

Swing



Swing

- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpi nner
 - JTextFi el d
 - JFormattedTextFi el d
 - JLi st
 - JTabl e
 - JTree
 - weitere Komponenten
- Zusammenfassung



Grundlegende Eigenschaften von Swing

- Leichtgewichtige Komponenten
 - nur Top-Level-Fenster, Dialoge und grafische Primitivoperationen auf Systemoperationen abgebildet
 - alles Weitere in Java
- Pluggable Look-and-Feel
 - Look-and-Feel nicht abhängig vom Betriebssystem
 - unterschiedliche Look-and-Feel verfügbar: Metal, Motif und Windows
 - Umschaltung des Look-and-Feel (auch zur Laufzeit)
- Model-View-Controller-Prinzip
 - neue Architektur der GUI-Komponenten nach dem Model-View-Controller-Prinzip



Pluggable Look and Feel

- Bestimmt Aussehen der Swing-Komponenten
- Auswahl im Programm

```
UI Manager. setLookAndFeel (
    UI Manager. getCrossPlatformLookAndFeelClassName()
    // oder UI Manager. getSystemLookAndFeelClassName()
    // oder "javax.swing.plaf.metal.MetalLookAndFeel"
    // oder "com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
    // oder "com.sun.java.swing.plaf.motif.MotifLookAndFeel"
);
```

- Austausch nach dem Start

```
UI Manager. setLookAndFeel (InfName);
SwingUtilities.updateComponentTreeUI (frame);
frame. pack();
```

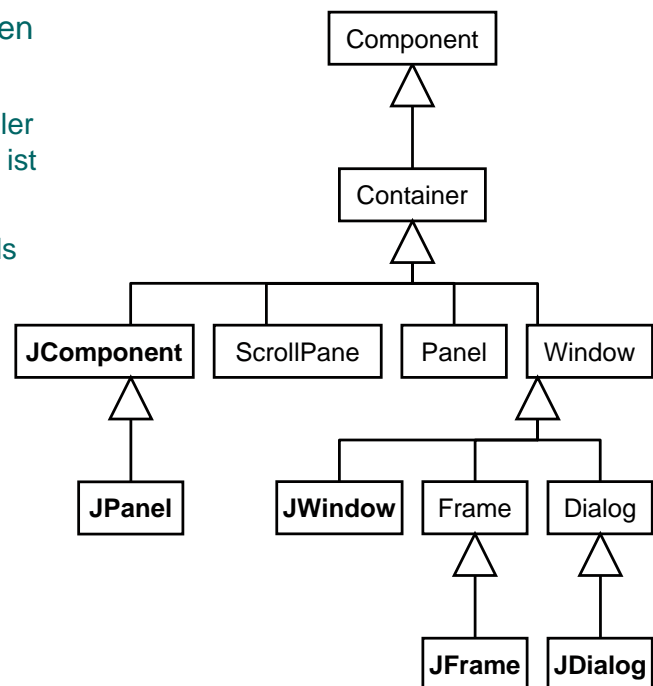
- Spezifikation auf der Kommandozeile

```
java -Dswing.defaultLookAndFeel=javax.swing.plaf.metal.MetalLookAndFeel MyApp
```



Einbettung und Swing in AWT

- Swing-Komponenten sind in die Hierarchie der AWT-Komponenten eingebettet
 - JComponent als Basisklasse aller leichtgewichtigen Komponenten ist von Container abgeleitet
 - JFrame, JDialog, JWindow als Top-Level-Windows erweitern Frame, Dialog, Window
- Damit werden grundlegende Mechanismen von AWT geerbt
 - Komponentenhierarchie
 - Ereignismechanismus



Top-Level-Windows in Swing

- JWindow**
 - rahmenloses Fenster
 - hat eine `ContentPane`, um Komponenten aufzunehmen

```
window.getContentPane().add(component)
```
- JFrame**
 - Typisch für Applikationsfenster
 - mit Rahmen, Systemmenü und Standardschaltfläche
- JDialog**
 - für modale und nicht-modale Dialoge
 - werden mit `Parent-Frame` erzeugt
 - hat wie `JWindow` eine `ContentPane`, um Komponenten aufzunehmen

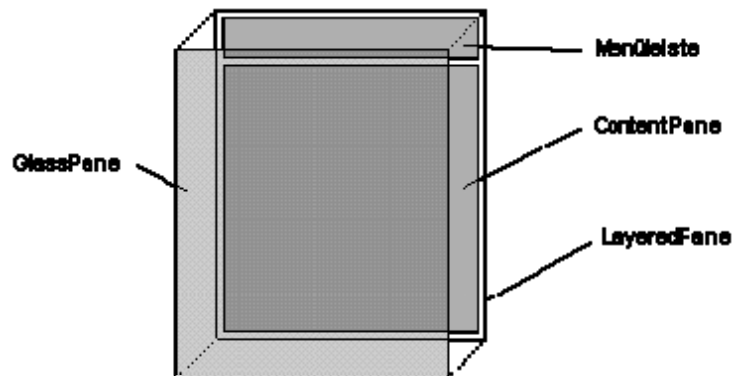


JFrame

JFrame ist aus mehreren Panes aufgebaut

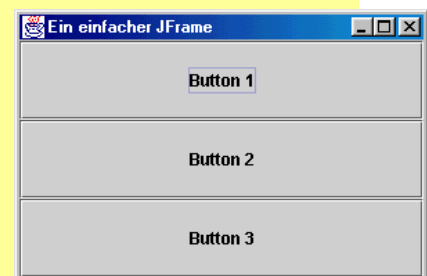
- RootPane: Wurzel, beinhaltet Border, Titel, LayeredPane
- LayeredPane: beinhaltet Menüleiste und ContentPane
- ContentPane: eigentliche Inhaltsbereich
- GlassPane: vorgelagerte, durchsichtige Pane für Effekte, die den ganzen Frame betreffen

```
JRootPane getRootPane()  
Container getContentPane()  
JLayeredPane getLayeredPane()  
Component getGlassPane()
```



Aufbau einer JFrame-Applikation

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.WindowConstants;  
  
public class Swing01_JFrameTest {  
    JFrame frame;  
    JPanel panel;  
  
    public Swing01_JFrameTest() {  
        frame = new JFrame("Ein einfacher JFrame");  
        frame.addWindowListener(new WindowAdapter() {  
            public void windowClosing(WindowEvent event) {  
                event.getWindow().setVisible(false);  
                event.getWindow().dispose();  
                System.exit(0);  
            }  
        });  
        Container contentPane = getContentPane();  
        contentPane.setLayout(new GridLayout(3, 1));  
        contentPane.add(new JButton("Button 1"));  
        contentPane.add(new JButton("Button 2"));  
        contentPane.add(new JButton("Button 3"));  
        frame.setLocation(100, 100);  
        frame.setSize(400, 400);  
        frame.setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        Swing01_JFrameTest app =  
            new Swing01_JFrameTest();  
    }  
}
```



Aufbau einer JFrame-Applikation

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
```

```
public class Swing01_JFrameTest {
```

```
    JFrame frame;
    JPanel panel;
```

```
    public Swing01_JFrameTest() {
```

```
        frame = new JFrame("Ein einfacher JFrame");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        Container contentPane = getContentPane();
        contentPane.setLayout(new GridLayout(3, 1));
        contentPane.add(new JButton("Button 1"));
        contentPane.add(new JButton("Button 2"));
        contentPane.add(new JButton("Button 3"));
```

```
        frame.setLocation(100, 100);
        frame.setSize(400, 400);
        frame.setVisible(true);
```

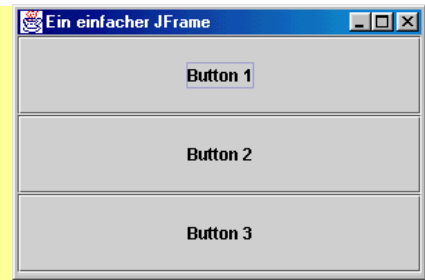
```
    }
```

```
    public static void main(String[] args) {
```

```
        Swing01_JFrameTest app =
            new Swing01_JFrameTest();
```

```
    }
```

```
}
```



WindowConstants

- DO_NOTHING_ON_CLOSE
- HIDE_ON_CLOSE
- DISPOSE_ON_CLOSE
- EXIT_ON_CLOSE



Swing

- Einführung
- **Painting bei Swing**
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung



Painting in Swing

- Painting bei Swing in AWT-Painting eingebettet: Aufruf von

```
public void update(Graphics g)
```

und

```
public void paint(Graphics g)
```

Aber:

- paint bei JComponent folgend implementiert

```
public void paint(Graphics g) {  
    ...  
    paintComponent(co);  
    paintBorder(co);  
    ...  
    paintChildren(co);  
    ...  
}
```

Zeichnen der Komponente
Zeichnen eines Rands

Zeichnen der Kinder

- Üblicherweise wird nur paintComponent(Graphics g) überschrieben

```
public class MyJComponent  
protected void paintComponent(Graphics g) {  
    // my custom painting ...  
}
```



Painting in Swing

- Graphics2D

- ist speziellens Graphics-Klasse
- mächtige Graphikfunktionen

- Plus viele weitere Features:

- Double-Buffering bereits implementiert und bei jeder JComponent einschaltbar

```
public void setDoubleBuffered(boolean db)
```

- Verschiedene Borders

```
void setBorder(Border b)
```

- TooltipText

```
void setToolTipText(String text)
```

- Transparenter Hintergrund

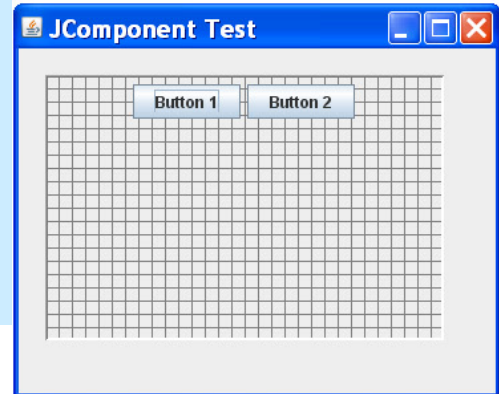
```
void setOpaque(boolean opaque)
```

- ...



