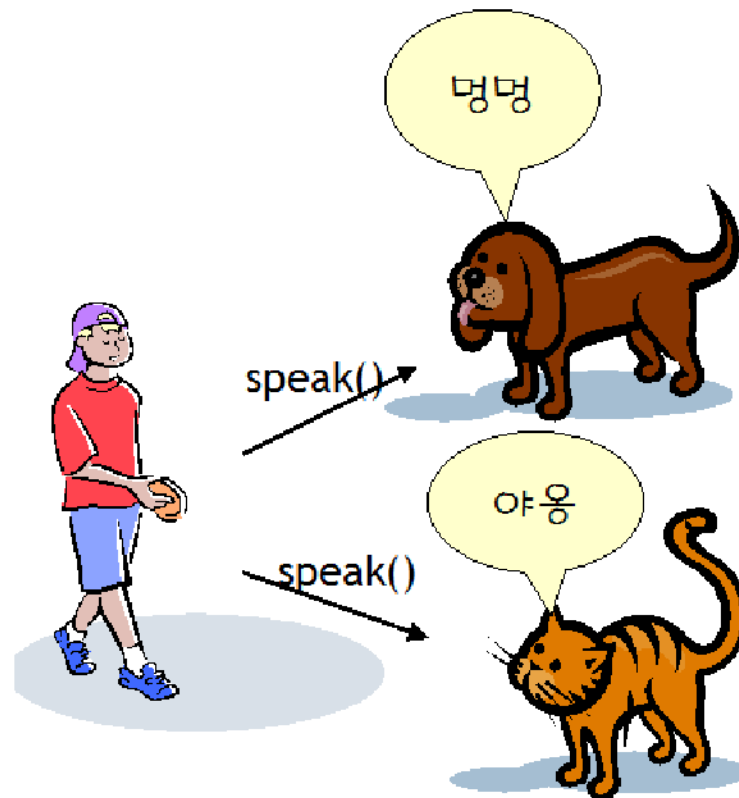




다형성이란?

- 다형성(**polymorphism**)이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것





상항 형변환

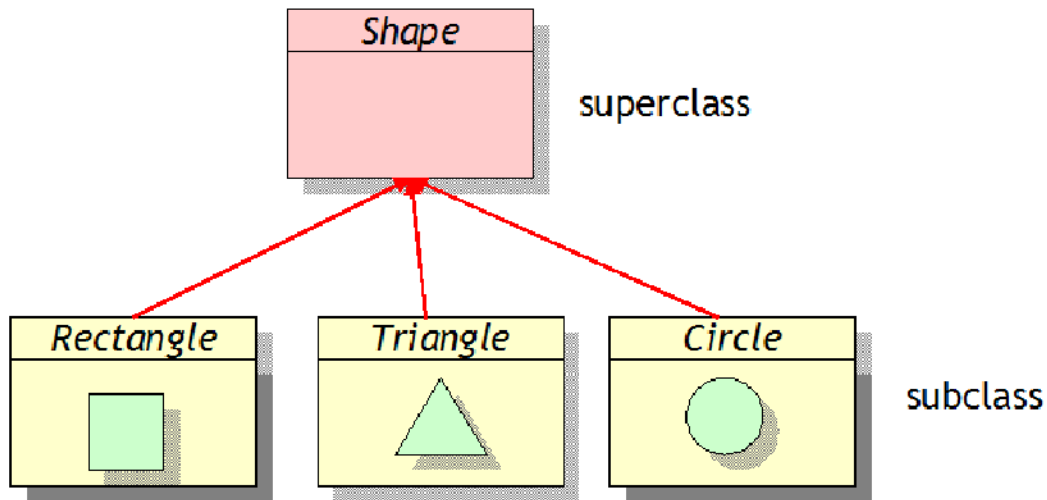


그림 12.6 도형의 상속 구조

Shape 타입
변수로 Rectangle
객체를 참조하니
틀린 거 같지만
올바른 문장!!

```
Shape s = new Rectangle(); // OK!
```





왜 그럴까?

- 서브 클래스 객체는 슈퍼 클래스 객체를 포함하고 있기 때문이다.

