Chapter 40 JTable and JTree
Objectives

- To display tables using JTable (§40.2).
- To process rows and columns usingTableModel, DefaultTableModel, TableColumnModel, DefaultTableColumnModel, and ListSelectionModel (§§40.3-40.5).
- To enable auto sort and filtering on table model (§40.4).
- To add rows and columns, delete rows and columns (§40.5).
- To render and edit table cells using the default renderers and editors (§40.6).
- To render and edit table cells using the custom renderers and editors (§40.7).
- To handle table model events (§40.8).
- To display data in a tree hierarchy using JTree (§40.9).
- To model the structure of a tree using using TreeModel and DefaultTreeModel (§40.10).
- To add, remove, and process tree nodes using TreeNode, DefaultMutableTreeNode, and TreePath (§40.11).
- To select tree nodes and paths using TreeSelectionModel and DefaultTreeSelectionModel (§40.12).
- To render and edit tree nodes using the default and custom renderers and editors (§40.14).
JTable

JTable is a Swing component that displays data in rows and columns in a two-dimensional grid.
JTable and Its Supporting Models

NOTE: All the supporting interfaces and classes for JTable are grouped in the javax.swing.table package.
The JTable Class

- autoCreateColumnsFromModel: boolean
- autoResizeMode: int
- cellEditor: TableCellEditor
- cellSelectionEnabled: boolean
- columnModel: TableColumnModel
- columnSelectionAllowed: boolean
- editingColumn: int
- editingRow: int
- gridColor: java.awt.Color
- intercellSpacing: Dimension
- model: TableModel
- rowCount: int
- rowHeight: int
- rowMargin: int
- rowSelectionAllowed: boolean
- selectionBackground: java.awt.Color
- selectionForeground: java.awt.Color
- showGrid: boolean
- selectionMode: int
- selectionModel: ListSelectionModel
- showHorizontalLines: boolean
- showVerticalLines: boolean
- tableHeader: JTableHeader

Indicates whether the columns are created in the table (default: true).
Specifies how columns are resized (default: SUBSEQUENT_COLUMNS).
Specifies a cell editor.
Specifies whether individual cells can be selected (Obsolete since JDK 1.3).
Maintains the table column data.
Specifies whether the rows can be selected (default: false).
Specifies the column of the cell that is currently being edited.
Specifies the row of the cell that is currently being edited.
The color used to draw grid lines (default: GRAY).
Specifies the horizontal and vertical margins between cells (default: 1, 1).
Maintains the table model.
Read-only property that counts the number of rows in the table.
Specifies the row height of the table (default: 16 pixels).
Specifies the vertical margin between rows (default: 1 pixel).
Specifies whether the rows can be selected (default: true).
The background color of selected cells.
The foreground color of selected cells.
Specify whether the grid lines are displayed (write-only, default: true).
Specifies a selection mode (write-only).
Specifies a selection model.
Specifies whether the horizontal grid lines are displayed (default: true).
Specifies whether the vertical grid lines are displayed (default: true).
Specifies a table header.

Creates a default JTable.
Creates a JTable with the specified number of empty rows and columns.
Creates a JTable with the specified row data and column header names.
Creates a JTable with the specified table model.
Creates a JTable with the specified table model and table column model.
Creates a JTable with the specified table model, table column model, and selection model.
Creates a JTable with the specified row data and column data in vectors.
Adds a new column to the table.
Deselects all selected columns and rows.
Edits the cell if it is editable.
Returns the default editor for the column.
Returns the default renderer for the column.
Sets the default editor for the column.
Sets the default renderer for the column.
Example: Table Properties Demo

Problem: This example demonstrates the use of several JTable properties. The example creates a table and allows the user to choose an Auto Resize Mode, specify the row height and margin, and indicate whether the grid is shown.

Run TablePropertiesDemo
Table Models

JTable delegates data storing and processing to its table data model. A table data model must implement the TableModel interface, which defines the methods for registering table model listeners, manipulating cells, and obtaining row count, column count, column class, and column name.

