```
// TimeClient.java - NETWORK CLOCK CLIENT
//
// MODULE INDEX
// NAME                 CONTENTS
// getServerTime        Get the current time from the server
// getServerTimeRange   Get server time range
// getCurrentTimeRange  Adjust the time range for the time elapsed since the
//                      sample was taken
// Refresh.run          Refresh the display
// Resync.run           Resynchronise the time with the server time
// paint                Paint the clock face
// update               Update the clock face
// init             Initiation
// start                Start execution of the applet
// stop                 Stop execution of the applet
// getAppletInfo        Return applet information
//
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 26-09-12 JS  Original
//
//-------------------------------------------------------------------------

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

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

// CLASS DECLARATION

public class TimeClient
extends Applet
{
    //---------------------------------------------------------------------

    // DEFINITIONS

    static final int    ORIGIN_X = 150;      // Origin in the X direction
    static final int    ORIGIN_Y = 150;      // Origin in the Y direction
    static final int    RADIUS = 100;        // Radius of the clock face
    static final int    MINUTE_RADIUS = 95; // Radius of the minute hand
    static final int    HOUR_RADIUS = 65;    // Radius of the hour hand
    static final int    ERROR_RATE = 10;     // Error rate in seconds per day

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

    // TIME RANGE CLASS

    class TimeRange {
    long        fromTime;    // From-time in millis
    long        toTime;      // To-time in millis
```

```
    long        sampleTime; // time at which the range applies
    }

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

    // INSTANCE DATA

    TimeRange       currentRange;   // Current time range
    long        sampleTime;         // Time at which range applies
    Thread      refreshThread;      // Refresh thread
    Thread      resyncThread;       // Resynchronise thread

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

    // GET THE CURRENT TIME FROM THE SERVER

    GregorianCalendar
    getServerTime ()
    throws IOException
    {
    URL         url;                // URL instance
    HttpURLConnection   con;        // Connection instance
    InputStreamReader   isr;        // Input stream reader
    StreamTokenizer     strTok;     // Stream tokenizer
    int         ch;                 // Input character
    int         day, month, year;   // Date components
    int         hour, min, sec;     // Time components
    GregorianCalendar   cal;        // Calendar instance

    // Months

    final String[] MONTH_ARR =
        {"jan", "feb", "mar", "apr", "may", "jun",
        "jul", "aug", "sep", "oct", "nov", "dec"};

    // Open a connection to the time server

    url = new URL (getCodeBase(), "TIME");
    con = (HttpURLConnection) url.openConnection();
    con.setDoOutput (false);
    con.setDoInput (true);
    if (con.getResponseCode() != HttpURLConnection.HTTP_OK)
        throw new RuntimeException ("rejected HTTP request:"
        + " code=" + con.getResponseCode()
        + " message=" + con.getResponseMessage());
    isr = new InputStreamReader (con.getInputStream(), "UTF-8");

    // Initialise the stream tokenizer

    strTok = new StreamTokenizer (isr);
    strTok.resetSyntax();
    strTok.wordChars ('a', 'z');
    strTok.wordChars ('A', 'Z');
    strTok.wordChars ('<', '<');
    strTok.wordChars ('>', '>');
    strTok.wordChars ('/', '/');
    strTok.whitespaceChars (' ', ' ');
```

```
    strTok.whitespaceChars ('\t', '\t');
    strTok.whitespaceChars ('\r', '\r');
    strTok.whitespaceChars ('\n', '\n');
    strTok.parseNumbers ();
    strTok.lowerCaseMode (true);

    // Validate <html>

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_WORD
        || ! strTok.sval.equals("<html>"))
        throw new RuntimeException ("missing <html>");

    // Validate <body>

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_WORD
        || ! strTok.sval.equals("<body>"))
        throw new RuntimeException ("missing <body>");

    // Skip the day

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_WORD)
        throw new RuntimeException ("missing day-of-week");

    // Skip the comma

    strTok.nextToken();
    if (strTok.ttype != ',')
        throw new RuntimeException ("missing comma");

    // Read the day of the month

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_NUMBER)
        throw new RuntimeException ("missing day-of-month");
    if (strTok.nval < 0 || strTok.nval > 31)
        throw new RuntimeException ("invalid day-of-month");
    day = (int)strTok.nval;

    // Read the month name

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_WORD)
        throw new RuntimeException ("missing month");
