```
//Imported class to create Point2D objects that represent Triangle corners
import java.awt.geom.Point2D;

/**
 * This class represents a geometric triangle object. This class has only one
 * constructor with 6 inputs, which represent 3 coordinate, which is used to
 * create the corners of the triangle being constructed. After the creation
 * of the triangle the user has access to request the point objects with
 * getter methods. This class has methods that also properties of the
 * triangle, such as side length, angle (in degrees), perimeter, area, if its
 * equilateral, if its a right triangle, centroid, and incenter coordinates.
 */
public class Triangle
{
  //This value will be used to compare double values, to make sure they are
  //similar enough to assume equality.
  private final double CLOSE_ENOUGH_ERROR = 0.00001;
  //Point2D objects that represent the corners of a triangle.
  private Point2D cornerOne;
  private Point2D cornerTwo;
  private Point2D cornerThree;

  /**
   * This is a constructor for a Triangle object. This constructor takes in
   * 6 parameters, which include the x and y coordinate values for the
   * corner coordinates of the Triangle object being created.
   *
   * @param x1 x-coordinate of corner 1
   * @param y1 y-coordinate of corner 1
   * @param x2 x-coordinate of corner 2
   * @param y2 y-coordinate of corner 2
   * @param x3 x-coordinate of corner 3
   * @param y3 y-coordinate of corner 3
   */
  public Triangle(double x1, double y1, double x2, double y2, double x3,
                  double y3)
  {
    //Initialize the Point2D objects to their respective coordinate inputs.
    cornerOne = new Point2D.Double(x1, y1);
    cornerTwo = new Point2D.Double(x2, y2);
    cornerThree = new Point2D.Double(x3, y3);
  }

  /**
   * Returns the length of side 1, from the first to the second corner, as a
   * double value.
   *
   * @return Length of side 1
   */
  public double getSideLength1()
  {
    //Call the private method calcSideLength() to get double value
```

```
  double sideLength = calcSideLength(cornerOne, cornerTwo);
  return sideLength;
}

/**
 * Returns the length of side 2, from the second to the third corner, as a
 * double value.
 *
 * @return Length of side 2
 */
public double getSideLength2()
{
  //Call the private method calcSideLength() to get double value
  double sideLength = calcSideLength(cornerTwo, cornerThree);
  return sideLength;
}

/**
 * Returns the length of side 3, from the third to the first corner, as a
 * double value.
 *
 * @return Length of side 3
 */
public double getSideLength3()
{
  //Call the private method calcSideLength() to get double value
  double sideLength = calcSideLength(cornerThree, cornerOne);
  return sideLength;
}

/**
 * Getter for cornerOne Point2D Object
 * @return Point2D Object for first corner of triangle
 */
public Point2D getCornerOne()
{
  return cornerOne;
}

/**
 * Getter for cornerTwo Point2D Object
 * @return Point2D Object for second corner of triangle
 */
public Point2D getCornerTwo()
{
  return cornerTwo;
}

/**
 * Getter for cornerThree Point2D Object
 * @return Point2D Object for third corner of triangle
 */
```

```
public Point2D getCornerThree()
{
  return cornerThree;
}

/**
 * Returns the angle (in degrees) of current Triangle object's first corner
 * as a double value. This method will return a -1000.0 value if the
 * requested target corner is out of the range [1, 2, 3].
 * @return angle of requested corner
 */
public double getAngle(int targetCorner)
{
  //Return the corner angle requested depending on user input.
  if (targetCorner == 1)
  {
    return calcCornerAngle(cornerOne, cornerTwo, cornerThree);
  }
  else if (targetCorner == 2)
  {
    return calcCornerAngle(cornerTwo, cornerOne, cornerThree);
  }
  else if (targetCorner == 3)
  {
    return  calcCornerAngle(cornerThree, cornerOne, cornerTwo);
  }

  //Return a -1000.0 value to represent an exception/error if not in range.
  return -1000.0;
}

/**
 * Returns the perimeter of current Triangle object.
 * @return Perimeter value as a double
 */
public double getPerimeter()
{
  //Calculate the perimeter
  double perimeter = getSideLength1() + getSideLength2() + getSideLength3();
  return perimeter;
}

/**
 * Returns the area of current Triangle object using Heron's
 * formula as a double value.
 * @return Area of triangle as a double value
 */
public double getArea()
{
  //Utilize Heron's formula to get the area of a triangle
  //mathwarehouse.com/geometry/triangles/area/herons-formula-triangle-area.php
```

```
  //Values required for Heron's formula.
  double halfPerim = getPerimeter() / 2; //Half perimeter value
  double side1 = getSideLength1();
  double side2 = getSideLength2();
  double side3 = getSideLength3();

  //Heron's formula for calculating area with half perimeter and side values
  double area =
        Math.sqrt(halfPerim * (halfPerim - side1) * (halfPerim - side2)
              * (halfPerim - side3));
  return area;
}

/**
 * Returns true if current Triangle object is an equilateral triangle, and
 * false if not.
 * @return Boolean value for an equilateral triangle check
 */
public boolean isEquilateral()
{
  //If a triangle is equilateral, then it has 3 60 degree angles.

