

JTable



Topics

- Basic Tables
- Overview of Related Classes
- JTable Class in Detail
- Table Data Models
 - interface, abstract, default, listener, event
- Table Columns
 - class, model, listener, event
- Table Headers
- Selection of Rows, Columns, and Cells
- Cell Renderers and Editors
- Row Sorting
- Database Access

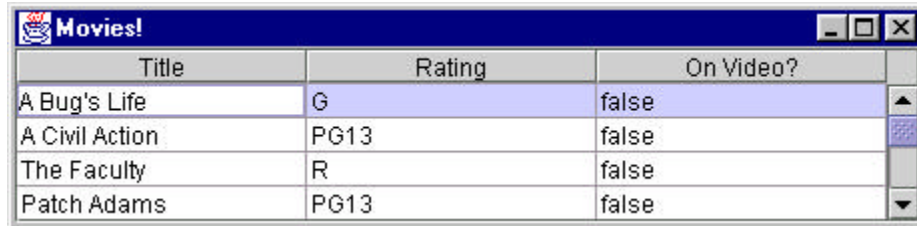


JTable Overview

- Displays data in a table
- Features
 - user can
 - select rows, columns, or cells
 - reorder columns by dragging headers
 - resize columns by dragging border between them
 - edit cell values
 - sort rows based on selected column
 - requires extra coding
 - code can
 - change cell values
 - add/remove/move columns
 - customize cell rendering
 - customize cell editing
- Supported by
 - JTable class in javax.swing package
 - many classes and interfaces in javax.swing.table and javax.swing.event



Basic Example



Title	Rating	On Video?
A Bug's Life	G	false
A Civil Action	PG13	false
The Faculty	R	false
Patch Adams	PG13	false

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

By default, this table

- is editable
- only allows row selections
- displays all cells as JLabels

```
public class BasicMovieApp extends JFrame {

    public static void main(String[] args) {
        new BasicMovieApp().setVisible(true);
    }

    private BasicMovieApp() {
        super("Movies!");

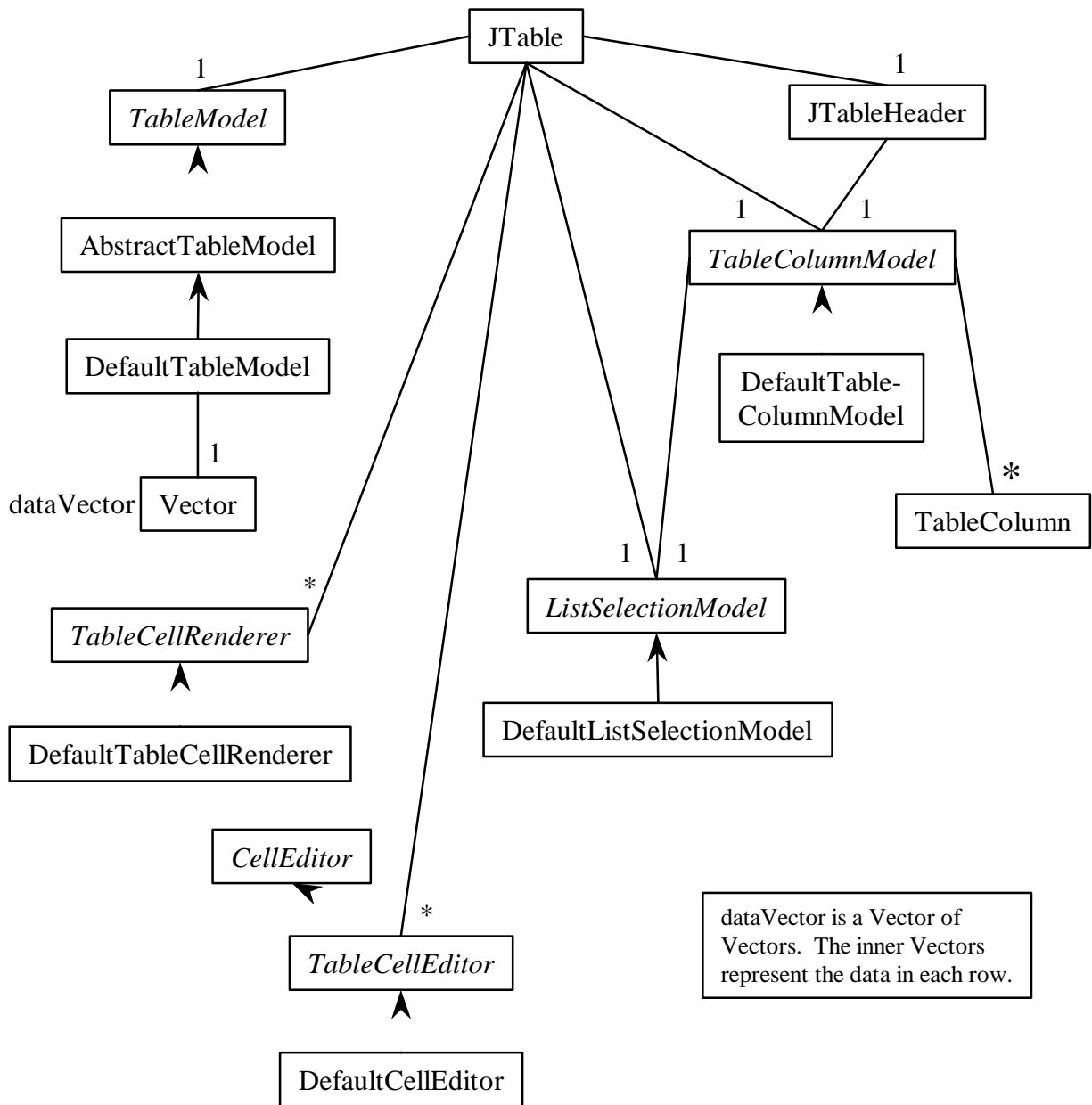
        String[] columnNames =
            {"Title", "Rating", "On Video?"};
        Object[][] data = {
            {"A Bug's Life", "G", new Boolean(false)},
            {"A Civil Action", "PG13", new Boolean(false)},
            // ... more movies omitted ...
        };

        JTable table = new JTable(data, columnNames);
        getContentPane().add(new JScrollPane(table),
                               BorderLayout.CENTER);

        pack();
    }
}
```



