```
// AirMonitor.java - AIR POLLUTION MONITOR
//
// This program uses two optimisations to achieve the required performance.
//
// First, instead of evaluating the weighting function for each
// pixel/sensor combination, the values of the weighting function are
// loaded into a lookup table for each x-y offset from the pixel being
// evaluated.  This optimisation generates a performance improvement of a
// factor of around four.
//
// Second, the surface of the air pollution map is divided into grid
// quadrants.  A preliminary scan of the sensor data determines which
// quadrant each sensor lies in.  The API interpolation function only
// processes the sensors that lie in the grid quadrants within the
// significance range of the weighting function.  This achieves a
// performance improvement of a factor of around 10.
//
// MODULE INDEX
// NAME                                 CONTENTS
// AirData                              Air pollution data structure
// Refresher                            Refresher class
// Refresher.run                        Code to be executed in the event
//                                      processing thread
// Refresher.Refresher                  Refresher constructor
// ImageGenerator                       Image generator class
// ImageGenerator.ImageGenerator        Image generator constructor
// ImageGenerator.dmsToTenths           Convert a degrees, minutes and seconds
//                                      string into tenths of seconds
// ImageGenerator.rawWeight             Raw weighting function
// ImageGenerator.createWeightTable     Create weight lookup table
// ImageGenerator.weight                Weighting function
// ImageGenerator.readAirDataRec        Read an air data record
// ImageGenerator.estimateApi           Estimate the API at a point
// ImageGenerator.getPixelColor         Get pixel colour
// ImageGenerator.generateImage         Generate image
// ImageGenerator.run                   Image generator main line
// init                                 Initiation
// paint                                Paint the image
// start                                Start execution of the applet
// stop                                 Stop execution of the applet
// getAppletInfo                        Get applet information
// imageUpdate                          Image updated callback function
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 26-09-06     JS      Original
// 03-09-07     SHL     Check for null element in airDataArr
// 03-09-07     SHL     Exclude download time from refresh time statistic
//
//-----------------------------------------------------------------------------

// IMPORTS

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.lang.*;

//-----------------------------------------------------------------------------
```

```
// AIR POLLUTION MONITOR CLASS DEFINITION

public class AirMonitor
extends Applet
implements ImageObserver
{
    //---------------------------------------------------------------------------

    // DEFINITIONS

    static private final String AIRDATA_NAME = "airdata.txt";
                                        // Air pollution data file name
    static private final String FROM_LONG = "10120000";
                                        // From longitude (dddmmsst)
    static private final String FROM_LAT = "00250000";
                                        // From latitude (dddmmsst)
    static private final String TO_LONG = "10150000";
                                        // From longitude (dddmmsst)
    static private final String TO_LAT = "00320000";
                                        // From latitude (dddmmsst)
    static private final double SCALE = 330000;
                                        // Scale
    static private final double INCH_PER_MILE = 72913;
                                        // Inches per nautical mile
    static private final double TENTHS_PER_MILE = 600;
                                        // Tenths of a second per nautical mile
    static private final long REFRESH_PERIOD = 30000;
                                        // Refresh period in milliseconds
    static private final int INI_AIR_DATA_CAP = 600;
                                        // Initial air data vector capacity
    static private final int AIR_DATA_CAP_INC = 100;
                                        // Air data vector capacity increment
    static private final double A = 0.9;
                                        // The factor `A'
    static private final Color GOOD_COLOR = new Color (0, 204, 255);
                                        // Good status color
