САНКТ-ПЕТЕРБУРЖСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

*Кафедра Вычислительной техники*

**Лабораторная работа №4**

Выполнил:

студент II курса группы 2125

Припадчев Артём

Проверит:

Харитонова А.Е.

Санкт-Петербург

2013

**Задание:** Доработать программу из лабораторной работы №3 следующим образом. Реализовать приложение на базе Swing API, которое отображает на экране заданную область и заданные компоненты пользовательского интерфейса, с помощью которых вводятся данные о координатах точек и параметре R.

При щелчке мышкой по графику должна отображаться точка, цвет которой зависит от попадания или непопадания в область, при этом компоненты графического интерфейса должны отображать значения координат точки. При задании значений координат точки и R на графике должна также отображаться точка соответствующего цвета.

Согласно полученному варианту необходимо реализовать анимацию с использованием Java-потоков.

**Приложение должно использовать следующие элементы:**

Для задания координаты X использовать JList.

Для задания координаты У - JRadioButton.

Для задания R - JSpinner.

Для отображения координат установленной точки - JTextArea.

Элементы необходимо группировать с использованием менеджера компоновки BorderLayout.

В рамках групп необходимо использовать BoxLayout .

При изменении радиуса должна осуществляться перерисовка фигуры с сохранением масштаба.

При отрисовке области в качестве цвета фона использовать светло-желтый цвет.

Для заливки области использовать темно-зеленый цвет.

Приложение должно включать анимацию следующего вида:  после установки размер точки должен увеличиваться.

Многопоточность должна быть реализована с помощью расширения класса Thread.

**Код программы**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

public class Lab4 {

 private static float x, y, r;

 private static float MAX\_COORD = 20f;

 private static Contour contour;

 private static JPanel toolPanel;

 private static JLabel xAndy;

 public static void contourRepaint() {

 contour.repaint();

 }

 public static void main(String[] args) {

 JFrame frame = new JFrame("Lab4");

 frame.setVisible(true);

 frame.setSize(620, 720);

 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

 contour = new Contour(MAX\_COORD);

 frame.add(contour, BorderLayout.CENTER);

 contour.addMouseListener(new MouseAL());

 toolPanel = new JPanel();

 frame.add(toolPanel, BorderLayout.SOUTH);

 toolPanel.setLayout(new FlowLayout());

 // X

 try {

 String[] coordXArray = {"-4", "-3", "-2", "-1", "0", "1", "2", "3", "4"};

 JList listCoordX = new JList(coordXArray);

 listCoordX.setLayoutOrientation(JList.VERTICAL);

 listCoordX.addListSelectionListener(new JLSelectionListener());

 toolPanel.add(listCoordX);

 } catch (Exception e) {

 System.out.println("Something went wrong");

 }

 // Y

 JRadioButton rbm4 = new JRadioButton("-4");

 JRadioButton rbm3 = new JRadioButton("-3");

 JRadioButton rbm2 = new JRadioButton("-2");

 JRadioButton rbm1 = new JRadioButton("-1");

 JRadioButton rb0 = new JRadioButton("0");

 JRadioButton rb1 = new JRadioButton("1");

 JRadioButton rb2 = new JRadioButton("2");

 JRadioButton rb3 = new JRadioButton("3");

 JRadioButton rb4 = new JRadioButton("4");

 toolPanel.add(rbm4);

 toolPanel.add(rbm3);

 toolPanel.add(rbm2);

 toolPanel.add(rbm1);

 toolPanel.add(rb0);

 toolPanel.add(rb1);

 toolPanel.add(rb2);

 toolPanel.add(rb3);

 toolPanel.add(rb4);

 ButtonGroup groupOfYRadioButton = new ButtonGroup();

 groupOfYRadioButton.add(rbm4);

 groupOfYRadioButton.add(rbm3);

 groupOfYRadioButton.add(rbm2);

 groupOfYRadioButton.add(rbm1);

 groupOfYRadioButton.add(rb0);

 groupOfYRadioButton.add(rb1);

 groupOfYRadioButton.add(rb2);

 groupOfYRadioButton.add(rb3);

 groupOfYRadioButton.add(rb4);

 rbm4.addActionListener(new RBActionListener());

 rbm3.addActionListener(new RBActionListener());

 rbm2.addActionListener(new RBActionListener());

 rbm1.addActionListener(new RBActionListener());

 rb0.addActionListener(new RBActionListener());

 rb1.addActionListener(new RBActionListener());

 rb2.addActionListener(new RBActionListener());

 rb3.addActionListener(new RBActionListener());

 rb4.addActionListener(new RBActionListener());

