Grundlagen der Informatik für Ingenieure I

15 AWT - Abstract Window Toolkit - Teil2

15.1 Geschachtelte Komponenten

- *AWT - Panels , Applets* ist eines davon, können geschachtelt werden, wobei jedes *Panel* über seinen eigenen *layout manager* verfügen kann.

- Die Vorgehensweise ist analog der beim Kreieren von primitiven Komponenten:

```
setLayout( new GridLayout( 1, 2, 10, 10 ) );
Panel panel1 = new Panel();
Panel panel2 = new Panel();
add( panel1 );
add( panel2 );
```

Panel sind Objekte; sie werden referenziert über ihre Variablen. Das individuelle Layout des *panel1* kann also wie folgt bestimmt werden:

```
panel1.setLayout( new FlowLayout() );
panel1.add( new Button( “up”) );
panel1.add( new Button(“Down”) );
```

15.2 Weitere UI - Komponenten

- Folgende weitere Komponenten werden in diesem Kapitel eingeführt :
  - *Text Areas*
  - *Scrolling Lists*
  - *Scroll Bars und Sliders*
  - *Scrolling Panes*
  - *Canvases*
  - *Cursors*

  wobei jeweils kurz
  - die Definition
  - die Konstruktoren
  - die (gängigen) Methoden
  - und das *Event-Handling*

  dargestellt werden, gefolgt von jeweils einem Beispiel.

1 Text Areas

- Während *TextFields* nur eine Zeile Text enthalten können, können *TextAreas* beliebigen Text enthalten, wobei je nach Konstruktor dann Scrollbars bereitgestellt werden, wenn der Text im vordimensioniertem Fenster nicht vollständig dargestellt werden kann .

- Konstruktoren:
  - *TextArea()* - Kreiert eine leere *TextArea*. In Abhängigkeit vom *layout manager* wird die Größe des Bereichs automatisch dem jeweils vorhandenen Text angepasst.
  - *TextArea(String)* - Kreiert eine *TextArea* mit dem gegebenen Text-String. In Abhängigkeit vom *layout manager* wird die Größe des Bereichs automatisch dem jeweils vorhandenen Text angepasst.
  - *TextArea(String, int, int)* - Kreiert eine *TextArea* mit dem gegebenen Text-String in der durch den 2. und 3. Parameter bestimmten Dimension.

1 Text Areas

- *TextArea(String, int, int, int)* - Kreiert eine *TextArea* mit dem gegebenen Text-String in der durch den 2. und 3. Parameter bestimmten Dimension. Zusätzlich wird durch den 4. Parameter das Scrollbar-Layout bestimmt:
  - TextArea.SCROLLBARS_BOTH
  - TextArea.SCROLLBARS_HORIZONTAL_ONLY
  - TextArea.SCROLLBARS_VERTICAL_ONLY
  - TextArea.SCROLLBARS_NONE

1 Text Areas

- Beispiel:

```
/* text areas */

import java.awt.*;

public class TextAreaTest extends java.applet.Applet {

  public void init() {
    String str =
     "Once upon a midnight dreary, while I pondered, weak and weary,\n" +
     "Over many a quaint and curious volume of forgotten lore,\n" +
     "While I nodded, nearly napping, suddenly there came a tapping,\n" +
     "As of some one gently rapping, rapping at my chamber door.\n" +
     "\"'Tis some visitor,\" I muttered, \"tapping at my chamber door-\n" +
     "Only this, and nothing more.\"\n\n" +
     "Ah, distinctly I remember it was in the bleak December,\n" +
     "And each separate dying ember wrought its ghost upon the floor.\n" +
     "Eagerly I wished the morrow;- vainly I had sought to borrow\n" +
     "From my books surcease of sorrow- sorrow for the lost Lenore-\n" +
     "For the rare and radiant maiden whom the angels name Lenore-\n" +
     "Nameless here for evermore.";

     add( new TextArea( str, 10, 30, TextArea.SCROLLBARS_BOTH ) );
  }
}
```

