

K07 JTable

- Inhalt
 1. JTable
 1. Übersicht
 2. Einfache Tabelle
 3. Konfiguration
 4. Selektion
 5. Zugriff auf Inhalt
 2. Modelle
 1. Tabellenmodell
 2. Spaltenmodell
 3. Renderer

Lernziele

- Sie kennen die Möglichkeiten von JTable und sind in der Lage diese in eigenen Applikationen einzusetzen
- Sie kennen die Modelle
 - Tabellenmodell
 - Spaltenmodell
- Sie können durch den Einsatz von Renderern das Erscheinungsbild von Zellen bestimmen

1.1 Übersicht (1)

- JTable
 - Eines der wichtigsten Dialogelemente ist eine Tabelle
 - Sehr anspruchsvoll
 - *JTable*
 - Swing-Klasse, welche Programmierarbeit erleichtert
 - Darstellung von *tabellarisch* Daten
 - *textuell*
 - *graphisch*
 - Weitreichende Möglichkeiten
 - Konfiguration der Tabelle
 - Anpassen von Inhalten
 - Reagieren auf Benutzerereignisse

1.1 Übersicht (2)

- Konstruktoren

```
public JTable(Object[][] rowData, Object[] columnNames)
```

- Daten in zweidimensionalem Array (Zeilen/Spalten)
- Array mit Bezeichnung der Spaltenköpfe

```
public JTable(Vector rowData, Vector columnNames)
```

- Datenvektor (Vektor mit Vektoren)
- Vektor mit Bezeichnung der Spaltenköpfe

```
public JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm))
```

- TableModel stellt Daten zur Verfügung
- TableColumnModel definiert Spalten
- ListSelectionModel definiert Verhalten bei Selektion

1.2 Einfache Tabelle: Beispiel (1)

- Daten in zweidimensionalem Array

```
public interface JTableDemo1Data
{
    public static final String[][] DATA =
    {
        { " 1/2002", "33", "Die Neulinge: JBuilder6 und Forte for Java 3"},
        { " 2/2002", "36", "Enterprise Development"},
        { " 3/2002", "34", "Web Frameworks"},
        { " 4/2002", "31", "Java Web Start und JNLP im Überblick"},
        { " 5/2002", "40", "Java Message Service: Einführung und Praxis"},
        { " 6/2002", "35", "Java meets Office"},
        { " 7/2002", "35", "Java Datenbank API"},
        { " 8/2002", "34", "Eigene JSP TagLibs entwickeln"},
        { " 9/2002", "39", "Suchmaschinen entwickeln mit Apache und Lucene"}
    };

    public static final String[] COLHEADS =
    { "Ausgabe", "Seite", "Titelthema" };
}
```

1.2 Einfache Tabelle: Beispiel (2)

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

public class JTableDemo1 extends JFrame
    implements JTableDemo1Data
{
    public JTableDemo1()
    {
        super("JTableDemo1");
        JTable table = new JTable(DATA, COLHEADS);
        Container cp = getContentPane();
        cp.add(new JLabel("Titelthemen Javamagazin:"), "North");
        cp.add(new JScrollPane(table), "Center");
    }

    public static void main(String[] args)
    {
        JTableDemo1 frame = new JTableDemo1();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocation(100, 100);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```



The screenshot shows a Java Swing window titled "JTableDemo1". Inside the window, there is a table with the title "Titelthemen Javamagazin:". The table has three columns: "Ausgabe", "Seite", and "Titelthema". The data rows are as follows:

| Ausgabe | Seite | Titelthema |
|---------|-------|--|
| 1/2002 | 33 | Die Neulinge: JBuilder6 und Forte for Java 3 |
| 2/2002 | 36 | Enterprise Development |
| 3/2002 | 34 | Web Frameworks |
| 4/2002 | 31 | Java Web Start und JNLP im Überblick |
| 5/2002 | 40 | Java Message Service: Einführung und Praxis |
| 6/2002 | 35 | Java meets Office |
| 7/2002 | 35 | Java Datenbank API |

