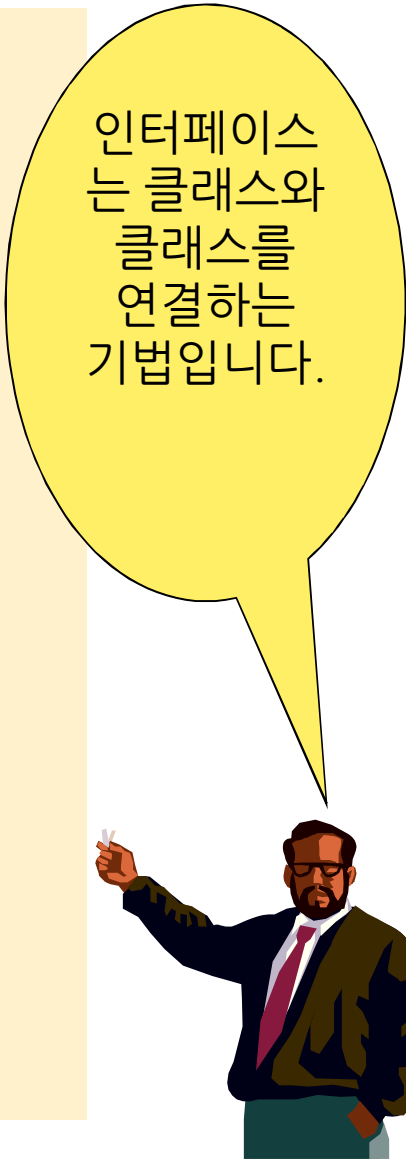
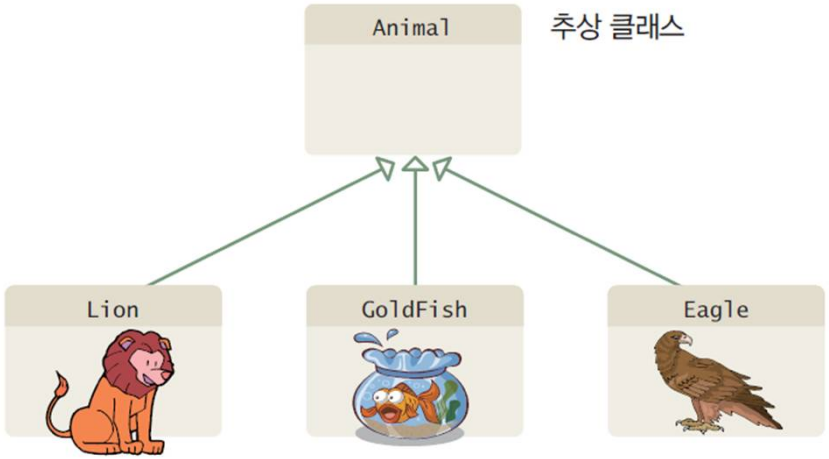
다형성와 인터페이스

- 추상 클래스와 추상 메서드
  - 추상(abstract) 클래스 : 추상 메서드가 들어 있는 클래스
  - 추상(abstract) 메서드 : abstract가 붙은 메서드
  - 추상(abstract) 메서드를 사용하는 이유는
    - 자식 클래스 메서드의 override를 강제하기 위해서
- 인터페이스
  - **인터페이스란?**
    - **abstract 메서드만 있는 클래스. 필드는 없다**
  - **인터페이스 왜 쓰는가?**
    - 규약이 만족되면 다 받아줘.
    - 개발자에게 반드시 메서드를 구현하게 만들기 위해서.
  - **인터페이스 정의와 구현**
  - 인터페이스 상속
- 자바에서는 단일 상속만 허용.
- **인터페이스를 이용하여 다중 상속의 효과를 구현**



abstract 메서드를 왜 만드나요?




```
public abstract class Animal  {
    public abstract void move();        // ;으로 종료됨을 유의!
    ...
};
```

추상 메소드 정의

- 추상 메서드는 abstract가 붙은 메서드.
- 추상 클래스는 추상 메서드가 들어 있는 클래스
- 추상 메서드를 사용하는 이유는 자식클래스에서 **Overriding을 강제**하고자 할 때

추상 클래스와 추상 메서드 예제

Shape.java

```
public abstract class Shape {
    int x, y;
    public void move(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public abstract void draw();
};
```

추상 클래스 Shape를 선언한다. 추상 클래스로는 객체를 생성할 수 없다.

