```java
//ArchimedeanSpiralApplet.java
//Programmer: Randy Miller
//Created: April 3, 2015

import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.event.*;

import static java.lang.Math.*;

public class ArchimedeanSpiralApplet extends JApplet
{
    JButton button1 = new JButton("Start"),
            button2 = new JButton("Stop");

    JSlider speedBar;
    PaintingPanel centerPanel;
    JPanel southPanel = new JPanel();
    int width, height;



    public void init()
    {

        width  =  Integer.parseInt(getParameter("width"));
        height =  Integer.parseInt(getParameter("height"));

        System.err.printf("width = %d, height = %d%n", width, height);

        centerPanel = new PaintingPanel();

        add(centerPanel, BorderLayout.CENTER);
        centerPanel.setToolTipText("The spiral is given parametrically by x = k \u0398 cos \u0398, y = k \u0398 sin \u0398");

        southPanel.add(new JLabel("Rotation Speed: "));
        speedBar = new JSlider(JSlider.HORIZONTAL, 5, 50, 5);
        speedBar.addChangeListener
          /*(event ->   {
                            centerPanel.timer.setDelay((int)(1000 / speedBar.getValue()));
                            centerPanel.timer.restart();
                        }
```

```
            );*/
            (new ChangeListener()
                {
                    public void stateChanged(ChangeEvent event)
                    {
                        centerPanel.timer.setDelay((int)(1000 / speedBar.getValue()));
                        centerPanel.timer.restart();
                        button1.setEnabled(false);
                        button2.setEnabled(true);
                    }
                }
            );
        southPanel.add(speedBar);
        button1.addActionListener
              /*(event -> {centerPanel.timer.start(); button1.setEnabled(false);});*/
                (new ActionListener()

                    {
                        public void actionPerformed(ActionEvent event)
                        {
                            centerPanel.timer.restart();
                            button1.setEnabled(false);
                            button2.setEnabled(true);
                        }
                    }
                );
        southPanel.add(button1);
        button2.addActionListener
          /*(event -> {centerPanel.timer.stop(); button2.setEnabled(false);});*/
            (new ActionListener()

                {
                    public void actionPerformed(ActionEvent event)
                    {
                        centerPanel.timer.stop();
                        button1.setEnabled(true);
                        button2.setEnabled(false);
                    }
                }
            );
        southPanel.add(button2);
        button1.setEnabled(false);
        button2.setEnabled(true);
        southPanel.setPreferredSize(new Dimension(width, 50));
        add(southPanel, BorderLayout.SOUTH);
```

```java

        setVisible(true);
    }

    class PaintingPanel extends JComponent //implements Runnable
    {

        final double PI = acos(-1.0);


        final int NUMBER_TURNS = 6,
                  POINTS_PER_TURN = 60,
                  PREFERRED_WIDTH = /*500*/ Math.min(width, ArchimedeanSpiralApplet.this.height - 50);

        double maximumRadius = (double)PREFERRED_WIDTH / 2;


        int updateInterval = 200;

        double xPoints[] = new double[NUMBER_TURNS * POINTS_PER_TURN];
        double yPoints[] = new double[NUMBER_TURNS * POINTS_PER_TURN];
        double xGuides[] = new double[NUMBER_TURNS * POINTS_PER_TURN];
        double yGuides[] = new double[NUMBER_TURNS * POINTS_PER_TURN];

        double rotation = 0.0;

        javax.swing.Timer timer;

        public PaintingPanel()
        {
            System.err.printf("PREFERRED_WIDTH is %d%n", PREFERRED_WIDTH);
            System.err.printf("maximumRadius is %f%n", maximumRadius);

            /*initialize path arrays:
              radius = k * theta for an Archimedean spiral;
              @ theta = NUMBER__TURNS * 2 * PI && radius = maximumRadius,
              -> k == maximumRadius / (NUMBER_OF_TURNS * 2 * PI). A point in either
              array with index i corresponds to a theta =  (2 * PI * i) / POINTS_PER_TURN,
              the points in xPoints being given by radius * Math.cos(theta) and
              the points in yPoints being given by radius * Math.sin(theta)*/

            for (int i = 1; i <= NUMBER_TURNS * POINTS_PER_TURN; i++)
            {
                xPoints[i - 1]
                    = ((i * maximumRadius) / (POINTS_PER_TURN * NUMBER_TURNS)) * cos(2.0 * PI * i / POINTS_PER_TURN);
```

```
                yPoints[i - 1]
