1.124J Foundations of Software Engineering

Problem Set 7 - Solution

Due Date: Thursday 11/2/00

Problem 1:[100%]

PoleSimulator.java

```
import javax.swing.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import java.text.*;
import java.awt.geom.CubicCurve2D;
import java.net.*;
import java.io.*;
import java.util.*;

public class PoleSimulator extends JFrame
                    implements Runnable, ActionListener
{
   int frameNumber;
   long startTime ;
   int delay;
 Thread animatorThread;
   boolean frozen = true;

 int dx=500, dy=500;
 boolean dataFlag=false;

   JMenuBar menuBar ;
   JMenu fileMenu;
   JMenuItem importMI, exitMI;

 JToolBar toolBar;
```

```
JButton startButton, stopButton, continueButton, resetButton ;
JSeparator separator1, separator2, separator3;
PolePanel polePanel;
JPanel contentPane;

PoleData poleData;



  public PoleSimulator(String str)
   {
super(str);

poleData = new PoleData();

contentPane = new JPanel() ;
  contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.Y_AXIS));

setMenuBar();

setToolBar();

  setPolePanel();

  setContentPane(contentPane);

     pack();
  setVisible(true);

setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

addWindowListener(new WindowAdapter()
              {
            public void windowClosing(WindowEvent e)
                 {
      dispose(); System.exit(0);
      }
              }                );

}

private void setMenuBar()
 {
```

```
    menuBar = new JMenuBar();

    fileMenu = new JMenu("File");
    menuBar.add(fileMenu);
    importMI = new JMenuItem("Import Data");
    importMI.addActionListener(this);
    fileMenu.add(importMI);
    exitMI = new JMenuItem("Exit");
    exitMI.addActionListener(this);
    fileMenu.add(exitMI);

  setJMenuBar(menuBar);
  }


private void setToolBar()
  {
  toolBar = new JToolBar();

  startButton = new JButton(new ImageIcon("start.gif"));
  startButton.setMnemonic(KeyEvent.VK_S);
  startButton.setToolTipText("Start");
  startButton.setActionCommand("Start");
    startButton.addActionListener(this);
  startButton.setEnabled(false);
  startButton.setBackground(Color.gray);
  toolBar.add(startButton);

  separator1 = new JSeparator();
    separator1.setBackground(Color.black);
  toolBar.add(separator1);

  stopButton = new JButton(new ImageIcon("stop.gif"));
  stopButton.setMnemonic(KeyEvent.VK_B);
  stopButton.setToolTipText("Stop");
  stopButton.setActionCommand("Stop");
  stopButton.addActionListener(this);
    stopButton.setEnabled(false);
  stopButton.setBackground(Color.gray);
  toolBar.add(stopButton);

  separator2 = new JSeparator();
    separator2.setBackground(Color.black);
```

```
  toolBar.add(separator2);

    continueButton = new JButton(new ImageIcon("continue.gif"));
  continueButton.setMnemonic(KeyEvent.VK_C);
  continueButton.setToolTipText("Continue");
  continueButton.setActionCommand("Continue");
  continueButton.addActionListener(this);
    continueButton.setEnabled(false);
    continueButton.setBackground(Color.gray);
  toolBar.add(continueButton);

  separator3 = new JSeparator();
    separator3.setBackground(Color.black);
  toolBar.add(separator3);

  resetButton = new JButton(new ImageIcon("reset.gif"));
  resetButton.setMnemonic(KeyEvent.VK_R);
  resetButton.setToolTipText("Reset");
  resetButton.setActionCommand("Reset");
  resetButton.addActionListener(this);
    resetButton.setEnabled(false);
    resetButton.setBackground(Color.gray);
  toolBar.add(resetButton);

  contentPane.add(toolBar);
  }

private void setPolePanel()
 {
    polePanel = new PolePanel(poleData);
 polePanel.setMinimumSize(new Dimension(dx,dy-100));
 polePanel.setPreferredSize(new Dimension(dx,dy));

 contentPane.add(polePanel);
 }


  public void start()
    {
      if(!frozen)
       {
        if(animatorThread == null)
          {
```

```
            animatorThread = new Thread(this);
           }
         animatorThread.start();
       }
    }


public void stop()
    {
       animatorThread = null;
    }



public void actionPerformed(ActionEvent e)
   {
      Object src = e.getSource();

      if(src == exitMI)
        System.exit(0);
   else if (src == importMI)
 readData();
   else if (src == startButton)
        startSimulation();
      else if(src == stopButton)
        stopSimulation();
      else if(src == continueButton)
        continueSimulation();
      else if(src == resetButton)
        resetSimulation();
      else if(src == exitMI)
