

# UML과 순서도

편집: 홍익대학교  
김혜영

# 1.1 개 요

- **순서도(flowchart)**

- 순서도란 어떤 업무를 처리하는 과정을 순서적으로 나타내는 것으로서 필요한 데이터나 제품 등 이동 과정도 함께 나타낼 수 있는 도표를 의미.
- 반드시 컴퓨터의 이용을 전제로 하는 것은 아님.
- 컴퓨터의 이용을 전제로 한 경우에 제기된 문제를 분석하고, 문제 해결을 위한 순서와 방법을 나타내는 순서도 작성 방법에 대하여 고찰

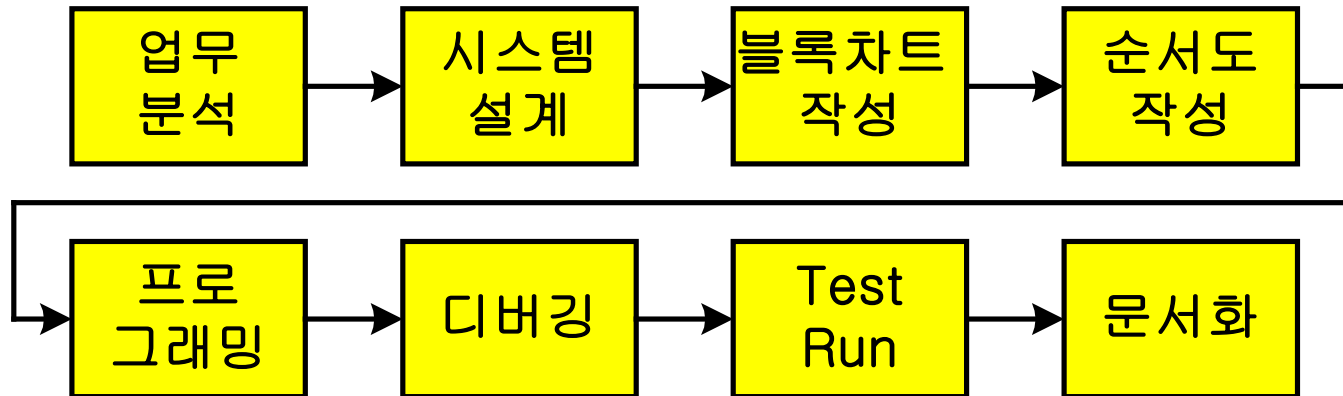
- **컴퓨터 프로그램(computer program)**

- 컴퓨터가 이해할 수 있는 정해진 규칙에 따라 표현하여야 하며, 이러한 규칙을 프로그램 언어(program language)라 한다.

- **프로그래밍(programming)**

- 컴퓨터로 어떤 일을 처리하기 위하여 프로그램 언어로 프로그램을 작성하는 것
- 프로그램을 작성하는 사람을 프로그래머(programmer)라 한다.

# 1.2 프로그램 작성 단계



## • 업무 분석

- 처리하려는 업무를 분석하여 컴퓨터로 처리하는 것이 타당한지를 검토하며, 문제와 처리의 범위를 결정한다. 업무 분석 시에는 그 업무를 부분적인 면 뿐 아니라, 전체적인 요소를 잘 분석해서 부분과 전체와의 관계를 항상 고려해야 한다. 또한 입력과 출력 매체를 결정하고 입출력 데이터의 형식을 결정한다. 그리고 어떠한 방법으로 처리하는 것이 경제적이고 능률적인가를 고려하여 적용하는 기법을 결정하며 처리 순서를 정한다.

# 1.2 프로그램 작성 단계(Cont'd)

## • 시스템 설계

- 업무의 분석이 끝나면 여러 단계를 통하여 수 작업으로 처리하고 있는 작업을 컴퓨터로 처리할 방안을 설계해야 한다.

## • 블록 차트 작성

- 프로그램은 여러 개의 명령들이 모인 것이므로 문제 처리를 위하여 논리를 세우고 이를 보다 쉽게 파악하고 검토할 수 있도록 표준 기호를 사용하여 그림으로 나타낸다. 블록 차트(block chart)는 업무의 처리 순서를 크게 나누어 처리의 흐름을 개략적인 도표로 표시한 것이며 순서도를 작성하기 위한 기초 자료로도 이용되고 시스템의 설계자와 현 업무 담당자와의 대화용으로도 사용된다.

# 1.2 프로그램 작성 단계(Cont'd)

## • 순서도 작성

- 순서도는 업무의 처리 내용을 컴퓨터의 사고 체계에 의하여 세밀한 요소까지 분석하고 각 요소간의 관계를 도표로서 표시한 것이다.
- 프로그램 논리는 대부분 순서도에서 검토하고 개선하며 순서도에 따라 프로그램 언어를 사용하여 원시 프로그램을 작성할 수 있다.
- 그러므로 프로그램 논리는 순서도에서 충분히 검토하여 완전하고 효율이 좋은 흐름이 되었을 때 프로그램을 입력한다.

## • 프로그래밍(programming)

- 프로그램 논리가 순서도로 표현되면 프로그래밍 언어를 사용하여 명령어로 표현한다. 적절한 언어를 선택하여 그 언어의 문법에 준해 기술한다.
- 이때 일정한 언어로 작성하는 과정을 프로그래밍 또는 코딩이라 한다. 프로그램을 프로그래밍 언어로 표현한 것을 원시 프로그램이라 한다.

# 1.2 프로그램 작성 단계(Cont'd)

## • 수정(debugging)

- 입력된 프로그램은 언어 번역 프로그램에 의하여 기계어로 번역.
- 이 때 문법적으로 잘못된 부분이 있으면 그 내용을 인쇄 장치나 화면에 출력.
- 프로그램이 일단 완성되면 작성된 프로그램이 정확한가 또는 논리적으로 잘못 작성된 부분이 있는가를 테스트하여야 하는데, 이와 같이 프로그램의 정확성을 테스트하는 과정을 디버깅(debugging)이라 한다.
- 보통 디버깅은 테스트 상의 체크, 기계를 사용하는 테스트, 실제 데이터를 사용하는 테스트로 구분된다.

## • Test Run

- 디버깅이 완료되고 프로그램 상에 이상이 없게 되면 모의 데이터를 사용하여 프로그램의 이상 여부를 최종 테스트한다.
- Test Run의 결과 올바른 결과를 얻지 못하면 다시 디버깅 과정을 반복하게 되며, Test Run이 완전히 끝난 다음에 실제의 Run에 착수한다. 이와 같이 하여 완전한 프로그램이 마련되면 실제 데이터를 처리하게 되는데 이 과정을 러닝(running)이라 한다.

# 1.2 프로그램 작성 단계(Cont'd)

- 문서화(documentation)

- 문서화는 반드시 필요하며 프로그램을 가장 빨리 파악할 수 있도록 작성
- 담당자가 변경되어도 쉽게 파악할 수 있도록 상세하게 작성되어야 한다.

# 1.3 순서도의 개념

- 알고리즘(algorithm)

- 어떤 문제를 해결하려고 할 때 존재하는 규칙들을 나열하여 수학에서의 용어를 컴퓨터의 용어로서 사용하는 것.
- 즉, 알고리즘이란 어떤 주어진 문제를 유한 단계 내에서 어떤 해를 구할 수 있게끔 나타낸 명확한 단계의 연속을 의미.

