Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

Root-Fenster

JComponent

JLabel/JTextField

JList

JTable

Das Hauptfenster

Die Toplevel-Fenster (z. B. `JFrame`) bestehen aus einem Hauptfenster, dem so genannten *root pane*.

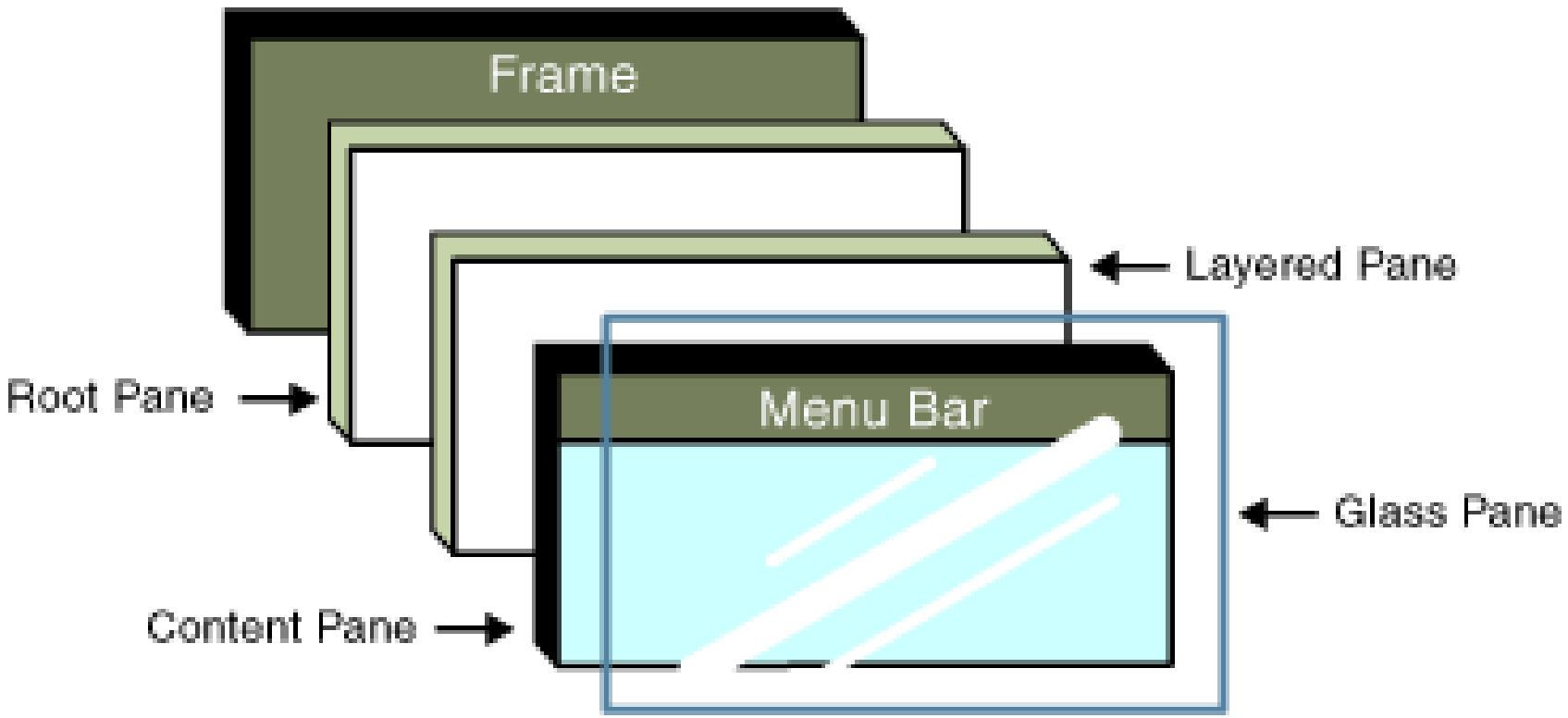
Dieses besteht aus

Layered Pane Hierin sind enthalten:

- der eigentliche Inhalt des Fensters, das *content pane*
- der *menu bar*

Glass Pane eine Art Glasscheibe über dem gesamten Fenster. Dieser kann verwendet werden, um Effekte über dem Fenster erscheinen zu lassen.

