

어서와 *Java*는 처음이지!

제7장 상속

- Super 키워드
- 상속과 생성자
- 상속과 다형성

자바에도 상속이 있군요!
다른 클래스의 코드를
상속받을 수 있나요?

네, 상속은 기존 클래스의
코드를 재사용하는 아주 좋은
기법입니다. 많이 사용해 보세요.





super를 사용하여 부모 클래스 멤버 접근

Parent.java

```
01 public class Parent {  
02     public void print() {  
03         System.out.println("부모 클래스의 print() 메소드");  
04     }  
05 }
```

Child.java

```
01 public class Child extends Parent {  
02     public void print() {  
03         super.print();  
04         System.out.println("자식 클래스의 print() 메소드 ");  
05     }  
06     public static void main(String[] args) {  
07         Child obj = new Child();  
08         obj.print();  
09     }  
10 }
```

메소드 오버라이드



부모 클래스의 메소드 호출



상속과 생성자

- 서브 클래스의 객체가 생성될 때, 서브 클래스의 생성자만 호출될까?
- 아니면 수퍼 클래스의 생성자도 호출되는가?



상속과 생성자

```
class Base{  
    public Base(String msg) {  
        System.out .println("Base() 생성자");  
    }  
};
```

```
class Derived extends Base {  
    public Derived() {  
        System.out .println("Derived() 생성자");  
    }  
};
```

```
public class Test {  
    public static void main(String[] args) {  
        Derived r = new Derived();  
    }  
};
```



실행 출력

- 생성자의 호출 순서는 (부모 클래스의 생성자) → (자식 클래스의 생성자) 순으로 된다.

Base() 생성자

Derived() 생성자



명시적인 생성자 호출

- super 를 이용하여서 명시적으로 슈퍼 클래스의 생성자 호출

```
class Shape {
```

```
    public Shape(String msg) {  
        System.out.println("Shape 생성자() " + msg);  
    }
```

```
};
```

```
public class Rectangle extends Shape {
```

```
    public Rectangle() {  
        super("from Rectangle"); // 명시적인 호출  
        System.out.println("Rectangle 생성자()");  
    }
```

```
};
```



묵시적인 생성자 호출

```
class Shape {  
    public Shape(String msg) {  
        System.out.println("Shape 생성자()");  
    }  
};
```

```
class Rectangle extends Shape {  
    public Rectangle() {  
        System.out.println("Rectangle 생성자()");  
    }  
};
```



Shape 생성자
Rectangle 생성자



다형성이란?

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





자바에서의 자료형 검사

- 클래스 A의 참조 변수로 클래스 B의 객체를 참조할 수는 없다.

```
class A {  
    A() {}  
}  
  
class B {  
    B() {}  
}  
  
public class TypeTest1 {  
    public static void main(String args[]) {  
        A a = new B(); // NO!  
    }  
}
```



