1. Refer to the P7A handout. Write a boolean-valued instance method `noCrosses` for the class `closedPolyline` that yields true if the polyline does not cross itself.

2. Refer to the P7A handout. Write a constructor `polygon(point P0, point P1, point P2, point P3)` for the class `closedPolyline` that creates a polyline with vertices `P0`, `P1`, `P2`, and `P3`. Set the reference point to `(0,0)` and assume that `P0`, `P1`, `P2`, and `P3` are distinct. The polyline that is created should have the property that it has no crossovers. (Before you make `P0`, `P1`, `P2`, and `P3` the relative vertices in that order, you should check to make sure that the line segment connecting `P0` and `P1` does not intersect the line segment that connects `P2` and `P3`. If that is not the case then you will have to “reorder” how `P0`, `P1`, `P2`, and `P3` are “mapped into” the relative vertex array.)

3. Refer to the P7B handout. Write a boolean-valued instance method `seeHorsey` for the class `chess` that returns true if a rook is threatened by a knight.

4. Refer to the P7B handout. Assume that the chessboard has red and black tiles and that the upper left tile is red. Is there a connection between the color of a tile and the sum of its row and column index? (a) Write an integer-valued instance method `redThreat` for the class `chess` that returns the number of threatened red tiles. (b) Write a boolean-valued instance method `bishopNoBump` for the class `chess` that returns true if there is at most one bishop on a red tile and at most one bishop on a black tile.

5. Suppose `T` is a given `Triangle` object. Write a Java fragment that assigns to a boolean variable `singleQuad` the value true if all three vertices are in the same quadrant and false otherwise. Assume that none of `T`'s vertices are on the x or y axis.

6. Assume that `T1` is a `rightTriangle` object and `T2` is a `Triangle3D` object. Write a Java fragment that assigns to the boolean variable `B` the value true if the area of `T1` is smaller than the area of `T2`.


   ```java
   Triangle[] T = new Triangle[m]
   ```

all of these triangles.
// An instance of this class is a triangle.
public class Triangle
{
    protected point P0, P1, P2; // The triangle’s vertices

    // Constructor for a triangle with vertices v0, v1, v2.
    public Triangle(point v0, point v1, point v2) { }

    // Constructor for equilateral triangle with
    // center cent and radius r.
    public Triangle(point cent, double r) { }

    // Constructor for the empty triangle
    public Triangle(){}

    // Yields the area of this triangle.
    public double Area(){}

    // Yields a 3-by-2 array whose rows house the x and y
    // coordinates of the triangle’s vertices.
    public double[][] getVertices(){}
}

// An instance of this class is a right triangle.
public class rightTriangle extends Triangle
{
    // Constructor for right triangle with 90 degree
    // angle at P0 and legs with length a and b.
    public rightTriangle(point Q, double a, double b) { }

    // The length of the hypotenuse of this triangle.
    public double hypot(){}

    // The area of this triangle.
    public double Area(){}
}

// An instance of this class is a triangle with a z-coordinate to be
// considered its distance from the observer.
import java.awt.*;
public class Triangle3D extends Triangle
{
    protected double z; // Distance to observer.
    protected Color c; // Color

    // Constructor for triangle with vertices v0, v1, v2, and
    // observer distance zVal.
    public Triangle3D(point v0, point v1, point v2, double zVal, Color cVal){}

    // Constructor for equilateral triangle with radius r center cent, and
    // observer distance zVal.
    public Triangle3D(point cent, double r, double zVal, Color cVal){}

    // Yields true if this triangle is closer to the
    // observer than T.
    public boolean closerThan(Triangle3D T){}

    // Yields true if this triangle is smaller in area than T.
    public boolean smallerThan(Triangle3D T){}

    // Yields the color of this triangle
    public Color get_c(){}

    // Permutes the components in T so that the triangles range from closest to furthest away.
    public static void sort(Triangle3D[] T){}
}