

Root-Fenster JComponent JLabel/JTextField JList JTable

Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

1 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

Root-Fenster

JComponent

JLabel/JTextField

JList

JTable

2 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

Das Hauptfenster

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

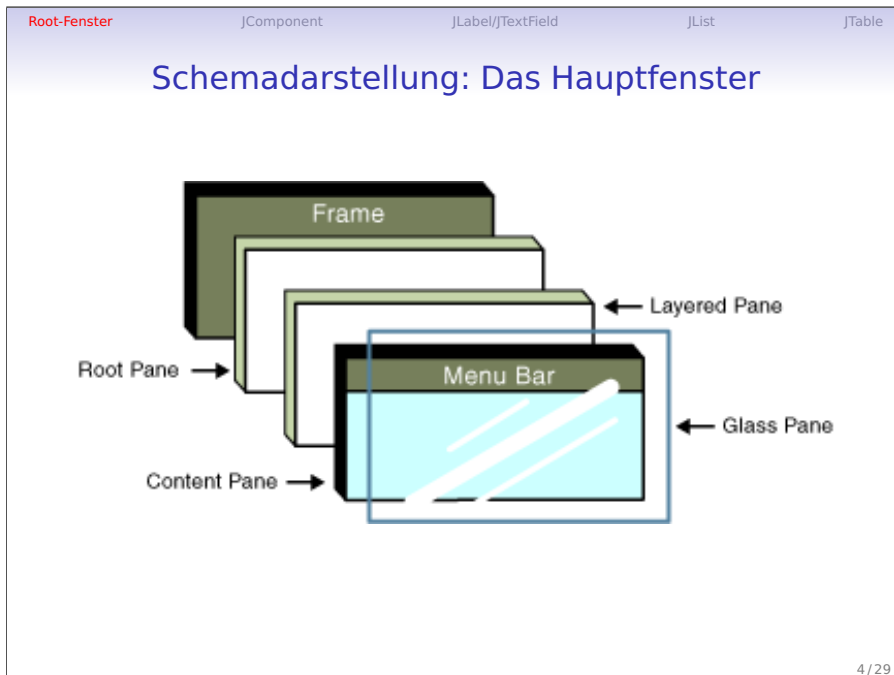
Dieses besteht aus

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.

3 / 29



Root-Fenster JComponent JLabel/JTextField JList JTable

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.

5 / 29

Root-Fenster **JComponent** JLabel/JTextField JList JTable

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

6 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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");
```

7 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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)

8 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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);
```

9 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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()`

10/29

Root-Fenster JComponent JLabel/JTextField JList JTable

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!

11/29

Root-Fenster JComponent JLabel/JTextField JList JTable

JLabel

Konstruktoren/Methoden von `JLabel`:

`JLabel(String), JLabel(Icon), JLabel(String, int), ...`

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

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

12/29

Root-Fenster JComponent JLabel/JTextField JList JTable

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

13 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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.

14 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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);
    }
}

```

15 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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();
}

```

16 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

Resultat



17 / 29

Root-Fenster JComponent JLabel/JTextField JList JTable

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.

18 / 29

```

Root-Fenster      JComponent      JLabel/JTextField      JList      JTable
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);
        ...
    }
}
19/29

```

```

Root-Fenster      JComponent      JLabel/JTextField      JList      JTable
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("");
    }
}
20/29

```

```

Root-Fenster      JComponent      JLabel/JTextField      JList      JTable
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.

21/29

```

Root-Fenster JComponent JLabel/JTextField JList **JTable**

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.setFillViewportHeight(true);
        this.add(new JScrollPane(myTable),BorderLayout.PAGE_START);
        this.pack();
        this.setVisible(true);
    }
}

```

22 / 29

Root-Fenster JComponent JLabel/JTextField JList **JTable**

Resultat



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

23 / 29

Root-Fenster JComponent JLabel/JTextField JList **JTable**

Eigene Tabellenmodelle

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

Hierin können folgende Methoden überschrieben werden:

<code>int getColumnCount()</code>	Zahl der Spalten
<code>int getRowCount()</code>	Zahl der Zeilen
<code>Object getValueAt(row,col)</code>	Element in Zeile <i>row</i> und Spalte <i>col</i>
<code>void setValueAt(obj, row, col)</code>	Setzt das Element in Zeile <i>row</i> und Spalte <i>col</i> auf <i>obj</i>

24 / 29

Beispiel für ein Tabellenmodell

Im folgenden wird ein Tabellenmodell implementiert, das für eine Menge von Zahlen $0 \dots n - 1$ 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.

25 / 29

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;
    }
}
```

26 / 29

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.setFillViewportHeight(true);
        applyButton.addActionListener(this);
        this.add(scrollPane, BorderLayout.PAGE_START);
        this.pack();
        this.setVisible(true);
    }
}
```

27 / 29

Root-Fenster JComponent JLabel/JTextField JList **JTable**

Die Klasse App (Forts.)

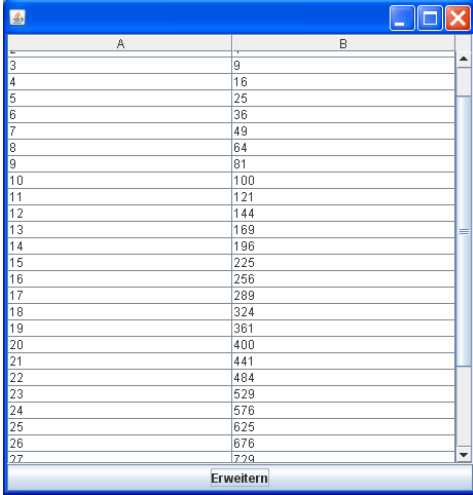
```
public void actionPerformed(ActionEvent event) {
    myModel.increase();
}

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

28 / 29

Root-Fenster JComponent JLabel/JTextField JList **JTable**

Resultat



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

Erweitern

29 / 29