상향 형변환

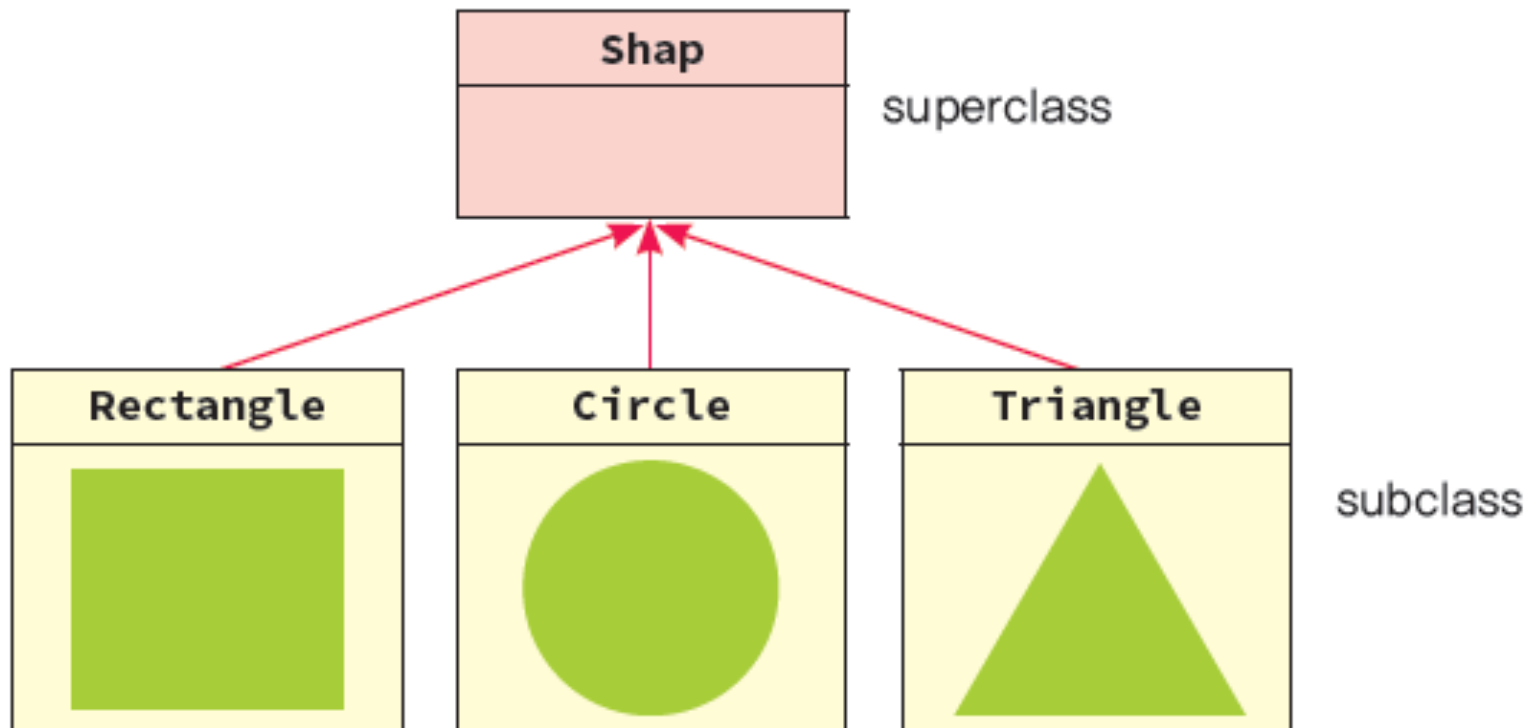
- 부모 클래스의 참조 변수는 자식 클래스의 객체를 참조할 수 있다!

```
class A {  
    A() {}  
}  
class B extends A {  
    B() {}  
}  
public class TypeTest1 {  
    public static void main(String args[]) {  
        A a = new B(); // OK!  
    }  
}
```



상속과 다형성

- 하나의 예로 Rectangle, Triangle, Circle등의 도형 클래스가 부모 클래스인 Shape 클래스로 부터 상속되었다고 가정하자.





상향 형변환

```
class Shape {  
    protected int x, y;  
}  
  
class Rectangle extends Shape {  
    private int width, height;  
}  
  
class Triangle extends Shape {  
    private int base, height;  
}  
  
class Circle extends Shape {  
    private int radius;  
}
```

