Line 52 invokes method **addElement** of class **DefaultListModel** to add the new philosopher to the list. The **DefaultListModel** will notify the **JList** that the model changed, and the **JList** will update the display to include the new list item.

Lines 58–71 create a **JButton** for deleting a philosopher from the **DefaultListModel**. Lines 67–68 in method **actionPerformed** invoke method **getSelectedValue** of class **JList** to get the currently selected philosopher and invoke method **removeElement** of class **DefaultListModel** to remove the philosopher. The **DefaultListModel** will notify the **JList** that the model changed, and the **JList** will update the display to remove the deleted philosopher. Lines 74–84 lay out the GUI components and set **JFrame** properties for the application window.

3.5 JTable

JTable is another **Swing** component that implements the delegate-model architecture. **JTable**s are delegates for tabular data stored in **TableModel** implementations. Interface **TableModel** declares methods for retrieving and modifying data (e.g., the value in a certain table cell) and for retrieving and modifying metadata (e.g., the number of columns and rows). The **JTable** delegate invokes **TableModel** methods to build its view of the **TableModel** and to modify the **TableModel** based on user input.

Figure 3.13 describes the methods defined in interface **TableModel**. Custom implementations of interface **TableModel** can use arbitrary internal representations of the tabular data. For example, the **DefaultTableModel** implementation uses **Vector**s to store the rows and columns of data. In Chapter 8, JDBC, we implement interface **TableModel** to create a **TableModel** that represents data stored in a JDBC **ResultSet**. Figure 3.14 illustrates the delegate-model relationship between **JTable** and **TableModel**.

Method	Description
void addTableModelListener(TableModelListener listener)	Add a **TableModelListener** to the **TableModel**. The **TableModel** will notify the **TableModelListener** of changes in the **TableModel**.
void removeTableModelListener(TableModelListener listener)	Remove a previously added **TableModelListener** from the **TableModel**.
Class getColumnClass(int columnIndex)	Get the **Class** object for values in the column with specified **columnIndex**.
int getColumnCount()	Get the number of columns in the **TableModel**.
String getColumnName(int columnIndex)	Get the name of the column with the given **columnIndex**.
int getRowCount()	Get the number of rows in the **TableModel**.

Fig. 3.13 **TableModel** interface methods and descriptions (part 1 of 2).

Method	Description
`Object getValueAt( int rowIndex, int columnIndex )`	Get an **`Object`** reference to the value stored in the **`TableModel`** at the given row and column indices.
`void setValueAt( Object value, int rowIndex, int columnIndex )`	Set the value stored in the **`TableModel`** at the given row and column indices.
`boolean isCellEditable( int rowIndex, int columnIndex )`	Return **`true`** if the cell at the given row and column indices is editable.

Fig. 3.13 **`TableModel`** interface methods and descriptions (part 2 of 2).

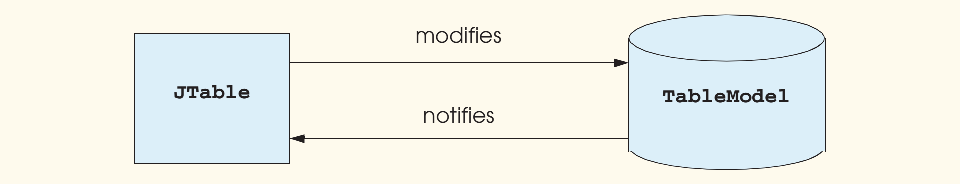


Fig. 3.14 **`JTable`** and **`TableModel`** delegate-model architecture.

`PhilosophersJTable` (Fig. 3.15) displays philosopher information in a **`JTable`** using a **`DefaultTableModel`**. Class **`DefaultTableModel`** implements interface **`TableModel`** and uses **`Vector`**s to represent the rows and columns of data. Line 24 creates the **`philosophers DefaultTableModel`**. Lines 27–29 add columns to the **`DefaultTableModel`** for the philosophers' first names, last names and years in which they lived. Lines 32–53 create rows for seven philosophers. Each row is a **`String`** array whose elements are the philosopher's first name, last name and the year in which the philosopher lived, respectively. Method **`addRow`** of class **`DefaultTableModel`** adds each philosopher to the **`DefaultTableModel`**. Line 56 creates the **`JTable`** that will act as a delegate for the **`philosophers DefaultTableModel`**.