Beispiel Painting in Swing

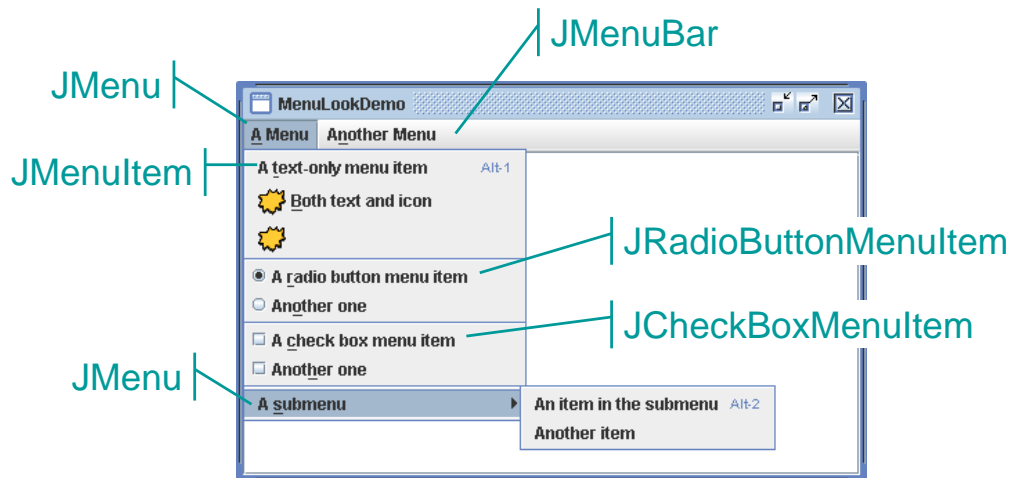
```
class GridJComponent extends JComponent {  
  
    public GridJComponent() {  
        setLayout(new FlowLayout());  
        JButton b1 = new JButton("Button 1");  
        JButton b2 = new JButton("Button 2");  
        setBorder(BorderFactory.createLoweredBevelBorder());  
        add(b1);  
        add(b2);  
    }  
  
    protected void paintComponent(Graphics g) {  
        int width = getSize().width;  
        int height = getSize().height;  
        g.setColor(Color.gray);  
        for (int i = 0; i < width; i += 10) {  
            g.drawLine(i, 0, i, height);  
        }  
        for (int i = 0; i < height; i += 10) {  
            g.drawLine(0, i, width, i);  
        }  
    }  
}
```



Swing

- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung





Menüs und Untermenüs

- **Menüleiste**
 - Instanz von `JMenuBar`
 - Setzen mittels `setJMenuBar` des `JFrame`
- **Menü**
 - Instanz von `JMenu` (erweitert `JMenuItem`)
 - Hinzufügen zu Menüleiste oder Menü mit Methode `add`
 - Einfügen einer Trennlinie mit `addSeparator`
- **Menüeintrag**
 - Instanz von `JMenuItem`
 - Hinzufügen zu Menü mittels `add`
 - Kann Symbol und Tastenkürzel besitzen
 - Verfügbare Varianten:
 - `JCheckBoxMenuItem`
 - `JRadioButtonMenuItem`



Eigenschaften eines Menüeintrages

- Mnemonik
 - Unterstrichener Buchstabe für Tastaturnavigation im Menü
 - Setzen mit `setMnemonic`
 - Konstanten von `KeyEvent` für Tasten verwenden
- Tastenkombination
 - Erscheint rechts neben Menütext
 - `KeyStroke.getKeyStroke(int keyCode, int modifiers)`
 - Modifizierer: `KeyEvent.ALT_MASK`, `KeyEvent.CTRL_MASK`, `KeyEvent.SHIFT_MASK`
 - Setzen mittels `setAccelerator`
- Symbol
 - Laden von Datei mittels `ImageIcon`-Konstruktor
 - Setzen mit `setIcon`



Beispiel: Menüs

```
// create and set menu bar
MenuBar = new JMenuBar();
setMenuBar(menuBar);

// create menu
menu = new JMenu("A Menu");
menu.setMnemonic(KeyEvent.VK_A);
menuBar.add(menu);

// create menu item
ImageIcon icon = new ImageIcon("images/middle.gif");
MenuItem = new JMenuItem("Both text and icon", icon);
menuItem.setMnemonic(KeyEvent.VK_B);
menuItem.setAccelerator(KeyStroke.getKeyStroke(
    KeyEvent.VK_B, KeyEvent.CTRL_MASK));
menuItem.addActionListener(this);
menu.add(menuItem);

// add separator
menu.addSeparator();
```



Arbeiten mit Key-Codes und KeyStroke

- `java.awt.event.KeyEvent`-Klasse enthält Konstante für alle Tasten

```
VK_0, VK_1, ...  
VK_A, VK_B, ... VK_Z,  
VK_SPACE, VK_TAB, ...  
VK_F1, VK_F2, ...  
VK_SHIFT, VK_CONTROL, VK_ALT,
```

- `javax.swing.KeyStroke`-Klasse abstrahiert von Tasten und repräsentiert "Tastenaktion"

Erzeugen von KeyStroke aus Taste + Modifiers

```
KeyStroke.getKeyStroke(char keyChar)  
KeyStroke.getKeyStroke(char keyChar, int modifiers )  
KeyStroke.getKeyStroke(String s)  
KeyStroke.getKeyStrokeForEvent(KeyEvent anEvent)
```

wobei `modifiers` als Kombination (+) folgender Konstanten der Klasse `java.awt.event.InputEvent`-Klasse gebildet wird

```
SHIFT_DOWN_MASK, CTRL_DOWN_MASK, META_DOWN_MASK,  
ALT_DOWN_MASK, ALT_GRAPH_DOWN_MASK
```



PopupMenu

- `PopupMenu` müssen explizit geöffnet werden

```
JMenuItem copy = new JMenuItem("Kopieren");  
copy.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Kopieren gewählt.");  
    }  
});  
  
JMenuItem paste = new JMenuItem("Einfügen");  
paste.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("Einfügen gewählt.");  
    }  
});  
popup = new JPopupMenu();  
popup.add(copy);  
popup.add(paste);  
  
getContentPane().addMouseListener(new MouseAdapter() {  
    public void mousePressed(MouseEvent event) {  
        if (event.isPopupTrigger())  
            popup.show(event.getComponent(),  
                event.getX(), event.getY());  
    }  
});
```



Actions

- Action-Objekte realisieren Aktionen unabhängig von GUI-Elementen
- können an mehrere GUI-Elemente gebunden werden
- Klasse `Action`
 - implementiert `ActionListener`
 - hat `enabled`-State
 - unterschiedliche Werte mit `getValue` und `putValue` verwalten

```
interface Action extends java.awt.event.ActionListener {  
    void actionPerformed(ActionEvent e)  
    boolean isEnabled()  
    void setEnabled(boolean b)  
    Object getValue(String key)  
    void putValue(String key, Object value)  
    void addPropertyChangeListener(PropertyChangeListener listener)  
    void removePropertyChangeListener(PropertyChangeListener listener)  
}
```

- Klasse `AbstractAction` stellt Basisimplementierung bereit



Actions (2)

- Mittels
 - `getValue(String key)`
 - `putValue(String key, Object value)`

können Eigenschaften der Aktion verwaltet werden. Z.B.:

- `NAME`: Name der Aktion
- `MNEMONIC_KEY`: Setzen der Mnemonik
- `SHORT_DESCRIPTION`: Benutzt für Tooltip etc.
- `LONG_DESCRIPTION`: Zum Beispiel für kontextsensitive Hilfe
- `ACCELERATOR_KEY`: Tastenkombination
- `SMALL_ICON`: Ablegen eines ImageIcons
- beliebige andere

```
putValue(Action.NAME, name);  
putValue(Action.SMALL_ICON, icon);  
putValue(Action.SHORT_DESCRIPTION,  
    "Ändern der Grundflächenfarbe in " + name.toLowerCase());  
putValue("color", c);
```



Actions (3)

- Action-Objekte werden an GUI-Komponenten gebunden
 - an Buttons
 - an MenuItem
 - an PopupMenus
 - an KeyStrokes

```
// Button
add(new JButton(sampleAction));

// Menu
menu.add(new MenuItem(sampleAction));

// Popup
popupMenu.add(sampleAction);

// KeyStroke siehe nächste Folie
```



