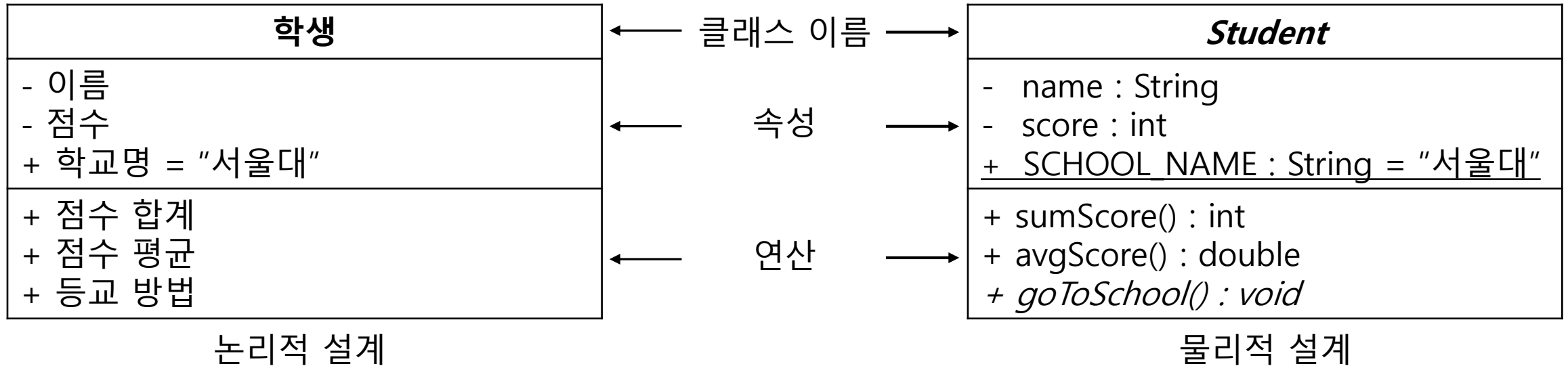


# 클래스 다이어그램

# ▶ 클래스 다이어그램

**정적(구조) 다이어그램**으로 UML모델링에서 가장 일반적으로 사용  
시스템의 구조와 구조 간 상호 관계를 나타내며  
시스템의 논리적 및 물리적 구성요소 설계 시 주로 활용

## ✓ 클래스의 표현



## ▶ 클래스 다이어그램


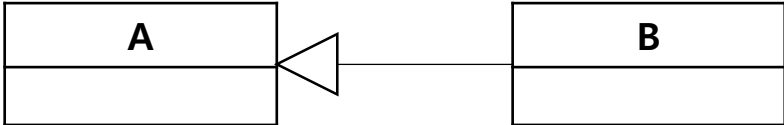
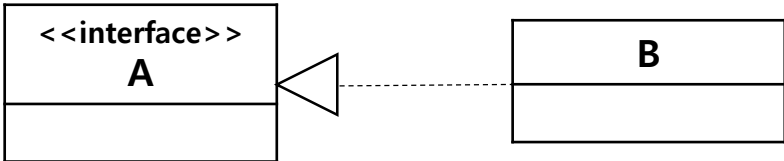

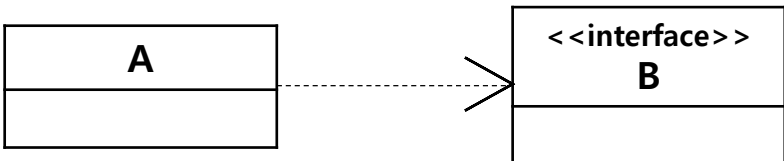
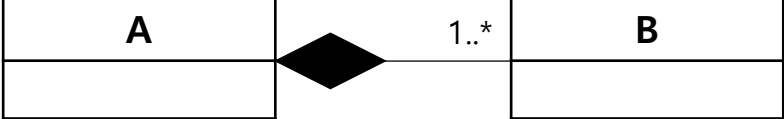
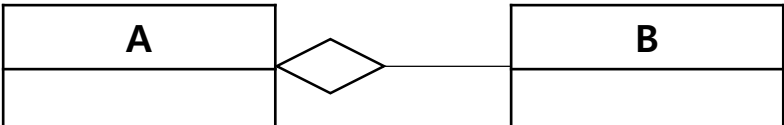
### ✓ 접근제한자