1.3 Konfiguration (1)

- Wichtigste Methoden

| Methode | Beschreibung |
|---|--|
| <code>void setRowHeight(int newHeight)</code> | Setzt Gesamthöhe der Zeile |
| <code>void setRowMargin(int rowMargin)</code> | Setzt freibleibender Raum oberhalb und unterhalb der Zeile |
| <code>void setIntercellSpacing(Dimension newSpacing)</code> | Setzt horizontalen Rand zwischen den Zellen |
| <code>void setShowGrid(boolean b)</code> | Schaltet alle Begrenzungslinien Ein/Aus |
| <code>void setShowHorizontalLines(boolean b)</code> | Schaltet horizontale Begrenzungslinien Ein/Aus |
| <code>void setShowVerticalLines(boolean b)</code> | Schaltet vertikale Begrenzungslinien Ein/Aus |
| <code>void setGridColor(Color newColor)</code> | Setzt Farbe der Begrenzungslinien |
| <code>void setSelectionForeground(Color selectionForeground)</code> | Setzt Vordergrundfarbe für selektierte Zellen |
| <code>void setSelectionBackground(Color selectionBackground)</code> | Setzt Hintergrundfarbe für selektierte Zellen |
| <code>void setAutoResizeMode(int mode)</code> | Bestimmt Verhalten der Tabelle bei Grössenänderungen |

1.3 Konfiguration (2)

- Konstanten für *setAutoResizeMode*

| Methode | Beschreibung |
|---|---|
| <code>AUTO_RESIZE_OFF</code> | Keine automatische Grössenanpassung der übrigen Spalten. Wurde die Tabelle in JScrollPane verpackt, bekommt sie nötigenfalls einen horizontalen Schieberegler. |
| <code>AUTO_RESIZE_LAST_COLUMN</code> | Die letzte Spalte wird zum Grössenausgleich verwendet. Dadurch reduziert sich der Platz für die letzte Spalte, wenn eine andere Spalte vergrössert wird, und er erhöht sich, wenn sie verkleinert wird. |
| <code>AUTO_RESIZE_NEXT_COLUMN</code> | Die rechts neben der modifizierten Spalte liegende Spalte wird zum Grössenausgleich verwendet. |
| <code>AUTO_RESIZE_SUBSEQUENT_COLUMNS</code> | Die Grössenänderung wird gleichmässig auf alle nachfolgenden Spalten verteilt. |
| <code>AUTO_RESIZE_ALL_COLUMNS</code> | Die Grössenänderung wird auf alle Spalten der Tabelle verteilt. |

1.4 Selektion (1)

- Selektionsmodi

| Methode | Beschreibung |
|---|---|
| <code>void setRowSelectionAllowed(boolean flag)</code> | Erlaubt Selektion von Zeilen |
| <code>void setColumnSelectionAllowed(boolean flag)</code> | Erlaubt Selektion von Spalten |
| <code>void setSelectionMode(int selectionMode)</code> | Setzt Modus für Zellenselektion (siehe JSDK Doku) |
| <code>void setCellSelectionEnabled(boolean flag)</code> | Erlaubt Selektion von Zellen |

- Abfrage der Selektion

| Methode | Beschreibung |
|---|--|
| <code>int getSelectedRow()</code> | Gibt selektierte Zeile zurück |
| <code>int getSelectedColumn()</code> | Gibt selektierte Spalte zurück |
| <code>int[] getSelectedRows()</code> | Gibt selektierte Zeilen in Array zurück |
| <code>int[] getSelectedColumns()</code> | Gibt selektierte Spalten in Array zurück |

1.4 Selektion (2)

- Verändern der Selektion

| Methode | Beschreibung |
|---|---|
| <code>void selectAll()</code> | Selektiert gesamte Tabelle |
| <code>void setRowSelectionInterval(int index0, int index1)</code> | Mehrere zusammenhängende Zeilen können selektiert werden |
| <code>void addRowSelectionInterval(int index0, int index1)</code> | Nimmt mehrere zusammenhängende Zeilen zur Selektion hinzu |
| <code>void removeRowSelectionInterval(int index0, int index1)</code> | Entfernt mehrere zusammenhängende Zeilen von der Selektion |
| <code>void setColumnSelectionInterval(int index0, int index1)</code> | Mehrere zusammenhängende Spalten können selektiert werden |
| <code>void addColumnSelectionInterval(int index0, int index1)</code> | Nimmt mehrere zusammenhängende Spalten zur Selektion hinzu |
| <code>void removeColumnSelectionInterval(int index0, int index1)</code> | Entfernt mehrere zusammenhängende Spalten von der Selektion |