Zuordnung von KeyStrokes zu Actions

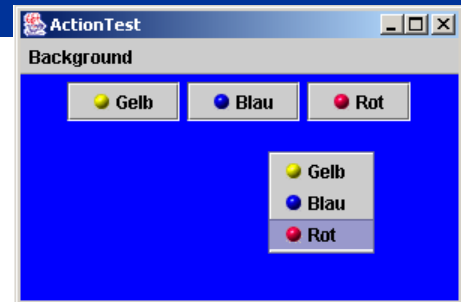
- Komponenten verwalten:
 - InputMap-Tabellen: Zuordnung von KeyStroke-Objekten zu Aktionsnamen
`inputMap.put(KeyStroke.getKeyStroke("ctrl S"), "panel.sample");`
 - ActionMap-Tabelle: Zuordnung von Aktionsnamen zu Action-Objekten
`ActionMap actionMap = component.getActionMap();`
`actionMap.put("panel.sample", sampleAction);`
- Es gibt 3 InputMap-Tabellen (die ausgehend von der Komponente mit Fokus nacheinander untersucht werden)
 - WHEN_FOCUSED: verwendet, wenn Komponente Fokus hat
`InputMap inputMap = component.getInputMap(JComponent.WHEN_FOCUSED);`
 - WHEN_ANCESTOR_OF_FOCUSED_COMPONENT: verwendet, wenn eine Unterkomponente den Fokus hat
`InputMap inputMap =`
`component.getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);`
 - WHEN_IN_FOCUSED_WINDOW: verwendet, wenn die Komponente im selben Fenster als Komponente mit Fokus



Beispiel: ColorAction (1)

Ändern die Hintergrundfarbe mit ColorActions

- ColorAction implementieren



```
public class ColorAction extends AbstractAction {
    public ColorAction(String name, ImageIcon icon, Color c) {
        putValue(Action.NAME, name);
        putValue(Action.SMALL_ICON, icon);
        putValue(Action.SHORT_DESCRIPTION,
            "Ändern der Grundflächenfarbe in " + name.toLowerCase());
        putValue("color", c);
    }

    public void actionPerformed(ActionEvent event) {
        Color c = (Color) getValue("color");
        setBackground(c);
        repaint();
    }
}
```



Beispiel: ColorAction (2)

- Panel mit ColorButtons implementieren
 - Action-Objekte erzeugen

```
class ActionPanel extends JPanel {
    Action yellowAction, blueAction, redAction;
    public ActionPanel() {
        yellowAction = new ColorAction("Gelb", new ImageIcon(
            "yellow-ball.gif"), Color.YELLOW);
        blueAction = new ColorAction("Blau", new ImageIcon(
            "blue-ball.gif"), Color.BLUE);
        redAction = new ColorAction("Rot", new ImageIcon(
            "red-ball.gif"), Color.RED);
    }
}
```

- Buttons mit Actions erzeugen und anfügen

```
add(new JButton(yellowAction));
add(new JButton(blueAction));
add(new JButton(redAction));
```



Beispiel: ColorAction (3)

- KeyStrokes mit Aktionsnamen verbinden

```
InputMap imap = getInputMap(  
    JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);  
  
imap.put(KeyStroke.getKeyStroke("ctrl G"), "panel.yellow");  
imap.put(KeyStroke.getKeyStroke("ctrl B"), "panel.blue");  
imap.put(KeyStroke.getKeyStroke("ctrl R"), "panel.red");
```

- Aktionsnamen mit Action-Objekten verbinden

```
ActionMap amap = getActionMap();  
amap.put("panel.yellow", yellowAction);  
amap.put("panel.blue", blueAction);  
amap.put("panel.red", redAction);  
}
```



Beispiel: ColorAction (4)

- PopUpMenu definieren

```
// Kontextmenü erzeugen  
popUp = new JPopupMenu();  
popUp.add(yellowAction);  
popUp.add(blueAction);  
popUp.add(redAction);  
  
this.addMouseListener(new MouseAdapter() {  
    public void mousePressed(MouseEvent e) {  
        if (e.getButton() == MouseEvent.BUTTON3)  
            popUp.show(e.getComponent(), e.getX(), e.getY());  
    }  
});
```

- Im Frame Menu mit Actions definieren

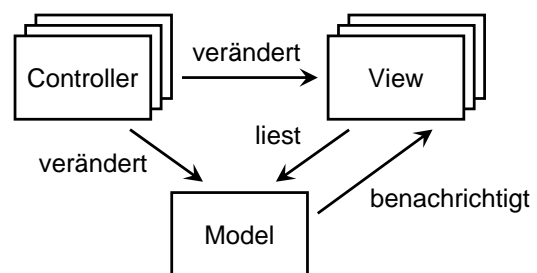
```
JMenuBar menuBar = new JMenuBar();  
this.setJMenuBar(menuBar);  
JMenu bgMenu = new JMenu("Background");  
menuBar.add(bgMenu);  
bgMenu.add(new JMenuItem(panel.yellowAction));  
bgMenu.add(new JMenuItem(panel.blueAction));  
bgMenu.add(new JMenuItem(panel.redAction));
```



- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung

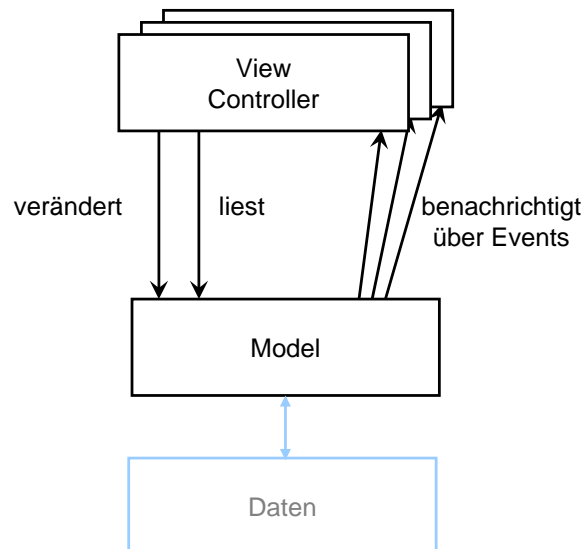


- Entkoppelung von Model, View und Controller
 - Kombiniertes Muster
 - Stammt aus Smalltalk
 - Flexibilität
 - Wiederverwendbarkeit
- Model
 - Daten, die das Programm verwaltet
 - Sorgt dafür, dass alle Views nachgeführt werden
- View
 - Darstellung der Daten
 - Zu jeder View gehört ein Controller
- Controller
 - Interpretation von Tastatureingaben und Mausklicks



MVC in Swing

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



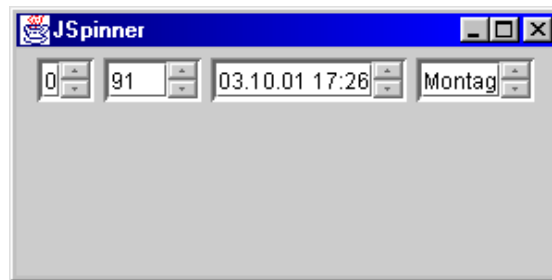
Swing

- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung

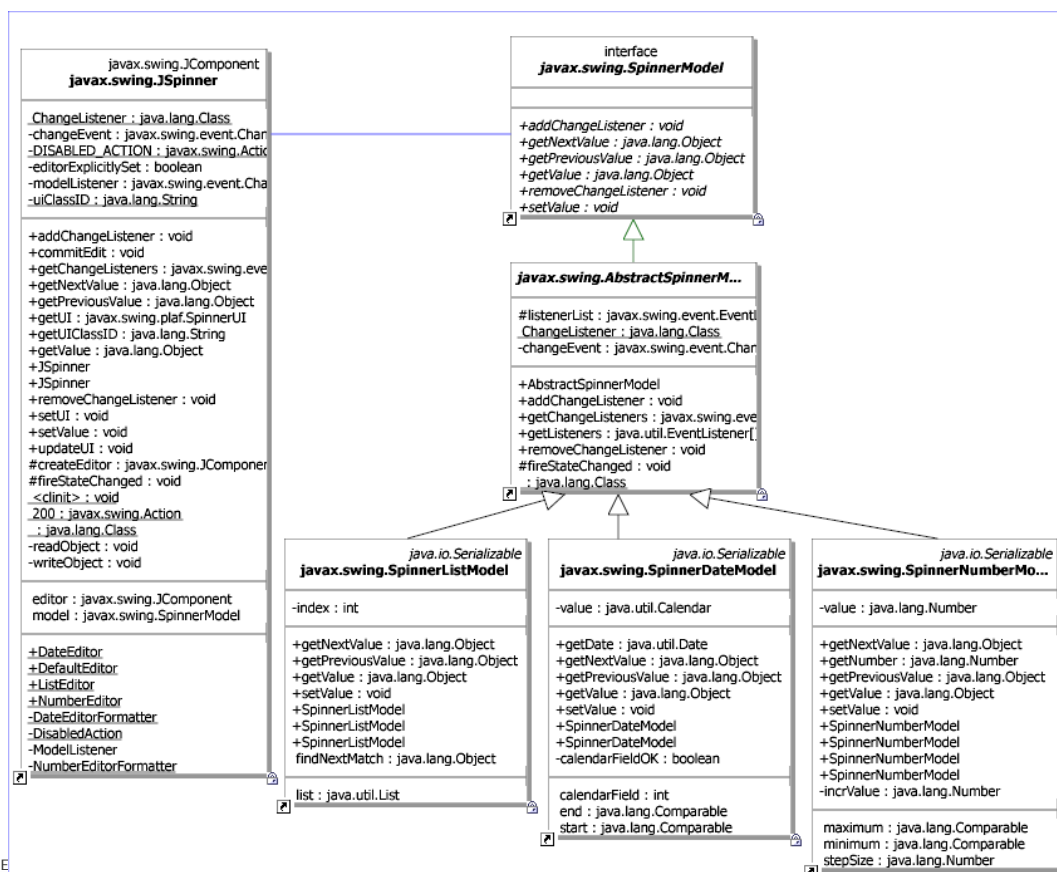


JSpinner

- 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);
}
```

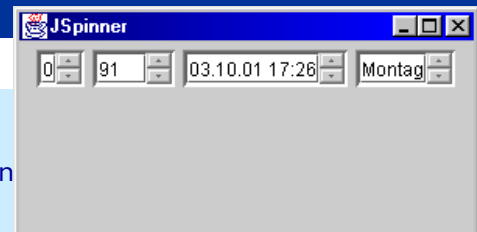
```
public class ChangeEvent extends EventObject {
    public ChangeEvent(Object source) {
        super(source);
    }
}
```



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, initial 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());
            }
        });
        ...
    }
}
```



- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - **JTextField**
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung



JTextField

- JTextField ist eine Komponente für die einfache Eingabe von kurzem Text
- Hat Model vom Typ Document
- Folgende Methoden stehen zur Verfügung
 - Konstruktoren mit Angabe der Breite und eines initialen Texts

```
public JTextField(int columns)
public JTextField(String text)
public JTextField(String text, int columns)
```

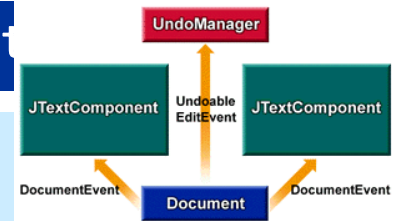
- Zugriff auf String-Wert

```
public String getText()
public void setText(String t)
public String getText(int offs, int len)
```



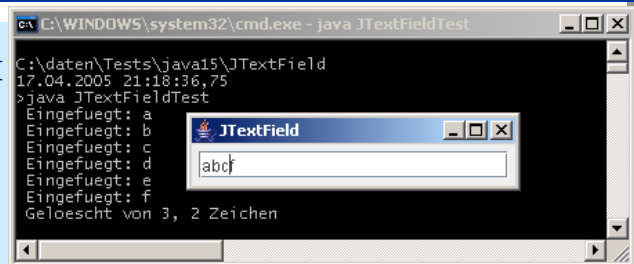
Interface javax.swing.text.Document

```
public interface Document {  
  
    public int getLength();  
  
    public void addDocumentListener(DocumentListener listener);  
    public void removeDocumentListener(DocumentListener listener);  
  
    public void addUndoableEditListener(UndoableEditListener listener);  
    public void removeUndoableEditListener(UndoableEditListener listener);  
  
    public Object getProperty(Object key);  
    public void putProperty(Object key, Object value);  
  
    public void remove(int offs, int len) throws BadLocationException;  
    public void insertString(int offset, String str, AttributeSet a)  
        throws BadLocationException;  
  
    public String getText(int offset, int length) throws BadLocationException;  
    public void getText(int offset, int length, Segment txt) throws BadLocationException;  
  
    public Position getStartPosition();  
    public Position getEndPosition();  
    public Position createPosition(int offs) throws BadLocationException;  
  
    public Element[] getRootElement();  
    public Element getDefaultRootElement();  
  
    public void render(Runnable r);  
    public static final String StreamDescriptionProperty = "stream";  
    public static final String TitleProperty = "title";  
}
```



Beispiel JTextField: Ausgabe der Editieroperationen

```
import ...;  
public class JTextFieldTest extends JFrame {  
    JTextField field;  
    public JTextFieldTest() {  
        super("JTextField");  
        Container cp = getContentPane();  
        cp.setLayout(new FlowLayout());  
        field = new JTextField(20);  
        cp.add(field);  
        field.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.out.println("Neue Eingabe: " + field.getText());  
            }  
        });  
        field.getDocument().addDocumentListener(new DocumentListener() {  
            public void insertUpdate(DocumentEvent e) {  
                try {  
                    System.out.println("Eingefuegt: " +  
                        field.getDocument().getText(e.getOffset(), e.getLength()));  
                } catch (BadLocationException exc) { exc.printStackTrace(); }  
            }  
            public void removeUpdate(DocumentEvent e) {  
                System.out.println("Geloescht von " + e.getOffset() + ", " +  
                    e.getLength() + " Zeichen");  
            }  
            public void changedUpdate(DocumentEvent e) {}  
        });  
        pack();  
    }  
}
```

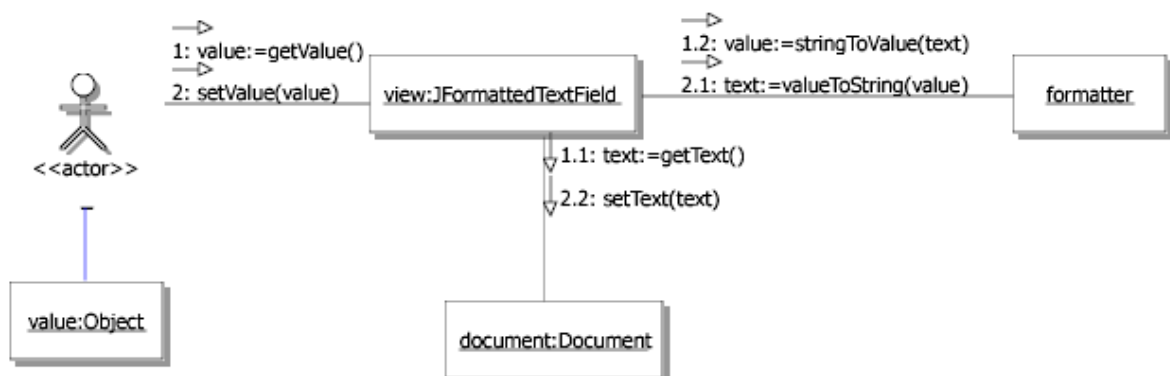


- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung



JFormattedTextField

- JFormattedTextField erlaubt die formatierte Eingabe von beliebigen Werten
 - Formatter wird verwendet, um zwischen String-Repräsentation und Objektwert zu konvertieren
 - Verifikation, dass korrekte Zeichenkette eingegeben wurde
 - Unterschiedliche Reaktionen bei ungültigen Eingaben
 - Filtern von ungültigen Eingaben



Erzeugen von JFormattedTextFie ld

Beim Anlegen eines JFormattedTextFie ld muss Formatter definiert werden

- Erzeugen mit Format-Objekten: Formatter wird auf Basis des Format-Objekts erstellt

```
JFormattedTextFie ld i ntFie ld = new JFormattedTextFie ld(  
    NumberFormat.getIntegerInstance());  
  
JFormattedTextFie ld currencyFie ld = new JFormattedTextFie ld(  
    NumberFormat.getCurrencyInstance());  
  
JFormattedTextFie ld dateFie ld = new JFormattedTextFie ld(  
    DateFormat.getDateInstance());  
  
JFormattedTextFie ld dateFie ld = new JFormattedTextFie ld(  
    DateFormat.getDateInstance(DateFormat.SHORT));
```

(siehe dazu NumberFormat und DateFormat)



Erzeugen von JFormattedTextFie ld (Forts.)

- Mit DefaultFormatter

- stringValue arbeitet mit Objekt-Konstruktor mit String-Parameter
- valueToString ruft toString auf

```
DefaultFormatter formatter = new DefaultFormatter();  
JFormattedTextFie ld urlFie ld = new JFormattedTextFie ld(formatter);  
urlFie ld.setVal ue(new URL("http://java.sun.com"));
```

- Mit MaskFormatter und Definition einer EingabeMaske

```
MaskFormatter formatter = new MaskFormatter("###-##-####");  
formatter.setPlaceholderCharacter('0');  
JFormattedTextFie ld ssnFie ld = new JFormattedTextFie ld(formatter);  
ssnFie ld.setVal ue("078-05-1120");
```

Wobei in der Maske die folgenden Zeichen verwendet werden

- # eine Ziffer
- ? ein Buchstabe
- U ein Buchstabe, konvertiert in einen Großbuchstaben
- L ein Buchstabe, konvertiert in einen Kleinbuchstaben
- A ein Buchstabe oder eine Ziffer
- H eine Hexadezimalziffer [0-9A-Fa-f]
- * beliebiges Zeichen



Schreiben eines eigenen Formatters

- Ableiten von DefaultFormatter
- Überschreiben der Methoden `valueOfToString` und `stringToValue`

Beispiel: Formatter für IP-Adressen

```
class IPAddressFormatter extends DefaultFormatter
{
    public String valueToString(Object value)
        throws ParseException
    {
        if (!(value instanceof byte[]))
            throw new ParseException("Kein byte[]", 0);
        byte[] a = (byte[])value;
        if (a.length != 4)
            throw new ParseException("Länge != 4", 0);
        StringBuffer buffer = new StringBuffer();
        for (int i = 0; i < 4; i++)
        {
            int b = a[i];
            if (b < 0) b += 256;
            buffer.append(String.valueOf(b));
            if (i < 3) buffer.append('.');
        }
        return buffer.toString();
    }
}
```



Schreiben eines eigenen Formatters (Forts.)

```
...
public Object stringToValue(String text) throws ParseException
{
    StringTokenizer tokenizer = new StringTokenizer(text, ".");
    byte[] a = new byte[4];
    for (int i = 0; i < 4; i++)
    {
        int b = 0;
        try
        {
            b = Integer.parseInt(tokenizer.nextToken());
        }
        catch (NumberFormatException e)
        {
            throw new ParseException("Keine Ganzzahl", 0);
        }
        if (b < 0 || b >= 256)
            throw new ParseException("Byte außerhalb des Bereichs", 0);
        a[i] = (byte)b;
    }
    return a;
}
```