기호	예약어	적용 범위
+	public	전체
#	protected	같은 패키지 + 상속 관계
~	(default)	같은 패키지
-	private	같은 클래스

### ✓ 클래스 다이어그램 표기 방법

표현법	적용 범위	예약어
<u>field</u> / <u>method</u> (밑줄)	속성, 연산	static
FIELD (대문자)	속성	final
<i>Class</i> / <i>method</i> (기울임)	클래스 명, 연산	abstract

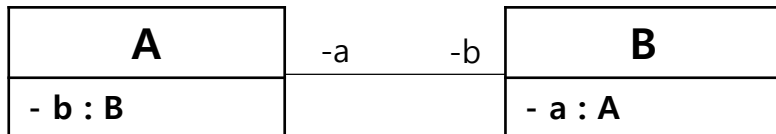
## ▶ 클래스 다이어그램의 관계

관계	표기법	의미
연관 관계		클래스 A와 클래스 B는 연결되어 있다.
일반화 관계		클래스 B는 클래스 A의 하위 클래스이다.
실체화 관계 (인터페이스 실현 관계)		클래스 B는 인터페이스 A를 실현한다.
의존 관계		클래스 A는 클래스 B에 의존한다.
인터페이스 의존 관계		클래스 A는 인터페이스 B에 의존한다.
합성관계		클래스 A는 클래스 B를 한 개 이상 포함하고 있다.
집합 관계		클래스 B는 클래스 A의 부분이다.

## ▶ 연관 관계(Association)

한 클래스가 **필드로 다른 클래스를 참조**할 때를 의미.

클래스 간의 관련성을 뜻하는 것으로 메시지 전달의 통로 역할을 함

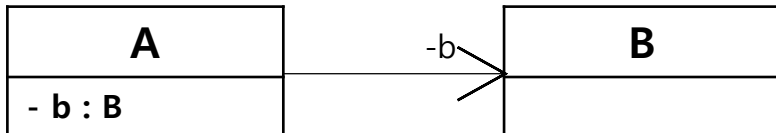


```
public class A{
    private B b;
}
```

```
public class B{
    private A a;
}
```

## ✓ 방향성이 있는 연관 관계

방향성은 메시지 전달의 방향을 뜻하며 반대 방향은 불가능



```
public class A{
    private B b;
}
```

```
public class B{
}
```

## ▶ 연관 관계

### ✓ 연관 관계의 다중성

관계를 맺을 수 있는 실제 상대 객체의 수를 다중성을 통하여 지정 가능  
동일한 의미/역할의 **복수 개의 객체**와의 관계

A	-a	-b	B
- b : List<B>	1	0..*	- a : A

```
public class A{  
    private List<B> b;  
}
```

```
public class B{  
    private A a;  
}
```

### ✓ 다중 연관

동일한 클래스 간의 존재하는 **복수 개의 연관 관계**를 뜻 함  
다른 의미/역할의 복수 개의 객체와의 관계

A	-a1	-b1	B
- b1 : List<B>	1	1..*	- a1 : A
- b2 : List<B>	-a2	-b2	- a2 : List<A>
	1..*	0..*	

```
public class A{  
    private List<B> b1;  
    private List<B> b2;  
}
```

```
public class B{  
    private A a1;  
    private List<A> a2;  
}
```

# ▶ 의존 관계(Dependency)

## ✓ 의존 관계 (지역 변수, 매개 변수)

두 클래스의 **연산 간의 호출 관계**를 표현한 것으로 제공자의 변경이 이용자에 영향을 미칠 수 있음을 의미함.

