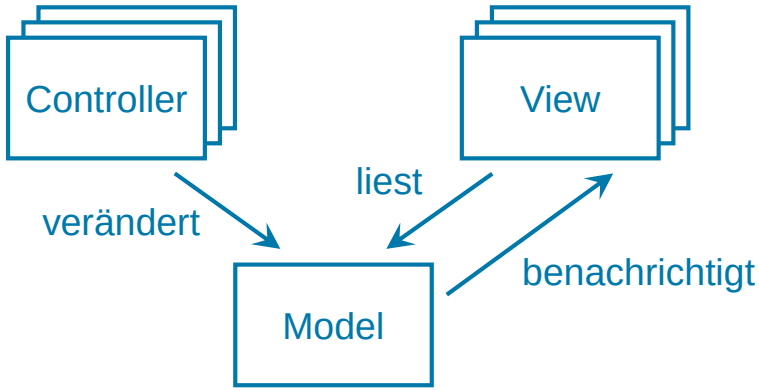
PRAKTIKUM SOFTWAREENTWICKLUNG 2

MVC

MVC

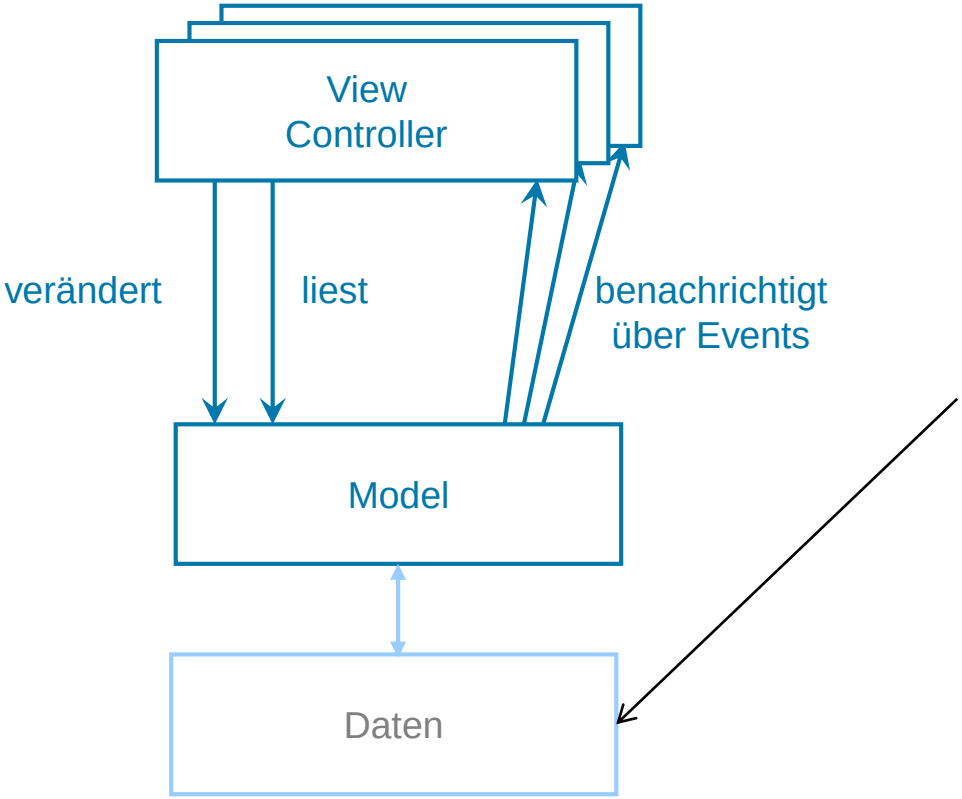
- Entkoppelung von Model, View und Controller
  - Kombiniertes Muster
  - Stammt aus Smalltalk
  - Flexibilität
  - Wiederverwendbarkeit



- Model
  - Daten, die das Programm verwaltet
  - Sorgt dafür, dass alle Views nachgeführt werden
- View
  - Darstellung der Daten
  - Zu jeder View gehört ein Controller
- Controller
  - Interpretation von Tastatureingaben und Mausklicks

MVC in Swing

- Keine Trennung von View und Controller
- Model bietet
  - Eine abstrakte Schnittstelle für Datenzugriffe
  - Verwendet Ereignismechanismus