1 Text Areas

- *TextArea Methods*

Method	Action
getColumns()	Returns the width of the text area, in characters or columns
getRows()	Returns the number of rows in the text area (not the number of rows of text that the text area contains)
insert(String, int)	Inserts the string at the given position in the text (text positi-ons start at 0)
replace(String, int, int)	Replaces the text between the given integer positions with the new string

- Eventhandling
  Hier unterscheidet sich das *Handling* nicht vom den des *TextFields:*
  - *addFocusListener()*
    - *focusGained()*
    - *FocusLost()*
  - *addTextListener()*
    - *textValueChanged()*

2 Scrolling Lists

- *ScrollingLists* sind dem *ChoiceMenu* ähnlich, jedoch wird statt der *Checkboxes* eine Liste angeboten, deren Einträge man selektieren kann. Die Anzahl der sichtbaren Einträge ist konfigurierbar.

- Konstruktoren:
  Man instantiiert erst die *ScrollingLists* und fügt dann mit *add()* die *items* hinzu:
  - *List()* - Kreiert eine leere *ScrollingList*; nur ein *item* ist selektierbar
  - *LIST(int)* - Kreiert eine *ScrollingList* mit einer gegebenen Anzahl von sichtbaren *item-slots*, nur ein *item* ist selktierbar.
  - *List(int, boolean)* - Kreiert eine *ScrollingList* mit einer gegebenen Anzahl von sichtbaren *item-slots* und der Möglichkeit mehrere *items* zu selktieren, wenn der zweite Parameter den Wert *“true”* hat.

2 Scrolling Lists

- Beispiel:

```
/* scrolling lists */

import java.awt.*;

public class ListsTest extends java.applet.Applet {

  public void init() {
    List lst = new List(5, true);

    lst.add( "Hamlet" );
    lst.add( "Claudius" );
    lst.add( "Gertrude" );
    lst.add( "Polonius" );
    lst.add( "Horatio" );
    lst.add( "Laertes" );
    lst.add( "Ophelia" );

   add(lst);
  }
}
```


2 Scrolling Lists

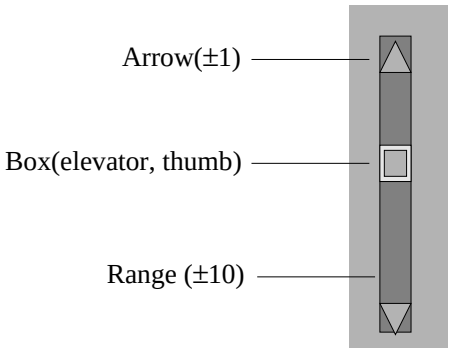
- *ScrollingList Methods*

Method	Action
getItem(int)	Returns the string item at the given position
getItemCount()	Returns the number of items in the menu
getSelectedIndex()	Returns the index position of the item that's selected (used for lists that allow only single selections)
getSelectedIndexes()	Returns an array of index positions (used for lists that allow multiple selections)
getSelectedItem()	Returns the currently selected item as a string
getSelectedItems()	Returns an array of strings containing all the selected items
select(int)	Selects the item at the given position

- Eventhandling
  - *addActionListener()*
    - *actionPerformed()*
  - *addItemListener()*
    - *itemStateChanged() - list select/deselect*
    - *getItem() - welches item?*
    - getStateChange() - item select/deselect

3 Scroll Bars und Slider

- Bisher waren *Scrollbars* Teile von Komponenten.
- Man hat jedoch auch die Möglichkeit, *Srollbars* als seperate Komponenten einzuführen um einen Interval von Integerwerten zu manipulieren.
- Diese Werte sind dann im Applet oder in der Applikation geeignet auf “Konstrukte” abzubilden auf die der *Scrollbar* wirken soll.



