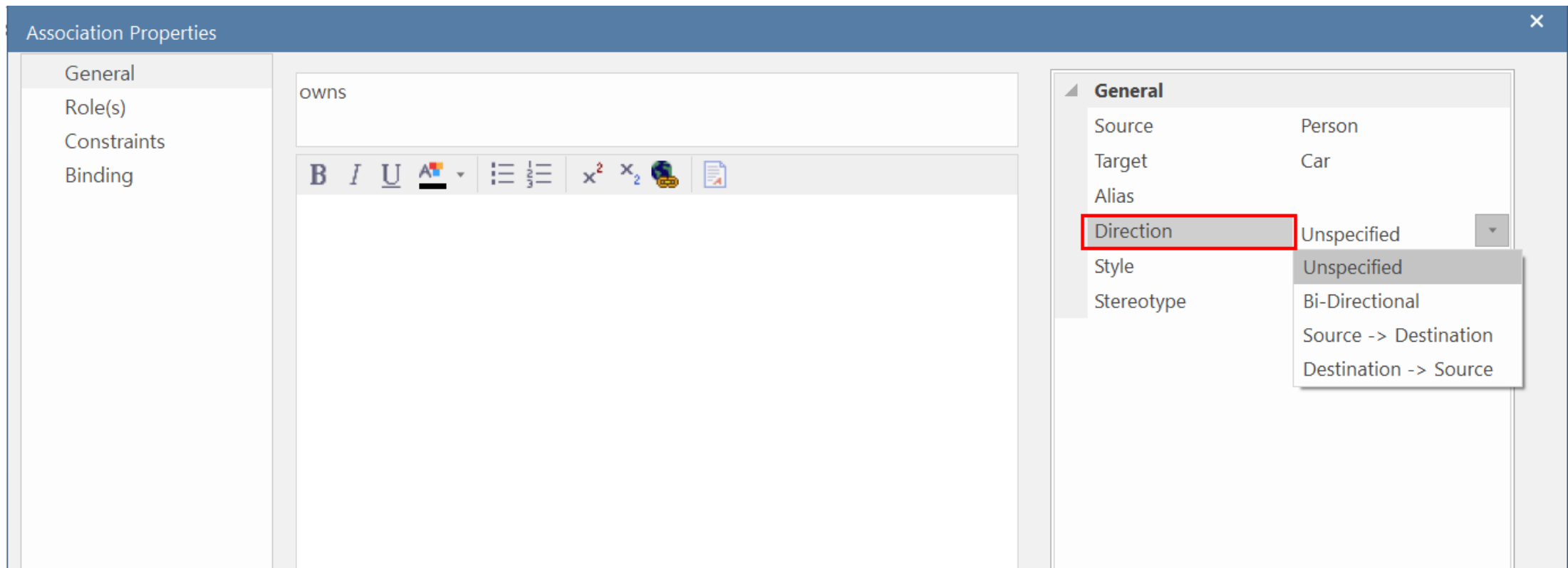
확인

취소

도움말

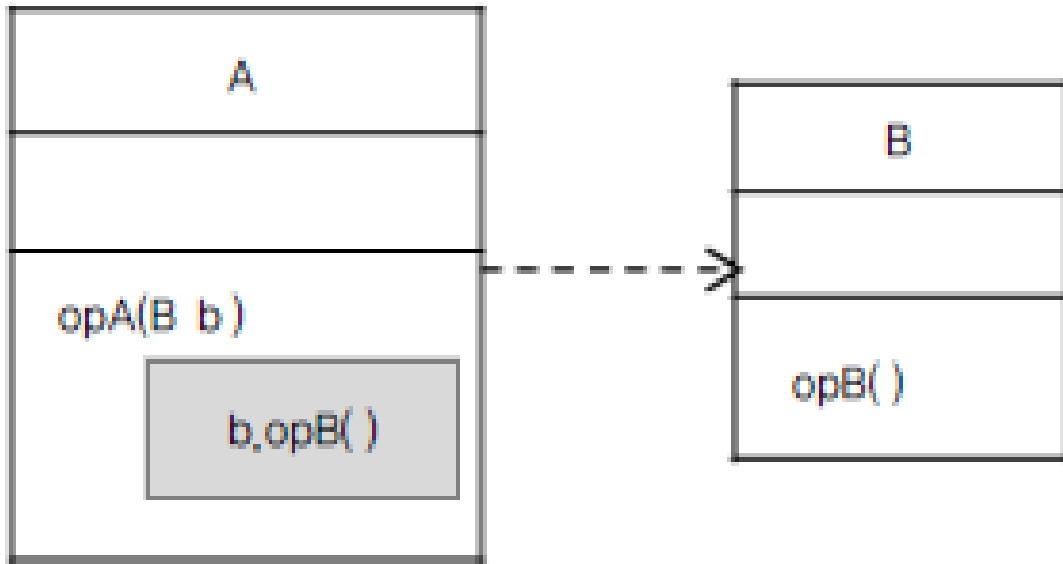
실습

- [참고] 만약 단방향 연관 관계를 표현하고자할 때

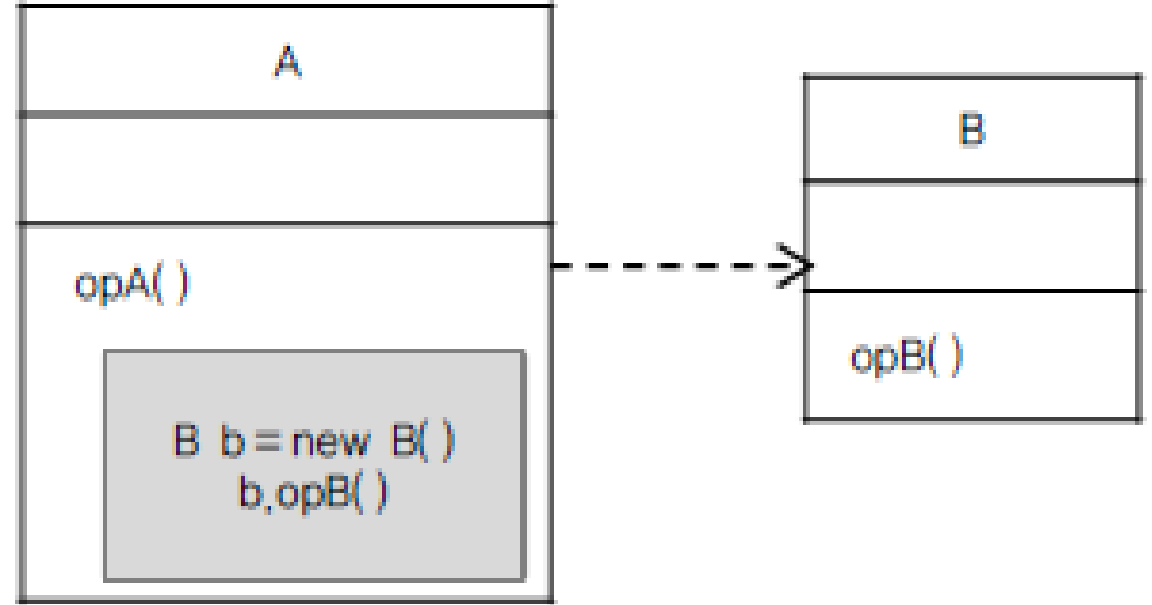


의존(dependency) 관계

- 의존(dependency) 관계 (일시적인 관계)
 - 클래스가 서로 연결되어(연관되어) 있음을 나타내는 관계
 - 두 클래스가 필요에 따라서 일시적으로 연관을 맺는 경우
(매개변수나 함수 내부에서 객체 생성에 의한 호출을 통해 구현, 클래스 구조에 영향을 안준다.)
 - 한 클래스가 다른 클래스를 (일시적으로) 사용할 때 발생하는 관계
 - 결합력이 가장 약한 관계, 일시적인 관계
 - 연관 관계와의 차이점은 '두 클래스의 관계가 한 메소드를 실행하는 동안처럼 짧은 시간 동안만 유지 '



매개변수에 의한 의존 관계



객체 생성에 의한 의존 관계

- 연관(Association) : 두 객체가 생성과 동시에 지속적인 연관을 맺는 경우

(예) Person객체는 생성후, 항상 Money 객체를 가지는 경우에는 Person과 Money는 연관 관계를 가지고, 아래와 같이 프로그램에서 표현한다

```
public class Person {
```

```
    private Money money; // Person 객체가 만들어지면서 Money 객체가 동시에 생성되어 유지
```

```
    .....
```

```
}
```

- 의존(Dependency) : 두 객체가 필요에 따라서 일시적으로 연관을 맺는 경우

(예) Person객체는 생성후, 돈을 쓰는 경우(useMoney)에만, Money 객체와 연결하여 돈을 사용하는 경우에는 의존 관계라고 하고, 아래와 같은 프로그램 형태로 표현한다

```
public class Person {
```

```
    .....
```

```
    public void useMoney(Money m) { //돈을 사용한 경우, 돈의 크기에 따라 관련 객체를 기동하고  
        m.spendMoney(total);      // 기동된 객체의 메소드를 이용하여 소비행위를 표현한다
```

```
    ..
```

```
}
```