- 알고리즘 작성기법

- 하향식 설계(top-down design) 기법 :
- 상향식 설계(bottom-up) 기법 :

- 순서도

- 처리하고자 하는 문제를 분석하여 그 처리 순서를 단계화시켜 상호간의 관계를 일정한 기호를 사용하여 일목요연하게 나타낸 그림



# 1.3 순서도의 개념 (Cont'd)

- 순서도 작성으로 인하여 얻을 수 있는 효과

- ① 문제 처리의 과정을 논리적으로 파악할 수 있어 정확성 여부에 대한 쉬운 판단 가능.
- ② 문제를 해석하고 분석하며, 타인에게 전달하는 것이 용이.
- ③ 프로그램의 보관, 보수, 유지의 자료로서 활용이 가능.
- ④ 프로그램 코딩의 기본 자료로서 활용 가능 .

# 1.4 순서도의 기호

- 순서도 기호의 종류

- 기본 기호(basic symbol)
- 프로그래밍 관련 기호(symbols related to programming)
- 시스템 관련 기호(symbols related to systems)


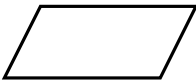
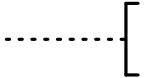

- 순서도의 일반적인 작성 규칙

- ① 기호의 내부에 처리할 내용을 기입한다.
- ② 흐름도의 방향을 위에서 아래로, 좌에서 우로 하는 것을 원칙으로 하되 반대일 경우에는 반드시 화살표를 사용한다.
- ③ 흐름선은 교차시켜도 아무런 논리적 관계를 갖지 않는다.
- ④ 흐름선 두 개 이상이 모여 한 줄이 될 수 있다.
- ⑤ 기호의 모형상 가로, 세로의 비율은 정확하게 정해지지는 않았으나, 잘 구분될 수 있도록 해야 한다.

# 1.4 순서도의 기호 (Cont'd)



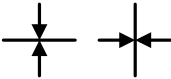

## □ 기본 기호(basic symbol)

◎ 시스템 순서도와 프로그램 양 쪽에서 공동으로 사용되는 기본적인 기호

기 호	이 름	용 도
	처리 (process)	지정된 동작, 각종 연산, 값이나 기억 장소의 변화, 데이터의 이동 등의 모든 처리 과정을 나타낸다.
	입출력 (input/output)	일반적인 입력과 출력 처리를 나타낸다.
	주해 (comment annotation)	표현된 기호를 보다 구체적으로 설명하며, 점선은 해당 기호까지 연결한다.
	연결자 (connector)	흐름이 다른 곳으로 연결됨과 다른 곳에서 연결되는 입구를 나타내며, 화살표와 기호 내에 쓰여진 이름의 연결 관계를 나타낸다.

# 1.4 순서도의 기호 (Cont'd)

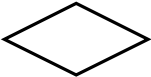
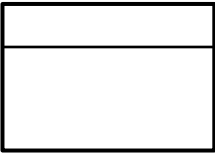


## □ 기본 기호(basic symbol)

기 호	이 름	용 도
	페이지 연결자 (offpage connector)	흐름이 다른 페이지로 연결됨과 다른 페이지에서 부터 연결되는 입구를 나타내며, 연결자와 같은 방법으로 사용한다.
	흐름선 (flow line)	처리의 흐름을 나타내며, 선이 연결되는 순서대 로 진행됨을 나타낸다.
		여러 개의 흐름이 한 곳으로 모여서 하나가 됨을 나타낸다.
	출력	일반적인 프린터 출력 처리를 나타낸다

# 1.4 순서도의 기호 (Cont'd)

## □ 프로그래밍 관련 기호

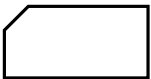
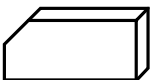
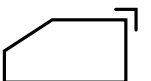
◎프로그램의 논리를 나타내는 기호로서 기본 기호와 함께 프로그램 순서도를 작성하는 경우에 사용














기 호	이 름	용 도
	의사 결정 (decision)	주어진 조건에 따라 비교하여 해당되는 조건을 찾아서 흐름이 결정되게 하며, 왼쪽이나 오른쪽 또는 아래로 흐름이 분기하게 된다.
	함수문 (서브—루틴)	부 프로그램처리
	준비 (preparation)	기억 장소의 할당, 초기값 설정, 설정된 스위치의 변화, 인덱스 레지스터의 변화, 순환 처리를 위한 준비 등을 나타낸다.
	단자 (terminator)	프로그램 순서도의 시작과 끝을 나타낸다.

# 1.4 순서도의 기호 (Cont'd)

## □ 시스템 관련 기호

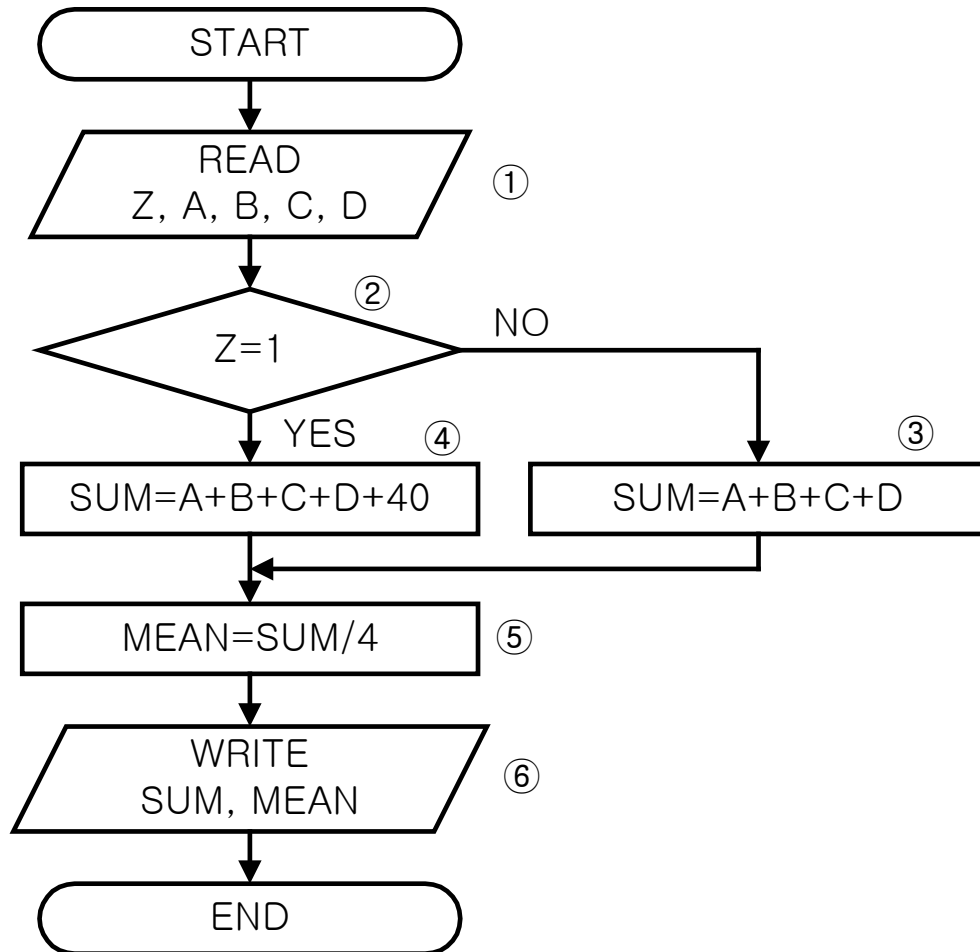
◎시스템을 분석하고 설계할 때, 데이터가 어느 매체에서부터 처리되어 어느 매체로 변환되어 해당되는 곳으로 흐르는지를 나타내는 경우에 사용되는 기호들로서 기본 기호와 함께 시스템 순서도를 작성하는 데에 사용

기 호	이 름	용 도
	펀치 카드 (punched card)	펀치 카드 매체에 의한 입력과 출력을 나타낸다.
	카드 뭉치 (card deck)	펀치 카드가 모여 있음을 나타낸다.
	카드 파일 (card file)	펀치 카드에 레코드가 모여서 파일을 구성하고 있음을 나타낸다.
	서류 (document)	각종 원시 데이터가 기록된 서류나 종이 매체에 출력되는 결과 및 문서화된 각종 서류를 나타낸다.