 // R

 SpinnerNumberModel model = new SpinnerNumberModel((int) MAX\_COORD / 2, 1, (int) MAX\_COORD / 2, 1);

 JSpinner js = new JSpinner(model);

 js.addChangeListener(new JSSpinnerListener());

 toolPanel.add(js);

 //Button Check

 JButton check = new JButton("Check");

 check.addActionListener(new BtnCheckListener());

 toolPanel.add(check);

 // Coordinates

 xAndy = new JLabel("");

 toolPanel.add(xAndy);

 xAndy.setText("x = " + x + ", y = " + y);

 }

 static class MouseAL extends MouseAdapter {

 public void mouseReleased(MouseEvent e) {

 x = contour.pixelsToX(e.getX());

 y = contour.pixelsToY(e.getY());

 contour.drawing(x, y);

 xAndy.setText("x = " + String.format("%.2f", x) + ", y = " +

 String.format("%.2f", y));

 }

 }

 static class JLSelectionListener implements ListSelectionListener {

 @Override

 public void valueChanged(ListSelectionEvent e) {

 JList<String> cb = (JList<String>) e.getSource();

 x = Integer.parseInt(cb.getSelectedValue());

 }

 }

 static class RBActionListener implements ActionListener {

 @Override

 public void actionPerformed(ActionEvent e) {

 JRadioButton b = (JRadioButton) e.getSource();

 if (b.isSelected()) {

 y = Integer.parseInt(b.getText());

 }

 }

 }

 static class JSSpinnerListener implements ChangeListener {

 public void stateChanged(ChangeEvent e) {

 JSpinner source = (JSpinner) e.getSource();

 r = (Integer) source.getValue();

 contour.setR(r);

 }

 }

 static class BtnCheckListener implements ActionListener {

 public void actionPerformed(ActionEvent e) {

 contour.drawing(x, y);

 xAndy.setText("x = " + String.format("%.2f", x) + ", y = " + String.format("%.2f", y));

 }

 }

}

public abstract class Figure {

 public abstract boolean hitInArea(Mark mark);

 public abstract boolean hitOnBorder(Mark mark);

}

public class FRectangle extends Figure {

 private float x, y;

 private float width, height;

 public float getX() {

 return x;

 }

 public float getY() {

 return y;

 }

 public float getWidth() {

 return width;

 }

 public float getHeight() {

 return height;

 }

 public FRectangle(float x, float y, float width, float height) {

 this.x = x;

 this.y = y;

 this.width = width;

 this.height = height;

 }

 @Override

 public boolean hitInArea(Mark mark) {

 float xMark = mark.getX();

 float yMark = mark.getY();

 if ((xMark > x) && (xMark < (x + width)) && (yMark < y) && (yMark > (y + height))) {

 return true;

 } else

 return false;

 }

 @Override

 public boolean hitOnBorder(Mark mark) {

 float xMark = mark.getX();

 float yMark = mark.getY();

 if (((xMark == x) && (yMark <= y) && (yMark >= y + height)) || ((xMark == x + width) && (yMark <= y) && (yMark >= y + height)) ||

 ((yMark == y) && (xMark >= x) && (xMark <= x + width)) || ((yMark == y + height) && (xMark >= x) && (xMark <= x + width))) {

 return true;

 } else

 return false;

 }

}

public class FTriangle extends Figure {

 private Mark A;

 private Mark B;

 private Mark C;

 public Mark getA() {

 return A;

 }

 public Mark getB() {

 return B;

 }

 public Mark getC() {

 return C;

 }

 public FTriangle(Mark A, Mark B, Mark C) {

 this.A = A;

 this.B = B;

 this.C = C;