Overview of Related Classes



dataVector is a Vector of Vectors. The inner Vectors represent the data in each row.

italic indicates interface



JTable Class

- Delegates many details to these
 - TableModel
 - manages cell data
 - TableColumnModel
 - manages a collection of TableColumnns
 - manages column selections
 - JTableHeader
 - component that displays the table header
 - controls color and font
 - controls whether column resizing and reordering are allowed
 - ListSelectionModel
 - manages row selections
 - TableCellRenderer
 - draws cell values
 - TableCellEditor
 - allows editing of cell values



JTable Class (Cont'd)

- Method highlights

- getting fundamental helper objects

`TableColumnModel getColumnModel()` ← knows column selections

`TableModel getModel()`

`ListSelectionModel getSelectionModel()` ← knows row selections

`TableHeader getTableHeader()`

- working with selections

`void setCellSelectionEnabled(boolean flag)`

`void setColumnSelectionAllowed(boolean flag)` - defaults to false

`void setRowSelectionAllowed(boolean flag)` - defaults to true

`void setColumnSelectionInterval(int index0, int index1)`

`void setRowSelectionInterval(int index0, int index1)`

`void addColumnSelectionInterval(int index0, int index1)`

`void addRowSelectionInterval(int index0, int index1)`

`void clearSelection()`

columns are not
selected by clicking
on a column heading!



JTable Class (Cont'd)

- More method highlights

- controlling visual attributes

```
void setPreferredSize(Dimension size)
void setRowHeight(int height)
void setSelectionBackground(Color selectionBackground)
void setSelectionForeground(Color selectionForeground)
void setShowGrid(boolean b) - both horizontal & vertical
void setShowHorizontalLines(boolean b)
void setShowVerticalLines(boolean b)
void setGridColor(Color newColor)
void setDefaultRenderer(Class columnClass,
                        TableCellRenderer editor)
void setDefaultEditor(Class columnClass,
                      TableCellEditor editor)
```