기 호	이 름	의 미
	터미널	순서도의 시작과 끝을 표시
	준 비	배열선언, 초기설정 등에 사용
	흐름선(flow-line)	순서도 기호간 연결 및 흐름을 표시
	반복(Loop)	반복 수행
	수작업	off-line 이나 키보드로 작업
	통신	통신회선으로 접속
	입 력	데이터의 입 력 시 사용
	비교, 판단	조건에 의한 비교, 판단 후 분기를 할 경우에 사용
	결합	같은 페이지에서 순서도 흐름을 연결
	서브루틴	부 프로그램 처리
	페이지 결합	순서도 흐름이 다른 페이지로 연결될 경우 사용
	주석	주석이나 설명을 표시
	출 력	출력

## 2.2 선택형 (Cont'd)

### • 선택형의 예



① 입력 장치를 통하여 Z, A, B, C, D의 값을 읽어 들여 주기억장치에 기억

② Z의 값이 1일 때는 군필자이기 때문에 총점에 40점을 가산시키고, 그렇지 않을 경우는 군 미필자이므로 자기의 시험점수만을 가지고 총점을 산출

③ 군 미필자이므로 네 과목(A, B, C, D)에서 얻는 점수를 합한 결과를 SUM에 기억

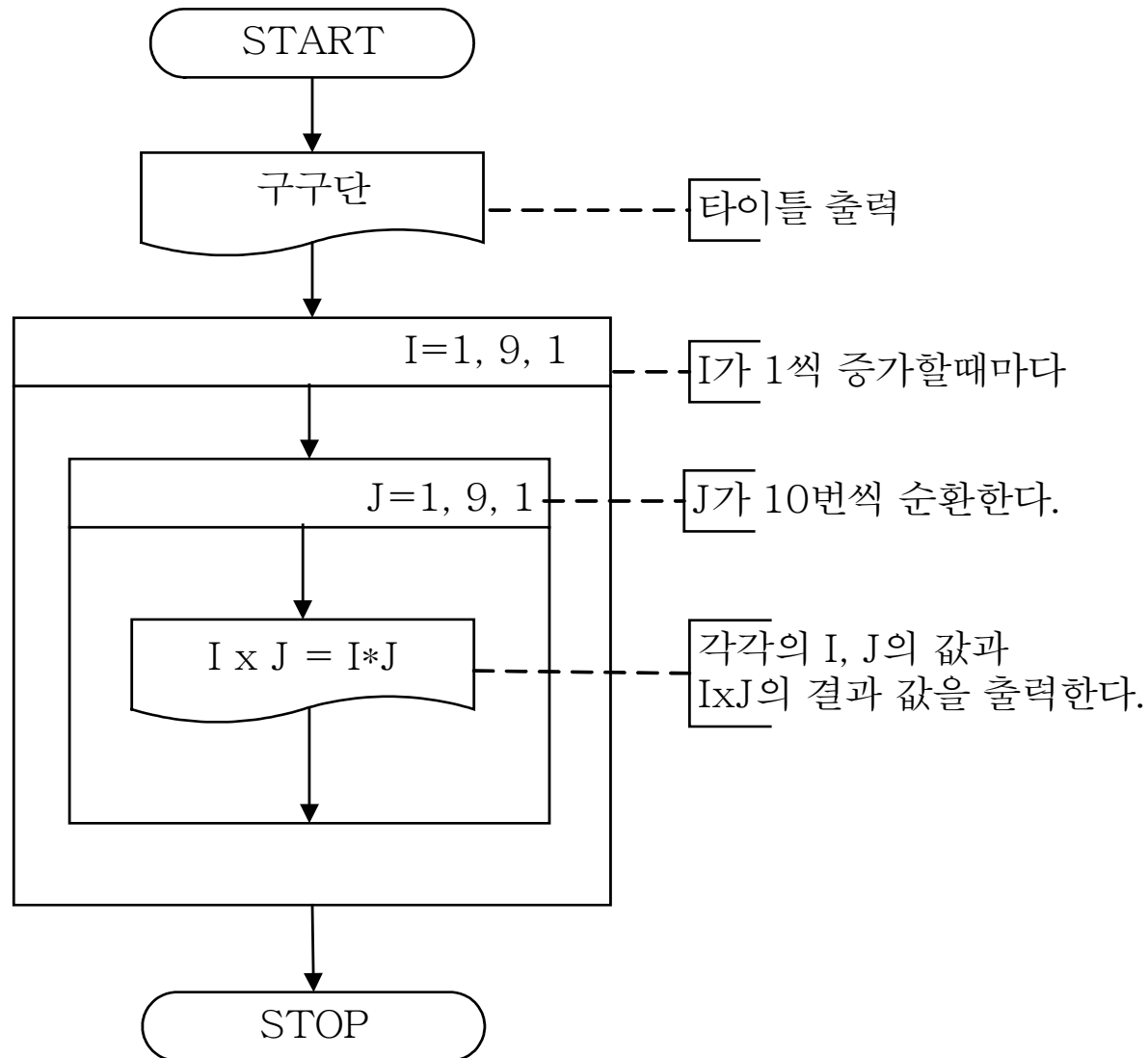
④ 군필자에 총점을 구하는 과정으로 실기 시험에서 얻은 점수에 40점을 가산하여 SUM에 기억

⑤ 평균을 구하여 MEAN에 기억

⑥ 총점(SUM)과 평균(MEAN)을 출력 장치를 통하여 인쇄.



# 순서도 6.1 구구단을 출력하는 순서도



## 8. 공백을 이용한 문자 출력하기

- 다음의 결과를 위해 문자 "\*"를 출력하기 전에 공백을 출력
- 즉, 한 줄씩 내려갈 때마다 공백이 같이 증가하면서 출력  
두 번째 줄은 한 칸, 세 번째는 두 칸을 띄우면서 문자 출력
- 이는 하나의 반복문 안에 두개의 반복문이 존재하는 경우

\*\*\*\*\*

\*\*\*\*\*

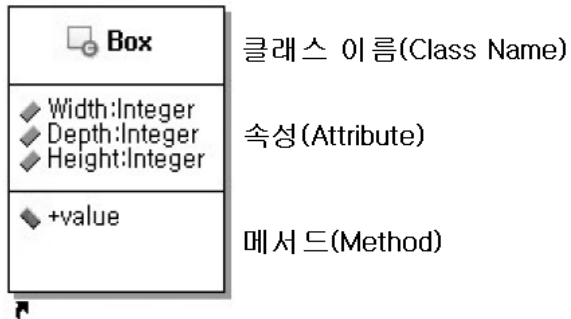
\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