 }

 @Override

 public boolean hitInArea(Mark mark) {

 float ABCSquare = Square(A, B, C);

 float ABDSquare = Square(A, B, mark);

 float BCDSquare = Square(B, C, mark);

 float CADSquare = Square(C, A, mark);

 float Sum = ABDSquare + BCDSquare + CADSquare;

 if (Math.abs((ABCSquare - Sum)) <= 0.01)

 return true;

 else

 return false;

 }

 private float GetSide(float x1, float y1, float x2, float y2) {

 return (float) Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

 }

 private float Square(Mark A, Mark B, Mark C) {

 float a = GetSide(A.getX(), A.getY(), B.getX(), B.getY());

 float b = GetSide(B.getX(), B.getY(), C.getX(), C.getY());

 float c = GetSide(C.getX(), C.getY(), A.getX(), A.getY());

 float p = a + b + c;

 p /= 2.0f;

 float square = (float) Math.sqrt(p \* (p - a) \* (p - b) \* (p - c));

 return square;

 }

 @Override

 public boolean hitOnBorder(Mark mark) {

 float ABDSquare = Square(A, B, mark);

 float BCDSquare = Square(B, C, mark);

 float CADSquare = Square(C, A, mark);

 if ((ABDSquare <= 0.01 || BCDSquare <= 0.01 || CADSquare <= 0.01) && hitInArea(mark))

 return true;

 else

 return false;

 }

}

public class FQuarterOfCircle extends Figure {

 private float radius;

 private int startAngle, arcAngle;

 private Quarter quarter;

 public float getRadius() {

 return radius;

 }

 public int getStartAngle() {

 return startAngle;

 }

 public int getArcAngle() {

 return arcAngle;

 }

 public FQuarterOfCircle(float radius, Quarter quarter) {

 this.radius = radius;

 this.quarter = quarter;

 switch (quarter) {

 case First:

 startAngle = 0;

 break;

 case Second:

 startAngle = 90;

 break;

 case Third:

 startAngle = 180;

 break;

 case Fourth:

 startAngle = 270;

 break;

 }

 arcAngle = 90;

 }

 @Override

 public boolean hitInArea(Mark mark) {

 float xMark = mark.getX();

 float yMark = mark.getY();

 boolean inside = false;

 switch (quarter) {

 case First: {

 if (((xMark != 0) && (yMark != 0)) && (xMark > 0) && (yMark > 0) && ((Math.pow(xMark, 2) + Math.pow(yMark, 2)) < Math.pow(radius, 2)))

 inside = true;

 break;

 }

 case Second: {

 if (((xMark != 0) && (yMark != 0)) && (xMark < 0) && (yMark > 0) && ((Math.pow(xMark, 2) + Math.pow(yMark, 2)) < Math.pow(radius, 2)))

 inside = true;

 break;

 }

 case Third: {

 if (((xMark != 0) && (yMark != 0)) && (xMark < 0) && (yMark < 0) && ((Math.pow(xMark, 2) + Math.pow(yMark, 2)) < Math.pow(radius, 2)))

 inside = true;

 break;

 }

 case Fourth: {

 if (((xMark != 0) && (yMark != 0)) && (xMark > 0) && (yMark < 0) && ((Math.pow(xMark, 2) + Math.pow(yMark, 2)) < Math.pow(radius, 2)))

 inside = true;

 break;

 }

 }

 if (inside) return true;

 else

 return false;

 }

 @Override

 public boolean hitOnBorder(Mark mark) {

 float xMark = mark.getX();

 float yMark = mark.getY();

 boolean onBorder = false;

 if (xMark == 0 || yMark == 0 || (Math.pow(xMark, 2)) + Math.pow(yMark, 2) == Math.pow(radius, 2)) {

 switch (quarter) {

 case First: {

 if ((xMark >= 0) && (yMark >= 0) && (Math.pow(xMark, 2)) + Math.pow(yMark, 2) <= Math.pow(radius, 2))

 onBorder = true;

 break;

 }

 case Second: {

 if ((xMark <= 0) && (yMark >= 0) && (Math.pow(xMark, 2)) + Math.pow(yMark, 2) <= Math.pow(radius, 2))

 onBorder = true;

 break;