3 Scroll Bars und Slider

- *Scrollbars* bestehen aus 3 Komponenten:
  - Pfeile an beiden Enden, die üblicherweise “kleineren” Veränderungen dienen (+, - 1)
  - einen Bereich zwischen den Pfeilen, der “größeren” Veränderungen dient(+,- 10) und
  - einen Knopf (*thumb, elevator*) mit dem man in quasi-analoger Weise innerhalb des “Intervals” die Werte verändert, wobei der Ort des Knopfes die relative Position innerhalb des “Intervals” beschreibt.

3 Scroll Bars und Slider

- Konstruktoren:
  - *Scrollbar()* - kreiert einen *Scrollbar* in vertikaler Orientierung, Minimum/Maximum = 0.
  - *Scrollbar(int)* - kreiert einen *Scrollbar*, Minimum/Maximum = 0. Der Parameter beschreibt die Orientierung mit
    - Scrollbar.HORIZONTAL
    - Scrollbar.VERTICAL
  - *Scrollbar(int, int, int, int, int)* - kreiert einen Scrollbar wobei
    - der 1. Parameter die Orientierung mit
      - Scrollbar.HORIZONTAL und
      - Scrollbar.VERTICAL,
    - der 2. Parameter den initialen Wert,
    - der 3. Parameter die Höhe bzw. die Weite , je nach Orientierung
    - der 4. Parameter und 5 Parameter Minimum und Maximum definieren.

3 Scroll Bars und Slider

```
/* sliders (scrollbars) */
import java.awt.*;
import java.awt.event.*;

public class SliderTest extends java.applet.Applet
  implements AdjustmentListener {
  Label l;

  public void init() {
    setLayout(new GridLayout(1,2));
    l = new Label("1", Label.CENTER);
    add(l);
    Scrollbar sb = new
       Scrollbar(Scrollbar.HORIZONTAL,0,0,1,100);
    sb.addAdjustmentListener(this);
    add(sb);
  }
  public Insets getInsets() {
    return new Insets(15,15,15,15);
  }
  public void adjustmentValueChanged(AdjustmentEvent e) {
      int v = ((Scrollbar)e.getSource()).getValue();
      l.setText(String.valueOf(v));
      repaint();
  }
}
```

3 Scroll Bars und Slider

Method	Action
getMaximum()	Returns the maximum value
getMinimum()	Returns the minimum value
getOrientation()	Returns the orientation of this scroll bar: 0 is Scrollbar.HORIZONTAL; 1 is Scrollbar.VERTICAL.
getValue()	Returns the scroll bar's current value
setValue(int)	Sets the current value of the scroll bar
setUnitIncrement(int inc)	Changes the increment for how far to scroll when the endpoints of the scroll bar are selected. The default is 1.
getUnitIncrement(int inc)	Returns the increment for how far to scroll when the endpoints of the scroll bar are selected
setBlockIncrement(int inc)	Changes the increment for how far to scroll when the inside range of the scroll bar is selected. The default is 10
getBlockIncrement()	Returns the increment for how far to scroll when the inside range of the scroll bar is selected

3 Scroll Bars und Slider

- Eventhandling:
  - *addAdjustmentListener()*
    - *adjustmentValueChanged()*
    - *getAdjustmenType()*

Event ID	What It Represents
SCROLL_ABSOLUTE	Generated when a scroll bar's box has been moved
SCROLL_LINE_DOWN	Generated when a scroll bar's bottom or left endpoint (button) is selected
SCROLL_LINE_UP	Generated when a scroll bar's top or right endpoint (button) is selected
SCROLL_PAGE_DOWN	Generated when the scroll bar's field below (or to the left of) the box is selected
SCROLL_PAGE_UP	Generated when the scroll bar's field above (or to the right of) the box is selected

4 Scrolling Panes

- Um die Implementierung von scrollbaren Flächen zu vereinfachen wurden mit Java 1.1 die *Scrolling Panes* eingeführt.