- other methods

```
void addColumn(TableColumn aColumn)
void editCellAt(int row, int column)
void setAutoResizeMode(int mode) - explained later
void setSelectionMode(int selectionMode)
    SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION,
    MULTIPLE_INTERVAL_SELECTION
void setValueAt(Object aValue, int row, int column)
```



TableModel Interface

(doing all the work and getting none of the freebies)

- Manages data displayed by a JTable

- could also be used as the model for charts

- Methods

```
int getRowCount()  
int getColumnCount()  
Class getColumnClass(int columnIndex)  
    • used to determine renderer and editor  
String getColumnName(int columnIndex)  
    • used for column header  
boolean isCellEditable(int rowIndex, int columnIndex)  
Object getValueAt(int rowIndex, int columnIndex)
```

TableModel.setHeaderValue(String)
takes precedence over this.

```
void setValueAt(Object aValue,  
                int rowIndex,  
                int columnIndex)  
    • called by JTable when cell edits are made so they can be saved  
    • JTable automatically adds itself as a listener of its model  
    • could be called by other objects so must let the JTable (and other listeners)  
      know the model has changed by calling  
void tableChanged(TableModelEvent e)  
    – AbstractTableModel (on next page) provides  
      convenience methods for doing this
```

covered later

```
void addTableModelListener  
    (TableModelListener l)  
    • to be notified of changes to the table data and structure  
void removeTableModelListener(TableModelListener l)
```



AbstractTableModel Class

(doing some of the work and getting some of the freebies)

- Implements TableModel
- Must define these methods

```
int getRowCount();
int getColumnCount();
Object getValueAt(int rowIndex, int columnIndex);
```
- Override remaining TableModel methods to change default behavior
 - supplies these defaults
 - the class of all cells is reported as Object
 - class-specific renderers and editors aren't used
 - columns are named like in spreadsheets
 - cells are not editable
 - to make some or all cells editable, implement isCellEditable() to return true and setValueAt() to save edited values



AbstractTableModel Class (Cont'd)

- Adds convenience methods that create TableModelEvents and pass to listeners

more specific {

```
void fireTableCellUpdated(int row, int column)
void fireTableRowsDeleted(int firstRow, int lastRow)
void fireTableRowsInserted(int firstRow, int lastRow)
void fireTableRowsUpdated(int firstRow, int lastRow)
```

less specific {

```
void fireTableDataChanged()
– can be called even if the number of rows changes
void fireTableStructureChanged()
– call this when the number, names, or types of the columns changes
```

```
void fireTableChanged(TableModelEvent e)
– you describe the change in the TableModelEvent
– using the other "fire" methods is easier
```



DefaultTableModel Class

(doing none of the work and getting all of the freebies)

- Extends AbstractTableModel
- Stores cell data in a Vector of Vectors
 - one Vector per row
- Override methods to change default behavior
 - supplies these defaults
 - the class of all cells is reported as Object
 - class-specific renderers and editors aren't used
 - columns are named like in spreadsheets
 - unless the column names are passed to the constructor, setDataVector(), or setColumnIdentifiers()
 - cells are editable
 - to make some or all cells read-only, override isCellEditable()
- Adds methods beyond those in AbstractTableModel ... highlights are

```
Vector getDataVector()  
void setDataVector(Object[][] newData, Object[] columnIDs)  
void setDataVector(Vector newData, Vector columnIDs)  
void addColumn(Object columnID)  
void addColumn(Object columnID, Object[] columnData)  
void addRow(Object[] rowData)  
void insertRow(int row, Object[] rowData)  
void removeRow(int index)  
void setColumnIdentifiers(Object[] columnIDs)
```

there's also a version
that takes a Vector
instead of Object[]



Movie Class

```
public class Movie {  
  
    public static final int NR = 0; // not rated  
    public static final int G = 1;  
    public static final int PG = 2;  
    public static final int PG13 = 3;  
    public static final int R = 4;  
  
    public static final String[] RATINGS =  
        {"NR", "G", "PG", "PG13", "R"};  
  
    private String title;  
    private int rating;  
    private boolean onVideo;  
  
    public Movie(String title) {  
        this(title, NR);  
    }  
  
    public Movie(String title, int rating) {  
        this.title = title;  
        this.rating = rating;  
    }  
}
```