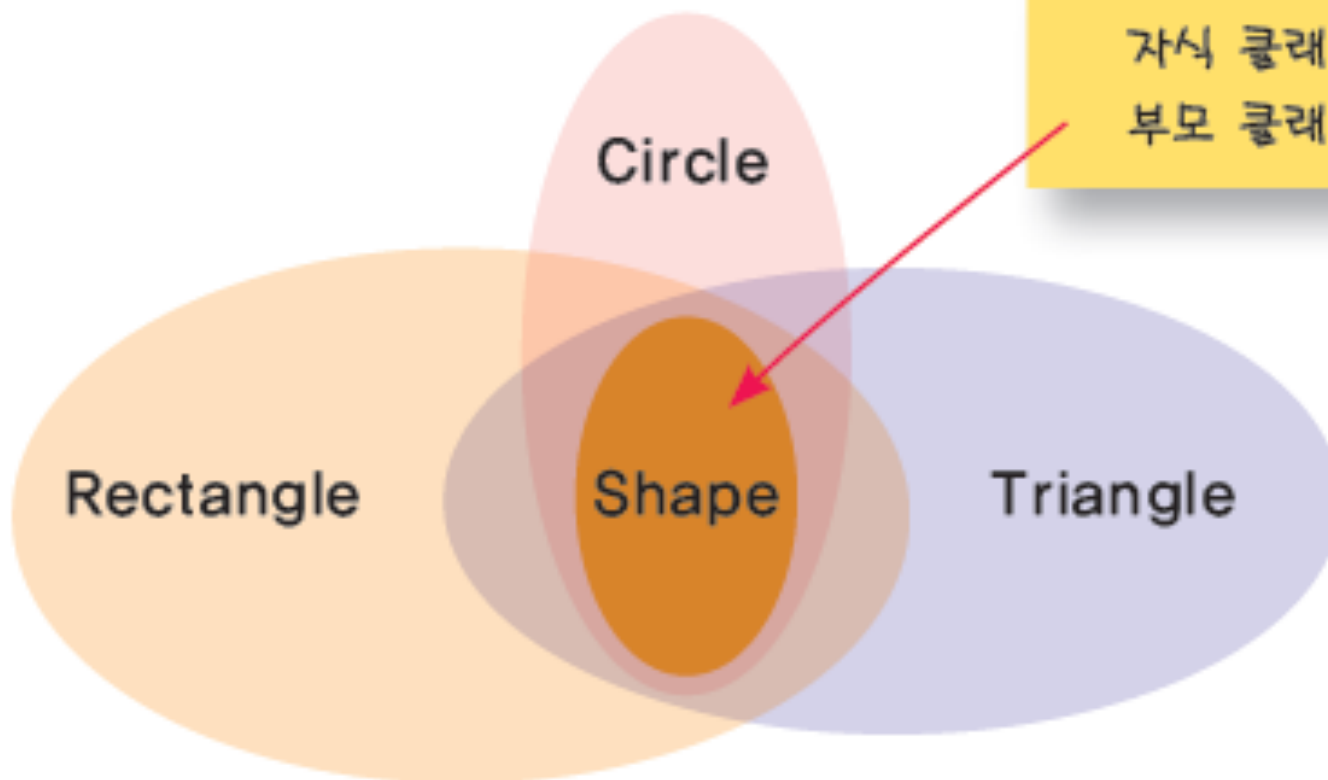


```
public class ShapeTest {  
    public static void main(String arg[]) {  
        Shape s1, s2;  
  
        s1 = new Shape(); // ① 당연하다.  
        s2 = new Rectangle(); // ② Rectangle 객체를  
                                // Shape 변수로 가리킬 수 있을까?  
    }  
}
```



왜 그럴까?

- 서브 클래스 객체는 수퍼 클래스 객체를 포함하고 있기 때문이다.



자식 클래스 객체는
부모 클래스 객체를 포함한다.