- Konstruktoren:
  - *ScrollPane()* - kreiert eine *ScrollPane*. Scrollbars werden automarisch dann hinzugefügt, wenn die Komponente innerhalb des *ScrollPane* größer ist als das *Pane* selbst.
  - *ScrollPane(int)* - kreiert eine *ScrollPane*. Mit dem Parameter kann man die gewünschten Eigenschaften hinsichtlich der Scrollbars beeinflussen:
    - ScrollPane.SCROLLBARS_ALWAYS
    - ScrollPane.SCROLLBARS_AS_NEEDED
    - ScrollPane.SCROLLBARS_NEVER

4 Scrolling Panes

- *ScrollPane Methods*:

Method	Action
getScrollPosition()	Returns a Point object representing the position within the child that is displayed at the top-left corner of the pane
setScrollPosition(int, int)	Scrolls the panel to the given position within the child
setScrollPosition(Point)	Scrolls the panel to the given Point position within thechild
getHAdjustable()	Returns an Adjustable object representing the state of the horizontal scroll bar
getVAdjustable()	Returns an Adjustable object representing the state of thevertical scroll bar
getViewportSize()	Returns a Dimension object representing the size of the scroll panel's view port

- Eventhandling:
  - *addAdjustmentListener()*

5 Canvases

- *Canvases* sind einfache *Panels*, die nur für die Ausgabe von Graphik und *Images* geeignet sind.

- Beispiel:

```
Canvas can = new Canvas();
add(can);
```


6 Cursors

- *Cursors* sind Symbole, die die Position der Maus anzeigen.

- Es ist ab Java 1.1 möglich zu jeder *Component* spezielle *Cursor* zu definieren.

- Hierzu dient die Methode *setCursor()*, als Argument wird ein *Cursor Object* übergeben.

- Weitere Methoden:
  - *int getType():* der Returnparameter ist einer der Klassenkonstanten die den *Cursor-Typ* bezeichnen (Siehe Tabelle nächste Seite).
  - *Cursor getPredifinedCursor(int):* Parameter ist eine der Klassenvariablen. Sollte dieser *Cursor* im *"current" Environment* nicht verfügbar sein, wird die *Run-Time-Exception* **`IllegalArgumentException`** ausgeöst. (Es nicht zu erwarten, dass jede denkbare Umgebung über den vollen *Cursorsatz* verfügt)
  - *Cursor getDefaultCursor():* gibt den vordefinierten *Default-Cursor* zurück.

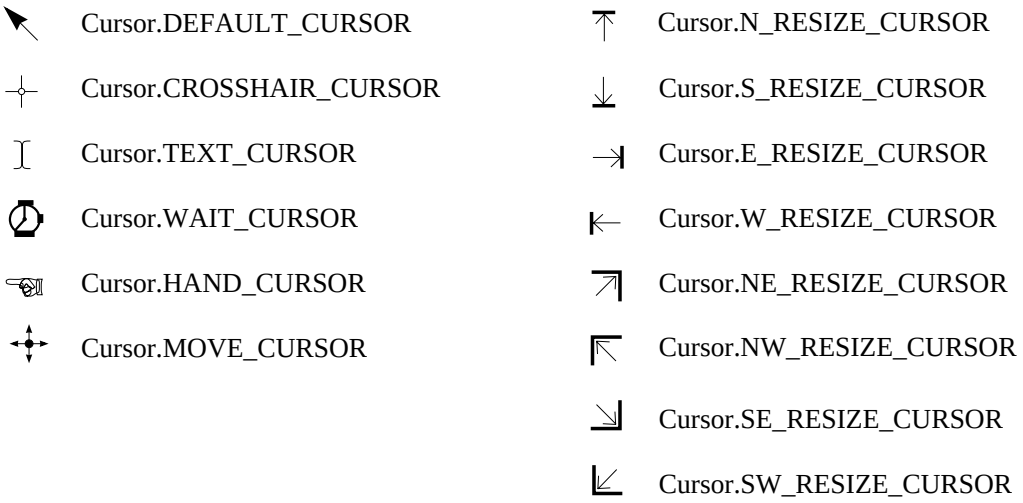
6 Cursors

- Konstruktoren:
  - *Cursor(int)* - kreiert einen Cursor, wobei der Wert des Parameters das Symbol nach folgender Tabelle bestimmt (Klassenvariable):

