Diploma in Computer Science
Computer Science Tripos II(G)

Java Case Studies

Martin Richards `mr@cl.cam.ac.uk`

Michaelmas 2003

(Version November 13, 2003)

1 Introduction

These notes contain the source of two Java programs than will be used as case studies in this course. The first is an animation of a sundial designed for a house in norther Ghana and the second is a simple spreadsheet program that is one of the demonstation programs in the Java distribution from Sun.

2 SunDial

2.1 Makefile

```
WWW = /homes/mr/public_html

run:    SunDial.class
    appletviewer SunDial.html

SunDial.class:  SunDial.java
    rm -f *.class
    javac SunDial.java

copy:   clean
    (cd ..; cp -r Sundial/* /homes/mr/Sundial)

pub:    SunDial.class
    (cp -r *.class Makefile $(WWW)/sundial; \
    cp -r SunDial.java $(WWW)/sundial; \
    cp -r SunDial.java $(WWW)/sundial/SunDial.src )

clean:
    rm -f *.class *~
```

2.2 SunDial.html

```
<html>
<body>
<h1>
A Sundial Designed for a House in Damongo, Ghana
</h1>
<p>

<applet code="SunDial.class" width=700 height=500>
Sorry -- your browser does not support java applets
```

```
</applet>

<p>
The source code for this applet is in:
<A HREF="SunDial.java">SunDial.java</A>.

<P>
A current image of the new
<A HREF="http://www.uk.research.att.com/sundial/">sundial</A>
in Pembroke College may be of interest.
Bear in mind that it may not be sunny in Cambridge right now.

</body>
</html>
```

2.3 SunDial.java

```
/*
This program animates a sundial designed for Geraldine's back garden
of a house in Damongo, Ghana.

Re-implemented by: Martin Richards -- (c) November 2002
In its present form it is still incomplete and un polished.

Original implementation using Java 1.0 May 1988
Updated from Java 1.0 to Java 1.1 -- October 2000
Complete re-design for Java 1.4.1 -- October 2002

I gratefully acknowledge Frank King's help with mathematics of
sundials, and I have used the approximation to the equation of time
given in a note by Art Carlson in:
http://www.ipp.mpg.de/~awc/sundial.html

This program was originally implemented using the Sheets Hypertext
System developed by the Gwydion Group at Carnegie Mellon University.


Overview

The setup looks as follows:
```

```
                 Sun
                   \
                    \         (0,-2,1)
     (2,-2,1) J------\-----------M------------------I (-2,-2,1)
             /        \         /                  /|
            /          \ Roof  /                  / |
           /            \     /                  /  |
 (2,-1,1) G--------------\---/------------------H   |
          |   |           \ /                   |  West
          |   |    (0,0,1)<P>                   |  Wall
     East |   |      South |\ Wall              |   |
     Wall |   |            | \                  |   |
          |   D------------|--\-----------------|   C (-2,-2,0)
          |  /             |   \   Ground       |  /
          | /              |    \               | /
          |/     4pm       |noon \   8am        |/
 (2,-1,0) E      /         | /    \   \         F (-2,-1,0)
         /     /    (0,0,0)Q/          \       /
        /------------------/--------\---------/
       /   /              /         <S>  \   /
      /  /               /                \ /
     A-------------------------------------B
  (2,2,0)             North            (-2,2,0)
```

The x-, y- and z-axes are East, North and Up respectively. The house in Damongo is at latitude 9.08N and longitude 1.81W. Being close to the Grenwich Meridian, the time there is GMT, and since it is close to the equator the sun is high in the sky at midday and can even be vertically overhead.

Points in the diagram have been given names. ABCD are the four corners of the 4x4 square concrete "garden" on the ground, Q is the centre of this square and is taken as the origin of the coordinate system. EDJG, FCIH and DCIJ are the vertices of the east, west and south walls respectively, and GHIJ are the vertices of the roof. M is the mid point of IJ and P is the position vertically above Q where there is a wooden star with a hole to cast a shadow on the ground at point S. Lines painted on the ground allow the position of S to be interpreted as a time of day. With the equation of time correction the result is a time of day accurate to within a minute or two.

The program attempts to follow the MVC (Model-View-Control) style of programming. The java classes used in this program are: SunDial, Model, Viewer, Controls, EyePic, ScrPoint and Point3D.

SunDial

This contains the main program. It initialised the system by creating instances of the Model, the Viewer and the Controller classes and then behaves as the overall controller of the application. It responds to invocations of init, start, stop and destroy that may be called by appletviewer or a browser.

Model

The model contains quantities relating to the physical structure of the scene including the the discription of the house and garden and the position of the sun's shadow (S). It also contains the optional trace of how point S has moved and displayed by the viewer appears as yellow curves on the ground. If the time steps are in days the well known figure of eigth picture is generated illustrating the equation of time.

It receives time updates fron the controller causing it to recompute the location of S and associated values. When anything in the model changes it causes the viewer to repaint the scene.

Viewer

The viewer contains variables and methods used to convert the model into a 2 dimensional image on the screen. It contains fields representing the view point and field of view, as well as methods to paint the scene. In principle, other viewers are possible such as one to write an ASCII representation of the scene to file. Such aleternative viewers are not implemented in this program.

Controls

The controller contains facilities to allow the user to change the current state of the system such as the date and time and the position of the view point. Both keyboard and mouse events are used. The controller uses a thread to generate a uniform sequence of time updates to cause the animation. The thread runs at low priority so that the user's actions take precedence.

The controller has no variables repesenting the state of the system, preferring to update the state by calling methods of the model and the viewer. It does however have a description of the control panel and methods to act on any control events generated by the user. A painting method is defined for the control panel but called explicitly from the viewer when a new image must be generated. This is so that the controls can overlay the main scene paited by the viewer.

EyePic

This class defines a component that appears in the control panel. Its purpose is to allow the user to line of sight of the view point. The distance of the view point is separately controlled by a scroll bar in the controller panel. The line of sight from the view point to the house is represented by the coordinates (eyeR) of where it passes through the roof level (z=1.0) on its way to the origin (Q=(0,0,0)). This point (eyeD) is a field belonging to the viewer, as is eyeD the distance between the view point and the origin.