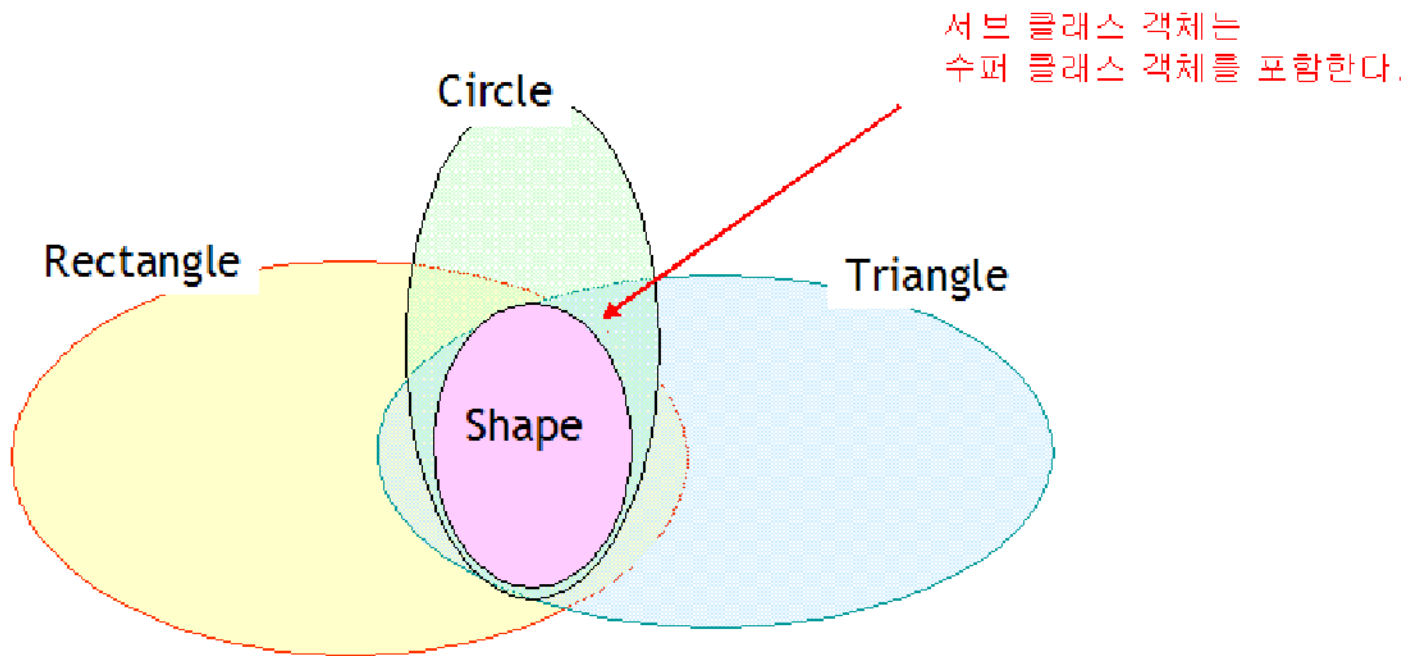
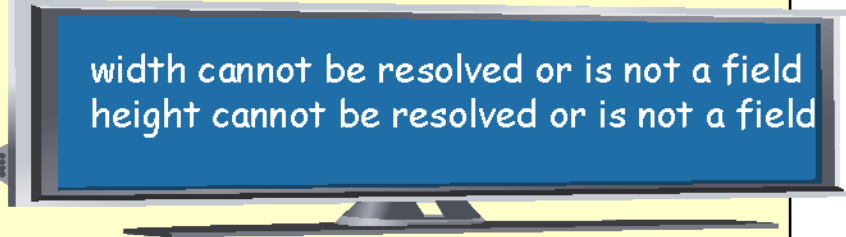


그림 12.7 서브 클래스와 슈퍼 클래스의 포함 관계



예제

```
class Shape {  
    int x, y;  
}  
class Rectangle extends Shape {  
    int width, height;  
}  
public class ShapeTest {  
    public static void main(String arg[]) {  
        Shape s;  
        Rectangle r = new Rectangle();  
        s = r;  
        s.x = 0;  
        s.y = 0;  
        s.width = 100;  
        s.height = 100;  
    }  
}
```