        exitSimulation();
   }


public void readData()
   {
  System.out.println("Getting the pole data");

String file = "http://web.mit.edu/1.124/www/pole.data";

     int n = 0;
```

```
    try
     {
       URL url = new URL(file);
       Vector vT = new Vector();
  Vector vD = new Vector();;

       URLConnection connection = url.openConnection();
       InputStreamReader inStrReader = new InputStreamReader(connection.getInputStream());
       StreamTokenizer strTokenizer = new StreamTokenizer(inStrReader);

       while(strTokenizer.nextToken()!= strTokenizer.TT_EOF)
          {
            n++;
            vT.addElement(new Double((double)strTokenizer.nval));
            strTokenizer.nextToken();
            vD.addElement(new Double((double)strTokenizer.nval));
          }

  poleData.times = new double[n];
       poleData.displacements = new double[n];

double t2, t1;
poleData.maxDisplacement = 0.0;

  for(int i=0; i<n; i++)
   {
    poleData.times[i] = (new Double((vT.elementAt(i)).toString())).doubleValue();
    poleData.displacements[i] = (new Double((vD.elementAt(i)).toString())).doubleValue();
           if(Math.abs(poleData.displacements[i]) > poleData.maxDisplacement)
    poleData.maxDisplacement = Math.abs(poleData.displacements[i]);
    }

t2 = new Double((vT.elementAt(1)).toString()).doubleValue();
t1 = new Double((vT.elementAt(0)).toString()).doubleValue();
  poleData.dt = t2 - t1;

       inStrReader.close();
dataFlag =true;
startButton.setEnabled(true);
       }
    catch(MalformedURLException e)
       {
        System.out.println(" MalformedURLException: " + e);
```

```
        }
      catch(IOException e)
        {
          System.out.println("IOException: " + e.getMessage());
        }

System.out.println("\n  dt = " + poleData.dt);
poleData.points = n;
      System.out.println("\n" + poleData.points + " values have been read");
   poleData.duration = poleData.points * poleData.dt;
   }



public void startSimulation()
    {
    System.out.println("Starting the simulation");
  initializeSimulator();

        frozen = false;

        startButton.setEnabled(false);
        stopButton.setEnabled(true);
  continueButton.setEnabled(false);
        resetButton.setEnabled(false);

  start();
      }



public void stopSimulation()
    {
    System.out.println("Stopping the simulation");
        frozen = true;

        startButton.setEnabled(false);
        stopButton.setEnabled(false);
  continueButton.setEnabled(true);
        resetButton.setEnabled(true);

        animatorThread = null;
      }
```

```
 public void continueSimulation()
    {
    System.out.println("Continuing the simulation");

       frozen = false;

       startButton.setEnabled(false);
       stopButton.setEnabled(true);
  continueButton.setEnabled(false);
       resetButton.setEnabled(false);

  start();
     }


  public void resetSimulation()
    {
       System.out.println("Simulation has been reset");

  frozen = true;
  poleData.step = 0;
  poleData.t = 0.0;

       startButton.setEnabled(false);
       stopButton.setEnabled(false);
  continueButton.setEnabled(false);
       resetButton.setEnabled(false);

  if(dataFlag)
  startButton.setEnabled(true);

  polePanel.repaint();
       animatorThread = null;
     }


public void exitSimulation()
  {
       System.out.println("Exiting the program....");
   System.exit(0);
  }
```

```
 public void run()
   {
     Thread.currentThread().setPriority(Thread.MIN_PRIORITY);

     startTime = System.currentTimeMillis();
delay = (int)(1000.0 * poleData.dt);

     while(Thread.currentThread() == animatorThread
       &&  (frameNumber<=poleData.points) )
        {
          repaint();

           try
            {
              startTime += delay;
              Thread.sleep(Math.max(0, startTime-System.currentTimeMillis()));
             }
 catch(InterruptedException e)
             {
   break;
   }
 poleData.times[poleData.step] = poleData.t;

 poleData.t += poleData.dt;
          poleData.step = frameNumber++;
        }



if(frameNumber > poleData.points)
    {
        startButton.setEnabled(false);
        stopButton.setEnabled(false);
    continueButton.setEnabled(false);
        resetButton.setEnabled(true);
 poleData.step = 0;
    }
    }


private void initializeSimulator()
  {
  poleData.t = 0.0;
```

```
 frameNumber = 0;
 }


  public static void main(String args[])
   {
     new PoleSimulator("Pole Simulator");
   }

}
```