추상 클래스라고 하더라도 추상 메소드가 아닌 보통의 메소드도 가질 수 있음을 유의하라.

추상 메소드를 선언한다. 추상 메소드를 하나라도 가지면 추상 클래스가 된다. 추상 메소드를 가지고 있는데도 abstract를 class 앞에 붙이지 않으면 컴파일 오류가 발생한다.

Rectangle.java

```
public class Rectangle extends Shape {
    int width, height;
    public void draw() {      // 추상 메소드 구현
        System.out.println("사각형 그리기 메소드");
    }
};
```

서브 클래스 Rectangle에서 수퍼 클래스의 추상 메소드 draw()가 실제 메소드로 구현한다. 서브 클래스에서 추상 메소드를 구현하지 않으면 컴파일 오류가 발생한다.

Circle.java

```
public class Circle extends Shape {
    int radius;
    public void draw() {
        System.out.println("원 그리기 메소드");
    }
};
```

추상 메소드 draw()가 실제 메소드로 구현한다.

중간 점검

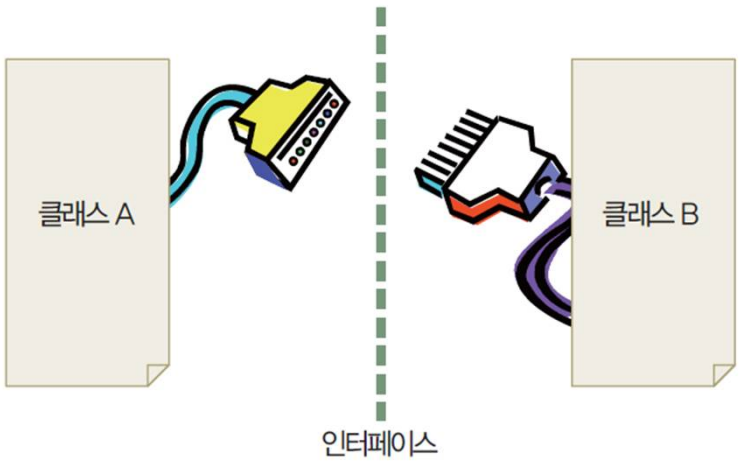
중간점검

1. 추상 클래스의 주된 용도는 무엇인가?
2. 추상 클래스는 일반 메소드를 포함할 수 있는가?
3. 추상 클래스를 상속받으면 반드시 추상 메소드를 구현하여야 하는가?

인터페이스란?

- **인터페이스는 abstract 메서드만 있는 클래스. 필드는 없다.**
- **인터페이스를 왜 쓰는가?**
  - 구현(자식) 클래스에서 **메서드의 Overriding을 강제**하고자 할 때

