Федеральное государственное автономное образовательное учреждение высшего образования
Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет прикладной информатики и компьютерных технологий

Программирование Интернет Приложений

Лабораторная работа №4

Вариант: 2013

Группа: P3218

Студент: Петкевич Константин

Преподаватель: Гаврилов Антон Валерьевич

г. Санкт-Петербург

2016

Текст Задания: 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.

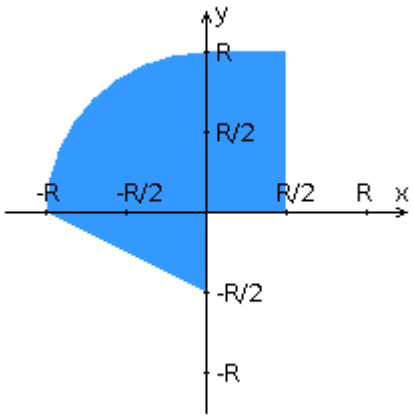
Согласно полученному варианту необходимо реализовать анимацию с использованием Java-потоков.

Приложение должно использовать следующие элементы:

- Для задания координаты X использовать JComboBox.
- Для задания координаты У - JCheckBox.
- Для задания R - JSlider.
- Для отображения координат установленной точки - JTextField.
- Элементы необходимо группировать с использованием менеджера компоновки GridLayout.
- В рамках групп необходимо использовать BoxLayout .
- При изменении радиуса должна осуществляется перерисовка точек с пересчетом масштаба.
- При отрисовке области в качестве цвета фона использовать белый цвет.
- Для заливки области использовать коричневый цвет.

Приложение должно включать анимацию следующего вида:
точки на области должны плавно исчезать через 7 секунд после установки
Условие запуска анимации: вход в область одной из точек при изменении радиуса.

Многопоточность должна быть реализована с помощью расширения класса Thread.