예제

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

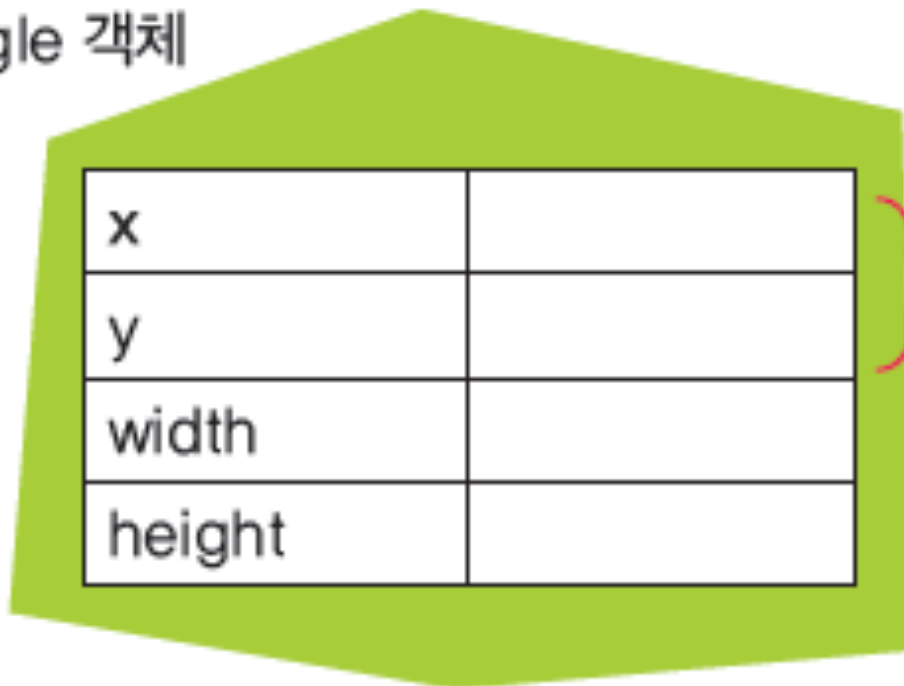
width cannot be resolved or is not a field
height cannot be resolved or is not a field

컴파일 오류가 발생한다.
s를 통해서만 **Rectangle**
클래스의 필드와 메소드에
접근할 수 없다.



다형성

Rectangle 객체



Shape s;

Rectangle r;



형변환

- `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 슈퍼 클래스 객체를 참조하는 것으로 일반적인 상황에서는 컴파일 오류이다.

```
Rectangle r;
```