## 클래스의 구성 요소

- 클래스 구성 요소 : 클래스 이름, 속성, 메서드



[표 4-1] 메서드의 종류와 기호

메서드의 종류	부호	내용
public	+	자신의 속성이나 동작을 외부에 공개하는 접근 제어
private	-	상속된 파생 클래스만이 액세스할 수 있는 접근 제어
protected	#	구조체의 멤버 함수만 접근할 수 있으며 외부에서 액세스할 수 없는 접근 제어

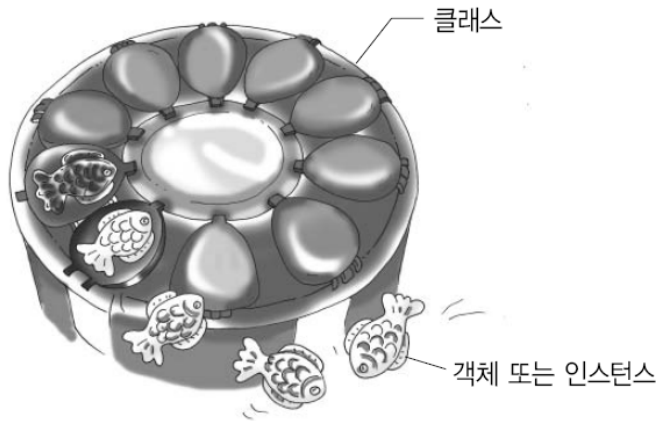
[그림 4-1] 클래스 구성 요소

[코드 4-1] [그림 4-1]에 대한 자바 코드

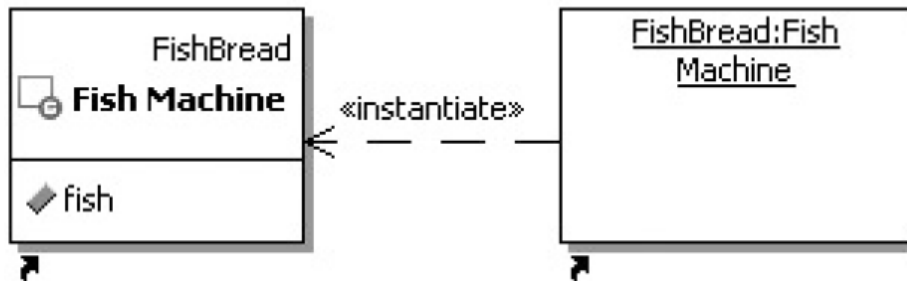
```
01 // class name
02 Box{
03     // attribute
04     private int width;
05     private int depth;
06     private int height;
07     // method
08     void value(){
09     }
10 }
```

- 클래스(class)
  - ✓ 공통의 속성, 메서드(오퍼레이션), 관계, 의미를 공유하는 객체들의 집합
- 속성(attribute)
  - ✓ 클래스의 구조적 특성에 이름을 붙인 것으로 특성에 해당하는 인스턴스가 보유할 수 있는 값의 범위를 기술
  - ✓ 속성은 영문자 소문자로 시작
- 메서드(method)
  - ✓ 오퍼레이션이라고도 하며,
  - ✓ 이름, 타입, 매개변수들과 연관된 행위를 호출하는데 요구되는 제약 사항들을 명세하는 클래스의 행위적 특징

- 객체 : 실제 현실에서 존재하는 사물을 의미
- 클래스 : 객체들을 추상화한 개념

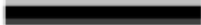


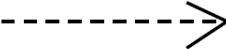
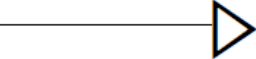
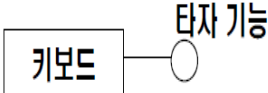



[그림 4-2] 붕어빵 기계와 붕어빵(클래스와 객체의 관계)



[그림 4-3] 클래스와 객체와의 관계

## 나. Class Diagram의 Class 간의 관계

관계 유형	설명	표기법
연관관계 (Association Relationship)	두 클래스 간에 서로 어떠한 연관을 가지고 있는 의미 표기법 : 1(1개), 0..1(0개 또는 1개), *(다수), 1..*(1개 또는 다수)	 가장 일반적 'Has a' 관계
집합연관관계 (Aggregation Relationship)	클래스와 클래스 간의 부분과 전체의 관계를 의미	 부서 - 직원
복합연산관계 (Composition Relationship)	집합연관관계와 같이 부분과 전체의 관계이나 전체 클래스 소멸 시 부분 클래스도 같이 소멸하는 관계	 테이블 - 다리
의존관계 (Dependency Relationship)	한 클래스의 변화가 다른 클래스에 영향을 미치는 관계	
상속관계 (Generalization /Inheritance)	클래스간 상속관계, 객체지향의 상속관계를 의미하고 일반화를 의미	
인터페이스	실체화(Realization)를 인터페이스로 표현 하나의 객체(추상객체)가 다른 객체(구상객체)에 의해 오퍼레이션을 실체화하는 관계	
직접 연관	화살표가 의미하는 방향성은 참조하는 쪽과 참조 당하는 쪽을 구분하는 연관관계	

- **예제** : 다음의 다이어그램은 하나의 계좌에 입금되는 클래스(Account)와 객체를 생성하여 실행하는 메인 메서드를 포함하는 클래스(Application)로 구성

[코드 4-2] 객체 생성의 자바 코드

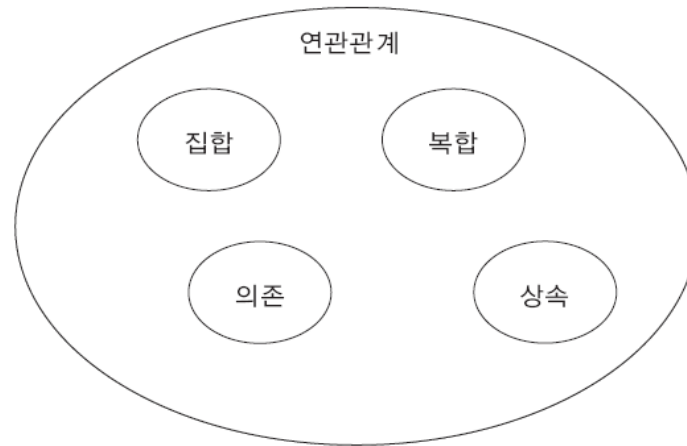
```

01 class Account
02 {
03     int balance;
04     int deposit(int amount){
05         balance = balance + amount;
06         return (balance);
07     }
08 }
09
10 class Application{
11
12     public static void main(String args[])
13     {
14         Account account1;
15         // 객체 생성
16         account1 = new Account();
17         account1.balance = 5000;
18         account1.deposit(5000);
19         System.out.println("Balance = " +account1.balance);
20     }
21 }

