import com.jogamp.opengl.\*;

import com.jogamp.opengl.awt.GLCanvas;

import com.jogamp.opengl.glu.GLU;

import javax.swing.\*;

import java.io.\*;

import java.util.\*;

class ThirdGLEventListener implements GLEventListener {

private ArrayList<Pair> drawingPixels;

*/\*\**

*\* Take care of initialization here.*

*\*/*

@SuppressWarnings("DuplicatedCode")

public void init(GLAutoDrawable gld) {

GL2 gl = gld.getGL().getGL2();

/\*

\* Interface to the GLU library.

\*/

GLU glu = new GLU();

gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

gl.glViewport(-250, -150, 250, 150);

gl.glMatrixMode(GL2.*GL\_PROJECTION*);

gl.glLoadIdentity();

glu.gluOrtho2D(-250.0, 250.0, -150.0, 150.0);

}

/\*

\* Take care of drawing here.

\*/

public void display(GLAutoDrawable drawable) {

GL2 gl = drawable.getGL().getGL2();

gl.glClear(GL2.*GL\_COLOR\_BUFFER\_BIT*);

gl.glColor3d(0.44, 0.0, 0.57);

gl.glPointSize(3);

gl.glBegin(GL2.*GL\_POINTS*);

for (Pair pair : drawingPixels) {

gl.glVertex2d(pair.x, pair.y);

}

gl.glEnd();

}

public void reshape(GLAutoDrawable drawable, int x, int y, int width,

int height) {

}

@SuppressWarnings("unused")

public void displayChanged(GLAutoDrawable drawable,

boolean modeChanged, boolean deviceChanged) {

}

public void dispose(GLAutoDrawable arg0) {

}

public void setDrawingPixels(ArrayList<Pair> drawingPixels) {

this.drawingPixels = drawingPixels;

}

}

@SuppressWarnings("SuspiciousNameCombination")

class MidPointCircleCalculation {

public ArrayList<Pair> takeInputAndInitialize() throws IOException {

final PrintWriter pw = new PrintWriter(new BufferedOutputStream(System.*out*));

final BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));

pw.print("Please enter the center of the circle seperated by a space : ");

pw.flush();

StringTokenizer tk = new StringTokenizer(br.readLine());

int x = Integer.*parseInt*(tk.nextToken()), y = Integer.*parseInt*(tk.nextToken());

if (Math.*abs*(x) > 250) throw new RuntimeException("Value of x must need to be between -250 and 250");

if (Math.*abs*(y) > 150) throw new RuntimeException("Value of y must need to be between -150 and 150");

Pair center = new Pair(x, y);

pw.print("Please enter the radius of the circle : ");

pw.flush();

tk = new StringTokenizer(br.readLine());

int radius = Integer.*parseInt*(tk.nextToken());

if (radius < 0) throw new RuntimeException("Radius can't be negative! It must need to be positive integer!");

if (Math.*abs*(x) + radius > 250 || Math.*abs*(y) + radius > 150)

throw new RuntimeException("Out of the drawing pixel range!");

pw.print("Please enter the number of inner circle(s) : ");

pw.close();

tk = new StringTokenizer(br.readLine());

int numberOfInnerCircle = Integer.*parseInt*(tk.nextToken());

br.close();

if (numberOfInnerCircle < 0)

throw new RuntimeException("The number of inner circles must need to be a positive integer!");

ArrayList<Pair> drawingPixels = findPixelsForInnerCircle(radius, center, numberOfInnerCircle);

drawingPixels.addAll(findDrawingPixels(center, radius));

return drawingPixels;

}

private ArrayList<Pair> findPixelsForInnerCircle(int radiusOfTheOuterCircle,

Pair centerOfTheOuterCircle, final int NUMBER\_OF\_INNER\_CIRCLE) {

ArrayList<Pair> pixels = new ArrayList<>();

// If the number of inner circle values is equal to zero,

// then don't need to calculate anything.