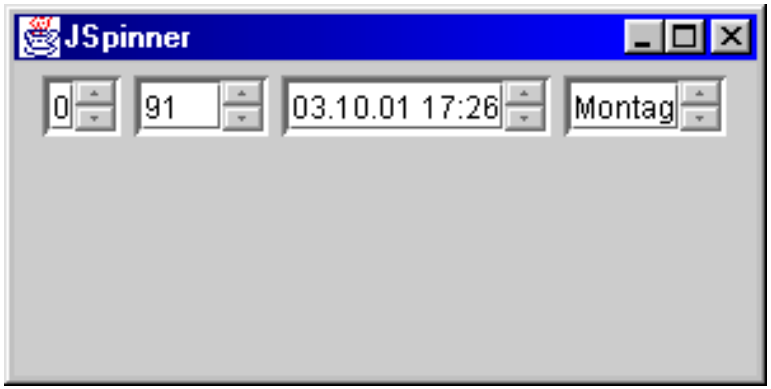


MVC

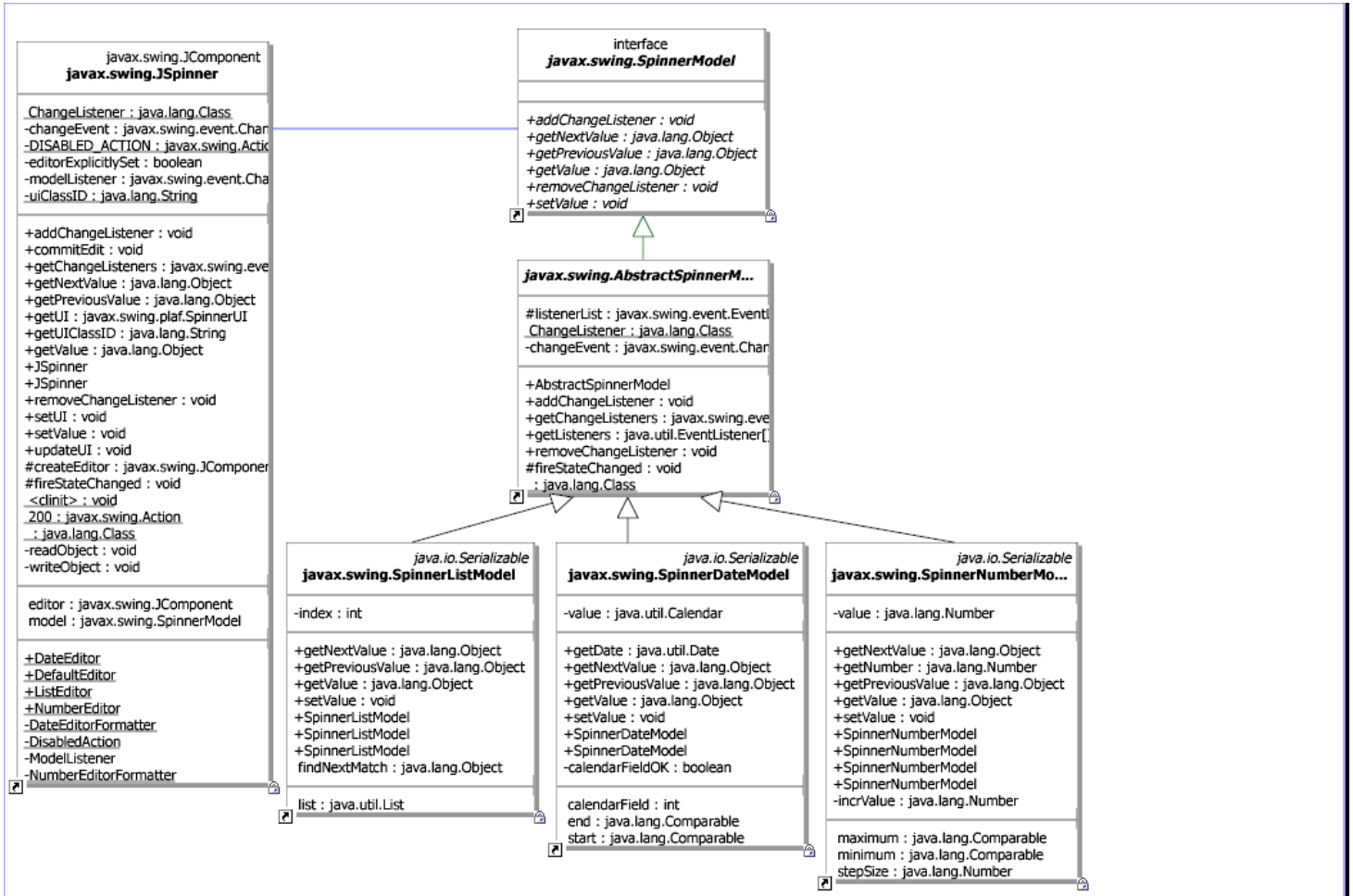
- JSpinner
- JList
- JTable
- JTree

JSpinner

- Für vordefinierte, sortierte Menge von Werten
- Eingabe
  - Textuell
  - Auf- oder absteigendes Durchlaufen der Werte
- JSpinner implementiert View und Controller
- SpinnerModel stellt eine abstrakte Definition für das Modell bereit



JSpinner Klassendiagramm



Interface SpinnerModel

- Lesen und Schreiben des aktuellen Werts
- Zugriff auf Nachfolger und Vorgänger
- Quelle von ChangeEvents

```
public interface SpinnerModel {
    Object getValue();
    void setValue(Object value);
    Object getNextValue();
    Object getPreviousValue();
    void addChangeListener(ChangeListener l);
    void removeChangeListener(ChangeListener l);
}
```

```
public interface ChangeListener extends EventListener {
    void stateChanged(ChangeEvent e);
}
```

```
public class ChangeEvent extends EventObject {
    public ChangeEvent(Object source) {
        super(source);
    }
}
```

Beispiel Verwendung JSpinner

```
public class JSpinnerTest extends JFrame {
    private static final String[] WDAYS = {
         "Montag",  "Dienstag", "Mittwoch", "Donnerstag",
         "Freitag", "Samstag",  "Sonntag"
      };
    JSpinner spinner1, spinner2, spinner3, spinner4;

    public JSpinnerTest() {
      //Default-Spinner für Ganzzahlen
      spinner1 = new JSpinner();
      //Spinner für Vielfache von 7 beginnend mit 49 und bis 126, intial 91
      spinner2 = new JSpinner(new SpinnerNumberModel(91, 49, 126, 7));
      //Spinner für Datum/Uhrzeit
      spinner3 = new JSpinner(new SpinnerDateModel());
      //Spinner für Wochentage
      spinner4 = new JSpinner(new SpinnerListModel(WDAYS));
      //Anfügen eines ChangeListeners an spinner4
      spinner4.getModel().addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
         System.out.println("Jetzt ist " +
                            spinner4.getModel().getValue());
      }
    });
    ...
}
```

MVC

- JSpinner
- JList
- JTable
- JTree

JList

- JList erlaubt die bequeme Ausgabe von Listen von beliebigen Objekten
- JList unterstützt
  - Listen von beliebigen Objekten
  - Lange Listen
  - Listen die sich dynamisch aufbauen (nicht explizit vorhanden sein müssen)
  - Selektion von einzelnen und mehreren Elementen
  - Ausgabe der Elemente in unterschiedlicher Art
- JList arbeitet mit ListModel
- JList arbeitet mit Renderern für die Ausgabe

Interface ListModel

- Zugriff auf die Elemente
- Bestimmung der aktuellen Anzahl
- Registrieren von ListDataChangeListener

```
public interface ListModel<E> {
    int getSize();
    E getElementAt(int index);
    void addListDataListener(ListDataListener l);
    void removeListDataListener(ListDataListener l);
}
```

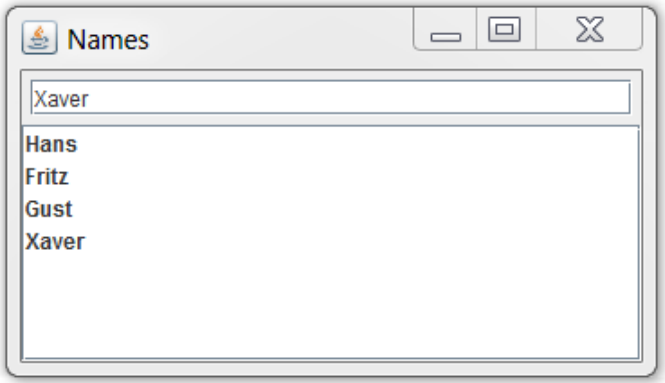
```
public interface ListDataListener extends EventListener {
    void intervalAdded(ListDataEvent e);
    void intervalRemoved(ListDataEvent e);
    void contentsChanged(ListDataEvent e);
}
```

```
public class ListDataEvent extends EventObject {
    ...
    public int getType() { return type; }
    public int getIndex0() { return index0; }
    public int getIndex1() { return index1; }
}
```

DefaultListModel

- Default-Implementierung von ListModel
  - mit Methode addElement
- Beispiel: Liste von Namen durch JTextField angefügt

```
DefaultListModel<String> nameModel = new DefaultListModel<String>();
JList<String> nameList = new JList<String>(nameModel);

JTextField nameTf.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        nameModel.addElement(nameTf.getText());
    }
});
```

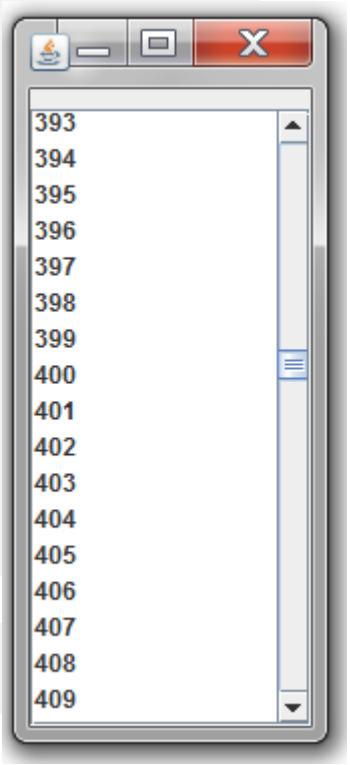


Spezielles ListModel

- ListModel abgeleitet von AbstractListModel
  - Überschreiben von getElementAt und getSize
- Beispiel: IntListModel erzeugt dynamisch Integer-Elemente

```
class IntListModel extends AbstractListModel<Integer> {

    private final int n;

    public IntListModel(int n) {
        this.n = n;
    }

    public int getSize() {
        return n;
    }

    public Integer getElementAt(int i) {
        return new Integer(i);
    }
}
```

```
IntListModel intsModel = new IntListModel(1000);
Jlist<Integer> intList = new JList<>(intsModel);
```



