

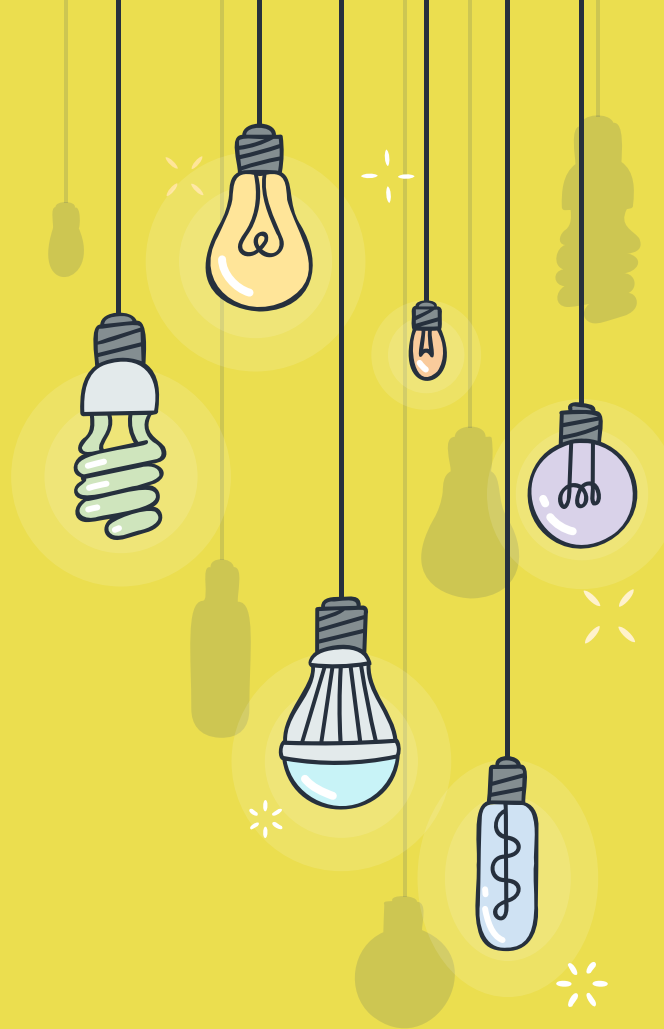
CLASS DIAGRAM

1. 클래스 다이어그램
2. 클래스 다이어그램의 관계
3. 클래스 다이어그램 예제



1

클래스 다이어그램



* 클래스 다이어그램

+ 클래스 다이어그램

- × 객체의 타입 표현
- × 타입 간의 정적(구조) 다이어그램
- × 시스템의 논리적 및 물리적 구성요소 설계 시 주로 활용

+ 클래스의 표현

| | |
|----|-----------|
| + | public |
| # | protected |
| ~ | default |
| - | private |
| 밑줄 | static |

| Student |
|--|
| - name : String - score : int + <u>PI : double = 3.14 {readOnly}</u> |
| + getTotalScore() : int + getAvgScore() : double + goToSchool() : void |

→ 클래스 이름

→ 속성
(readOnly는 상수를 의미)

→ 오퍼레이션

* STEREO TYPE (스테레오 타입)

+ 기본 요소 외에 추가적인 확장요소를 나타냄

× 길러멧(guillemet, « ») 에 적는다.

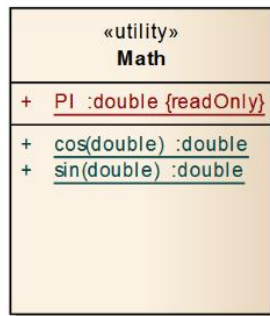
◆ 길러멧 기호 : 쌍
꺾쇠와는 좀 다른
것으로 폰트 크기보다
작다

× <<interface>>

× <<utility>>

× <<abstract>>

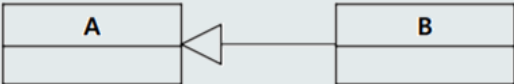




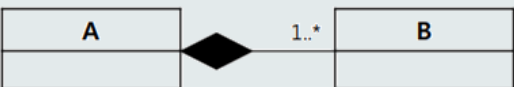

× <<enumeration>> 등



```
2
3 public interface Developer {
4     public void writeCode();
5 }
6
7
8
```

```
3 public class Math {
4
5     public static final double PI = 3.14159;
6
7     public static double sin(double theta) {
8         // Sine 계산...
9         return 0;
10    }
11    public static double cos(double theta) {
12        // Cosine 계산...
13        return 0;
14    }
15 }
```