// Just return an empty List.

if (NUMBER\_OF\_INNER\_CIRCLE == 0) return pixels;

final double ONE\_FULL\_REVOLUTION\_IN\_RADIANS = 2 \* Math.*PI*;

final double RADIUS\_OF\_INNER\_CIRCLE = radiusOfTheOuterCircle / 2.0;

final double DEL\_THETA = ONE\_FULL\_REVOLUTION\_IN\_RADIANS / NUMBER\_OF\_INNER\_CIRCLE;

double curThetaValueInRadians = 0.0;

while (curThetaValueInRadians < ONE\_FULL\_REVOLUTION\_IN\_RADIANS) {

int x = (int) Math.*round*((Math.*cos*(curThetaValueInRadians)

\* RADIUS\_OF\_INNER\_CIRCLE)) + centerOfTheOuterCircle.x;

int y = (int) Math.*round*((Math.*sin*(curThetaValueInRadians)

\* RADIUS\_OF\_INNER\_CIRCLE)) + centerOfTheOuterCircle.y;

pixels.addAll(findDrawingPixels(new Pair(x, y),

(int) RADIUS\_OF\_INNER\_CIRCLE));

curThetaValueInRadians += DEL\_THETA;

}

return pixels;

}

private ArrayList<Pair> findDrawingPixels(Pair center, int radius) {

ArrayList<Pair> zoneOnePixels = midPointCircle(radius);

zoneOnePixels.addAll(findPixelsForOtherZone(zoneOnePixels));

ArrayList<Pair> originalPixels = new ArrayList<>();

for (Pair pair : zoneOnePixels) {

originalPixels.add(findOriginalPixels(pair, center));

}

return originalPixels;

}

private ArrayList<Pair> findPixelsForOtherZone(ArrayList<Pair> zoneOneOriginalPixel) {

ArrayList<Pair> pixelsForOtherZones = new ArrayList<>();

for (Pair pair : zoneOneOriginalPixel) {

pixelsForOtherZones.add(findPixelForZoneZero(pair));

pixelsForOtherZones.add(findPixelForZoneTwo(pair));

pixelsForOtherZones.add(findPixelForZoneThree(pair));

pixelsForOtherZones.add(findPixelForZoneFour(pair));

pixelsForOtherZones.add(findPixelForZoneFive(pair));

pixelsForOtherZones.add(findPixelForZoneSix(pair));

pixelsForOtherZones.add(findPixelForZoneSeven(pair));

}

return pixelsForOtherZones;

}

private ArrayList<Pair> midPointCircle(int radius) {

ArrayList<Pair> pixels = new ArrayList<>();

int d = 1 - radius;

int x = 0, y = radius;

while (x <= y) {

pixels.add(new Pair(x, y));

if (d >= 0) {

y--;

d += x + x - y - y + 5;

} else {

d += x + x + 3;

}

x++;

}

return pixels;

}

private Pair findOriginalPixels(Pair zoneOnePixel, Pair originalCenter) {

return new Pair(zoneOnePixel.x + originalCenter.x,

zoneOnePixel.y + originalCenter.y);

}

private Pair findPixelForZoneZero(Pair zoneOnePixel) {

return new Pair(zoneOnePixel.y, zoneOnePixel.x);

}

private Pair findPixelForZoneTwo(Pair zoneOnePixel) {

return new Pair(-zoneOnePixel.x, zoneOnePixel.y);

}

private Pair findPixelForZoneThree(Pair zoneOnePixel) {

return new Pair(-zoneOnePixel.y, zoneOnePixel.x);

}

private Pair findPixelForZoneFour(Pair zoneOnePixel) {

return new Pair(-zoneOnePixel.y, -zoneOnePixel.x);

}

private Pair findPixelForZoneFive(Pair zoneOnePixel) {

return new Pair(-zoneOnePixel.x, -zoneOnePixel.y);

}

private Pair findPixelForZoneSix(Pair zoneOnePixel) {

return new Pair(zoneOnePixel.x, -zoneOnePixel.y);

}

private Pair findPixelForZoneSeven(Pair zoneOnePixel) {

return new Pair(zoneOnePixel.y, -zoneOnePixel.x);

}

}

class Pair {

int x, y;

public Pair(int x, int y) {

this.x = x;

this.y = y;

}

}

public class Task {

public static void main(String[] args) throws IOException {

//getting the capability object of GL2 profile

MidPointCircleCalculation midPointCircleCalculation = new MidPointCircleCalculation();

ArrayList<Pair> pixels = midPointCircleCalculation.takeInputAndInitialize();

final GLProfile profile = GLProfile.*get*(GLProfile.*GL2*);

GLCapabilities capabilities = new GLCapabilities(profile);

// The canvas

final GLCanvas glcanvas = new GLCanvas(capabilities);

ThirdGLEventListener b = new ThirdGLEventListener();

// Setting the pixels for drawing the circles

b.setDrawingPixels(pixels);

glcanvas.addGLEventListener(b);

glcanvas.setSize(400, 400);

//creating frame

final JFrame frame = new JFrame("Lab 3");

//adding canvas to frame

frame.add(glcanvas);

frame.setSize(640, 480);

frame.setVisible(true);

}

}