Selektion

- Bei `JList` kann man Index des selektierten Elements abfragen

```
int idx = list.getSelectedIndex();
```

- `JList` wirft `ListSelectionEvent`

```
public interface ListSelectionListener extends EventListener {
   void valueChanged(ListSelectionEvent e);
}
```

```
public class ListSelectionEvent extends EventObject {
    public ListSelectionEvent(Object source, int firstIndex,
          int lastIndex, boolean isAdjusting) { … }
    public int getFirstIndex() { … }
    public int getLastIndex() { … }
    public boolean getValueIsAdjusting() { … }
    ...
}
```

```
list.addListSelectionListener(e -> { … });
```

- Es gibt unterschiedliche Selektionsmodi, die man über die JList-Methode

  `void setSelectionMode(int mode)`

  setzen kann; mögliche Modi sind (Konstante des Interfaces `ListSelectionModel`)
  - ☐ `SINGLE_SELECTION`
  - ☐ `SINGLE_INTERVAL_SELECTION`
  - ☐ `MULTIPLE_INTERVAL_SELECTION`

Renderer für Listenelemente

- Man hat die Möglichkeit die Ausgabe der Elemente zu bestimmen
- Interface ListCellRenderer definiert Methode
  - Component getListCellRenderer(...)

  welche für eine Liste und einen Wert an einer Position und Information, ob Zelle selektiert und Komponente den Fokus hat, eine Component zum Rendern liefert

```
public interface ListCellRenderer<E> {
    Component getListCellRendererComponent(
                    JList<? extends E> list,
                    E value,
                    int index,
                    boolean isSelected,
                    boolean cellHasFocus);
}
```

Achtung: `ListCellRenderer` rendert selbst nicht, sondern liefert eine `Component`!!

- Methode

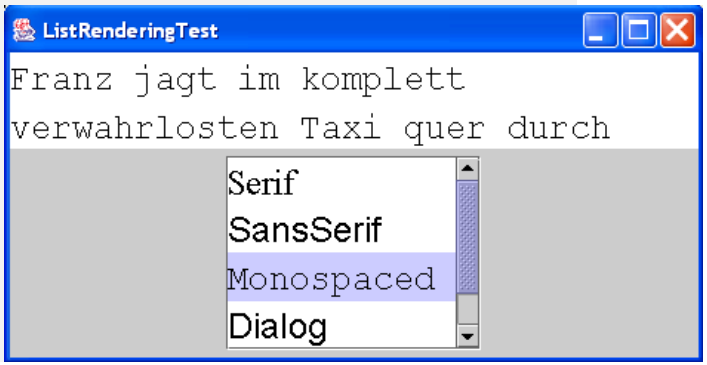
  `void setCellRenderer(ListCellRenderer<E> r)`

  erlaubt die Installation eines besonderen ListCellRenderers.

Beispiel ListCellRenderer

- Dieser Renderer rendert Font-Objekte, indem der Name des Fonts im entsprechenden Font ausgegeben wird

```
class FontCellRenderer implements ListCellRenderer<Font> {
   private JLabel label;

   public FontCellRenderer() {
      label = new JLabel();
      label.setOpaque(true); // must do this for background to show up
   }

   public Component getListCellRendererComponent(
           JList<? extends Font> list, Font font, int index,
           boolean isSelected, boolean cellHasFocus) {
      if (isSelected) {
         label.setForeground(list.getSelectionForeground());
         label.setBackground(list.getSelectionBackground());
      } else {
         label.setForeground(list.getForeground());
         label.setBackground(list.getBackground());
      }
      label.setFont(font);
      label.setText(font.getFamily());
      return label;
   }
}
```



MVC

- JSpinner
- JList
- JTable
- JTree

JTable

- JTable erlaubt die Ausgabe und die Bearbeitung von Tabellen
- Modell und Art der Bearbeitung ist analog zu JList
  - TableModel
  - Selection, SelectionEvents und SelectionModes
  - TableCellRenderer
- zusätzlich können Zellen in Tabellen editierbar sein; dazu wird ein
  - TableCellEditor

  herangezogen

Interface TableModel

- Anzahl der Zeilen und Spalten
- Name und Typ für die Spalte
- Prüfen, ob Zelle editierbar
- Lesen und Schreiben der Zellen
- Registrieren von TableModelListener

```
public interface TableModel {
    public int getRowCount();
    public int getColumnCount();
    public String getColumnName(int columnIndex);
    public Class getColumnClass(int columnIndex);
    public boolean isCellEditable(int rowIndex, int columnIndex);
    public Object getValueAt(int rowIndex, int columnIndex);
    public void setValueAt(Object aValue, int rowIndex, int columnIndex);
    public void addTableModelListener(TableModelListener l);
    public void removeTableModelListener(TableModelListener l);
}
```

```
public interface TableModelListener
    extends java.util.EventListener {
  public void tableChanged(
                TableModelEvent e);
}
```

```
public class TableModelEvent
    extends java.util.EventObject{
  public int getFirstRow() { … };
  public int getLastRow() { … };
  public int getColumn() { … };
  public int getType() { … }
}
```

TableModel (1/2)

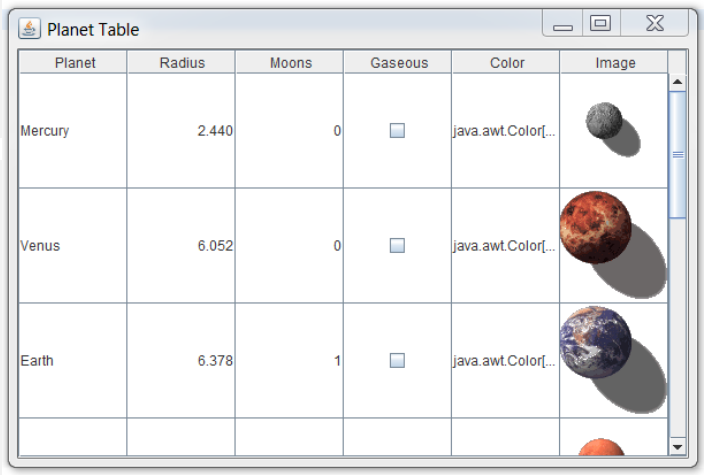
- Basisklasse AbstractTableModel
  - EventHandling
  - Default-Implementierungen
- Beispiel: Tabelle mit Planetendaten (siehe Horstman and Cornell: Core Java II)

```
class PlanetTableModel extends AbstractTableModel {

   private Object[][] cells = {
         { "Mercury", 2440.0, 0, false, Color.YELLOW, new ImageIcon("Mercury.gif") },
         { "Venus", 6052.0, 0, false, Color.YELLOW, new ImageIcon("Venus.gif") },
         { "Earth", 6378.0, 1, false, Color.BLUE, new ImageIcon("Earth.gif") },
         { "Mars", 3397.0, 2, false, Color.RED, new ImageIcon("Mars.gif") },
         { "Jupiter", 71492.0, 16, true, Color.ORANGE, new ImageIcon("Jupiter.gif") },
         { "Saturn", 60268.0, 18, true, Color.ORANGE, new ImageIcon("Saturn.gif") },
         { "Uranus", 25559.0, 17, true, Color.BLUE, new ImageIcon("Uranus.gif") },
         { "Neptune", 24766.0, 8, true, Color.BLUE, new ImageIcon("Neptune.gif") },
         { "Pluto", 1137.0, 1, false, Color.BLACK, new ImageIcon("Pluto.gif") } };

   private String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color", "Image" };

   public String getColumnName(int c) {
      return columnNames[c];
   }

   public Class<?> getColumnClass(int c) {
      return cells[0][c].getClass();
   }

   public int getColumnCount() {
      return cells[0].length;
   }
...
```

