```java
public abstract class GeometricObject {
    protected String color = "blue";
    public void printArea() {
        System.out.printf("The area of %s is %f\n", this, this.area());
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public abstract double area();         // Must be overridden
}


public class Point extends GeometricObject {
    private double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double distTo (Point p) {
        return Math.sqrt(Math.pow(this.x-p.x,2)+Math.pow(this.y-p.y,2));
    }
    public double area() {
        return 0;
    }
    public double getX() {
        return this.x;
    }
    public double getY() {
        return this.y;
    }
}
```

```
public class Circle extends GeometricObject {
    private Point center;
    private double radius;
    private static double maxRadius = −1;
    private static Circle maxCircle = null;
    public Circle(Point center, double radius) throws Exception {
        if (radius < 0)
            throw new Exception ("Circle with negative radius.");
        circleNoException(center, radius);
    }
    private void circleNoException(Point center, double radius) {
        this.center = center;
        this.radius = radius;
        this.color = "pink";
        if (radius > maxRadius) {
            maxRadius = radius;
            maxCircle = this;
        }
    }
    public Circle(double radius) throws Exception {
        this(new Point(0.,0.), radius);
    }
    public Circle(Point center) { // No exception possible
        circleNoException(center, 1.0);
    }
    public Circle() {             // No exception possible
        circleNoException(new Point(0.,0.), 1.0);
    }
    public Point getCenter() {
        return this.center;
    }
    public double getRadius() {
        return this.radius;
    }
    public double area () {
        return Math.PI * radius * radius;
    }
    public static Circle largestCircle() {
        if (maxRadius < 0)
            System.out.println("No circles created, so maximum circle is null");
        return maxCircle;
    }
    public boolean equals(Object obj) {
        if (obj instanceof Circle)
            return radius == ((Circle)obj).radius; // Only radius counts
        return false;
    }
}
```

```
public class Quadrilateral extends GeometricObject { private Point p1,
    p2, p3, p4; public Quadrilateral (Point p1, Point p2, Point p3,
    Point p4) { // should check these points for a convex
    quadrillateral in this order this.p1 = p1; this.p2 = p2; this.p3 =
    p3; this.p4 = p4; } public double area() { // diagonal gives 2
    triangles; use semiperimeter formula double diag = p1.distTo(p3);
    double s12 = p1.distTo(p2); double s23 = p2.distTo(p3); double s34
    = p3.distTo(p4); double s41 = p4.distTo(p1); double semi1 = (s12 +
    s23 + diag) / 2; double semi2 = (diag + s34 + s41) / 2; return
    Math.sqrt(semi1 * (semi1-s12) * (semi1-s23) * (semi1-diag)) +
    Math.sqrt(semi2 * (semi2-s34) * (semi2-s41) * (semi2-diag)); } }


public class Rhombus extends Quadrilateral {
    private double sideLength;
    public Rhombus (Point p1, Point p2, Point p3, Point p4) throws Exception {
        super(p1, p2, p3, p4);
        double s12 = p1.distTo(p2);
        if (s12!=p2.distTo(p3) || s12!=p3.distTo(p4) || s12!=p4.distTo(p1))
            throw new Exception ("Rhombus with unequal sides.");
        sideLength = s12;
    }
    public Rhombus (Point p, double sideLength, double theta) {
        super (p,
               new Point(p.getX()+sideLength*Math.cos(theta),
                         p.getY()+sideLength*Math.sin(theta)),
               new Point(p.getX()+sideLength*(Math.cos(theta)+1.),
                         p.getY()+sideLength*Math.sin(theta)),
               new Point(p.getX()+sideLength, p.getY()));
        this.sideLength = sideLength;
    }
    public double getSideLength() {
        return sideLength;
    }
}



public class Rectangle extends Quadrilateral {
    private double width, height;
    public Rectangle(Point p1, Point p2, Point p3, Point p4) {
        super(p1, p2, p3, p4);
        width = p1.distTo(p2);
        height = p2.distTo(p3);
        // should check that all angles are right angles via slopes neg recip
        // or check both widths equal, both heights equal, both diags equal
    }
    public Rectangle(Point p1, Point p3) { // Sides parallel to the axes
        super(p1, new Point(p3.getX(),p1.getY()),
              p3, new Point(p1.getX(),p3.getY()));
        width  = Math.abs(p1.getX()-p3.getX());
        height = Math.abs(p1.getY()-p3.getY());
        // Nothing to check
    }
    public double getWidth() {
        return width;
    }
    public double getHeight() {
        return height;
    }
    public double area() {
        return width * height;
    }
}
```

```
public class TestQuad {
    public static void main (String[] args) throws Exception {
        Point origin = new Point(0.0,0.0);
        Point p1 = new Point(1.0,0.0);
        Point p2 = new Point(1.0,1.0);
        Point p3 = new Point(0.0,1.0);
        Point p4 = new Point(5.0,5.0);
        Quadrilateral quad1 = new Quadrilateral(origin, p3, new Point(5.,0.),
                                        new Point(5.,-8.));
        Rectangle rect1 = new Rectangle (origin,p1,p2,p3);
        Rectangle rect2 = new Rectangle (origin, new Point(1.0,4.0));
        Rhombus rhom1;
        try {
            rhom1 = new Rhombus(origin,origin,p2,p3);
        }
        catch (Exception ex) {
            System.out.printf("rhom1 error: %s  Using unit square.\n",
                              ex.getMessage());
            rhom1 = new Rhombus (origin, 1.0, Math.PI/2.0);
        }
        p1.printArea();
        printColor(rhom1);      // Polymorphic call (section 11.7)
        printColor(rect1);      // Polymorphic call
        quad1.printArea();      // Inherited method (section first try)
        rect2.printArea();      // Inherited method
        myPrintArea(rhom1);     // Dynamic binding/dispatch (section second try)
        myPrintArea(p4);        // Dynamic binding/dispatch
        quad1 = rect1;          // Implicit casting (section 11.9)
        quad1.printArea();      // Dynamic binding/dispatch
        Circle c1 = new Circle();
        Circle c2 = new Circle(2.);
        Circle c3 = new Circle();
        System.out.println(c3.equals(c1));
        Circle c = Circle.largestCircle();
        c.printArea();
        System.out.println(c.getColor());
        GeometricObject geo1 = new Rectangle(origin, p2);
        GeometricObject geo2 = new Rhombus(origin, new Point(3,4),
                                    new Point(8,4), new Point(5,0));
        System.out.println(geo1.area());
    }
    public static void printColor(GeometricObject geoObj) {
        System.out.printf("The color of %s is %s\n",
                          geoObj.toString(), geoObj.getColor());
    }
    public static void myPrintArea(GeometricObject geoObj) {
        System.out.printf("My area of %s is %f\n",
                          geoObj.toString(), geoObj.area());
    }
}
```