Java Swing in Theorie und Praxis

Inhaltsverzeichnis

Buch 1: SWING FOR NEWBIES

Buch 2: SWING FOR SMARTIES

ÜBUNGEN

SWING FOR NEWBIES

1) Was ist Swing?

- Swing ist eines von mehreren Application Programming Interface (API) der sogenannten Java Foundation Classes (JFC). Hinter API verbergen sich sämtliche Packages, die zur Programmierung in Java zur Verfügung stehen.
- Swing steht für Java Klassen, die genauso wie die AWT Klassen, zur Entwicklung plattformunabhängiger grafischer Benutzeroberflächen verwendet werden
- Mittels Programmen wie Forte oder JBuilder lassen sich relativ schnell und einfach Swing Oberflächen erstellen
- Mit Java Swing lassen sich unterschiedliche Programme erstellen:
- Swing-Applikationen
- Swing-Applets
- Servlets (ähnlich den Applets, auf Servern eingesetzt)

2) Applets und Applikationen

- Applikationen sind Programme, die nicht an das Internet oder das Intranet einer Firma gebunden
  sind, sondern lokal auf einem Computer laufen
- Für die Ausführung der Applikationen ist die von Java bekannte Main-Methode verantwortlich
- Der Begriff Applet steht für little application
- Applets benötigen immer einen Swing-fähigen Internet-Browser für die Ausführung
- Applets enthalten keine Main-Methode!
- Für den Start eines Applets ist eine HTML-Datei erforderlich

a) Einführendes Beispiel (Hello World)

```
public class HelloWorldApplikation extends JFrame {

    public HelloWorldApplikation () {

        super("HelloWorldApplikation ");
        FlowLayout layout = new FlowLayout(FlowLayout.CENTER);

        Container contentPane = getContentPane();
        contentPane.setLayout(layout);

        JLabel l1 = new JLabel();
        JButton b1 = new JButton("HelloWorld");

        contentPane.add(l1);
        contentPane.add(b1);

        l1.setText("Hello World");
    }

    public static void main(String[]args) {

        JFrame rahmen = new HelloWorldApplikation ();

        WindowListener listener = new WindowAdapter() {

            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(listener);
        rahmen.setSize(300,100);
        rahmen.setVisible(true);
    }
}
```


```
<html>
<body>
<applet code    = "HelloWorldApplet.class"
    width   = 500
    height = 200>
</applet>
<body>
<html>
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HelloWorldApplet extends JApplet
{
    public void init() {
        FlowLayout layout = new FlowLayout(FlowLayout.CENTER);
        Container contentPane = getContentPane();
            contentPane.setLayout(layout);
            JLabel l1 = new JLabel();
        JButton b1 = new JButton("HelloWorld");
        contentPane.add(l1);
        contentPane.add(b1);
        l1.setText("Hello World");
    }

    public void start() {}

    public void stop() {}

    public void destroy () {}

}
```

3) Die Erweiterung von AWT

- Swing ist eine Erweiterung des Abstract Windowing Toolkits (AWT)
- Viele Komponenten sind sowohl als AWT-, wie auch als Swing-Komponenten verfügbar
- Die Swing-Komponenten sind allerdings wesentlich mächtiger
- Unterscheiden kann man die verschiedenen Komponenten an ihrem Namen, z.B.:
- Ein Button ist in AWT unter java.awt.Button zu finden, in Swing unter javax.swing.JButton
- Das x in Paketnamen javax besagt, das es sich um ein extended-Pakete handelt
- Bei Swing Komponenten wird ein J vorangestellt

4) Komponenten und Container

Es gibt zwei Arten von Komponenten:
- heavyweigth components (Schwergewichtige Komponenten)
- lightweigth components (Leichtgewichtige Komponenten)
- heavyweigth components werden in einem undurchsichtigen Fenster dargestellt und dürfen keine anderen heavyweigth components enthalten
- lightweigth components dagegen können einen durchsichtigen Hintergrund haben und beliebige Formen annehmen
- Zur Gruppe der heavyweigth components zählen Rahmen (JFrame), Applets und Dialoge
- Nahezu alle Swing- Komponenten sind aber lightweigth components und müssen in heavyweigth components platziert werden, da sie kein eigenes Fenster haben

a) Top-Level-Container

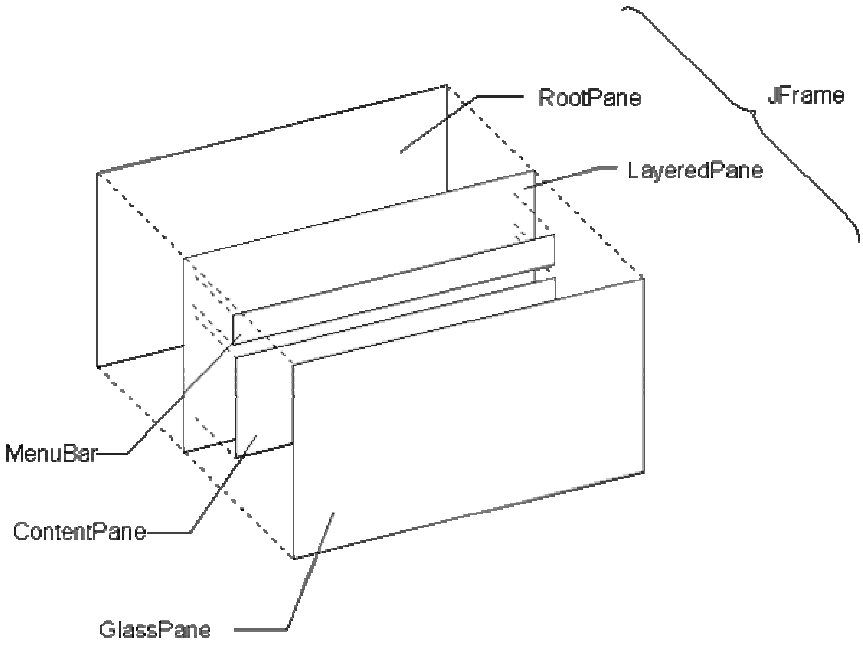
- Zu den Komponenten des Top-Level- Container zählen die heavyweigth components JFrame,
  JDialog und JApplet
- Diese Komponenten bilden praktisch den äußeren Rahmen eines Programms und alle anderen
- Komponenten, die in dieser Anwendung verwendet werden, befinden sich innerhalb eines Top-Level-Containers
- Für die Platzierung anderer Komponenten in einen Top-Level-Container ist noch eine Zwischenkomponente erforderlich, das so genannte content pane
- Neue Komponenten werden also nicht dem Hauptcontainer zugefügt, sondern dem content pane, das selber ein Teil des Hauptcontainers ist

b) General-Purpose-Container

- General-Purpose-Container sind Behälter für allgemeine Anwendungen. Zu dieser Gruppe von Komponenten zählen u.a. Panels. Ein Panel dient als Behälter für weitere Komponenten, wie z.B. Schieberegler, Schaltflächen, Eingabefelder und Listboxen.

c) Special-Purpose-Container

- Special-Purpose-Container sind Behälter für spezielle Einsatzgebiete. Zu dieser Gruppe zählen die
  so genannten Internal frames, Layered panes sowie die Root panes.
- Internal frames (Interne Rahmen) verhalten sich ähnlich den Frames und werden gebraucht, wenn
  mehrere Frames gleichzeitig geöffnet sein sollen.
- Layered panes werden zu Darstellung von überlappenden Komponenten verwendet
- Jeder Container, der über einen Root pane verfügt, enthält auch automatisch ein Layered pane
- Root pane werden automatisch von den Top-Level-Containern erzeugt



Aus: Java als erste Programmiersprache, Goll/Weiß/Müller 3. Auflage

d) Weitere Komponenten

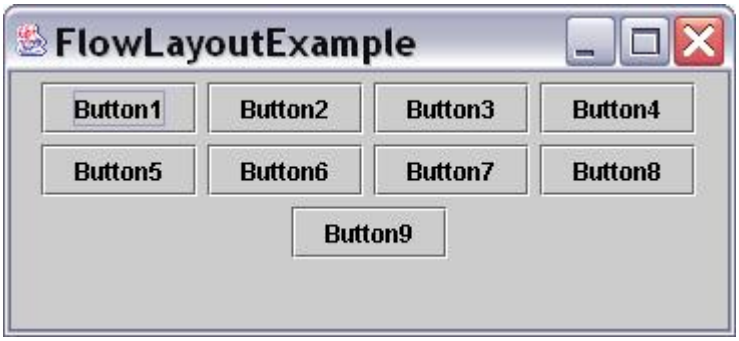
- Basic Controls sind Schaltflächen, Radiobuttons oder Checkboxen, sowie Listenfelder, Comboboxen, Menüs, Schieberegler und Textfelder.
- Nicht-editierbare Komponenten zur Anzeige von Informationen sind Fortschrittsbalken, Tooltips und Labels.

e) Layout-Manager

Es gibt sechs verschiedene Layout Manager:
- FlowLayout
- GridLayout
- BorderLayout
- CardLayout

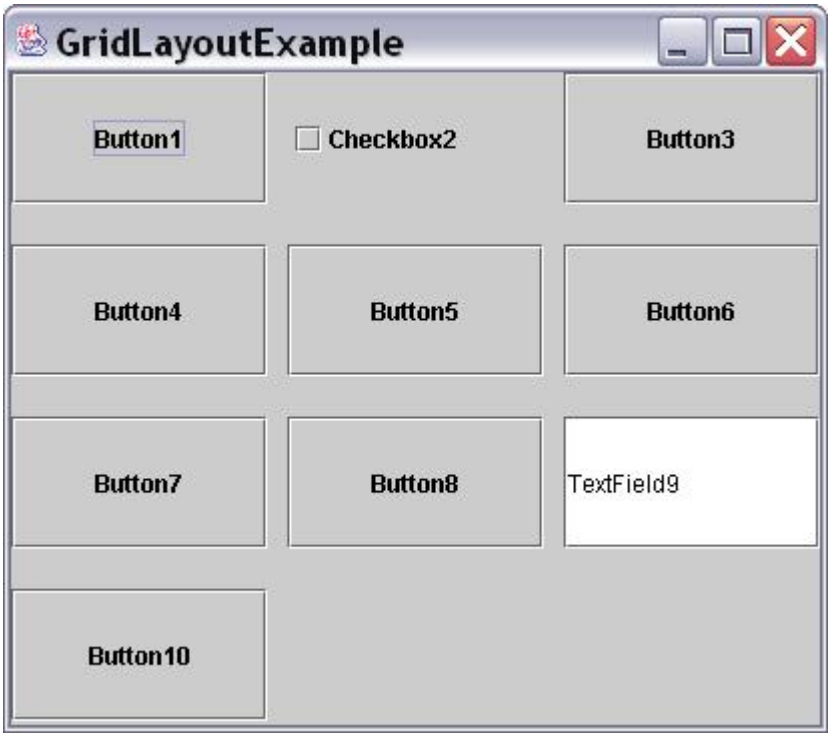
- GridBagLayout
- BoxLayout
- Jedem Bereich kann eine Komponente zugeordnet werden.
- Wenn mehrere Komponenten in einen Bereich angeordnet werden sollen, muss man einen Trick anwenden:
- Es muss eine Instanz der Klasse Box erzeugt werden, welche dann mehrere Komponenten aufnehmen kann und diese Box dann einem Bereich im BorderLayout zuordnen.