ScrPoint

This class is used to represent the integer coordinates of points on the screen. It has two fields x and y but no methods.

Point3

This provides a representation of points in 3 dimensions, together with a collection of useful operations on them. Its methods are: add, sub, mul(by a scalar), innerProd and crossProd. The method cosines calculates the point of intersection of the line from the origin to the point and a unit sphere centred at the origin.

```
*/

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.math.*;
import java.util.*;
```

```
public class SunDial extends Applet
      implements KeyListener
{
    private Model model;
    private Viewer viewer;
    private Controls controls;
    private Thread ticker;

    public static int xSize, ySize;  // The applet size


    // Definitions of overriden Applet methods
    // init      The applet has been loaded into the system
    // start     The applet should start execution
    // stop      The applet should stop execution
    // destroy   The applet is being reclaimed -- close it down

    public void init() {
        Component root = (Component) this;
        while (root.getParent() != null) root = (Component) root.getParent();
        // Temporary fudge to solve to a visibility problem
        root.setLocation(new Point(50,50));

        xSize = getBounds().width;
        ySize = getBounds().height;
    //System.out.println("width=" + xSize + " height=" + ySize);


    //System.out.println("SunDial: calling new Model()");

        model = new Model();     // Create the model

        //System.out.println("SunDial: calling new Viewer()");
        // Create a viewer for this model.
        viewer = new Viewer(model, xSize, ySize);

        // Create the controller
        //System.out.println("SunDial: calling new Controls()");
        controls = new Controls(model, viewer);

        ticker = new Thread(controls); // Create the ticker thread
```

```
        ticker.setPriority(Thread.NORM_PRIORITY-2);

        add(viewer);
        addKeyListener(this);
        viewer.requestFocus();
        //System.out.println("Applet init: done");
    }

    public void start() {
        //System.out.println("Applet start called");
        ticker.start();
    }

    public void stop() {
        System.out.println("stop called");
        controls.die();  // Tell the Ticker thread to commit suicide
        try {
           ticker.join(); // Wait for the thread in controls to die
        } catch (InterruptedException e) {}
        remove(viewer);
        //showStatus("");    // Important little cleanup job.
        System.out.println("returning from stop");
    }

    public void destroy() {
    //System.out.println("destroy called");
    }

    // Applet KeyBoard Events

    public void keyTyped   (KeyEvent e) { }
    public void keyReleased(KeyEvent e) { }
    public void keyPressed (KeyEvent e) {
        int ch = e.getKeyChar();
        System.out.println("SunDial: keyPressed ch=" + ch);
        controls.keych(ch);
    }
}



// ******************** The Model ************************************
```

```
class Model
{
    // This class contains constant, variables and methods that
    // describe the parameters of the sundial. These include  the
    // coordinates of various points in the scene, the colour scheme,
    // the trace of the sun if present, the simulated time and
    // related quantities. It includes methods to compute the position
    // of the sun's shadow on the ground.

    // Typically there will only be one instance of this class.


    // First define the constant points

    public Point3 A, B, C, D, E, F, G, H, I, J, M, P, Q;
    public Point3 S;  // Position of the sun's bean hits on the ground
    // (re-calculated every time the simulated time changes).

    Color colGround, colEwall, colWwall, colSwall;
    Color colRoof, colMarks, colSun, colBackground;

    final static double CANCER_LATITUDE = 23.45;  // Degrees

    private double longitude=0; // Degrees west of the Grenwich Meridian
    private double latitude=0;  // Degrees north of the equator
    private double declination; // Declination of the sun in radians
    private double sinDec, cosDec;
    private double phi;         // (latitude - 90) in radians
    private double sinPhi, cosPhi;

    public double minsSlow;     // Sundial error in minutes.

    public int month=1, day=1, hour=0, min=0; // The simulated time

    public Vector tracePoints = null;
    public boolean penDown = false;

    public Model() {      // The Constructor
        //System.out.println("Model: init entered");
    // Set points in the scene
        A = new Point3(  2,  2,  0);
```

```
        B = new Point3( -2,  2,  0);
        C = new Point3( -2, -2,  0);
        D = new Point3(  2, -2,  0);
        E = new Point3(  2, -1,  0);
        F = new Point3( -2, -1,  0);
        G = new Point3(  2, -1,  1);
        H = new Point3( -2, -1,  1);
        I = new Point3( -2, -2,  1);
        J = new Point3(  2, -2,  1);
        M = new Point3(  0, -2,  1);
        P = new Point3(  0,  0,  1);
        Q = new Point3(  0,  0,  0);
        S = new Point3(  0,  0,  0);

        // Set the colours of various surfaces and the sun.
        colGround     = new Color(120, 100,  60);
        colEwall      = new Color(160, 130,  60);
        colWwall      = new Color(140, 110,  20);
        colSwall      = new Color(200, 170,  70);
        colRoof       = new Color(210, 180, 150);
        colMarks      = new Color(255,   0,   0);