1.5 Zugriff auf Inhalt


- Zugriff auf Daten

| Methode | Beschreibung |
|--|--|
| <code>int getRowCount()</code> | Gibt Anzahl Zeilen zurück |
| <code>int getColumnCount()</code> | Gibt Anzahl Spalten zurück |
| <code>Object getValueAt(int row, int column)</code> | Gibt Objekt an Position (row, column) zurück |
| <code>void setValueAt(Object aValue, int row, int column)</code> | Setzt Objekt an Position (row, column) |

- Editieren von Daten

| Methode | Beschreibung |
|--|--|
| <code>boolean isEditing()</code> | Gibt true zurück, wenn eine Zelle der Tabellen gerade editiert wird |
| <code>int getEditingRow()</code> | Gibt Zeile zurück, in der eine Zelle der Tabelle gerade editiert wird |
| <code>int getEditingColumn()</code> | Gibt Spalte zurück, in der eine Zelle der Tabelle gerade editiert wird |
| <code>boolean editCellAt(int row, int column)</code> | Programm erzwingt das Editieren einer Zelle |

2.1 Tabellenmodell

- **Tabellenmodell**
 - Verantwortlich für die Beschaffung der Daten
- 
- ```

graph LR
 Model([Model
TableModel]) --> View([View
JTable])

```
- **Default Tabellenmodell**
    - *DefaultTableModel*
  - **Verwenden von eigenen Tabellenmodellen**
    - Daten *komplex* strukturiert sind
    - Ein Array die Daten nicht verwalten kann
    - Daten an *externe Quellen* gebunden sind
  - **Erzeugen von eigenen Tabellenmodellen**
    - Durch Implementation des Interface *TableModel*
    - *TableModel* aus *javax.swing.table*

### 2.1.1 TabelModel

- Die wichtigsten Methoden des Interface

| Methode                                                                    | Beschreibung                                                                            |
|----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <code>int getRowCount()</code>                                             | Gibt Anzahl Zeilen im Modell zurück                                                     |
| <code>int getColumnCount()</code>                                          | Gibt Anzahl Spalten im Modell zurück                                                    |
| <code>String getColumnName(int columnIndex)</code>                         | Liefert Spaltenüberschrift                                                              |
| <code>Class getColumnClass(int columnIndex)</code>                         | Liefert Datentyp der Spalte zurück                                                      |
| <code>boolean isCellEditable(int rowIndex, int columnIndex)</code>         | Gibt true zurück, wenn Wert editiert werden kann                                        |
| <code>Object getValueAt(int rowIndex, int columnIndex)</code>              | Erfragt einen bestimmten Wert im Modell                                                 |
| <code>void setValueAt(Object aValue, int rowIndex, int columnIndex)</code> | Schreibt einen geänderten Wert in das Modell zurück                                     |
| <code>void addTableModelListener(TableModelListener l)</code>              | Registriert einen Listener beim Modell, der über Änderungen am Modell unterrichtet wird |
| <code>void removeTableModelListener(TableModelListener l)</code>           | Deregistriert einen Listener vom Modell                                                 |

## 2.1.2 TabelModel: Beispiel (1)

- Es wird ein Modell entworfen, welches eine sehr *grosse Tabelle* repräsentiert (1000 \* 1000 Elemente)
- Es sollen nur die tatsächlich belegten Elemente gespeichert werden
- Für die Verwaltung wird eine *Hashtable* verwendet
- Als *Schlüssel* wird ein String mit den *Zellen-Koordinaten* verwendet
- Der Zugriff auf die Daten im Modell erfolgt dann über diesen Schlüssel
- Um das Modell nicht von Grund auf zu implementieren, wird die abstrakte Klasse *AbstractTableModel* verwendet

