

# JTable

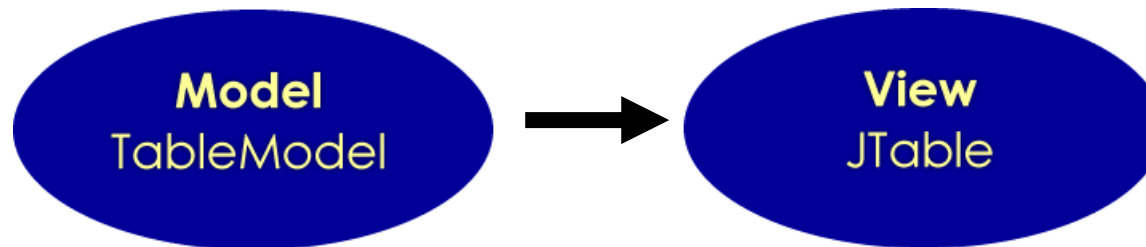
# JTable

- Darstellen von **2-dimensionalen** Daten
- Daten können **editiert** werden

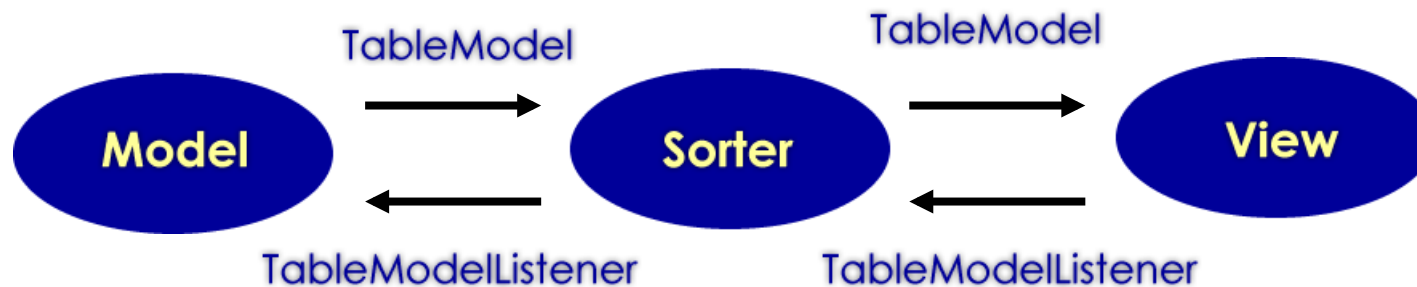
First Name	Last Name	Favorite Color	Favorite Sport	Favorite Number	Favorite Food
Mike	Albers	0, 255, 0	Soccer	44	
Mark	Andrews	255, 0, 0	Baseball	2	
Tom	Ball	0, 0, 255	Football	99	

# Model - View - Controller

- Swing-Eigenheiten müssen berücksichtigt werden (**Container**, **MVC-Konzept**)



- Sortierung über **TableSorter**



# Abstrakte Modelle

- **TableModel**  
Aufgabe: Daten zu speichern und zu verarbeiten.  
Entkopplung View <-> Modell gemäß MVC-Pattern.  
Eigenes **Tabellenmodell** mit Hilfe von **AbstractTableModel**
- **TableColumnModel**  
Repräsentiert alle Spalten der JTable. Default-Spaltenmodell: **DefaultTableColumnModel**
- **ListSelectionModel**  
Registriert Selektion von Zellen und Zeilen der JTable.

## „Einfache“ Konstruktoren

- `public JTable(Object[][]rowData, Object[]columnNames)`  
darzustellende Daten und Spaltenköpfe werden in Form eines **Arrays** übergeben
- `public JTable(Vector rowData, Vector columnNames)`  
darzustellende Daten und Spaltenköpfe werden in Form eines **Vektors** übergeben

## Einfaches Beispiel

```
String rowData[ ][ ] = {  
    { "Japan", "245" }, { "USA", "240" }, { "Italien", "220" },  
    { "Spanien", "217" }, { "Türkei", "215" }, { "England", "214" },  
    { "Frankreich", "190" }, { "Griechenland", "185" },  
    { "Deutschland", "180" }, { "Portugal", "170" } };
```