        colBackground = new Color(100, 220, 100);
        colSun        = Color.yellow;

        // Set the location of the sundial
        setPosition(9.08, 1.818); // Lat. and Long. of Damongo, Ghana
        //setDeclination(0.0);      // Set the declination to zero

        setDeclination(month, day);
        setMinsSlow(month, day);
        S = shadowPosition(hour + min/60.0);

        //System.out.println("Model: init done");
    }

    public void setTime(int month, int day, int hour, int min) {
    //System.out.println("Model.setTime");
        this.month = month;
        this.day = day;
        this.hour = hour;
        this.min = min;
        setDeclination(month, day);
        setMinsSlow(month, day);
```

```
    S = shadowPosition(hour, min);
    if (penDown) tracePoint(S);
}

public void tracePoint(Point3 p) {
  if (tracePoints == null) tracePoints = new Vector(100, 100);
  double time = hour + min/60.0;
  if(time<6.5 || time>17.5) p = null;
  tracePoints.addElement(p);
}


private double radians(double angle) {
  return angle*Math.PI/180;
}

public void setPosition(double lat, double lng) {
  latitude = lat;
  longitude = lng;
  phi = radians(latitude - 90);
  sinPhi = Math.sin(phi);
  cosPhi = Math.cos(phi);
}

public void setDeclination(double degrees) {
  declination = radians(degrees);
  sinDec = Math.sin(declination);
  cosDec = Math.cos(declination);
}

public void setDeclination(int month, int day) {
  int days = daysRelMar22(month, day);
  declination = Math.asin(Math.sin(radians(360.0*days/365)) *
                          Math.sin(radians(CANCER_LATITUDE)));
  sinDec = Math.sin(declination);
  cosDec = Math.cos(declination);
}

public int daysRelMar22(int month, int day) {
  // Return the number of days relative to march 22
  return daysRelJan1(month, day) - 31 - 28 - 22;
}
```

```
  public int daysRelJan1(int month, int day) {
    // month =  1 for Jan,
    //       = 12 for December
    // day   ranges from 1 to 31, we assume it is not a leap year
    int[] daysInMonth = {31,28,31,30,31,30,31,31,30,31,30,31};
    for (int m = 0; m<month-1; m++) day += daysInMonth[m];
    // Return the number of days relative to January 1
    return day - 1;
  }

  public void setMinsSlow(double mins) {
    minsSlow = mins;
  }

  public void setMinsSlow(int month, int day) {
    int n = daysRelJan1(month, day) + 1;
    double b = radians(360.0*(n-81.0)/364.0);
    minsSlow = -9.87*Math.sin(2*b) + 7.53*Math.cos(b) + 1.5*Math.sin(b);
  }

  public Point3 shadowPosition(int hour, int min) {
    return shadowPosition(hour + min/60.0);
  }

  public Point3 shadowPosition(double hour) {
    hour -= minsSlow/60.0; // Put in the sundial error
    double h = radians(hour*360.0/24.0 - longitude);
    double sinh = Math.sin(h);
    double cosh = Math.cos(h);
    double xtop = sinh*cosDec;
    double ytop = cosh*cosDec*cosPhi - sinDec*sinPhi;
    double bot  = cosh*cosDec*sinPhi + sinDec*cosPhi;
    double x = -xtop/bot;
    double y = -ytop/bot;

    return new Point3(x, y, 0);
  }
}
```

```
//******************* The Viewer *********************************

class Viewer extends Panel
           implements MouseListener,
                      MouseMotionListener,
                      KeyListener
{
    // This class constructs a screen image of the model and
    // has a paint method to cause it to be displayed. Its is called
    // by the controller whenever anything about the viewing
    // parameters (such as the eye position) change, and its paint
    // method is called whenever a new image must be generated.

    private Model m;            // This is a handle to the model

    public Point3 eyeG;         // The eye target on the ground.
    public Point3 eyeR;         // The eye target on the roof.
    public double eyeD;         // The eye distance
    private Point3 eyePosition; // The eye coordinates.

    // Direction cosines for the eye: Right, Up and Forward.
    // These are recomputed every time the eye position changes.
    private Point3 rCosines, uCosines, fCosines;

    private int xSize, ySize; // Current size of the screen in pixels.
    public double scale;      // Scale measured in pixels per radian.
    public ScrPoint orig;     // Current pixel position of the origin.

    private Controls cp;    // The controls panel -- so that Viewer
                            // can overlay it on top of the scene

    Image canvasImage;      // For double buffering the animation
    Graphics canvasG;
    Font font;
    FontMetrics fm;


    public Viewer(Model m, int xs, int ys){          // The constructor
    this.m = m;
        xSize = xs;
        ySize = ys;

        eyeG = new Point3( 0, 0, 0);
```

```
    eyeR = new Point3( -1, 1, 1);
    eyeD = 10;

    // The following are given new values on the first call of
    // updateImage is called.
    scale = 800.0;              // Set the field of view.
    orig = new ScrPoint(xs/2, ys/2); // Initial position of the origin
                                // in the screen image.
    addMouseListener(this);
    addMouseMotionListener(this);
    addKeyListener(this);
    requestFocus();    // So that keyboard events are noticed
}

public void setCp(Controls cp) {
this.cp = cp;
}

// Methods concerned with displaying the sundial image

private void setDirectionCosines() {
//System.out.println("eyeR="+eyeR.x+", "+eyeR.y+", "+eyeR.z);
  fCosines = eyeG.sub(eyeR).cosines();
  rCosines = (new Point3(-fCosines.y, fCosines.x, 0)).cosines();
  uCosines = rCosines.crossProd(fCosines);