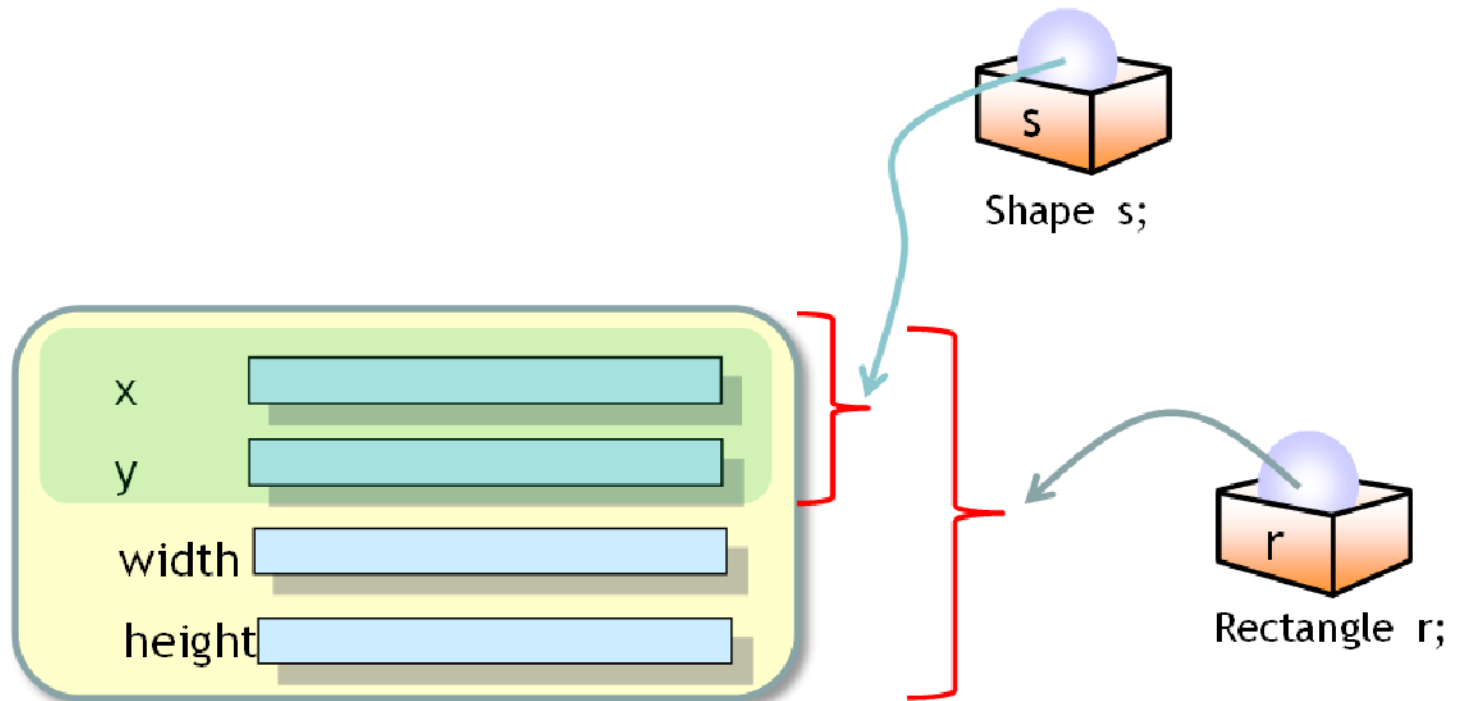


width cannot be resolved or is not a field
height cannot be resolved or is not a field

컴파일 오류가
발생한다. **s**를
통해서는 **Rectangle**
클래스의 필드와
메소드에 접근할 수
없다.



다형성



Rectangle 객체



형 변환

- `Shape s = new Rectangle();`
- `s`를 통하여 `Rectangle` 클래스의 필드와 메소드를 사용하고자 할 때는 어떻게 하여야 하는가?

`((Rectangle) s).setWidth(100);`



하향 형변환

- 서브 클래스 참조 변수로 수퍼 클래스 객체를 참조하는 것으로 일반적인 상황에서는 컴파일 오류이다.

```
Rectangle r;  
r = new Shape(); // NOT OK!
```