 }

 case Third: {

 if ((xMark <= 0) && (yMark <= 0) && (Math.pow(xMark, 2)) + Math.pow(yMark, 2) <= Math.pow(radius, 2))

 onBorder = true;

 break;

 }

 case Fourth: {

 if ((xMark >= 0) && (yMark <= 0) && (Math.pow(xMark, 2)) + Math.pow(yMark, 2) <= Math.pow(radius, 2))

 onBorder = true;

 break;

 }

 }

 }

 if (onBorder) return true;

 else

 return false;

 }

}

public enum Quarter {

 First,

 Second,

 Third,

 Fourth

}

import javax.swing.\*;

import java.awt.\*;

import java.util.LinkedList;

public class Contour extends JPanel {

 // the emerging point parameters

 private int x, y;

 private int pointRadius;

 private boolean inside;

 private LinkedList<Mark> points;

 private float contourRadius;

 private float graphWidth, graphHeight;

 LinkedList<EmergingPointDraw> listEmergingPointDraw = new LinkedList<EmergingPointDraw>();

 LinkedList<Figure> listFigure;

 public Contour(float R) {

 points = new LinkedList<Mark>();

 graphWidth = 10.0f;

 graphHeight = 10.0f;

 if (R > graphHeight - 6 || R > graphWidth - 6) {

 graphHeight = R + 6;

 graphWidth = R + 6;

 contourRadius = R / 2;

 } else

 contourRadius = R;

 }

 // px and py are pixels

 private void addPoint(int px, int py) {

 points.add(new Mark(pixelsToX(px), pixelsToY(py)));

 }

 // x and y are coordinates

 private void addPoint(float x, float y) {

 points.add(new Mark(x, y));

 }

 private boolean hitInArea(Mark mark) {

 boolean inside = false;

 int countHitOnBorder = 0;

 for (Figure figure : listFigure) {

 if (figure.hitInArea(mark)) {

 inside = true;

 }

 if (figure.hitOnBorder(mark))

 countHitOnBorder++;

 }

 if (countHitOnBorder == 1) inside = false;

 else if (countHitOnBorder > 1)

 inside = true;

 return inside;

 }

 // creates an animation for a newly added point

 public void drawing(float xx, float yy) {

 x = xToPixels(xx);

 y = yToPixels(yy);

 listEmergingPointDraw.add(new EmergingPointDraw(x, y));

 listEmergingPointDraw.get(listEmergingPointDraw.size() - 1).start();

 }

 public void setR(float r) {

 if (r > graphHeight || r > graphWidth)

 contourRadius = graphHeight >= graphWidth ? graphWidth - 6 : graphHeight - 6;

 else

 contourRadius = r;

 repaint();

 }

 public int xToPixels(float x) {

 int m = getWidth() / 2;

 return (int) (m \* (1 + x \* 2 / graphWidth));

 }

 public int yToPixels(float y) {

 int m = getHeight() / 2;

 return (int) (m \* (1 - y \* 2 / graphHeight));

 }

 public float pixelsToX(int p) {

 float m = getWidth() / 2;

 return (p - m) / m \* graphWidth / 2;

 }

 public float pixelsToY(int p) {

 float m = getHeight() / 2;

 return (m - p) / m \* graphHeight / 2;

 }

 private void drawEmergingPoint(Graphics g) {

 if (!listEmergingPointDraw.isEmpty()) {

 for (int i = 0; i < listEmergingPointDraw.size(); i++) {

 if (listEmergingPointDraw.get(i).getRadius() >= 10) {

 addPoint(listEmergingPointDraw.get(i).getX(), listEmergingPointDraw.get(i).getY());

 listEmergingPointDraw.remove(i);

 } else {

 Color c;

 if (hitInArea(new Mark(pixelsToX((listEmergingPointDraw.get(i).getX())), pixelsToY(listEmergingPointDraw.get(i).getY()))))

 c = Color.GREEN;

 else

 c = Color.RED;

 g.setColor(c);

 g.fillOval(listEmergingPointDraw.get(i).getX() - listEmergingPointDraw.get(i).getRadius() / 2, listEmergingPointDraw.get(i).getY() - listEmergingPointDraw.get(i).getRadius() / 2, listEmergingPointDraw.get(i).getRadius(), listEmergingPointDraw.get(i).getRadius());