  //System.out.println("r="+rCosines.x+", "+rCosines.y+", "+rCosines.z);
  //System.out.println("u="+uCosines.x+", "+uCosines.y+", "+uCosines.z);
  //System.out.println("f="+fCosines.x+", "+fCosines.y+", "+fCosines.z);

  //System.out.println("r.u="+rCosines.innerProd(uCosines));
  //System.out.println("r.f="+rCosines.innerProd(fCosines));
  //System.out.println("u.f="+uCosines.innerProd(fCosines));

  //System.out.println("lengths="+rCosines.length()+", "
  //                              +uCosines.length()+", "
  //                              +fCosines.length());

  //System.exit(123);

  // Flip if screen up is not up!!
  if(uCosines.z < 0) {
    rCosines = rCosines.mul(-1);
```

```
      uCosines = uCosines.mul(-1);
    }
    // l-, u- and f-Cosines are unit vectors in the directions
    // screen left, screen up and forward, respectively.
    // They are orthongonal.
    //System.out.println("setDirectionCosines: done");
  }

  public void setOrigin(int x, int y) {
      orig.x = x;
      orig.y = y;
      repaint(20);
  }

  public void setEyeR(Point3 p) {
  eyeR = p;
      setEyePosition();
  }

  public void setEyeD(double d) {
  eyeD = d;
      setEyePosition();
  }

  public void setEyePosition() {
      //System.out.println("eyeR = "+
      //                   eyeR.x+","+eyeR.y+","+eyeR.z+" eyeD="+eyeD);
  //this.eyeR = eyeR;
  //this.eyeD = eyeD;
      setDirectionCosines();
      eyePosition = eyeG.sub(fCosines).mul(eyeD);
      //System.out.println("eyePosition = "+
      //                    eyePosition.x+","+
      //                    eyePosition.y+","+
      //                    eyePosition.z);
      repaint(20);
  }

  public Point3 screenPoint(Point3 p) {
    Point3 pCosines = p.sub(eyePosition).cosines();

    double x = rCosines.innerProd(pCosines);
    double y = uCosines.innerProd(pCosines);
```

```
    double z = fCosines.innerProd(pCosines);

    //System.out.println("p   = "+p.x+", "+p.y+", "+p.z);
    //System.out.println("xyz = "+x+", "+y+", "+z);

    if(z>0.01) { x /= z;   y /= z;   }
    else       { x /= 0.01; y /= 0.01; }
    //System.out.println("xyz = "+x+", "+y+", "+z);
    //System.out.println("scale = "+scale);

    // If z>0 the point is in front of the eye and so
    // may be in the field of view,
    // If z<0 it is behind the eye and should not be displayed.
    return new Point3(orig.x+scale*x, orig.y-scale*y, z);
  }

// Painting

// The picture depends on eyePosition in Viewer and S in Model,
// both of which might change while updateImage is executing.
// The use of sychronization should be considered.

private int xView(Point3 p) {
  return (int) Math.round(screenPoint(p).x);
}

private int yView(Point3 p) {
  return (int) Math.round(screenPoint(p).y);
}

public void updateImage(Graphics g) {
    Dimension d = getSize();
    double width  = d.width;
    double height = d.height;

    //System.out.println("updateImage: called");
  if (xSize!=width || ySize!=height) {
  //System.out.println("updateImage: size changed");
  //System.out.println("xSize=" + xSize + " ySize=" + ySize);
  //xSize = getSize().width;
  //ySize = getSize().height;
  //System.out.println("width=" + width + " height=" + height);
      scale = xSize*0.8;
```

```
        orig.x = (int) (xSize * 0.5);
        orig.y = (int) (xSize * 0.4);
        //System.out.println(" xSize="+xSize+
        //                   " ySize="+ySize+
        //                   " orig.x="+orig.x+
        //                   " orig.y="+orig.y+
        //                   " scale="+scale);

      canvasImage = null;
      //System.out.println("updateImage: calling createImage");
      canvasImage = createImage(xSize, ySize);
      //System.out.println("updateImage: calling canvasImage.getGraphics");
      canvasG = canvasImage.getGraphics();
      font = new Font("TimesRoman", Font.BOLD, 12);
      canvasG.setFont(font);
      fm = canvasG.getFontMetrics();

      // Re-position the control panel
      //System.out.println("updateImage: calling setLocation");
      if(cp!=null) cp.setLocation(0, ySize-100);
      setSize(xSize, ySize);
      setLocation(0, 0);
    }
    //System.out.println("updateImage: calling draw(...)");
      draw(canvasG);
      //System.out.println("updateImage: calling cp.paint(...)");
      if(cp!=null) cp.paint(canvasG);
      //System.out.println("updateImage: calling g.drawImage(...)");
      g.drawImage(canvasImage, 0, 0, this);
      //System.out.println("updateImage: g.drawImage done");
  }

  public void paint(Graphics g) {
    update(g);
  }

  public void update(Graphics g) {
   updateImage(g);
  }

  public void draw(Graphics g) {
    g.setColor(m.colBackground);
    g.fillRect(0, 0, xSize, ySize);
```

```
    paintGround(g);
    paintGroundPic(g);

    Point3 p = eyePosition;

    if (p.z<= 1.0) paintRoof(g);

    if (p.y< -1.0) paintSunLight(g);
    if (p.y>=-2.0) paintSouthWall(g);

    if (p.x<0.0) { paintEastWall(g);
                   paintWestWall(g);
    } else       { paintWestWall(g);
                   paintEastWall(g);
    }
    if (p.z > 1.0) paintRoof(g);

    if (p.y< -2.0) paintSouthWall(g);

    if (p.y>=-1.0) paintSunLight(g);

    g.setColor(Color.black);

    // Debug the screen projection
    //for(double a=0.0; a<1.0; a+=0.1) {
    //    Point3 t = m.A.mul(a).add(m.D.mul(1-a));
    // g.fillRect(xView(t), yView(t), 2, 2);
    //}
    g.drawString("Animation of a sundial designed for a House"+
                 " in Damongo, Ghana", 100,20);