Movie Class (Cont'd)

```
public int getRating() { return rating; }
public String getRatingString() { return RATINGS[rating]; }
public String getTitle() { return title; }
public boolean isOnVideo() { return onVideo; }

public void setTitle(String title) { this.title = title; }
public void setRating(int rating) { this.rating = rating; }
public void setOnVideo(boolean onVideo)
    { this.onVideo = onVideo; }

public void setRatingString(String ratingString) {
    for (int i = 0; i < RATINGS.length; i++) {
        if (RATINGS[i].equals(ratingString)) {
            rating = i;
            return;
        }
    }
    System.err.println
        ("bad value " + ratingString +
         " passed to Movie.setRatingString()");
}

public String toString() {
    return getTitle() + " - " + getRatingString();
}
}
```



MovieTableModel

```
import java.util.*;
import javax.swing.table.*;

public class MovieTableModel extends AbstractTableModel {

    public static final int TITLE_COLUMN = 0;
    public static final int RATING_COLUMN = 1;
    public static final int ON_VIDEO_COLUMN = 2;

    private Vector movies = new Vector();

    public void addMovie(Movie movie) {
        movies.addElement(movie);
        int index = movies.size() - 1;
        fireTableRowsInserted(index, index);
    }

    public int getRowCount() {
        return movies.size();
    }

    public int getColumnCount() {
        return 3;
    }

    public boolean isCellEditable(int rowIndex,
                                   int columnIndex) {
        return columnIndex != 0; // title is not editable
    }
}
```



MovieTableModel (Cont'd)

```
public Class getColumnClass(int columnIndex) {
    switch (columnIndex) {
        case TITLE_COLUMN: return String.class;
        case RATING_COLUMN: return String.class;
        case ON_VIDEO_COLUMN: return Boolean.class;
        default: return null;
    }
}

public String getColumnName(int columnIndex) {
    switch (columnIndex) {
        case TITLE_COLUMN: return "Title";
        case RATING_COLUMN: return "Rating";
        case ON_VIDEO_COLUMN: return "On Video?";
        default: return null;
    }
}

public Object getValueAt(int rowIndex, int columnIndex) {
    Movie movie = (Movie) movies.elementAt(rowIndex);
    switch (columnIndex) {
        case TITLE_COLUMN:
            return movie.getTitle();
        case RATING_COLUMN:
            return movie.getRatingString();
        case ON_VIDEO_COLUMN:
            return new Boolean(movie.isOnVideo());
        default: return null;
    }
}
```



MovieTableModel (Cont'd)

```
public void setValueAt(Object aValue,
                       int rowIndex,
                       int columnIndex) {
    Movie movie = (Movie) movies.elementAt(rowIndex);

    try {
        switch (columnIndex) {
            case TITLE_COLUMN:
                movie.setTitle((String) aValue);
                break;
            case RATING_COLUMN:
                movie.setRatingString((String) aValue);
                break;
            case ON_VIDEO_COLUMN:
                movie.setOnVideo
                    (((Boolean) aValue).booleanValue());
                break;
        }

        fireTableCellUpdated(rowIndex, columnIndex);
    } catch (ClassCastException cce) {
        cce.printStackTrace();
    }
}
```



Creating a JTable Using MovieTableModel

```
MovieTableModel model = new MovieTableModel();

model.addMovie(new Movie("A Bug's Life", Movie.G));
model.addMovie(new Movie("A Civil Action", Movie.PG13));
model.addMovie(new Movie("The Faculty", Movie.R));
model.addMovie(new Movie("Patch Adams", Movie.PG13));
model.addMovie(new Movie("The Prince of Egypt", Movie.PG));
model.addMovie(new Movie("Shakespeare in Love", Movie.R));
model.addMovie(new Movie("Star Trek: Insurrection", Movie.PG));
model.addMovie(new Movie("Waking Ned Devine", Movie.PG));
model.addMovie(new Movie("You've Got Mail", Movie.PG13));

JTable table = new JTable(model);

myJFrame.getContentPane().add
    (new JScrollPane(table), BorderLayout.CENTER);
```