    month = 0;
    while (month < MONTH_ARR.length &&
        ! MONTH_ARR[month].equals (strTok.sval)) month ++ ;
    if (month >= MONTH_ARR.length)
        throw new RuntimeException ("invalid month");
    month ++ ;

    // Skip the year

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_NUMBER)
```

```
        throw new RuntimeException ("missing year");
    if (strTok.nval < 0 || strTok.nval > 3000)
        throw new RuntimeException ("invalid year");
    year = (int)strTok.nval;

    // Read the hour

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_NUMBER)
        throw new RuntimeException ("missing hour");
    if (strTok.nval < 0 || strTok.nval > 23)
        throw new RuntimeException ("invalid hour");
    hour = (int)strTok.nval;

    // Skip the colon

    strTok.nextToken();
    if (strTok.ttype != ':')
        throw new RuntimeException ("missing colon");

    // Read the minutes

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_NUMBER)
        throw new RuntimeException ("missing minutes");
    if (strTok.nval < 0 || strTok.nval > 59)
        throw new RuntimeException ("invalid minutes");
    min = (int)strTok.nval;

    // Skip the colon

    strTok.nextToken();
    if (strTok.ttype != ':')
        throw new RuntimeException ("missing colon");

    // Read the seconds

    strTok.nextToken();
    if (strTok.ttype != StreamTokenizer.TT_NUMBER)
        throw new RuntimeException ("missing seconds");
    if (strTok.nval < 0 || strTok.nval > 59)
        throw new RuntimeException ("invalid seconds");
    sec = (int)strTok.nval;

    // Quietly the remainder of the response

    isr.close ();

    // Return a calendar instance

        return new GregorianCalendar (year, month, day, hour, min, sec);
    }

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

    // GET SERVER TIME RANGE
```

```
TimeRange
getServerTimeRange ()
throws IOException
{
long            startTime;   // Query start time
long            endTime;     // Query end time
GregorianCalendar   serverTime; // Current server time
TimeRange       timeRange;   // Received time range

startTime = System.currentTimeMillis();
serverTime = getServerTime ();
endTime = System.currentTimeMillis();
// Just incase someone changed the client time
if (endTime < startTime) endTime = startTime;
timeRange = new TimeRange();
timeRange.fromTime = serverTime.getTimeInMillis();
timeRange.toTime = timeRange.fromTime + endTime - startTime;
timeRange.sampleTime = endTime;
return timeRange;
}

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

// ADJUST THE TIME RANGE FOR THE TIME ELAPSED SINCE THE SAMPLE WAS TAKEN

TimeRange
getCurrentTimeRange ()
{
TimeRange       range;       // The current time range

// Make a copy of the current time range and update it for
// the time elapsed since the sample was taken

range = new TimeRange ();
range.sampleTime = System.currentTimeMillis();
synchronized (currentRange) {
    range.fromTime = currentRange.fromTime
    + range.sampleTime - currentRange.sampleTime
    - (range.sampleTime - currentRange.sampleTime) * ERROR_RATE
        / (24*60*60);
    range.toTime = currentRange.toTime
    + range.sampleTime - currentRange.sampleTime
    + (range.sampleTime - currentRange.sampleTime) * ERROR_RATE
        / (24*60*60);
}
return range;
}

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

// REFRESH THE DISPLAY

class Refresh implements Runnable {
public void
run ()
{
   try {
```

```
        for (;;) {
            Thread.sleep (1000);
            repaint (0, 0, getSize().width, getSize().height);
        }
        }
        catch (InterruptedException e) {
        // Empty
        }
    }
    }

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

    // RESYNCHRONISE THE TIME WITH THE SERVER TIME

    class Resync implements Runnable {
    public void
    run ()
    {
       TimeRange        oldRange;    // Old server time range
       TimeRange        newRange;    // New server time range

       try {
        for (;;) {

            // Wait for the resynchronisation period

            Thread.sleep (60000);

            // Quietly disregard I/O exceptions

            try {

            // Fetch a new time sample from the server

            newRange = getServerTimeRange ();

            // Adjust the current time range for the
            // imprecision of the local clock

            oldRange = getCurrentTimeRange ();

            // Block use of the current range while it is
            // being updated

            synchronized (currentRange) {

                // If the new range and old range intersect
                // set the new range to the intersection of the
                // old range and the new range

                if (
                newRange.toTime > oldRange.fromTime &&
                newRange.fromTime < oldRange.toTime
                ) {
                if (newRange.fromTime > oldRange.fromTime)