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



하향 형변환

- 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;
```

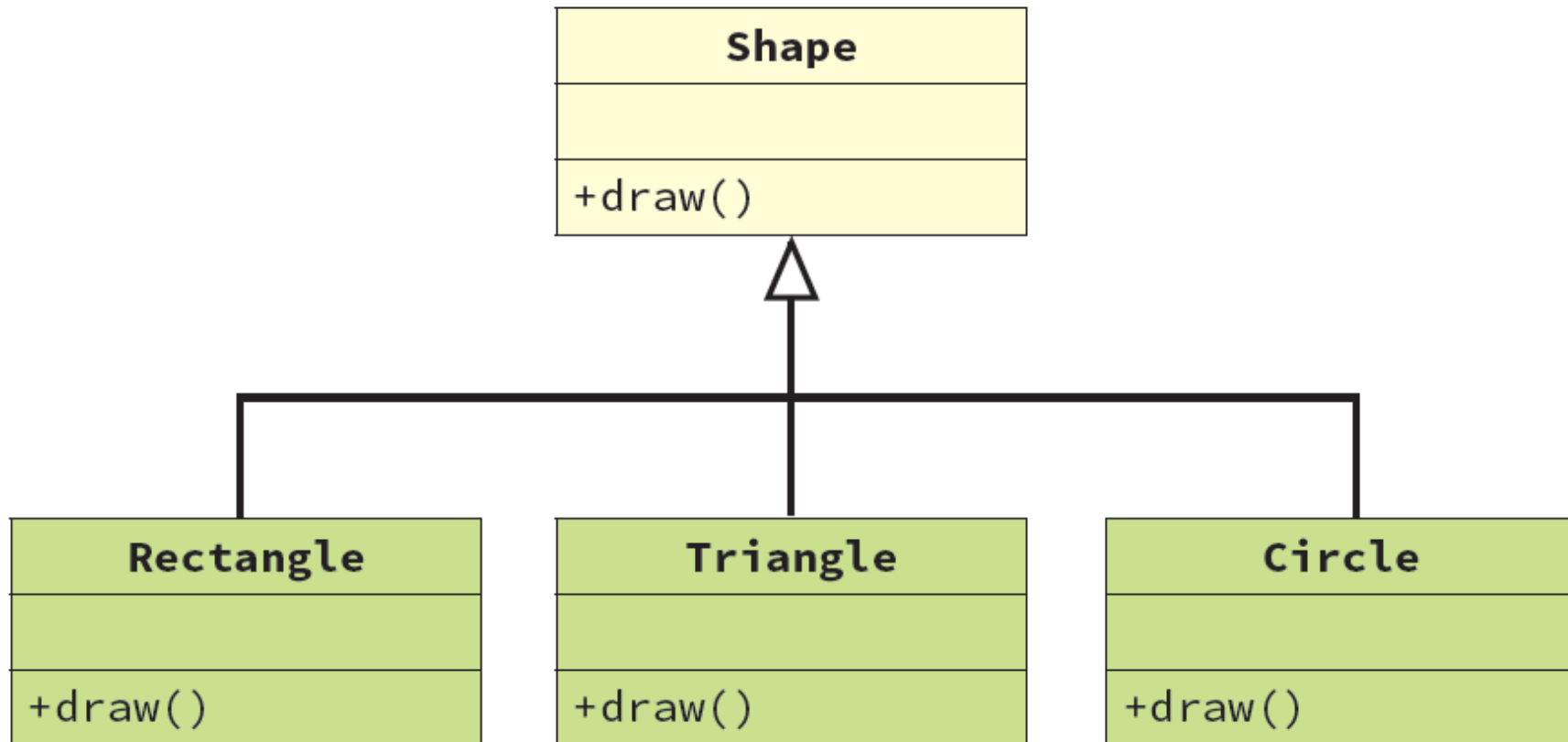
```
r->width = 100;
```

```
r->height = 100;
```



동적 메소드 호출

- 다형성은 객체들이 동일한 메시지를 받더라도 각 객체의 타입에 따라서 서로 다른 동작을 하는 것





```
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");  
    }  
}
```



```
public class ShapeTest {  
    public static void main(String arg[]) {  
        Shape s1, s2, s3, s4;  
  
        s1 = new Shape();  
        s2 = new Rectangle();  
        s3 = new Triangle();  
        s4 = new Circle();  
  
        s1.draw();  
        s2.draw();  
        s3.draw();  
        s4.draw();  
    }  
}
```



실행 결과

Shape Draw

Rectangle Draw

Triangle Draw

Circle Draw



동적 바인딩

- 메소드 호출을 실제 메소드의 몸체와 연결하는 것을 바인딩(binding)이라고 한다.
- 자바 가상 머신(JVM)은 실행 단계에서 객체의 타입을 보고 적절한 메소드를 호출하게 된다. 이것을 동적 바인딩(dynamic binding) 또는 가상 메소드 호출(virtual method invocation)이라고 한다.

```
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();  
        }  
    }  
}
```



예제

실행결과



Rectangle Draw
Triangle Draw
Circle Draw



어떤 장점이 있을까?

```
class Cylinder extends Shape {  
    private int radius, height;  
  