 }

 }

 }

 }

 private void drawGraphBody(Graphics g) {

 listFigure = new LinkedList<Figure>();

 listFigure.add(new FRectangle(0, 0, contourRadius, -contourRadius));

 listFigure.add(new FQuarterOfCircle(contourRadius, Quarter.Third));

 listFigure.add(new FTriangle(new Mark(0, 0), new Mark(-contourRadius, 0), new Mark(0, contourRadius / 2)));

 g.setColor(new Color(45, 179, 0));

 for (Figure figure : listFigure) {

 if (figure instanceof FRectangle)

 g.fillRect(xToPixels(((FRectangle) figure).getX()), yToPixels(((FRectangle) figure).getY()),

 xToPixels(((FRectangle) figure).getWidth()) - xToPixels(0), yToPixels(((FRectangle) figure).getHeight()) - yToPixels(0));

 if (figure instanceof FTriangle) {

 int[] xPoints = {xToPixels(((FTriangle) figure).getA().getX()), xToPixels(((FTriangle) figure).getB().getX()), xToPixels(((FTriangle) figure).getC().getX())};

 int[] yPoints = {yToPixels(((FTriangle) figure).getA().getY()), yToPixels(((FTriangle) figure).getB().getY()), yToPixels(((FTriangle) figure).getC().getY())};

 g.fillPolygon(xPoints, yPoints, 3);

 }

 if (figure instanceof FQuarterOfCircle) {

 g.fillArc(xToPixels(-((FQuarterOfCircle) figure).getRadius()),

 yToPixels(((FQuarterOfCircle) figure).getRadius()),

 2 \* (xToPixels(((FQuarterOfCircle) figure).getRadius()) - xToPixels(0)) + 1,

 2 \* (yToPixels(-((FQuarterOfCircle) figure).getRadius()) - yToPixels(0)) + 1,

 ((FQuarterOfCircle) figure).getStartAngle(),

 ((FQuarterOfCircle) figure).getStartAngle());

 }

 }

 // coordinate lines

 g.setColor(Color.BLACK);

 g.drawLine(0, yToPixels(0), this.getWidth(), yToPixels(0));

 g.drawLine(xToPixels(0), this.getHeight(), xToPixels(0), 0);

 }

 private void drawAddedPoints(Graphics g) {

 for (Mark p : points)

 if (hitInArea(p)) {

 g.setColor(Color.GREEN);

 g.fillOval(xToPixels(p.getX()) - 5,

 yToPixels(p.getY()) - 5, 10, 10);

 } else {

 g.setColor(Color.RED);

 g.fillOval(xToPixels(p.getX()) - 5,

 yToPixels(p.getY()) - 5, 10, 10);

 }

 }

 @Override

 public void paintComponent(Graphics g) {

 super.paintComponent(g);

 setBackground(new Color(255, 230, 153));

 drawGraphBody(g);

 drawAddedPoints(g);

 drawEmergingPoint(g);

 }

}

public class EmergingPointDraw extends Thread {

 private final int x;

 private final int y;

 private int radius;

 public int getX() {

 return x;

 }

 public int getY() {

 return y;

 }

 public int getRadius() {

 return radius;

 }

 public EmergingPointDraw(int x, int y) {

 this.x = x;

 this.y = y;

 this.radius = 0;

 }

 @Override

 public void run() {

 while (this.radius <= 10) {

 this.radius++;

 try {

 Thread.sleep(50);

 } catch (Exception e) {

 System.out.println("Error in run method EmergingPointDraw class");

 }

 Lab4.contourRepaint();

 }

 }

}

public class Mark {

 private final float x;

 private final float y;

 public Mark(float x, float y) {

 this.x = x;

 this.y = y;

 }

 public float getX() { return x; }

 public float getY() { return y; }

}

**Вывод:** в процессе работы были изучены базовые принципы работы с Swing API, реализацией потоков с помощью класса Thread, классом-слушателем и классом-событием.