f) Layout-Beispiele



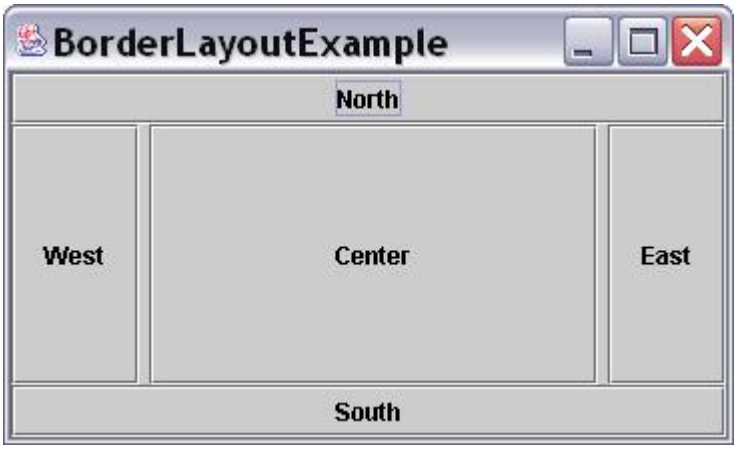

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FlowLayoutExample extends JFrame {

    public FlowLayoutExample() {

        super("FlowLayoutExample");
        FlowLayout layout = new FlowLayout(FlowLayout.CENTER);
        Container contentPane = getContentPane();
        contentPane.setLayout(layout);

        JButton button1 = new JButton("Button1");
        JButton button2 = new JButton("Button2");
        JButton button3 = new JButton("Button3");
        JButton button4 = new JButton("Button4");
        JButton button5 = new JButton("Button5");
        JButton button6 = new JButton("Button6");
        JButton button7 = new JButton("Button7");
        JButton button8 = new JButton("Button8");
        JButton button9 = new JButton("Button9");
```

```
        contentPane.add(button1);
        contentPane.add(button2);
        contentPane.add(button3);
        contentPane.add(button4);
        contentPane.add(button5);
        contentPane.add(button6);
        contentPane.add(button7);
        contentPane.add(button8);
        contentPane.add(button9);
    }

    public static void main(String[]args)    {

        JFrame rahmen = new FlowLayoutExample();

        WindowListener listener = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };
        rahmen.addWindowListener(listener);
        rahmen.pack();
        rahmen.setVisible(true);
    }
}
```



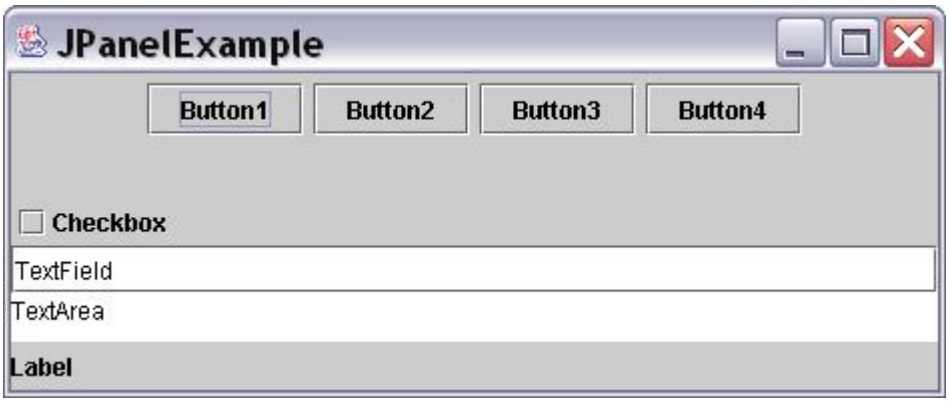
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class GridLayoutExample extends JFrame{

    public GridLayoutExample(){

        super("GridLayoutExample");

        Container contentPane = getContentPane();
        contentPane.setLayout(new GridLayout(4,3,10,20));

         JButton button1       = new JButton("Button1");
         JCheckBox checkbox2 = new JCheckBox("Checkbox2");
         JButton button3       = new JButton("Button3");
         JButton button4       = new JButton("Button4");
         JButton button5       = new JButton("Button5");
         JButton button6       = new JButton("Button6");
         JButton button7       = new JButton("Button7");
         JButton button8       = new JButton("Button8");
         JTextField textfield9   = new JTextField("TextField9");
         JButton button10      = new JButton("Button10");

        contentPane.add(button1);
        contentPane.add(checkbox2);
        contentPane.add(button3);
        contentPane.add(button4);
        contentPane.add(button5);
        contentPane.add(button6);
        contentPane.add(button7);
        contentPane.add(button8);
        contentPane.add(textfield9);
        contentPane.add(button10);
    }
```

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

        JFrame rahmen = new GridLayoutExample();
        WindowListener listener = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(listener);
        rahmen.setSize(300,100);
        rahmen.setVisible(true);
    }
}
```




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BorderLayoutExample extends JFrame{

    public BorderLayoutExample(){

        super("BorderLayoutExample");

        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout(5,0));

         JButton buttonNorth= new JButton("North");
         JButton buttonSouth   = new JButton("South");
         JButton buttonCenter  = new JButton("Center");
         JButton buttonWest = new JButton("West");
         JButton buttonEast  = new JButton("East");
```

```
        contentPane.add("North" , buttonNorth);
        contentPane.add("South" , buttonSouth);
        contentPane.add("Center" , buttonCenter);
        contentPane.add("West" , buttonWest);
        contentPane.add("East", buttonEast);
    }

    public static void main(String[]args) {

        JFrame rahmen = new BorderLayoutExample();
        WindowListener listener = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(listener);
        rahmen.pack();
        rahmen.setVisible(true);
    }
}
```



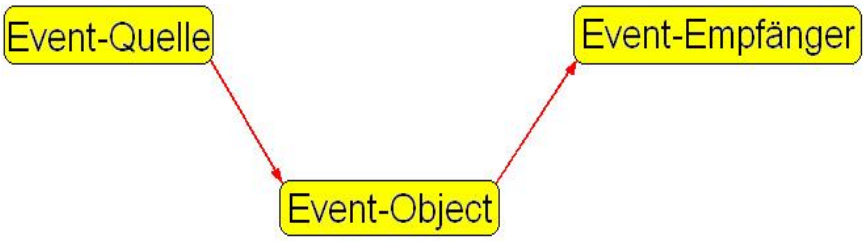

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class JPanelExample extends JFrame{

    public JPanelExample(){

        super("JPanelExample");
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
```

```
        JPanel northPanel   = new JPanel();
        JPanel southPanel   = new JPanel();

        contentPane.add("North" , northPanel);
        contentPane.add("South" , southPanel);

        JButton button1 = new JButton("Button1");
        JButton button2 = new JButton("Button2");
        JButton button3 = new JButton("Button3");
        JButton button4 = new JButton("Button4");

        northPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

        northPanel.add(button1);
        northPanel.add(button2);
        northPanel.add(button3);
        northPanel.add(button4);

         JCheckBox checkbox    = new JCheckBox("Checkbox");
         JTextField textField  = new JTextField("TextField");
         JTextArea textArea    = new JTextArea("TextArea");
         JLabel label          = new JLabel("Label");

        southPanel.setLayout(new GridLayout(4,0));

        southPanel.add(checkbox);
        southPanel.add(textField);
        southPanel.add(textArea);
        southPanel.add(label);
    }

    public static void main(String[]args){
        JFrame rahmen = new JPanelExample();
        WindowListener listener = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(listener);
        rahmen.pack();
        rahmen.setVisible(true);
    }
}
```

SWING FOR SMARTIES

1) Events

Ein Event wird von AWT oder SWING -Komponenten als Reaktion auf eine Eingabe eines Benutzers generiert. AWT und SWING verwenden dieselben Ereignisse.
Die Ereignisbehandlung beruht auf dem Delegations-Modell. Durch das Verwenden des Delegation-Modell werden die GUI –und Anwendungsklassen von einander getrennt.
Das Event-Objekt verarbeitet die Ereignisse nicht selbst sondern gibt das Ereignis an einen so genannten EventListener weiter.
EventListener tragen ein spezielles Interface ein und melden sich bei
Bedarf bei der Event-Quelle an.
Es steht für jedes EventListener-Interface mit mehr als einer Methode eine Adapterklasse zur Verfügung.

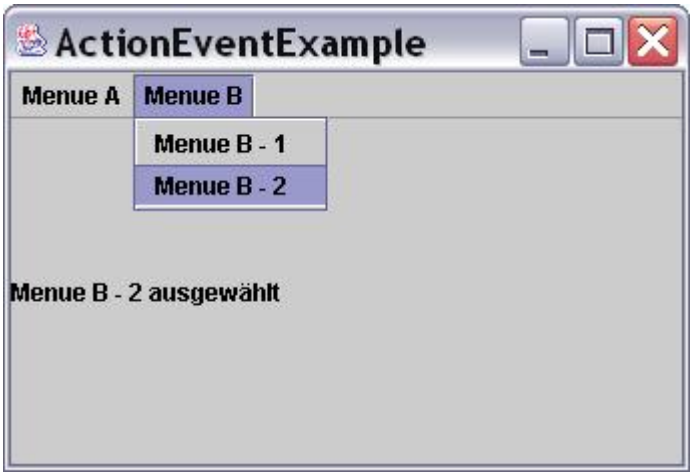


a) Action

Der ActionEvent ist der bekannteste und am häufigsten verwendetste Eventtyp. Er wird ausgelöst durch das betätigen von Buttons, das Drücken der Enter -oder Returntaste in einem Textfeld und die Auswahl eines Eintrages in einem Menü.

Die Methode zum Anmelden des ActionEventListener: **addActionListener(...)**
Die Methode zur Verarbeitung des **ActionEvent**: **actionPerformed(ActionEvent ae)**

Wichtige Methoden: **getActionCommand()** um zu erkennen welche Componente den ActionEvent
erzeugt hat




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ActionEventExample extends JFrame implements ActionListener {

    private JMenuBar jMenuBar1;
    private JMenu jMenu1;
    private JMenu jMenu2;
    private JMenuItem jMenuItem1;
    private JMenuItem jMenuItem2;
    private JMenuItem jMenuItem3;
    private JMenuItem jMenuItem4;
    private JLabel jLabel1;

    public ActionEventExample() {

        super("ActionEventExample");

        BorderLayout layout = new BorderLayout();
        Container contentPane = getContentPane();
        contentPane.setLayout(layout);

        jLabel1 = new JLabel("Das Menue wurde noch nicht bedient");

        jMenuBar1 = new JMenuBar();
        jMenu1 = new JMenu();
        jMenu2 = new JMenu();
        jMenuItem1 = new JMenuItem();
        jMenuItem2 = new JMenuItem();
        jMenuItem3 = new JMenuItem();
        jMenuItem4 = new JMenuItem();
```

```
        setJMenuBar(jMenuBar1);
        jMenu1.setText("Menue A");
        jMenu2.setText("Menue B");
        jMenuItem1.setText("Menue A - 1");
        jMenuItem2.setText("Menue A - 2");
        jMenuItem3.setText("Menue B - 1");
        jMenuItem4.setText("Menue B - 2");

        jMenu1.add(jMenuItem1);
        jMenu1.add(jMenuItem2);
        jMenu2.add(jMenuItem3);
        jMenu2.add(jMenuItem4);
        jMenuBar1.add(jMenu1);
        jMenuBar1.add(jMenu2);

        jMenuItem1.addActionListener(this);
        jMenuItem2.addActionListener(this);
        jMenuItem3.addActionListener(this);
        jMenuItem4.addActionListener(this);

        contentPane.add("Center", jLabel1);

        pack();
    }

    public static void main(String[]args) {

        JFrame rahmen = new ActionEventExample();

        WindowListener listener = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                    System.exit(0);
            }
        };

        rahmen.addWindowListener(listener);
        rahmen.show();
    }
```