Schemadarstellung: Das Hauptfenster



Der Content-Pane

Der *Content Pane* ist der Container, der Swing-Komponenten (Labels, Buttons, . . .) aufnimmt. Vor Java 5 konnten einem Toplevel-Container die Komponenten daher nicht direkt hinzu gefügt werden, und typischer Code sah so aus:

```
JFrame myFrame = new JFrame();
Container contentPane = myFrame.getContentPane();
contentPane.add(new JTextField ());
```

Seit Java 5 kann nun direkt mit `add` gearbeitet werden.

Die Basisklass JComponent

`JComponent` ist die Basisklasse aller Swing-Komponenten und unterstützt folgende Features:

- Tool Tips
- Rahmen
- Einstellungen zum Look & Feel
- Unterstützung von Layout-Managern
- Behandlung von Tastatur-Ereignissen
- Accessibility für Menschen mit Behinderungen

Tool Tips

Tool Tips erscheinen, wenn die Maus länger auf einer Komponente verweilt, und sollen dem Nutzer eine Hilfe bieten. Tool Tips können einfach mit der Methode `setToolTipText(String)` definiert werden.

Beispiel:

```
JTextField eingabe = new JTextField();
eingabe.setToolTipText("Text eingeben");
```

Rahmen

Um Komponenten können Rahmen gemalt werden. Die einfachsten und wichtigsten sind:

LineBorder eine einfache Linie

EtchedBorder eine Linie mit 3D-Effekt, entweder vertieft oder erhöht

BevelBorder die gesamte Komponente erscheint vertieft oder erhöht

EmptyBorder unsichtbarer Rahmen (für den aber Platz reserviert sein kann)

Beispiel

```
JFrame frame = new JFrame();
frame.setLayout(new GridLayout(0,1,5,5));
JLabel label = new JLabel("Label 1");
label.setBorder(BorderFactory.createLineBorder(Color.red));
frame.add(label);
label = new JLabel("Label 2");
label.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.RAISED));
frame.add(label);
label = new JLabel("Label 3");
label.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED));
frame.add(label);
label = new JLabel("Label 4");
label.setBorder(BorderFactory.createRaisedBevelBorder());
frame.add(label);
label = new JLabel("Label 4");
label.setBorder(BorderFactory.createLoweredBevelBorder());
frame.add(label);
frame.pack();
frame.setVisible(true);
```

Look & Feel

Zum Ändern des Look & Feel einer Anwendung kann man die Klasse `UIManager` verwenden:
`UIManager.setLookAndFeel(String)`

Gültige Parameter sind

- `UIManager.getCrossPlatformLookAndFeel()`
- `UIManager.getSystemLookAndFeel()`

Unterstützung von Layout-Managern

Um den Layout-Manager bei der Anzeige der Komponenten zu unterstützen, existieren folgende Methoden:

setPreferredSize(Dimension) zur Angabe der optimalen Größe einer Komponente

setMaximumSize(Dimension) zur Angabe der maximalen Größe

setMinimumSize(Dimension) zur Angabe der minimalen Größe

Die Dimension kann hierbei durch

```
new Dimension(x,y)
```

angegeben werden.

Achtung: Der Layout-Manager muss sich nicht an die Vorgaben halten!

JLabel

Konstruktoren/Methoden von `JLabel`:

`JLabel(String)`, `JLabel(Icon)`, `JLabel(String, int)`, ...	
`void setText(String)`	Text setzen
`String getText()`	Text lesen
`void setDisplayedMnemonic(char)`	Tastatur-Shortcut setzen
`void setLabelFor(Component)`	Komponente, an die der Shortcut weiter geleitet wird
`void setHorizontalAlignment(int)`	hor. Ausrichtung

Statische int-Attribute in `SwingConstants`: `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`

JTextField

Konstruktoren/Methoden von `JTextField`:

`JTextField()`, `JTextField(String)`, `JTextField(int)`, ...	
`void setEditable(boolean)`	Feld beschreibbar?
`void addActionListener(...)`	Listener setzen
`void removeActionListener(...)`	Listener entfernen
`String getText()`	Text lesen
`String getSelectedText()`	Markierten Text lesen

Die Klasse `JList`

Die Klasse `JList` stellt einen Container für Auswahllisten zur Verfügung.