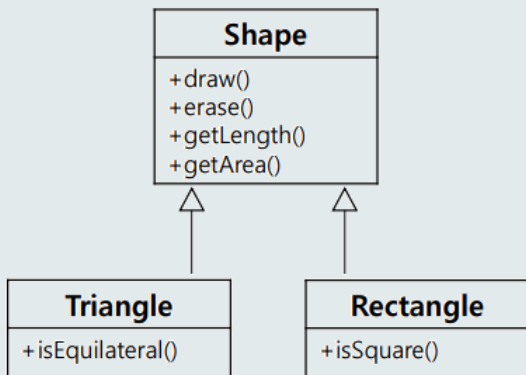
* 클래스 다이어그램의 관계

| 관계 | 표기법 | 의미 |
|-------------------------|---|-------------------------------|
| 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. |
| 연관 관계 |  | 클래스 A와 클래스 B는 연관 관계를 가지고 있다. |
| 합성(포함)관계 |  | 클래스 A는 클래스 B를 한 개 이상 포함하고 있다. |
| 집합 관계 |  | 클래스 B는 클래스 A의 부분이다. |

* 일반화 관계

+ 일반화 관계

× 상속 관계



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

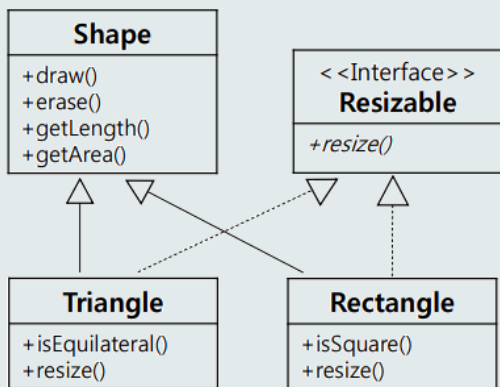
```
public class Triangle extends Shape {
    public boolean isEquilateral() {...}
}
```

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

* 실체화(인터페이스 실현) 관계

+ 실체화(인터페이스 실현) 관계

× 인터페이스에 명시된 기능을 클래스에 의해 구현한 관계



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

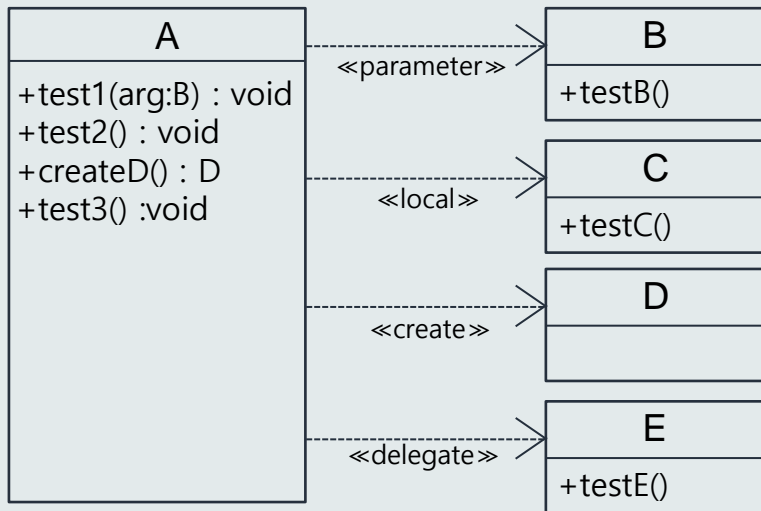
```
public class Triangle extends Shape
    implements Resizable {
    public boolean isEquilateral() {...}
    public void resize() {...}
}
```