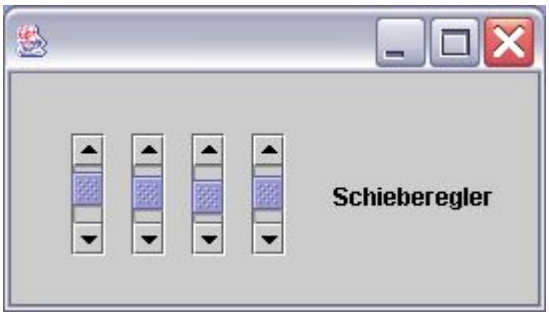
```
    public void actionPerformed(java.awt.event.ActionEvent actionEvent) {
        Object quelle = actionEvent.getSource();
        if(quelle == jMenuItem1)
            jLabel1.setText("Menue A - 1 ausgewählt");
        if(quelle == jMenuItem2)
            jLabel1.setText("Menue A - 2 ausgewählt");
        if(quelle == jMenuItem3)
            jLabel1.setText("Menue B - 1 ausgewählt");
        if(quelle == jMenuItem4)
            jLabel1.setText("Menue B - 2 ausgewählt");
    }
}
```

b) Adjustment

Der AdjustementEvent tritt auf wenn Rollbalken oder Scrollbars bedient werden. Hier repräsentiert der Scrollbar oder Schieberegler einen bestimmten Wert der über den AdjustmentEvent erfragt werden kann.

Die Methode zur Anmeldung des AdjustmentEventListener: **addAdjustmentListener(...)**
Die Methode zur Verarbeitung des **AdjustmentEvent**:
adjustmentValueChanged(AdjustmentEvent adj)

Wichtige Methoden: **getValue()** zum Auslesen des Wertes des Schiebereglers als **int**



c) Item

Der ItemEvent tritt bei der Verwendung von Checkboxen, in Menüeinträgen, Buttons, Comboboxen und Radiobuttons auf.

Die Methode zur Anmeldung des ItemEventListener: **addItemListener(...)**
Die Methode zur Verarbeitung des **ItemEvent**:
itemStateChanged(ItemEvent ie)

Wichtige Methoden: **getSource()** um zu erkennen welches Objekt den Event ausgelöst hat

d) Key

Um Daten einzugeben ist die Tastatur noch immer die wichtigste Möglichkeit. Durch das drücken, halten, loslassen, von Tasten werden KeyEvents ausgelöst.

Die Methode zur Anmeldung des ItemEventListener: **addKeyListener(...)**
Die Methode zur Verarbeitung des **KeyEvent:**

- Drücken einer Taste bewirkt **keyPressed(KeyEvent ke)**
- Loslassen einer Taste bewirkt: **keyReleased(KeyEvent ke)**
- Kurzes Antippen einer Taste bewirkt: **keyTyped(KeyEvent ke)**

Wichtige Methoden:

- **getKeyChar()** liefert den Wert eines gedrückten Zeichens als char-Objekt zurück
- **getKeyCode()** liefert den Wert eines gedrückten Zeichens als int-Wert zurück der den Zeichencode des Zeichens entspricht
- **getKeyString()** liefert über **getKeyChar()** hinaus bei Tasten wie F9, Alt, Strg oder Tab einen ganzen String als Beschreibung z.B. bei Pos1 liefert die Funktion dann auch Pos1 und nicht nur ein char-Objekt

e) Window

Der WindowsEvent wird bei jedem Fenster zum Schließen verwendet. Ohne die Verwendung des WindowsEvent müsste die Konsole über die Eingabeaufforderung geschlossen (Strg-C) geschlossen werden. Durch den WindowEvent werden die Funktionen Minimieren, Maximieren und Schließen zur Verfügung gestellt.

Die Methode zur Anmeldung des WindowEventListener: **addWindowListener(...)**
Die Methode zur Verarbeitung des **WindowEvent:**

- Schließen des Fenster: **windowClosing(WindowsEvent we)**
- Aktivieren des Fenster: **windowActivated(WindowsEvent we)**
- Deaktivieren des Fenster: **windowDeactivated(WindowsEvent we)**
- vom Minimiert- zum Normalzustand: **windowDeiconified(WindowsEvent we)**
- vom Normal- zum Minimiertzustand: **windowIconified(WindowsEvent we)**
- Öffnen des Fenster: **windowOpened(WindowsEvent we)**

f) Mouse

Die Mouse ist für graphische Anwendungen das wichtigste Eingabegerät, deshalb werden von Java dafür Events zur Verfügung gestellt.
Der MouseListener reagiert nur auf die Verwendung der verschiedenen Mouse-Tasten. Für die Bewegungen der Mouse bzw. für das Anzeigen der Mouse-Position wird ein MouseMotionEvent verwendet.

Die Methode zur Anmeldung des MouseEventListener: **addMouseListener(...)**
Die Methode zur Verarbeitung des **MouseEvent:**

- Klicken auf eine Komponente: **mouseClicked(MouseEvent me)**
- Maus betritt eine Komponente: **mouseEntered(MouseEvent me)**
- Maus verlässt eine Komponente: **mouseExited(MouseEvent me)**
- Maustaste auf einer Komponente gedrückt: **mousePressed(MouseEvent me)**
- Maustaste wieder loslassen: **mouseReleased(MouseEvent me)**

g) MouseMotion

Der MouseMotionListener ist die Ergänzung zum MouseListener der sich nur um die Ereignisse bei den Mouse-Tasten kümmert. Der MouseMotionListener behandelt normale Mausbewegungen und Mausbewegungen mit gedrückter Maustaste. Die erste Funktion ist sinnvoll wenn man die Position der Maus auf dem Bildschirm oder in einem Programm erhalten möchte. Die zweite Funktion findet Verwendung beim Verschieben vom Fenster oder Komponenten.

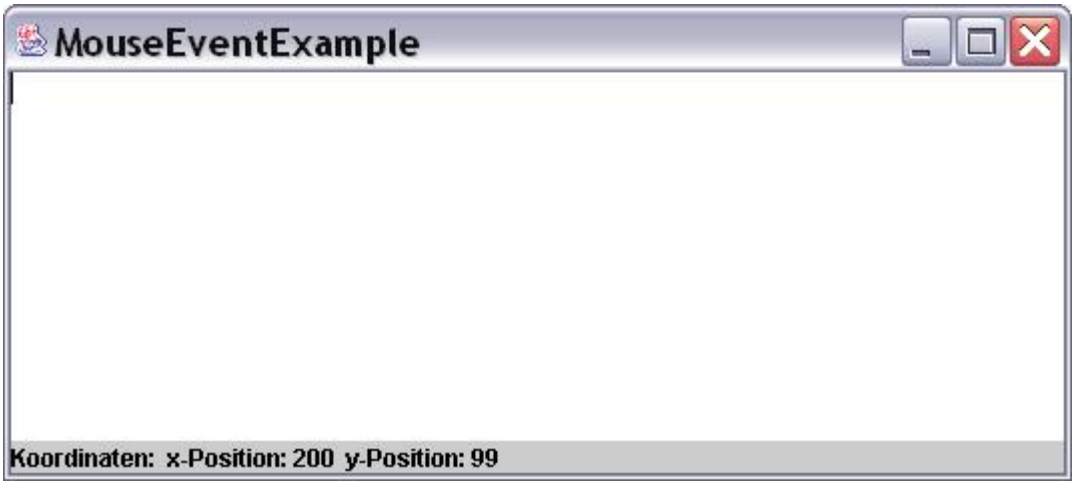
Die Methode zur Anmeldung des MouseMotionEventListener:
addMouseMotionListener(...)
Die Methode zur Verarbeitung des **MouseMotionEvent:**

- normale Bewegung der Maus: **mouseMoved(MouseEvent me)**
- Mausbewegung mit gedrückter Maustaste: **mouseDragged(MouseEvent me)**

Wichtige Methoden:

- **getX()** liefert den horizontalen Wert der Maus-Position
- **getY()** liefert den vertikalen Wert der Maus-Position

Die beiden Funktion können nur innerhalb der Funktion **mouseMoved()** verwendet werden.