```



[그림 4-4] 연관관계의 종류



[그림 4-5] 연관관계의 종류

## ● 연관관계

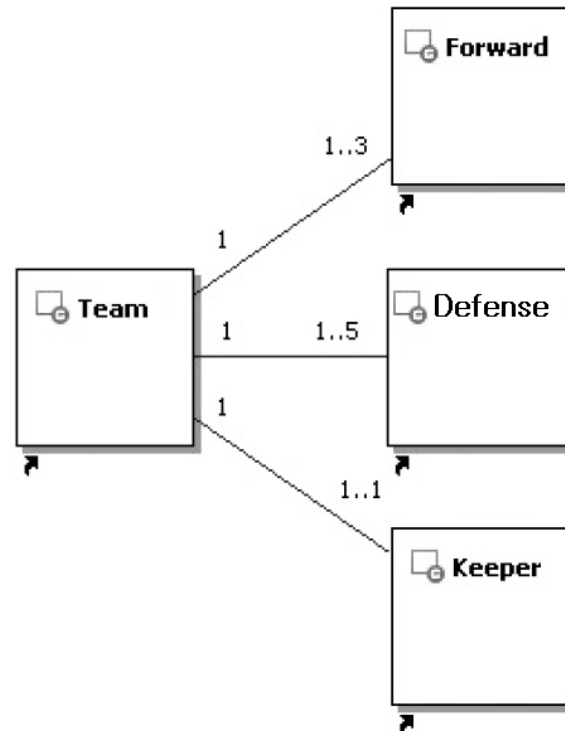
- 연관관계는 클래스가 서로 개념적으로 연결된 선을 의미
- 클래스 다이어그램에서 연관관계는
  - ✓ 서로 개념적으로 연관없는 클래스는 없으므로 의존, 상속, 집합, 복합 등의 관계와 같이 표시하는 건 별로 중요하지 않다.
- '연관관계'에 대한 내용은 DB 설계의 ER-D의 내용과 연관지어 생각하면 쉽다.
- **예제** : 축구팀과 선수와의 관계



[그림 4-7] 팀과 선수의 연관관계



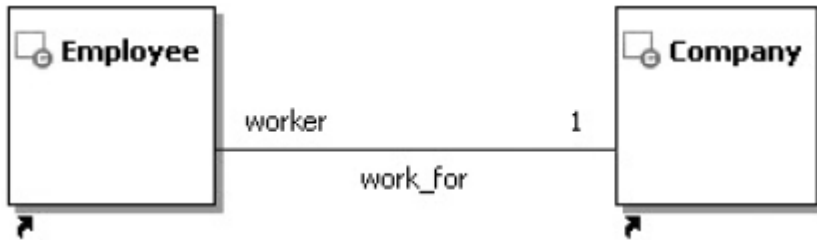
[그림 4-8] 선수와 구단의 연관관계



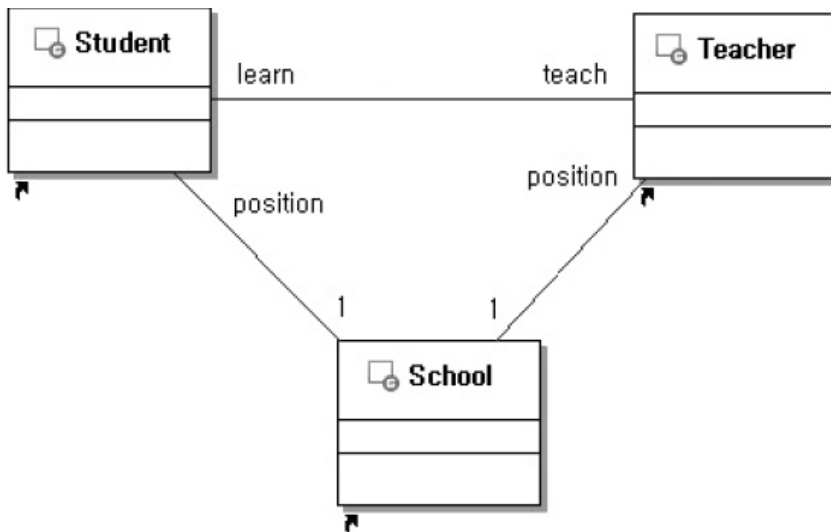
[그림 4-9] 하나의 클래스와 여러 클래스와의 연관관계



## ▶ 연관관계의 예들



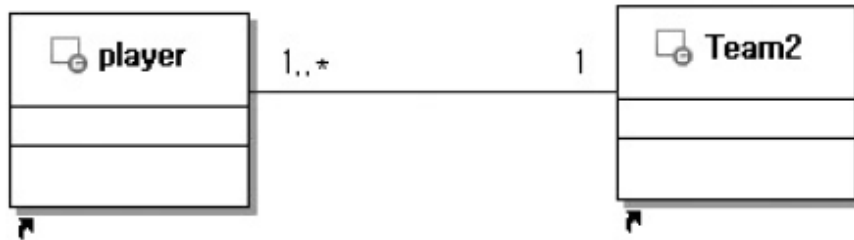
[그림 4-10] 사원과 회사 사이의 연관관계



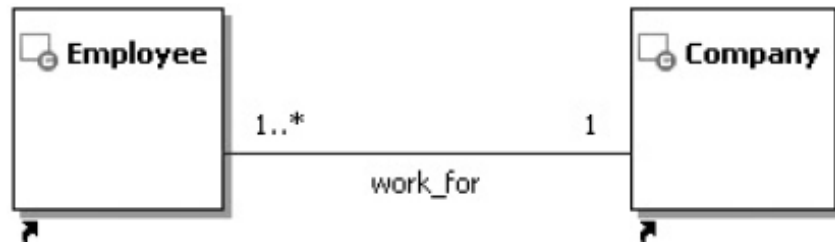
[그림 4-11] 학생, 교사, 학교 사이의 연관관계

## ● 연관관계의 다중성

- 다중성 : 두 클래스의 연관관계에서 실제로 연관을 가지는 객체의 수를 나타낸 것



[그림 4-12] 선수와 팀의 다중성



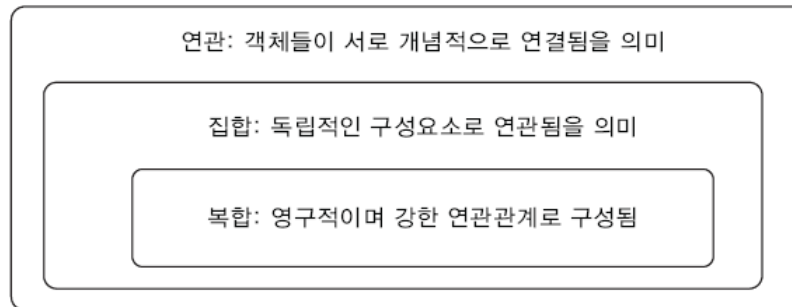
[그림 4-13] 회사와 사원의 다중성 관계

다중성 표현	의미
1	한 객체와 연관된다. 표시하지 않아도 되는 기본값이다.
0..1	0개 또는 1개의 객체와 연관된다.
0..*	0개 또는 많은 수의 객체가 연관됨을 나타낸다.
*	0..*와 동일하다
1..*	1개 이상의 객체와 연관된다.
1..12	1개에서 12개까지의 객체가 연관됨을 나타낸다.
1..2, 4, 11	1개에서 2개까지 또는 4개 또는 11개의 객체가 연관됨을 나타낸다.