The AbstractTableModel class provides partial implementations for most of the methods in TableModel. It takes care of the management of listeners and provides some conveniences for generating TableModelEvents and dispatching them to the listeners.

The DefaultTableModel provides concrete storage for data using a vector.
Table column models manage columns in a table. They can be used to select, add, move, and remove table columns. A table column model must implement the `TableColumnModel` interface, which defines the methods for registering table column model listeners, and for accessing and manipulating columns.

**javax.swing.table.TableColumnModel**

- `addColumn(aColumn: TableColumn): void`
- `getColumn(columnIndex: int): TableColumn`
- `getColumnCount(): int`
- `getColumnIndex(columnIdentifier:Object): int`
- `getColumnMargin(): int`
- `getColumns(): Enumeration`
- `getColumnSelectionAllowed(): boolean`
- `getSelectedColumnCount(): int`
- `getSelectedColumns(): void`
- `getSelectionModel(): ListSelectionModel`
- `moveColumn(columnIndex: int, newIndex: int): void`
- `removeColumn(column: TableColumn): void`
- `setColumnMargin(newMargin: int): void`
- `setColumnSelectionAllowed(flag: boolean): void`
- `setSelectionModel(newModel: ListSelectionModel): void`

**javax.swing.table.DefaultTableColumnModel**

DefaultTableColumnModel is a concrete class that implements `TableColumnModel` and stores its columns in a vector and contains an instance.
The **TableColumn** Class

The column model deals with all the columns in a table. The `TableColumn` class is used to model an individual column in the table. An instance of `TableColumn` for a specified column can be obtained using the `getColumn(index)` method in `TableColumnModel` or the `getColumn(columnIdentifier)` method in `JTable`.

```java
javax.swing.table.TableColumn
+cellEditor: TableCellEditor
+cellRenderer: TableCellRenderer
+headerRenderer: TableCellRenderer
+headerValue: Object
+identifier: Object
+maxWidth: int
+minWidth: int
+modelIndex: int
+preferredWidth: int
+resizable: boolean
+width: int

+TableColumn()
+TableColumn(modelIndex: int)
+TableColumn(modelIndex: int, width: int)
+TableColumn(modelIndex: int, width: int, cellRenderer: TableCellRenderer)
+sizeWidthToFit(): void
```

- The editor for editing a cell in this column.
- The renderer for displaying a cell in this column.
- The renderer for displaying the header of this column.
- The header value of this column.
- The identifier for this column.
- The maximum width of this column.
- The minimum width of this column (default: 15 pixels).
- The index of the column in the table model (default: 0).
- The preferred width of this column (default: 75 pixels).
- Indicates whether this column can be resized (default: true).
- Specifies the width of this column (default: 75 pixels).

- Constructs a default table column.
- Constructs a table column for the specified column.
- Constructs a table column with the specified column and width.
- Constructs a table column with the specified column, width, and cell renderer.
- Resizes the column to fit the width of its header cell.
The JTableHeader Class

JTableHeader is a GUI component that manages the header of the JTable (see Figure 40.29). When you create a JTable, an instance of JTableHeader is automatically created and stored in the tableHeader property.

```java
javafx.swing.table.JTableHeader
+columnModel: TableColumnModel
+draggedColumn: TableColumn
+draggedDistance: TableCellRenderer
+reorderingAllowed: boolean
+resizingAllowed: boolean
+resizingColumn: TableColumn
+table: JTable

+JTableHeader()
+JTableHeader(TableColumnModel cm)
```

The TableColumnModel of the table header.
The column being dragged.
The distance from its original position to the dragged position.
Whether reordering of columns is allowed (default: true).
Whether resizing of columns is allowed (default: true).
The column being resized.
The table for which this object is the header.