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MouseEventExample extends JFrame {

   private JTextArea textArea   = new JTextArea(5, 30);
   private Jlabel               = new JLabel();

   public MouseEventExample() {

        super("MouseEventExample");

        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());

        contentPane.add(textArea, BorderLayout.CENTER);
        contentPane.add(label, BorderLayout.SOUTH);

        textArea.addMouseMotionListener(new MouseHandler());

        pack();
   }

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

        WindowListener listener = new WindowAdapter() {
             public void windowClosing(WindowEvent e) {
                  System.exit(0);
             }
        };

        rahmen.addWindowListener(listener);
        rahmen.show();
   }

   class MouseHandler extends MouseMotionAdapter {
        public void mouseMoved(MouseEvent e) {
             label.setText("Koordinaten:    x-Position: " + e.getX() + "
                                       y-Position: " + e.getY());
        }
   }
}
```

h) Component

Der ComponentEventListener wird zum verschieben, verstecken, und wieder sichtbar machen von Komponenten verwendet.

Die Methode zur Anmeldung des ComponentEventListener: **addComponentListener(...)**
Die Methode zur Verarbeitung des **ComponentEvent:**

- Aufruf bei Veränderung der Komponentengröße: **componentResized(ComponentEvent ce)**
- Aufruf bei Verschieben der Komponente: **componentMoved (ComponentEvent ce)**
- Aufruf wenn die Komponente wieder sichtbar wird: **componentShown(ComponentEvent ce)**
- Aufruf wenn die Komponente unsichtbar wird: **componentHidden(ComponentEvent ce)**

i) Focus

Um in einer Anwendung die Tastatur verwenden zu können muss die Anwendung den Focus haben, sprich die gerade verwendete Anwendung sein. Die Objekte die den Focus besitzen sind stärker umrandet und haben die stärkere Farbe im Fensterkopf.

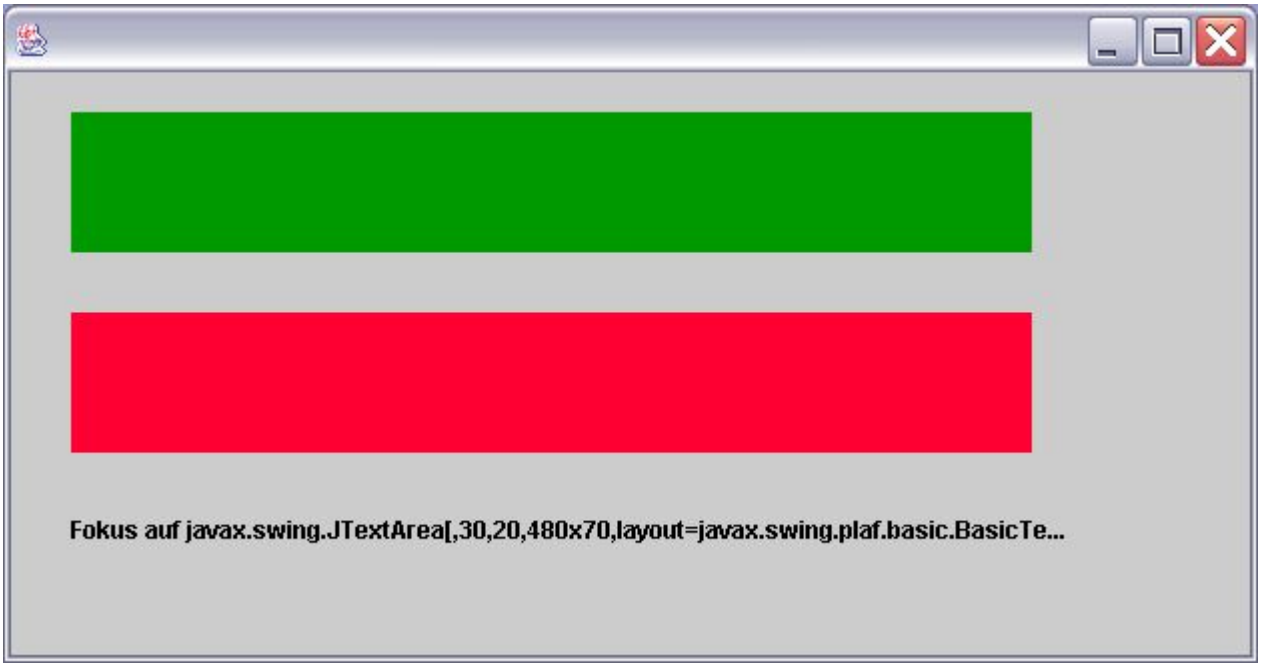
In einer einzelnen Anwendung hat auch immer nur eine Komponente den Focus und ist durch eine Eingabemarke gekennzeichnet, den so genannten Cursor

Die Methode zur Anmeldung des FocusEventListener: **addFocusListener(...)**
Die Methode zur Verarbeitung des **FocusEvent:**

- bei Verlust des Focus: **focusLost(FocusEvent fe)**
- bei Erhalt des Focus: **focusGained(FocusEvent fe)**

Wichtige Methoden:

- **getSource()** liefert die Komponente die den Event ausgelöst hat



j) ListSelection

Der ListSelectionEvent wird eingesetzt wenn über List -oder Komboboxen Elemente ausgewählt werden sollen.

Bei der Behandlung der ListSelectionEvents ist eine Besonderheit zu erwähnen. Anstatt des normalerweise zu importierenden Paketes java.awt.event.* ist das Paket javax.swing.event.* zu importieren. Der Grund dafür ist das es zwei unterschiedliche addListSelectionListener(...) gibt.

Die Methode zur Anmeldung des ListSelectionListener: **addListSelectionListener(...)**
Die Methode zur Verarbeitung des **ListSelectionEvent:**

- Meldet sich wenn ein anderer Wert in der Liste ausgewählt wird:
  valueChanged(ListSelectionEvent lse)

Wichtige Methoden:

- **addSelectionInterval(a, b)** bestimmt das Intervall mit den Werten die schon am Anfang
  ausgewählt sind
- **getValueIsAdjusting()** liefert zurück ob ein Listenelement ausgewählt ist oder nicht
- **getFirstIndex()** gibt an, welches Element das erste aller selektierten in der Liste ist
- **getLastIndex()** gibt an, welches Element das letzte aller selektierten in der Liste ist

k) Text

Der TextEventListener verarbeitet Events im Bereich TextArea und TextField. Die Verarbeitung von Veränderungen im TextArea und TextField werden über den TextEventListener verarbeitet.

Die Methode zur Anmeldung des TextEventListener: **addTextListener(...)**
Die Methode zur Verarbeitung des **TextEvent:**

- Meldet wenn sich Änderungen im TextArea oder im TextField ergeben
  textValueChanged(TextEvent te)

2) Implementierung

Für die Implementierung von Listenern gibt es mehrere Möglichkeiten. Bei allen Implementierungsmöglichkeiten stehen immer die Methoden addXYZListener() und removeXYZListener() zum Anmeldung bzw. zum Abmelden in der Klasse zur Verfügung. Der Event wird über Klasse erzeugt in der addXYZListener() und removeXYZListener() deklariert sind.

a) über Interface

Die erste Variante zur Implementierung eines Events ist über Listener. Über
implements XYZListener wird das zum Event passende Interface geladen und das Interface stellt dann seine Methoden zur Verfügung bzw. wünscht das alle seine Methoden überschrieben werden, ansonsten meldet der Compiler die Klasse soll als abstrakte Klasse dargestellt werden, weil nicht alle Methoden des Interface in der Klasse deklariert sind.

Public class XYZEventDemo extends Applet implements XYZListener

b) oder mit Adapterklasse

Die zweite Variante ist die Implementierung über Adapterklasse. Das Prinzip ist ähnlich, es werden wieder Methoden in der Benutzerklasse überschrieben, aber in diesem Fall nur die vom Benutzer erwünschten und nicht alle wie bei der Lösung über Interface. Die Adapterklasse implementiert selbst das Interface und schreibt leere Methoden in ihren Klassenrumpf was zur Folge hat das die Benutzer nur die Methoden überladen müssen diese wirklich brauchen und nicht alle.
Da man in Java aber nur von einer Klasse erben kann, aber beliebig viele Interfaces implementieren kann ist oft die Interface-Lösung einfacher realisierbar sein.

Public class XYZHandler extends XYZAdapter

Adapterklassen gibt es nur zu EventListener mit mehr als einer ausführbaren Methode.

c) Liste über EventListenerInterfaces und Adapterklassen

Event-Listener-Interface	**Adapterklasse**	**Methoden**
Action	---	actionPerformed(ActionEvent)
Adjustment	---	AdjustmentValueChanged (AdjustmentEvent)
Item	---	ItemStateChanged(ItemEvent)
Key	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
Window	WindowAdapter	windowClosing(WindowsEvent we) windowActivated(WindowsEvent we) windowDeactivated(WindowsEvent we) windowDeiconified(WindowsEvent we) windowIconified(WindowsEvent we) windowOpened(WindowsEvent we)
Mouse	MouseAdapter	mouseClicked(MouseEvent me) mouseEntered(MouseEvent me) mouseExited(MouseEvent me) mousePressed(MouseEvent me) mouseReleased(MouseEvent me)
MouseMotion	MouseMotionAdapter	mouseDragged(MouseEvent me) mouseMoved(MouseEvent me)
Component	ComponentAdapter	componentResized(ComponentEvent ce) componentMoved (ComponentEvent ce) componentShown(ComponentEvent ce) componentHidden(ComponentEvent ce)
Focus	FocusAdapter	focusGained(FocusEvent fe) focusLost(FocusEvent fe)
ListSelection	---	valueChanged(ListSelectionEvent lse)
Text	---	TextValueChanged(TextEvent te)

3) Aufbau eines EventHandlers

Für die Implementierung des EventHandlers gibt es mehrere Möglichkeiten. Zum einen die Implementierung direkt in der Klasse selbst über

Public class XYZHandler extends XYZAdapter
oder
Public class XYZEventDemo extends Applet implements XYZListener

oder über eine innere Klasse.

a) in der gleichen Klasse

Wenn der Event über die Benutzerklasse direkt behandelt werden soll, sind einige Fragestellungen zu klären.

- der XYZListener muss von der Benutzerklasse selbst implementiert werden oder über eine Adapterklasse zur Verfügung gestellt werden.
  Problematisch wird es, wenn mehrere Adapter zur Verfügung gestellt werden sollen, denn das ist so nicht möglich.
  Die Implementierung mehrerer Interfaces ist dagegen problemlos möglich.

- die Anmeldung des EventListeners erfolgt über die Methode:
  addXYZListener(**this**);
  this deshalb weil die Klasse selbst den XYZListener implementiert

- die Methoden zur Bearbeitung der Events (z.b **actionPerformed(ActionEvent e)**) können an beliebiger Stelle in der Klasse geschrieben werden

b) als innere Klasse

Die zweite und wesentlich komfortabler Lösung ist über eine innere Klasse.

- der Name der innere Klasse ist frei wählbar, aber oft werden wie XYZBearbeiter oder XYZHandler verwendet.

- Der XYZListener wird von der inneren Klasse implementiert oder über eine Adapterklasse zur Verfügung gestellt werden.
  Das Problem mit mehreren Adaptern kann dadurch gelöst werden in dem man jeder Eventgruppe eine eigene innere Klasse zur Verfügung stellt.

  Die Implementierung mehrerer Interfaces ist dagegen problemlos möglich, wobei man in einer inneren Klasse aus Übersichtlichkeitsgründen nur eine Eventgruppe bearbeiten sollte. Für jeden weiteren Event bzw. Eventgruppe sollte jeweils eine neue

innere Klasse definiert werden.

- die Anmeldung des EventListeners erfolgt über die Methode:
  addXYZListener(**new EventHandler()**);
-
- Die innere Klasse sieht so aus (innere Klasse **fett**):