(대상 클래스의 객체를 전달 받거나, 생성하여 사용하는 것을 의미.)



```
public class A{
    public void test1(B arg){
        arg.testB();
    }
}
```

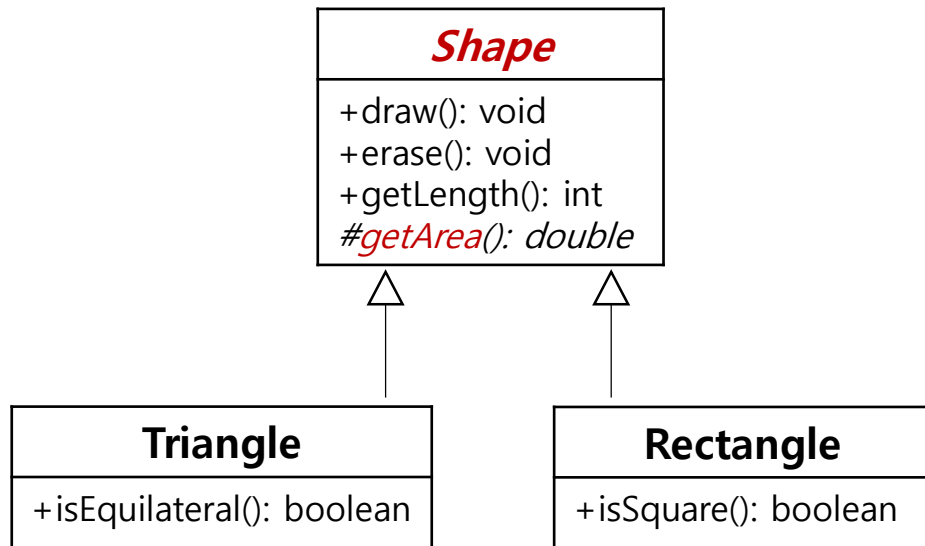


```
public class C{
    public void test2(){
        D d= new D ();
        d.testD();
    }
}
```

# ▶ 일반화 관계(Generalization)

## ✓ 일반화 관계 (상속)

보다 일반적인 클래스와 보다 구체적인 클래스 간의 관계를 뜻하는 것으로  
한 클래스(상위 클래스)가 다른 클래스(하위 클래스)보다 일반적인 개념/대상 임을 의미하는 관계



```
public abstract class Shape {
    public void draw() {...}
    public void erase() {...}
    public int getLength() {...}
    protected abstract double getArea();
}
```

```
public class Triangle extends Shape {
    public boolean isEquilateral() {...}
    protected double getArea() {...}
}
```

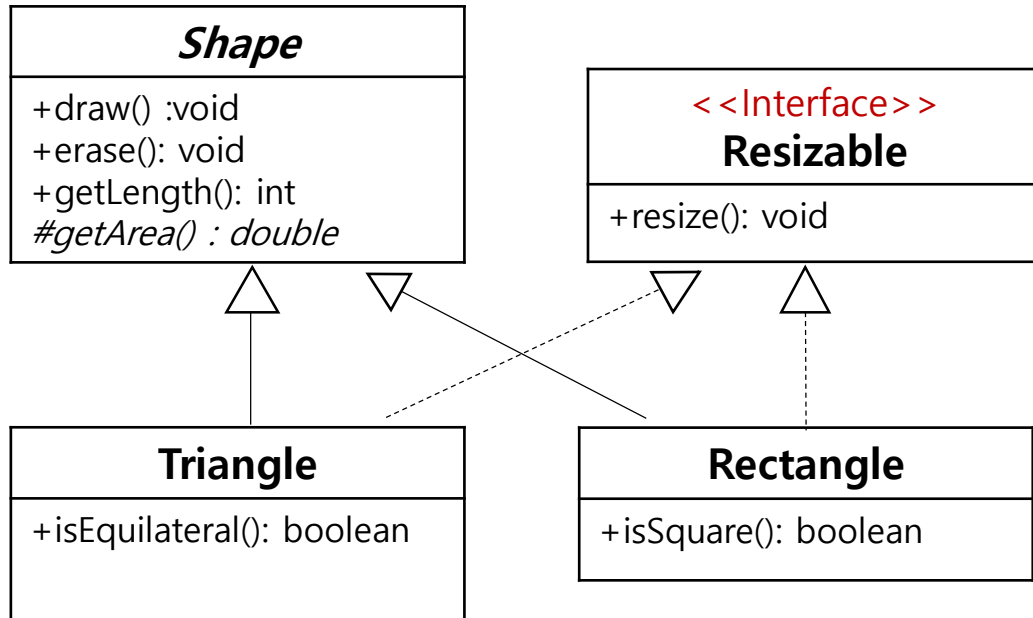
```
public class Rectangle extends Shape {
    public boolean isSquare() {...}
    protected double getArea() {...}
}
```