    static private final Color MODERATE_COLOR = new Color (0, 255, 64);
                                        // Moderate status color
    static private final Color UNHEALTHY_COLOR = new Color (255, 255, 0);
                                        // Unhealthy status color
    static private final Color VERY_UNHEALTHY_COLOR = new Color (255, 153, 0);
                                        // Very unhealthy status color
    static private final Color HAZARDOUS_COLOR = new Color (255, 0, 0);
                                        // Hazardous status color
    static private final Color EMERGENCY_COLOR = new Color (255, 255, 255);
                                        // Emergency status color
    static private final double MIN_SIGNIFICANT_WEIGHT = 1.0e-6;
                                        // Minimum significant weight
    static private final int GRID_SIZE = 30;
                                        // Grid size in pizels

    //---------------------------------------------------------------------------

    // AIR POLLUTION DATA STRUCTURE

    private class AirData {
        int             x;              // X position in pixels
        int             y;              // Y position in pixels
        int             api;            // Air pollution index
        int             next;           // Next element in grid position
    };

    //---------------------------------------------------------------------------
```

```
// INSTANCE DATA

private Image       currentImage;   // Current image being displayed
private ImageGenerator imageGenerator; // Reference to image generator

//---------------------------------------------------------------------------

// REFRESHER CLASS

private class Refresher
implements Runnable
{
    //-----------------------------------------------------------------------

    // CLASS INSTANCE VARIABLES

    Image           newImage;       // New image to refresh

    //-----------------------------------------------------------------------

    // CODE TO BE EXECUTED IN THE EVENT PROCESSING THREAD

    public void run ()
    {
        currentImage = newImage;
        repaint ();
    }

    //-----------------------------------------------------------------------

    // CONSTRUCTOR

    Refresher (
        Image       img)            // New image
    {
        newImage = img;
    }
}

//---------------------------------------------------------------------------

// IMAGE GENERATOR CLASS

private class ImageGenerator
extends Thread
{
    //-----------------------------------------------------------------------

    // CLASS INSTANCE DATA

    private int     fromLong;       // From longitude in tenths of seconds
    private int     fromLat;        // From latitude in tenths of seconds
    private int     toLong;         // To longitude in tenths of seconds
    private int     toLat;          // To latitude in tenths of seconds
    private double  netScale;       // Effective scale (tenths to pixels)
    private int     mapWidth;       // Air polution map width in pixels
    private int     mapHeight;      // Air polution map height in pixels
    private int     weightRange;    // Weight range
    private double[] weightTable;   // Weight table
    private int     gridWidth;      // Grid width
    private int     gridHeight;     // Grid height
    private int     gridRange;      // Grid search range
    private int[]   gridHeaders;    // Grid bucket headers
```

```
//---------------------------------------------------------------------

// CONSTRUCTOR

ImageGenerator ()
{
    super ("ImageGenerator");
}

//---------------------------------------------------------------------

// CONVERT A DEGREES, MINUTES AND SECONDS STRING TO TENTHS OF SECONDS

private int
dmsToTenths (
    String      dms)            // Degrees, minutes and seconds
{
    int         tenths;         // Tenths of seconds

    tenths = Character.digit (dms.charAt(0), 10);
    tenths = tenths * 10 + Character.digit (dms.charAt(1), 10);
    tenths = tenths * 10 + Character.digit (dms.charAt(2), 10);
    tenths = tenths * 6 + Character.digit (dms.charAt(3), 10);
    tenths = tenths * 10 + Character.digit (dms.charAt(4), 10);
    tenths = tenths * 6 + Character.digit (dms.charAt(5), 10);
    tenths = tenths * 10 + Character.digit (dms.charAt(6), 10);
    tenths = tenths * 10 + Character.digit (dms.charAt(7), 10);
    return tenths;
}

//---------------------------------------------------------------------

// RAW WEIGHTING FUNCTION

double
rawWeight (
    int                 x,              // Horizontal offset