                    = ((i * maximumRadius) / (POINTS_PER_TURN * NUMBER_TURNS)) * sin(2.0 * PI * i / POINTS_PER_TURN);


            /*initialize arrays encoding external guide points for use with the GeneralPath::quadTo()
              function; the guide points are fixed by the intersection of the tangents to any two adjoining
              points of the spiral, which for the short arcs here intended is approximated by a circle
              (sufficiently far from the center of the spiral)*/

                double  x_i = (i == 1 ? 0 : xPoints[i - 2]),
                        y_i = (i == 1 ? 0 : yPoints[i - 2]),
                        x_f = xPoints[i - 1],
                        y_f = yPoints[i - 1];

                if (y_i != 0 && y_f != 0)
                {
                    xGuides[i - 1] =
                        ((y_f + (x_f * x_f / y_f)) - (y_i + (x_i * x_i / y_i))) / ((x_f / y_f) - (x_i / y_i));
                    yGuides[i - 1] = y_i - (x_i / y_i) * (xGuides[i - 1] - x_i);
                }
                else if (y_i == 0)
                {
                    xGuides[i - 1] = x_i;
                    yGuides[i - 1] = y_f - (x_f / y_f) * (xGuides[i - 1] - x_f);
                }
                else
                {
                    xGuides[i - 1] = x_f;
                    yGuides[i - 1] = y_i - (x_i / y_i) * (xGuides[i - 1] - x_i);
                }

            }

            //new Thread(this).start();

            /*Swing Timer with a lambda:
            new javax.swing.Timer(200, event ->
                    {
                        rotation -= PI / 8;
                        repaint();
                    }
                ).start();
            */

            //adding a reference for use with start and stop buttons:
```

```
        timer = new javax.swing.Timer(updateInterval, new ActionListener()
                        {
                            public void actionPerformed(ActionEvent event)
                            {
                                rotation -= PI / 8;
                                repaint();
                            }
                        }
                    );
        timer.start();
    }

    /*I am going to fix the image size in this app, because there is too much mathematical calculation to
      put onto the animating call to repaint()*/

    public Dimension getPreferredSize()
    {
        return  new Dimension(PREFERRED_WIDTH, PREFERRED_WIDTH);
    }


    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2D = (Graphics2D)g;

        g2D.setColor(new Color(24, 0, 96));
        g2D.fillRect(0,  0,  getWidth(),  getHeight());

        g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                             RenderingHints.VALUE_ANTIALIAS_ON);

        //the origin is translated here to facilitate the mathematics in the constructor:
        g2D.translate(maximumRadius, maximumRadius);

        //a clip is defined here to create a graceful fade at the outer edge of the spiral:
        Shape reducedEllipse = new Ellipse2D.Double(
                   -maximumRadius * ((double)(NUMBER_TURNS - 0.5) / NUMBER_TURNS),
                   -maximumRadius * ((double)(NUMBER_TURNS - 0.5) / NUMBER_TURNS),
                    maximumRadius * 2 * ((double)(NUMBER_TURNS - 0.5) / NUMBER_TURNS),
                    maximumRadius * 2 * ((double)(NUMBER_TURNS - 0.5) / NUMBER_TURNS));
        g2D.setClip(reducedEllipse);

        //the dynamically-set argument of this affine transformation creates the animation:
        g2D.rotate(rotation);
```

```java

            GeneralPath spiral = new GeneralPath();
            spiral.moveTo(0, 0);

            //the linear approximation seems to give a better result close to the center...

            for (int i = 1; i <= (3 * POINTS_PER_TURN); i++)
            {
                spiral.lineTo(xPoints[i - 1], yPoints[i - 1]);
            }

            //while the point-controlled curves seem to look better near the perimeter...

            for (int i = (3 * POINTS_PER_TURN) + 1; i <= NUMBER_TURNS * POINTS_PER_TURN; i++)
            {
                spiral.quadTo(xGuides[i - 1], yGuides[i - 1], xPoints[i - 1], yPoints[i - 1]);
            }

            g2D.setPaint(new Color(96, 0, 128));
            g2D.setStroke(new BasicStroke(/*20*/ (int) (maximumRadius /12) , BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
            g2D.draw(spiral);
        }
        /*
        public void run()
        {
            while(true)
            {
                rotation -= PI / 8;
                repaint();
                try {
                    Thread.sleep(updateInterval);
                } catch (InterruptedException exception) {Thread.currentThread().interrupt();}
            }
        }*/
    }

}

```