Hierbei sind eine rein vertikale und eine tabellenartige Anordnung möglich. Weiterhin kann neben Einzelauswahl die Auswahl von Bereichen und mehreren Elementen zugelassen werden.

Über die Methode `addListSelectionListener` kann ein entsprechender Listener beim Selektieren benachrichtigt werden. Dieser muss die Methode `valueChanged` implementieren.

Beispiel

```
public class App extends JFrame implements ListSelectionListener {
  private JList list;
  public App() {
    super("Meine Anwendung");
    this.setLayout(new BorderLayout());
    String[] data = {"eins","zwei","drei","vier","fuenf"};
    list = new JList(data);
    list.setLayoutOrientation(JList.VERTICAL);
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    list.addListSelectionListener(this);
    this.add(list,BorderLayout.PAGE_START);
    this.pack();
    this.setVisible(true);
  }
```

Beispiel (Forts.)

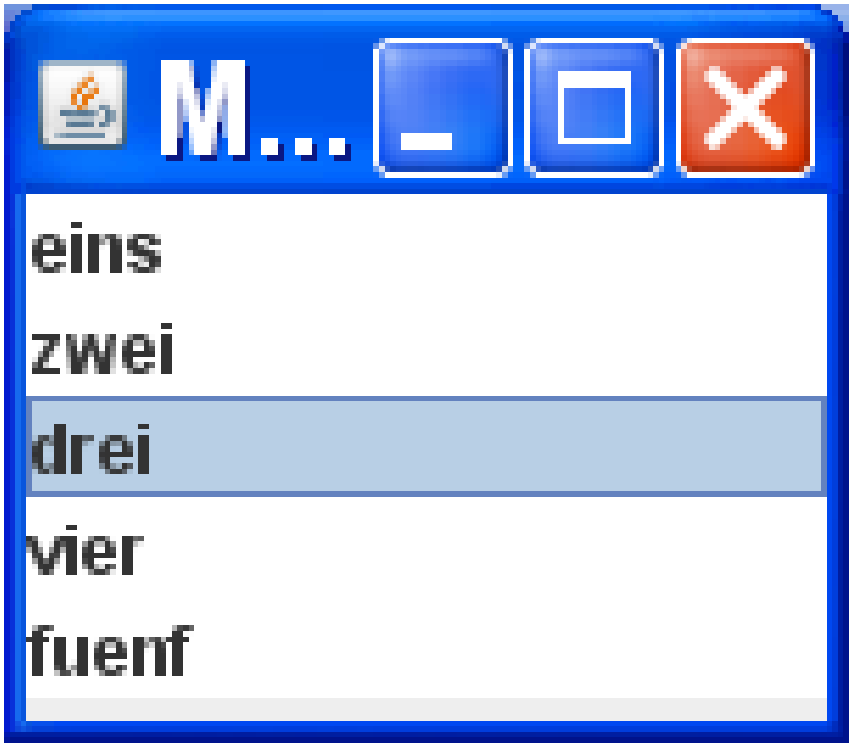
```
    public void valueChanged(ListSelectionEvent event) {
        if (event.getValueIsAdjusting()) return;

        if (list.getSelectedIndex() != -1) {
            System.out.println(list.getSelectedValue());
        }

    }

    public static void main(String[] args) {
        App myApp = new App();
    }
}
```

Resultat



ListModel

In der im Beispiel verwendeten Form ist `JList` sehr starr. Aus der Auswahlliste können z. B. keine Elemente hinzu gefügt oder entfernt werden.

Möchte man flexiblere Listen, so kann man ein `ListModel` verwenden.

- In den meisten Fällen reicht `DefaultListModel`.
- Möchte man ein eigenes `ListModel` implementieren, so kann man die Klasse `AbstractListModel` als Basisklasse verwenden.