Constructs a JTableHeader with a default TableColumnModel.
Constructs a JTableHeader with with a TableColumnModel.
Auto Sort and Filtering

Auto sort and filtering are two useful new features in JDK 1.6. To enable auto sort on any column in a JTable, create an instance of TableRowSorter with a table model and set JTable’s rowSorter with this JTable instance, as follows:

```java
TableRowSorter<TableModel> sorter =
    new TableRowSorter<TableModel>(tableModel);
jTable.setRowSorter(sorter);
```
Example: Modifying Rows and Columns

Problem: This example demonstrates the use of table models, table column models, list-selection models, and the TableColumn class. The program allows the user to choose selection mode and selection type, to add or remove rows and columns, and to save, clear, and restore table.
Table Renderers and Editors

Table cells are painted by cell renderers. By default, a cell object's string representation \( \text{toString()} \) is displayed and the string can be edited as it was in a text field. \textbf{JTable} maintains a set of predefined renderers and editors, listed in Table 40.1, which can be specified to replace default string renderers and editors.

The predefined renderers and editors are automatically located and loaded to match the class returned from the \texttt{getColumnClass()} method in the table model. To use a predefined renderer or editor for a class other than \texttt{String}, you need to create your own table model by extending a subclass of \texttt{TableModel}. In your table model class, you need to override the \texttt{getColumnClass()} method to return the class of the column, as follows:

```java
public Class getColumnClass(int column) {
    return getValueAt(0, column).getClass();
}
```

By default, all cells are editable. To prohibit a cell from being edited, override the \texttt{isCellEditable(int rowIndex, int columnIndex)} method in \texttt{TableModel} to return \texttt{false}. By default, this method returns \texttt{true} in \texttt{AbstractTableModel}. 

Liang, Introduction to Java Programming, Ninth Edition, (c) 2013 Pearson Education, Inc. All rights reserved.
Example: Using Predefined Table Renderers and Editors

Problem: Write a program that displays a table for the books. The table consists of three rows with column names Title, Copies Needed, Publisher, Date Published, In-Stock, and Book Photo, as shown in Figure 40.32. Display all the columns using the predefined renderers and editors. Assume dates and icons are not editable; prohibit users from editing of these two columns.
Custom Table Renderers and Editors

Predefined renderers and editors are convenient and easy to use, but their functions are limited. The predefined image icon renderer displays the image icon in a label. The image icon cannot be scaled. If you want the whole image to fit in a cell, you need to create a custom renderer.

A custom renderer can be created by extending the DefaultTableCellRenderer, which is a default implementation for the TableCellRender interface. The custom renderer must override the getTableCellRendererComponent() to return a component for rendering the table cell. The getTableCellRendererComponent() is defined as follows:

```java
public Component getTableCellRendererComponent
    (JTable table, Object value, boolean isSelected,
     boolean isFocused, int row, int column)
```

This method signature is very similar to the getListItemRendererComponent() method used to create custom list cell renderers.
**Example: Using Custom Table Renderers and Editors**

Problem: Revise Example 40.9, “Using Predefined Table Renderers and Editors,” to display scaled image icons and to use a custom combo editor to edit the cells in the Publisher column.

Run `CustomTableCellRenderEditorDemo`
Table Events

JTable does not fire table events. It fires the events such as MouseEvent, KeyEvent, and ComponentEvent inherited from its superclass JComponent. Table events are fired by table models, table column models, and table-selection models whenever changes are made to these models. Table models fire TableModelEvent when table data are changed. Table column models fire TableColumnModelEvent when columns are added, removed, or moved, or when the column selection changes. Table-selection models fire ListSelectionEvent when the selection changes.
**Example: Using Table Events**

Problem: This example demonstrates handling table events. The program displays messages on a text area when a row or a column is selected, when a cell is edited, or when a column is removed.
**JTree**

**JTree** is a Swing component that displays data in a treelike hierarchy.

![Diagram of a JTree component](image)

- **Root**
- **Leaf**
Tree Models

While JTree displays the tree, the data representation of the tree is handled by TreeModel, TreeNode, and TreePath. TreeModel represents the entire tree, TreeNode represents a node, and TreePath represents a path to a node. Unlike the ListModel or TableModel, the tree model does not directly store or manage tree data. Tree data are stored and managed in TreeNode and TreePath.