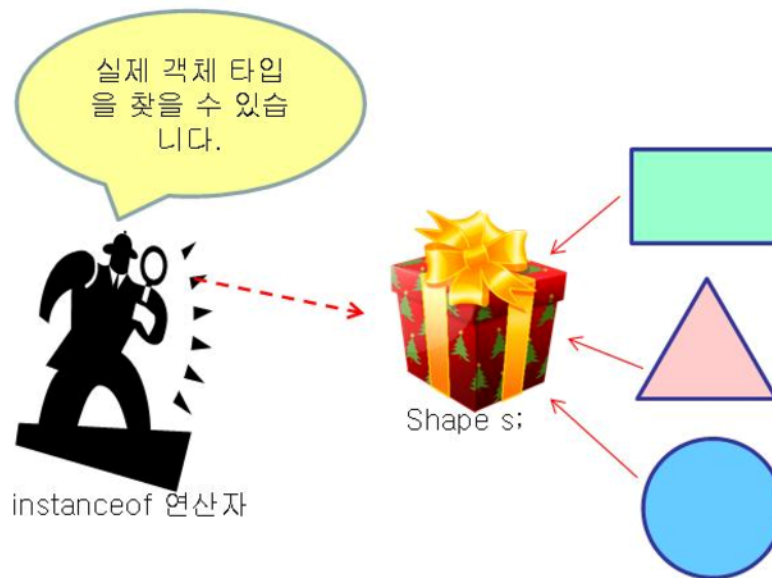
- 만약 서브 클래스 객체인데 형변환에 의하여 일시적으로 수퍼 클래스 참조 변수에 의하여 참조되고 있는 경우는 가능

```
Rectangle r;  
Shape s;  
s = new Rectangle();  
r = (Rectangle)s;  
r->width = 100;  
r->height = 100;
```



객체의 실제타입

```
Shape s = getShape();  
if (s instanceof Rectangle) {  
    System.out.println("Rectangle이 생성되었습니다");  
} else {  
    System.out.println("Rectangle이 아닌 다른 객체가 생성되었습니다");  
}
```





