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 {

*/\*\**

*\* Interface to the GLU library.*

*\*/*

private GLU glu;

private ArrayList<Pair> pixels;

*/\*\**

*\* Take care of initialization here.*

*\*/*

public void init(GLAutoDrawable gld) {

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

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*);

/\*

\* put your code here

\*/

gl.glColor3d(0.80, 0.90, 0.15);

gl.glPointSize(10.0f);

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

for (Pair p : pixels) {

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

}

gl.glEnd();

}

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

int height) {

}

public void displayChanged(GLAutoDrawable drawable,

boolean modeChanged, boolean deviceChanged) {

}

public void dispose(GLAutoDrawable arg0) {

}

public void setPixels(ArrayList<Pair> pixels) {

this.pixels = pixels;

}

}

class MidPointComputation {

private int findZone(int x1, int y1, int x2, int y2) {

int zone;

// what if it's just a point.

if (x1 == x2 && y1 == y2) return 0;

int dx = x2 - x1, dy = y2 - y1;

if (dy >= 0 && dx > dy) zone = 0;

else if (dx >= 0 && dy > 0 && dy >= dx) zone = 1;

else if (dx <= 0 && dy > 0 && dy > Math.*abs*(dx)) zone = 2;

else if (dx < 0 && dy >= 0 && Math.*abs*(dx) >= dy) zone = 3;

else if (dx < 0 && dy <= 0 && Math.*abs*(dx) > Math.*abs*(dy)) zone = 4;

else if (dx <= 0 && dy < 0 && Math.*abs*(dy) >= Math.*abs*(dx)) zone = 5;

else if (dx >= 0 && dy < 0 && Math.*abs*(dy) > dx) zone = 6;

else if (dx > 0 && dy <= 0 && dx >= Math.*abs*(dy)) zone = 7;

else throw new RuntimeException("Can't determine zone!!! Fix it");

return zone;

}

@SuppressWarnings("SuspiciousNameCombination")

private Pair convertToZoneZero(int originalZone, int x, int y) {

Pair points = new Pair();

switch (originalZone) {

case 0:

points.x = x;

points.y = y;

break;

case 1:

points.x = y;

points.y = x;

break;

case 2:

points.x = y;

points.y = -x;

break;

case 3:

points.x = -x;

points.y = y;

break;

case 4:

points.x = -x;

points.y = -y;

break;

case 5:

points.x = -y;

points.y = -x;

break;

case 6:

points.x = -y;

points.y = x;

break;

case 7:

points.x = x;

points.y = -y;

break;

default:

throw new RuntimeException("Zone must need to between 0 and 7");

}

return points;

}

@SuppressWarnings("SuspiciousNameCombination")

private Pair convertToOriginalZone(int originalZone, int x, int y) {

Pair points = new Pair();

switch (originalZone) {

case 0:

points.x = x;

points.y = y;

break;

case 1:

points.x = y;

points.y = x;

break;

case 2:

points.x = -y;

points.y = x;

break;

case 3:

points.x = -x;

points.y = y;

break;

case 4:

points.x = -x;

points.y = -y;

break;

case 5:

points.x = -y;

points.y = -x;

break;

case 6:

points.x = y;

points.y = -x;

break;

case 7:

points.x = x;

points.y = -y;

break;

default:

throw new RuntimeException("Zone should be between 0 to 7!");

}

return points;

}

private ArrayList<Pair> calculateDrawingPixels(int x1, int y1, int x2, int y2) {

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

int dx, dy, d, incE, incNE, x, y;

dx = x2 - x1;

dy = y2 - y1;

d = (2 \* dy) - dx;

incE = 2 \* dy;

incNE = 2 \* (dy - dx);

y = y1;

for (x = x1; x <= x2; x++) {

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

if (d > 0) {

d = d + incNE;

y = y + 1;

} else {

d = d + incE;

}

}

return pixels;

}

private ArrayList<Pair> findDrawingPixels(boolean isLastDigit, int x1, int y1,

int x2, int y2) {

int originalZone = findZone(x1, y1, x2, y2);

Pair convertedPoints1 = convertToZoneZero(originalZone, x1, y1);

Pair convertedPoints2 = convertToZoneZero(originalZone, x2, y2);

// time to call the calculateDrawingPixels method and get all the points to for zone 1

ArrayList<Pair> pixels = calculateDrawingPixels(convertedPoints1.x, convertedPoints1.y,

convertedPoints2.x, convertedPoints2.y);