TableModel (2/2)

```
...
   public int getRowCount() {
      return cells.length;
   }

   public Object getValueAt(int r, int c) {
      return cells[r][c];
   }

   public void setValueAt(Object obj, int r, int c) {
      cells[r][c] = obj;
   }

   public boolean isCellEditable(int r, int c) {
      return c == PLANET_COLUMN || c == MOONS_COLUMN || c == GASEOUS_COLUMN
            || c == COLOR_COLUMN;
   }

   public static final int PLANET_COLUMN = 0;
   public static final int MOONS_COLUMN = 2;
   public static final int GASEOUS_COLUMN = 3;
   public static final int COLOR_COLUMN = 4;

}
```

```
public class PlanetTableFrame extends JFrame {

   public PlanetTableFrame() {
      setTitle("Planet Table");
      TableModel model = new PlanetTableModel();
      JTable table = new JTable(model);
      add(new JScrollPane(table), BorderLayout.CENTER);
   }
}
```



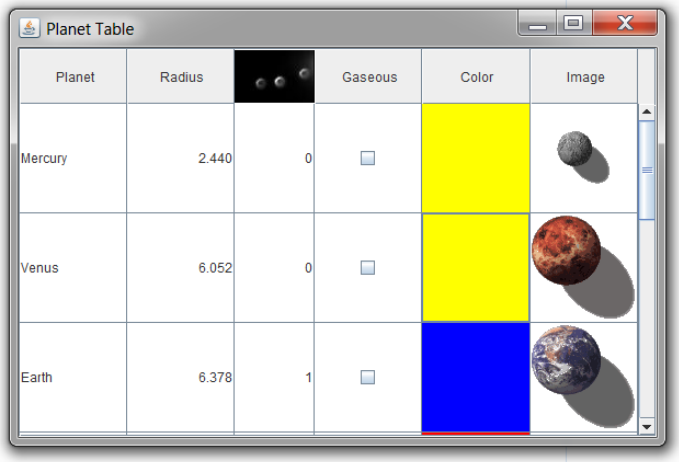
TableCellRenderer

- TableCellRenderer werden für die Darstellung der Zellen verwendet

```
public interface TableCellRenderer {
   Component getTableCellRendererComponent(JTable table, Object value,
         boolean isSelected, boolean hasFocus, int row, int column);
}
```

- Beispiel ColorTableCellRenderer: Color-Werte werden mit JPanel mit entsprechenden Hintergrund dargestellt

```
class ColorTableCellRenderer extends JPanel implements TableCellRenderer {
   public Component getTableCellRendererComponent(JTable table, Object value,
         boolean isSelected, boolean hasFocus, int row, int column) {
      setBackground((Color) value);
      if (hasFocus) {
         setBorder(UIManager.getBorder("Table.focusCellHighlightBorder"));
      } else {
         setBorder(null);
      }
      return this;
   }
}
```



Installation eines TableCellRenderers

- Man kann TableCellRenderer auf 2 Arten installieren
- Variante 1: TableCellRenderer für alle Werte einer bestimten Klasse
  - Festlegen einer Klasse für eine Spalte im TableModel

```
class PlanetTableModel extends AbstractTableModel {
   public Class getColumnClass(int c) {
      return cells[0][c].getClass();
   }
```

  - und Bestimmen eines Renderers für die Klasse im JTable

```
table.setDefaultRenderer(Color.class, new ColorTableCellRenderer());
```

- Variante 2: TableCellRenderer für eine TableColumn
  - Zugriff auf das TableColumn-Objekt mittels getColumn von JTable
  - Setzen des Renderers bei dem TableColumn-Objekt

```
TableColumn column = table.getColumn(colName);
column.setCellRenderer(new ColorTableCellRenderer());
```

TableColumnModel

Steuerung der Spalten über TableColumnModel

- Zugriff mit getColumnModel bei Table

```
TableColumnModel columnModel = table.getColumnModel();
```

- von TableColumnModel Zugriff auf Column-Info

```
TableColumn moonColumn = columnModel.getColumn(PlanetTableModel.MOONS_COLUMN);
```

- Steuerung des Aussehens der Spalte

```
moonColumn.setHeaderRenderer(table.getDefaultRenderer(ImageIcon.class));
moonColumn.setHeaderValue(new ImageIcon("Moons.gif"));
moonColumn.setResizable(true);
moonColumn.setPreferredWidth(50);
```

- Editor für Spalte

```
moonColumn.setCellEditor(new DefaultCellEditor(moonCombo));
```

TableCellEditor

- Damit Zelle editierbar ist, muss im TableModel die Methode isEditable true liefern

```
class PlanetTableModel extends AbstractTableModel {
   public boolean isCellEditable(int r, int c) {
      return c == NAME_COLUMN || c == MOON_COLUMN
         || c == GASEOUS_COLUMN || c == COLOR_COLUMN;
   }
```

- Mittels TableCellEditors kann man die Zellen editieren
  - Liefert Komponente zum Editieren
  - Stellt Interface für Interaktion bereit (CellEditor)

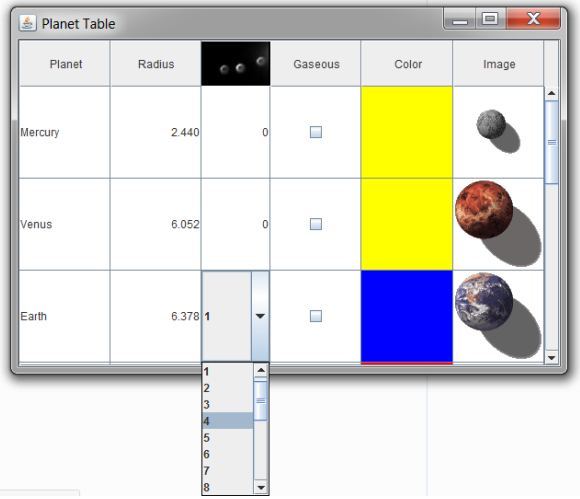
```
public interface TableCellEditor extends CellEditor {
   Component getTableCellEditorComponent(JTable table, Object value,
                  boolean isSelected, int row, int column);
}
```

```
public interface CellEditor {
   public Object getCellEditorValue();
   public boolean isCellEditable(EventObject anEvent);
   public boolean shouldSelectCell(EventObject anEvent);
   public boolean stopCellEditing();
   public void cancelCellEditing();
   public void addCellEditorListener(CellEditorListener l);
   public void removeCellEditorListener(CellEditorListener l);
}
```

DefaultTableCellEditor

- Mit DefaultTableCellEditor kann man aus JTextField, JCheckBox und JComboBox ein TableCellEditor konstruieren
- Beispiel: Im folgenden Beispiel wird ein Editor für Anzahl der Monde eines Planeten realisiert (siehe Beispiel Planetentabelle)
  - Erzeugen einer JComboBox mit Items 0 bis 20
  - Erzeugen eines DefaultTableCellEditors mit JComboBox
  - Installation als CellEditor für die Spalte

```
JComboBox<Integer> moonCombo = new JComboBox<>();
for (int i = 0; i <= 20; i++)
    moonCombo.addItem(new Integer(i));

TableColumnModel columnModel = table.getColumnModel();

TableColumn moonColumn
    = columnModel.getColumn(PlanetTableModel.MOON_COLUMN);
moonColumn.setCellEditor(new DefaultCellEditor(moonCombo));
```