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


* 의존 관계

+ 의존 관계

- × 어떤 클래스가 다른 클래스를 참조하는 것
 - ◆ 참조의 형태 : 객체 생성, 객체사용, 메서드 호출, 객체리턴, 매개변수로 객체를 받는 것 등

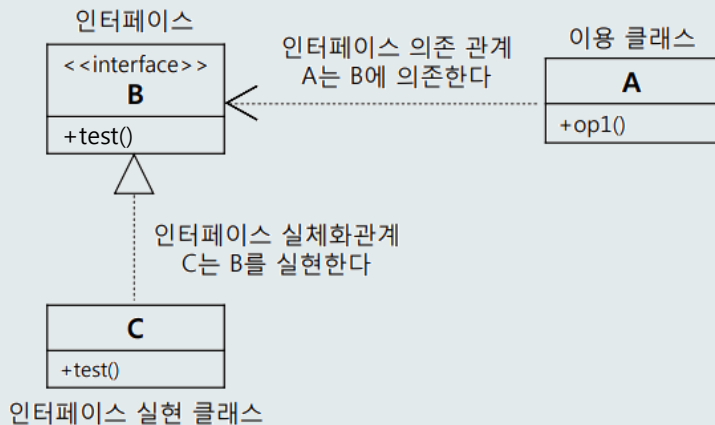


```
public class A {
    private E e;
    public void test1(B arg) {
        arg.testB();
    }
    public void test2() {
        C c = new C();
        c.testC();
    }
    public D createD() {
        return new D();
    }
    public void test3() {
        e.testE();
    }
}
```

* 인터페이스 의존 관계

+ 인터페이스 의존 관계

- × 인터페이스와 인터페이스 이용자 간의 이용관계를 표현할 때 사용 될 수 있음

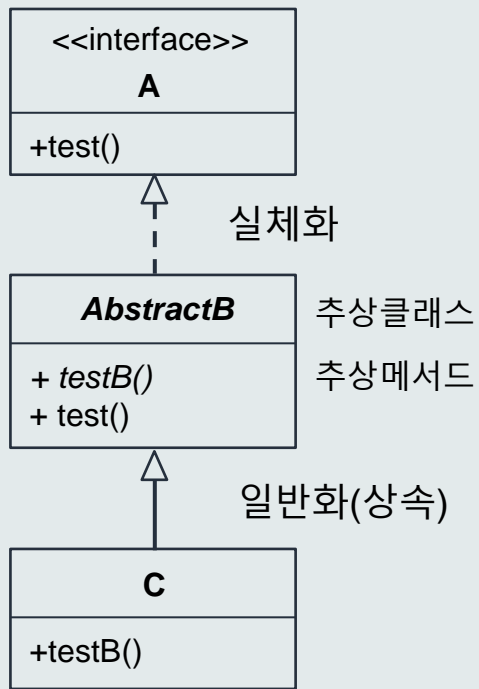


```
public interface B{  
    void test();  
}
```

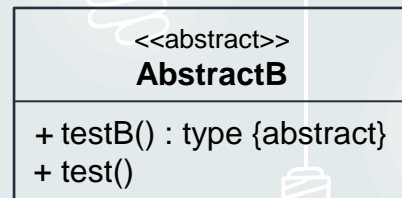
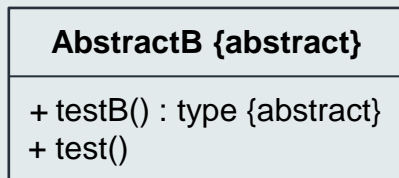
```
public class C implements B{  
    public void test() {...}  
}
```

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

* 인터페이스 의존 관계



* 이외 추상클래스 표기법



* 연관관계

+ 연관관계(Association)