Lines 59–72 create a **`JButton`** and **`ActionListener`** for adding a new philosopher to the **`DefaultTableModel`**. Line 66 in method **`actionPerformed`** creates a **`String`** array of three empty elements. Line 69 adds the empty **`String`** array to the **`DefaultTableModel`**. This causes the **`JTable`** to display a blank row at the bottom of the **`JTable`**. The user can then type the philosopher's information directly into the **`JTable`** cells. This demonstrates the **`JTable`** delegate acting as a controller, because it modifies the **`DefaultTableModel`** based on user input.

```
1   // PhilosophersJTable.java
2   // MVC architecture using JTable with a DefaultTableModel
3   package com.deitel.advjhtp1.mvc.table;
```

Fig. 3.15 **`PhilosophersJTable`** application demonstrating **`JTable`** and **`DefaultTableModel`** (part 1 of 4).

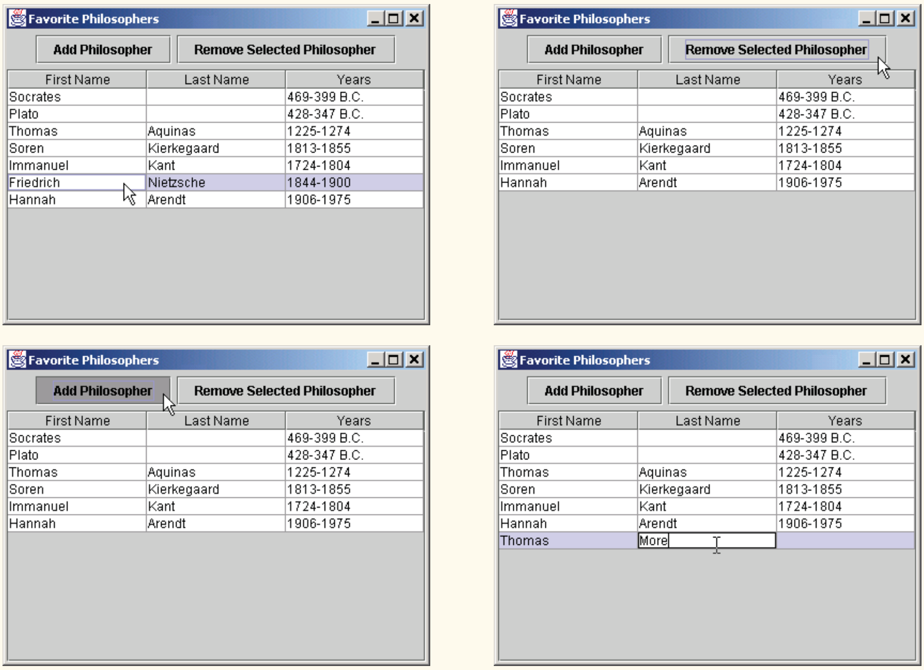
```java

// Java core packages
import java.awt.*;
import java.awt.event.*;

// Java extension packages
import javax.swing.*;
import javax.swing.table.*;

public class PhilosophersJTable extends JFrame {

   private DefaultTableModel philosophers;
   private JTable table;

   // PhilosophersJTable constructor
   public PhilosophersJTable()
   {
      super( "Favorite Philosophers" );

      // create a DefaultTableModel to store philosophers
      philosophers = new DefaultTableModel();

      // add Columns to DefaultTableModel
      philosophers.addColumn( "First Name" );
      philosophers.addColumn( "Last Name" );
      philosophers.addColumn( "Years" );

      // add philosopher names and dates to DefaultTableModel
      String[] socrates = { "Socrates", "", "469-399 B.C." };
      philosophers.addRow( socrates );

      String[] plato = { "Plato", "", "428-347 B.C." };
      philosophers.addRow( plato );

      String[] aquinas = { "Thomas", "Aquinas", "1225-1274" };
      philosophers.addRow( aquinas );

      String[] kierkegaard = { "Soren", "Kierkegaard",
         "1813-1855" };
      philosophers.addRow( kierkegaard );

      String[] kant = { "Immanuel", "Kant", "1724-1804" };
      philosophers.addRow( kant );

      String[] nietzsche = { "Friedrich", "Nietzsche",
         "1844-1900" };
      philosophers.addRow( nietzsche );

      String[] arendt = { "Hannah", "Arendt", "1906-1975" };
      philosophers.addRow( arendt );