Beispiel ColorTableEditor (1/2)

- Editor mit ColorChooser als Popup-Dialog

```
class ColorTableCellEditor extends AbstractCellEditor implements TableCellEditor {
   private JColorChooser colorChooser;
   private JDialog colorDialog;
   private JPanel panel;

   public ColorTableCellEditor() {

      panel = new JPanel();

      // prepare color dialog
      colorChooser = new JColorChooser();
      colorDialog = JColorChooser.createDialog(null, "Planet Color", false,
            colorChooser, new ActionListener() // OK button listener
            {
               public void actionPerformed(ActionEvent event) {
                  stopCellEditing();
               }
            }, new ActionListener() // Cancel button listener
            {
               public void actionPerformed(ActionEvent event) {
                  cancelCellEditing();
               }
            });
      colorDialog.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent event) {
            cancelCellEditing();
         }
      });
   }
...
```

Erzeugen des ColorDialogs

Beispiel ColorTableEditor(2/2)

```
...
    public Component getTableCellEditorComponent(JTable table,
           Object value, boolean isSelected, int row, int column) {
        colorChooser.setColor((Color) value);
        return panel;
    }

    public boolean shouldSelectCell(EventObject anEvent) {
        colorDialog.setVisible(true);
        return true;
    }

    public void cancelCellEditing() {
        colorDialog.setVisible(false);
        super.cancelCellEditing();
    }

    public boolean stopCellEditing() {
        colorDialog.setVisible(false);
        super.stopCellEditing();
        return true;
    }

    public Object getCellEditorValue() {
        return colorChooser.getColor();
    }

}
```

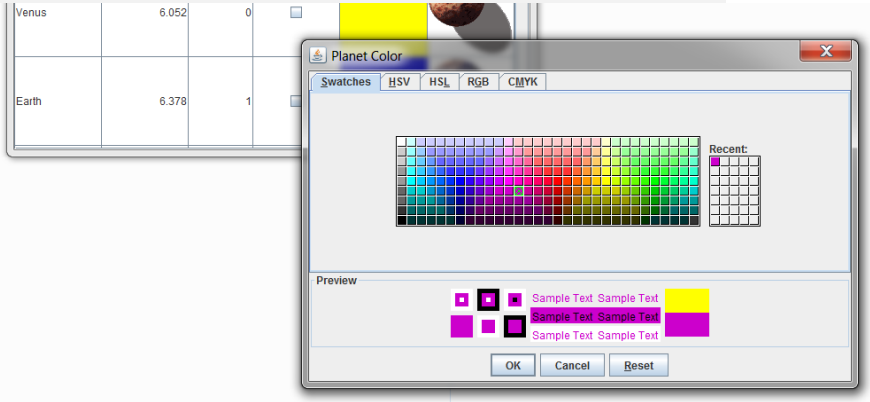
Liefert Component zur Darstellung des Feldes

Aufruf bei Aktivierung des Editorvorgangs

Aufruf bei Abbruch des Editorvorgangs

Aufruf bei Beenden des Editorvorgangs

Wert der übernommen wird

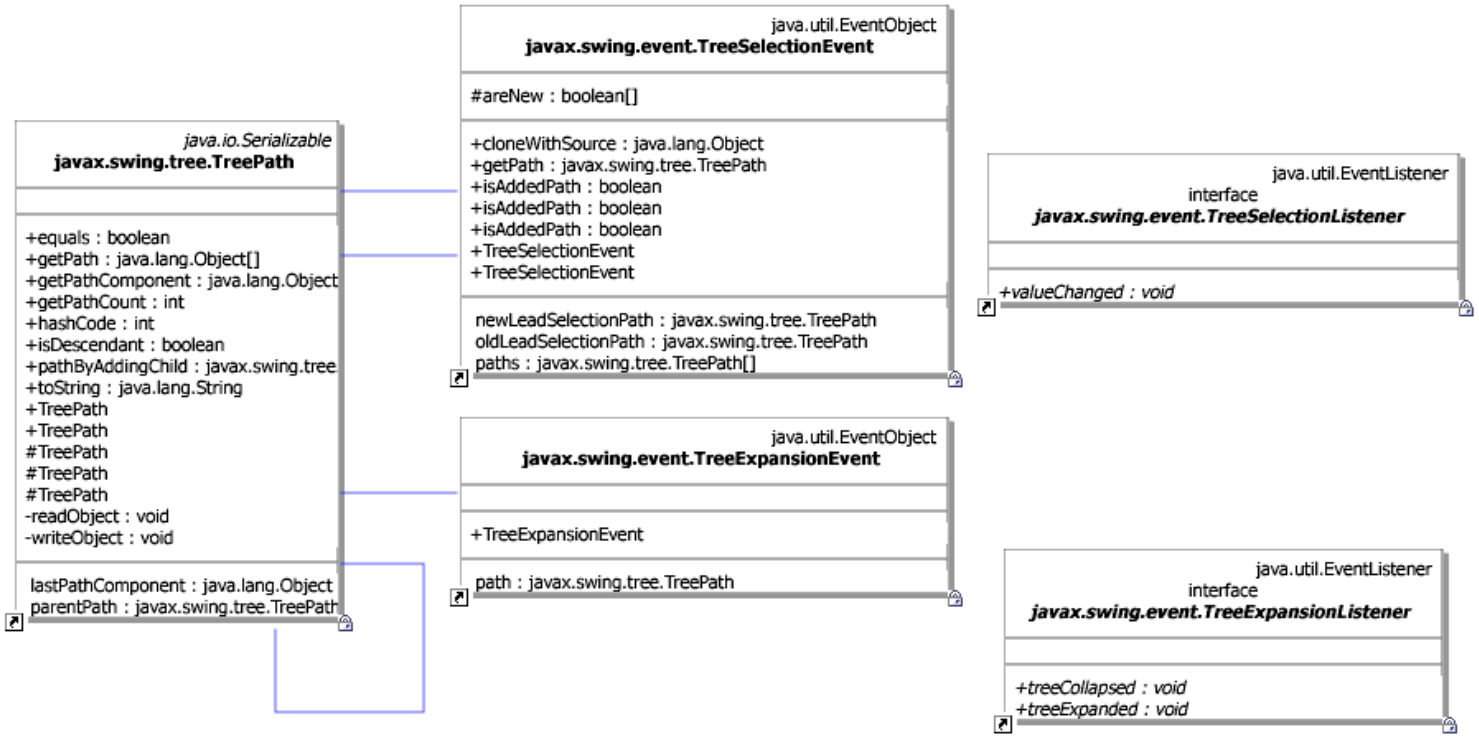


MVC

- JSpinner
- JList
- JTable
- JTree

JTree

- Treeview mit expandierbarem Baum (wie Windows Explorer)
- Im Aufbau und Funktionsweise analog zu JTable
- JTree verwendet Ereignisse:
  - TreeSelectionEvent
  - TreeExpansionEvent



INTERFACE TREEMODEL

- TreeModel definiert
  - Zugriff auf Knoten im Baum
  - Änderung bei Pfaden
  - Registerierung von TreeModelListener

Interface TreeModel

```
public interface TreeModel {

    public Object getRoot();
    public Object getChild(Object parent, int index);
    public int getChildCount(Object parent);
    public boolean isLeaf(Object node);
    public int getIndexOfChild(Object parent, Object child);
    public void valueForPathChanged(TreePath path, Object newValue);

    void addTreeModelListener(TreeModelListener l);
    void removeTreeModelListener(TreeModelListener l);
}
```