메소드의 매개 변수

- 메소드의 매개 변수로 슈퍼 클래스 참조 변수를 이용한다.
-> 다형성을 이용하는 전형적인 방법

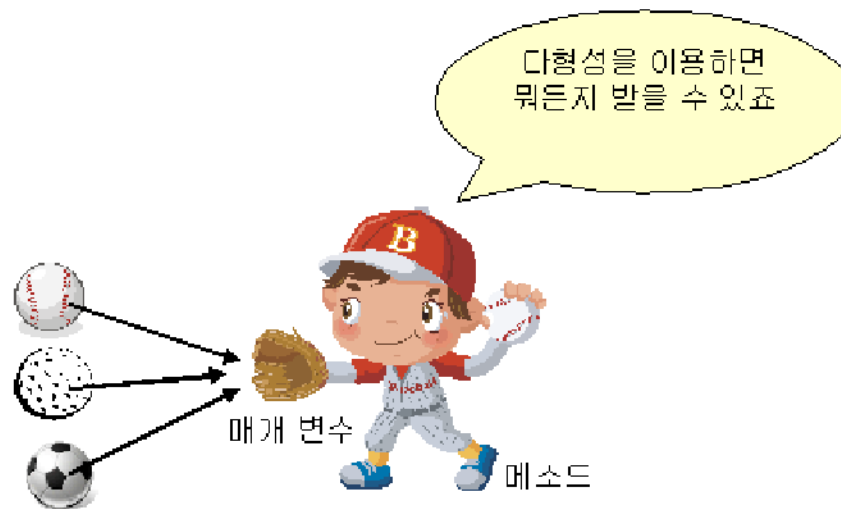


그림 12.9 다형성을 이용하는 메소드의 매개 변수



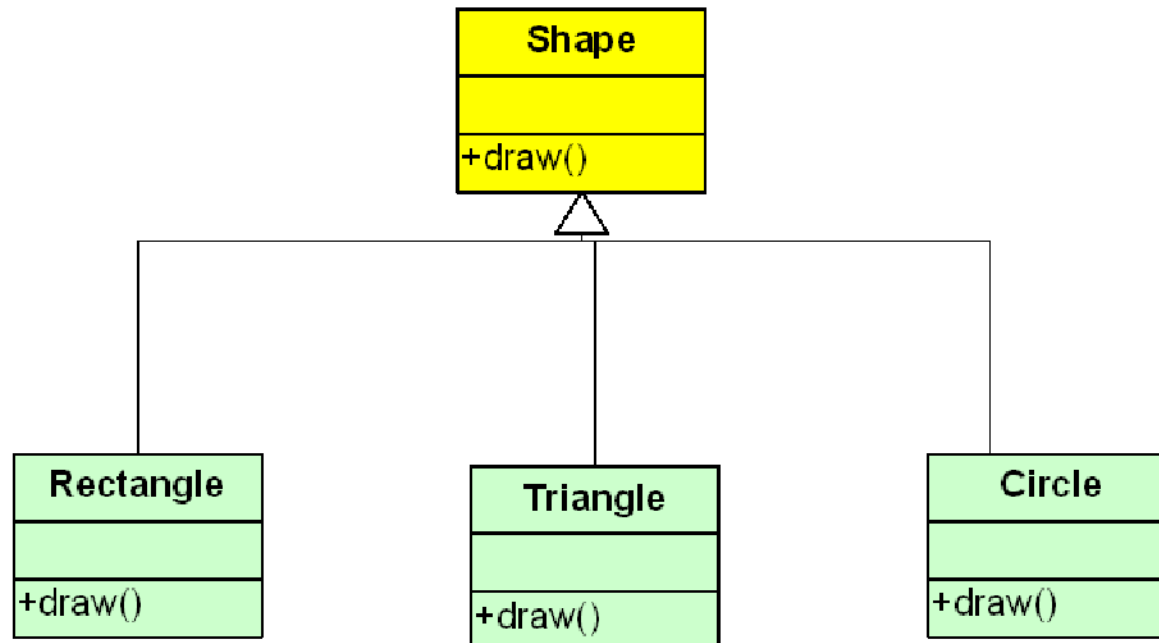
다형성의 이용

- 메소드의 매개 변수를 선언시

```
public static double calcArea(Shape s) {  
  
    double area = 0.0;  
    if (s instanceof Rectangle) {  
        int w = ((Rectangle) s).getWidth();  
        int h = ((Rectangle) s).getHeight();  
        area = (double) (w * h);  
    }  
    ...    // 다른 도형들의 면적을 구한다.  
    return area;  
}
```



동적 바인딩



```
Shape s = new Rectangle();    // OK!
s.draw();                    // 어떤 draw()가 호출되는가?
```

Shape의 draw()가 호출되는 것이 아니라 Rectangle의 draw()가 호출된다. s의 타입은 Shape이지만 s가 실제로 가리키고 있는 객체의 타입이 Rectangle이기 때문이다.



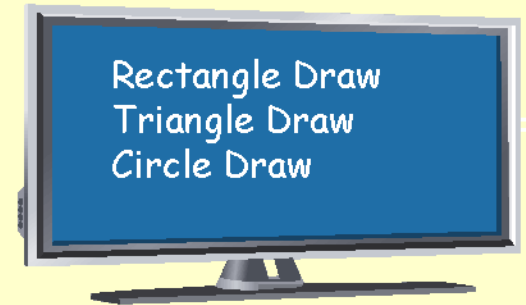
예제

```
class Shape {  
    protected int x, y;  
    public void draw() {        System.out.println("Shape Draw");    }  
}  
class Rectangle extends Shape {  
    private int width, height;  
    public void draw() {        System.out.println("Rectangle Draw");    }  
}  
class Triangle extends Shape {  
    private int base, height;  
    public void draw() {        System.out.println("Triangle Draw");    }  
}  
class Circle extends Shape {    private int radius;  
    public void draw() {        System.out.println("Circle Draw");    }  
}
```



예제

```
public class ShapeTest {  
    private static Shape arrayOfShapes[];  
    public static void main(String arg[]) {  
        init();  
        drawAll();  
    }  
    public static void init() {  
        arrayOfShapes = new Shape[3];  
        arrayOfShapes[0] = new Rectangle();  
        arrayOfShapes[1] = new Triangle();  
        arrayOfShapes[2] = new Circle();  
    }  
    public static void drawAll() {  
        for (int i = 0; i < arrayOfShapes.length i++) {  
            arrayOfShapes[i].draw();  
        }  
    }  
}
```



어떤 draw()가 호출되는가?



다형성의 장점

- 만약 새로운 도형 클래스를 작성하여 추가한다고 해보자.

```
class Cylinder extends Shape {  
    public void draw(){  
        System.out.println("Cylinder Draw");  
    }  
};
```

- drawAll() 메소드는 수정할 필요가 없다.

```
public static void drawAll() {  
    for (int i = 0; i < arrayOfShapes.length; i++) {  
        arrayOfShapes[i].draw();  
    }  
}
```