TableModelListener & TableModelEvent

- The JTable is a TableModelListener
- TableModelListener interface methods

```
void tableChanged(TableModelEvent e)
```

- TableModelEvent class methods

```
int getColumn()
```

- returns the index of the affected column
- can be ALL_COLUMNS

```
int getFirstRow()
```

- returns the index of the first affected row
- can be HEADER_ROW

```
int getLastRow()
```

- returns the index of the last affected row
- can be HEADER_ROW

```
int getType()
```

- returns the type of change
- INSERT, UPDATE, or DELETE



VideoStore

(decides when to order new videos)

```
import javax.swing.event.*;

public class VideoStore implements TableModelListener {
    private String name;

    public VideoStore(String name) { this.name = name; }

    public void tableChanged(TableModelEvent e) {
        if (e.getColumn() ==
            MovieTableModel.ON_VIDEO_COLUMN &&
            e.getType() != TableModelEvent.DELETE) {
            int firstRow = e.getFirstRow();
            int lastRow = e.getLastRow();
            MovieTableModel tableModel =
                (MovieTableModel) e.getSource();
            for (int row = firstRow; row <= lastRow; row++) {
                Boolean onVideo =
                    (Boolean) tableModel.getValueAt
                        (row, MovieTableModel.ON_VIDEO_COLUMN);
                if (onVideo.booleanValue()) {
                    String title =
                        (String) tableModel.getValueAt
                            (row, MovieTableModel.TITLE_COLUMN);
                    System.out.println
                        (name + " should order " + title);
                }
            }
        }
    }
}

VideoStore store = new VideoStore("BlockBreaker");
model.addTableModelListener(store);
```

type is
INSERT or
UPDATE



TableColumnModel Interface

- Implemented by DefaultTableColumnModel
- Holds a collection of TableColumn
- Supports adding, removing, and moving columns
- Supports column selections
 - controls whether they are allowed
 - knows which are currently selected
 - uses a ListSelectionModel which supports ListSelectionListeners
- Supports TableColumnModelListeners

To get the one associated with a JTable, call getColumnModel().

- Method highlights

```
int getColumnCount()  
TableColumn getColumn(int columnIndex)  
Enumeration getColumns()  
int[] getSelectedColumns()  
void addColumn(TableColumn column)  
void moveColumn(int index, int newIndex)  
void removeColumn(TableColumn column)  
void addColumnModelListener(TableColumnModelListener x)
```



TableColumnModelListener & TableColumnModelEvent

- TableColumnModelListener interface methods

```
void columnAdded(TableColumnModelEvent e)
void columnRemoved(TableColumnModelEvent e)
void columnMoved(TableColumnModelEvent e)
void columnMarginChanged(ChangeEvent e)
void columnSelectionChanged(ListSelectionEvent e)
```

- TableColumnModelEvent class methods

```
void getFromIndex()
    • returns index of column that was moved or removed
void getToIndex()
    • returns new index of column after being moved or added
```



TableColumn Class

- Encapsulates the following
 - header value (Object)
 - min, max, current, and preferred width (int)
 - defaults are 15, Integer.MAX_VALUE, 75, and 75
 - header renderer (TableCellRenderer)
 - renderer to be used for all cells in the column (TableCellRenderer)
 - editor to be used for all cells in the column (TableCellEditor)
- Fires PropertyChangeEvents for
 - column width, header renderer, cell renderer, cell editor
- Method highlights

```
void setHeaderValue(Object aValue)
void setPreferredWidth(int newWidth) - see next page
void setCellEditor(TableCellEditor anEditor)
void setCellRenderer(TableCellRenderer aRenderer)
```



Setting Column Widths

- Two ways to get a reference to a TableColumn

```
TableColumn tc =  
    table.getColumn(heading-string);
```

OR

```
TableColumnModel tcm = table.getColumnModel();  
TableColumn tc = tcm.getColumn(column-index);
```

- To set the width of a column

```
tc.setPreferredWidth(pixel-width);
```



Controlling Column Resizing

- Users can drag lines between column headings to resize columns

- enabled by default

more on JTableHeader on next page

- turn off with

```
table.getTableHeader().setResizingAllowed(false);
```

- Set resizing mode with

```
table.setAutoResizeMode(int mode);
```

- Valid modes are

- AUTO_RESIZE_OFF

- only the column being resized changes
 - all other columns maintain their width
 - table width changes unlike all other modes

throws exception if last column is resized!