javax.swing.JTree

- model: TreeModel
- anchorPath: TreePath
- leadPath: TreePath
- selectionModel: TreeSelectionModel
- cellEditor: TreeCellEditor
- cellRenderer: TreeCellEditor

TreeNode

TreeModel

DefaultTreeModel

MutableTreeNode

TreePath

DefaultMutableTreeNode

TreeSelectionModel

DefaultTreeSelectionModel

TreeCellRenderer

DefaultTreeCellRenderer

TreeCellEditor

DefaultTreeCellEditor

The TreeSelectionModel interface handles tree node selection.

The DefaultTreeCellRenderer class provides a default tree node renderer that can display a label and/or an icon in a node.

The DefaultTreeCellEditor can be used to edit the cells in a text field.
The JTree Class

java.awt.JTree

#cellEditor: TreeCellEditor
#cellRenderer: TreeCellRenderer
#editable: boolean
#model: TreeModel
#rootVisible: boolean
#rowHeight: int
#scrollsOnExpand: boolean
#selectionModel: TreeSelectionModel
#showsRootHandles: boolean
#toggleClickCount: int
- anchorSelectionPath: TreePath
- expandsSelectedPaths: boolean
- leadSelectionPaths: TreePath

- JTree()
- JTree(value: java.util.Hashtable)
- JTree(value: Object[])
- JTree(newModel: TreeModel)
- JTree(root: TreeNode)
- JTree(root: TreeNode, asksAllowsChildren: boolean)
- JTree(value: Vector)

- addSelectionPath(path: TreePath): void
- addSelectionPaths(paths: TreePath[]): void
- addSelectionRow(row: int): void
- addSelectionRows(rows: int[]): void
- clearSelection() : void
- collapsePath(path: TreePath): void

- getSelectionPath(): TreePath
- getSelectionPaths(): TreePath[]
- getLastSelectedPathComponent() 
- getRowCount():int
- removeSelectionPath(path: TreePath): void
- removeSelectionPaths(paths: TreePath[]):void

Specifies a cell editor used to edit entries in the tree.
Specifies whether individual cells can be selected (Obsolete since JDK 1.3).
Specifies whether the cells are editable (default: false).
Maintains the tree model.
Specifies whether the root is displayed (depending on the constructor).
Specifies the height of the row for the node displayed in the tree (default: 16 pixels).
If true, when a node is expanded, as many of the descendants are scrolled to be visible (default: 16 pixels).
Models the set of selected nodes in this tree.
Specifies whether the root handles are displayed (default: true).
Number of mouse clicks before a node is expanded (default: 2).
The path identified as the anchor.
True if paths in the selection should be expanded (default: true).
The path identified as the lead.
Creates a JTree with a sample tree model, as shown in Figure 24.35.
Creates a JTree with an invisible root and the keys in the Hashtable key/value pairs as its children.
Creates a JTree with an invisible root and the elements in the array as its children.
Creates a JTree with the specified tree model.
Creates a JTree with the specified tree node as its root.
Creates a JTree with the specified tree node as its root and decides whether a node is a leaf node in the specified manner.
Creates a JTree with an invisible root and the elements in the vector as its children.
Adds the specified TreePath to the current selection.
Adds the specified TreePaths to the current selection.
Adds the path at the specified row to the current selection.
Adds the path at the specified rows to the current selection.
Clears the selection.
Ensures that the node identified by the specified path is collapsed and viewable.
Returns the path from the root to the first selected node.
Returns the paths from the root to all the selected nodes.
Returns the last node in the first selected TreePath.
Returns the number of rows currently being displayed.
Removes the node in the specified path.
Removes the node in the specified paths.
Example: Simple Tree Demo

Problem: Write a program to create four trees: a default tree using the no-arg constructor, a tree created from an array of objects, a tree created from a vector, and a tree created from a hash table. Enable the user to dynamically set the properties for rootVisible, rowHeight, and showsRootHandles.
**TreeModel and DefaultTreeModel**

TreeModel contains the structural information about the tree, and tree data are stored and managed by TreeNode.

