作成者: mitsuya.85u@ray.ocn.ne.jp

3.1. 図形描画クラス（概念と実体、継承と多態）

概念と実体、継承と多態の実験を行います。
図形描画ソフトをイメージとしたプログラムを考えてみます。図形描画ソフトでは、直線や円や矩形を扱います。大量の図形を管理するために配列を使うことにしてみます。扱う図形が直線だけなら以下のような直線型の配列を用意すれば済みます。

```
PolyLine[]  plyLines;
```

しかし、図形描画ソフトでは円や矩形も扱います。では、以下のようにPolyLine型と独立してCircle型やRectangle型を用意してみましょう。

```
PolyLine[]    polyLines;
Circle[]      circles;
Rectangle[]   rectangles;
```

確かにこれでもOKです。では、図形をすべて再描画するときにはどうなるでしょうか。最初にすべての直線を描き、次にすべての円を描き、次にすべての矩形を描き、、、、としていかなければなりません。

でも、実際の図形描画ソフトではこのように図形の種類ごとに描画していることはありません。そうでないと、前述のような再描画の時に図形のそれぞれが分割されて再描画されてしまします。このような場合には、ユーザーが描いた順に再描画します。異なる色で複数の図形を重ねて描くと、最後に描いた図形が最後に描画されて、結果として一番上に表示されるはずです。

こうした処理を行うためには、「たくさんの図形型を一つの配列で管理」できなければいけません。これが多態のポイントとなるところです。

以下のソースコードの DrawPolyクラス の mainメソッド 部分を確認してください。

【ソースコード】DrawPoly.java

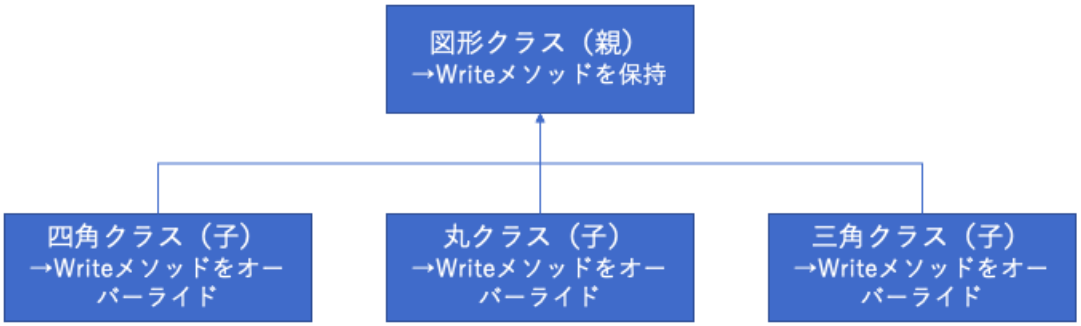
```
class Point2D{    //2次元の座標値
    double x;
    double y;
}
abstract class Shape{    //図形を描画する
    int color;              //共通フィールド
    int lineType;
    
    abstract void draw();  //共通メソッド
}
class PolyLine extends Shape{
    Point2D[] points;
    public void draw(){
        System.out.println("折れ線を描画");
    }
}
class Circle extends Shape{
    Point2D center;
    double radius;
    public void draw(){
        System.out.println("円を描画");
    }
}
class Rectangle extends Shape{
    Point2D point1;
    Point2D point2;
    public void draw(){
        System.out.println("多角形を描画");
    }
}
public class DrawPoly{
    public static void main(String[] args){
        Shape[] shapes=new Shape[5];
        shapes[0]=new PolyLine();
        shapes[1]=new Circle();
        shapes[2]=new PolyLine();
        shapes[3]=new PolyLine();
        shapes[4]=new Rectangle();
        
        for(int i=0;i<shapes.length;i++){
            shapes[i].draw();
            System.out.println("");
        }
    }
}
```

つまり、汎化クラスのメソッドをオーバライドした継承クラスのオブジェクトを汎化クラスの配列に格納することで、個々のオブジェクトのメソッドが

固有の振る舞いをする、ということです。

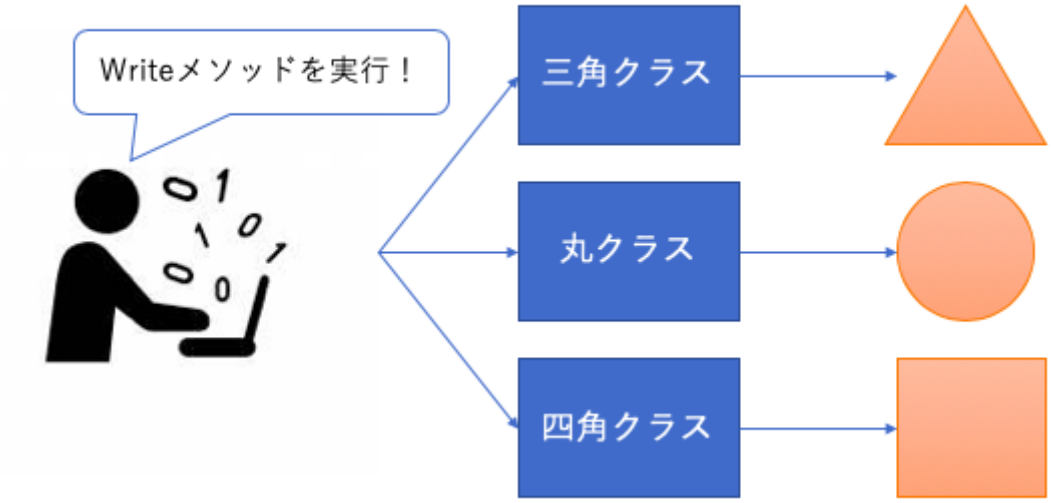
【問題】
上記コードを参考に、もうちょっと簡単に．．．

図形クラスから継承によって作られた子クラス群があるとします



各子クラスは親クラスから継承したWriteメソッドをオーバーライドしているとします。

これらの子クラス群に同じwiteメソッドを実行すると・・・以下の図のように動作します。



このように同じ呼び出し方なのに異なる動作をするという特性。
この特性から多態性/ポリモーフィズムという名前がつけられています。

では、上記の概念を実装してみてください。

クラス名：Zukeiクラス　　図形クラス

Shikakuクラス	四角クラス
Yenクラス	円クラス
Sankakuクラス	三角クラス
WritePolyクラス	神様クラス