    public void draw(){  
        System.out.println("Cylinder Draw");  
    }  
}
```

- 위와 같은 새로운 클래스가 추가되더라도 다른 코드는 변경할 필요가 없다.

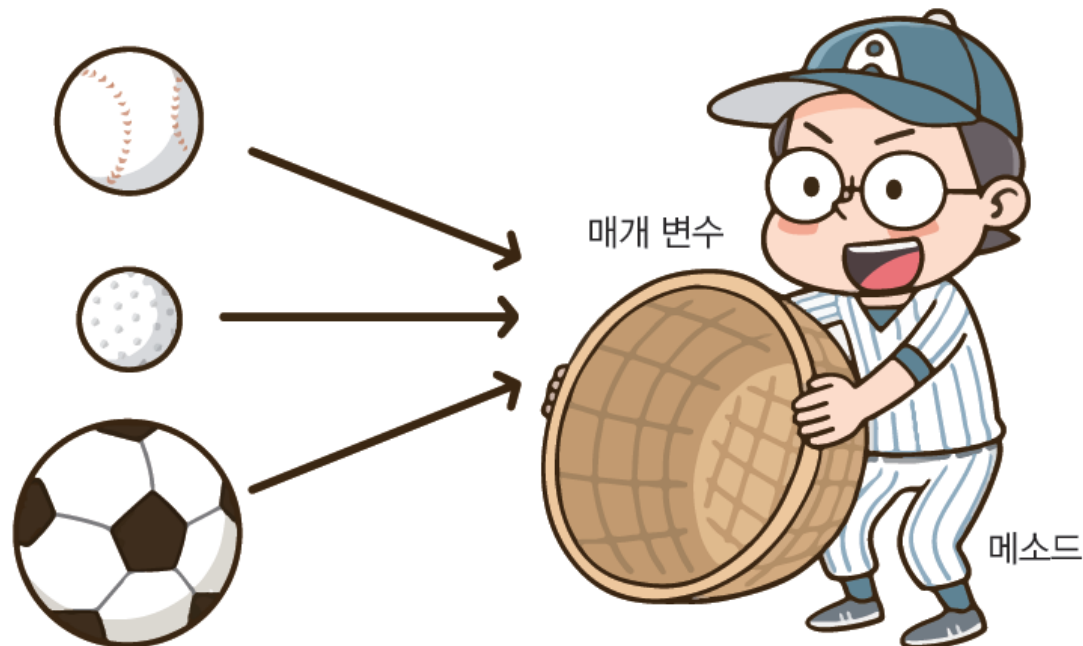


메소드의 매개 변수

- 메소드의 매개 변수로 부모 클래스 참조 변수를 이용한다.

-> 다형성을 이용하는 전형적인 방법

다형성을 이용하면
뭐든지 받을 수 있죠





예제

```
public class ShapeTest {  
    public static void printLocation(Shape s) {  
        System.out.println("x=" + s.x + " y=" + s.y);  
    }  
    public static void main(String arg[]) {  
        Rectangle s1 = new Rectangle();  
        Triangle s2 = new Triangle();  
        Circle s3 = new Circle();  
  
        printLocation(s1);  
        printLocation(s2);  
        printLocation(s3);  
    }  
}
```



LAB: 동적 메소드 호출

- 강아지와 고양이를 나타내는 클래스를 작성하자. 이들 클래스의 부모 클래스로 Animal 클래스를 정의한다. 강아지와 고양이 클래스의 sound() 메소드를 호출하면 각 동물들의 소리가 출력되도록 프로그램을 작성해보자.

실행결과

Animal 클래스의 sound()

멍멍

야옹





```
class Animal {  
    void sound() {  
        System.out.println("Animal 클래스의 sound()");  
    }  
}  
  
class Dog extends Animal {  
    void sound() {  
        System.out.println("멍멍");  
    }  
}  
  
class Cat extends Animal {  
    void sound() {  
        System.out.println("야옹");  
    }  
}
```



```
public class DynamicCallTest {  
    public static void main(String args[]) {  
        Animal animal = new Animal();  
        Dog dog = new Dog();  
        Cat cat = new Cat();  
  
        Animal obj;  
  
        obj = animal;  
        obj.sound();  
  
        obj = dog;  
        obj.sound();  
  
        obj = cat;  
        obj.sound();  
    }  
}
```



IS-A 관계

- is-a 관계: “~은 ~이다” 와 같은 관계
- 상속은 is-a 관계이다.
- 자동차는 탈것이다(Car is a Vehicle).
- 강아지는 동물이다(Dog is a animal).