- × 한 클래스가 다른 클래스를 필드로 가진다.
 - ◆ 클래스 간의 관련성을 뜻함. 메시지 전달의 통로 역할



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

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

+ 방향성이 있는 연관 관계

- × 방향성은 메시지 전달의 방향을 의미. 반대 방향 불가능
 - ◆ 클래스 A는 클래스 B를 필드로 가짐



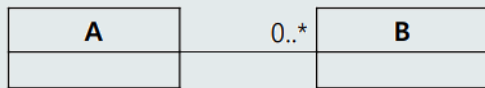
```
public class A{
    private B b;
}
```

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

* 연관관계

+ 연관 관계의 다중성

- × 관계를 맺을 수 있는 실제 상대 객체의 수를 다중성을 통하여 지정 가능
- × 동일한 의미/역할의 복수 개의 객체와의 관계



```

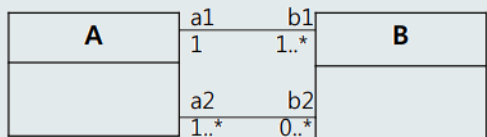
public class A{
    private Collection<B> b;
}
  
```

```

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

+ 다중 연관

- × 동일한 클래스 간의 존재하는 복수 개의 연관 관계를 의미
- × 다른 의미/역할의 복수 개의 객체와의 관계



```

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

```

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

* 합성(포함) 관계

+ 합성(포함) 관계(Composite Aggregation)

- × 전체와 부분의 관계(강한 집합관계)
 - ◆ 회사가 망하면 직원도 함께 없어짐



회사는 직원으로 구성된다
직원은 회사의 부분이다

```
public class Company {
    private List<Member> member;

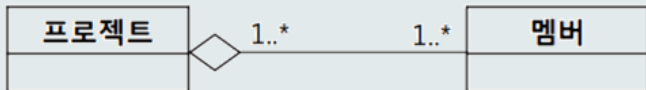
    public Company() {
        member.add(new Member());
    }
}
```

* 집합 관계

+ 집합 관계(Aggregation)

× 전체와 부분의 관계

- ◆ 연관관계와 개념적으로 차이는 있지만 코드는 차이가 없음



멤버는 프로젝트의 부분이다
프로젝트는 멤버로 구성된다

```
public class Project {
    private List<Member> member;

    public Project(Member m) {
        member.add(m);
    }
}
```

* 예제 1

- 1개의 리그에는 팀(구단)이 10개 팀이 있다
- 리그는 1개 이상의 경기장에서 진행을 한다
- 경기장은 1번에 1개의 리그만 경기할 수 있다
- 경기장에는 여러 개의 경기를 할 수 있다
- 경기는 여러 개의 경기장에서 경기를 한다
- 2팀(구단)이 1경기를 한다
- 팀(구단)은 1명 이상의 선수들이 있다
- 야구선수는 1개 팀(구단)에 속해 있다
- 야구선수는 0개 이상의 경기에 참가할 수 있다
- 1경기에는 9명 이상의 선수(출전선수)들이 참가한다
- 야구선수는 주전선수와 후보선수로 나뉜다(일반화 관계)

- 야구선수는

* 속성

private String 선수코드

private String 이름

private String[2] 포지션

private int 경기수 = 0

private int 경력 단, 경력은 0초과이어야 한다

protected int승점

* 오퍼레이션

public int 연봉조정(int 승점)

을 가지고 있다

- 연봉조정 메소드 승점이 5점 이상인 선수만 할 수 있다

연봉 = 연봉 + 승점으로 계산한다.

- 연봉조정이 되면 연봉클래스에 계산된다

* 예제 2

+ 유스케이스 다이어그램과 클래스 다이어그램 만들기

상품 관리 시스템

상품 관리 시스템에는 크게 입고와 출고 두 가지 기능이 있다
입고 기능은 입고가 발생했을 때 입고된 제품의 제품명, 제품코드,
입고가격, 입고량을 등록한다. 현재 재고량을 입고된 만큼 추가한다.
출고 기능은 출고 요청이 발생했을 때 제품명과 판매량을 요청하고
창고에서 출고한다. (출고의 속성으로 판매단가가 들어있다)
입고와 출고는 재고 현황에 등록되고 관리된다

THANKS!

+ Any questions?