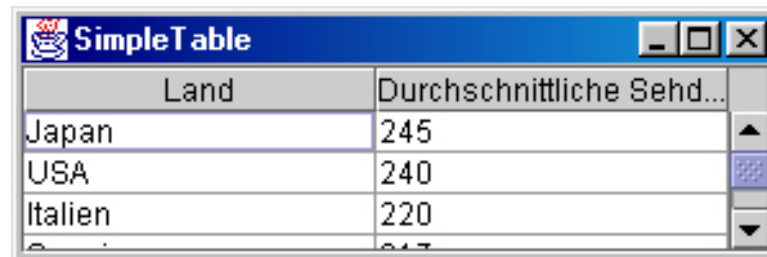
```
String columnNames[ ] = {  
    "Land", "Durchschnittliche Sehdauer pro Tag in Minuten"};
```

```
//Tabelle erzeugen
```

```
JTable table = new JTable( rowData, columnNames );
```

```
//Tabelle dem Container hinzufügen
```

```
fenster.getContentPane( ). add( new JScrollPane( table ) );
```



Land	Durchschnittliche Sehdauer pro Tag in Minuten
Japan	245
USA	240
Italien	220

# Konstrukturen mitTableModel

- `public JTable(TableModel dm)`  
explizite Übergabe des Tabel-Modells an den Konstruktor
- `public JTable(TableModel dm, TableColumnModel cm)`  
explizite Übergabe des Table- und des Spalten-Modells an den Konstruktor
- `public JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)`  
allgemeinste Möglichkeit, eine Tabelle zu erstellen,  
explizite Übergabe aller drei Modelle an den Konstruktor

## ohne TableModel

### „Einfache“ Konstruktoren - ohne TableModel:

- Keine Unterscheidung der Datentypen – alle Daten werden als **String** angezeigt.
- Alle Zellen automatisch editierbar.
- alle Daten müssen in einem Array oder Vector übergeben werden - nicht möglich, Daten direkt aus einer Datenbank zu holen.



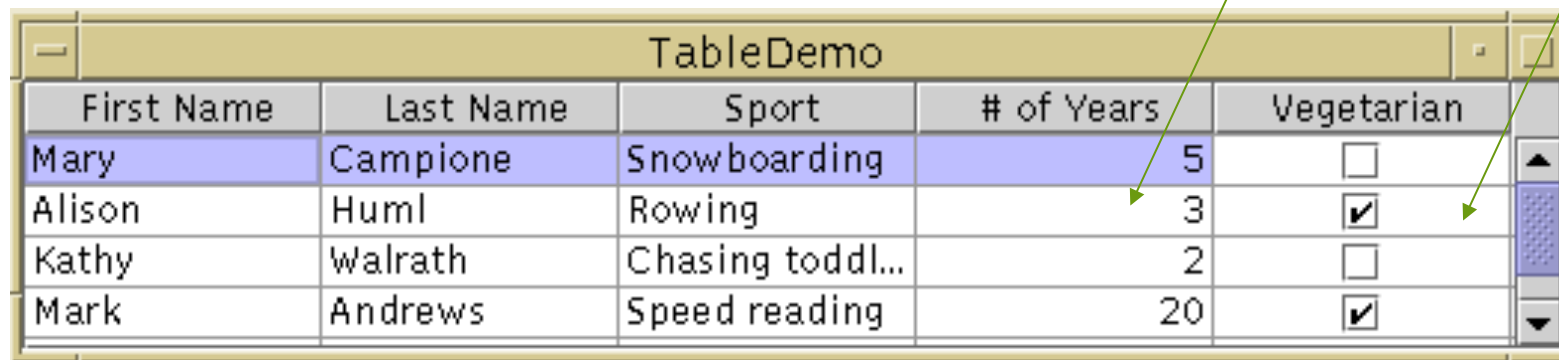
First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Chasing toddl...	2	false
Mark	Andrews	Speed reading	20	true



## mit TableModel

Bei Implementierung eines **AbstractTableModel**:

- **Einschränkung der Editierbarkeit** möglich
- **Datentypabhängige** Darstellung der Daten
  - hilft Datentypen in einem besten Format darzustellen
  - Erkennung von **num. Zahlen** -> rechtsbündige Darstellung
  - Datentyp **boolean** wird mit einer Checkbox dargestellt



First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Chasing toddl...	2	<input type="checkbox"/>
Mark	Andrews	Speed reading	20	<input checked="" type="checkbox"/>