## 2.1.2 TabelModel: Beispiel (2)

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

public class JTableDemo2Data extends AbstractTableModel {
 private int size;
 private Hashtable data;

 //Konstruktor
 public JTableDemo2Data(int size) {
 this.size = size;
 this.data = new Hashtable();
 }

 //Methoden für das TableModel-Interface
 public int getRowCount() {
 return size;
 }

 public int getColumnCount() {
 return size;
 }
}
```



## 2.1.2 TabelModel: Beispiel (3)

```
public String getColumnName(int columnIndex) {
 return "C" + columnIndex;
}

public Class getColumnClass(int columnIndex) {
 return String.class;
}

public boolean isCellEditable(int rowIndex, int columnIndex) {
 return rowIndex < size && columnIndex < size;
}

public Object getValueAt(int rowIndex, int columnIndex) {
 String key = "[" + rowIndex + "," + columnIndex + "]";
 String value = (String)data.get(key);
 return value == null ? "-" : value;
}
```

## 2.1.2 TabelModel: Beispiel (4)

```
public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
 String key = "[" + rowIndex + "," + columnIndex + "]";
 String value = (String)aValue;
 if (value.length() <= 0) {
 data.remove(key);
 } else {
 data.put(key, value);
 }
}

//Zusätzliche Methoden
public void printData() {
 Enumeration e = data.keys();
 while (e.hasMoreElements()) {
 String key = (String)e.nextElement();
 System.out.println("At " + key + ": " + (String)data.get(key));
 }
}
```

## 2.1.2 TabelModel: Beispiel (5)

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

public class JTableDemo2 extends JFrame implements ActionListener {
 JTable table;
 JTableDemo2Data tableModel;

 public JTableDemo2() {
 super("JTableDemo2");
 tableModel = new JTableDemo2Data(1000);
 table = new JTable(tableModel, null);
 table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
 table.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
 table.setCellSelectionEnabled(true);

 Container cp = getContentPane();
 cp.add(new JScrollPane(table), "Center");

 JButton button = new JButton("Ausgabe");
 button.addActionListener(this);
 cp.add(button, "South");
 }
}
```



## 2.1.2 TabelModel: Beispiel (6)

```
public void actionPerformed(ActionEvent event) {
 tableModel.printData();
}

public static void main(String[] args) {
 JTableDemo2 frame = new JTableDemo2();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setLocation(100, 100);
 frame.setSize(320, 200);
 frame.setVisible(true);
}
}
```



## 2.2 Spaltenmodell

- *Spaltenmodell*
  - Verantwortlich für die Eigenschaften der Spalten
- Default *Spaltenmodell*
  - *DefaultTableColumnModel*
- Verwenden von eigenen *Spaltenmodellen*
  - Kontrolle von Spalten bezüglich:
    - Auswahl, Hinzufügen, Entfernen, Bewegen
    - Editoren, Darstellung (Renderer)
- Erzeugen von eigenen *Spaltenmodellen*
  - Durch Implementation des Interface *TableColumnModel*
  - *TableColumnModel* aus *javax.swing.table*

### 2.2.1 TableColumnModel

- Die wichtigsten Methoden des Interface

| Methode                                                                 | Beschreibung                                                                            |
|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <code>int getColumnCount()</code>                                       | Gibt Anzahl Spalten im Modell zurück                                                    |
| <code>void addColumn(TableColumn aColumn)</code>                        | Fügt eine Spalte am Ende des Spalten-Array hinzu                                        |
| <code>void removeColumn(TableColumn column)</code>                      | Entfernt die Spalte aus dem Spalten-Array                                               |
| <code>void moveColumn(int columnIndex, int newIndex)</code>             | Verschiebt Spalte an neue Position                                                      |
| <code>boolean getColumnSelectionAllowed()</code>                        | Gibt true zurück, wenn Spalte selektiert werden kann                                    |
| <code>void setColumnSelectionAllowed(boolean flag)</code>               | Setzt Selektierbarkeit der Spalte                                                       |
| <code>void addColumnModelListener(TableColumnModelListener x)</code>    | Registriert einen Listener beim Modell, der über Änderungen am Modell unterrichtet wird |
| <code>void removeColumnModelListener(TableColumnModelListener x)</code> | Deregistriert einen Listener vom Modell                                                 |

