Exercise: 'Swing' and JDBC

The Exercise

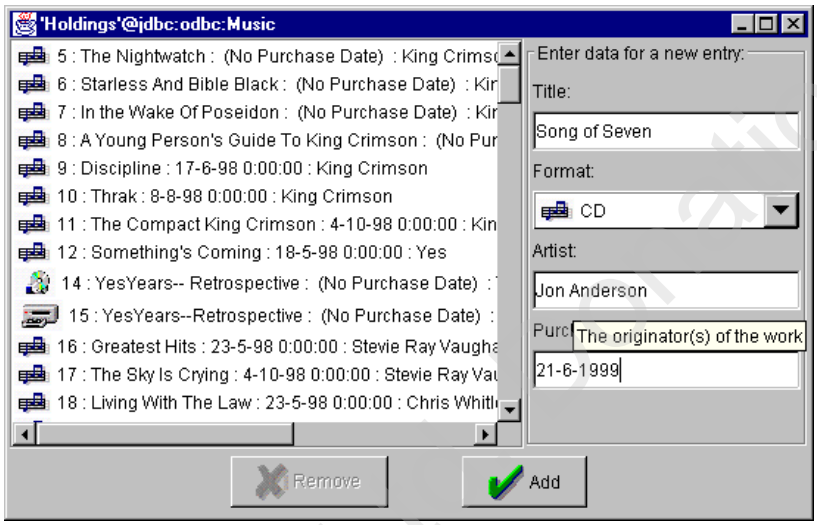
This two-part exercise involves building a library system called 'MusicMan' for maintaining a small CD/videotape /Laser Disk collection.

In the first section, you will gain familiarity with a portion of Java's 'Swing' "standard extension." The second section will make use of Java's JDBC facility to manipulate a simple database.

Note: this is quite a long exercise…although it is quite capable, Swing is not necessarily a trivial API to use!

Part 1: Building the Application

This is a screen shot of the application that you will build:



The code given below should start you off. You will need to:

- add the necessary import statements
- define the LabelledCombo and LabelledTextField classes
- define the ComboBoxRenderer and MusicEntryListCellRenderer classes according to the appropriate Swing documentation
- add code to enable and disable the JButtons appropriately
- add code associated with the Add button's ActionListener
- perform some other miscellaneous coding