    //System.out.println("draw: returning");
  }

  private void paintGround(Graphics g) {
       int[] xpts = new int[4];
       int[] ypts = new int[4];
       xpts[0] = xView(m.A); ypts[0] = yView(m.A);
       xpts[1] = xView(m.B); ypts[1] = yView(m.B);
       xpts[2] = xView(m.C); ypts[2] = yView(m.C);
       xpts[3] = xView(m.D); ypts[3] = yView(m.D);
       g.setColor(m.colGround);
       g.fillPolygon(xpts, ypts, 4);
  }
```

```
private void drawSunPath(int month, int day, Graphics g) {
  int x1=0, y1=0, x2=0, y2=0;
  m.setDeclination(month, day);
  m.setMinsSlow(0.0);
  boolean first = true;
  for (double hour=8; hour<=16.01;hour++) {
    Point3 p = m.shadowPosition(hour);
    x2 = xView(p);
    y2 = yView(p);
    if(!first) g.drawLine(x1,y1,x2,y2);
    x1 = x2;
    y1 = y2;
    first = false;
  }
}

private void paintGroundPic(Graphics g) {
     g.setColor(m.colMarks);

     drawSunPath( 3, 22, g);  // Spring equinox
     drawSunPath( 6, 21, g);  // Summer equinox
     drawSunPath(12, 21, g);  // Winter equinox

     // now draw the hour lines
     for (double hour = 8; hour<16.01; hour++) {
       m.setDeclination(30.0);
       Point3 p1 = m.shadowPosition(hour);
       m.setDeclination(-30.0);
       Point3 p2 = m.shadowPosition(hour);
       g.drawLine(xView(p1), yView(p1),
                  xView(p2), yView(p2));
     }
     // now draw the sun trace (if any)
     if (m.tracePoints != null) {
       g.setColor(Color.yellow);
       Point3 lastPoint = null;
       Enumeration pts = m.tracePoints.elements();
       while(pts.hasMoreElements()) {
         Point3 p = (Point3) pts.nextElement();
         if(lastPoint!=null && p!=null) {
           int x1 = xView(lastPoint),
               y1 = yView(lastPoint),
```

```
                x2 = xView(p),
                y2 = yView(p);
            g.drawLine(x1, y1, x2, y2);
     }
          lastPoint = p;
       }
     }
}

private void paintSouthWall(Graphics g) {
     int[] xpts = new int[4];
     int[] ypts = new int[4];
     xpts[0] = xView(m.C); ypts[0] = yView(m.C);
     xpts[1] = xView(m.I); ypts[1] = yView(m.I);
     xpts[2] = xView(m.J); ypts[2] = yView(m.J);
     xpts[3] = xView(m.D); ypts[3] = yView(m.D);
     g.setColor(m.colSwall);
     g.fillPolygon(xpts, ypts, 4);
}

private void paintEastWall(Graphics g) {
     int[] xpts = new int[4];
     int[] ypts = new int[4];
     xpts[0] = xView(m.G); ypts[0] = yView(m.G);
     xpts[1] = xView(m.J); ypts[1] = yView(m.J);
     xpts[2] = xView(m.D); ypts[2] = yView(m.D);
     xpts[3] = xView(m.E); ypts[3] = yView(m.E);
     g.setColor(m.colEwall);
     g.fillPolygon(xpts, ypts, 4);
}

private void paintWestWall(Graphics g) {
     int[] xpts = new int[4];
     int[] ypts = new int[4];
     xpts[0] = xView(m.H); ypts[0] = yView(m.H);
     xpts[1] = xView(m.I); ypts[1] = yView(m.I);
     xpts[2] = xView(m.C); ypts[2] = yView(m.C);
     xpts[3] = xView(m.F); ypts[3] = yView(m.F);
     g.setColor(m.colWwall);
     g.fillPolygon(xpts, ypts, 4);
}

private void paintRoof(Graphics g) {
```

```
     int[] xpts = new int[8];
     int[] ypts = new int[8];
     xpts[0] = xView(m.G); ypts[0] = yView(m.G);
     xpts[1] = xView(m.J); ypts[1] = yView(m.J);
     xpts[2] = xView(m.I); ypts[2] = yView(m.I);
     xpts[3] = xView(m.H); ypts[3] = yView(m.H);
     g.setColor(m.colRoof);
     g.fillPolygon(xpts, ypts, 4);
     g.setColor(Color.black);
     g.drawLine(xView(m.P), yView(m.P),
                xView(m.M), yView(m.M));
     // Now draw the star (the gnomon)
     double a = 0.10;
     double b = 0.05;
     xpts[0] = xView(new Point3( 0, -b, 1.0));
     ypts[0] = yView(new Point3( 0, -b, 1.0));
     xpts[1] = xView(new Point3(-a, -a, 1.0));
     ypts[1] = yView(new Point3(-a, -a, 1.0));
     xpts[2] = xView(new Point3(-b,  0, 1.0));
     ypts[2] = yView(new Point3(-b,  0, 1.0));
     xpts[3] = xView(new Point3(-a,  a, 1.0));
     ypts[3] = yView(new Point3(-a,  a, 1.0));
     xpts[4] = xView(new Point3( 0,  b, 1.0));
     ypts[4] = yView(new Point3( 0,  b, 1.0));
     xpts[5] = xView(new Point3( a,  a, 1.0));
     ypts[5] = yView(new Point3( a,  a, 1.0));
     xpts[6] = xView(new Point3( b,  0, 1.0));
     ypts[6] = yView(new Point3( b,  0, 1.0));
     xpts[7] = xView(new Point3( a, -a, 1.0));
     ypts[7] = yView(new Point3( a, -a, 1.0));
     g.drawPolygon(xpts, ypts, 8);
}

private void paintSunLight(Graphics g) {
     double time = m.hour + m.min/60.0;
     //System.out.println("paintSunLight"+time);
     if(time<6.5 || time>17.5) return;
     int x1=xView(m.P),
         y1=yView(m.P),
         x2=xView(m.S),
         y2=yView(m.S);
     //System.out.println("paintSunLight");
     g.setColor(m.colSun);
```

```
        g.fillOval(x1-2, y1-1, 4, 2);
        g.setColor(m.colSun);
        g.fillOval(x2-2, y2-1, 4, 2);
        g.drawLine(10*(x1-x2)+x2, 10*(y1-y2)+y2, x2, y2);
  }

    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered (MouseEvent e) { }
    public void mouseExited  (MouseEvent e) { }
    public void mouseClicked (MouseEvent e) { }
    public void mousePressed (MouseEvent e) {
      mousePos(e.getX(), e.getY());
    }

    // Mouse motion events
    public void mouseMoved   (MouseEvent e) { }
    public void mouseDragged (MouseEvent e) {
        mousePos(e.getX(), e.getY());
    }

    private void mousePos(int x, int y) {
        //System.out.println("Viewer: calling setOrigin " + x + " y=" +y);
        requestFocus();
        setOrigin(x, y);
        repaint(20);
    }

    // KeyBoard Events

    public void keyTyped   (KeyEvent e) { }