Verifikation

- Bei Abgabe des Fokus wird geprüft, ob die aktuelle Zeichenkette gültig ist
 - kann der Formatter die Zeichenkette in ein Objekt umwandeln dann gültig, sonst ungültig
- Mit Methode
`boolean isValid()`
kann man das auch explizit prüfen
- Bei ungültiger Eingabe gibt sich ein entsprechendes Verhalten, das man mit
`void setFocusLostBehavior(int behavior)`
einstellen kann



Verhalten bei Abgabe des Focus

- **COMMIT_OR_REVERT** (Standardverhalten)
 - Wenn gültig, den Wert übernehmen (Methode `commitEdit()`),
 - Wenn ungültig, den letzten Wert ins Ausgabefeld zurückschreiben
- **COMMIT**
 - Wenn gültig wie oben
 - Wenn ungültig, den letzten Wert und den Inhalt des Ausgabefeldes behalten
- **PERSIST**
 - Obwohl gültig, den Wert nicht übernehmen, sondern erst bei expliziten Aufruf von `commitEdit()`, `setValue()` oder `setText()` übernehmen
- **REVERT**
 - Den Wert nicht übernehmen und auch das Ausgabefeld mit dem alten Wert überschreiben



JFormattedTextFie ld Weiteres

Neben Formatter kann für JFormattedTextFie ld folgendes definiert werden

- DocumentFie lter:

- Erlauben die Filterung von Eingabezeichen

```
JFormattedTextFie ld intFie ld3 = new JFormattedTextFie ld(  
    new Internati onal Formatter(NumberFormat. getIntegerInstance()) {  
        protected DocumentFie lter getDocumentFie lter() {  
            return filter;  
        }  
        private DocumentFie lter filter = new IntFie lter();  
    });
```

- InputVeri fier:

- Explizite Kodierung der Gültigkeitsprüfung in eigenen Klasse

```
intFie ld4. setInputVeri fier(new FormattedTextFie ldVeri fier());
```

```
class FormattedTextFie ldVeri fier extends InputVeri fier{  
    public boolean verif y(JComponent component){  
        JFormattedTextFie ld fie ld = (JFormattedTextFie ld)component;  
        return fie ld. isEdi tVal id();  
    }  
}
```



Swing

- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpi nner
 - JTextFie ld
 - JFormattedTextFie ld
 - JLi st
 - JTabl e
 - JTree
 - weitere Komponenten
- Zusammenfassung

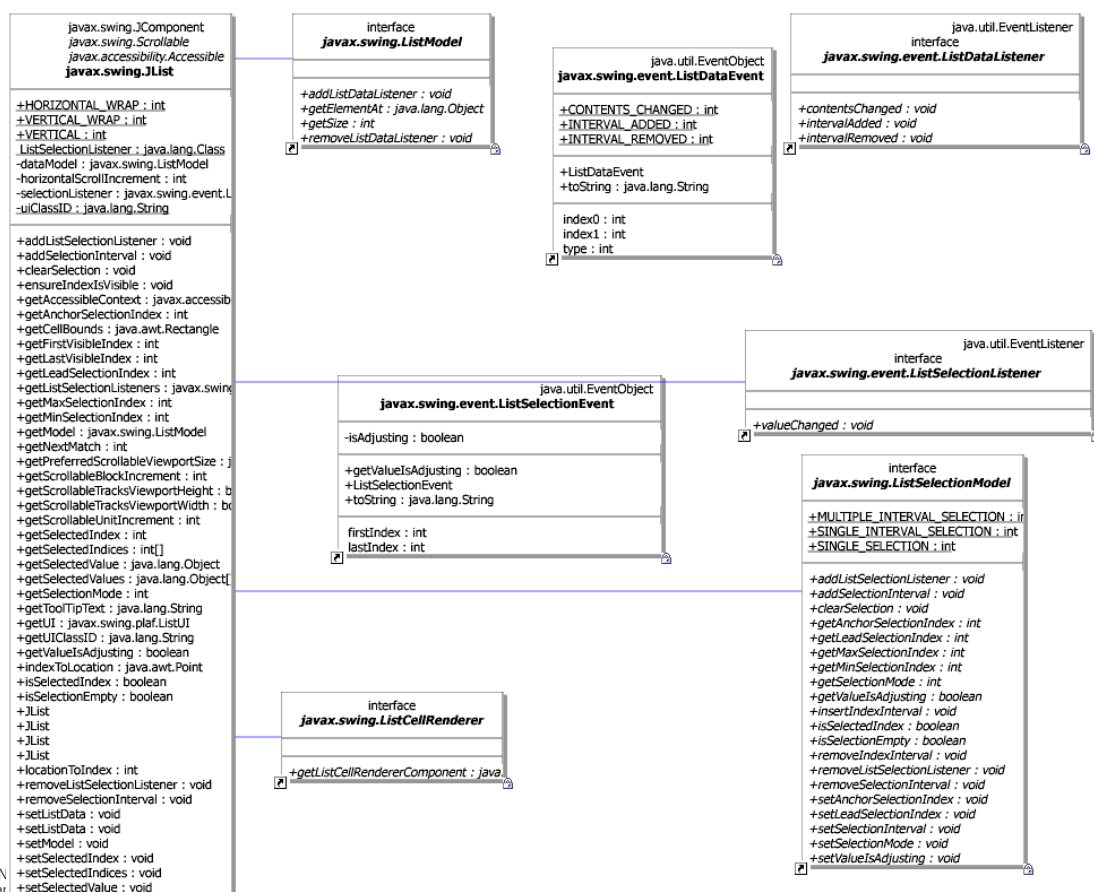


JList

- 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



JList Klassendiagramm



Interface ListModel

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

```
public interface ListModel {
    int getSize();
    Object 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; }
}
```



Konstruktion von JList-Komponenten

- Konstruktion von JList möglich mit

- Object-Array

```
private static final String[] WDAYs = {"Montag", "Dienstag",
    "Mittwoch", "Donnerstag", "Freitag", "Samstag", "Sonntag"};
```

```
public JListTest() {
    Container cp = getContentPane();
    JList list1 = new JList(WDAYs);
    JScrollPane scrollPane = new JScrollPane(list1);
    cp.add(list1);
    ...
}
```

- Vector

```
private static Vector v = new Vector();
```

```
public JListTest() {
    v.add("gestern"); v.add("heute"); v.add("morgen"); ...
    JList list2 = new JList(v);
    ...
}
```



Konstruktion von JList-Komponenten (Forts.)

- ListModel (Beispiel Generieren von Integers)

```
public ListTest() {
    ...

    list1 = new JList(new IntListModel(100));
    ...
}

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

    public int getSize() {
        return n;
    }

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

    private int n;
}
```



Klasse DefaultListModel

- Arbeitet wie Vector
- Plus ListModel Interface

Siehe Beispiel PointsFrameWithJList (Download von LVA-Homepage)

- Statt PointsList-Datenmodell DefaultListModel
- Statt PropertyChangeEvents ListDataEvents



Selektion

- Elemente in `JList` kann man selektieren und Selektion abfragen
- `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 getValueAdjusting() { ... }  
    ...  
}
```

- Es gibt unterschiedliche Selektionsmodi, die man über die `JList`-Methode
`void setSelectionMode(int mode)`
setzen kann; mögliche Modi sind (Konstante des Interfaces `ListSelectionMode`)
 - `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 getListCellRendererComponent(...)
```

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

```
public interface ListCellRenderer {  
    Component getListCellRendererComponent(  
        JList list,  
        Object value,  
        int index,  
        boolean isSelected,  
        boolean cellHasFocus);  
}
```

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

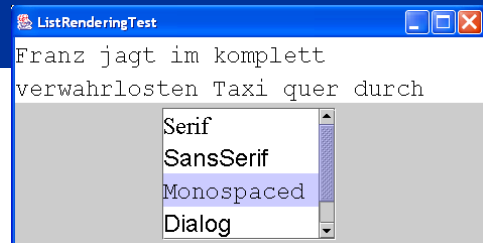
- Methode
`void setCellRenderer(ListCellRenderer)`

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 extends JLabel implements ListCellRenderer {
    public FontCellRenderer() {
        setOpaque(true); // must do this for background to show up
    }

    public Component getListCellRendererComponent(
        JList list, Object value, int index, boolean isSelected,
        boolean cellHasFocus) {
        if (isSelected) {
            setForeground(list.getSelectedForeground());
            setBackground(list.getSelectedBackground());
        } else {
            setForeground(list.getForeground());
            setBackground(list.getBackground());
        }
        Font font = (Font) value;
        setFont(font);
        setText(font.getFamily());
        return this;
    }
}
```

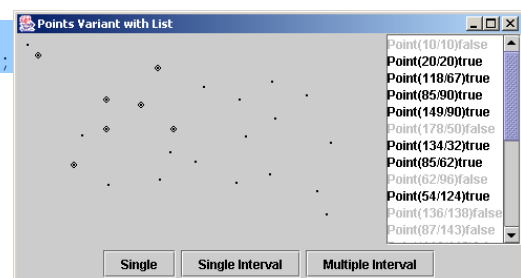


Beispiel PointsFrameWithJList (1)

```
class PointCellRenderer implements ListCellRenderer {
    public Component getListCellRendererComponent(
        final JList list,
        final Object value,
        final int index,
        final boolean isSelected,
        final boolean cellHasFocus) {

        SelectablePoint p = (SelectablePoint) value;
        JLabel label = new JLabel("Point(" + p.x + "/" + p.y + ")");
        if (isSelected) {
            label.setForeground(Color.BLACK);
        } else {
            label.setForeground(Color.LIGHT_GRAY);
        }
        return label;
    }
}
```