```
 public class UserClass {

      private int …;
      …
      …

      public UserClass {
          …
          komponente.addXYZListener(new XYZHandler());
          komp.addXYZListener(new XYZBearbeiter());
          …
      }

      class XYZHandler implements XYZListener {
          public void handlingMethod(XYZ Event) {
              …
              …
              …
          }
      }

      class XYZBearbeiter extends XYZAdapter {
          public void handlingMethod(XYZ Event) {
              …
              …
              …
          }
      }
}
```

c) anonyme Klasse

Die dritte Möglichkeit ist über eine anonyme Klasse. Dort wird die Handler-Klasse direkt in den addXYZListener() eingebaut bzw. er wird in der Methode erzeugt.

- Die Möglichkeiten sind ähnlich wie bei inneren Klassen
- Es kann aber in einer anonymen Klasse immer nur eine Methode bearbeitet werden
- Die anonyme Klasse sieht so aus (anonyme Klasse **fett**):

```
public class UserClass {

    private int …;
    …
    …

    public UserClass {
        …
        komponente.addXYZListener(new XYZListener(){
            public void xyzmethod(XYZEvent e){
                …
                …
            }
        );
        …
        komp.addXYZListener(new XYZAdapter(){
            public void xyzmethod(XYZEvent e){
                …
                …
            }
        );
        …
    }
}
```

d) Übersicht über Komponenten und die zugehörigen Events

Komponente	Action	Text	Focus	Mouse	Key	Item	ListSelection	Window
JButton	*		*	*	*	*		
JCheckBox	*		*	*	*	*		
JColorChooser			*	*	*			
JComboBox	*		*	*	*	*		
JDialog			*	*	*			*
JEditorPane			*	*	*			
JFileChooser	*		*	*	*			
JFrame			*	*	*			*
JInternalFrame			*	*	*			
JList			*	*	*		*	
JMenu			*	*	*			
JMenuItem	*					*		
JOptionPane			*	*	*			
JPasswordField			*	*	*			
JPopupMenu	*		*	*	*			
JProgressBar			*	*	*			
JRadioButton	*		*	*	*	*		
JSlider			*	*	*			
JTabbedPane			*	*	*			
JTable			*	*	*		*	
JTextArea	*	*	*	*	*			
JTextField		*	*	*	*			
JTextPane			*	*	*			
JToggleButton	*		*	*	*	*		
JTree			*	*	*			
JViewport			*	*	*			

Quelle: Java-Swing Ralf Jesse bhv

4) Weitere Komponenten

Die in diesem Kapitel dargestellten Komponenten sollen einen weiteren Einblick auf Java-Swing liefern. Die Vorstellung stellt Java-Swing nur auszugsweise vor und ist nicht vollständig. Weitere Informationen sind in der JavaDokumentation zu finden.

a) Look and Feel

In Swing ist es möglich das Erscheinungsbild der Java-Anwendungen zu ändern. Dies geschieht über das sogenannte Look and Feel der Anwendung.

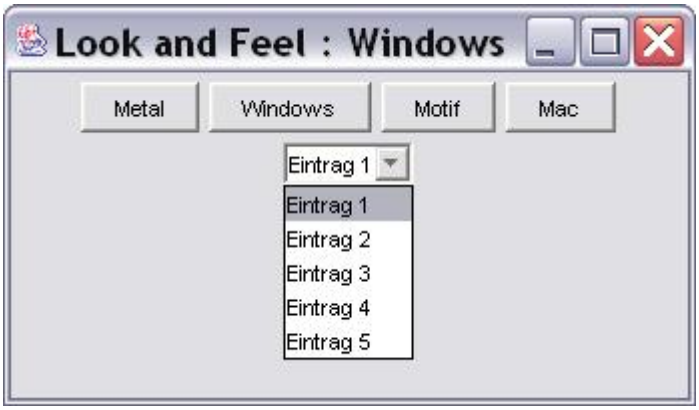
Einstellung des Look and Feel erfolgt folgendermaßen:

```
public static void main(String[]args){
    try {
        UIManager.setLookAndFeel(getCrossPlatformLookAndFeelClassName());
    } catch(Exception e) {}
    // ab hier wird das UserInterface erzeugt
    // Menues, Bottons, Labels und andere Komponenten
}
```

Es werden vier verschiedene Look and Feel – Arten von Java zur Verfügung gestelllt.
Der Standard LookAndFeel ist der Metal-LookAndFeel.
Es ist daraufhinzuweisen das aus lizenzrechtlichen Gründen nicht alle Motive in allen Betriebssystemen zur Verfügung stehen.

1.Variante: Metal-LookAndFeel
Als Standard-Variante ist es in allen Betriebssystemen verfügbar und muss nicht explizit implementiert werden, sondern wird Standard mäßig bereitgestellt.

2.Variante: Windows-LookAndFeel
Bei diesem LookAndFeel handelt es sich um eine Variante die speziell auf Windows zugeschnitten ist. Die Variante ist nicht für alle Plattformen z.B. für Apple Macintosh verfügbar.



3.Variante: Motif-LookAndFeel

Diese Variante ist typisch für die graphische Oberfläche von Sun Solaris-Maschinen.

4.Variante: Macintosh-LookAndFeel

Diese Variante ist typisch für die graphische Oberfläche von Macintosh-Maschinen.
Die Variante ist nicht für alle Plattformen z.B. für Windows verfügbar.

Beispiel zur Steuerung des LookAndFeel

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LookAndFeelExample extends JFrame implements ActionListener {

    // Globale Bezeichner fuer Look and Feels
    // 1. Metal
    static String metal = "javax.swing.plaf.metal.MetalLookAndFeel";
    // 2. Windows
    static String windows = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
    // 3. Motif
    static String motif = "com.sun.java.swing.plaf.mofif.MotifLookAndFeel";
    // 4. MacIntosh
    static String mac = "javax.swing.plaf.mac.MacLookAndFeel";

    static JButton button1 = new JButton("Metal");
    static JButton button2 = new JButton("Windows");
    static JButton button3 = new JButton("Motif");
    static JButton button4 = new JButton("Mac");

    // Zusaetzliches Element: JComboBox
    final static String[] eintraege = new String[] {
       "Eintrag 1",
       "Eintrag 2",
       "Eintrag 3",
       "Eintrag 4",
       "Eintrag 5",
    };

    static JComboBox combox = new JComboBox(eintraege);

    // Konstruktor der Klasse LookAndFeelExampel
    public LookAndFeelExample() {

        // Titel der Applikation
        super("LookAndFeelExampel");

        // Content pane ermitteln und Layout festlegen
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        // Hinzufügen der ActionListener
```

```
    button1.addActionListener(this);
    button2.addActionListener(this);
    button3.addActionListener(this);
    button4.addActionListener(this);

    // Hinzufügen der Schaltflaechen
    contentPane.add(button1);
    contentPane.add(button2);
    contentPane.add(button3);
    contentPane.add(button4);

    // Combox hinzufuegen
    contentPane.add(combox);

    // Content pane festlegen
    setContentPane(contentPane);
}

public void actionPerformed(ActionEvent e) {

    Object quelle = e.getSource();

    // Look and Feel : Metal
    if (quelle == button1) {
        try {
            this.setTitle("Look and Feel : Metal");
            UIManager.setLookAndFeel(metal);
        } catch(Exception e1) {
            this.setTitle("Hat leider nicht geklappt...");
        }
        SwingUtilities.updateComponentTreeUI(this.getContentPane());
    }

    // Look and Feel : Windows
    if (quelle == button2) {
        try {
            this.setTitle("Look and Feel : Windows");
            UIManager.setLookAndFeel(windows);
        } catch(Exception e2) {
            this.setTitle("Hat leider nicht geklappt...");
        }
        SwingUtilities.updateComponentTreeUI(this.getContentPane());
    }

    // Look and Feel : Motif
```

```
        if (quelle == button3) {
            try {
                this.setTitle("Look and Feel : Motif");
                UIManager.setLookAndFeel(motif);
            } catch(Exception e3) {
                this.setTitle("Hat leider nicht geklappt...");
            }
            SwingUtilities.updateComponentTreeUI(this.getContentPane());
        }

        // Look and Feel : Macintosh
        if (quelle == button4) {
            try {
                this.setTitle("Look and Feel : Macintosh");
                UIManager.setLookAndFeel(mac);
            } catch(Exception e4) {
                this.setTitle("Hat leider nicht geklappt...");
            }
            SwingUtilities.updateComponentTreeUI(this.getContentPane());
        }
    }

    // Eintrittspunkt in Applikation
    public static void main(String[] args) {
        JFrame rahmen = new LookAndFeelExample();
        WindowListener wl = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };
        rahmen.addWindowListener(wl);
        rahmen.setSize(300, 200);
        rahmen.setVisible(true);
    }
}
```

b) Borders

Über die Klasse JComponent von der die meisten Swing-Komponenten abgeleitet sind, ist es möglich über die Methode **setBorder()** verschiedene Rahmen zur Verfügung zu stellen.

AbstractBorder: Basisklasse der Rahmentypen. Standardmäßig ist dieser leer. Größe nicht festgelegt.

BevelBorder: Rahmen mit Schrägkanten. Dreidimensionaler Effekt.

CompondBorder: aus zwei Rahmen zusammengesetzter Rahmentyp

EmptyBorder: leerer und transparenter Rahmen. Rahmen wird quasi nicht angezeigt

EtchedBorder: Rahmen erscheint als ob etwas weggeätzt wäre

LineBorder: einfarbiger Rahmen beliebiger Stärke

MatteBorder: Effekt wie bei einem matten Foto

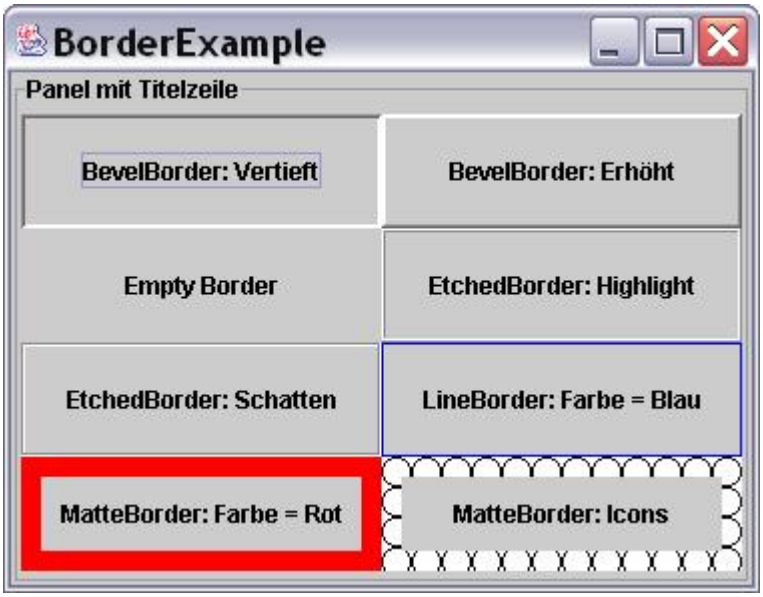
SoftBevelBorder: wie EtchedBorder, Kanten aber weich gezeichnet

TitledBorder: beschrifteter Rahmen

Das Border wird auf Komponenten wie Buttons, Listen und Boxen, aber auch für Panels angewendet.