    public void keyReleased(KeyEvent e) { }
    public void keyPressed (KeyEvent e) {
        int ch = e.getKeyChar();
        //System.out.println("Viewer: keyPressed ch=" + ch);
        cp.keych(ch);
    }
}


class Controls extends Panel
    implements  ActionListener,       // for buttons
                ItemListener,         // for choice
                AdjustmentListener,   // for the scroll bar
```

```
                KeyListener,          // for key event
                Runnable

{
    // Variables associated with the controls panel.

    final static int cW = 300, cH = 100; // Width and Height of the panel

    private int sz;

    private Model m;
    private Viewer v;

    private int month, day, hour, min;
    private int dayStep, hourStep, minStep;

    private boolean alive, stepping;    // Controls the ticker thread
    public int tickmsecs;
    public boolean penDown;

    // The components of this panel
    EyePic eyePic;
    Scrollbar eyeSB;
    Button bMonthU, bDayU, bHourU, bMinU;
    Button bMonthD, bDayD, bHourD, bMinD;
    Button bStep, bTrace, bClear, bQuit;
    Choice chAmount;
    Label lbFar, lbNear;
    Label lbError, lbDate, lbTime, lbCorr;

    // The constructor
    Controls(Model m, Viewer v) {
        this.m = m; // Receive the Model and Viewer to control
        this.v = v;

        createComponents(this);
        reshapeComponents(this);
        eyeSB.setValue(75);
        //System.out.println("Controls: calling v.setEyeD()");
        //System.out.println("Controls: calling eyeSB.getvalue="+eyeSB.getValue());
        v.setEyeD(SVtoD(eyeSB.getValue()));
```

```
        month = 7;
        day   = 21;  dayStep = 0;
        hour  = 12; hourStep = 0;
        min   = 0;   minStep = 12;

        tickmsecs = 200; // 5 ticks per seconds !!

        alive    = true;
        stepping = true;
        penDown  = false;

        v.setCp(this); // Tell Viewer where the Contols panel is
        v.add(this);

        // Now add the listeners
        addKeyListener(this);
    }


    public void run() {
    //System.out.println("SunDial: run entered");

    // This code runs as a separate thread updating the
        // sun's position at frequent intervals while stepping is true.
        // It commits suicide (by returning from run) when die is true.
        while (alive) {
        if (stepping) stepTime();
        //System.out.println("timer.run: calling sleep");
            try { Thread.sleep(tickmsecs); } catch(Exception e) {}
        }
        // The thread dies when control leaves run.
    }

    public void die() {
        alive = false;
    }

    public void setStepping(boolean b) {
        stepping = b;
    }


    // Methods concerned with the controls panel.
```

```
  private void createComponents(Panel cp) {
    cp.setLayout(null);

    cp.add(eyePic = new EyePic(m, v, this, 90));

    cp.add(eyeSB = new Scrollbar(Scrollbar.VERTICAL, 0,25,0,125));
    eyeSB.addAdjustmentListener(this);

    cp.setBackground(new Color(180,160,130));

    cp.add(lbFar  = new Label("Far", Label.LEFT));
    cp.add(lbNear = new Label("Near", Label.LEFT));

    cp.add(lbDate  = new Label("JUL 21", Label.CENTER));
    cp.add(bMonthU = new Button("+"));
    bMonthU.addActionListener(this);
    cp.add(bMonthD = new Button("-"));
    bMonthD.addActionListener(this);
    cp.add(bDayU   = new Button("+"));
    bDayU.addActionListener(this);
    cp.add(bDayD   = new Button("-"));
    bDayD.addActionListener(this);

    cp.add(lbTime = new Label("13:15", Label.CENTER));
    cp.add(bHourU = new Button("+"));
    bHourU.addActionListener(this);
    cp.add(bHourD = new Button("-"));
    bHourD.addActionListener(this);
    cp.add(bMinU  = new Button("+"));
    bMinU.addActionListener(this);
    cp.add(bMinD  = new Button("-"));
    bMinD.addActionListener(this);

    cp.add(lbError = new Label("Fast by 12:22", Label.RIGHT));

    cp.add(bStep = new Button("Stop Stepping by"));
    bStep.addActionListener(this);

    cp.add(chAmount = new Choice());
    chAmount.addItem("1 min");
    chAmount.addItem("5 min");
    chAmount.addItem("12 mins");
```

```
    chAmount.addItem("1 hour");
    chAmount.addItem("1 day");
    chAmount.addItem("5 days");
    chAmount.addItem("15 days");
    chAmount.select ("12 mins");
    chAmount.addItemListener(this);

    cp.add(bClear = new Button("Clear trace"));
    bClear.addActionListener(this);
    cp.add(bTrace = new Button("Pen Down"));
    bTrace.addActionListener(this);
    cp.add(bQuit  = new Button("Quit"));
    bQuit.addActionListener(this);
  }

  private void reshapeComponents(Panel cp) {
    cp.setBounds      (  0,  0, 460, 100);

    eyePic.setBounds  (  5,  5,  90,  90);
    eyeSB.setBounds   (100,  5,  14,  90);
    lbFar.setBounds   (120,  5,  40,  15);
    lbNear.setBounds  (120, 80,  40,  15);

    bMonthU.setBounds (170,  5,  10,  10);
    bMonthD.setBounds (170, 15,  10,  10);
    lbDate.setBounds  (180,  7,  50,  20);
    bDayU.setBounds   (230,  5,  10,  12);
    bDayD.setBounds   (230, 15,  10,  10);

    bHourU.setBounds  (270,  5,  10,  10);
    bHourD.setBounds  (270, 15,  10,  10);
    lbTime.setBounds  (280,  7,  50,  20);
    bMinU.setBounds   (330,  5,  10,  10);
    bMinD.setBounds   (330, 15,  10,  10);

    lbError.setBounds (350,  7, 100,  20);

    bStep.setBounds   (170, 41, 120,  20);
    chAmount.setBounds(295, 40,  80,  20);
    //chAmount.setSize(80, 20); // To get Netscape to work!

    bClear.setBounds  (170, 70,  90,  20);
    bTrace.setBounds  (283, 70,  93,  20);