- AUTO_RESIZE_NEXT_COLUMN

- resizes next column to keep constant table width

- AUTO_RESIZE_LAST_COLUMN

- resizes last column to keep constant table width

- AUTO_RESIZE_SUBSEQUENT_COLUMNS

- proportionally resizes all subsequent columns to keep constant table width

- AUTO_RESIZE_ALL_COLUMNS

- proportionally resizes all columns to keep constant table width



JTableHeader Class

- Component that displays the table header
 - can set color and font
- Controls whether column resizing and reordering are allowed
 - both are allowed by default
- Method highlights

```
void setForeground(Color c)
void setBackground(Color c)
void setFont(Font f)
void setReorderingAllowed(boolean b)
void setResizingAllowed(boolean b)
```



Setting Table Size

- Assuming the JTable will be placed in a JScrollPane, set the preferred viewport size

```
table.setPreferredScrollableViewportSize  
    (new Dimension(width, height));
```

- To calculate full table width

```
int width = 0;  
TableModel tcm = table.getColumnModel();  
Enumeration e = tcm.getColumns();  
while (e.hasMoreElements()) {  
    TableColumn tc = (TableColumn) e.nextElement();  
    width += tc.getPreferredWidth();  
}
```

- To calculate full table height

```
int height = table.getRowCount() *  
    (table.getRowHeight() + 1);
```

for grid lines



ListSelectionModel Interface

- Implemented by DefaultListSelectionModel
- One for rows and one for columns
- To be notified of table row selections

```
table.getSelectionModel().  
    addListSelectionListener(l);
```

← a ListSelectionListener

- To be notified of table column selections

```
table.getColumnModel().getSelectionModel().  
    addListSelectionListener(l);
```

- ListSelectionListener interface method

```
void valueChanged(ListSelectionEvent e)
```

- ListSelectionEvent class

- `getFirstIndex()`
 - returns index of first row/column selected
- `getLastIndex()`
 - returns index of last row/column selected
- `getValueIsAdjusting()`
 - indicates whether this event is part of a series of ListSelectEvents and it is not the last one



TableCellRenderer Interface

- A “renderer” creates a Component that displays the value of a table cell
 - default renderer uses JLabel to display the `toString()` value of each table cell
- Two ways to specify
 - by Class of the object to be rendered

```
table.setDefaultRenderer  
    (Class columnClass, TableCellRenderer renderer);
```
 - by column (takes precedence)

```
tableColumn.setCellRenderer(TableCellRenderer);
```
 - can also set renderer for specific column headers

```
tableColumn.setHeaderRenderer(TableCellRenderer);
```
- TableCellRenderer interface
 - only one method

```
Component getTableCellRendererComponent  
    (JTable table,  
     Object value,  
     boolean isSelected,  
     boolean hasFocus,  
     int row,  
     int column)
```
 - can return different rendering components based on
 - the value or its class
 - whether the cell is selected or has focus
 - the row and column that contains the value

two methods of getting
TableColumn objects
were covered earlier

when a row is selected, all cells
in the row are selected, but
only the one clicked has focus



MovieRatingTableCellRenderer

```
import java.awt.*;
import javax.swing.*;
import javax.swing.table.TableCellRenderer;

public class MovieRatingTableCellRenderer extends JLabel
implements TableCellRenderer {
    private static final Font NORMAL_FONT =
        new Font("Serif", Font.ITALIC, 18);
    private static final Font SELECTED_FONT =
        new Font("Serif", Font.ITALIC + Font.BOLD, 18);

    public Component getTableCellRendererComponent
        (JTable table, Object value,
         boolean isSelected, boolean hasFocus,
         int row, int column) {
        if (value == null) return null;

        String rating = (String) value;
        Color color =
            rating.equals(Movie.RATINGS[Movie.NR]) ? Color.red :
            rating.equals(Movie.RATINGS[Movie.G]) ? Color.orange :
            rating.equals(Movie.RATINGS[Movie.PG]) ? Color.green :
            rating.equals(Movie.RATINGS[Movie.PG13]) ? Color.blue :
            rating.equals(Movie.RATINGS[Movie.R]) ? Color.magenta :
            Color.black;

        setText(rating);
        setFont(isSelected ? SELECTED_FONT : NORMAL_FONT);
        setForeground(color);
        return this;
    }
}
```



