```
00001:
00002: import java.util.Scanner;
00003: import java.awt.*;
00004:
00005: /**
00006:  * This program asks for four points and tells whether its square, rectangle,
00007:  * kite, etc also pops up graph if points between 0 and 10.
00008:  *
00009:  * @author WahabEhsan
00010:  */
00011: public class Quadrilateral {
00012:
00013:     /**
00014:      * Main method prints the intro and runs the interFace method
00015:      *
00016:      * @param args
00017:      */
00018:     public static void main(String[] args) {
00019:
00020:         System.out.println("\t\t\tQuadrilateral Program");
00021:         System.out.println("Enter the integer x, y coordinates for 4 points that form a simple\n"
00022:                 + "quadrilateral, where Point 1 is connected to Point 2, Point 2 is\n"
00023:                 + "connected to Point 3, Point 3 is connected to Point 4, and Point 4\n"
00024:                 + "is connected to Point 1. This program will tell you if the Quadrilateral is\n"
00025:                 + "a Square, Rectangle, Parallelogram, Kite, Rhombus or Other shape.\n"
00026:                 + "If all x,y coordinates are between 0 and 10, your shape will be displayed!\n");
00027:
00028:         userInterface();
00029:
00030:     }
00031:
00032:     /**
00033:      * Prompts the user for x and y for four points, then makes a point object
```

```
00034:      * for each point also if points between 0 and 10, makes a graph pop up.
00035:      */
00036:     public static void userInterface() {
00037:
00038:         Scanner input = new Scanner(System.in);
00039:
00040:         System.out.print("Point 1 x y: ");
00041:         handling(input); //Runs method to see if input valid
00042:         int x1 = input.nextInt();
00043:         handling(input);//Runs method to see if input valid
00044:         int y1 = input.nextInt();
00045:         Point p1 = new Point(x1, y1);
00046:         System.out.print("Point 2 x y: ");
00047:         handling(input);//Runs method to see if input valid
00048:         int x2 = input.nextInt();
00049:         handling(input);//Runs method to see if input valid
00050:         int y2 = input.nextInt();
00051:         Point p2 = new Point(x2, y2);
00052:         System.out.print("Point 3 x y: ");
00053:         handling(input);//Runs method to see if input valid
00054:         int x3 = input.nextInt();
00055:         handling(input);//Runs method to see if input valid
00056:         int y3 = input.nextInt();
00057:         Point p3 = new Point(x3, y3);
00058:         System.out.print("Point 4 x y: ");
00059:         handling(input);//Runs method to see if input valid
00060:         int x4 = input.nextInt();
00061:         handling(input);//Runs method to see if input valid
00062:         int y4 = input.nextInt();
00063:         Point p4 = new Point(x4, y4);
00064:
00065:         if (x1 <= MAX && x2 <= MAX && x3 <= MAX && x4 <= MAX
00066:                 && y1 <= MAX && y2 <= MAX && y3 <= MAX && y4 <= MAX
```

```
00067:                 && x1 >= MIN && x2 >= MIN && x3 >= MIN && x4 >= MIN
00068:                 && y1 >= MIN && y2 >= MIN && y3 >= MIN && y4 >= MIN) {
00069:             Drawing.drawShape(p1, p2, p3, p4);
00070:         } //if points greater than 10 or less then 0, doesn't bring up graph
00071:         //Every point has its seperate statement
00072:
00073:         process(p1, p2, p3, p4); //calls the process method whihc prints the shape
00074:
00075:     }
00076:     /**
00077:      * Max value of point for graph to be displayed
00078:      */
00079:     public static final int MAX = 10;
00080:     /**
00081:      * Min value of point for graph to be displayed
00082:      */
00083:     public static final int MIN = 0;
00084:
00085:     /**
00086:      * Prints the type of shape after getting feed back from a method that
00087:      * corresponds
00088:      *
00089:      * @param p1 Point one
00090:      * @param p2 Point two
00091:      * @param p3 Point three
00092:      * @param p4 Point four
00093:      */
00094:     public static void process(Point p1, Point p2, Point p3, Point p4) {
00095:         if (isSquare(p1, p2, p3, p4)) {
00096:             System.out.println("\nSqaure"); //Prints Square if returned true