Your instructor should be able to supply you with the various images and database files needed for this exercise.

```
// import statements...

public class MusicMan extends JFrame
  {
  static final String LASER = "Laser Disk",
                      CD = "CD",
                      VIDEO = "Video Tape";
  final static ImageIcon CDIcon = new ImageIcon("CD.gif"),
    LDIcon = new ImageIcon("LD.gif"),
    QueryIcon = new ImageIcon("Query.gif"),
    TapeIcon = new ImageIcon("Tape.gif");
  final static int WIDTH = 20;
  DefaultListModel model;
  JList holdingsList;
  LabelledCombo combo;
  LabelledTextField titleText,
            purchaseText,
            artistText;
  JButton addButton = new JButton ("Add"),
          rmButton = new JButton ("Remove");
  long entryNumber = 0;
```

```
private ImageIcon selectIcon (String s)
  {
  ImageIcon i = QueryIcon;
  if (s != null)
    {
    if (LASER.equals (s))
      i = LDIcon;
    else if (CD.equals (s))
      i = CDIcon;
    else if (VIDEO.equals (s))
      i = TapeIcon;
    }
  return (i);
  }

private class ComboBoxRenderer extends DefaultListCellRenderer
  { /* left as exercise */ }

private class MusicEntryListCellRenderer extends DefaultListCellRenderer
  { /* left as exercise */ }

private class MusicEntry
  {
  private long ID;
  private String title,
                 format,
                 purchase,
                 artist;

  public MusicEntry (long ID, String title, String format,
                     String purchase, String artist)
    {
    this.ID = ID;
    this.title = title;
    this.format = format;
    this.purchase = purchase;
    this.artist = artist;
    }

  public ImageIcon getIcon ()
    { return (selectIcon (format)); }

  public String getName ()
    {
    final String SP = " : ";

    return (ID + SP + title + SP + purchase + SP + artist);
    }
  };

private class LabelledCombo extends JPanel
  {
  // a combination JComboBox and JLabel. Left as exercise
  // constructor signature:
  // public LabelledCombo (String [] options, String toolTipText,
  //                       String label, ListCellRenderer lcr, boolean editable)
  }

private class LabelledTextField extends JPanel
  {
  // combination JTextField and JLabel. Left as exercise
  // constructor signature:
  // public LabelledTextField (int width, String label,
  //                           String toolTipText, DocumentListener l)
  }

public MusicMan (String tableName)
  {
  setTitle ("'" + tableName + "'");

  model = new DefaultListModel ();
  holdingsList = new JList (model);

  Container contentPane = getContentPane ();
```

```
Box centerBox = Box.createHorizontalBox ();

// account for bug: see JDC bug #4223100
if (System.getProperty ("java.version").equals ("1.2.1"))
  holdingsList.setSelectionModel
    (
    new DefaultListSelectionModel ()
      {
      public void removeIndexInterval (int index0, int index1)
        {
        super.removeIndexInterval (index0, index1);
        fireValueChanged (index0, index1, getValueIsAdjusting ());
        }
      }
    );
holdingsList.getSelectionModel ().addListSelectionListener
  (
  new ListSelectionListener ()
    {
    public void valueChanged (ListSelectionEvent e)
      { rmButton.setEnabled ( /* true iff there is a selection */ ); }
    }
  );
holdingsList.setSelectionMode (ListSelectionModel.SINGLE_SELECTION);
holdingsList.setCellRenderer (new MusicEntryListCellRenderer ());

DocumentListener myListener = new DocumentListener ()
  {
  public void changedUpdate (DocumentEvent e)
    { addButton.setEnabled ( /* true iff there is data in all entry fields */ ); }
  public void insertUpdate (DocumentEvent e) { changedUpdate (e); };
  public void removeUpdate (DocumentEvent e) { changedUpdate (e); };
  };

JPanel righthand = new JPanel ();
righthand.setLayout (new BorderLayout ());
righthand.setBorder (new TitledBorder ("Enter data for a new entry:"));

JPanel entryArea = new JPanel ();
entryArea.setLayout (new GridLayout (4, 1));
entryArea.add (titleText =
  new LabelledTextField (WIDTH, "Title:", "The title of the work", myListener));
entryArea.add (combo =
  new LabelledCombo (new String [] { CD, LASER, VIDEO },
                     "Make your selection", "Format:",
                     new ComboBoxRenderer (), false));
entryArea.add (artistText = /* left as exercise */ );
entryArea.add (purchaseText = /* left as exercise */ );
righthand.add (BorderLayout.NORTH, entryArea);

centerBox.add (new JScrollPane (holdingsList));
centerBox.add (righthand);

JPanel buttons = new JPanel ();
((FlowLayout) buttons.getLayout()).setHgap (50);

rmButton.setEnabled (false);
rmButton.setIcon (new ImageIcon ("no.gif"));
buttons.add (rmButton);
rmButton.addActionListener
  (
  new ActionListener ()
    {
    public void actionPerformed (ActionEvent e)
      {
      Object o = holdingsList.getSelectedValue ();
      if (o != null)
        model.removeElement (o);
      }
    }
  );

addButton.setEnabled (false);
addButton.setIcon (new ImageIcon ("yes.gif"));
buttons.add (addButton);
```

```
    addButton.addActionListener
      (
      new ActionListener ()
        {
        public void actionPerformed (ActionEvent e)
          {
          // gather the data from the various text and combo boxes
          // make a new MusicEntry instance 'me' and add it to the list model

          holdingsList.setSelectedValue (me, true);
          holdingsList.ensureIndexIsVisible (holdingsList.getSelectedIndex ());
          holdingsList.clearSelection ();
          }
        }
      );

    contentPane.add (BorderLayout.CENTER, centerBox);
    contentPane.add (BorderLayout.SOUTH, buttons);
    }

  public static void main (String [] args)
    {
    try
      {
      UIManager.setLookAndFeel (UIManager.getSystemLookAndFeelClassName ());
      MusicMan m = new MusicMan (args [0]);
      m.addWindowListener
        (
        new WindowAdapter ()
          {
          public void windowClosing (WindowEvent e)
            { System.exit (0); }
          }
        );
      m.setSize (550, 350);
      m.setVisible (true);
      }
    catch (Exception e)
      {
      e.printStackTrace (System.err);
      }
    }
  }
```