# ▶ 실체화 관계(Realization)

## ✓ 실체화(인터페이스 구현) 관계

인터페이스에 명시된 기능을 상속받은 클래스에서 **구현한 관계** 의미



```
public interface Resizable {  
    void resize();  
}
```

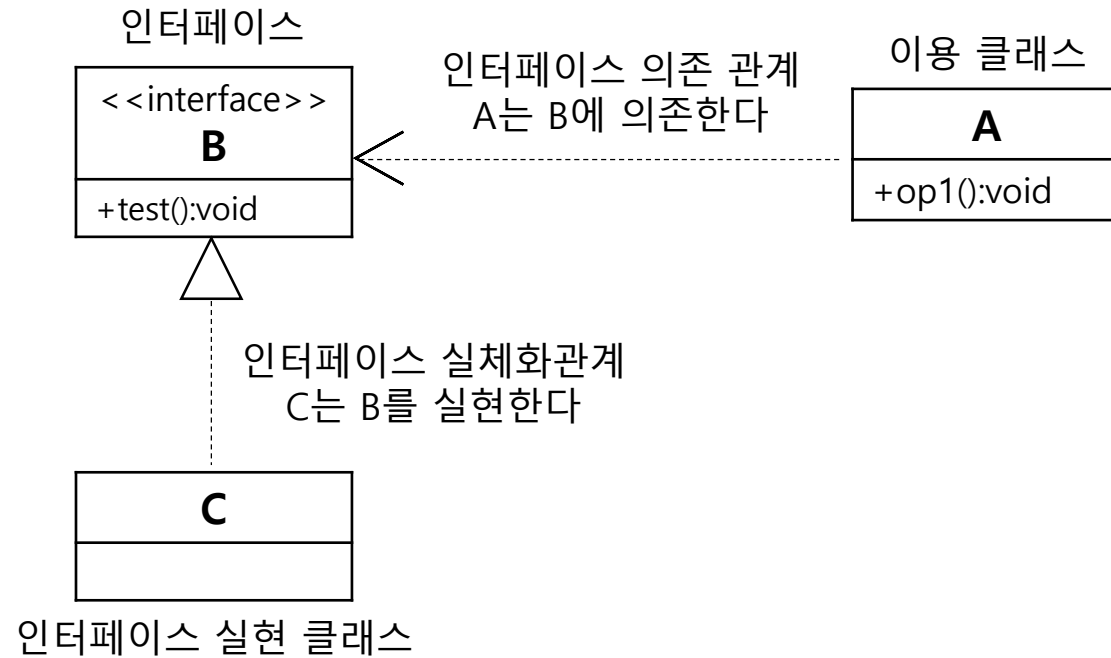
```
public class Triangle extends Shape  
    implements Resizable {  
    public boolean isEquilateral() {...}  
    protected double getArea() {...}  
    public void resize() {...}  
}
```

```
public class Rectangle extends Shape  
    implements Resizable {  
    public boolean isSquare() {...}  
    protected double getArea() {...}  
    public void resize() {...}  
}
```