    int                 y)              // Vertical offset
{
    double              d;              // Distance

    d = x * x + y * y;
    d *= (netScale * netScale) / (TENTHS_PER_MILE * TENTHS_PER_MILE);
    d /= A * A;
    return Math.exp (-d);
}

//---------------------------------------------------------------------

// CREATE THE WEIGHT LOOKUP TABLE

private void
createWeightTable ()
{
    int                 x;              // Horizontal position
    int                 y;              // Vertical position
    double              w;              // Weight

    // Determine the range before the weight of a point
    // becomes insignificant

    weightRange = 0;
    do {
        weightRange ++ ;
```

```
        w = rawWeight (weightRange, 0);
    } while (w >= MIN_SIGNIFICANT_WEIGHT);

    // Create the weight table

    weightTable = new double [weightRange*weightRange];

    // Load the weight table

    for (x = 0; x < weightRange; x++)
        for (y = 0; y < weightRange; y++)
            weightTable[y*weightRange + x] = rawWeight (x, y);
}

//---------------------------------------------------------------------

// WEIGHTING FUNCTION

double
weight (
    int                 x,              // Horizontal position
    int                 y,              // Vertical position
    int                 xi,             // Sensor horizontal position
    int                 yi)             // Sensor vertical position
{
    x -= xi;
    if (x < 0) x = -x;
    y -= yi;
    if (y < 0) y = -y;
    if (x < weightRange && y < weightRange)
        return weightTable [y*weightRange + x];
    return 0;
}

//---------------------------------------------------------------------

// READ AN AIR DATA RECORD

private AirData
readAirDataRec (
    InputStream         airDataStream)  // airdata.txt input stream
throws IOException
{
    int                 ch;             // Input byte
    int                 longHemi;       // Longitude hemisphere
    int                 longTenths;     // Longitude in tenths of secs
    int                 latHemi;        // Latitude hemisphere
    int                 latTenths;      // Latitude in tenths of secs
    int                 api;            // Air pollution index
    int                 i;              // General purpose index
    StringBuffer        dms;            // Degrees, minutes, seconds
    AirData             airDataRec;     // Air data record

    // Read the longitude hemisphere

    ch = airDataStream.read();
    if (ch == -1) return null;
    if (ch != 'E' && ch != 'W')
        throw new RuntimeException ("invalid longitude hemisphere");
    longHemi = ch;

    // Read the longitude

    dms = new StringBuffer();
```

```
    for (i = 0; i < 8; i++) {
        ch = airDataStream.read();
        if (ch == -1) return null;
        if ( ! Character.isDigit ((char)ch))
            throw new RuntimeException ("invalid longitude char");
        dms.append ((char)ch);
    }
    longTenths = dmsToTenths (dms.toString());
    if (longHemi == 'W') longTenths = -longTenths;

    // Read the latitude hemisphere

    ch = airDataStream.read();
    if (ch == -1) return null;
    if (ch != 'N' && ch != 'S')
        throw new RuntimeException ("invalid latitude hemisphere");
    latHemi = ch;

    // Read the latitude

    dms = new StringBuffer();
    for (i = 0; i < 8; i++) {
        ch = airDataStream.read();
        if (ch == -1) return null;
        if ( ! Character.isDigit ((char)ch))
            throw new RuntimeException ("invalid latitude char");
        dms.append ((char)ch);
    }
    latTenths = dmsToTenths (dms.toString());
    if (latHemi == 'S') latTenths = -latTenths;

    // Read the air pollution index

    api = 0;
    for (i = 0; i < 3; i++) {
        ch = airDataStream.read();
        if (ch == -1) return null;
        if ( ! Character.isDigit ((char)ch))
            throw new RuntimeException ("invalid API char");
        api = api * 10 + Character.digit ((char)ch, 10);
    }

    // Skip the new line

    ch = airDataStream.read();
    if (ch == '\r')
        ch = airDataStream.read();
    if (ch != '\n' && ch != -1)
        throw new RuntimeException ("invalid end-of-line");

    // Load and return the air data record

    airDataRec = new AirData();
    airDataRec.x = (int)((longTenths - fromLong) / netScale);
    airDataRec.y = (int)((latTenths - fromLat) / netScale);
    airDataRec.api = api;
    return airDataRec;
}

//---------------------------------------------------------------------

// ESTIMATE THE API AT A POINT

double
```

```
estimateApi (
    AirData[]           airDataArr,     // Air data array