Using MovieRatingTableCellRenderer

```
TableColumnModel tcm = table.getColumnModel();  
TableColumn tc =  
    tcm.getColumn(MovieTableModel.RATING_COLUMN);  
tc.setCellRenderer  
    (new MovieRatingTableCellRenderer());
```



TableCellEditor Interface

- Cells can be editable
 - determined by calling `isCellEditable()` method of the Table's `TableModel`
 - if the class of a cell value is `Boolean`, `JCheckbox` is be used
 - otherwise `JTextField` is used
 - must double-click to put in edit mode
 - can easily use `JComboBox`
 - see example below
 - using other components is more work
 - very similar to `MovieRatingTableCellRenderer`

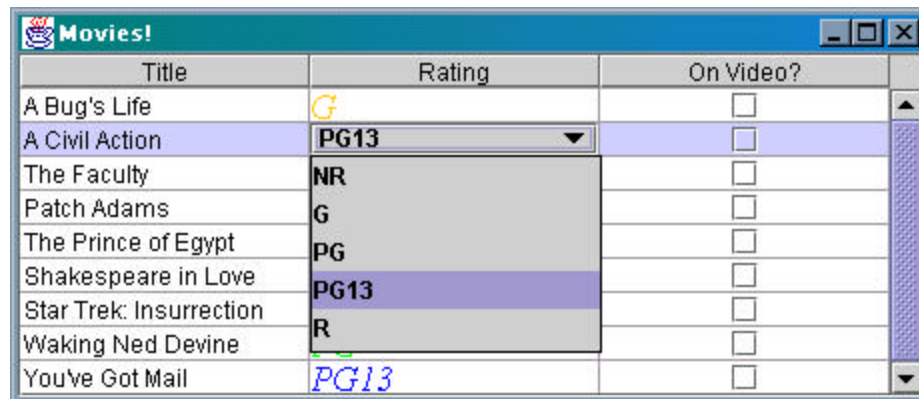
- **JComboBox cell editor**

```
JComboBox comboBox = new JComboBox(Movie.RATINGS);
TableModel tcm = table.getColumnModel();
TableColumn tc =
    tcm.getColumnModel().getColumn(MovieTableModel.RATING_COLUMN);
tc.setCellEditor(new DefaultCellEditor(comboBox));
```

- implements `TableCellEditor`
- cells edited using this should contain one of the strings in `Movie.RATINGS`
- if not then the first string in the `JComboBox` is selected



TableCellEditor Interface (Cont'd)



Title	Rating	On Video?
A Bug's Life	G	<input type="checkbox"/>
A Civil Action	PG13	<input type="checkbox"/>
The Faculty	NR	<input type="checkbox"/>
Patch Adams	G	<input type="checkbox"/>
The Prince of Egypt	PG	<input type="checkbox"/>
Shakespeare in Love	PG13	<input type="checkbox"/>
Star Trek: Insurrection	R	<input type="checkbox"/>
Waking Ned Devine		<input type="checkbox"/>
You've Got Mail	PG13	<input type="checkbox"/>

- TableCellEditor interface
 - only one method
 - Component getTableCellEditorComponent
(JTable table,
Object value,
boolean isSelected,
int row,
int column)
 - can return different editing components based on
 - the value or its class
 - whether the cell is selected
 - the row and column that contains the value
 - implements CellEditor so seven other methods must be written
 - see SliderEditor example on next page



SliderEditor

```
import java.awt.Component;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.TableCellEditor;

public class SliderEditor extends JSlider
implements TableCellEditor {

    private transient Vector listeners = new Vector();
    private int savedValue;

    public SliderEditor(int min, int max) {
        super(min, max, min);
    }

    /**
     * @return the Component to be used for editing.
     */
    public Component getTableCellEditorComponent
        (JTable table, Object value,
         boolean isSelected, int row, int column) {
        savedValue = ((Integer) value).intValue();
        setValue(savedValue);
        return this;
    }
}
```