# Konfiguration

## Ränder

**public void setRowHeight( int newHeight)**

Ändert die **Zeilenhöhe** für alle Zeilen der Tabelle

**public void setRowMargin( int rowMargin)**

Es wird der obere und untere **freibleidende** Platz jeder Zeile festgelegt

## Gitternetzlinien

**public void setShowGrid( boolean b);**

Legt fest, ob die **Gitternetzlinien** sichtbar oder unsichtbar sind

**public void setShowHorizontalLines( boolean b);**

Blendet die **horizontalen Gitternetzlinien** ein oder aus

**public void setShowVerticalLines( boolean b);**

Blendet die **vertikalen Gitternetzlinien** ein oder aus

# Konfiguration

## Farben

**public void setGridColor( Color newColor);**

Ändert die Farbe der **Gitternetzlinien**

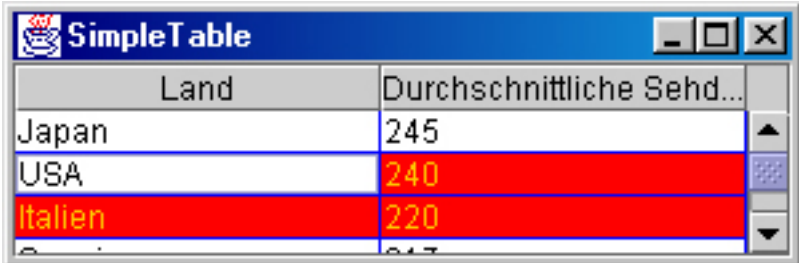
**public void setSelectionForeground( Color colForeground);**

Ändert die **Textfarbe** des *selektierten* Textes

**public void setSelectionBackground( Color colBackground);**

Ändert die **Hintergrundfarbe** der *selektierten* Zellen

```
table.setGridColor(Color.blue);  
table.setSelectionForeground(Color.orange);  
table.setSelectionBackground(Color.red);
```



Land	Durchschnittliche Sehd...
Japan	245
USA	240
Italien	220

# Konfiguration

Verhalten bei **Zellbreitenänderung** durch Benutzer

**public void setAutoResizeMode(int Mode);**

Bestimmt das Verhalten der Tabelle, nachdem die **Breite** einer einzelnen

JTable.KONSTANTE	int	Bedeutung
AUTO_RESIZE_OFF	0	Es erfolgt keine automatische Größenanpassung der übrigen Spalten. Wurde die Tabelle in JScrollPane verpackt, bekommt sie nötigenfalls einen horizontalen Schieberegler.
AUTO_RESIZE_NEXT_COLUMN	1	Die rechts neben der modifizierten Spalte liegende Spalte wird zum Größenausgleich verwendet.
AUTO_RESIZE_SUBSEQUENT_COLUMN	2	Die Größenänderung wird gleichmäßig auf alle nachfolgenden Spalten verteilt.
AUTO_RESIZE_LAST_COLUMN	3	Die letzte Spalte wird zum Größenausgleich verwendet.
AUTO_RESIZE_ALL_COLUMNS	4	Die Größenänderung wird auf alle Spalten der Tabelle verteilt.

# Konfiguration

## Zellenbreite einstellen über das Spaltenmodell

// mögliche Einstellungen

**setMinWidth** // Mindestgröße der Zellenbreite

**setMaxWidth** // Maximale Größe der Zellenbreite

**setPreferredWidth** // bevorzugte Zellenbreite

```
meineTabelle.getColumnModel().getColumn(0).setMinWidth(25);
```

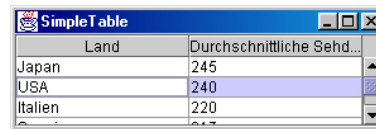
```
meineTabelle.getColumnModel().getColumn(0).setMaxWidth(35);
```

```
meineTabelle.getColumnModel().getColumn(1).setPreferredWidth(175);
```

# Zeilen-/Spalten-Selektion

**public void setRowSelectionAllowed( boolean flag)**

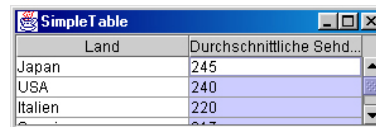
Ermöglicht das **zeilenweise** Selektieren



Land	Durchschnittliche Sehd...
Japan	245
USA	240
Italien	220

**public void setColumnSelectionAllowed( boolean flag)**

Ermöglicht das **spaltenweise** Selektieren



Land	Durchschnittliche Sehd...
Japan	245
USA	240
Italien	220

*Sind beide auf false gesetzt, wird zellenweise selektiert. Der Defaultwert ist zeilenweises Selektieren.*

