```
// LoanVal.java - LOAN VALUATOR
//
// MODULE INDEX
// NAME                         CONTENTS
// presentValue                 Calculate the present value of the repayments
// actionPerformed              Process an action performed event
// init                         Initiation
// paint                        Paint the image
// start                        Start execution of the applet
// stop                         Stop execution of the applet
// getAppletInfo                Get applet information
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 04-10-06     JS      Original
// 26-10-06     HYC     Set the effective interest rate field un-focusable
//
//-----------------------------------------------------------------------------

// IMPORTS

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.text.*;

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

// LOAN VALUATOR CLASS DEFINITION

public class LoanVal
extends Applet
implements ActionListener
{
    //-------------------------------------------------------------------------

    // INSTANCE DATA

    static final int    MAX_PAY_TYPES = 5;      // Maximum payment types
    static final int    TRIAL_CNT = 16;         // Number of trials
    TextField           principalField;         // Principal field
    TextField []        startField;             // Start month fields
    TextField []        periodField;            // Period fields
    TextField []        quantityField;          // Payment quantity fields
    TextField []        amountField;            // Payment amount fields
    Button              calcButton;             // Calculate button
    TextField           rateField;              // Interest rate field

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

    // CALCULATE THE PRESENT VALUE OF THE REPAYMENTS

    double
    presentValue (
        int []          start,          // Starting month
        int []          period,         // Period between payments
        int []          quantity,       // Number of payments
        double []       amount,         // Payment amounts
        int             typeCnt,        // Number of payment types
        double          rate)           // Discount rate
    {
        int             i, j;           // General purpose index
        double          value;          // Net present value
        int             month;          // Month number

        value = 0;
```

```
    for (i = 0; i < typeCnt; i++) {
        for (j = 0; j < quantity[i]; j++) {
            month = start[i] + j * period[i];
            value += amount[i] / Math.pow (1.0 + rate/1200.0, month);
        }
    }
    return value;
}

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

// PROCESS AN ACTION PERFORMED EVENT

public void
actionPerformed (
    ActionEvent     event)          // Action event
{
    double          principal;      // Principal amount
    int []          start;          // Starting month
    int []          period;         // Period between payments
    int []          quantity;       // Number of payments
    double []       amount;         // Payment amounts
    int             typeCnt;        // Number of payment types
    boolean         decodeFault;    // Decode fault flag
    int             i;              // General purpose index
    double          h, l, m;        // Trial and error variables

    if (event.getActionCommand().equals ("calculate")) {

        // Create the field value arrays

        start = new int [ MAX_PAY_TYPES ];
        period = new int [ MAX_PAY_TYPES ];
        quantity = new int [ MAX_PAY_TYPES ];
        amount = new double [ MAX_PAY_TYPES ];

        // Reset the fault flags and colours

        decodeFault = false;
        principalField.setBackground (Color.WHITE);
        for (i = 0; i < MAX_PAY_TYPES; i++) {
            startField[i].setBackground (Color.WHITE);
            periodField[i].setBackground (Color.WHITE);
            quantityField[i].setBackground (Color.WHITE);
            amountField[i].setBackground (Color.WHITE);
        }

        // Decode the principal amount

        try {
            principal = Double.parseDouble (principalField.getText());
        } catch (NumberFormatException e) {
            principalField.setBackground (Color.RED);
            decodeFault = true;
            principal = 0; // Keep the compiler quiet
        }

        // Process each payment type

        typeCnt = 0;
        for (i = 0; i < MAX_PAY_TYPES; i++) {
            if (
                startField[i].getText().trim().length() != 0 ||
                periodField[i].getText().trim().length() != 0 ||
                quantityField[i].getText().trim().length() != 0 ||
                amountField[i].getText().trim().length() != 0
```

```
    ) {
        try {
            start[typeCnt] = Integer.parseInt
                (startField[i].getText());
        } catch (NumberFormatException e) {
            startField[i].setBackground (Color.RED);
            decodeFault = true;
        }
        try {
            period[typeCnt] = Integer.parseInt
                (periodField[i].getText());
        } catch (NumberFormatException e) {
            periodField[i].setBackground (Color.RED);
            decodeFault = true;
        }
        try {
            quantity[typeCnt] = Integer.parseInt
                (quantityField[i].getText());
        } catch (NumberFormatException e) {
            quantityField[i].setBackground (Color.RED);
            decodeFault = true;
        }
        try {
            amount[typeCnt] = Double.parseDouble
                (amountField[i].getText());
        } catch (NumberFormatException e) {
            amountField[i].setBackground (Color.RED);
            decodeFault = true;
        }
        typeCnt ++ ;
    }
}

// If there was a decode fault, display an error in the
// interest rate field and return.

if (decodeFault) {
    rateField.setText ("ERROR");
    return;
}

// Check that the repayments will result in an interest rate
// in the range 0 to 100.