Methode: panel.setBorder(BorderFactory.XYZBorder(XYZBorder.XYZSTATIC));

Die genauere Beschreibung der einzelnen Varianten ist über die Java- Dokumentation möglich.

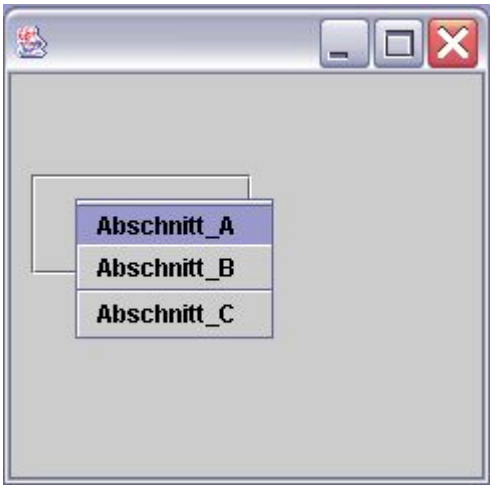



```
import java.awt.*;
import java.awt.event.*;
import java.awt.Color.*;
import javax.swing.*;
import javax.swing.border.*;

public class BorderExample extends JFrame {

    public BorderExample() {

        super("BorderExample");

        Container contentPane = getContentPane();
        JPanel panel = new JPanel();

        panel.setLayout(new GridLayout(4,2));

        JButton button1 = new JButton("BevelBorder: Vertieft");
        JButton button2 = new JButton("BevelBorder: Erhöht");
        JButton button3 = new JButton("Empty Border");
        JButton button4 = new JButton("EtchedBorder: Highlight");
        JButton button5 = new JButton("EtchedBorder: Schatten");
        JButton button6 = new JButton("LineBorder: Farbe = Blau");
        JButton button7 = new JButton("MatteBorder: Farbe = Rot");
        JButton button8 = new JButton("MatteBorder: Icons");
```

```
        panel.setBorder(BorderFactory.createTitledBorder("Panel mit Titelzeile"));
        button1.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
        button2.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
        button3.setBorder(BorderFactory.createEmptyBorder());
        button4.setBorder(BorderFactory.createEtchedBorder

    (getBackground().darker(),getBackground().brighter()));
        button5.setBorder(BorderFactory.createEtchedBorder

    (getBackground().brighter(),getBackground().darker()));
        button6.setBorder(BorderFactory.createLineBorder(Color.blue));
        button7.setBorder(BorderFactory.createMatteBorder(10,10,10,10,Color.red));
        button8.setBorder(BorderFactory.createMatteBorder
                                                (10,10,10,10,new
ImageIcon("grafik.gif")));
        panel.add(button1);
        panel.add(button2);
        panel.add(button3);
        panel.add(button4);
        panel.add(button5);
        panel.add(button6);
        panel.add(button7);
        panel.add(button8);
        contentPane.add(panel);
    }

    public static void main(String []args) {

        JFrame rahmen = new BorderExample();
        WindowListener wl = new WindowAdapter(){
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(wl);
        rahmen.setSize(300, 200);
        rahmen.setVisible(true);
    }
}
```

c) Popup-Menüs

Normale Menüs befinden sich am oberen Rand eines Fensters und sind immer sichtbar.
Bei Popup-Menüs verhält es sich anders, sie werden erst zur Laufzeit sichtbar und werden durch ein spezielles Ereignis erzeugt, z.B. einen ActionEvent an einem Button.
Popup-Menüs werden durch die Klasse JPopupMenu definiert und durch die Methode show() sichtbar gemacht. Die Methodik zum Aufbau eines Menüs läuft genauso ab wie bei einem normalen Menü. So werden als Untereintrag genauso JMenuItem verwendet.



d) JProgressBar

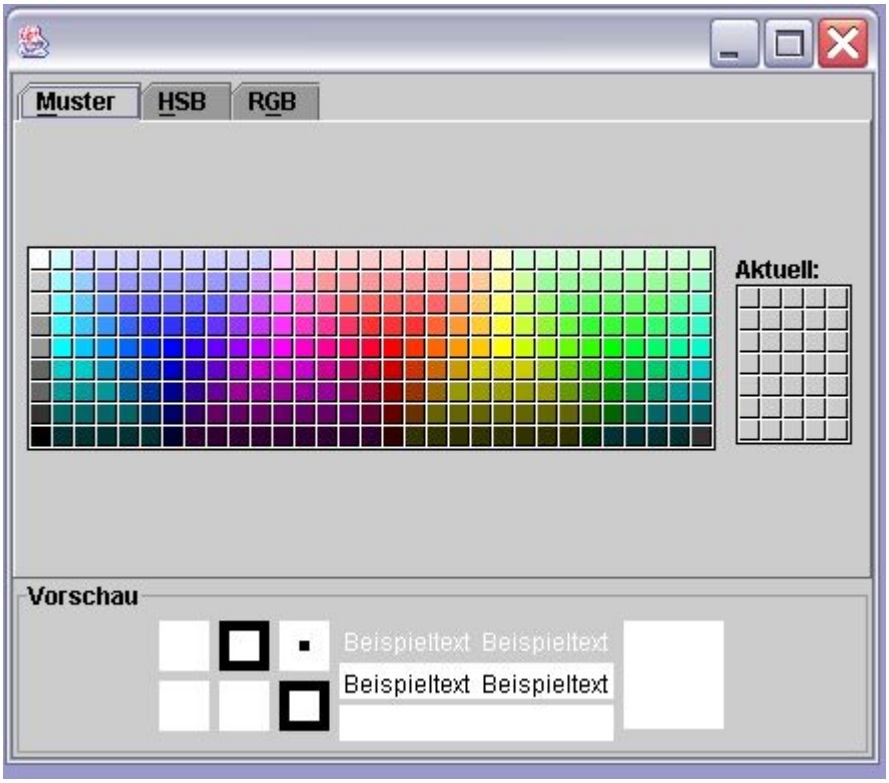
Um zu erkennen wie ein Programm bei der Abarbeitung eines Prozesses (Berechnungen, Dateizugriffe, Kopiervorgänge) fortschreitet wird ein ProgressBar eingesetzt. Der ProgressBar zeigt jeweils immer den aktuellen Fortschritt des Prozesses an bzw. das der Prozess noch aktiv ist und der Rechner noch nicht abgestürtzt ist.

Es gibt verschiedene Methoden um einen ProgressBar richtig einzustellen und die volle Funktionalität zu nutzen. Mit der Methode **setOrientation()** wird festgelegt, ob der ProgressBar horizontal oder vertikal angeordnet ist. Als Werte werden JProgressBar.VERTICAL und JProgressBar.HORIZONTAL eingesetzt.
Mit der Methode **setStringPainted()** kann optional der Fortschrittswert in Prozent angegeben werden. Die Methode ist boolean und hat nur true und false als mögliche Werte. Es ist allerdings auch möglich mit der Methode **setString()** einen festen Textwert anzugeben. Mit den Methoden **setMinimum()** und **setMaximum()** werden die Unter- und Obergrenze des Intervalls festgelegt. Die Werte lassen sich mit **getMinimum()** und **getMaximum()** ermitteln. Die Methode **getPercentComplete()** liefert einen double-Wert zurück, der den aktuellen Wert des Fortschrittsbalken darstellt.
Die wichtigste Methode ist **setValue()** durch die, das Programm in der Lage ist den Fortschrittsbalken zu ändern.

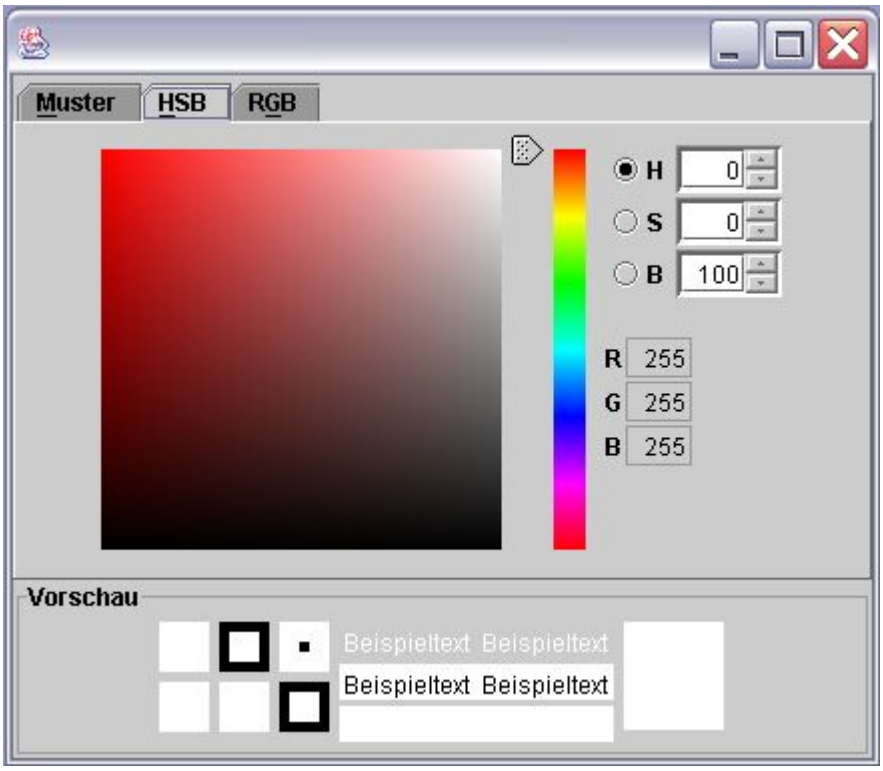
e) JColorChooser

Über diese Klasse wird eine Farbauswahl ermöglicht. Es bestehen drei verschiedene Möglichkeiten dies zu realiseren.