# ▶ 인터페이스 의존 관계

## ✓ 인터페이스 의존 관계(다형성, 동적 바인딩)

인터페이스와 인터페이스 이용자 간의 이용관계를 표현할 때 사용 될 수 있음

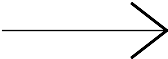
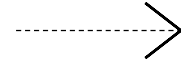


```
public interface B{
    void test();
}
```

```
public class C implements B{
    public void test() {...}
}
```

```
public class A{
    public void op1(){
        B b = new C();
        b.test();
    }
}
```

## ▶ 연관 관계와 의존 관계

	연관 관계	의존 관계
역할	메시지 전달의 통로	
표현식	class A  class B	class A  class B
관계의 발생 형태	A 클래스의 <b>필드</b> 에서 B 클래스 참조	A 클래스의 <b>메소드 매개변수</b> 또는 <b>메소드 내부</b> 에 B 클래스 참조
관계의 지속 범위	A 클래스의 생명주기	참조된 A 클래스 메소드의 생명주기
방향성	양방향 가능	단방향

## ▶ 집합 관계(Aggregation)

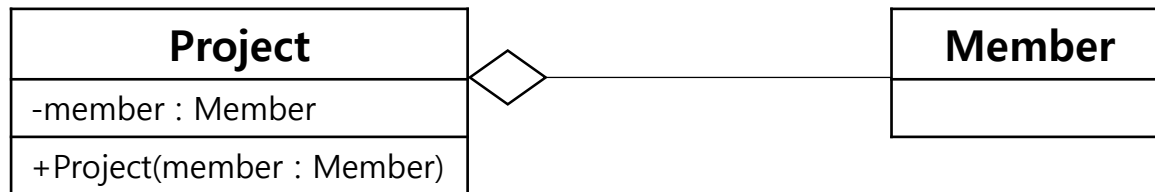
두 대상 간의 포함(소속) 표현

### ✓ 집합 관계

부분 객체가 다수의 전체 객체에 의해 공유 될 수 있음

→ 전체 객체가 사라져도 부분 객체는 존재한다.

ex) 생성자에서 참조값을 인자로 받아 멤버 변수를 세팅



프로젝트는 멤버로 구성된다  
멤버는 프로젝트의 부분이다  
멤버는 다른 프로젝트에도 공유된다

```
public class Project{
    private Member member;

    public Project(Member member){
        this.member = member;
    }
}
```

# ▶ 합성 관계(Composition)

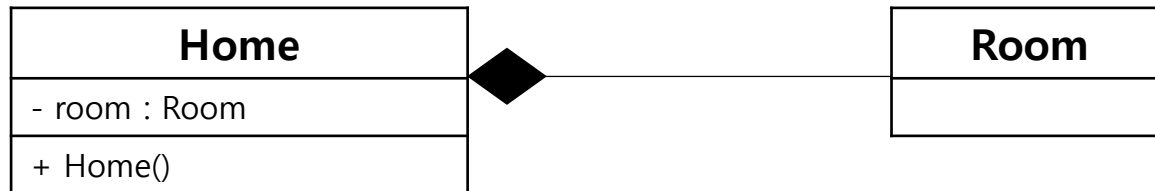
두 대상 간의 포함(소속) 표현

## ✓ 합성 관계

부분 객체가 오직 하나의 전체 객체에 포함 될 수 있음

→ 전체 객체가 사라지면 부분 객체도 사라진다.

ex) 생성자에서 필드에 대한 객체를 생성



집은 방으로 구성된다.  
방은 집의 부분이다.  
집이 사라지면 방도 사라진다.

```
public class Home{
    private Room room;

    public Home(){
        this.room = new Room();
    }
}
```