



CLASS DIAGRAM

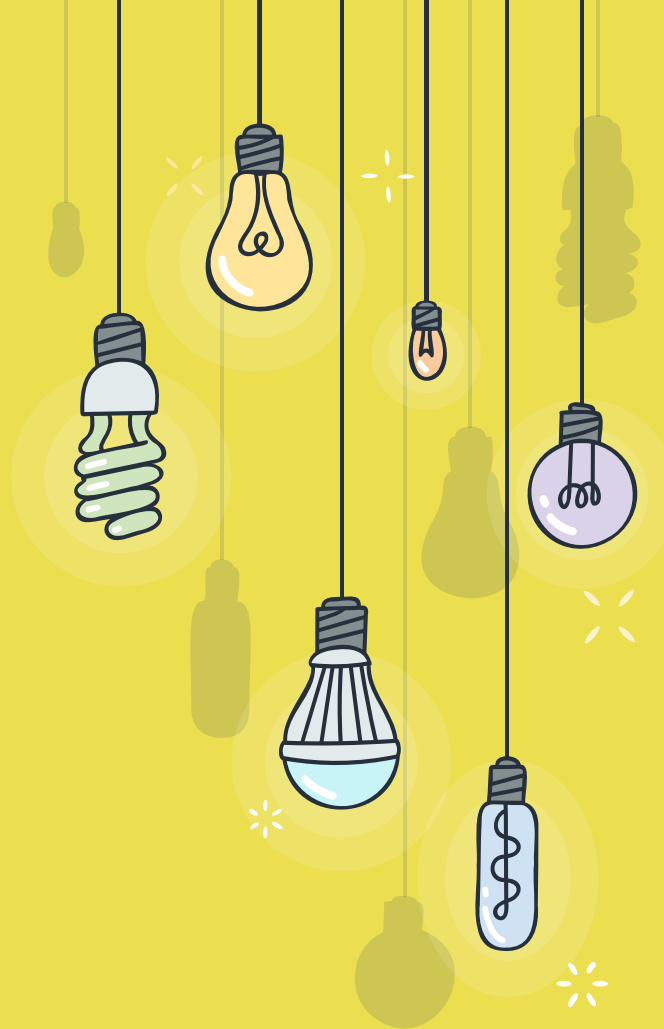
1. 클래스 다이어그램

2. 클래스 다이어그램의 관계



1

클래스 다이어그램



* 클래스 다이어그램

+ 클래스 다이어그램



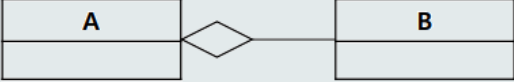
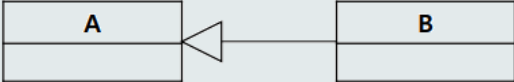



- × 객체의 타입 표현
- × 타입 간의 정적(구조) 다이어그램
- × 시스템의 논리적 및 물리적 구성요소 설계 시 주로 활용

+ 클래스의 표현

+	public
#	protected
~	default
-	private

Student	→ 클래스 이름
- name : String - score : int + <u>PI : double = 3.14</u>	→ 속성
+ getTotalScore() : int + getAvgScore() : double + goToSchool() : void	→ 오퍼레이션

* 클래스 다이어그램의 관계

관계	표기법	의미
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		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.

* 연관관계

+ 연관관계

- ✕ 한 클래스가 다른 클래스를 필드로 가지면서 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



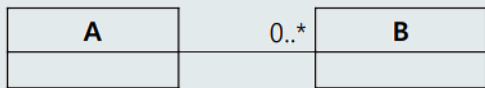
```
public class A{
    private B b;
}
```

```
public class B{
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



```

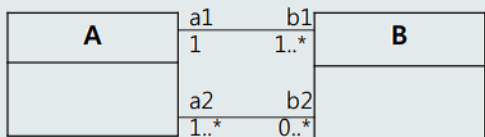
public class A{
    private Collection<B> b;
}
  
```

```

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) 관계와 집합 관계

+ 포함 관계

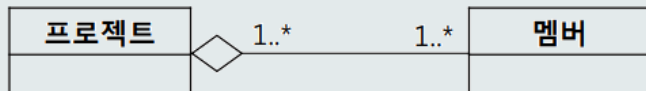
- × 양 측 클래스의 관계를 1:1의 관계로 나타냄
 - ◆ 부분 객체가 오직 하나의 전체 객체에 포함 될 수 있음



회사는 직원으로 구성된다
직원은 회사의 부분이다

+ 집합 관계

- × 연관 관계를 좀 더 세분화 하여 양 측 클래스의 관계를 1:N의 관계로 나타냄
 - ◆ 부분 객체가 다수의 전체 객체에 의해 공유 될 수 있음

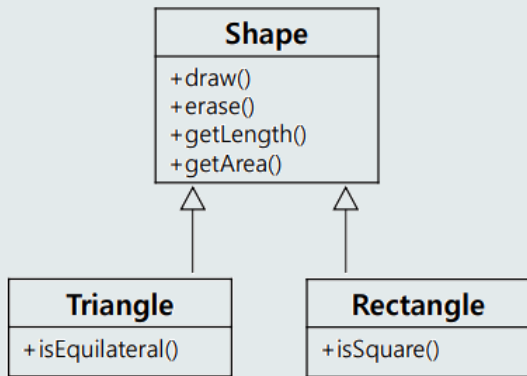


멤버는 프로젝트의 부분이다
프로젝트는 멤버로 구성된다

* 일반화 관계

+ 일반화 관계

- × 한 클래스(상위 클래스)가 다른 클래스(하위 클래스)보다 일반적인 개념/대상 임을 의미하는 관계(상속)