    int                 x,              // Horizontal position
    int                 y)              // Vertical position
{
    double              wt;             // Weight of this point
    double              sumApiWt;       // Sum of APIs and weights
    double              sumWt;          // Sum of weights
    int                 i;              // General purpose index
    AirData             airData;        // Air pollution data reference
    int                 gx, gy;         // Grid co-ordinates
    int                 gxMin;          // Grid horizontal minimum
    int                 gxMax;          // Grid horizontal maximum
    int                 gyMin;          // Grid vertical minimum
    int                 gyMax;          // Grid vertical maximum

    // Calculate the grid limits

    gx = x / GRID_SIZE;
    gy = y / GRID_SIZE;
    gxMin = gx - gridRange;
    if (gxMin < 0) gxMin = 0;
    gxMax = gx + gridRange + 1;
    if (gxMax > gridWidth) gxMax = gridWidth;
    gyMin = gy - gridRange;
    if (gyMin < 0) gyMin = 0;
    gyMax = gy + gridRange + 1;
    if (gyMax > gridHeight) gyMax = gridHeight;

    // Accumulate sums for the sensors that fall within
    // the grid limits

    sumApiWt = 0;
    sumWt = 0;
    for (gx = gxMin; gx < gxMax; gx++) {
        for (gy = gyMin; gy < gyMax; gy++) {
            for (
                i = gridHeaders[gy*gridWidth + gx];
                i != -1;
                i = airData.next
            ) {
                airData = airDataArr[i];
                wt = weight (x, y, airData.x, airData.y);
                sumApiWt += wt * airData.api;
                sumWt += wt;
            }
        }
    }
    if (sumWt == 0) return 0;
    return sumApiWt / sumWt;
}

//-------------------------------------------------------------------------

// GET PIXEL COLOUR

Color
getPixelColor (
    AirData[]           airDataArr,     // Air data array
    int                 x,              // Horizontal position
    int                 y)              // Vertical position
{
    double              api;            // Air pollution index
```

```
    api = estimateApi (airDataArr, x, y);
    if (api < 51)
        return GOOD_COLOR;
    if (api < 101)
        return MODERATE_COLOR;
    if (api < 201)
        return UNHEALTHY_COLOR;
    if (api < 301)
        return VERY_UNHEALTHY_COLOR;
    if (api < 501)
        return HAZARDOUS_COLOR;
    return EMERGENCY_COLOR;
}

//-----------------------------------------------------------------------

// GENERATE IMAGE

Image
generateImage (
    AirData[]           airDataArr)     // Air data array
{
    Image               newImage;       // New screen image
    Graphics            g;              // Graphics context of image
    int                 x, y;           // Screen co-ordinates
    int                 gx, gy;         // Grid co-ordinates
    int                 gi;             // Grid index
    int                 i;              // General purpose index

    // Initialise the grid headers and footers

    for (gx = 0; gx < gridWidth; gx++)
        for (gy = 0; gy < gridHeight; gy++)
            gridHeaders[gy*gridWidth + gx] = -1;

    // Put the sensors into the appropriate grids

    for (i = 0; i < airDataArr.length && airDataArr[i] != null ; i++) {
        gx = airDataArr[i].x / GRID_SIZE;
        gy = airDataArr[i].y / GRID_SIZE;
        gi = gy * gridWidth + gx;
        airDataArr[i].next = gridHeaders[gi];
        gridHeaders[gi] = i;
    }

    // Set the colour of each pixel

    newImage = createImage (mapWidth, mapHeight);
    g = newImage.getGraphics();
    for (x = 0; x < mapWidth; x++) {
        for (y = 0; y < mapHeight; y++) {
            g.setColor (getPixelColor (airDataArr, x, y));
            g.fillRect (x, mapHeight-y-1, 1, 1);
        }
    }
    return newImage;
}

//-----------------------------------------------------------------------

// IMAGE GENERATOR MAIN LINE

public void
run ()
```

```
{
    int                 resolution;     // Screen resolution (pix/inch)
    long                startTime;      // Time image generation started
    long                waitTime;       // Time to wait for next refresh
    URL                 airDataUrl;     // airdata.txt URL
    HttpURLConnection   airDataCon;     // airdata connection instance
    InputStream         airDataStream;  // airdata.txt input stream
    Vector<AirData>     airDataVec;     // Air pollution data vector
    AirData             airDataRec;     // Air data record
    Image               newImage;       // New image

    // Convert the bounds

    fromLong = dmsToTenths (FROM_LONG);