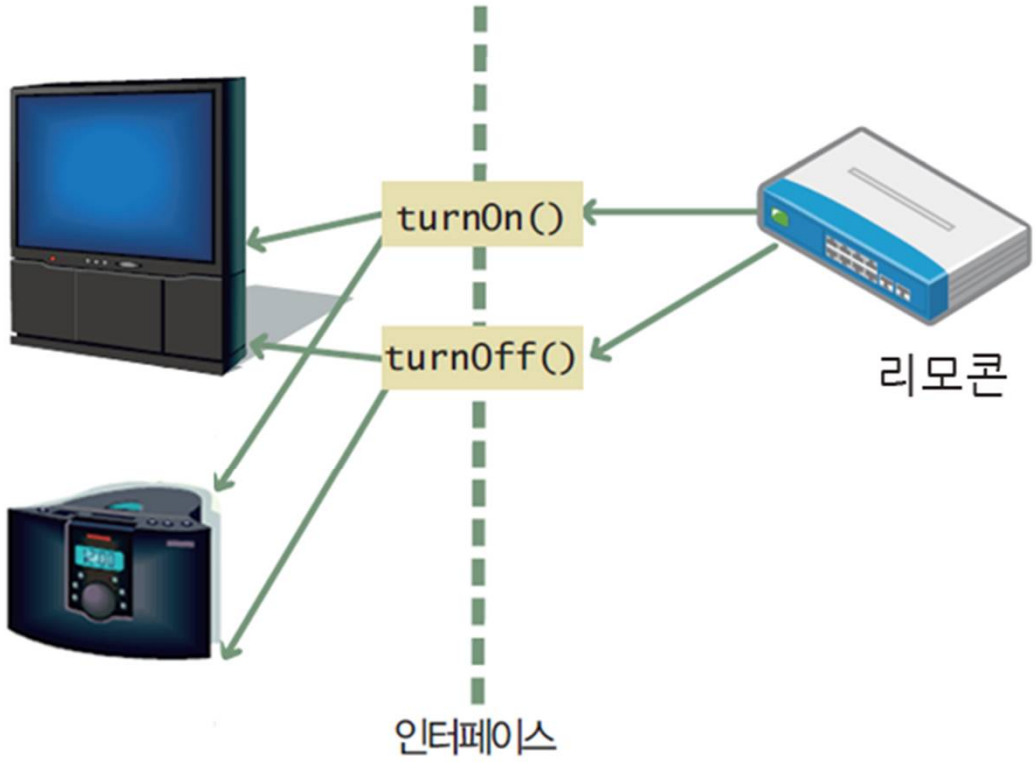
- 인터페이스 용도
  - 인터페이스는 클래스 간의 상호 작용을 나타낸다.



인터페이스란?

인터페이스란?

인터페이스를 왜 쓰는가?



홈네트워크 예제

인터페이스 선언과 구현

- 인터페이스(interface): abstract 메소드들로만 이루어진다.

  - 인터페이스(interface) 선언

```
public interface 인터페이스_이름 {
    반환형   추상 메소드1(...);
    반환형   추상 메소드2(...);
    ...
}
```

인터페이스 안에는 추상 메소드들이 정의된다.

  - 인터페이스(interface) 구현

```
public class 클래스_이름 implements 인터페이스_이름 {
    반환형   추상 메소드1(...) {
        .....
    }
    반환형   추상 메소드2(...) {
        .....
    }
}
```

인터페이스를 구현하는 클래스는 추상 메소드의 몸체를 구현하여야 한다.

인터페이스 리모콘 on/off 예제

```
public interface RemoteControl {
    // 추상 메소드 정의
    public void turnOn();        // 가전 제품을 켠다.
    public void turnOff();       // 가전 제품을 끈다.
}
```

```
public class Television implements RemoteControl {
    public void turnOn()
    {
        // 실제로 TV의 전원을 켜기 위한 코드가 들어 간다.
        ...
    }
    public void turnOff()
    {
        // 실제로 TV의 전원을 끄기 위한 코드가 들어 간다.
        ...
    }
}
```

```
public class  Fridge  implements RemoteControl {
    public void turnOn()
    {
        // 실제로 TV의 전원을 켜기 위한 코드가 들어 간다.
        ...
    }
    public void turnOff()
    {
        // 실제로 TV의 전원을 끄기 위한 코드가 들어 간다.
        ...
    }
}
```