dependency 선택하여 선 연결



집합(aggregation) 관계

☞ 집합(aggregation)/합성(composition) 관계

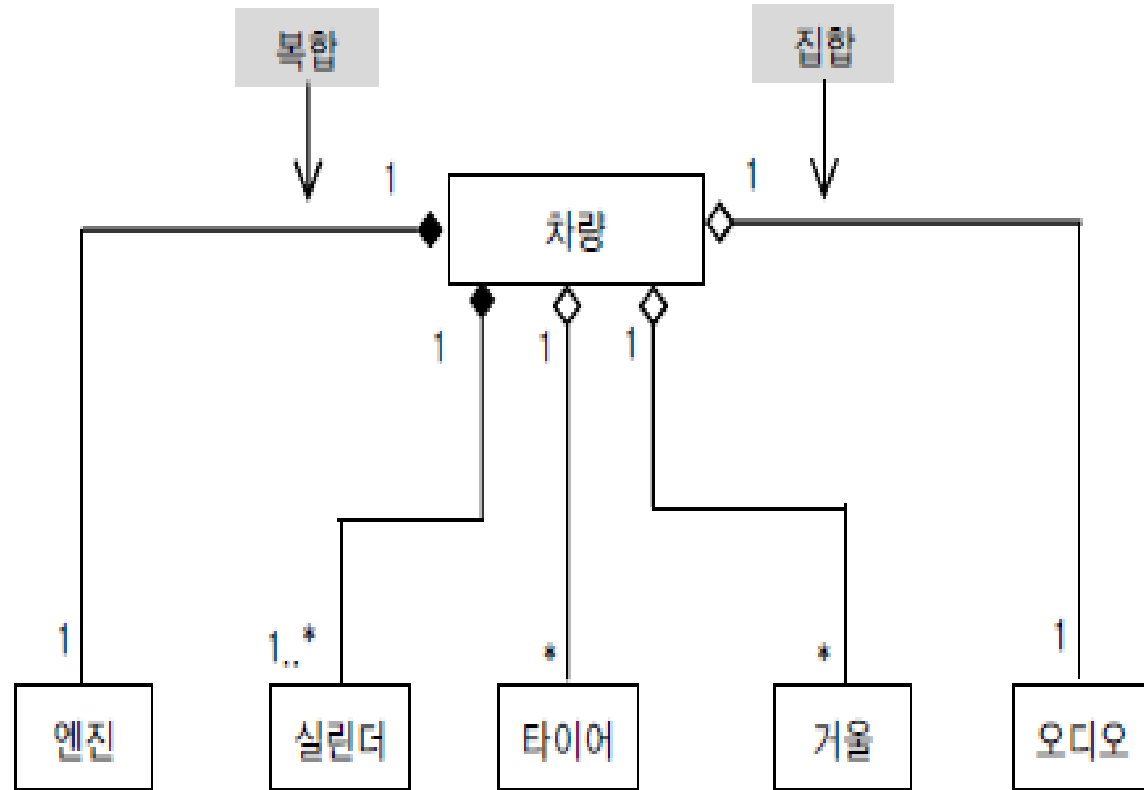
- 특별한 형태의 연관 관계(지속적), 전체와 부분과의 관계를 명확하게 명시하고자할 때 사용
- 여러 부속 객체(부품)들이 조립되어 하나의 객체가 구성되는 것

■ 집합(aggregation) 관계

- 전체가 없어져도 독립적으로 존재, 예: 데스크탑 컴퓨터
- 전체 객체와 부분 객체의 생명주기가 다르다. 부분 객체를 여러 전체 객체가 공유할 수 있다.

■ 합성(composition) 관계

- 전체가 없어지면 같이 없어지는 존재, 예 : 노트북 컴퓨터
- 전체 객체가 없어지면 부분 객체도 없어진다. 부분 객체를 여러 전체 객체가 공유할 수 없다.



합성 관계
엔진, 실린더 객체의 생성(소멸)은 차량 객체의 생성(소멸)과 관련된다.
(차량 객체 안에서 엔진과 실린더 생성)

```

class Vehicle {
    Engine e;
    Collection<Cylinder> c;
    .....
    Collection<Tire> t;
    Collection<Mirror> m;
    Audio a;
}
  
```

집합 관계
타이어, 거울, 오디오 객체의 생성(소멸)은 차량 객체와 관련이 없다.
(차량의 메소드가 아닌 다른 곳에서 타이어, 거울, 오디오 객체 생성)