```

Fig. 3.15 **PhilosophersJTable** application demonstrating **JTable** and **DefaultTableModel** (part 2 of 4).

```
      // create a JTable for philosophers DefaultTableModel
      table = new JTable( philosophers );

      // create JButton for adding philosophers
      JButton addButton = new JButton( "Add Philosopher" );
      addButton.addActionListener(
         new ActionListener() {

            public void actionPerformed( ActionEvent event )
            {
               // create empty array for new philosopher row
               String[] philosopher = { "", "", "" };

               // add empty philosopher row to model
               philosophers.addRow( philosopher );
            }
         }
      );

      // create JButton for removing selected philosopher
      JButton removeButton =
         new JButton( "Remove Selected Philosopher" );

      removeButton.addActionListener(
         new ActionListener() {

            public void actionPerformed( ActionEvent event )
            {
               // remove selected philosopher from model
               philosophers.removeRow(
                  table.getSelectedRow() );
            }
         }
      );

      // lay out GUI components
      JPanel inputPanel = new JPanel();
      inputPanel.add( addButton );
      inputPanel.add( removeButton );

      Container container = getContentPane();
      container.add( new JScrollPane( table ),
         BorderLayout.CENTER );
      container.add( inputPanel, BorderLayout.NORTH );

      setDefaultCloseOperation( EXIT_ON_CLOSE );
      setSize( 400, 300 );
      setVisible( true );

   } // end PhilosophersJTable constructor

```

Fig. 3.15 **PhilosophersJTable** application demonstrating **JTable** and **DefaultTableModel** (part 3 of 4).

```
106     // execute application
107     public static void main( String args[] )
108     {
109        new PhilosophersJTable();
110     }
111  }
```



First Name	Last Name	Years
Socrates		469-399 B.C.
Plato		428-347 B.C.
Thomas	Aquinas	1225-1274
Soren	Kierkegaard	1813-1855
Immanuel	Kant	1724-1804
Friedrich	Nietzsche	1844-1900
Hannah	Arendt	1906-1975

First Name	Last Name	Years
Socrates		469-399 B.C.
Plato		428-347 B.C.
Thomas	Aquinas	1225-1274
Soren	Kierkegaard	1813-1855
Immanuel	Kant	1724-1804
Hannah	Arendt	1906-1975

First Name	Last Name	Years
Socrates		469-399 B.C.
Plato		428-347 B.C.
Thomas	Aquinas	1225-1274
Soren	Kierkegaard	1813-1855
Immanuel	Kant	1724-1804
Hannah	Arendt	1906-1975

First Name	Last Name	Years
Socrates		469-399 B.C.
Plato		428-347 B.C.
Thomas	Aquinas	1225-1274
Soren	Kierkegaard	1813-1855
Immanuel	Kant	1724-1804
Hannah	Arendt	1906-1975
Thomas	More	

Fig. 3.15 **`PhilosophersJTable`** application demonstrating **`JTable`** and **`DefaultTableModel`** (part 4 of 4).

Lines 75–88 create a **`JButton`** and **`ActionListener`** for removing a philosopher from the **`DefaultTableModel`**. Lines 84–85 in method **`actionPerformed`** retrieve the currently selected row in the **`JTable`** delegate and invoke method **`removeRow`** of class **`DefaultTableModel`** to remove the selected row. The **`DefaultTableModel`** notifies the **`JTable`** that the **`DefaultTableModel`** has changed, and the **`JTable`** removes the appropriate row from the display. Lines 96–97 add the **`JTable`** to a **`JScrollPane`**. **`JTables`** will not display their column headings unless placed within a **`JScrollPane`**.

3.6 JTree

`JTree` is one of the more complex Swing components that implements the delegate-model architecture. **`TreeModel`**s represent hierarchical data, such as family trees, certain types of file systems, company management structures and document outlines. **`JTree`**s act as delegates (i.e., combined view and controller) for **`TreeModel`**s.