**public void setCellSelectionEnabled( boolean flag)**

**true** = ermöglicht das Selektieren einer **Gruppe einzelner Zellen**

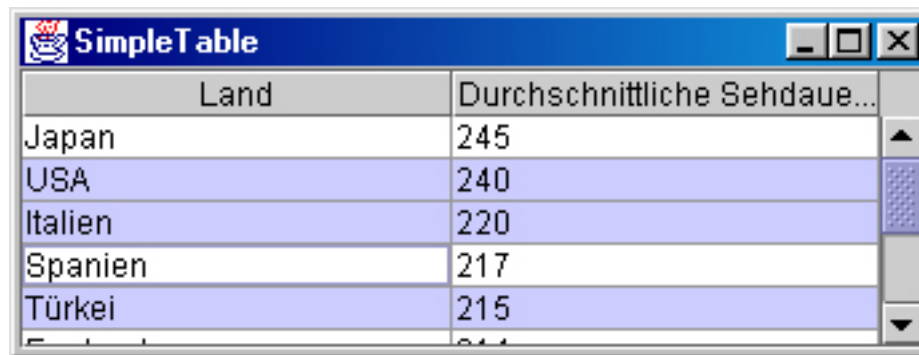
**false** = es kann nur **eine Zelle** selektiert werden

# Selektionsmodi

**public void setSelectionMode( int selectionMode )**

Legt fest, ob man nur einzelne, zusammenhängende oder mehrere Elemente

JTable.KONSTANTE	int	Bedeutung
SINGLE_SELECTION	0	Nur einzelne Elemente
SINGLE_INTERVAL_SELECTION	1	Einen zusammenhängenden Bereich
MULTIPLE_INTERVAL_SELECTION	2	Mehrere Elemente, unabhängig davon ob sie einzeln oder zusammenhängend sind



Land	Durchschnittliche Sehdaue...
Japan	245
USA	240
Italien	220
Spanien	217
Türkei	215
F...	211

```
table.setSelectionMode( ListSelektionModel. MULTIPLE_INTERVAL_SELECTION );
```

# Abfragen der Selektion

## Bei einfacher Auswahl

**public int getSelectedRow()**

Liefert die selektierte Zeile

**public int getSelectedCol()**

Liefert die selektierte Spalte

## Bei mehrfacher Auswahl

**public int[ ] getSelectedRows()**

Liefert bei mehrfacher Auswahl ein Array of Integer mit den selektierten Zeilen

**public int[ ] getSelectedColumns()**

Liefert bei mehrfacher Auswahl ein Array of Integer mit den selektierten Zeilen

**Sollte nichts selektiert sein, so ist der Rückgabewert -1 oder ein leeres Array.**



# Editieren

## Editieren durch den Benutzer

**public boolean isEditing();**

Liefert true, wenn die Tabelle sich im Editierzustand befindet

**public int getEditingRow();**

Liefert die Zeile, die gerade editiert wird (kein Editierzustand, Rückgabewert -1)

**public int getEditingColumn();**

Liefert die Spalte, die gerade editiert wird (kein Editierzustand, Rückgabewert -1)

## Editieren durch das Programm

**public boolean editCellAt(int row, int column)**

Leitet den Editierzustand der Tabelle an der Zelle row,column ein

## AbstractTableModel

- Ein eigenes Tabellenmodell muss das Interface **TableModel** aus dem Paket **javax.swing.table** implementieren.
- Die Klasse **AbstractTableModel** implementiert die meisten Methoden des **TableModel**, und kann als Vorlage dienen.

# Interface TableModel

Das Interface **TableModel** definiert folgende Methoden:

//liefert die Anzahl der Zeilen und Spalten

public int getRowCount()

public int getColumnCount()

//liefert die Spaltenüberschrift

public String getColumnName(int columnIndex)

//liefert den Typ der Elemente einer Spalte

public Class getColumnClass(int columnIndex)

//darf eine bestimmte Zelle editiert werden ja / nein

public boolean isCellEditable(int rowIndex, int columnIndex)

//Wert einer bestimmten Zelle abfragen / zurückschreiben

public Object getValueAt(int rowIndex, int columnIndex)

public void setValueAt(Object aValue, int rowIndex, int columnIndex)

//TableModelListener registriert bzw. deregistriert

public void addTableModelListener(TableModelListener l)

public void removeTableModelListener(TableModelListener l)

## Erweiterung des AbstractTableModel:

```
import javax.swing.table.AbstractTableModel;

public class DatenModell extends AbstractTableModel {

    public int getColumnCount() {
        int colCount = ...; return colCount;
    }

    public int getRowCount() {
        int rowCount = ... ; return rowCount;
    }

    public String getColumnName( int col){
        return ...;
    }

    public Object getValueAt( int rowIndex, int columnIndex) {
        String feldwert = ...; return feldwert;
    }
}
```