```

```
    bQuit.setBounds   (410, 70,  40,  20);
  }

  public void toggleStepping() {
    stepping = !stepping;
  }

  public void incMonth() {
    if(++month>12) month = 1;
    updateLabels();
  }

  public void decMonth() {
    if(--month<1) month = 12;
    updateLabels();
  }

  public void incDay() {
    if(++day>daysInMonth(month)) { incMonth(); day = 1; }
    updateLabels();
  }

  public void decDay() {
    if(--day<1) { decMonth(); day = daysInMonth(month); }
    updateLabels();
  }

  public void incHour() {
    if(++hour>=24) hour = 0;
    updateLabels();
  }

  public void decHour() {
    if(--hour<0) hour = 23;
    updateLabels();
  }

  public void incMin() {
    if(++min>=60) { min = 0; incHour(); }
    updateLabels();
  }

  public void decMin() {
```

```
    if(--min<0) { min = 59; decHour(); }
    updateLabels();
  }

  private void stepTime() {
    min += minStep;
    if(min>=60) { min -= 60; hour++; }
    hour += hourStep;
    if(hour>=18) {
       hour += 12; // Skip 18.00 to 06.00
       if (penDown) m.tracePoint(null);
    }
    if(hour>=24) { hour -= 24; day++; }
    day += dayStep;
    if(day>daysInMonth(month)) {
      day -= daysInMonth(month);
      month++;
    }
    if(month>12) month=1;
    //System.out.println("month = " + month +
    //                  " day = "   + day +
    //                  " hour = "  + hour +
    //                  " min = "   + min);
    updateLabels();
  }

  private void updateLabels() {
      m.setTime(month, day, hour, min);

      int errSecs = (int)(60.0 * m.minsSlow);
      int absSecs = Math.abs(errSecs);

      lbDate.setText(""+monString(month)+" "+day);
      lbTime.setText(""+hour+":"+min);
      lbError.setText((errSecs>0 ? " Slow by " : "Fast by ")+
                    absSecs/60+":"+absSecs%60);
      v.repaint(20);
      repaint(20);
  }

  private int daysInMonth(int month) {
    switch(month) {
    case  1: return 31;
```

```
    case  2: return 28;
    case  3: return 31;
    case  4: return 30;
    case  5: return 31;
    case  6: return 30;
    case  7: return 31;
    case  8: return 31;
    case  9: return 30;
    case 10: return 31;
    case 11: return 30;
    case 12: return 31;
    }
    return 0;
  }

  private String monString(int month) {
    switch(month) {
    case  1: return "JAN";
    case  2: return "FEB";
    case  3: return "MAR";
    case  4: return "APR";
    case  5: return "MAY";
    case  6: return "JUN";
    case  7: return "JUL";
    case  8: return "AUG";
    case  9: return "SEP";
    case 10: return "OCT";
    case 11: return "NOV";
    case 12: return "DEC";
    }
    return "???";
  }


    // Deal with all kinds of events

    private double SVtoD(int scrollValue) {
         return 3.5 + 30.0*(100.0-scrollValue)/100;
    }

  public void adjustmentValueChanged(AdjustmentEvent e) {
      // Handle scrollbar events
      // The scroll value is between 0 and 100
```

```
      //System.out.println("scroll value = " + e.getValue());
      double d = SVtoD(e.getValue());
      v.setEyeD(d);
  }

  public void actionPerformed(ActionEvent e) {
    // Deal with button actions
    Object source = e.getSource();
    if (source == bStep) {
        toggleStepping();
        bStep.setLabel(stepping ? "Stop Stepping by" :
                                  "Start Stepping by");
        return;
    }
    if (source == bTrace) {
        if (m.penDown) {
          m.penDown = false;
          m.tracePoint(null);
          bTrace.setLabel("Pen Down");
        } else {
          m.penDown = true;
          m.tracePoint(m.S);
          bTrace.setLabel("Pen Up");
        }
        return;
    }

    if (source == bClear) {
        m.tracePoints = null;
        v.repaint(20);
        return;
      }
      if (source == bQuit) {
      //System.out.println("Quitting");
      die(); // Cause the ticker thread to die
          return;
      }
      if (source == bMonthU) { incMonth(); return; }
      if (source == bMonthD) { decMonth(); return; }
      if (source == bDayU)   {   incDay(); return; }
      if (source == bDayD)   {   decDay(); return; }
      if (source == bHourU)  {  incHour(); return; }
      if (source == bHourD)  {  decHour(); return; }
```

```
    if (source == bMinU)   {   incMin(); return; }
    if (source == bMinD)   {   decMin(); return; }
}

public void itemStateChanged(ItemEvent e) {
  int days=0, mins=0;
  String s = (String)e.getItem();

  if(s == "1 min")   mins =  1;
  if(s == "5 min")   mins =  5;
  if(s == "12 mins") mins = 20;
  if(s == "1 hour")  mins = 60;
  if(s == "1 day")   days =  1;
  if(s == "5 days")  days =  5;
  if(s == "15 days") days = 15;

  dayStep  = days;
  hourStep = mins/60;
  minStep  = mins%60;
}

  // Method definitions for the MouseListener Interface
  //   mouseclicked    mouse pressed or released
  //   mouseEntered    mouse entered a component
  //   mouseExited     mouse exited a component
  //   mousePressed    mouse button pressed
  //   mouseReleased   mouse button released

  public void mouseReleased(MouseEvent e) { }
  public void mouseEntered (MouseEvent e) { }
  public void mouseExited  (MouseEvent e) { }
  public void mouseClicked (MouseEvent e) { }
  public void mousePressed (MouseEvent e) {
    mousePos(e.getX(), e.getY());
  }

  // Mouse motion events
  public void mouseMoved   (MouseEvent e) { }
  public void mouseDragged (MouseEvent e) {
      mousePos(e.getX(), e.getY());
  }

  private void mousePos(int x, int y) {
```

```