[표 4-2] 다중성의 표현

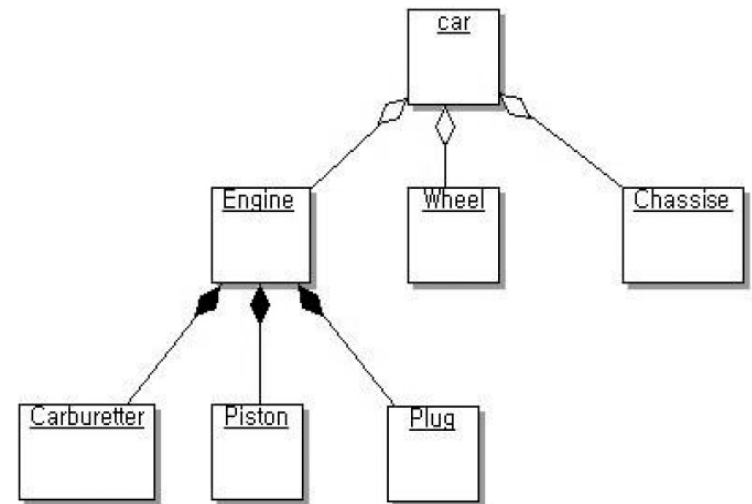
## ● 포함(집합과 복합)관계

- 집합(Aggregation)관계와 복합(Composition)관계 모두 연관관계에 포함되는 개념



[그림 4-14] 연관관계와 그 중 집합과 복합의 개념

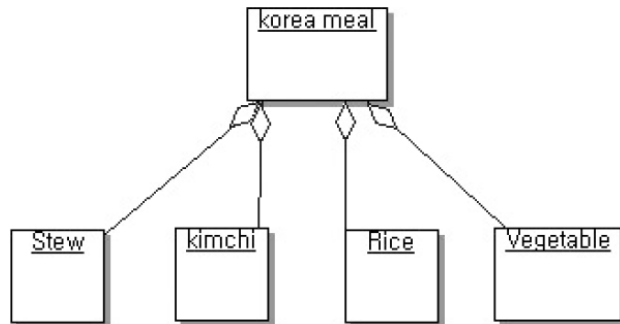
- 집합관계
  - ✓ 하나의 객체에 여러 개의 독립적인 객체들이 구성되는 경우
- 복합관계
  - ✓ 더 강한 관계로 구성
  - ✓ 엔진은 카뷰레터, 피스톤, 플러그
  - ✓ 엔진의 구성 요소는 더 강한 관계



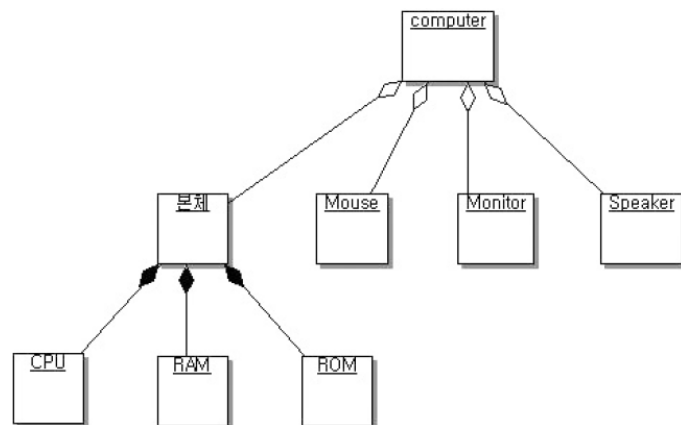
[그림 4-15] 차와 엔진, 바퀴, 차체(집합관계)  
엔진과 카뷰레터, 피스톤, 플러그(복합관계)

## ▶ 집합과 복합 예제

- 식사가 밥, 찌개, 김치, 나물 등으로 구성되어 있을 경우, 이들은 식사에 대한 구성 요소이기때문에 **집합관계**
- 컴퓨터도 마찬가지로 그 구성 요소로 이루어지면 **집합관계**
- 컴퓨터 본체는 여러 가지의 구성 요소가 존재하는데 이는 영구적인 요소이기 때문에 **복합관계**



[그림 4-16] 식사 : 밥, 찌개, 김치, 나물

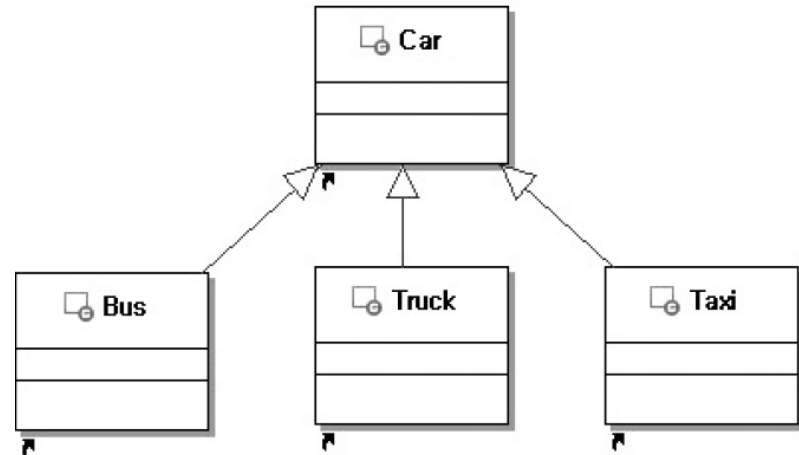


[그림 4-17] 컴퓨터와 모니터, 마우스, 키보드, 스피커(집합관계).  
본체와 CPU, ROM, RAM(복합관계)

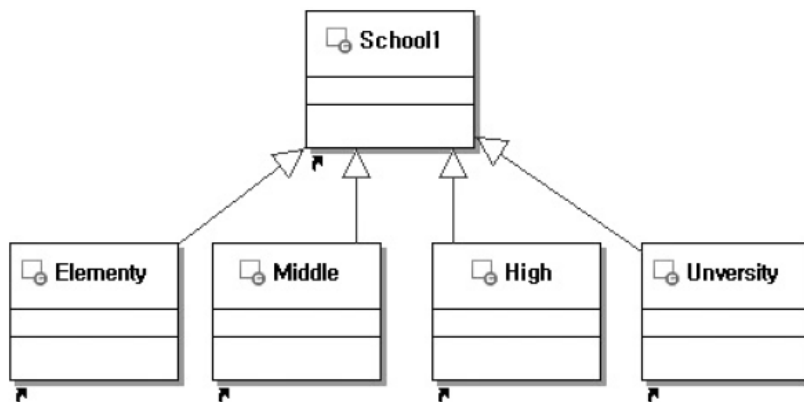
## 상속관계

- a\_kind\_of의 관계

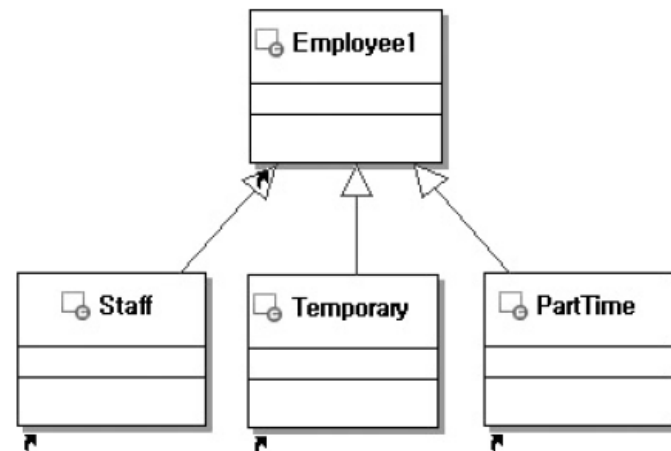
[그림 4-18] 차와 버스, 트럭, 택시(일반화관계)



- 상속관계라고도 한다.