```
list1 = new JList(points);
list1.setCellRenderer(new PointCellRenderer());
```



Beispiel PointsFrameWithJList (2)

```
list1 = new JList(points);
list1.setCellRenderer(new PointCellRenderer());
panel.add(list1);
list1.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
        Object[] selectedPoints = list1.getSelectedValues();
        markPointsSelected(selectedPoints);
    }
});
```

```
protected void markPointsSelected(Object[] selectedPoints) {
    Enumeration pointIterator = points.elements();
    while (pointIterator.hasMoreElements()) {
        SelectablePoint p = (SelectablePoint)pointIterator.nextElement();
        p.selected = false;
    }
    for (int i = 0; i < selectedPoints.length; i++) {
        SelectablePoint p = (SelectablePoint)selectedPoints[i];
        p.selected = true;
    }
    this.getContentPane().repaint();
}
```



Swing

- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung

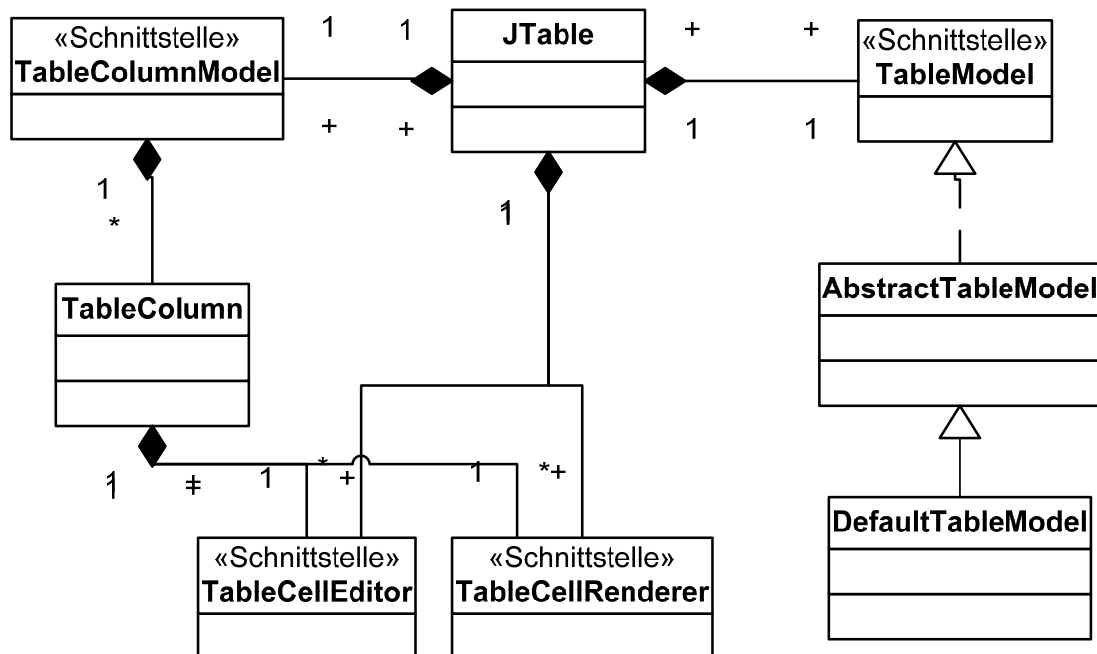


JTable

- 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



Aufbau JTable



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() { ... }
}
```



Erzeugen von JTable-Komponenten

- Konstruktion von JTable möglich mit
 - 2-dimensionalen Object-Array
 - + Spaltennamen

Planet	Radius	Monde	Gasförmig	Farbe
Merkur	2440.0	0	false	java.awt.Color[=255...
Venus	6052.0	0	false	java.awt.Color[=255...
Erde	6378.0	1	false	java.awt.Color[=0,g...
Mars	3397.0	2	false	java.awt.Color[=255...
Jupiter	71492.0	16	true	java.awt.Color[=255...
Saturn	60268.0	18	true	java.awt.Color[=255...
Uranus	25559.0	17	true	java.awt.Color[=0,g...
Neptun	24766.0	8	true	java.awt.Color[=0,g...
Pluto	1137.0	1	false	java.awt.Color[=0,g...

```
public PlanetTableFrame() {
    JTable table = new JTable(cells, columnNames);
    ...
}
private Object[][] cells =
    { { "Merkur", new Double(2440), new Integer(0), Boolean.FALSE, Color.yellow }, {
      "Venus", new Double(6052), new Integer(0), Boolean.FALSE, Color.yellow }, {
      "Erde", new Double(6378), new Integer(1), Boolean.FALSE, Color.blue }, {
      "Mars", new Double(3397), new Integer(2), Boolean.FALSE, Color.red }, {
      "Jupiter", new Double(71492), new Integer(16), Boolean.TRUE, Color.orange }, {
      "Saturn", new Double(60268), new Integer(18), Boolean.TRUE, Color.orange }, {
      "Uranus", new Double(25559), new Integer(17), Boolean.TRUE, Color.blue }, {
      "Neptun", new Double(24766), new Integer(8), Boolean.TRUE, Color.blue }, {
      "Pluto", new Double(1137), new Integer(1), Boolean.FALSE, Color.black }
    };

private String[] columnNames = { "Planet", "Radius", "Monde", "Gasförmig", "Farbe" };
}
```



Erzeugen mit TableModel : Bsp. InvestmentTable

- Implementierung von TableModel stellt Datenzugriff bereit und liefert die Informationen

```

...
TableModel model = new InvestmentTableModel(30, 5, 10);
JTable table = new JTable(model);
getContentPane().add(new JScrollPane(table), "Center");

class InvestmentTableModel extends AbstractTableModel {
    public InvestmentTableModel(int y, int r1, int r2) { ... }

    public int getRowCount() {
        return years;
    }

    public int getColumnCount() { ... }

    public Object getValueAt(int r, int c) {
        double rate = (c + minRate) / 100.0;
        int nperiods = r;
        double futureBalance = INITIAL_BALANCE * Math.pow(1 + rate, nperiods);
        return NumberFormat.getCurrencyInstance().format(futureBalance);
    }

    public String getColumnName(int c) {
        double rate = (c + minRate) / 100.0;
        return NumberFormat.getPercentInstance().format(rate);
    }

    private int years;
    private int minRate;
    private int maxRate;
}
    
```

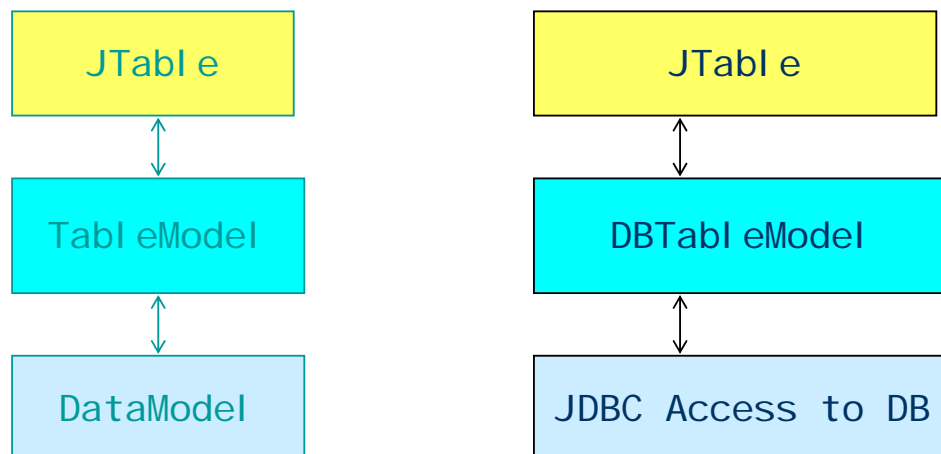
Berechnung des Kapitals für jedes Jahr bei unterschiedlichen Zinsen!!

	5%	6%	7%	8%	9%	10%
100.000,00 €	100.000,00 €	100.000,00 €	100.000,00 €	100.000,00 €	100.000,00 €	100.000,00 €
105.000,00 €	106.000,00 €	107.000,00 €	108.000,00 €	109.000,00 €	110.000,00 €	110.000,00 €
110.250,00 €	112.360,00 €	114.490,00 €	116.640,00 €	118.810,00 €	121.000,00 €	121.000,00 €
115.762,50 €	119.101,60 €	122.504,30 €	125.971,20 €	129.502,90 €	133.100,00 €	133.100,00 €
121.550,63 €	126.247,70 €	131.079,60 €	136.048,90 €	141.158,16 €	146.410,00 €	146.410,00 €
127.628,16 €	133.822,56 €	140.255,17 €	146.932,81 €	153.862,40 €	161.051,00 €	161.051,00 €
134.009,56 €	141.851,91 €	150.073,04 €	158.687,43 €	167.710,01 €	177.156,10 €	177.156,10 €
140.710,04 €	150.363,03 €	160.578,15 €	171.382,43 €	182.803,91 €	194.871,71 €	194.871,71 €
147.745,54 €	159.384,81 €	171.818,62 €	185.093,02 €	199.256,26 €	214.358,88 €	214.358,88 €
155.132,82 €	168.947,90 €	183.845,92 €	199.900,46 €	217.189,33 €	235.794,77 €	235.794,77 €
162.889,46 €	179.084,77 €	196.715,14 €	215.892,50 €	236.736,37 €	259.374,25 €	259.374,25 €
171.033,94 €	189.829,86 €	210.485,20 €	233.163,90 €	258.042,64 €	285.311,67 €	285.311,67 €
179.585,63 €	201.219,65 €	225.219,16 €	251.817,01 €	281.266,48 €	313.842,84 €	313.842,84 €
188.564,91 €	213.292,83 €	240.984,50 €	271.962,37 €	306.580,46 €	345.227,12 €	345.227,12 €