PolePanel.java

```
import javax.swing.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
import java.text.*;
import java.awt.geom.CubicCurve2D;

public class PolePanel extends JPanel
{
 CubicCurve2D.Double column;

 int xb, yb, ctrXb, ctrYb;
 int xt, yt, ctrXt, ctrYt;
 int baseX, baseY, baseWidth, baseHeight;
   final static BasicStroke columnStroke = new BasicStroke(5.0f);

 DecimalFormat df1 = new DecimalFormat("#0.##");
 DecimalFormat df2 = new DecimalFormat("#0.####");
 PoleData poleData = null;

   public PolePanel(PoleData d)
     {
  super();
  setBackground(Color.white);
  setForeground(Color.black);
  poleData = d;
  }
```

```
  public void paintComponent(Graphics g)
 {
super.paintComponent(g);
      Graphics2D g2 = (Graphics2D) g;
      g2.setStroke(columnStroke);

xb = ctrXb = (int) (0.5 * getWidth());
yb = (int) (0.7 * getHeight());
yt = (int) (0.2 * getHeight());
ctrYb = (int) (0.6 * getHeight());
ctrYt = (int) (0.2 * getHeight());

baseX = (int) (0.1 * getWidth());
baseWidth = (int) (0.8 * getWidth());
baseY = (int) (0.7 * getHeight());
baseHeight = (int) (0.1 * getHeight());

xt = ctrXt = (int) (0.5 * getWidth()) ;

if(poleData!=null && poleData.step>0)
    xt = ctrXt = (int) (0.5 * getWidth() + 0.2*getWidth()*
   poleData.displacements[poleData.step] /
poleData.maxDisplacement);

      column = new CubicCurve2D.Double(xb, yb, ctrXb, ctrYb,
         ctrXt, ctrYt, xt, yt);
g2.draw(column);

g2.fillRect(baseX, baseY, baseWidth, baseHeight);

if(poleData!=null && poleData.step>0)
  {
   g2.drawString("Time = " + df1.format(poleData.t),
                 (int) (0.2 * getWidth()),
                 (int) (0.90 * getHeight()));

   g2.drawString("Duration = " + df1.format(poleData.duration),
                  (int) (0.2 * getWidth()),
                  (int) (0.95 * getHeight()));
```

```
    g2.drawString("Displacement = " +
      df2.format(poleData.displacements[poleData.step]),
                (int) (0.6 * getWidth()),
                (int) (0.90 * getHeight()));

    g2.drawString("Max displacement = " + df2.format(poleData.maxDisplacement),
                 (int) (0.6 * getWidth()),
                 (int) (0.95 * getHeight()));
  }
 }

}
```

PoleData.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PoleData
{
 double times[]=null, displacements[]=null;
 double maxDisplacement;
   double t, dt, duration;
 int step, points;

   PoleData()
    {
  times = null;
  displacements = null;
       maxDisplacement = 0.0;
  step = 0;
  points = 0;
  t =0.0;
  dt = 0.0;
  duration = 0.0;
    }
}
```