  //For loop that will loop twice to check if any of those two angles are
  //not 60 degrees, if they are not this method will return false.
  for (int cornerCounter = 1; cornerCounter <= 2; cornerCounter++)
  {
    //Calculate difference for equivalency check
    double difference = Math.abs(60 - getAngle(cornerCounter));
    //If difference is greater than error range, this angle not 60 degrees
    if (difference > CLOSE_ENOUGH_ERROR)
    {
      //Return false as a non-60 degree angle has been found.
      return false;
    }
  }
  //If no non-60 values are found return true as it is equilateral.
  return true;
}

/**
 * Returns true if current Triangle object is a right triangle (has a 90
 * degree angle corner)
 * @return Boolean value for a right triangle check
 */
public boolean isRightTriangle()
{
  //For loop that checks if any corner angle is 90 degrees
  for (int cornerCounter = 1; cornerCounter <= 3; cornerCounter++)
  {
    //Calculate difference for equivalency check
    double difference = Math.abs(90 - getAngle(cornerCounter));
    //If difference is less than error range, angle is 90 degrees.
```

```
      if (difference < CLOSE_ENOUGH_ERROR)
      {
        //Return true as angle is 90 degrees, thus a right triangle.
        return true;
      }
    }
    //Return false if no right-angle has been found.
    return false;
}

/**
 * Returns the coordinates of current Triangle object as a Point2D object.
 * @return Coordinates for the centroid of current Triangle
 */
public Point2D getCentroid()
{
    //Centroid formula: formulas.tutorvista.com/math/centroid-formula.html

    //Values required for centroid formula mentioned above
    double xCoordinateSum =
            cornerOne.getX() + cornerTwo.getX() + cornerThree.getX();
    double yCoordinateSum =
            cornerOne.getY() + cornerTwo.getY() + cornerThree.getY();

    //X coordinate of the centroid
    double centroidX = xCoordinateSum / 3.0;
    //Y coordinate of the centroid
    double centroidY = yCoordinateSum / 3.0;

    //Create a Point2D object with the x and y values for the centroid
    Point2D centroidLocation = new Point2D.Double(centroidX, centroidY);
    //Return Point2D object
    return centroidLocation;
}

/**
 * Returns the incenter coordinates of as a Point2D object.
 * @return Coordinates of the incenter as a Point2D object
 */
public Point2D getIncenter()
{
    //Incenter formula: www.mathopenref.com/coordincenter.html

    //Get the side length of the opposing side of the angle
    double sideA = getSideLength2();
    double sideB = getSideLength3();
    double sideC = getSideLength1();

    //Calculate the x coordinate value with formula mentioned above
    double xValue =
            (sideA * cornerOne.getX() + sideB * cornerTwo.getX()
                    + sideC * cornerThree.getX()) / getPerimeter();
```

```
  //Calculate the y coordinate value with formula mentioned above
  double yValue =
        (sideA * cornerOne.getY() + sideB * cornerTwo.getY()
              + sideC * cornerThree.getY()) / getPerimeter();

  //Create a Point2D object that represents the incenter
  Point2D incenter = new Point2D.Double(xValue, yValue);
  //Return Point2D object
  return incenter;
}

/**
 * Returns the length between two specified corners of current Triangle
 * object.
 * @param firstCorner Point2D object representing the first target corner
 * @param secondCorner Point2D object representing the second target corner
 * @return Length of the requested side
 */
private double calcSideLength(Point2D firstCorner,
                        Point2D secondCorner)
{
  //Get the x and y value coordinates of each point
  double x1 = firstCorner.getX();
  double y1 = firstCorner.getY();
  double x2 = secondCorner.getX();
  double y2 = secondCorner.getY();

  //Calculate the distance through pythagorean formula with Math.hypot
  double distance = Math.hypot(x1 - x2, y1 - y2);
  //Return the distance
  return distance;
}

/**
 * Returns the degree of the requested angle. This method requires three
 * Point2D object inputs to calculate the required angle through an
 * implementation of law of cosines.
 * @param targetCornerA Point2D corner object of target angle (A)
 * @param otherCornerB Point2D corner object of other corner (B)
 * @param otherCornerC Point2D corner object of other corner (C)
 * @return Degree value (as a double) of target angle
 */
private double calcCornerAngle(Point2D targetCornerA, Point2D otherCornerB,
                         Point2D otherCornerC)
{

  //Required values for the law of cosines

  //Side length of each opposing side of the angle requested
  double sideLengthAngleA = calcSideLength(otherCornerB,
        otherCornerC);
```

```
    double sideLengthAngleB = calcSideLength(targetCornerA, otherCornerC);
    double sideLengthAngleC = calcSideLength(targetCornerA, otherCornerB);

    //Side lengths above are squared for calculations below
    double sideLengthA2 = Math.pow(sideLengthAngleA, 2);
    double sideLengthB2 = Math.pow(sideLengthAngleB, 2);
    double sideLengthC2 = Math.pow(sideLengthAngleC, 2);

    //Law of cosines mathworld.wolfram.com/LawofCosines.html
    double cosA = (sideLengthB2 + sideLengthC2 - sideLengthA2) /
          (2 * sideLengthAngleB * sideLengthAngleC);

    //Return the angle as a positive number in degree format
    double angle = Math.abs(Math.toDegrees(Math.acos(cosA)));
    //Return the angle calculated
    return angle;
  }
}
```