TableModel als Datenkapsel

- Oft soll ein TableModel als eine Zugriffsschicht auf das eigentliche Datenmodell verwendet werden
- TableModel hält daher keine Daten
- Z.B. Zugriff auf Datenbank



siehe dazu Beispiel v2ch6. ResultSetTableModel.java



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 implements TableCellRenderer  
{  
    public Component getTableCellRendererComponent(JTable table,  
        Object value, boolean isSelected, boolean hasFocus,  
        int row, int column) {  
        panel.setBackground((Color) value);  
        return panel;  
    }  
    private JPanel panel = new JPanel();  
}
```



Installation eines TableCellRenderers

Man kann TableCellRenderer auf 2 Arten installieren

- Variante 1: TableCellRenderer für alle Werte einer bestimmten 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.getColumnModel().getColumn(colName);  
column.setCellRenderer(new ColorTableCellRenderer());
```



TableCellEditor

- Damit Zelle editierbar ist, muss im TableModel die Methode `isCellEditable` `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 `TableCellEditor` kann man die Zellen editieren

- Liefert Komponente zum Editieren
- Stellt Interface für Interaktion bereit (`TableCellEditor`)

```
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 `DefaultTableCellEditor`s mit `JComboBox`
- Installation als `CellEditor` für die Spalte

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

TableModel columnModel = table.getColumnModel();

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



Beispiel Planetentabelle

Planet	Radius	Monde	Gasförmig	Farbe	Bild
Jupiter	71.492	16	<input checked="" type="checkbox"/>	Yellow	
Saturn	60.268	18	<input checked="" type="checkbox"/>	Yellow	
Uranus	25.559	11	<input checked="" type="checkbox"/>	Blue	

Siehe Programmcode v2ch6. TableCellRenderTest.java aus CoreJava 2



Eigener TableCellEditor

```
class DropDownEditor extends AbstractCellEditor
    implements TableCellEditor, ActionListener {
    private JComboBox listBox;

    public DropDownEditor() {
        listBox = new JComboBox(...);
        listBox.addActionListener(this);
    }

    public void actionPerformed(ActionEvent evt) {
        fireEditingStopped(); // make renderer reappear
    }

    public Object getCellEditorValue() {
        return new Integer(listBox.getSelectedIndex());
    }

    public Component getTableCellEditorComponent(
       .JTable table, Object value, boolean isSelected,
        int row, int column) {
        listBox.setSelectedIndex(((Integer) value).intValue());
        return listBox;
    }
}
```



Beispiel ColorTableCellEditor

```
class ColorTableCellEditor extends AbstractCellEditor implements TableCellEditor {
    ColorTableCellEditor() {
        panel = new JPanel();

        colorChooser = new JColorChooser();
        colorDialog = JColorChooser.createDialog(null,
            "Platzenfarbe", false, colorChooser,
            new ActionListener() { // Ereignisempfänger für OK-Schaltfläche
                public void actionPerformed(ActionEvent event) {
                    stopCellEditing();
                }
            },
            new ActionListener() { // Ereignisempfänger für Abbrechen-Schaltfläche
                public void actionPerformed(ActionEvent event) {
                    cancelCellEditing();
                }
            });
        colorDialog.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                cancelCellEditing();
            }
        });
    }
}
```

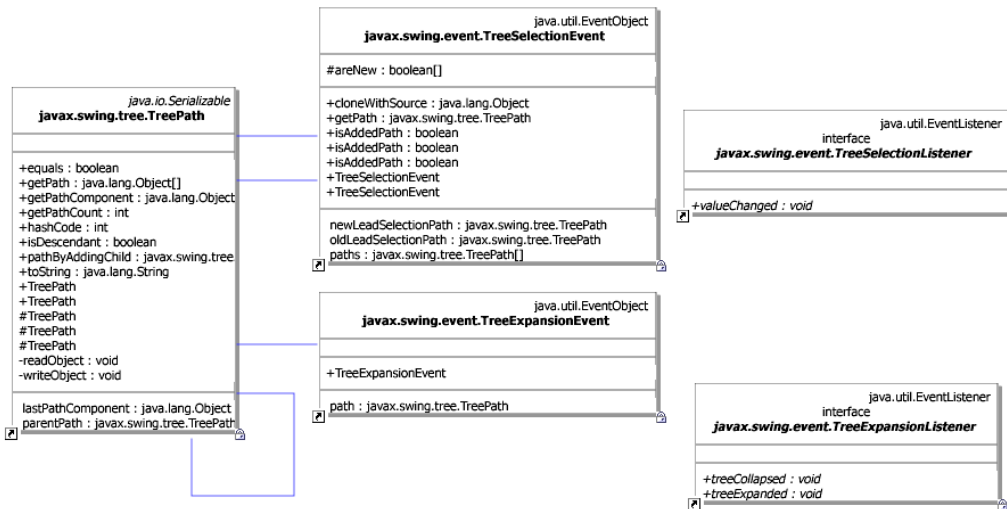


Swing

- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung

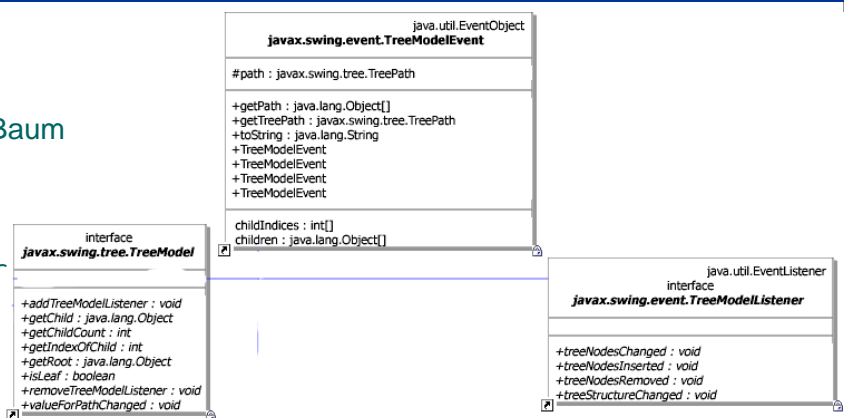


- Treeview mit expandierbarem Baum (wie Windows Explorer)
- Im Aufbau und Funktionsweise analog zu JTable
- JTree verwendet Ereignisse:
 - TreeSelectionEvent
 - TreeExpansionEvent



Interface TreeModel

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



```
public interface TreeModel {

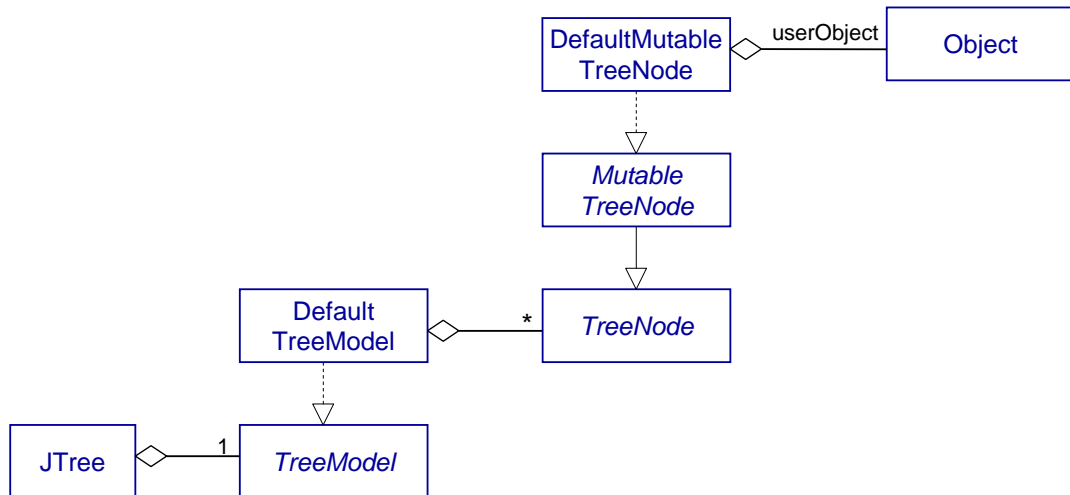
    public Object getRoot();
    public Object getChild(Object parent, int index);
    public int getChildCount(Object parent);
    public boolean isLeaf(Object node);

    public void valueForPathChanged(TreePath path, Object newValue);
    public int getIndexOfChild(Object parent, Object child);

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

Default tTreeModel

- Default tTreeModel bietet eine Implementierung eines TreeModels
- Aufbau eine Baumstruktur mit TreeNodes
- Default tMutableTreeNode ist Standardimplementierung von TeeNode: haben userObject !