# AbstractTableModel

## Ermitteln der Tabellendimensionen

**public int getRowCount()**

Liefert die Anzahl der Zeilen

**public int getColumnCount()**

Liefert die Anzahl der Spalten

## Auf den Inhalt zugreifen

**public Object getValueAt(int row, int column)**

Liefert den Inhalt einer Zelle

**public void setValueAt(Object value, int row, int column)**

Setzt den Wert einer Zelle mit value

### HINWEIS!

Die Indices **row** und **column** beginnen bei 0. Sollte eine Überschreitung der Datengrenzen erfolgen, so wird eine

**ArrayIndexOutOfBoundsException** ausgelöst.

## Beispiel:

```
public class DatenModell extends AbstractTableModel {  
  
    String data[ ][ ] = {  
        {"1", "Hugo Klein", "19 Sept 2005", "ja"},  
        {"2", "Gerda Gross", "18 Aug 2005", "ja"},  
        {"", "", "", ""} };  
    String feldname[ ] = { "Nr", "Name", "Datum",  
        "benachrichtigt?" };  
  
    public int getColumnCount() { return feldname.length; }  
  
    public int getRowCount() { return data.length; }  
  
    public String getColumnName( int col){ return feldname[ col ]; }  
    public Object getValueAt( int rowIndex, int columnIndex ) {  
        return data[rowIndex][columnIndex];  
    }  
}
```

## Zellen editierbar setzen

```
public class DatenModell extends AbstractTableModel {  
    ...  
  
    // row = Zeile, die editierbar gesetzt wird, beginnt bei 0  
    // col = Spalte, die editierbar gesetzt wird, beginnt bei 0  
    public boolean isCellEditable( int row, int col) {  
        if( col == 0 ) // nur 1. Spalte ..  
            return true; // ..ist editierbar  
        else  
            return false;  
    }  
    ...  
}
```

# AbstractTableModel

Für einige Methoden sind Standard-Implementierungen vorgegeben:

**getColumnCount(), getRowCount(), getValueAt()**  
**müssen** implementiert werden

**isCellEditable()** liefert per Default: *false*

**setValueAt()** ist per Default: *leer*  
**muss** bei *isCellEditable = true* überschrieben werden,  
sonst keine Darstellung des neuen Wertes möglich.



# TabellenModell

## Anbinden des Tabellenmodells an die Tabelle

// über den Konstruktor

**DatenModell meinDatenModell = new DatenModell();**

**JTable meineTabelle = new JTable( meinDatenModell );**

//nachträgliches Anbinden

**public void setModel( meinDatenModell )**

//Abfragen des Tabellenmodells

**public TableModel getModel()**

# TableSorter

Aufruf des JTable-Konstruktors mit Übergabe des TableSorters (inkl. Tabellenmodell):

- **DatenModell** **meinDatenModell** = new DatenModell();
- **TableSorter** **meinSorter**  
= new TableSorter( **meinDatenModell** );
- **JTable** **meineTabelle**  
= new JTable( **meinSorter** );
- **meinSorter.setTableHeader**(  
    **meineTabelle.getTableHeader()** );

## Listener

Sollen Änderungen an der **Selektion** oder dem **Zellenwert** erkannt werden, müssen Listener registriert werden.

**Selektion:** zwei **ListSelectionListener** registrieren

- auf Selektionsmodell (Tabelle -> nur Zeilen)
- auf Selektionsmodell (Spalte)

Bei jeder Änderung der Selektion wird **valueChanged** aufgerufen.

**Daten:** durch Aufruf von **addTableModelListener**  
wird ein **TableModelListener** registriert

Bei jeder Änderung des Modells wird **tableChanged** aufgerufen.

## ListSelectionListener

- **Klasse DefaultListSelectionModel** stellt Standard-Implementierung dar, kann ohne weitere Ableitung verwendet werden
- Implementierung des **Interface ListSelectionListener**

//Listener anbinden

```
ListSelectionModel lsm = table.getSelectionModel();
```

```
lsm.addListSelectionListener( new ListSelectionAdapter());
```

## ListSelectionListener

```
class ListSelectionAdapter implements
    ListSelectionListener {
    public void valueChanged( ListSelectionEvent ev){

        if( ev.getValueIsAdjusting()) return;

        ListSelectionModel lsm = (ListSelectionModel) ev.getSource();

        if( lsm.isSelectionEmpty() ){
            System.out.println("Zeilen nicht selektiert.");
        }else{
            int zeilenIndex = lsm.getMinSelectionIndex();
            int row = table.getSelectedRow();
            int col = table.getSelectedColumn();
        }
    } //end of valueChanged
} //end of inner class
```

# TableModelListener

//Table-Listener anbinden

```
meinDatenModell.addTableModelListener (new  
    TableModelListener () {  
        public void tableChanged(TableModelEvent ev){  
            ...  
        }  
    });
```