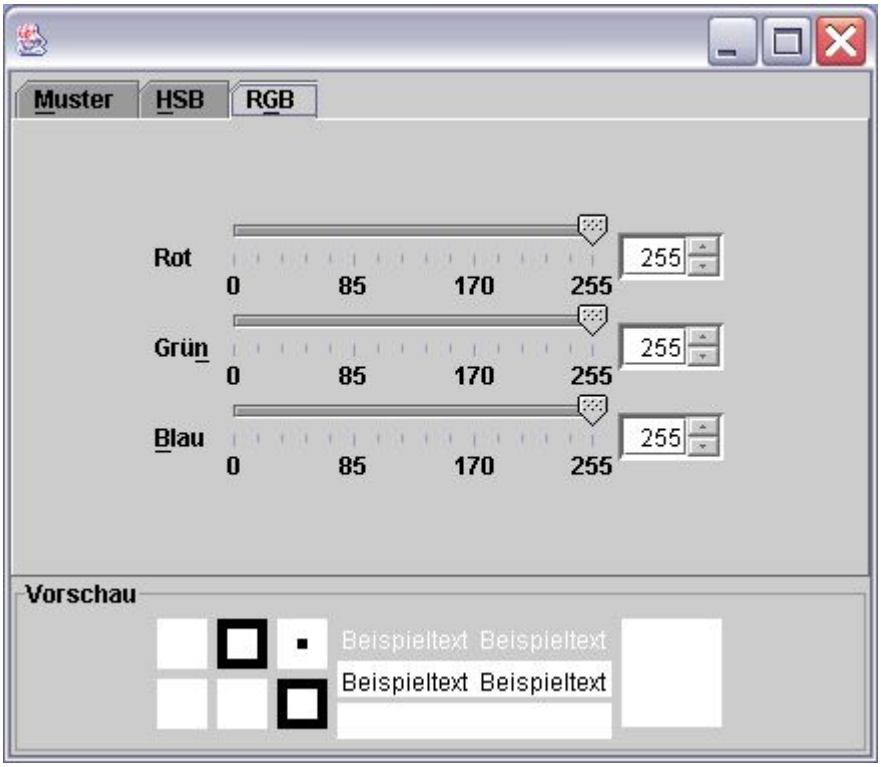
Die erste Möglichkeit ist die Auswahl über ein Muster. Die gewünschte Farbe muss nur ausgewählt werden und wird automatisch übernommen.



Die zweite Möglichkeit ist die Auswahl über einen Slider oder durch das Klicken auf einen Bereich im Area. Außerdem ist eine direkte Eingabe der Werte zusätzlich auch noch möglich.



Die dritte Möglichkeit besteht darin die Farbe über den Rot-Grün-Blau-Anteil selbst zu mischen.



f) JFileChooser

Ein sehr häufig verwendeter Anwendungsfall ist die Auswahl von Dateien. Folgende Dateioperationen müssen durchführbar sein: Öffnen, Speichern, Löschen, usw.

Der JFileChooser verfügt über mehrere Konstruktoren:

JFileChooser(File akt)
ein aktuelles Verzeichnis wird direkt dem Konstruktor übergeben

JFileChooser(File akt, FileSystemView fsv)
verhindert das löschen und anzeigen von Systemdateien, Dateiattributen oder versteckten Dateien

JFileChooser(FileSystemView fsv)
wie beim zweiten Konstruktor, nur ohne aktuelles Verzeichnis

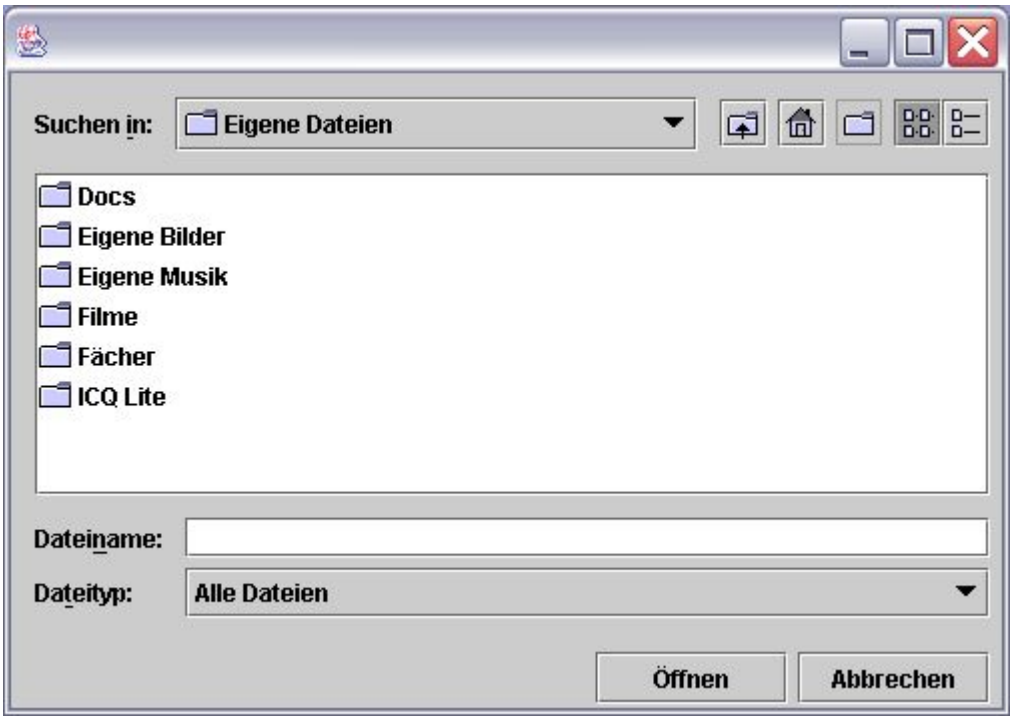
JFileChooser(String such)
Pfad für weitere Operationen wird direkt mit übergeben

JFileChooser(String such, FileSystemView fsv)
Kombination der letzten beiden

Die wichtigsten Methoden des JFileChooser

showSaveDialog()
öffnet Fenster zum Speichern von Dateien

showOpenDialog()
öffnet Fenster zum Öffnen von Dateien



g) JPasswordField

Das JPasswordField ist besondere Form des JTextField. Dort sollen Passwörter für den Zugang zu geschützten Daten eingegeben werden können. Um das ausspähen von Daten zu erschweren, werden die eingegebenen Zeichen schon bei der Eingabe durch andere Zeichen ersetzt.

Standardmäßig wird als Ersatzzeichen für die verdeckte Passworteingabe der Stern '*' verwendet. Mit der Methode **setEchoChar('&')** kann auch ein beliebig anderes Zeichen als ein Stern ausgewählt werden. Um das Password zu laden wird die Methode **getPassword()** verwendet. Die Methode liefert ein Array von Char-Objekten zurück. Um festzustellen ob überhaupt ein Ersatzzeichen gewählt ist verwendet man die Methode **echoCharIsSet()**.

h) verschiedene Panel-Typen

Bei AWT werden die Komponenten direkt einem Container hinzugefügt. In Swing werden die Komponenten Panel-Objekten zugeordnet. Swing stellt einige Panel-Typen zur Verfügung.

- **JLayeredPane** gibt der Anwendung eine gewisse Tiefe und macht Sie dreidimensional.

  Es gibt verschiedene Ebenen in denen Komponenten platziert werden können:

  DEFAULT_LAYER
  Ist die unterste Ebene auf der, die anderen Ebenen aufbauen und die, die meisten Komponenten aufnimmt.

  PALETTE_LAYER
  Diese Ebene nimmt bewegliche Elemente auf und befindet sich direkt über dem DEFAUL_LAYER. Komponenten in diesem Layer können andere Komponenten überlagern

  MODAL_LAYER
  Die Komponenten dieser Ebene überlagern immer alle anderen Komponenten

  POPUP_LAYER
  Diese Ebene enthält Popup-Dialoge, JToolText und andere Hilfskomponenten

  DRAG_LAYER
  Die Ebene liegt über allen anderen. Beim Verschieben liegen die Komponenten vorübergehend in dieser Ebene und kehren danach in die alte Ebene zurück.

  Zum Anzeigen und Ausblenden werden die Methoden **moveToFront()** und **moveToBack()** verwendet

- **JDesktopPane** erweitert die Funktionalität der Klasse JLayeredPane

  In diesem Pane werden normalerweise mehrere JInternalFrame platziert, wobei JInternalFrame jeder verschiedene Dokumente beinhalten kann

- **JViewport** hat die Funktionalität einer Lupe

  Über einen JViewport kann man Bilder und andere Dokumente die auf einer Seite keinen Platz haben verkleinern und vergrößern

- **JScrollPane** bietet über den JViewport hinaus noch Funktionalitäten an

  Wenn das Bild oder Dokument eine gewisse Größe überschritten hat, befinden sich Rollbalken an den Rändern. Außerdem können links und oben noch Lineale zum Überprüfen der Bildgröße angefügt werden.

- **JSplitPane** ermöglicht das wechselseitige Verkleinerung bzw. Vergrößern von zwei Komponenten

  Zwei Komponenten werden jeweils in den für sie vorgesehenen Bereich geladen. Wenn das eine Objekt verkleinert wird, wird das andere Objekt automatisch vergrößert und natürlich umgekehrt genauso.

i) JTable

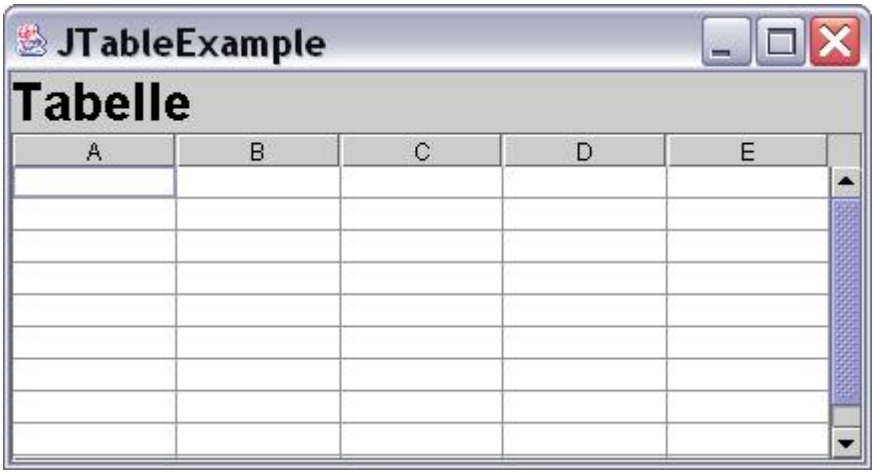
Ähnlich wie in Excel und Word ist es auch bei Swing möglich Tabellen einzufügen und zu bearbeiten. Die Klasse JTable stellt eine breite Palette von verschiedenen Möglichkeiten zur Bearbeitung von Tabellen bereit.

Im Regelfall wird eine Tabelle auf einem JScrollPane aufgesetzt um alle Werte der Tabelle anzeigen zu können.