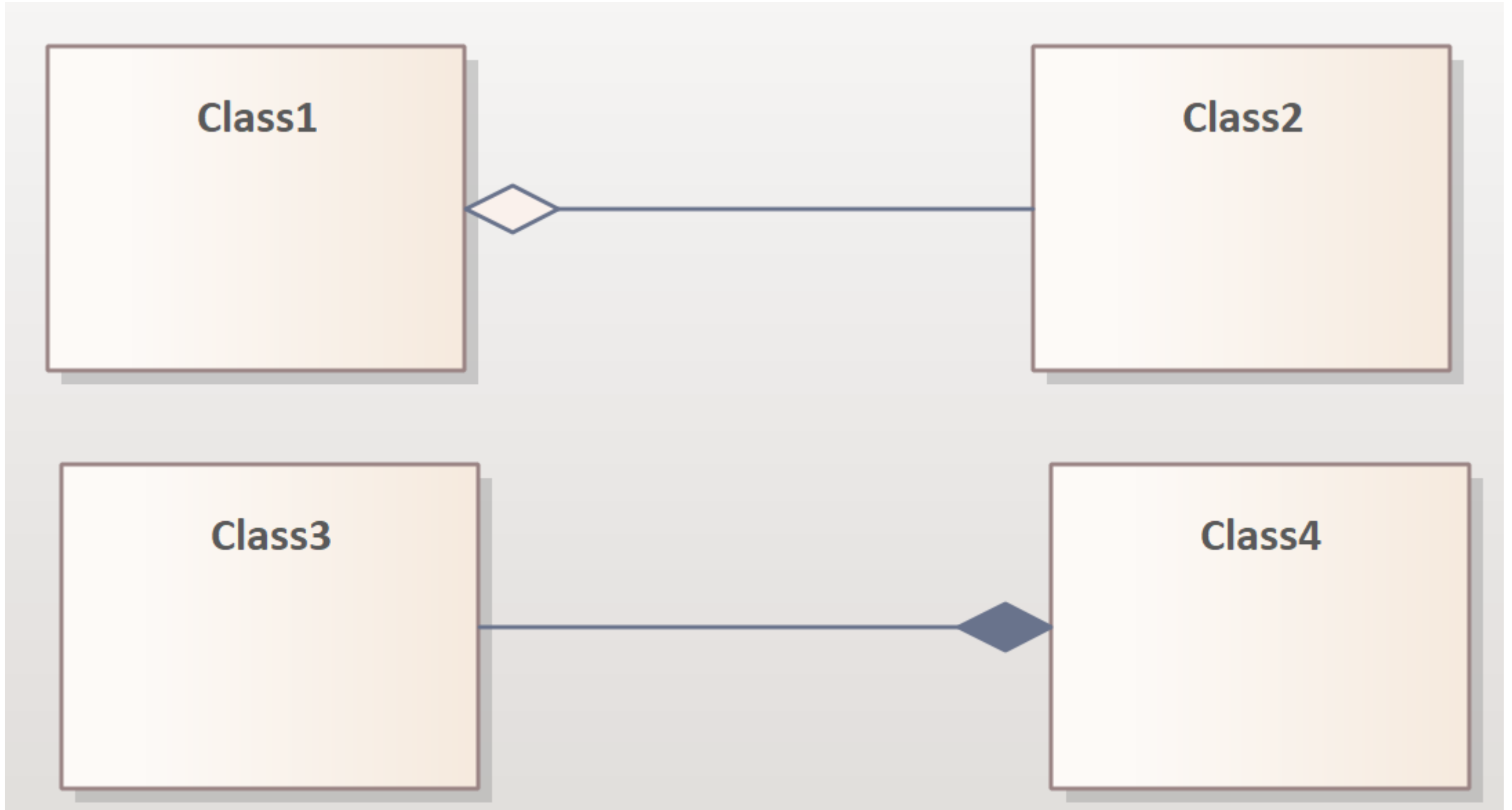
```
public class Notebook {  
    private Disk d;  
    private CPU c;  
    private MainMemory m;  
    private Modem md;  
  
    public Notebook() {  
        this.d=new Disk();  
        this.c=new CPU();  
        this.m=new MainMemory();  
        this.md=new Modem();  
    }  
}
```

NoteBook 객체가 사라지면, NoteBook 객체를 구성하는 Disk, CPU, MainMemory, Modem 객체도 죽는다
➔ 합성 관계이다

```
public class Desktop {  
    private Disk d;  
    private CPU c;  
    private MainMemory m;  
    private Modem md;  
  
    public Desktop(Disk d, CPU c, MainMemory m, Modem md) {  
        this.mb=mb;  
        this.c=c;  
        this.m=m;  
        this.ps=ps;  
    }  
}
```

Desktop 객체가 사라져도, Desktop 객체를 구성하는 Disk, CPU, MainMemory, Modem 객체는 남는다
➔ 집합 관계이다

집합(aggregation)/합성(composition) 선택하여 선연결





 **T h a n k y o u**

TECHNOLOGY

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Velit ex
plicabo ipsum, labore sed tempora ratione asperiores des
quenerat bore sed tempora rati jgert one bore sed tem!