    fromLat = dmsToTenths (FROM_LAT);
    toLong = dmsToTenths (TO_LONG);
    toLat = dmsToTenths (TO_LAT);

    // Calculate the net scale

    resolution = getToolkit().getScreenResolution();
    netScale = SCALE * TENTHS_PER_MILE / (INCH_PER_MILE * resolution);

    // Calculate the map dimensions in pixels

    mapWidth = (int)((toLong - fromLong) / netScale) + 1;
    mapHeight = (int)((toLat - fromLat) / netScale) + 1;

    // Create the weight lookup table

    createWeightTable ();

    // Calculate the grid parameters

    gridWidth = (mapWidth + GRID_SIZE - 1) / GRID_SIZE;
    gridHeight = (mapHeight + GRID_SIZE - 1) / GRID_SIZE;
    gridRange = (weightRange + GRID_SIZE - 1) / GRID_SIZE;
    gridHeaders = new int [gridWidth * gridHeight];

    // Catch interrupted exceptions

    try {

        // Execute refresh cycles until interrupted

        while ( ! Thread.interrupted()) {

            // Read the air pollution data file

            try {
                airDataUrl = new URL (getCodeBase(), AIRDATA_NAME);
                airDataCon = (HttpURLConnection)
                    airDataUrl.openConnection();
                airDataStream = airDataCon.getInputStream ();
                airDataVec = new Vector<AirData>
                    (INI_AIR_DATA_CAP, AIR_DATA_CAP_INC);
                for (;;) {
                    airDataRec = readAirDataRec (airDataStream);
                    if (airDataRec == null) break;
                    if (
                        airDataRec.x >= 0 &&
                        airDataRec.x < mapWidth &&
                        airDataRec.y >= 0 &&
                        airDataRec.y < mapHeight
```

```
                    )
                        airDataVec.add (airDataRec);
                }
                airDataStream.close ();
            }
            catch (IOException e) {
                throw new RuntimeException (e.toString());
            }

            // Record the time when image generation started

            startTime = System.currentTimeMillis();

            // Generate a new image

            newImage = generateImage (
                airDataVec.toArray(new AirData[1]));

            // Send the image to the main thread

            try {
                EventQueue.invokeAndWait (new Refresher(newImage));
            }
            catch (InterruptedException e) {
                throw e;
            }
            catch (Exception e) {
                throw new RuntimeException (e.toString());
            }

            // Wait for the next refresh

            System.out.println ("Refresh took " +
                (System.currentTimeMillis() - startTime) + "ms");
            waitTime = REFRESH_PERIOD -
                (System.currentTimeMillis() - startTime);
            if (waitTime > 0) sleep (waitTime);
        }
    }
    catch (InterruptedException e) {
        // Empty
    }
  }
}

//---------------------------------------------------------------------------

// INITIATION

public void
init ()
{
    // Initialise the image

    currentImage = null;

    // Initiate the Image Generator

    imageGenerator = new ImageGenerator ();
}

//---------------------------------------------------------------------------

// PAINT THE IMAGE
```

```
    public void
    paint (
        Graphics        g)              // Reference to graphics instance
    {
        if (currentImage != null)
            g.drawImage (currentImage, 0, 0, this);
    }

    //------------------------------------------------------------------------

    // START EXECUTION OF THE APPLET

    public void
    start ()
    {
        super.start ();
        imageGenerator.start ();
    }

    //------------------------------------------------------------------------

    // STOP EXECUTION OF THE APPLET

    public void
    stop ()
    {
        imageGenerator.interrupt ();
        try {
            imageGenerator.join ();
        }
        catch (InterruptedException e) {
            // Empty
        }
        super.stop ();
    }

    //------------------------------------------------------------------------

    // GET APPLET INFORMATION

    public String
    getAppletInfo ()
    {
        return "AirMonitor, Copyright, 2007, E-Genting Sdn. Bhd.";
    }

    //------------------------------------------------------------------------

    // IMAGE UPDATED CALLBACK FUNCTION

    public boolean
    imageUpdate (
        Image           img,
        int             flags,
        int             x,
        int             y,
        int             w,
        int             h)
    {
        return (flags & (ALLBITS|ABORT|ERROR)) == 0;
    }
}
```