DefaultTreeModel is a concrete implementation for TreeModel that uses TreeNodes.

Here are the methods of `javax.swing.tree.TreeModel`:
- `getChild(parent: Object, index: int): Object` returns the child of parent at the index in the parent's child array.
- `getChildCount(parent: Object): int` returns the number of children of the specified parent in the tree model.
- `getIndexOfChild(parent: Object, child: Object): int` returns the index of child in parent. If parent or child is null, returns –1.
- `getRoot(): Object` returns the root of the tree. Returns null if the tree is empty.
- `isLeaf(node: Object): boolean` returns true if the specified node is a leaf.
- `addTreeModelListener(listener: TreeModelListener): void` adds a listener for the TreeModelEvent posted after the tree changes.
- `removeTreeModelListener(listener: TreeModelListener): void` removes a listener previously added with addTreeModelListener.
- `valueForPathChanged(path: TreePath, newValue: Object): void` messaged when the user has altered the value for the item identified by path to newValue.

Here are the methods of `javax.swing.tree.DefaultTreeModel`:
- `asksAllowsChildren: Boolean` tells how leaf nodes are determined. True if only nodes that do not allow children are leaf nodes, false if nodes that have no children are leaf nodes.
- `root: TreeNode` the root of the tree.
- `DefaultTreeModel(root: TreeNode)` creates a DefaultTreeModel with the specified root.
- `DefaultTreeModel(root: TreeNode, asksAllowsChildren: boolean)` creates a DefaultTreeModel with the specified root and decides whether a node is a leaf node in the specified manner.
- `getFolderPathToRoot(aNode: TreeNode): TreeNode[]` returns asksAllowsChildren.
- `getPathToRoot(aNode: TreeNode): TreeNode[]` returns the nodes in an array from root to the specified node.
- `reload(): void` reloads the model (invoke this method if the tree has been modified).
- `removeNodeFromParent(node: TreeNode): void` removes the node from its parent.
TreeNode, MutableTreeNode, and DefaultMutableTreeNode

TreeNode stores models a single node in the tree.

MutableTreeNode defines a subinterface of TreeNode with additional methods for changing the content of the node, for inserting a new parent, and for removing the node itself.

DefaultMutableTreeNode is a concrete implementation of MutableTreeNode.

DefaultMutableTreeNode is a concrete implementation of MutableTreeNode.

These four methods return the first (last, next, and previous) leaf that is a descendant of this node. The first (last, next, and previous) leaf is recursively defined as the first (last, next, and previous) child of a leaf. These methods are defined by the node's implementation and return the root of the tree that contains this node.

DepthFirstEnumeration: These four methods return the first (last, next, and previous) leaf that is a descendant of this node. These methods are defined by the node's implementation and return the root of the tree that contains this node.

DefaultMutableTreeNode is a concrete implementation of MutableTreeNode.
Example: Tree Model Demo

Problem: Write a program to create two trees that displays world, continents, countries and states. The two trees display identical contents. The program also displays the properties of the tree in a text area.

TreeModelDemo Run
The TreePath Class

The TreePath class represents a path from an ancestor to a descendant in a tree.

```java
javax.swing.tree.TreePath
+TreePath(singlePath: Object)
+TreePath(path: Object)
+getLastPathComponent(): Object
+getParentPath(): TreePath
+getPath(): Object
+getPathComponent(element: int): Object
+getPathCount(): int
+isDescendant(aTreePath: TreePath): Boolean
+pathByAddingChild(child: Object): TreePath
```

Constructs a TreePath containing only a single element.
Constructs a path from an array of objects.
Returns the last component of this path.
Returns a path containing all but the last path component.
Returns an ordered array of objects containing the components of this TreePath.
Returns the path component at the specified index.
Returns the number of elements in the path.
Returns true if aTreePath contains all the components in this TreePath.
Returns a new path containing all the elements of this TreePath plus child.
TreeSelectionModel and DefaultTreeSelectionModel