Der Konstruktor der Klasse JTable: **JTable(spalte, zeile)**

Methoden zur Bearbeitung von Tabellen:

isCellEditable()	zeigt an ob ein Feld editierbar ist	Interface TableModel
setValue()	weißt einem Feld einen neuen Wert zu	
addColumn()	fügt Tabelle Spalte hinzu	Interface TableColumnModel
moveColumn()	verschiebt Spalte in Tabelle	



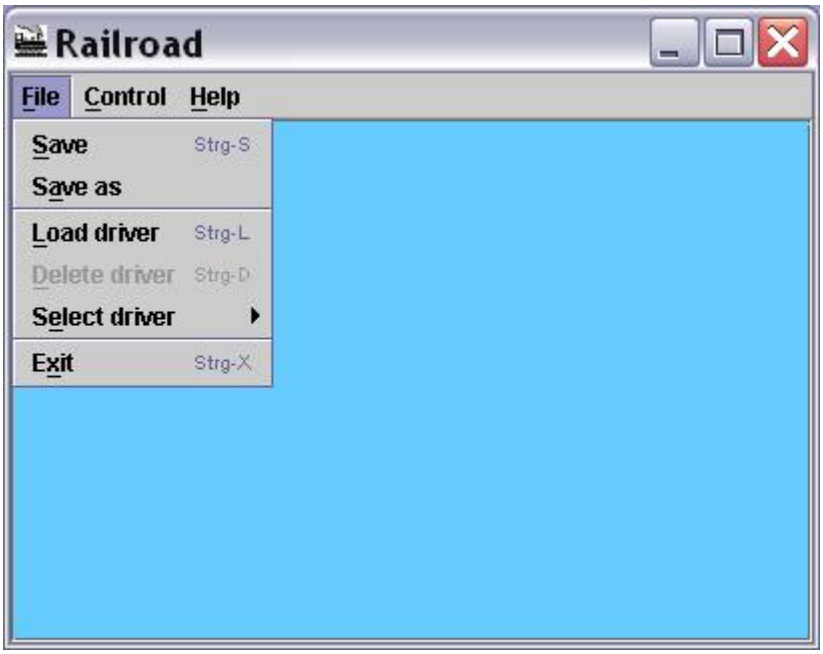
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JTableExample extends JFrame {

    JTable tab = new JTable(10, 5);

    public JTableExample() {

        super("JTableExample");

        Container contentPane = getContentPane();

        JScrollPane scrollPane = new JScrollPane(tab);
        JLabel label = new JLabel("Tabelle");

        label.setFont(new Font("Arial", Font.BOLD, 26));

        contentPane.setLayout(new BorderLayout());
        contentPane.add(label, BorderLayout.NORTH);
        contentPane.add(scrollPane, BorderLayout.CENTER);
    }

    public static void main(String []args) {

        JFrame rahmen = new JTableExample();
        WindowListener wl = new WindowAdapter(){
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(wl);
        rahmen.setSize(300, 200);
        rahmen.setVisible(true);
    }
}
```

j) Tooltip

Der Tooltip wird eingesetzt um Bedienoberflächen und Komponenten kurze Informationen beim Aufenthalt der Maus auf der Komponente mit zugeben. Die Klasse die, die Funktionalität zur Verfügung stellt ist **JToolTip**. Verwendete Methode heißt **setToolTipText(“Hier steht der Text“**);

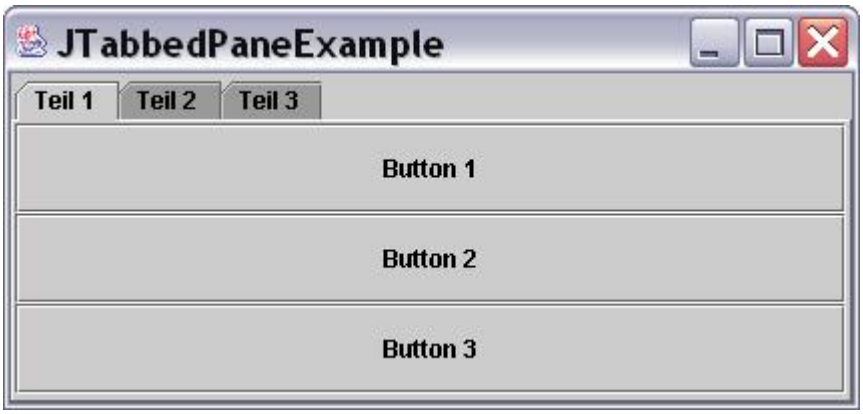
k) Shortcuts & Mnemonic

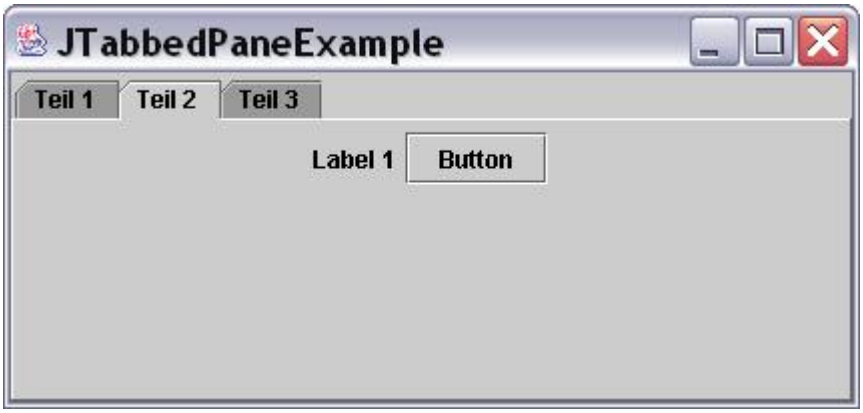
Über Shortcut und Mnemonic kann über Tastenkürzel eine Anwendung oder ein Menü gesteuert werden. Bekannte Shortcuts sind Strg-X zum Schließen einer Anwendung, oder Strg-S zum Speichern einer Anwendung. Die doppelte Verwendung desselben Kürzels ist verboten.
Ein Mnemonic wird mit der Alt-Taste + einen Buchstaben verwendet, wobei in demselbem Menü-Block immer jeder Buchstabe nur einmal verwendet werden darf. Im nächsten Menü darf der Buchstabe aber wieder verwendet werden. Der Mnemonic kennzeichnet sich dadurch, dass der Buchstabe, der als Mnemonic verwendet werden soll, im Wort unterstrichen ist.

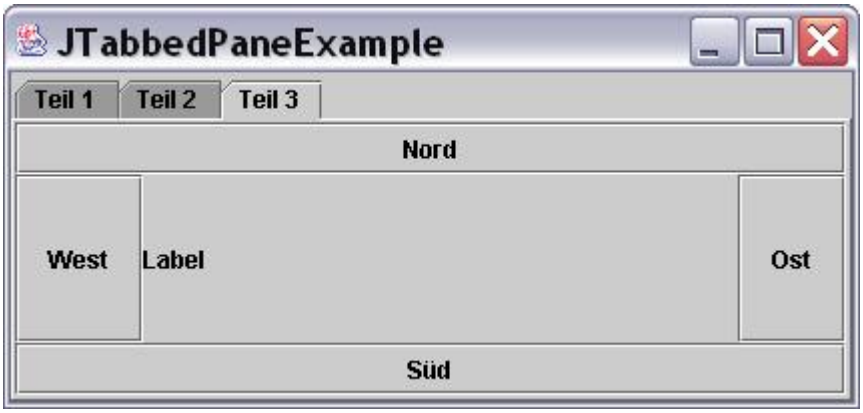


l) TabbedPane

Anwendungen, die mehrere unterschiedliche Oberflächen besitzen, lassen sich auf bequem über eine Art Karteikarte (TabbedPane) programmieren.







```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.URL;

public class JTabbedPaneExample extends JFrame {

    public JTabbedPaneExample() {

        super("JTabbedPaneExample");

        Container contentPane = getContentPane();

        JTabbedPane tabbedPane = new JTabbedPane();
        JPanel panel1 = new JPanel();
        JPanel panel2 = new JPanel();
        JPanel panel3 = new JPanel();

        panel1.setLayout(new GridLayout(3,0));
        panel1.add(new JButton("Button 1"));
        panel1.add(new JButton("Button 2"));
        panel1.add(new JButton("Button 3"));

        panel2.setLayout(new FlowLayout());
        panel2.add(new JLabel("Label 1"));
        panel2.add(new JButton("Button"));

        panel3.setLayout(new BorderLayout());
        panel3.add(new JButton("Nord"), BorderLayout.NORTH);
        panel3.add(new JButton("Ost"), BorderLayout.EAST);
        panel3.add(new JButton("Süd"), BorderLayout.SOUTH);
        panel3.add(new JButton("West"), BorderLayout.WEST);
        panel3.add(new JLabel("Label"));

        tabbedPane.add("Teil 1", panel1);
        tabbedPane.add("Teil 2", panel2);
        tabbedPane.add("Teil 3", panel3);

        contentPane.setLayout(new BorderLayout());
        contentPane.add(tabbedPane);
    }
```

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

        JFrame rahmen = new JTabbedPaneExample();
        WindowListener wl = new WindowAdapter(){
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        };

        rahmen.addWindowListener(wl);
        rahmen.setSize(300, 200);
        rahmen.setVisible(true);
    }
}
```

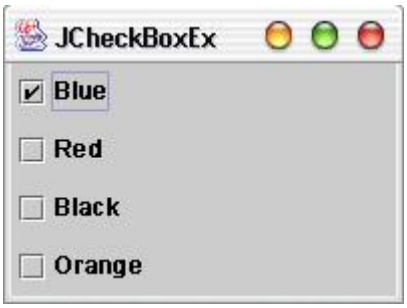
Übungen

1) JButton mit Icon

Aufgabenstellung:
Schreiben Sie eine Anwendung mit einem JFrame (Groesse: *200x100*,
Titel: " *JButton & Icon* "). In diesem Frame befindet sich ein JButton,
beschriftet durch "*Button mit Icon*" und einem ImageIcon.
Die Anwendung soll sich beenden, wenn der Frame geschlossen wird.

2) JCheckBoxes

Aufgabenstellung:
Schreiben Sie eine Anwendung mit einem JFrame (Groesse: *200x150*,
Titel: " *JCheckBoxEx* "). In diesem Frame befinden sich vier JCheckBoxes innerhalb einer ButtonGroup. Jede der CheckBoxes ist mit einer der Farben *Blue*, *Red*, *Black* und *Orange* beschriftet. Beim Starten der Anwendung ist *Blue* ausgewaehlt.
Die Anwendung soll sich beenden, wenn das Fenster geschlossen wird.



3) JProgressBar

Aufgabenstellung:
Schreiben Sie eine Anwendung mit einem JFrame (Groesse: *200x100*,
Titel: "*JProgressBarEx*"). Innerhalb des Frames befindet sich ein JProgressBar und zwei JButtons mit den Beschriftungen "*Start*" und "*Stop*".
Beim Drueken des Start-Buttons wird ein Thread initialisiert, der das Fortschreiten der ProgressBar bewirkt. Die ProgressBar nimmt dabei Werte zwischen *0* und *200* an, zwischen dem Hochzaehlen schlaeft der Thread jeweils fuer *60ms.*
Wird der Stop-Button gedruekt stoppt die ProgressBar und setzt den Fortschritt wieder auf *0*, so dass sie beim erneuten Starten wieder von vorne anfaengt.
Die Anwendung soll sich beenden, wenn das Fenster geschlossen wird.

Tipp: Implementieren sie in ProgressThread Variablen, die zum Stoppen gesetzt werden koennen.