                    currentRange.fromTime = newRange.fromTime;
```

```
                else
                    currentRange.fromTime = oldRange.fromTime;
                if (newRange.toTime < oldRange.toTime)
                    currentRange.toTime = newRange.toTime;
                else
                    currentRange.toTime = oldRange.toTime;
                }

                // If the ranges do not intersect, assume the
                // old range is defective and use the new range

                else {
                currentRange.fromTime = newRange.fromTime;
                currentRange.toTime = newRange.toTime;
                }

                // Update the sample time

                currentRange.sampleTime = newRange.sampleTime;
            }
            }
            catch (IOException e) {
            // Empty
            }
        }
        }
        catch (InterruptedException e) {
        // Empty
        }
    }
    }

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

    // PAINT THE CLOCK FACE

    public void
    paint (
    Graphics        g)      // Reference to graphics inst
    {
    long            currentTime;     // Current time
    TimeRange       range;       // The time range to show
    long            meanTime;    // The average time
    GregorianCalendar   cal;     // Calendar instance
    int         hour, min, sec; // Time components
    int         milli;           // Milliseconds
    double          fromSec;     // From-seconds
    double          deltaSec;    // Delta-seconds
    double          realHour;    // Real hour
    double          realMin;     // Real minute
    double          realSec;     // Real seconds

    // Get the current time range

    range = getCurrentTimeRange ();

    // Erase the background
```

```
g.setColor (Color.WHITE);
g.fillRect (0, 0, getSize().width, getSize().height);

// Fill the seconds arc

meanTime = (range.fromTime + range.toTime) / 2;
cal = new GregorianCalendar ();
cal.setTimeInMillis (range.fromTime);
sec = cal.get (Calendar.SECOND);
milli = cal.get (Calendar.MILLISECOND);
fromSec = sec + milli/1000.0;
deltaSec = (range.toTime - range.fromTime) / 1000.0;
if (deltaSec > 60.0) deltaSec = 60.0;
g.setColor (Color.GREEN);
g.fillArc (
    ORIGIN_X-RADIUS, ORIGIN_Y-RADIUS, 2*RADIUS, 2*RADIUS,
    (int)(((15.0 - (fromSec+deltaSec)) * 6.0) + 0.5),
    (int)(deltaSec*6.0 + 0.5)
);

// Draw the hours and minutes
// Also draw seconds just in case the arc width is negligible

meanTime = (range.fromTime + range.toTime) / 2;
cal = new GregorianCalendar ();
cal.setTimeInMillis (meanTime);
hour = cal.get (Calendar.HOUR);
min = cal.get (Calendar.MINUTE);
sec = cal.get (Calendar.SECOND);
milli = cal.get (Calendar.MILLISECOND);
realHour = hour + min/60.0 + sec/3600.0 + milli/3600000.0;
realMin = min + sec/60.0 + milli/60000.0;
realSec = sec + milli/1000.0;
g.drawLine (
    ORIGIN_X, ORIGIN_Y,
    ORIGIN_X + (int)(RADIUS*Math.sin(realSec*Math.PI/30.0)+0.5),
    ORIGIN_Y - (int)(RADIUS*Math.cos(realSec*Math.PI/30.0)+0.5)
);
g.setColor (Color.BLACK);
g.drawLine (
    ORIGIN_X, ORIGIN_Y,
    ORIGIN_X + (int)(HOUR_RADIUS*Math.sin(realHour*Math.PI/6.0) + 0.5),
    ORIGIN_Y - (int)(HOUR_RADIUS*Math.cos(realHour*Math.PI/6.0) + 0.5)
);
g.drawLine (
    ORIGIN_X, ORIGIN_Y,
    ORIGIN_X + (int)(MINUTE_RADIUS*Math.sin(realMin*Math.PI/30.0)+0.5),
    ORIGIN_Y - (int)(MINUTE_RADIUS*Math.cos(realMin*Math.PI/30.0)+0.5)
);

// Draw the outline of the clock

g.setColor (Color.BLACK);
g.drawOval (ORIGIN_X-RADIUS, ORIGIN_Y-RADIUS, 2*RADIUS, 2*RADIUS);
}
```

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

// UPDATE THE IMAGE

public void
update (
Graphics        g)      // Reference to graphics inst
{
// Avoid the automatic refresh of the background
// by calling paint directly

paint (g);
}

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

// INITIATION

public void
init ()
{
URL         url;            // URL instance
HttpURLConnection   con;    // Connection instance
InputStreamReader   isr;    // Input stream reader
int         ch;             // Input character
GregorianCalendar   cal;    // Calandar instance

// Initialise the super-class

super.init ();

// Read the initial time

try {
    currentRange = getServerTimeRange();
}
catch (IOException e) {
    throw new RuntimeException (e.toString());
}

// Initiate the refresh thread and the resynchronisation thread

refreshThread = new Thread (new Refresh());
resyncThread = new Thread (new Resync());
}

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

// START EXECUTION OF THE APPLET

public void
start ()
{
super.start ();
refreshThread.start ();
resyncThread.start ();
}
```

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

    // STOP EXECUTION OF THE APPLET

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

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

    // RETURN APPLET INFORMATION

    public String
    getAppletInfo ()
    {
    return "Network Clock Client";
    }
}
```