Beispiel: Arbeiten mit DefaultMutableTreeNode

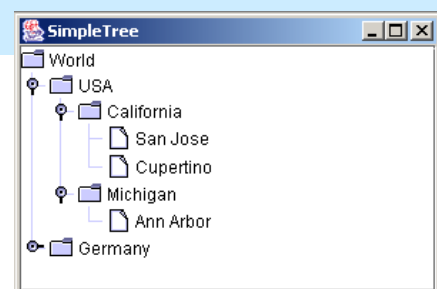
- Knoten anlegen und Baustruktur aufbauen

```
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);
...
```

userObject

- DefaultTreeModel und JTree erzeugen

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



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);
```



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

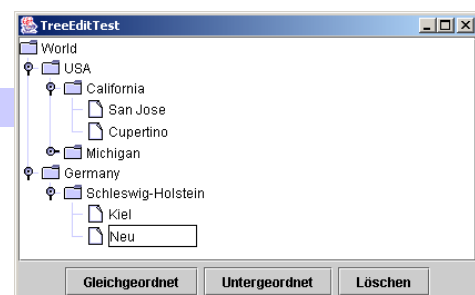
```
model.nodeChanged(node);
```

- Editieren von Knoten

- Setzen von `Editable`-Eigenschaft

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

- Editieren erfolgt durch `DefaultCellEditor`



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.setClientProperty("JTree.LineStyle", "Angled");  
tree.setClientProperty("JTree.LineStyle", "None");  
tree.setClientProperty("JTree.LineStyle", "Horizontal");
```

- Root

```
tree.setRootVisible(true);    Wurzelknoten angezeigt?  
tree.showRootHandles(true);  Root mit Aufklappsymbol
```

- Blätter

- normalerweise, die keine Unterknoten haben
- kann bei DefaultMutableTreeNode auch definiert werden

- zuerst beim JTree erlauben

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

- dann beim Knoten einstellen

```
DefaultMutableTreeNode node = ...;  
node.setAsksAllowsChildren(true);
```



Darstellungsoptionen (2)

- Installation eines eigenen Renderers (wie bei `JList` und `JTable`)
 - Implementieren eines `TreeCellRenderer` 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;
    }
}
```



- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung



Container

Top-Level Containers

Applet Dialog Frame

General-Purpose Containers

A Label on a Panel Scroll pane

Panel Scroll pane

Split pane Tabbed pane

Tool bar



Special-Purpose Containers

Special-Purpose Containers

[Internal frame](#)

[Layered pane](#)

[Root pane](#)



Basic Controls

Basic Controls

[Buttons](#)

[Combo box](#)

[List](#)

[Menu](#)

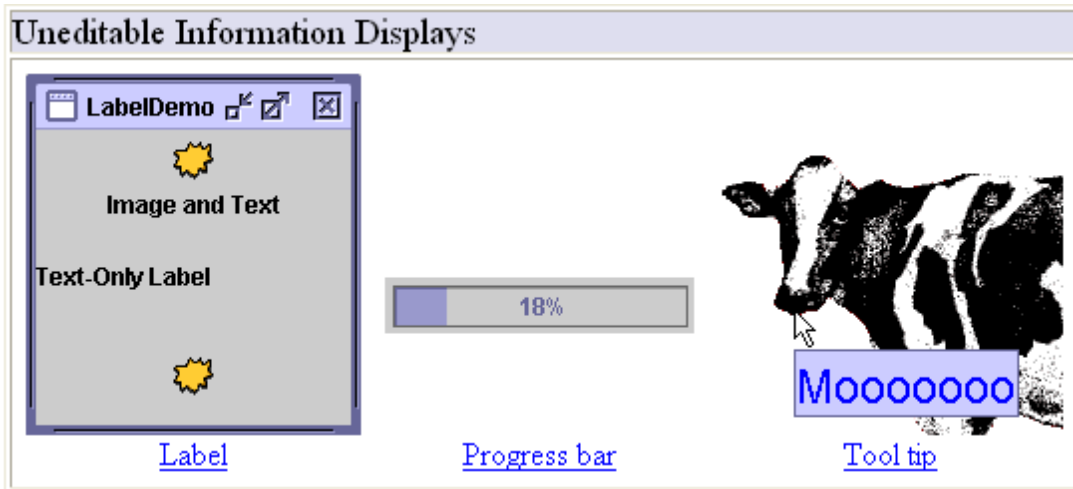
[Slider](#)

[Spinner](#)

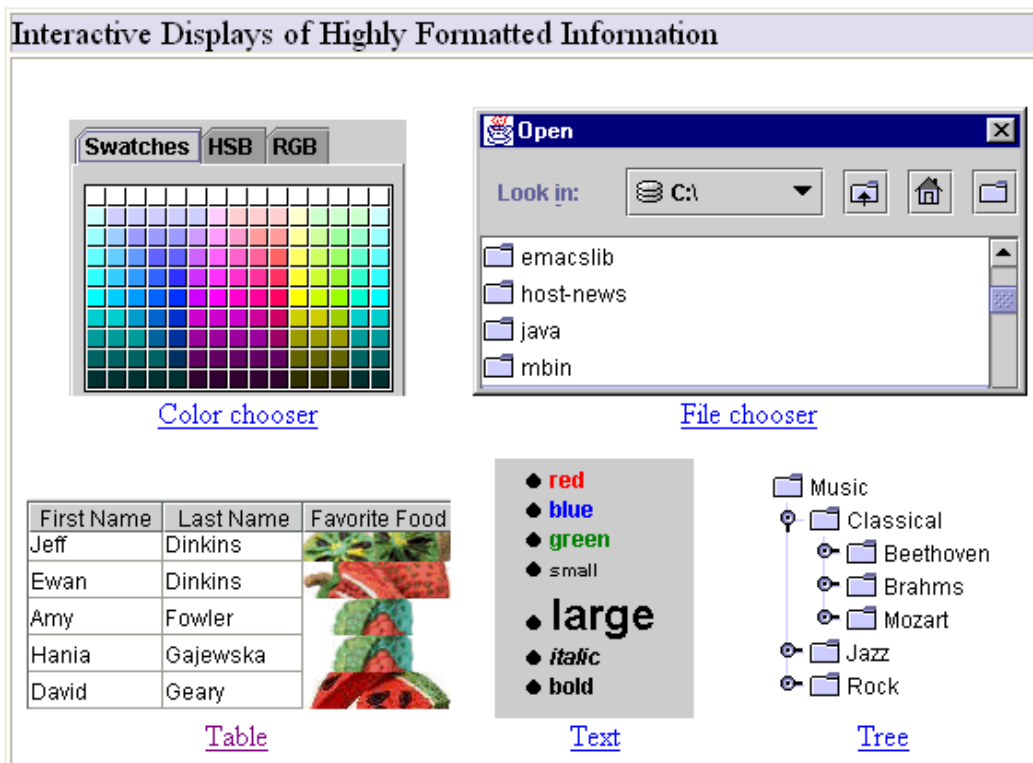
[Text field](#) or [Formatted text field](#)



Uneditable Information Displays



Interactive Displays of Highly Formatted Information



Vorgefertigte Dialogfenster

▪ Meldungsfenster

```
JOptionPane.showMessageDialog(  
    parentFrame, // frame in which dialog is displayed  
    message,     // the message to display  
    title,       // the title string for the dialog  
    messageType, // ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE,  
                // QUESTION_MESSAGE or PLAIN_MESSAGE  
);
```

▪ Eingabefenster

```
String input = JOptionPane.showInputDialog(  
    parentFrame, message, title, messageType);  
// returns the entered text or null if the user has canceled the input
```

```
int option = JOptionPane.showConfirmDialog(  
    parentFrame, message, title, optionType, messageType);  
// returns YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION or CLOSED_OPTION
```



Accessibility

- Unterstützung von Benutzerhilfen
 - z.B. Bildschirmlupe, Vorlesen des Bildschirminhalts
- Paket javax.accessibility
- Komponenten implementieren Accessible
 - Zugriff auf AccessibleContext
- AccessibleContext
 - Repräsentiert Mindestinformation über Komponente
 - z.B. Name, Beschreibung, Rolle, Zustand, unterstützte Aktionen, graphische Repräsentation, Text, Wert, ...



- Einführung
- Painting bei Swing
- Menus and Action
- Swing-Komponenten
 - JSpinner
 - JTextField
 - JFormattedTextField
 - JList
 - JTable
 - JTree
 - weitere Komponenten
- Zusammenfassung



- Anwendungen
 - Hauptfenster und Dialogfenster
 - Pluggable Look and Feel unter Swing
- Applets
 - Kleine Programme innerhalb von Webseiten
 - Eingeschränkte Berechtigungen
- Layout-Manager
 - Kontrollieren Position und Größe der Komponenten
- Ereignisse
 - Klick auf Schaltfläche, Texteingabe, Auswahl eines Listenelements, ...
 - Ereignisbehandler werden bei Komponente registriert
- Menus und Actions
- GUI-Elemente
 - MVC – Muster



Literatur

- The Java™ Tutor, Creating a GUI with JFC/Swing (The Swing Tutorial), <http://java.sun.com/docs/books/tutorial/uiswing/>
- Horstmann, Cornell, Core Java 2, Band 1 - Grundlagen, Markt und Technik, 2002: Kapitel 9, 11
- Horstmann, Cornell, Core Java 2, Band 2 - Expertenwissen, Markt und Technik, 2002: Kapitel 6
- Krüger, Handbuch der Java-Programmierung, 4. Auflage, Addison-Wesley, 2006, <http://www.javabuch.de>, Kapitel 37, 38



Programmbeispiele

- Beispiele zu den Swing-Komponenten
 - Download von Homepage
- Beispiel zu JList mit unterschiedlichen SelectionModes und eigenen Renderer
 - Download von Homepage
- Beispiel InvestmentTable: Erzeugung eines TableModel s:
 - Core Java 2, Band2: v2ch6. InvestmentTable.java
- Beispiel ResultSetTable: TableModel mit Datenbankanschluss:
 - Core Java 2, Band2: v2ch6. ResultSetTable.java
- Beispiel Planetentabelle: JTable mit Editor und Renderer:
 - Core Java 2, Band2: v2ch6. TableCellRenderTest.java