        System.out.println("SunDial: calling setOrigin " + x + " y=" +y);
        v.setOrigin(x, y);
        repaint(20);
    }

    // KeyBoard Events

    public void keyTyped   (KeyEvent e) { }
    public void keyReleased(KeyEvent e) { }
    public void keyPressed (KeyEvent e) {
        int ch = e.getKeyChar();
        System.out.println("Controls: keyPressed ch=" + ch);
        keych(ch);
    }

    public void keych(int key) {
      switch (key) {
            default:
                    System.out.println("Unexpected key pressed " + key);
                    return;
        case 'M':   incMonth(); return;
        case 'm':   decMonth(); return;
        case 'D':   incDay();   return;
        case 'd':   decDay();   return;
        case 'H':   incHour();  return;
        case 'h':   decHour();  return;
        case 'q':   die();      return;
            case 's':   toggleStepping(); return;
            case 'c':   m.tracePoints = null;
                        v.repaint(20);
                        return;

            case 'p':
                        if (m.penDown) {
                          m.penDown = false;
                          m.tracePoint(null);
                          bTrace.setLabel("Pen Down");
                        } else {
                          m.penDown = true;
                          m.tracePoint(m.S);
                          bTrace.setLabel("Pen Up");
                        }
                        return;
```

```
            case '<':
            case '>':
            case '+':
            case '-':
            case ' ':System.out.println("Key pressed " + key);

        // The following does not work
            case Event.UP:       v.eyeR.y--; return;
            case Event.DOWN:     v.eyeR.y++; return;
            case Event.LEFT:     v.eyeR.x--; return;
            case Event.RIGHT:    v.eyeR.x++; return;
    }
  }
}


class EyePic extends Panel //Canvas
  implements MouseListener, MouseMotionListener
{
    private Model m;
    private Viewer v;
    private Controls c;

    private int sz;

    public EyePic(Model m, Viewer v, Controls c, int sz) {
        this.m = m;
        this.v = v;
        this.c = c;
        this.sz = sz;
        addMouseListener(this);
        addMouseMotionListener(this);
    }

    public void update(Graphics g) {
        paint(g);
    }

    public void paint(Graphics g) {
        g.setColor(m.colBackground); // Draw the background
        g.fillRect(0, 0, sz, sz);
        g.setColor(m.colRoof);       // Draw the roof
```

```
    g.fillRect(WtoEx(m.D.x),              WtoEy(m.D.y),
               WtoEx(m.F.x)-WtoEx(m.D.x), WtoEy(m.F.y)- WtoEy(m.D.y));
    g.setColor(m.colGround);     // Draw the "garden"
    g.fillRect(WtoEx(m.E.x),              WtoEy(m.E.y),
               WtoEx(m.B.x)-WtoEx(m.E.x), WtoEy(m.B.y)- WtoEy(m.E.y));
    g.setColor(Color.white);     // Draw the eye line
    g.drawLine(WtoEx(v.eyeG.x), WtoEy(v.eyeG.y),
               WtoEx(v.eyeR.x), WtoEy(v.eyeR.y));
}

// Convert World coordinates to EyePic coordinates
//  eg   ( 4.0, -2.0) -> (0,   0)
//       (-4.0,  6.0) -> (sz, sz)
private int WtoEx(double x) {
    //System.out.println("Wx "+x+" -> "+(int) Math.round(sz * (4.0-x)/8.0));
    return (int) Math.round(sz * (4.0-x)/8.0);
}

private int WtoEy(double y) {
    //System.out.println("Wy "+y+" -> "+(int) Math.round(sz * (2.0+y)/8.0));
    return (int) Math.round(sz * (2.0+y)/8.0);
}

// Convert EyePic coordinates to World coordinates
//  eg   (0,   0)  ->  ( 4.0, -2.0)
//       (sz, sz)  ->  (-4.0,  6.0)
private double EtoWx(double x) {
      return 4.0 - 8.0*x/sz;
}

private double EtoWy(double y) {
      return -2.0 + 8.0*y/sz;
}


// EyePic mouse events
public void mousePressed(MouseEvent e) {
  seteye(e.getX(), e.getY());
}
public void mouseReleased(MouseEvent e) { }
public void mouseEntered (MouseEvent e) { }
public void mouseExited  (MouseEvent e) { }
public void mouseClicked (MouseEvent e) { }
```

```
    // EyePic mouse motion events
    public void mouseDragged(MouseEvent e) {
      seteye(e.getX(), e.getY());
    }
    public void mouseMoved(MouseEvent e) { }

    private void seteye(int x, int y) {
    //System.out.println("seteye: "+x+","+y+" sz="+sz);
        v.requestFocus();
        v.setEyeR(new Point3(EtoWx(x), EtoWy(y), 1));
        repaint(20);   // To repaint the eyePic
        v.repaint(20); // and repaint the sundial
    }
}


//********************************************** ScrPoint

// This class is used to represent points on the screen.

class ScrPoint
{
    int x, y;  // The pixel position of a point.

    public ScrPoint(int x, int y) {
        this.x = x;
        this.y = y;
  }
}

//********************************************** Point3

class Point3
{
  double x, y, z;

  public Point3(double x, double y, double z) {
      this.x = x;
      this.y = y;
      this.z = z;
  }
```

```
  public Point3 add(Point3 p) {
    return new Point3(x+p.x, y+p.y, z+p.z);
  }

  public Point3 sub(Point3 p) {
    return new Point3(x-p.x, y-p.y, z-p.z);
  }

  public Point3 mul(double a) {
    return new Point3(x*a, y*a, z*a);
  }

  public Point3 cosines() {
    double len = this.length();
    if(len==0) return new Point3(1.0,0.0,0.0);
    return new Point3(x/len, y/len, z/len);
  }

  public double length() {
    return Math.sqrt(x*x + y*y + z*z);
  }

  public double innerProd(Point3 p) {
    return x*p.x + y*p.y + z*p.z;
  }

  public Point3 crossProd(Point3 p) {
    return new Point3(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);
  }
}
```