## 2.2.2 TableColumnModel: Beispiel (1)

- Es wird ein Modell entworfen, welches den Spalten eine Default Breite vorgibt
- Jede Spalte der Tabelle kann individuell angepasst werden
- Um das Modell nicht von Grund auf zu implementieren, wird die Klasse *DefaultTableColumnModel* verwendet
- Beim Aufruf des Konstruktors wird neben dem *Spaltenmodell* auch ein *Tabellenmodell* benötigt
  - Dieses leiten in einer lokale Klasse von *AbstractTableModel* ab

## 2.2.2 TableColumnModel: Beispiel (2)

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

public class JTableDemo3 extends JFrame implements JTableDemo3Data
{
 public JTableDemo3()
 {
 super("JTableDemo3");

 //Spaltenmodell erzeugen
 DefaultTableColumnModel cm = new DefaultTableColumnModel();
 for (int i = 0; i < COLHEADS.length; ++i)
 {
 TableColumn col = new TableColumn(i, i == 2 ? 150 : 60);
 col.setHeaderValue(COLHEADS[i]);
 cm.addColumn(col);
 }
 }
}
```

## 2.2.2 TableColumnModel: Beispiel (3)

```
//Tabellenmodell erzeugen
TableModel tm = new AbstractTableModel()
{
 public int getRowCount() return DATA.length;

 public int getColumnCount() return DATA[0].length;

 public Object getValueAt(int row, int column)
 {
 return DATA[row][column];
 }
};

//Tabelle erzeugen und ContentPane füllen
JTable table = new JTable(tm, cm);
Container cp = getContentPane();
cp.add(new JLabel("Titelthemen Javamagazin:"), "North");
cp.add(new JScrollPane(table), "Center");
}
```

## 2.2.2 TableColumnModel: Beispiel (4)

```
public static void main(String[] args)
{
 JTableDemo3 frame = new JTableDemo3();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setLocation(100, 100);
 frame.setSize(350, 200);
 frame.setVisible(true);
}
```



The screenshot shows a Java Swing window titled "JTableDemo3". Inside the window, there is a table with the title "Titelthemen Javamagazin:". The table has three columns: "Ausgabe", "Seite", and "Titelthema". The data is as follows:

| Ausgabe | Seite | Titelthema                     |
|---------|-------|--------------------------------|
| 1/2002  | 33    | Die Neulinge: JBuilder6 und... |
| 2/2002  | 36    | Enterprise Development         |
| 3/2002  | 34    | Web Frameworks                 |
| 4/2002  | 31    | Java Web Start und JNLP im...  |
| 5/2002  | 40    | Java Message Service: Einfü... |
| 6/2002  | 35    | Java meets Office              |
| 7/2002  | 35    | Java Datenbank API             |
| 8/2002  | 34    | Eigene JSP TagLibs entwick...  |

## 2.3 Renderer

- *Rendering*
  - Vorgang, wie *Zellen dargestellt* werden
- Default *Renderer*
  - *DefaultTableCellRenderer*
- Verwenden von eigenen *Renderer*
  - Kontrolle von *Darstellung* der Zellen  
• Schriftart, Farbe, Formatierung, Grafik
- Erzeugen von eigenen *Renderer*
  - Durch Implementation des Interface *TableCellRenderer*
  - *TableCellRenderer* aus *javax.swing.table*

### 2.3.1 TableCellRenderer

- Die einzige Methoden im Interface

| Methode                                                                                                                                                                                           | Beschreibung                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>public Component getTableCellRendererComponent(<br/>    JTable table,<br/>    Object value,<br/>    boolean isSelected,<br/>    boolean hasFocus,<br/>    int row,<br/>    int column)</pre> | Referenz auf JTable, welche den Renderer aufruft<br>Der Wert, der vom Renderer dargestellt werden soll<br>true, wenn die Zelle selektiert ist<br>true, wenn die Zelle den Fokus erhält<br>Zeilen-Index<br>Spalten-Index |

- Arbeitet als *Factory-Methode*
- Wird immer dann aufgerufen, wenn zur Darstellung einer Zelle ein *Renderer* benötigt wird

## 2.3.2 TableCellRenderer: Beispiel (1)

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

public class JTableDemo4CellRenderer implements TableCellRenderer
{
 private Color lightBlue = new Color(160, 160, 255);
 private Color darkBlue = new Color(64, 64, 128);

 public Component getTableCellRendererComponent(JTable table, Object value,
 boolean isSelected, boolean hasFocus, int row, int column)
 {
 //Label erzeugen
 JLabel label = new JLabel((String)value);
 label.setOpaque(true);
 Border b = BorderFactory.createEmptyBorder(1, 1, 1, 1);
 label.setBorder(b);
 label.setFont(table.getFont());
 label.setForeground(table.getForeground());
 label.setBackground(table.getBackground());
 }
}
```