00097:         } else if (isRhombus(p1, p2, p3, p4)) {
00098:             System.out.println("\nRhombus");//Prints rhombus if returned true
00099:         } else if (isRectangle(p1, p2, p3, p4)) {
```

```
00100:             System.out.println("\nRectangle");//Prints rectangle if returned true
00101:         } else if (isParallelogram(p1, p2, p3, p4)) {
00102:             System.out.println("\nParallelogram");//Prints parallelogram if returned true
00103:         } else if (isKite(p1, p2, p3, p4)) {
00104:             System.out.println("\nKite");//Prints kite if returned true
00105:         } else {
00106:             System.out.println("\nOther");//Prints other if not met any other method
00107:         }
00108:     }
00109:
00110:     /**
00111:      * If input not integer, re-prompts the user
00112:      *
00113:      * @param input Scanner input from main method
00114:      */
00115:     public static void handling(Scanner input) {
00116:
00117:         while (!input.hasNextInt()) { //Shows "Invalid point" until user enters right point
00118:             System.out.print("Invalid Point: ");
00119:             input.next();
00120:         }
00121:     }
00122:
00123:     /**
00124:      * Distance Formula that finds out the distance between two points
00125:      *
00126:      * @param xa X one
00127:      * @param ya Y one
00128:      * @param xb X two
00129:      * @param yb Y two
00130:      * @return distance
00131:      */
00132:     public static double distanceFormula(int xa, int ya, int xb, int yb) {
```

```
00133: 
00134:         double x = Math.pow(xa - xb, 2); // X one minus X two squared
00135:         double y = Math.pow(ya - yb, 2);// Y one minus Y two squared
00136:         double distance = Math.sqrt(x + y);//taking sqareroot of the addition of x and y
00137: 
00138:         return distance;
00139:     }
00140: 
00141:     /**
00142:      * Tells whether the points make a right angle
00143:      *
00144:      * @param a Point starting angle
00145:      * @param b Point in the middle of angle
00146:      * @param c Point at the end of angle
00147:      * @return value of boolean depending on angle
00148:      */
00149:     public static boolean isRightAngle(Point a, Point b, Point c) {
00150:         double product = (b.x - a.x) * (c.x - b.x) + (b.y - a.y) * (c.y - b.y);
00151:         if (product == 0) { //if dot poduct is 0, return true for right angle
00152:             return true;
00153:         }
00154:         return false;
00155:     }
00156: 
00157:     /**
00158:      * Checks to see if points make a parallelogram
00159:      *
00160:      * @param p1 Point one
00161:      * @param p2 Point two
00162:      * @param p3 Point three
00163:      * @param p4 Point four
00164:      * @return value of boolean depending whether parallelogram or not
00165:      */
```

```
00166:     public static boolean isParallelogram(Point p1, Point p2, Point p3, Point p4) {
00167:         double distance1and2 = distanceFormula(p1.x, p1.y, p2.x, p2.y);
00168:         double distance1and4 = distanceFormula(p1.x, p1.y, p4.x, p4.y);
00169:         double distance2and3 = distanceFormula(p2.x, p2.y, p3.x, p3.y);
00170:         double distance3and4 = distanceFormula(p3.x, p3.y, p4.x, p4.y);
00171:         //Calls Distanceformula method with each parameters that are valid and possible
00172:
00173:         if ((distance1and2 == distance3and4) && (distance2and3 == distance1and4)) {
00174:             return true;
00175:         } //if side a and b are equal, and c and d are equal, returns true
00176:         return false;
00177:     }
00178:
00179:     /**
00180:      * Checks to see if points make a rectangle
00181:      *
00182:      * @param p1 Point one
00183:      * @param p2 Point two
00184:      * @param p3 Point three
00185:      * @param p4 Point four
00186:      * @return value of boolean depending whether rectangle or not
00187:      */
00188:     public static boolean isRectangle(Point p1, Point p2, Point p3, Point p4) {
00189:         if (isRightAngle(p1, p2, p3)
00190:                 && isRightAngle(p2, p3, p4)
00191:                 && isRightAngle(p3, p4, p1)