Aufgerufen von Editor mit geändertem Wert

```
public interface TreeModelListener extends EventListener {
     void treeNodesChanged(TreeModelEvent e);
     void treeNodesInserted(TreeModelEvent e);
     void treeNodesRemoved(TreeModelEvent e);
     void treeStructureChanged(TreeModelEvent e);
}
```

Änderung eines Knotenwertes
Anfügen von Knoten beim Vater
Löschen von Knoten beim Vater
Größeren Strukturänderungen

```
public class TreeModelEvent extends EventObject {
    public TreeModelEvent(Object source, Object[] path, int[] childIndices, Object[] children)
    public TreeModelEvent(Object source, TreePath path, int[] childIndices, Object[] children)
    public TreeModelEvent(Object source, Object[] path)
    public TreeModelEvent(Object source, TreePath path)
    public TreePath getTreePath()
    public Object[] getPath()
    public Object[] getChildren()
    public int[] getChildIndices()
}
```

`path` – Pfad zum Knoten, wo Änderung passiert ist (Vaterknoten)
`children` – geänderte Knoten ($\in$ Kinder des Vaterknotens)
`childIndices` – Indizes der geänderten Knoten (innerhalb der Kinder des Vaterknotens)

Beispiel TreeModel: Zahlenwerte

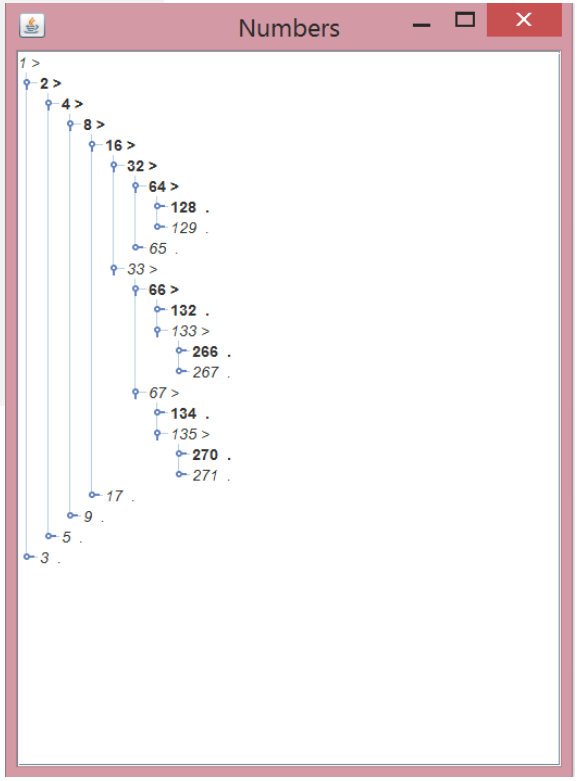
- Binärer Baum mit Knotenwerte Integers n dynamisch generiert
- mit Nachfolgern
  - n * 2
  - n * 2 + 1

```
public class IntTreeModel implements TreeModel {

  private int root;

  public IntTreeModel() {
    root = 1;
  }

  @Override
  public Object getRoot() {
    return root;
  }

  @Override
  public Object getChild(Object parent, int index) {
    int n = (int) parent;
    switch (index) {
    case 0:
      return n * 2;
    case 1:
      return n * 2 + 1;
    default:
      return null;
    }
  }
```

```
  @Override
  public int getChildCount(Object parent) {
    return 2;
  }

  @Override
  public int getIndexOfChild(Object parent,
                             Object child) {
    int n = (int) parent;
    int c = (int) child;
    if (c == n * 2)
      return 0;
    if (c == n * 2 + 1)
      return 1;
    return -1;
  }

  @Override
  public boolean isLeaf(Object node) {
    return false;
  }

  …
```

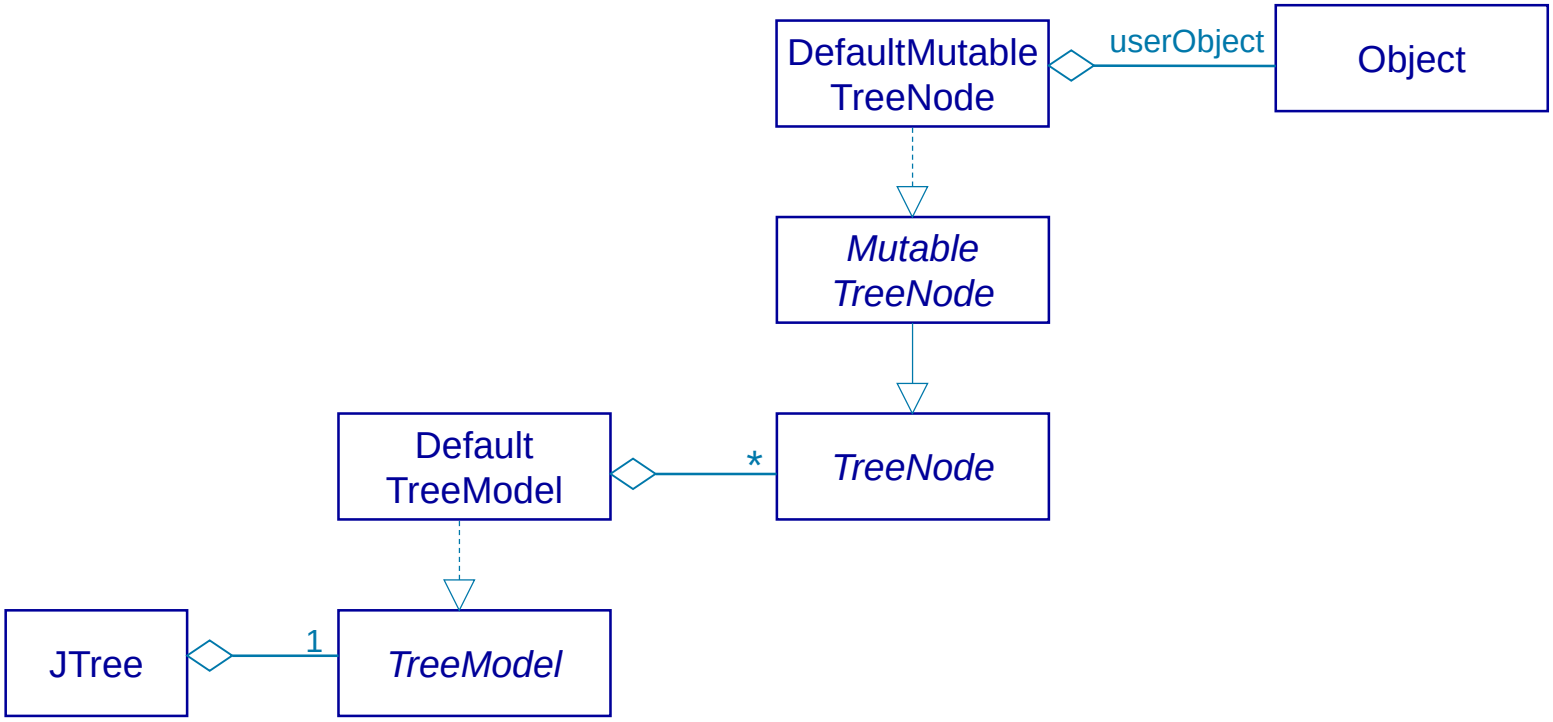
Beispiel TreeModel: Zahlenwerte

```
public class Main {

  public static void main(String[] args) {
    JFrame frame = new JFrame();

    frame.setTitle("Integers");
    frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

    JTree tree = new JTree(new IntTreeModel());
    frame.getContentPane().add(new JScrollPane(tree));

    frame.setLocation(400, 300);
    frame.pack();
    frame.setVisible(true);
  }

}
```



DefaultTreeModel

- DefaultTreeModel bietet eine Implementierung eines TreeModels
- Aufbau eine Baumstruktur mit TreeNodes
- DefaultMutableTreeNode ist Standardimplementierung von TeeNode: haben userObject !