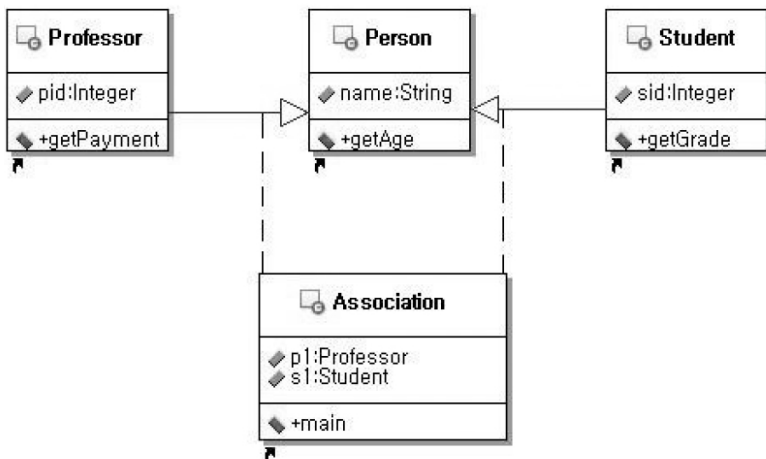


[그림 4-19] 학생과 초등학생, 중학생, 고등학생, 대학생(일반화관계)



[그림 4-20] 사원과 정사원, 계약사원, 아르바이트생(일반화관계)

## ▶ [예제 4-1] 사람과 교수, 학생 간의 상속과 연관관계



[그림 4-21] 사람과 교수, 학생 간의 일반화관계

[코드 4-3] 일반화관계 지바 코드

```

01 class Person
02 {
03     String name;
04     int getAge()
05     {
06         return 49;
07     }
08 }
09
10 class Student extends Person
11 {
12     int sid;
13     int getGrade(){
14         return sid-200400;
15     }

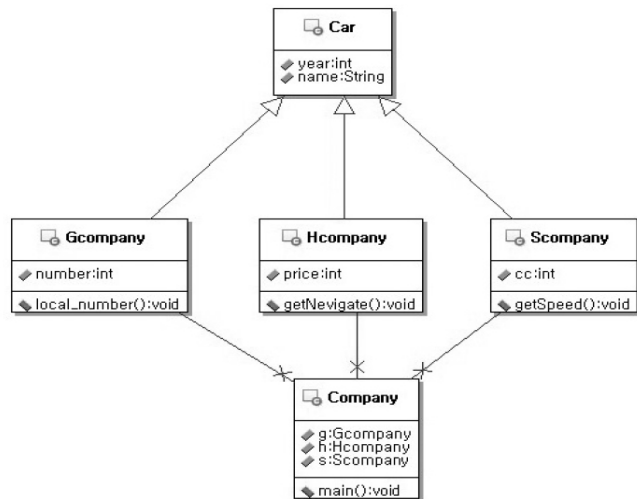
```

```

16 }
17
18 // Person으로 상속받기 때문에 name, getAge(), pid, getPayment()로 구성
19 class Professor extends Person
20 {
21     int pid;
22     int getPayment()
23     {
24         return pid+10000;
25     }
26 }
27
28 class Inheritance
29 {
30     public static void main(String[] args) {
31         Student s1 = new Student();
32         Professor p1 = new Professor();
33         s1.name = "홍길동";
34         s1.sid = 200401;
35         System.out.println("Student name : "+s1.name+ "Student
36                             ID : "+s1.sid);
37         System.out.println("Student Age : "+s1.getAge()+
38                             "Student Grade : "+s1.getGrade());
39         p1.name = "홍교수";
40         p1.pid = 1016;
41         System.out.println("Professor name : "+p1.name+
42                             "Professor ID : "+p1.pid);
43         System.out.println("Professor Age : "+p1.getAge()+
44                             "Professor Payment : "+p1.getPayment());
45     }
46 }

```

## ▶ [예제 4-2] 자동차와 자동차 제품 간 상속 및 연관관계



[그림 4-22] 자동차와  
자동차 제품 간 일반화관계

[코드 4-4] 일반화(상속)관계 자바 코드

```

01 class Car
02 {
03     String name;
04     int year;
05     int getYear()
06     {
07         return year;
08     }
09 }
10 class Gcompany extends Car
11 {
12     int number;
13     int getLocal_number(){
14         return number;
15     }
16 }
17 class Hcompany extends Car
18 {
19     int price;

```

①

```

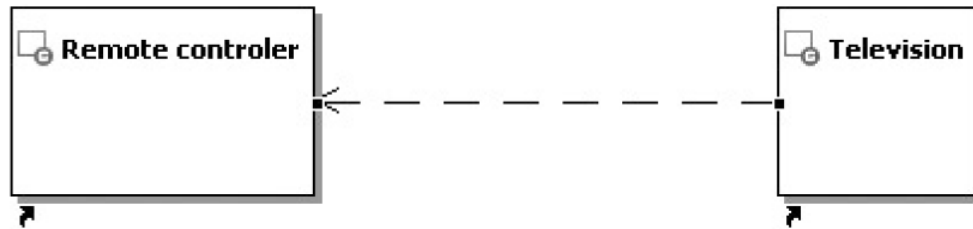
20     int getNevigate(){
21         return price-10000000;
22     }
23 }
24
25
26 class Scompany extends Car
27 {
28     int cc;
29     int getSpeed(){
30         return cc-1300;
31     }
32 }
33
34 public class Company
35 {
36     public static void main(String args[])
37     {
38         Gcompany g = new Gcompany();
39         Hcompany h = new Hcompany();
40         Scompany s = new Scompany();
41         g.name = "레강자";
42         g.number = 2001;
43         System.out.println("Car name: " +g.name+ "Car number: "
44                             "+g.number);
45         System.out.println("Car year: " +g.getYear()+ "Car
46                             number: " +g.getLocal_number());
47         h.name = "소나타";
48         h.price = 10000000;
49         System.out.println("Car name: " +h.name+ "Car price: "
50                             "+h.price);
51         System.out.println("Car year: " +h.getYear()+ "Car
52                             number: " +h.getNevigate());
53         s.name = "SM5";
54         s.cc = 2500;
55         System.out.println("Car name: " +s.name+ "Car price: "
56                             "+s.cc);
57         System.out.println("Car year: " +s.getYear()+ "Car
58                             number: " +s.getSpeed());
59     }
60 }

```

②

## ● 의존관계

- 의존관계 : 하나의 클래스가 또 다른 클래스를 사용하는 관계
- 다른 클래스의 멤버 함수를 사용하는 경우
- 하나의 클래스에 있는 멤버 함수의 인자가 변함에 따라 다른 클래스에 영향을 미칠 때의 관계를 의미

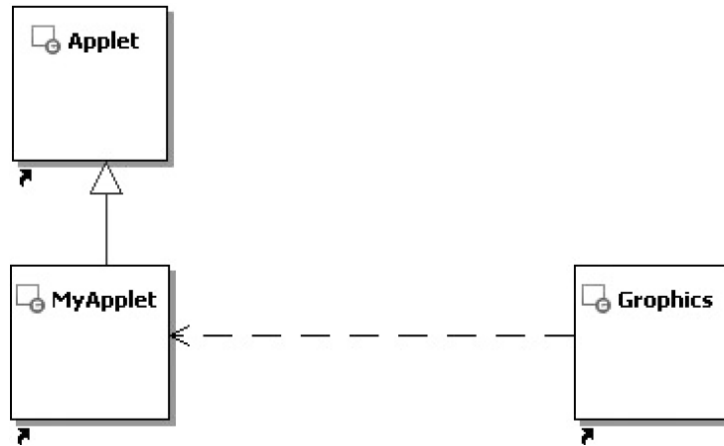


[그림 4-23] TV와 리모컨의 의존관계

- 다른 의존관계
  - ✓ 수업 → 교수
  - ✓ 전화기 → 버튼
  - ✓ 세탁기 → 손잡이
  - ✓ 자동차 → 기어



- 의존관계에 있어서 클래스 A가 클래스 B의 객체를 생성하는 경우
- 예 : 애플릿 프로그램인 MyApplet의 point( ) 메서드에서 text를 출력하기 위한 g.drawString( ) 메서드 호출할 경우