Class Variable	Cursor
Cursor.CROSSHAIR_CURSOR	A cross-hair (plus-shaped) cursor
Cursor.DEFAULT_CURSOR	The default cursor (usually a pointer or arrow)
Cursor.E_RESIZE_CURSOR	A cursor to indicate something is being resized
Cursor.HAND_CURSOR	A hand-shaped cursor (to move an object or the background)
Cursor.MOVE_CURSOR	A cursor to indicate that something is being moved
Cursor.N_RESIZE_CURSOR	The top edge of a window is being resized
Cursor.NE_RESIZE_CURSOR	The top-right corner of a window is being resized
Cursor.NW_RESIZE_CURSOR	The top-left corner of a window is being resized
Cursor.S_RESIZE_CURSOR	The bottom edge of a window is being resized
Cursor.SE_RESIZE_CURSOR	The bottom-right corner of the window is being resized
Cursor.SW_RESIZE_CURSOR	The bottom-left corner of the window is being resized
Cursor.TEXT_CURSOR	A text-entry cursor (sometimes called an I-beam)
Cursor.W_RESIZE_CURSOR	The left edge of a window is being resized
Cursor.WAIT_CURSOR	A long operation is taking place (usually an icon focus watch or an hourglass)

6 Cursors

- Symbole:



7 Components

- Die *Component class* ist die Superklasse aller *AWT*-Objekte, d. h. also, hier sind die Methoden definiert, die für alle *UI*-Elemente relevant sind:

Method	What It Does
getBackground()	Returns a Color object representing the component's background color
setBackground(Color)	Sets the component's background color.
getForeground()	Returns a Color object representing the component's current foreground color
setForeground(Color)	Sets the component's foreground color
getFont()	Returns a Font object representing the component's current font
setFont(Font)	Changes the component's current font
getSize()	Returns a Dimension object representing the component's current size You can then get to the individual width and height using size().width and size().height.
getMinimumSize()	The component's smallest possible size as a Dimension object. minimumSize() is usually used only by layout managers to determine how small it can draw a component; if you create a custom component, you'll want to override this method to return the minimum size of that component
getPreferredSize()	The component's preferred size (usually equal to or larger than the component's minimumSize()) as a Dimension object.

7 Components

Method	What It Does
setSize()	Changes the size of the applet to be the vurrent size. For custom components, you'll want to also call validate() after resizing the applet so that the layout can be redrawn.
contains(int, int)	Returns true if the given x and y coordinates are inside the component.
setSize()	Changes the size of the applet to be the vurrent size. For custom components, you'll want to also call validate() after resizing the applet so that the layout can be redrawn.
setVisible(boolean)	Returns true if the given x and y coordinates are inside the component. Shows a component previously hidden.
isVisible()	Returns true or false depending on whether this component is visible (not hidden).
isEnabled()	Returns true or false depending on whether the component is enabled.

15.3 RGB-To HSB-Converter

```
/* UI applet to convert RGB to HSB, and vice versa */

  import java.awt.*;

  public class ColorTest extends java.applet.Applet {
    private ColorControls RGBcontrols, HSBcontrols;
    private Canvas swatch;

  public void init() {
    setLayout(new GridLayout(1,3,5,15));

  // The color swatch
    swatch = new Canvas();
    swatch.setBackground(Color.black);

  // the subpanels for the controls
    RGBcontrols = new ColorControls(this, "Red", "Green", "Blue");
    HSBcontrols = new ColorControls(this, "Hue", "Saturation",
                                          "Brightness");
  //add it all to the layout
    add(swatch);
    add(RGBcontrols);
    add(HSBcontrols);
  }
```