Part 2: Modifying the Application to use a Database as a Model

The previous version of this application relied on the properties of the simple DefaultListModel to perform its data storage activities. This is acceptable for transient data but is insufficient for a 'real' application, which would need to save and restore its data across executions.

To augment this application, you will need to create an extension of DefaultListModel called MusicListModel. It is this extension that will handle the "nuts and bolts" of dealing with JDBC.

```
private class MusicListModel extends DefaultListModel
  {
  Connection connection;
  String tableName;

  public MusicListModel (String tableName) throws SQLException
    {
    this.tableName = tableName;

    final Properties credentials = new Properties ();
    credentials.put ("user", USER);
    credentials.put ("password", PASSWD);

    // get a connection from the DriverManager
    // select * from tableName
    // instantiate a MusicEntry object 'm' from each row returned
    super.addElement (m);
    }

  private long findMax (Statement query) throws SQLException
    { /* select MAX (ID) from tableName; */ }
```

```
  public void addElement (Object obj)
    {
    // insert the MusicEntry object 'm' into the database making use of
    // m.GetSQLValueString (). Be sure to m.setID (findMax ()) first…
    super.addElement (m);
    }

  public boolean removeElement (Object obj)
    {
    // delete this object from the database, setting res in the process
    return (res == 1 && super.removeElement (m));
    }
  }
```

You will need to supply this extension as model to the application's main JList, rather than allowing it to use the DefaultListModel that it currently uses.

You may also need to augment the basic MusicEntry class with get/set methods for its ID field and you will need to add a method called getSQLValueString (this method should return a string representation of the instance data formatted appropriately for use in an SQL INSERT statement). You will also need to add the following code to the MusicMan class:

```
static final String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver",
                    USER = "Music",
                    PASSWD = "MusicMan",
                    URL = "jdbc:odbc:Music";
static
  {
  try
    {
    Class.forName (DRIVER);
    }
  catch (Exception e)
    {
    e.printStackTrace (System.err);
    System.exit (-1);
    }
  }
```

The constants assume that a database called 'Music' exists on your computer and that this database is accessible via ODBC using Sun's ODBC-JDBC driver. The username/password for the database is 'Music'/'MusicMan'.

Part 3: a Refinement

There may be a substantial delay when the program starts up and loads its data from the JDBC database. You may want to put up a progress dialog to provide feedback to the user…you can do this using the following code:

```
// within the MusicMan class
private class ProgressWin extends JFrame
  {
  private JProgressBar progressBar;
  public ProgressWin (String title)
    {
    setTitle (title);
    progressBar = new JProgressBar ();
    Container c = getContentPane ();
    c.setLayout (new FlowLayout ());
    c.add (new JLabel (CDIcon));
    c.add (progressBar);
    pack ();
    Dimension frameSize = getSize (),
              screenSize = getToolkit ().getScreenSize ();
    setLocation ((screenSize.width / 2) - (frameSize.width / 2),
                 (screenSize.height / 3) - (frameSize.height / 2));
 }
  public JProgressBar getProgressBar ()
    {
    return (progressBar);
    }
  }
```

```
// in MusicMan's constructor
ProgressWin progWin = new ProgressWin ("Loading...");
progWin.setVisible (true);
model = new MusicListModel (tableName, progWin.getProgressBar ());
progWin.setVisible (false);
progWin.dispose ();
```

To complete the addition of this feature you will need to modify the MusicListModel constructor and insert appropriate calls to initialise the maximum and minimum limits for the JProgressBar. You should update the JProgressBar as each MusicEntry instance is retrieved from the database.