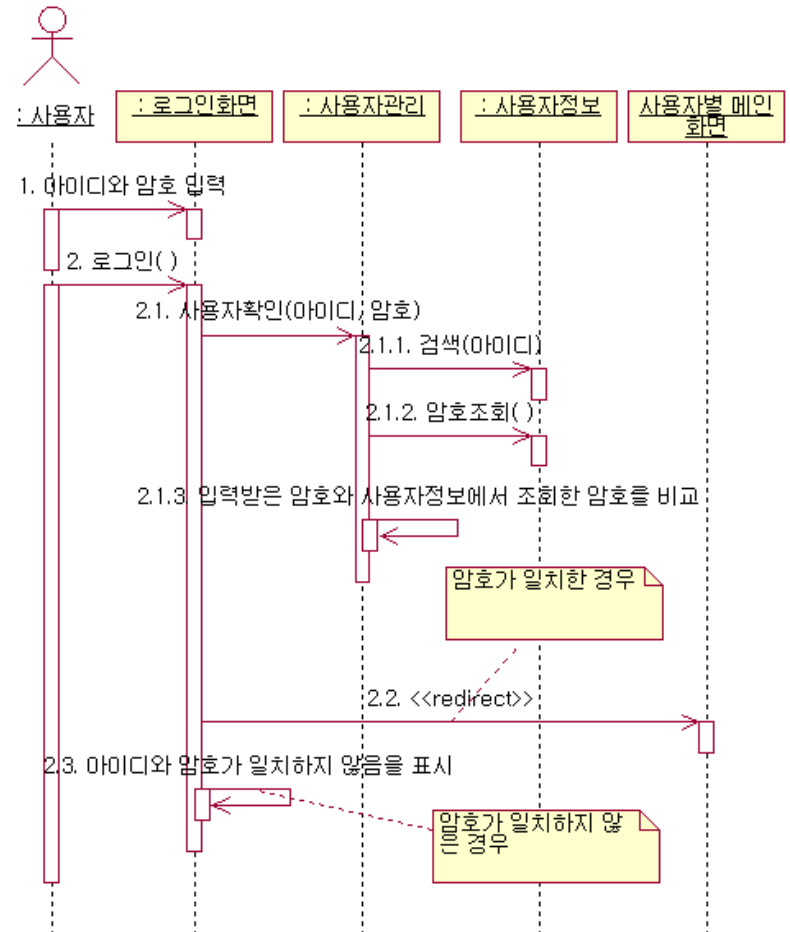
[그림 4-24] 메소드 호출의 경우(의존관계)

[코드 4-5] 애플릿 프로그램

```
01 public class MyApplet extends Applet
02 {
03     init();
04     public void start();
05     public void paint(Graphics G)
06     {
07         g.drawString("Hello Applet World",10,20);
08     }
09 }
```

# 시퀀스 다이어그램

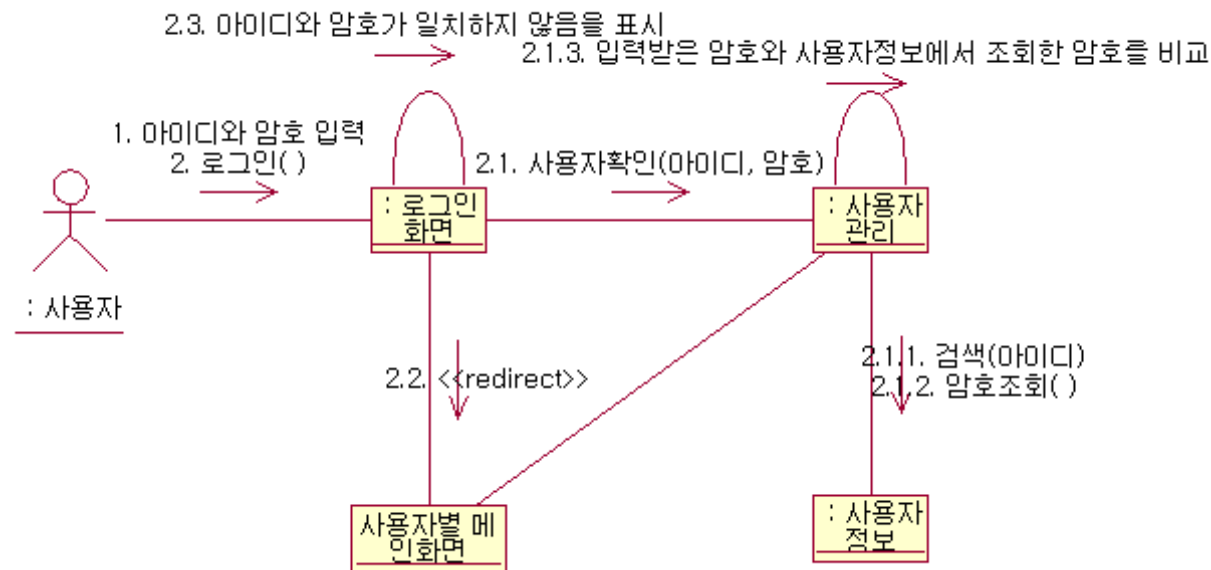
- 용도
  - 분석 및 설계 활동에서 객체들 상의 메시지 상호 작용을 표현
- 객체
  - 객체이름: 클래스이름
- 메시지
  - 메시지 이름은 메시지 수신 객체가 수행할 행동들을 의미
  - 상위에 배치된 메시지가 하위에 배치된 메시지보다 우선 전송



# 협력 다이어그램

- 용도

- 분석 및 설계 활동에서 객체들 상의 메시지 상호:



# 예제 1

- 키보드로 조정되는 움직이는 네모
  1. 사각 활동영역안의 상단에 검게 칠해진 네모가 나타나서 떨어지기 시작
  2. 키보드가 눌러지면 키보드값에 따라 한 칸씩 이동
    - 키보드값 J: 왼쪽으로 이동
    - 키보드값 I: 오른쪽으로 이동
    - 키보드값 X: 종료
  3. 움직이는 네모가 활동영역 바닥에 도착하면 상승, 천장에 도착하면 다시 하강
  4. 움직이는 네모가 활동영역 좌 우로 움직일때 좌우측 경계에 도달하면 더 이상 움직이지 않음

# 예제 2

- 간단한 채팅 (대화서비스) 프로그램
  - 대화자는 로그인 과정을 통해 대화실에 입실
  - 한 대화자가 대화실에 접속성공하면 다른 대화자들은 그 대화자의 입장 메시지를 받음
  - 대화에 참여하는 모든 대화자는 로그인 과정에서 사용한 이름으로 구분한다.
  - 대화자가 전송한 메시지는 대화실에 접속한 모든 대화자에게 보내진다.
  - 대화자는 “/bye”명령어에 의해 대화실을 퇴장
  - 한 대화자가 퇴장하면 다른 대화자는 그 대화자의 퇴장 메시지를 받는다.