// Time to convert back the pixels to the original zone

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

for (Pair p : pixels) {

Pair originalPoints = convertToOriginalZone(originalZone, p.x, p.y);

if (isLastDigit) originalPoints.x = originalPoints.x + 130;

originalPixels.add(originalPoints);

}

return originalPixels;

}

public ArrayList<Pair> takeInputAndFindNecessaryPixels() {

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

String studentId = takeInput();

try {

Long.*parseLong*(studentId);

} catch (Exception e) {

throw new RuntimeException("Given Input is not a student ID");

}

if (studentId.length() != 8) throw new RuntimeException("Student ID must be of length 8");

int seventhDigit = Integer.*parseInt*(String.*valueOf*(studentId.charAt(6))),

eightDigit = Integer.*parseInt*(String.*valueOf*(studentId.charAt(7)));

necessaryPixels.addAll(findPixelsForDigit(false, seventhDigit));

necessaryPixels.addAll(findPixelsForDigit(true, eightDigit));

return necessaryPixels;

}

private ArrayList<Pair> findPixelsForDigit(boolean isLast, int toCheck) {

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

switch (toCheck) {

case 0:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineTwo(isLast));

pixels.addAll(getPixelsForLineFour(isLast));

pixels.addAll(getPixelsForLineFive(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

break;

case 1:

pixels.addAll(getPixelsForLineSeven(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

break;

case 2:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineFour(isLast));

pixels.addAll(getPixelsForLineFive(isLast));

break;

case 3:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

pixels.addAll(getPixelsForLineFive(isLast));

break;

case 4:

pixels.addAll(getPixelsForLineTwo(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

break;

case 5:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineTwo(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

pixels.addAll(getPixelsForLineFive(isLast));

break;

case 6:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineTwo(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineFour(isLast));

pixels.addAll(getPixelsForLineFive(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

break;

case 7:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

break;

case 8:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineTwo(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineFour(isLast));

pixels.addAll(getPixelsForLineFive(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

break;

case 9:

pixels.addAll(getPixelsForLineOne(isLast));

pixels.addAll(getPixelsForLineTwo(isLast));

pixels.addAll(getPixelsForLineThree(isLast));

pixels.addAll(getPixelsForLineSeven(isLast));

pixels.addAll(getPixelsForLineSix(isLast));

break;

default:

throw new RuntimeException("Digit must be between [0 - 9]");

}

return pixels;

}

private ArrayList<Pair> getPixelsForLineOne(boolean isLast) {

return findDrawingPixels(isLast, -110, 80, -20, 80);

}

private ArrayList<Pair> getPixelsForLineTwo(boolean isLast) {

return findDrawingPixels(isLast, -110, 80, -110, 0);

}

private ArrayList<Pair> getPixelsForLineThree(boolean isLast) {

return findDrawingPixels(isLast, -110, 0, -20, 0);

}

private ArrayList<Pair> getPixelsForLineFour(boolean isLast) {

return findDrawingPixels(isLast, -110, 0, -110, -80);

}

private ArrayList<Pair> getPixelsForLineFive(boolean isLast) {

return findDrawingPixels(isLast, -110, -80, -20, -80);

}

private ArrayList<Pair> getPixelsForLineSix(boolean isLast) {

return findDrawingPixels(isLast, -20, -80, -20, 0);

}

private ArrayList<Pair> getPixelsForLineSeven(boolean isLast) {

return findDrawingPixels(isLast, -20, 0, -20, 80);

}

private String takeInput() {

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

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

pw.print("Please enter your student ID: ");

pw.close();

try {

String inputString = new StringTokenizer(br.readLine()).nextToken();

br.close();

return inputString;

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

class Pair {

int x, y;

Pair() {

}

Pair(int x, int y) {

this.x = x;

this.y = y;

}

}

public class Task {

public static void main(String[] args) {

MidPointComputation midPointComputation = new MidPointComputation();

ArrayList<Pair> pixels = midPointComputation.takeInputAndFindNecessaryPixels();

//getting the capabilities object of GL2 profile

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

GLCapabilities capabilities = new GLCapabilities(profile);

// The canvas

final GLCanvas glcanvas = new GLCanvas(capabilities);

ThirdGLEventListener b = new ThirdGLEventListener();

// Initializing the Pixels list's for drawing

b.setPixels(pixels);

glcanvas.addGLEventListener(b);

glcanvas.setSize(400, 400);

//creating frame

final JFrame frame = new JFrame("Lab 2");

//adding canvas to frame

frame.add(glcanvas);

frame.setSize(640, 480);

frame.setVisible(true);

}

}