The selection of tree nodes is defined in the TreeSelectionModel interface.

javax.swing.tree.TreeSelectionModel

- addSelectionPath(path: TreePath): void
- addSelectionPaths(paths: TreePath[]): void
- clearSelection(): void
- getLeadSelectionPath(): TreePath
- getSelectionCount(): int
- getSelectionPath(): TreePath
- getSelectionPaths(): TreePath[]
- getSelectionMode(): int
- removeSelectionPath(path: TreePath): void
- removeSelectionPaths(paths: TreePath[]): void
- setSelectionMode(mode: int): void
- setSelectionPath(path: TreePath): void
- setSelectionPaths(paths: TreePath[]): void
- addTreeSelectionListener(x: TreeSelectionListener): void
- removeTreeSelectionListener(x: TreeSelectionListener): void

The DefaultTreeSelectionModel class is a concrete implementation of the TreeSelectionModel, which maintains an array of TreePath objects representing the current selection.

javax.swing.tree.DefaultTreeSelectionModel

- addSelectionPath(path: TreePath): void
- addSelectionPaths(paths: TreePath[]): void
- clearSelection(): void
- getLeadSelectionPath(): TreePath
- getSelectionCount(): int
- getSelectionPath(): TreePath
- getSelectionPaths(): TreePath[]
- getSelectionMode(): int
- removeSelectionPath(path: TreePath): void
- removeSelectionPaths(paths: TreePath[]): void
- setSelectionMode(mode: int): void
- setSelectionPath(path: TreePath): void
- setSelectionPaths(paths: TreePath[]): void
- addTreeSelectionListener(x: TreeSelectionListener): void
- removeTreeSelectionListener(x: TreeSelectionListener): void

Adds the specified TreePath to the current selection.
Adds the specified TreePaths to the current selection.
Clears the selection.
Returns the last path in the selection.
Returns the number of paths in the selection.
Returns the first path in the selection.
Returns all the paths in the selection.
Returns the current selection mode.
Removes path from the selection.
Removes paths from the selection.
Sets the selection mode.
Sets the selection to path.
Sets the selection to paths.
Register a TreeSelectionListener.
Remove a TreeSelectionListener.
Example: Modifying Trees

Problem: Write a program to create two trees that displays the same contents: world, continents, countries and states, as shown in Figure 40.44. For the left tree on the left, enable the user to choose a selection mode, add a new child under the first selected node, and remove all the selected nodes.
Tree Node Rendering

DefaultTreeCellRenderer renderer =
    (DefaultTreeCellRenderer)jTree1.getCellRenderer();
renderer.setLeafIcon(yourCustomLeafImageIcon);
renderer setOpenIcon(yourCustomOpenImageIcon);
renderer.setClosedIcon(yourCustomClosedImageIcon);
renderer setBackgroundSelectionColor(Color.red);
// Customize editor
JComboBox jcboColor = new JComboBox();
jcboColor.addItem("red");
jcboColor.addItem("green");
jcboColor.addItem("blue");
jcboColor.addItem("yellow");
jcboColor.addItem("orange");

jTree1.setCellEditor(new DefaultCellEditor(jcboColor));
jTree1.setEditable(true);
Tree Rendering and Editing

defaultTreeCellEditor

    (new DefaultTreeCellEditor(jTree1,
    new DefaultTreeCellRenderer(),
    new DefaultCellEditor(jcboColor)));
Tree Events

JTree can fire TreeSelectionEvent and TreeExpansionEvent, among many other events. Whenever a new node is selected, JTree fires a TreeSelectionEvent. Whenever a node is expanded or collapsed, JTree fires a TreeExpansionEvent. To handle the tree selection event, a listener must implement the TreeSelectionListener interface, which contains a single handler named valueChanged method. TreeExpansionListener contains two handlers named treeCollapsed and treeExpanded for handling node expansion or node closing.