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

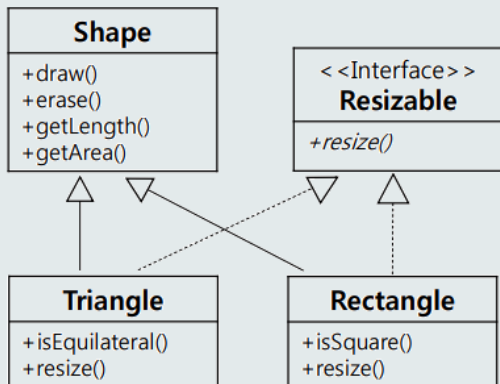
```
public class Triangle extends Shape {
    public boolean isEquilateral() {...}
}
```

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

* 실체화(인터페이스 실현) 관계

+ 실체화(인터페이스 실현) 관계

- × 인터페이스에 명시된 기능을 클래스에 의해서 구현한 관계 의미



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

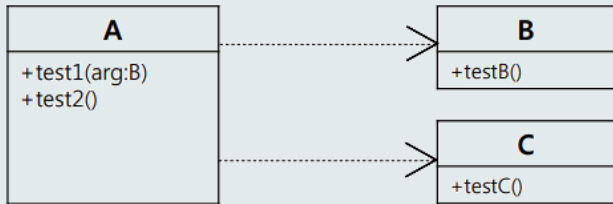
```
public class Triangle extends Shape
    implements Resizable {
    public boolean isEquilateral() {...}
    public void resize() {...}
}
```

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

* 의존 관계

+ 의존 관계

- × 어떤 클래스가 다른 클래스를 참조하는 것
- × 연산 간의 호출 관계를 표현한 것으로 제공자의 변경이 이용자에 영향을 미칠 수 있음을 의미(제공자의 변경이 이용자의 변경 유발)
- × 이용자는 의존 관계를 통해서 제공자의 연산을 호출할 수 있음

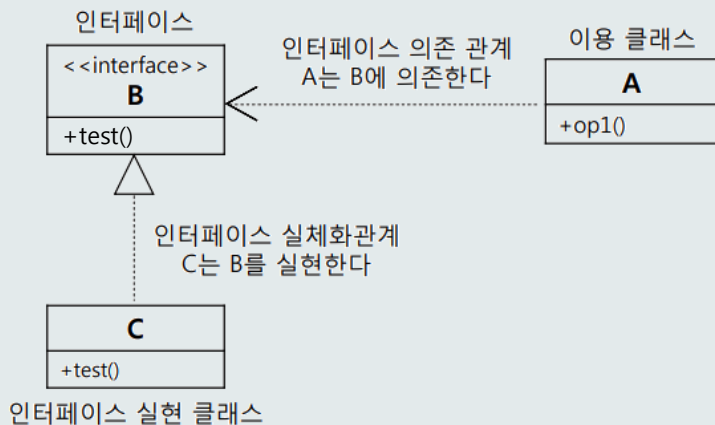


```
public class A{  
    public void test1(B arg){  
        arg.testB();  
    }  
    public void test2(){  
        C c = new C ();  
        c.testC();  
    }  
}
```

* 인터페이스 의존 관계

+ 인터페이스 의존 관계

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

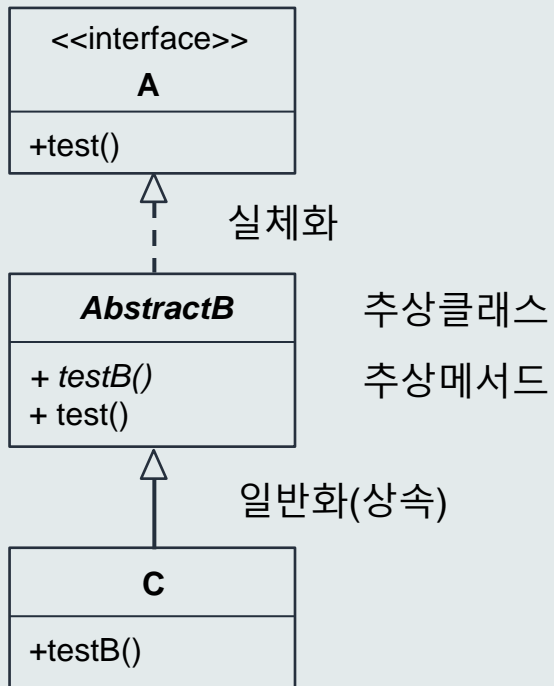


```
public interface B{
    void test();
}

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

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

* 인터페이스 의존 관계

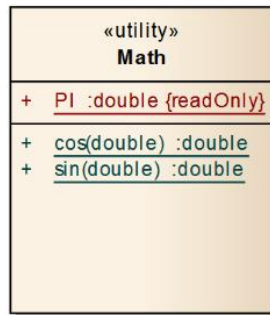




+ Stereo Type (스테레오 타입)

- × 기본 요소 외에 추가적인 확장요소를 나타내는 것으로 쌍 꺾쇠와 비슷하게 생긴 길러멧(guillemet, « »)에 적는다.

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



```
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 }
```

THANKS!

+ Any questions?