00192:                 && isRightAngle(p4, p1, p2)) {
00193:             return true;
00194:         }//returns true if all the angles are right angles
00195:         return false;
00196:     }
00197:
00198:     /**
```

```
00199:      * Checks to see if points make a square
00200:      *
00201:      * @param p1 Point one
00202:      * @param p2 Point two
00203:      * @param p3 Point three
00204:      * @param p4 Point four
00205:      * @return value of boolean depending whether square or not
00206:      */
00207:     public static boolean isSquare(Point p1, Point p2, Point p3, Point p4) {
00208:         double distance1and2 = distanceFormula(p1.x, p1.y, p2.x, p2.y);
00209:         double distance1and4 = distanceFormula(p1.x, p1.y, p4.x, p4.y);
00210:         double distance2and3 = distanceFormula(p2.x, p2.y, p3.x, p3.y);
00211:         double distance3and4 = distanceFormula(p3.x, p3.y, p4.x, p4.y);
00212:         //Calls Distanceformula method with each parameters that are valid and possible
00213:
00214:         if (distance1and2 == distance2and3
00215:                 && distance3and4 == distance1and4
00216:                 && distance1and2 == distance3and4
00217:                 && distance2and3 == distance1and4
00218:                 && distance1and2 == distance1and4
00219:                 && distance2and3 == distance3and4) {
00220:
00221:             if (isRightAngle(p1, p2, p3)
00222:                     && isRightAngle(p2, p3, p4)
00223:                     && isRightAngle(p3, p4, p1)) {
00224:                 return true;
00225:             }
00226:         }//returns true if all sides equal and all right angles
00227:         return false;
00228:     }
00229:
00230:     /**
00231:      * Checks to see if points make a rhombus
```

```
00232:      *
00233:      * @param p1 Point one
00234:      * @param p2 Point two
00235:      * @param p3 Point three
00236:      * @param p4 Point four
00237:      * @return value of boolean depending whether rhombus or not
00238:      */
00239:     public static boolean isRhombus(Point p1, Point p2, Point p3, Point p4) {
00240:         double distance1and2 = distanceFormula(p1.x, p1.y, p2.x, p2.y);
00241:         double distance1and4 = distanceFormula(p1.x, p1.y, p4.x, p4.y);
00242:         double distance2and3 = distanceFormula(p2.x, p2.y, p3.x, p3.y);
00243:         double distance3and4 = distanceFormula(p3.x, p3.y, p4.x, p4.y);
00244:         //Calls Distanceformula method with each parameters that are valid and possible
00245:
00246:         if (distance1and2 == distance2and3
00247:                 && distance3and4 == distance1and4
00248:                 && distance1and2 == distance3and4
00249:                 && distance2and3 == distance1and4
00250:                 && distance1and2 == distance1and4
00251:                 && distance2and3 == distance3and4) {
00252:             return true;
00253:
00254:         }//returns true if only all sides equal true
00255:         return false;
00256:     }
00257:
00258:     /**
00259:      * Checks to see if points make a kite
00260:      *
00261:      * @param p1 Point one
00262:      * @param p2 Point two
00263:      * @param p3 Point three
00264:      * @param p4 Point four
```

```
00265:      * @return value of boolean depending whether kite or not
00266:      */
00267:     public static boolean isKite(Point p1, Point p2, Point p3, Point p4) {
00268:
00269:         double distance1and2 = distanceFormula(p1.x, p1.y, p2.x, p2.y);
00270:         double distance2and3 = distanceFormula(p2.x, p2.y, p3.x, p3.y);
00271:         double distance3and4 = distanceFormula(p3.x, p3.y, p4.x, p4.y);
00272:         double distance1and4 = distanceFormula(p1.x, p1.y, p4.x, p4.y);
00273:         //Calls Distanceformula method with each parameters that are valid and possible
00274:
00275:         if ((distance1and2 == distance2and3
00276:                 && distance3and4 == distance1and4) || (distance3and4 == distance2and3
00277:                 && distance1and2 == distance1and4)) {
00278:             return true;
00279:         } //returns true only if two sides are equal to to other sides
00280:
00281:         return false;
00282:     }
00283: }
```