Beispiel

```
public class App extends JFrame
            implements ActionListener, ListSelectionListener {
    private JList list;
    private DefaultListModel listModel;
    private JTextField eingabe;
    public App() {
        ...
        listModel = new DefaultListModel();
        list = new JList(listModel);
        list.setLayoutOrientation(JList.VERTICAL);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION)
        list.addListSelectionListener(this);
        this.add(new JScrollPane(list),BorderLayout.PAGE_START);
        eingabe = new JTextField(20);
        eingabe.addActionListener(this);
        this.add(eingabe,BorderLayout.PAGE_END);
        ...
    }
```

Beispiel (Forts.)

```
    public void valueChanged(ListSelectionEvent event) {
        if (event.getValueIsAdjusting()) return;

        if (list.getSelectedIndex() != -1) {
            System.out.println(list.getSelectedValue());
        }
    }

    public void actionPerformed(ActionEvent event) {
        listModel.addElement(eingabe.getText());
        eingabe.setText("");
    }
}
```

Die Klasse JTable

Mit der Klasse `JTable` können Tabellen gestaltet werden, die z. B. Spaltentitel oder verschiebbare Spaltengrenzen besitzen. In der einfachsten Form erwartet der Konstruktor einen 2D-Array mit den Tabellendaten und einen Array mit den Spaltentiteln. Diese Form ist allerdings sehr starr, da die Tabellenfelder nicht geändert werden können und keine Zeilen hinzu gefügt oder geändert werden können.

Beispiel

```
public class App extends JFrame {
    public App() {
        super("Meine Anwendung");
        this.setLayout(new BorderLayout());
        String[] columnNames = {"First Name","Last Name","Hobby"};
        Object[][] data = {
                {"Mary", "Campione", "Snowboarding"},
                {"Alison", "Huml", "Rowing"},
                {"Kathy", "Walrath", "Knitting" },
                {"Sharon", "Zakhour", "Speed reading"},
                {"Philip", "Milne", "Pool"}
        };
        JTable myTable = new JTable(data, columnNames);
        myTable.setFillsViewportHeight(true);
        this.add(new JScrollPane(myTable),BorderLayout.PAGE_START)
        this.pack();
        this.setVisible(true);
    }
}
```

Resultat


Meine Anwendung
First Name
Last Name
Hobby
Mary
Campione
Snowboarding
Alison
Huml
Rowing
Kathy
Walrath
Knitting
Sharon
Zakhour
Speed reading
Philip
Milne
Pool

Eigene Tabellenmodelle

Um ein eigenes Tabellenmodell zu implementieren, erbt man am Einfachsten von der Klasse `AbstractTableModel`.

Hierin können folgende Methoden überschrieben werden:

`int getColumnCount()`	Zahl der Spalten
`int getRowCount()`	Zahl der Zeilen
`Object getValueAt(row,col)`	Element in Zeile *row* und Spalte *col*
`void setValueAt(obj, row, col)`	Setzt das Element in Zeile *row* und Spalte *col* auf *obj*

Beispiel für ein Tabellenmodell

Im folgenden wird ein Tabellenmodell implementiert, das für eine Menge von Zahlen $0 \ldots n-1$ in der ersten Spalte die Zahl selber und in der zweiten das Quadrat dieser Zahl anzeigt. Hierbei ist die Zahl der Zeilen n variabel und kann durch eine Methode `increase` jeweils um eins erhöht werden.

Die Klasse MyTableModel

```
public class MyTableModel extends AbstractTableModel {
    private int index = 0;
    public void increase() {
        index++;
        this.fireTableStructureChanged();
    }
    public int getColumnCount() {
        return 2;
    }
    public int getRowCount() {
        return index;
    }
    public Object getValueAt(int row, int col) {
        if (col == 0) {
            return new Integer(row);
        } else if (col == 1) {
            return new Integer(row*row);
        } else return null;
    }
}
```