인터페이스 리모콘 on/off 예제

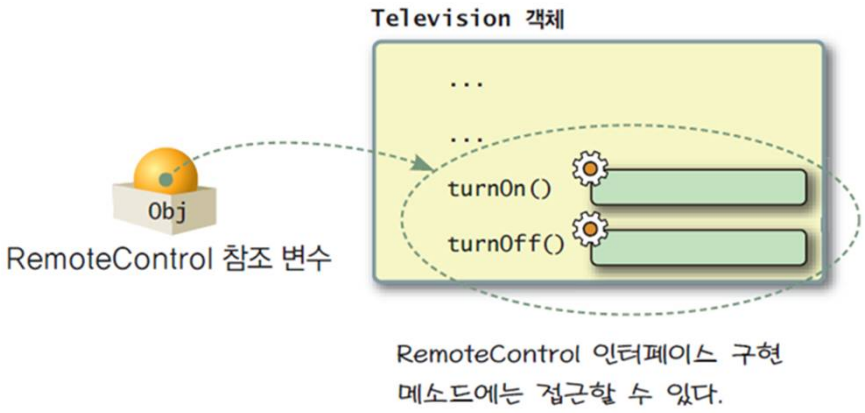
RemoteControlTest.java

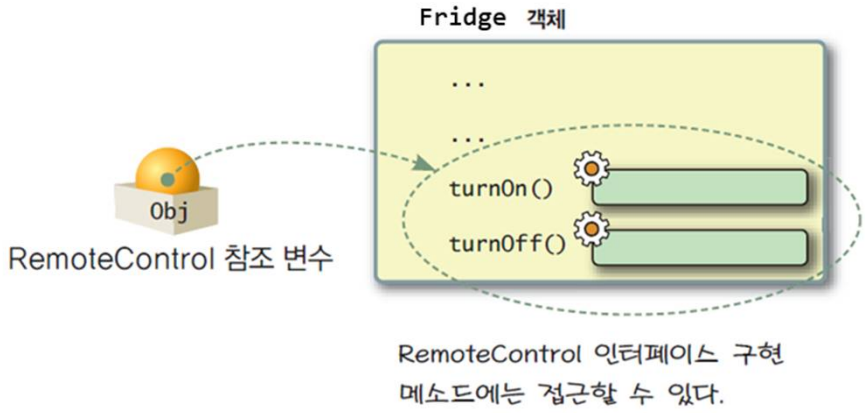
```
RemoteControl obj = new Television();
obj.turnOn();
obj.turnOff();
```

Television 객체이지만 RemoteControl 인터페이스를 구현하기 때문에 RemoteControl 타입의 변수로 가리킬 수 있다.

```
obj = new Fridge();
obj.turnOn();
obj.turnOff();
```

Fridge 객체이지만 RemoteControl 인터페이스를 구현하기 때문에 RemoteControl 타입의 변수로 가리킬 수 있다.





다중 인터페이스 구현

```
public interface SerialCommunication {
    void send();        // 시리얼 포트에 데이터를 전송한다.
    void receive();     // 시리얼 포트에서 데이터를 받는다.
}
```

```
public class Television implements RemoteControl, SerialCommunication
{
    ...
    // RemoteControl과 SerialCommunication의 메소드를 동시에 구현하여야 한다.
    public void turnON()  {  ...  }
    public void turnOFF()    {  ...  }
    public void send()   {  ...  }
    public void receive()  {  ...  }
}
```

2개의 인터페이스를 동시에 구현한다는 의미이다.

자바에서는 단일 상속만 허용.

자바에서는 인터페이스를 이용하여 다중 상속의 효과를 구현

인터페이스의 상속

```
public interface RemoteControl {
    public void turnON();          // 가전 제품을 켠다.
    public void turnOFF();         // 가전 제품을 끈다.
}
```

```
public interface AdvancedControl extends RemoteControl {
    public void volumeUp();          // 가전제품의 볼륨을 높인다.
    public void volumeDown();        // 가전제품의 볼륨을 낮춘다.
}
```

인터페이스도 다른 인터페이스를 상속 받을 수 있다

인터페이스와 다중 상속의 효과

```
class SuperA {   int x;   }
class SuperB {   int x;   }
class Sub extends SuperA, SuperB  // 만약에 다중 상속이 허용된다면
{
    ...
}
Sub obj = new Sub();
obj.x = 10;                       // obj.x는 어떤 수퍼 클래스의 x를 참조하는가?
```

```
class Sub extends Super implements Interface1, Interface2 {
    // 클래스의 정의
}
```

Super 클래스를 상속받으면서 동시에 Interface1, Interface2를 구현하는 클래스를 정의한다.

자바에서는 단일 상속만 허용.
자바에서는 인터페이스를 이용하면 다중 상속의 효과를 구현

예제-인터페이스를 이용한 다중 상속

package java16.다중상속;

```
class Shape {
        protected int x, y;
}
```
Shape.java

```
interface Drawable {
        void draw();
}
```
Drawable.java

```
public class Rectangle extends Shape implements Drawable {
        public int w, h;
        public void draw() {
            String s="";
            s = String.format("사각형 :: (x,y)=(%d,%d), (w,h)=(%d,%d)", x, y, w, h) ;
            System.out.println(s);
        }
}
```
Rectangle.java

```
public class ShapeTest {
       public static void main(String arg[]) {
           Rectangle r = new Rectangle();
           r.x = 100;    r.y = 200;
           r.w = 300;   r.h = 400;
           r.draw();
       }
}
```
ShapTest.java

중간 점검

중간점검

1. 인터페이스의 주된 용도는 무엇인가?
2. 하나의 클래스가 두 개의 인터페이스를 구현할 수 있는가?
3. 인터페이스 안에 인스턴스 변수를 선언할 수 있는가?