Beispiel: Arbeiten mit DefaultMutableTreeNode

userObject

- Knoten anlegen und Baustruktur aufbauen

```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");
DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
root.add(country);
DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
country.add(state);
DefaultMutableTreeNode city = new DefaultMutableTreeNode("San Jose");
state.add(city);
city = new DefaultMutableTreeNode("Cupertino");
state.add(city);
...
```

- DefaultTreeModel und JTree erzeugen

```
DefaultTreeModel model = new DefaultTreeModel(root);
JTree tree = new JTree(model);
```

Bearbeiten von Bäumen

- Folgende Operationen bei DefaultTreeModel dienen zum Bearbeiten von Bäumen (bewirken automatische Updates im JTree)
  - Anfügen von neuen Knoten mit

```
model.insertNodeInto(newNode, parentNode, index);
```

  - Löschen eines Knotens von neuen Knoten mit

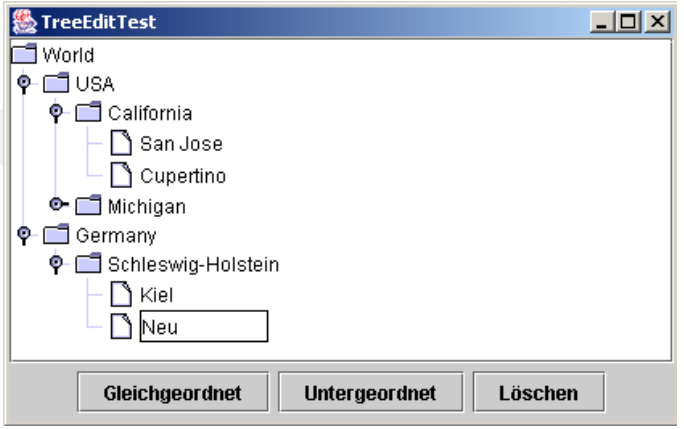
```
model.removeNodeFromParent(node);
```

  - Anzeige von Änderungen bei einem Knoten

```
model.nodeChanged(node);
```

- Editieren von Knoten
  - Setzen von Editable-Eigenschaft
  - Editieren erfolgt durch DefaultCellEditor

```
tree.setEditable(true);
```



TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

- Beispiel Num:
  - speichert Zahlenwert
  - erzeugt dynamisch Unterobjekte
  - hierarchisch organisiert (parent – children)
  - Werte veränderbar
  - mit ChangeEvent

```
public class Num {

  private final Num parent;
  private int value;
  private boolean expanded;
  private Num left;
  private Num right;
  private final List<NumChangedListener> listeners;

  public Num(int value) {
    this(null, value);
  }

  public Num(Num parent, int value) {
    this.parent = parent;
    this.value = value;
    listeners = new ArrayList<NumChangedListener>();
  }

  public Num getParent() {
    return parent;
  }

  …
```

Zugriff auf parent!

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

- Num ff

```
…
  public int getValue() {
     return value;
  }

  public void setValue(int value) {
     this.value = value;
     collapse();
     fireNumChanged();
  }

  public void setObjectValue(Object newValue) {
     value = (int) newValue;
     setValue(value);
  }

  public Num getLeft() {
     if (! expanded) {
        expand();
     }
     return left;
  }

  public Num getRight() {
     if (! expanded) {
        expand();
     }
     return right;
  }

  private void expand() {
     left = new Num(this, value * 2);
     right = new Num(this, value * 2 + 1);
     expanded = true;
  }

  private void collapse() {
     left = null;
     right = null;
     expanded = false;
  }
…
```

Bei Setzen Löschen der Kinder und feuern von ChangeEvent!

Zugriff auf linken und rechtes Kind mit dynamisches Generierung der Kinder (expand)

Expandieren und Löschen der Kinder

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

- Num ff

```
…
 public void addNumChangeListener
       (NumChangedListener l) {
   if (listeners.contains(l)) return;
   listeners.add(l);
 }

 public void removeNumChangeListener(NumChangedListener l) {
   listeners.remove(l);
 }

 private void fireNumChanged() {
   NumChangedEvent evt = new NumChangedEvent(this, this);
   for (NumChangedListener l: listeners) {
     l.numChanged(evt);
   }
 }


}
```

Change Events

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

- Klasse NumTreeModel

```
public class NumTreeModel implements TreeModel {

   private Num rootNode;
   private EventListenerList listenerList =
         new EventListenerList();
   private NumChangedListener numListener;

   public NumTreeModel(int rootValue) {
      rootNode = new Num(rootValue);
      numListener = new NumChangedListener() {
         @Override
         public void numChanged(NumChangedEvent e) {
            fireTreeNodesChangedEvent
                           (getPathToRoot(e.getNum()));
         }
      };
   }

   @Override
   public Object getRoot() {
      return rootNode;
   }

   @Override
   public Object getChild(Object parent, int index) {
      Num num = (Num) parent;
      Num child;
      switch (index) {
         case 0: child = num.getLeft(); break;
         case 1: child = num.getRight(); break;
         default: child = null;
      }
      if (child != null) {
         child.addNumChangeListener(numListener);
      }
      return child;
   }
   …
```

Listener auf Änderungen in Daten signalisiert die Änderungen an JTree

Liefern der Information über Knoten nach Interface von TreeModel

Anfügen eines Listeners bei Daten um von Änderungen in Daten zu erfahren!

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

- NumTreeModel ff

```
…
@Override
public int getChildCount(Object parent) {
   return 2;
}

@Override
public int getIndexOfChild(Object parent, Object child) {
   Num num = (Num) parent;
   if (child == num.getLeft()) return 0;
   if (child == num.getRight()) return 1;
   return -1;
}

@Override
public boolean isLeaf(Object node) {
   return false;
}

@Override
public void valueForPathChanged(TreePath path, Object nv) {
   Num node = ((Num) path.getLastPathComponent());
   node.setObjectValue(nv);
}

private TreePath getPathToRoot(Num num) {
   List<Num> path = new ArrayList<Num>();
   path.add(num);
   while (num.getParent() != null) {
      num = num.getParent();
      path.add(num);
   }
   Collections.reverse(path);
   return new TreePath(path.toArray(new Num[path.size()]));
}

…
```

Liefern der Information über Knoten nach Interface von TreeModel

Aufgerufen bei Editieroperationen; Setzt den Wert in den Daten und signalisiert Änderung

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

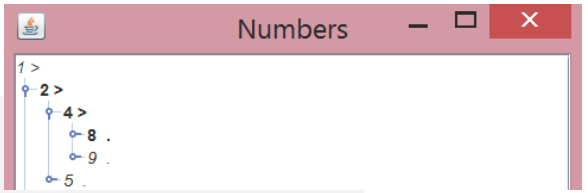
- NumTreeModel ff

```
  …
  //-- tree event
  @Override
  public void addTreeModelListener(TreeModelListener l) {
     listenerList.add(TreeModelListener.class, l);
  }

  @Override
  public void removeTreeModelListener(TreeModelListener l) {
     listenerList.remove(TreeModelListener.class, l);
  }

  private void fireTreeStrutureChangedEvent(TreePath path) {
     TreeModelEvent evt = new TreeModelEvent(this, path, new int[0], new Object[0]);
     for (TreeModelListener l : listenerList
          .getListeners(TreeModelListener.class)) {
        l.treeStructureChanged(evt);
     }
  }
}
```

Benachrichtigen des JTree von Änderungen in der Knotenstruktur!