Вывод: в результате выполнения данной лабораторной работой я построил пользовательский интерфейс на основе Swing API.Многопоточное программирование несет в себе много преимуществ, но и задает отдельные сложности. Например, возможно получение ситуации race-conditions. Опыт полученный в результате выполнения важен и является базовым для написания полноценных многопоточных программ с пользовательским интерфейсом.

```
public class PointController implements ActionListener, MouseListener {
 JTextField x, y;
 Graph graph;
 public PointController(JTextField x, JTextField y, Graph graph) {
  this.x = x;
  this.y = y;
  this.graph = graph;
 }

 @Override
 public void actionPerformed(ActionEvent e) {
  //set button is pressed(or such event fired)
  graph.addPoint(Double.parseDouble(x.getText()), Double.parseDouble(y.getText()));
 }

 @Override
 public void mouseClicked(MouseEvent e) {
  int x = e.getX();
  int y = e.getY();
  x -= graph.getWidth() / 2;
  y -= graph.getHeight() / 2;
  y *= -1;
  double realX = (double)x / graph.step;
  double realY = (double)y / graph.step;
  this.x.setText(Double.toString(Math.round(realX * 10000) / 10000.0));
  this.y.setText(Double.toString(Math.round(realY * 10000) / 10000.0));
  graph.addPoint(realX, realY);
 }


public class RController implements ChangeListener {
 JTextField r;
 Graph graph;
 public RController(JTextField r, Graph graph) {
  this.r = r;
  this.graph = graph;
 }

 @Override
 public void stateChanged(ChangeEvent event) {
  JSlider source = (JSlider)event.getSource();
  r.setText(Integer.toString(source.getValue()));
  graph.setR(source.getValue());
 }
}


public class XController implements ListSelectionListener {
 private JTextField x;

 public XController(JTextField x) {
  this.x = x;
 }

 @Override
 public void valueChanged(ListSelectionEvent e) {
  x.setText(
     ((JList)(e.getSource())).getSelectedValue().toString()
  );
```

```
  }
}

class YController implements ActionListener {
  private JTextField y;
  private int direction = 1;
  public YController(JTextField y) {
    super();
    this.y = y;
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    Double value = new Double(y.getText());
    if (command.equals("Y -= 1")) {
      value -= 1;
    } else if (command.equals("Y += 1")) {
      value += 1;
    }
    y.setText(value.toString());
  }
}

public class CoordinatesPanel extends JPanel {
  private JTextField y;
  private JTextField x;
  private YController yController;
  private XController xController;
  private RController rController;
  private Double[] presetX = {1.0, 2.0, 3.0, 4.0, 5.0};
  static final Integer
      MIN_R = 0,
      MAX_R = 50,
      INIT_R = MAX_R / 2;

  public CoordinatesPanel(Graph graph) {
    //…создание интерфейса
  }

  @Override
  public Dimension getPreferredSize() {
    return new Dimension(200, 200);
  }
  public JTextField getXField() {
    return x;
  }
  public JTextField getYField() {
    return y;
  }
}

class Graph extends JPanel{
  private static final Color
    BACKGROUND_COLOR = Color.WHITE,
    LINE_COLOR = Color.black,
    FIGURE_COLOR = Color.getHSBColor(25,86,55).darker().darker().darker().darker(),
    POINT_IN_COLOR = Color.magenta,
```

```
  POINT_OUT_COLOR = Color.black;

  private Shape shape;
  public int step = 5;
  public int r = 25;
  private int h = 300;
  private int w = 300;

  private static HashSet<GraphPoint2D> points = new HashSet<GraphPoint2D>();
  public Graph() {
    super();
    this.shape = recountShape();
    new Thread(new PointAppearAnimation(this, points)).start();
  }

 private Shape recountShape() {
  return new Shape(new Area[] {
        new Area(graphPoint2D -> graphPoint2D.Y() <= r // rect
              && graphPoint2D.Y() <= 0,
              0, r),
        new Area(graphPoint2D -> graphPoint2D.Y() <= -graphPoint2D.X() + r/2//triangle
              && graphPoint2D.Y() >= 0,
              0, r/2),
        new Area(graphPoint2D -> graphPoint2D.Y() <= Math.pow(Math.pow(-r/2, 2) - Math.pow(graphPoint2D.X(), 2), 0.5) // circ
              && graphPoint2D.Y() >= 0,
              -r/2, 0)
  });
 }


  public int getStep() {
    return this.step;
  }

  @Override
  public void paintComponent(Graphics g) {
    this.h = this.getHeight();
    this.w = this.getWidth();
    this.step = h /2  / (this.r != 0 ? this.r : 1);
    super.paintComponent(g);
    drawGraph(g);
  }

  public void setR(int r) {
    this.r = r;
    this.shape = recountShape();
    this.repaint();
  }

  @Override
  public Dimension getPreferredSize() {
    return new Dimension(400, 400);
  }

  public void addPoint(double x, double y) {
    GraphPoint2D point = new GraphPoint2D(x, y);
    Color pointColor = this.shape.contains(point) ? POINT_IN_COLOR : POINT_OUT_COLOR;
    point.color = pointColor;
    point.makeTransparent();
```

```
  points.add(point);
  this.repaint();
 }

 private void drawPoints(Graphics g) {
  Iterator<GraphPoint2D> iter = points.iterator();
  while (iter.hasNext()) {
   GraphPoint2D point = iter.next();
   Boolean contains = this.shape.contains(point);
   Color pointColor = contains ? POINT_IN_COLOR : POINT_OUT_COLOR;
   point.changeColor(pointColor); //set color point to be black or magenta
   drawPoint(g, point);
  }
 }

 private void drawPoint(Graphics g, GraphPoint2D point) {
  g.setColor(point.color);
  g.fillRect((int)(point.X() * step + (this.getWidth() / 2)), (int)(-point.Y() * step + (this.getHeight() / 2)), 3, 3);
 }

 private void drawSteps(Graphics g) {
  g.setColor(LINE_COLOR);
  for (int i = w / 2; i <= w; i += step) {
   g.drawLine(
      i,
      h / 2 + 2,
      i,
      h/ 2 - 2);

   g.drawLine(
      (w - i),
      h / 2 + 2,
      (w - i),
      h / 2 - 2);
  }

  for (int i = h / 2; i <= h; i += step) {
   g.drawLine(
      w / 2 + 1,
      i,
      w / 2 - 1,
      i);

   g.drawLine(
      w / 2 + 1,
      (h - i),
      w / 2 - 1,
      (h - i));
  }
 }

private void drawGraph(Graphics g) {
 g.setColor(BACKGROUND_COLOR);
 g.fillRect(0, 0, w, h);
 g.setColor(FIGURE_COLOR);
 //rect
 g.fillRect(w/2, h/2,  r*step, r*step);

 Polygon triangle = new Polygon();
```

```
  triangle.addPoint(w/2 + r*step / 2, h/2 );
  triangle.addPoint(w/2 , h/2 );
  triangle.addPoint(w/2  , h/2 - r*step / 2);
  g.fillPolygon(triangle);

  g.fillArc(w/2- (int)(r * step / 2), h/2 - (int)(r * step / 2), (int)(r * step), (int)(r * step), 90, 90);
  g.setColor(LINE_COLOR);

  //Coordinate lines
  g.drawLine(w / 2, 0, w / 2,  h);
  g.drawLine(0, h / 2, w, h / 2);
  drawSteps(g);
  //Draw points
  drawPoints(g);
 }


  class PointAppearAnimation extends Thread {
   int timer;
   Graph graph;
   HashSet<GraphPoint2D> animatedPoints;
   public PointAppearAnimation(Graph graph, HashSet<GraphPoint2D> animatedPoints) {
    this.timer = 7000;
    this.animatedPoints = animatedPoints;
    this.graph = graph;
   }

   @Override
   public void run() {
    int steps = 10;
    float alpha = 0;
    float alphaInc = 1.0f / steps;
    while (true) {
     for (GraphPoint2D point2D: this.animatedPoints) {
      if (point2D.color.getAlpha() != 0) point2D.decAlpha(alphaInc);
     }
     this.graph.repaint();
     try {
      Thread.sleep(timer / steps);
     } catch (InterruptedException e) {
      e.printStackTrace();
     }
    }
   }
  }
 }
```