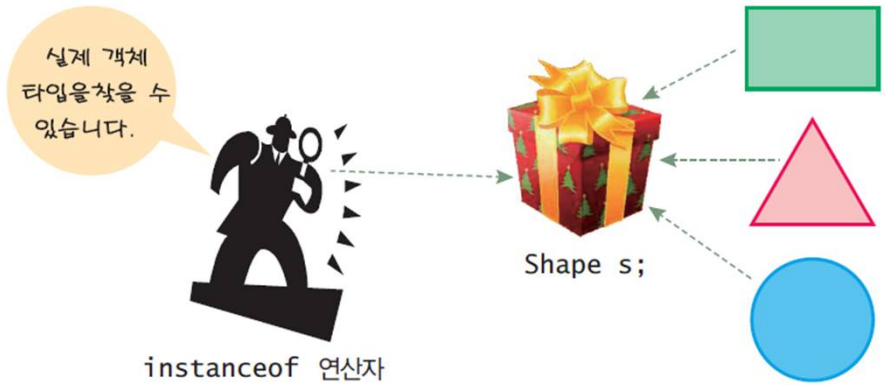
중간점검

1. 인터페이스로 참조 변수를 정의할 수 있는가?
2. 인터페이스를 이용하여서 상수를 공유하는 방법에 대해서 설명하라.
3. 자바에서는 어떻게 다중 상속을 도입하지 않고서도 다중 상속의 효과를 내는가?

객체의 타입을 알아내는 방법

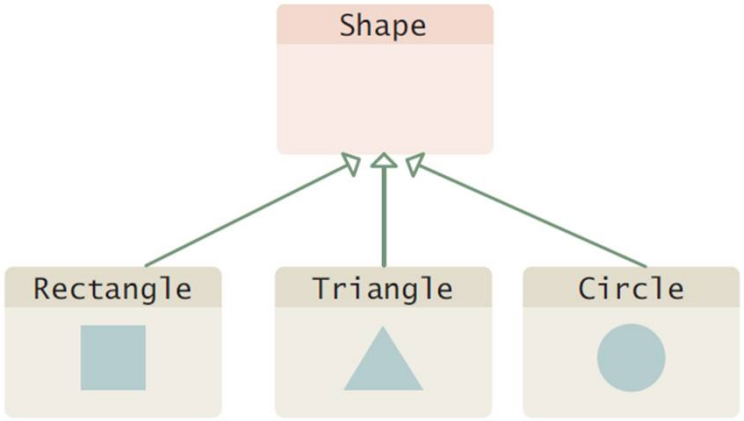
```
Shape s = getShape();
```

참조 변수 s가 가리키는 객체의 실제 타입은? Shape일 수도 있지만 Rectangle 일수도 있고 Circle 일수도 있다.




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

클래스 형변환



상향 형변환

서브 클래스 참조 변수로 수퍼 클래스 객체를 참조하는 경우.

```
Shape s;
s = new Shape();      // ❶ 당연
s = new Rectangle(); // ❷ OK
```