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen

- Renderer: Darstellung der Knoten beeinflussen



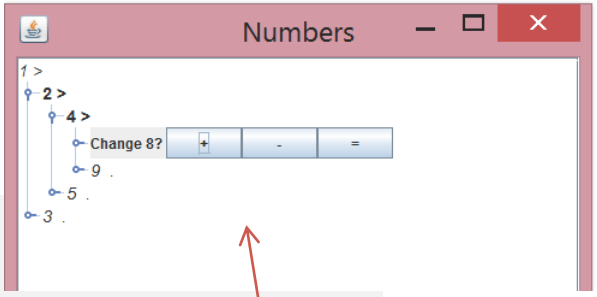

```
public class NumCellRenderer implements TreeCellRenderer {

   private static JLabel label;

   static {
      label = new JLabel();
   }

   @Override
   public Component getTreeCellRendererComponent(JTree tree, Object value,
         boolean selected, boolean expanded, boolean leaf, int row,
         boolean hasFocus) {
      Num node = (Num)value;
      if (node.getValue() % 2 == 0) {
         label.setFont(new Font("SansSerif", Font.BOLD, 14));
      } else {
         label.setFont(new Font("SansSerif", Font.ITALIC, 14));
      }
      label.setText(node.toString());
      return label;
   }

}
```

gerade Werte fett

ungerade Werte italic

Komponente, die retourniert wird wird für Ausgabe verwendet!

TreeModel Beispiel 3: Editierbare, hierarchische Strukturen



- Editor: Editieren der Knoten erlauben

```
public class NumCellEditor extends AbstractCellEditor implements
      TreeCellEditor {
   private Object inputValue;

   public NumCellEditor() { super(); }

   public Component getTreeCellEditorComponent(final JTree tree, Object cellValue,
                           boolean isSelected, boolean expanded, boolean leaf, int row) {
      final Num node = (Num) cellValue;
      final int value = node.getValue();
      final JPanel panel = new JPanel(new GridLayout(1, 4));
      final JLabel lbl = new JLabel("Change " + value + "? ");
      final JButton plusBtn = new JButton("+");
      final JButton minusBtn = new JButton("-");
      final JButton abortBtn = new JButton("=");
      plusBtn.addActionListener(a -> {
         inputValue = value + 1;
         stopCellEditing();
      });
      minusBtn.addActionListener(a -> {
         inputValue = value - 1;
         stopCellEditing();
      });
      abortBtn.addActionListener(a -> {
         inputValue = value;
         cancelCellEditing();
      });
      panel.add(lbl);
      panel.add(plusBtn);
      panel.add(minusBtn);
      panel.add(abortBtn);
      return panel;
   }

   @Override
   public Object getCellEditorValue() { return inputValue; }
}
```

Editierkomponente is Panel mit mehreren Buttons !

Action von plusButton:
Editierwert ist Wert +1
Ende der Editieroperation angezeigt

Action von cancelBtn:
Bricht Editieren ohne Übernehmen des Wertes ab

Komponente, die retourniert wird als „In place“ Editor verwendet!

Wert der bei Ende der Editoroperation gesetzt wird!

Selektion und Pfade

- TreePath stellt einen Pfad von Wurzel bis Knoten dar
  - mit getSelectionPath() erhält man Pfad zur aktuellen Selektion

```
TreePath selectionPath = tree.getSelectionPath();
```

  - mit getLastPathComponent von TreePath erhält man aktuell selektiertes Objekt

```
DefaultMutableTreeNode selectedNode =
    (DefaultMutableTreeNode )selectionPath.getLastPathComponent();
```

- Sichtbar Machen und Öffnen eines Pfades mit makeVisible und scrollPathToVisible

```
TreeNode[] nodes = model.getPathToRoot(node);
TreePath path = new TreePath(nodes);
tree.makeVisible(path);
tree.scrollPathToVisible(path);
```

Selektionsereignisse

- TreeSelectionEvents erlauben auf Selektion von Knoten zu horchen

```
interface TreeSelectionListener {
    void valueChanged(TreeSelectionEvent e);
}
```

```
public class  TreeSelectionEvent extends EventObject {
    public TreePath getPath()
    public TreePath[] getPaths()

}
```

```
public class  JTree extends JComponent {
    public void addTreeSelectionListener(TreeSelectionListener tsl)
    public void removeTreeSelectionListener(TreeSelectionListener tsl)
    …
}
```

Darstellungsoptionen (1)

- LineStyle

```
tree.putClientProperty("JTree.lineStyle", "Angled");
tree.putClientProperty("JTree.lineStyle", "None");
tree.putClientProperty("JTree.lineStyle", "Horizontal");
```

- Root

```
tree.setRootVisible(true);
tree.setShowsRootHandles(true);
```

Wurzelknoten angezeigt?
Root mit Aufklappsymbol

- Blätter
  - normalerweise, die die keine Unterknoten haben
  - kann bei `DefaultMutableTreeNode` auch definiert werden
    - zuerst beim `JTree` erlauben

```
tree.setAsksAllowsChildren(true);
```

    - dann beim Knoten einstellen

```
DefaultMutableTreeNode node = …;
node.setAllowsChildren(true);
```

Darstellungsoptionen (2)

- Installation eines eigenen Renderes (wie bei JList und JTable)
  - Implementieren eines TreeCellRenderers mit Methode getTreeCellRendererComponent

```
class MyTreeCellRenderer implements TreeCellRenderer {
    @Override
    public Component getTreeCellRendererComponent
        (JTree tree, Object value,boolean selected, boolean expanded,
         boolean leaf, int row, boolean hasFocus) {
        // ... set properties of label
        return label;
    }
    private JLabel label = new JLabel();
}

tree.setCellRenderer(new MyTreeCellRenderer());
```

- Icons und Schriftart: DefaultTreeCellRenderer unterstützt Installation von Icons und Schriften

```
DefaultTreeCellRenderer renderer =
        new DefaultTreeCellRenderer();
renderer.setLeafIcon(new ImageIcon("leaf.gif"));
renderer.setClosedIcon(new ImageIcon("closed.gif"));
renderer.setOpenIcon(new ImageIcon("open.gif"));
renderer.setFont(new Font("Serif", Font.ITALIC, 10));
tree.setCellRenderer(renderer);
```

Darstellungsoptionen (3)

- Erweitern des DefaultTreeCellRenderers (erweitert JLabel)

```
public class DefaultTreeCellRenderer
                extends JLabel implements TreeCellRenderer
```

```
class MyTreeCellRenderer extends DefaultTreeCellRenderer {
   @Override
   public Component getTreeCellRendererComponent
                (JTree tree, Object value, boolean sel,
                 boolean expanded, boolean leaf, int row,boolean hasFocus) {
      super.getTreeCellRendererComponent(tree, value, sel, expanded, leaf,
      row, hasFocus);
      DefaultMutableTreeNode node = (DefaultMutableTreeNode)value;
      // ... setFont, setText, ...
      return this;
   }
}
```