if (
    presentValue (start, period, quantity, amount, typeCnt, 0.0)
        < principal ||
    presentValue (start, period, quantity, amount, typeCnt, 100.0)
        > principal
) {
    rateField.setText ("****");
    return;
}

// Determine the effective interest rate by trial and error

h = 100.0;
l = 0.0;
m = 50.0; // Keep the compiler quiet
for (i = 0; i < TRIAL_CNT; i++) {
    m = (h + l) / 2;
    if (
        presentValue (start, period, quantity, amount, typeCnt, m)
            < principal
    )
        h = m;
```

```
            else
                l = m;
        }

        // Display the effective interest rate

        rateField.setText (new DecimalFormat("##0.00").format(m));
    }
}

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

// INITIATION

public void
init ()
{
    int             i;              // General purpose index
    FontMetrics     fontMetrics;    // Font metrics instance
    int             lineHeight;     // Line height
    int             colWidth;       // Column width
    int             y;              // Vertical position
    Label           label;          // A label

    // Load the dialog box units

    fontMetrics = getFontMetrics(getFont());
    lineHeight = fontMetrics.getHeight();
    colWidth = fontMetrics.charsWidth("0123456789".toCharArray(),0,10)/10;

    // Disable the default layout manager

    setLayout (null);

    // Set the background colour

    setBackground (Color.LIGHT_GRAY);

    // Create the amount borrowed field

    y = lineHeight * 2;
    label = new Label ("Amount borrowed");
    label.setBounds (colWidth*2, y, colWidth*30, lineHeight);
    add (label);
    principalField = new TextField ();
    principalField.setBounds (colWidth*50, y, colWidth*12, lineHeight*3/2);
    add (principalField);
    y += lineHeight * 3;

    // Create the payment fields

    label = new Label ("Month of");
    label.setBounds (colWidth*(20-2), y-lineHeight, colWidth*8, lineHeight);
    label.setAlignment (Label.CENTER);
    add (label);
    label = new Label ("first pay");
    label.setBounds (colWidth*(20-2), y, colWidth*8, lineHeight);
    label.setAlignment (Label.CENTER);
    add (label);
    label = new Label ("Period");
    label.setBounds (colWidth*(30-2), y, colWidth*8, lineHeight);
    label.setAlignment (Label.CENTER);
    add (label);
    label = new Label ("Number of");
    label.setBounds (colWidth*(40-2), y-lineHeight, colWidth*8, lineHeight);
    label.setAlignment (Label.CENTER);
```

```
    add (label);
    label = new Label ("payments");
    label.setBounds (colWidth*(40-2), y, colWidth*8, lineHeight);
    label.setAlignment (Label.CENTER);
    add (label);
    label = new Label ("Amount");
    label.setBounds (colWidth*50, y, colWidth*12, lineHeight);
    label.setAlignment (Label.RIGHT);
    add (label);
    y += lineHeight * 3/2;
    label = new Label ("Payments");
    label.setBounds (colWidth*2, y, colWidth*16, lineHeight);
    add (label);
    startField = new TextField [ MAX_PAY_TYPES ];
    periodField = new TextField [ MAX_PAY_TYPES ];
    quantityField = new TextField [ MAX_PAY_TYPES ];
    amountField = new TextField [ MAX_PAY_TYPES ];
    for (i = 0; i < MAX_PAY_TYPES; i++) {
            startField[i] = new TextField ();
            startField[i].setBounds (colWidth*20, y,
                    colWidth*4, lineHeight*3/2);
            add (startField[i]);
            periodField[i] = new TextField ();
            periodField[i].setBounds (colWidth*30, y,
                    colWidth*4, lineHeight*3/2);
            add (periodField[i]);
            quantityField[i] = new TextField ();
            quantityField[i].setBounds (colWidth*40, y,
                    colWidth*4, lineHeight*3/2);
            add (quantityField[i]);
            amountField[i] = new TextField ();
            amountField[i].setBounds (colWidth*50, y,
                    colWidth*12, lineHeight*3/2);
            add (amountField[i]);
            y += lineHeight * 2;
    }
    y += lineHeight * 1/2;

    // Create the calculate button

    calcButton = new Button ("Calculate");
    calcButton.setBounds (colWidth*50, y,
            colWidth*12, lineHeight*3/2);
    calcButton.setActionCommand ("calculate");
    calcButton.addActionListener (this);
    add (calcButton);
    y += lineHeight * 5/2;

    // Create the effective interest rate field

    label = new Label ("Effective interest rate");
    label.setBounds (colWidth*2, y, colWidth*30, lineHeight);
    add (label);
    rateField = new TextField ();
    rateField.setBounds (colWidth*50, y, colWidth*12, lineHeight*3/2);
    rateField.setEditable (false);
    rateField.setFocusable (false);
    add (rateField);
}

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

// PAINT THE IMAGE

public void
paint (
```

```
        Graphics        g)              // Reference to graphics instance
    {
        super.paint (g);
    }

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

    // START EXECUTION OF THE APPLET

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

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

    // STOP EXECUTION OF THE APPLET

    public void
    stop ()
    {
        super.stop ();
    }

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

    // GET APPLET INFORMATION

    public String
    getAppletInfo ()
    {
        return "LoanVal, Copyright, 2006, E-Genting Sdn. Bhd.";
    }
}
```