## 2.3.2 TableCellRenderer: Beispiel(2)

```
if (hasFocus) {
 label.setBackground(darkBlue);
 label.setForeground(Color.white);
} else if (isSelected) {
 label.setBackground(lightBlue);
} else {
 //Angezeigte Spalte in Modellspalte umwandeln
 column = table.convertColumnIndexToModel(column);
 if (column == 1) {
 int numpages = Integer.parseInt((String)value);
 if (numpages < 30) {
 label.setBackground(Color.red);
 } else if (numpages < 40) {
 label.setBackground(Color.orange);
 } else {
 label.setBackground(Color.yellow);
 }
 }
}
return label;
}
```

## 2.3.2 TableCellRenderer: Beispiel(3)

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

public class JTableDemo4 extends JFrame implements JTableDemo4Data
{
 public JTableDemo4()
 {
 super("JTableDemo4");

 JTable table = new JTable(DATA, COLHEADS);
 table.setDefaultRenderer(Object.class, new JTableDemo4CellRenderer());
 Container cp = getContentPane();
 cp.add(new JLabel("Titelthemen Javamagazin:"), "North");
 cp.add(new JScrollPane(table), "Center");
 }
}
```

## 2.3.2 TableCellRenderer: Beispiel (4)

```
public static void main(String[] args)
{
 JTableDemo4 frame = new JTableDemo4();
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setLocation(100, 100);
 frame.setSize(350, 200);
 frame.setVisible(true);
}
```



The screenshot shows a Java Swing window titled "JTableDemo4". Inside the window, there is a table with the title "Titelthemen Javamagazin:". The table has three columns: "Ausgabe", "Seite", and "Titelthema". The data is as follows:

| Ausgabe | Seite | Titelthema          |
|---------|-------|---------------------|
| 1/2002  | 20    | Die Neulinge: JB... |
| 2/2002  | 36    | Enterprise Devel... |
| 3/2002  | 34    | Web Frameworks      |
| 4/2002  | 31    | Java Web Start u... |
| 5/2002  | 40    | Java Message S...   |
| 6/2002  | 35    | Java meets Office   |
| 7/2002  | 25    | Java Datenbank ...  |
| 8/2002  | 34    | Eigene JSP TagL...  |



## Zusammenfassung

- Eines der wichtigsten Dialogelemente in anspruchsvollen Programmen ist die *Tabelle*
- Mit JTable haben sie eine *leistungsfähige Klasse* für die *tabellarische* Darstellung von Daten
- Mit dem Tabellenmodel *TableModel* können die Daten einer *JTable* einfach verwaltet werden
- Mit dem Spaltenmodel *TableColumnModel* können die Eigenschaften der Spalten kontrolliert werden
- Mit der Klasse *TableCellRenderer* kann das Erscheinungsbild von Zellen verändert werden

## @ Übungsaufgabe U07.1

- Gehen Sie im Internet auf die Seite

<http://java.sun.com/docs/books/tutorial/uiswing/components/table.html>

- Lesen Sie das Kapitel "How to Use Tables"
- Laden Sie die Beispiele herunter und spielen Sie mit den Möglichkeiten von Tabellen
- Versuchen Sie für den Address Manager eine Tabelleansicht mit Adressen zu implementieren