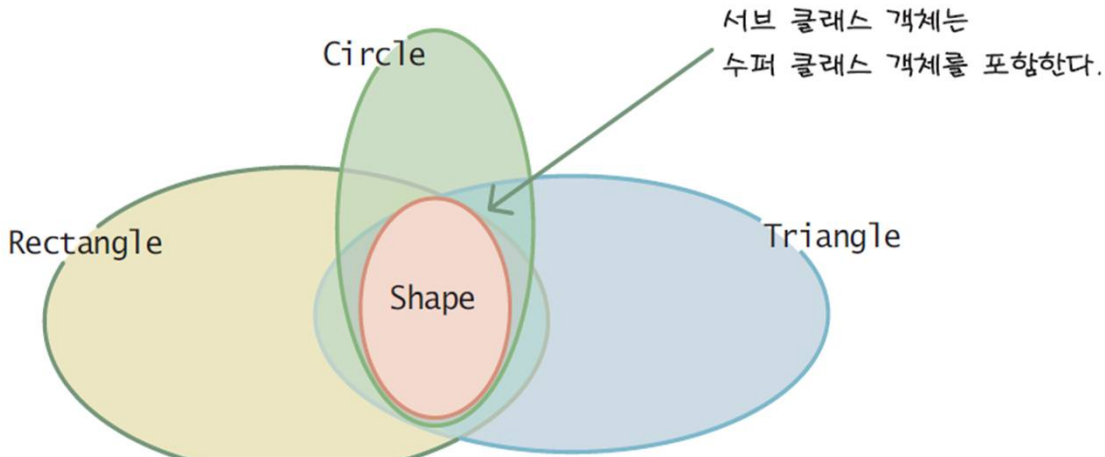
하향 형변환

수퍼 클래스 참조 변수로 서브 클래스 객체를 참조하는 경우.

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

클래스 형변환-왜 그럴까?

- 서브 클래스 객체는 수퍼 클래스 객체를 포함하고 있기 때문이다.



서브 클래스와 수퍼 클래스의 포함 관계

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

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

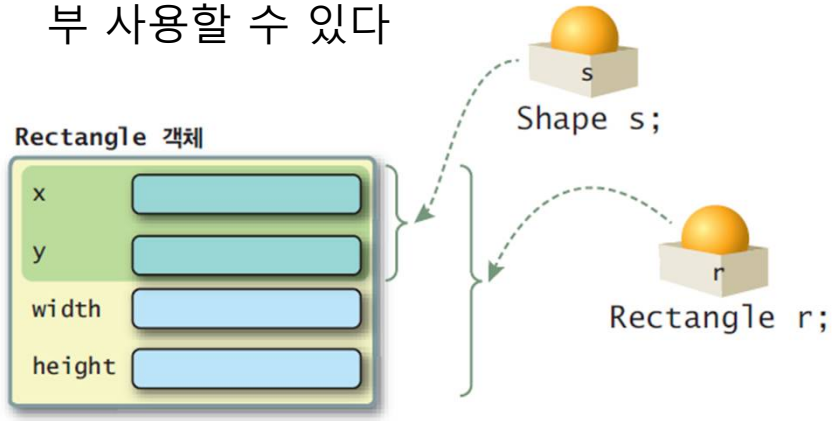
예제-클래스 형변환

```
public class Shape {
    int x, y;
}

public 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;
    }
}
```

s를 통해서는 x, y만을 사용할 수 있다. 그러나 r을 통해서는 모든 필드를 전부 사용할 수 있다




```
Rectangle r;
Shape s;
s = new Rectangle();

r = (Rectangle)s;

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

동적 바인딩

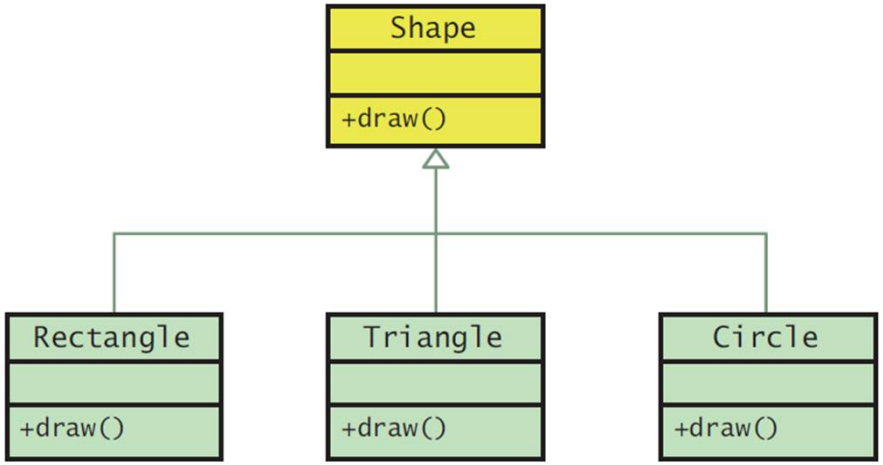


그림12-9. 도형의 UML

```
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-동적바인딩

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