SliderEditor (Cont'd)

```
public Object getCellEditorValue() {
    return new Integer(getValue());
}

/**
 * Can editing begin based on the event passed in?
 */
public boolean isCellEditable(EventObject anEvent) {
    return true;
}

/**
 * Should the cell be selected based on the event passed in?
 */
public boolean shouldSelectCell(EventObject anEvent) {
    return true;
}

public void addCellEditorListener(CellEditorListener l) {
    listeners.addElement(l);
}

public void removeCellEditorListener(CellEditorListener l) {
    listeners.removeElement(l);
}
```



SliderEditor (Cont'd)

```
/**
 * Cancels editing and discards the edited value.
 */
public void cancelCellEditing() {
    setValue(savedValue); // revert to last saved value

    ChangeEvent ce = new ChangeEvent(this);
    Vector copy = (Vector) listeners.clone();
    Enumeration e = copy.elements();
    while (e.hasMoreElements()) {
        CellEditorListener l =
            (CellEditorListener) e.nextElement();
        l.editingCanceled(ce);
    }
}
```



SliderEditor (Cont'd)

```
/**
 * Stops editing and accepts the edited value.
 * Returns false if editing wasn't stopped.
 * Useful for editors that validate and
 * can't accept invalid entries.
 */
public boolean stopCellEditing() {
    ChangeEvent ce = new ChangeEvent(this);
    Vector copy = (Vector) listeners.clone();
    Enumeration e = copy.elements();
    while (e.hasMoreElements()) {
        CellEditorListener l =
            (CellEditorListener) e.nextElement();
        l.editingStopped(ce);
    }

    return true;
}
}
```



MovieRatingEditor

```
import java.util.Hashtable;
import javax.swing.JLabel;

public class MovieRatingEditor extends SliderEditor {

    private static Hashtable labelTable =
        new Hashtable(Movie.RATINGS.length);

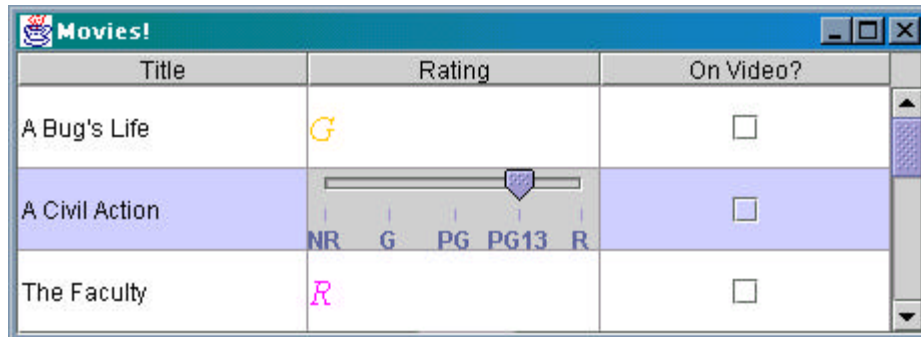
    static {
        for (int i = Movie.NR; i <= Movie.R; i++) {
            addLabel(i);
        }
    }

    public MovieRatingEditor() {
        super(Movie.NR, Movie.R);
        setPaintTicks(true);
        setMinorTickSpacing(1);
        setMajorTickSpacing(1);
        setSnapToTicks(true);
        setPaintLabels(true);
        setLabelTable(labelTable);
    }

    private static void addLabel(int ratingIndex) {
        labelTable.put
            (new Integer(ratingIndex),
             new JLabel(Movie.RATINGS[ratingIndex]));
    }
}
```



Using the MovieRatingEditor



- **Modify MovieTableModel**
 - `getValueAt()` must return an Integer for ratings instead of a String
 - `getColumnClass()` must return `Integer.class` instead of `String.class`
 - `setValueAt()` must accept an Integer for ratings instead of a String
- **To request this editor for the Rating column**

```
TableColumnModel tcm = table.getColumnModel();
TableColumn tc =
    tcm.getColumn(MovieTableModel.RATING_COLUMN);
tc.setCellEditor(new MovieRatingEditor());
table.setRowHeight(40); // so the editor will fit
```