Die Klasse App

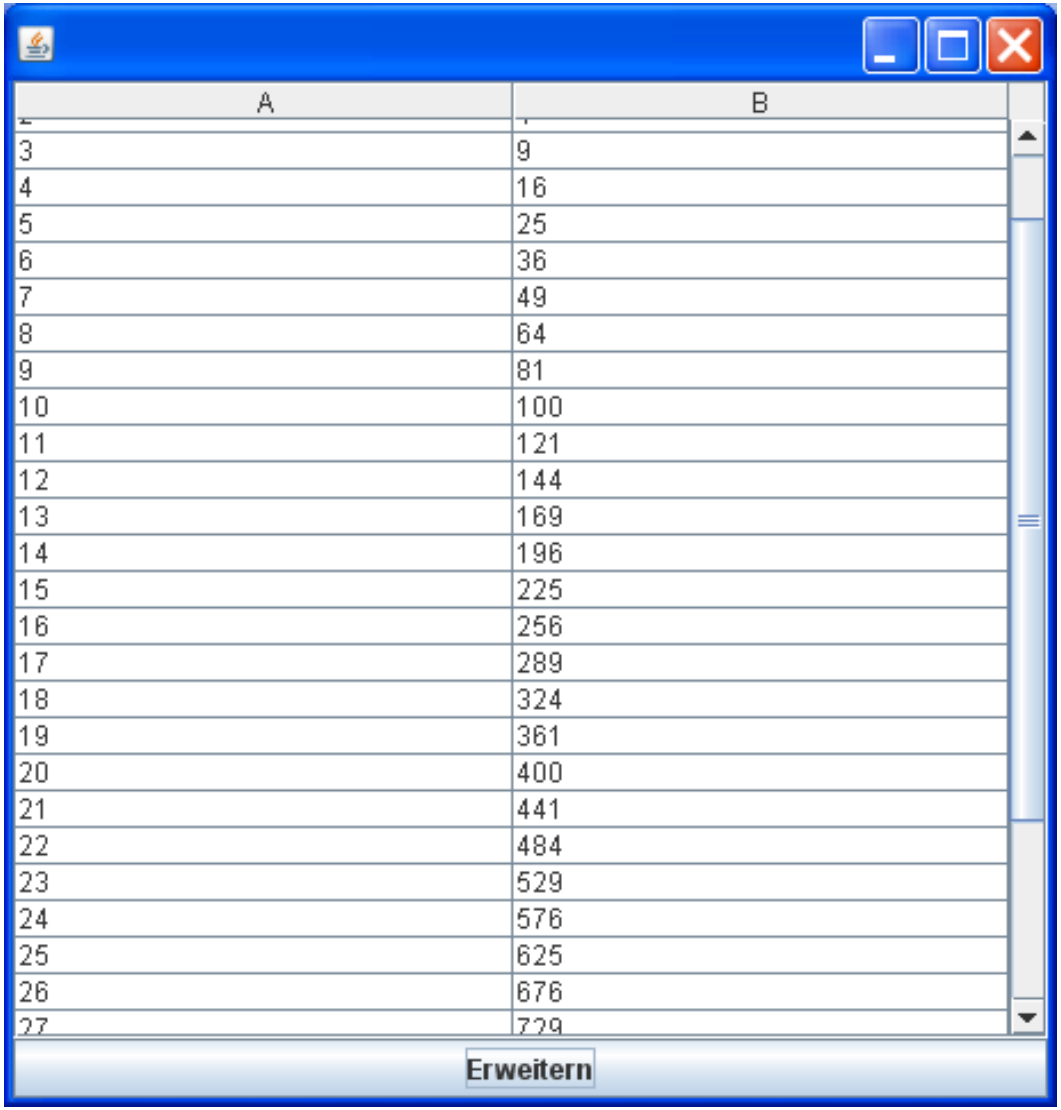
```
public class App extends JFrame implements ActionListener {
    private JButton applyButton;
    private MyTableModel myModel;
    public App() {
        super();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new BorderLayout());
        applyButton = new JButton("Erweitern");
        this.add(applyButton,BorderLayout.PAGE_END);
        myModel = new MyTableModel();
        JTable table = new JTable(myModel);
        JScrollPane scrollPane = new JScrollPane(table);
        table.setFillsViewportHeight(true);
        applyButton.addActionListener(this);
        this.add(scrollPane,BorderLayout.PAGE_START);
        this.pack();
        this.setVisible(true);
    }
```

Die Klasse App (Forts.)

```
    public void actionPerformed(ActionEvent event) {
        myModel.increase();
    }

    public static void main(String[] args) {
        App app = new App();
    }
}
```

Resultat


A
B
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
11 121
12 144
13 169
14 196
15 225
16 256
17 289
18 324
19 361
20 400
21 441
22 484
23 529
24 576
25 625
26 676
27 729
Erweitern