15.3 RGB-To HSB-Converter

```
  public Insets insets() {
    return new Insets(10,10,10,10);
  }
  void update(ColorControls controlPanel) {
    Color c;
  // get string values from text fields, convert to ints
    int value1 = Integer.parseInt(controlPanel.tfield1.getText());
    int value2 = Integer.parseInt(controlPanel.tfield2.getText());
    int value3 = Integer.parseInt(controlPanel.tfield3.getText());

    if (controlPanel == RGBcontrols) {// RGB has changed, convert HSB
      c = new Color(value1, value2,value3);

    // convert RGB values to HSB values
    float[] HSB = Color.RGBtoHSB(value1, value2, value3,
                                          (new float[3]));
    HSB[0] *= 360;
    HSB[1] *= 100;
    HSB[2] *= 100;

    // reset HSB fields
    HSBcontrols.tfield1.setText(String.valueOf((int)HSB[0]));
    HSBcontrols.tfield2.setText(String.valueOf((int)HSB[1]));
    HSBcontrols.tfield3.setText(String.valueOf((int)HSB[2]));
  }
```

15.3 RGB-To HSB-Converter

```
    else {
       // HSB has changed, update RGB
       c = Color.getHSBColor((float)value1 / 360,
                             (float)value2 / 100, (float)value3 / 100);

     // reset RGB fields
       RGBcontrols.tfield1.setText(String.valueOf(c.getRed()));
       RGBcontrols.tfield2.setText(String.valueOf(c.getGreen()));
       RGBcontrols.tfield3.setText(String.valueOf(c.getBlue()));
     }

   //update swatch
   swatch.setBackground(c);
   swatch.repaint();
   }
}
```


15.3 RGB-To HSB-Converter

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

class ColorControls extends Panel implements FocusListener,
ActionListener {
  TextField tfield1, tfield2, tfield3;
  ColorTest applet;

  ColorControls(ColorTest parent, String l1, String l2, String l3) {

  // get hook to outer applet parent
    applet = parent;

  //do layouts
    setLayout(new GridLayout(3,2,10,10));

    tfield1 = new TextField("0");
    tfield2 = new TextField("0");
    tfield3 = new TextField("0");

    add(new Label(l1, Label.RIGHT));
    tfield1.addFocusListener(this);
    tfield1.addActionListener(this);
    add(tfield1);
```

15.3 *RGB-To HSB-Converter*

```
    add(new Label(l2, Label.RIGHT));
    tfield2.addFocusListener(this);
    tfield2.addActionListener(this);
    add(tfield2);
    tfield3.addFocusListener(this);
    tfield3.addActionListener(this);
    add(new Label(l3, Label.RIGHT));
    add(tfield3);
  }

  public Insets getInsets() {
    return new Insets(10,10,0,0);
  }

  public void focusGained(FocusEvent e) {}
  public void focusLost(FocusEvent e) {
    applet.update(this);
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof TextField)
      applet.update(this);
  }
}
```


15.4 Applikationen mit AWT

- Selbstverständlich kann das AWT auch von Applikationen genutzt werden.
- Die “*main-applikation” class* erbt von der *Frame class* und implementiert z. B. das *Runnable Interface*, falls mehr als ein Thread verwendet werden soll.

```
class MyAWTApplication extends Frame
                  implements Runnable{
..}
```

- In der *main() method* wird zunächst ein Objekt dieser Klasse erzeugt.
- Da dieses Objekt von der *Frame class* erbt, wird ein *AWT-Window* kreiert, welches man dann erst einmal in der Größe so wie man es braucht korregiert und sichtbar macht.
- Die Dinge, die wir beim Applet in der Regel in der *init() method* implementiert haben, werden bei einer Applikation in der *main() method* implementiert.

15.4 Applikationen mit AWT

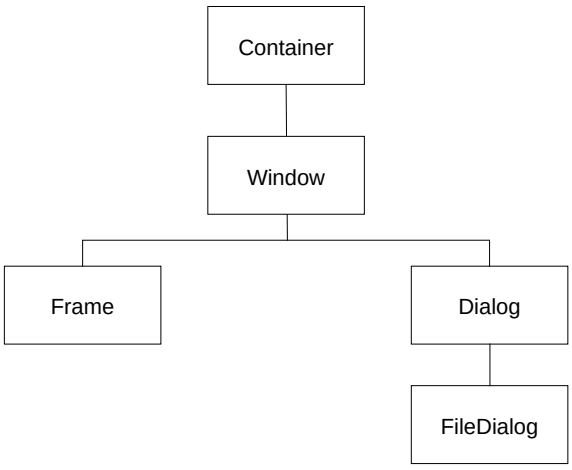
- Eine einfache AWT-Applikation:

```
import java.awt.*;

class MyAWTApplication extends Frame {

  MyAWTApplication(String titel) {
    super(titel);
    setLayout(new FlowLayout());
    add(new Button("OK"));
    add(new Button("Reset"));
    add(new Button("Cancel"));
    add(new Button("Hey!"));
  }

  public static void main(String args[]) {

  MyAWTApplication app = new MyAWTApplication(
                                "Hi! I'm an application");
    app.setSize(300,300);
    app.show();
  }
}
```


15.5 Windows, Frames und Dialog Boxes

- Zusätzlich zu den bisher besprochenen UI-Komponenten bietet das AWT auch Komponenten “außerhalb” einer Applet- und bzw. Browser-Umgebung an.

- Die Klassenhierachie zeigt die folgende Graphik:



1 Frames

- *Frames* sind Fenster mit
  - Titeln
  - *resize handles*
  - *close boxes*
  - *menu bars,*
  - etc.

  Sie erben von der *Windows -, Container -, Component class,* sodass alle bisher behandelten *UI* - Komponenten in gleicher Weise integriert werden können.

- Konstruktoren:
  - *Frame()* - kreiert einen *basic frame* ohne Titel
  - *Frame(String)* - kreiert einen *basic frame* mit einem Titel

1 Frames

- Kreiern eines *Windows* am Beispiel

```
window1 = new Frame("My Window");
window1.setLayout(new BorderLayout(10,20));
window1.add("North", new Button("Start"));
window1.add("Center", new Button("Move"));
```

  - Setzen der Größe: resize(width, hights)

```
window1.setSize(100, 200);
```

  - Mit *pack()* wird eine systemabhängige Minimalgröße definiert, die ausreicht sämtlich definierten Elemente darzustellen.

```
window1.pack();
```

  - Da ein kreiertes *Window* zunächst unsichtbar ist, muss man es explizit sichtbar machen: *show()*

```
windows1.show();
```

  - Das Gegenteil von *show()* ist *hide()*

```
windows1.hide();
```

15.5.1.1 Frames-Applet-Beispiel

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

public class PopupWindow extends java.applet.Applet {
    Frame window;
    Button open, close;

    public void init() {
        PopupActions handlebutton = new PopupActions(this);

        open = new Button("Open Window");
        open.addActionListener(handlebutton);
        add(open);

        close = new Button("Close Window");
        close.addActionListener(handlebutton);
        add(close);

        window = new BaseFrame1("A Popup Window");
        window.resize(150,150);
        window.show();
    }
}
```


15.5.1.1 Frames-Applet-Beispiel

```
/* actions for popup window */
import java.awt.*;
import java.awt.event.*;

public class PopupActions implements ActionListener {
  PopupWindow theApp;
  PopupActions(PopupWindow win) {
    theApp = win;
  }
  public void actionPerformed(ActionEvent e) {

    if (e.getSource() instanceof Button) {

      if (e.getSource() == theApp.open) {
        if (!theApp.window.isShowing())
          theApp.window.show();
      }
      else if (e.getSource() == theApp.close) {
        if (theApp.window.isShowing())
          theApp.window.hide();
      }
    }
  }
}
```

15.5.1.1 Frames-Applet-Beispiel

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

class BaseFrame1 extends Frame {
    String message = "This is a Window";
    Label l;

    BaseFrame1(String title) {
        super(title);
        setLayout(new BorderLayout());

        l = new Label(message, Label.CENTER);
        l.setFont(new Font("Helvetica", Font.PLAIN, 12));
        add("Center", l);
    }

    public Insets getInsets() {
        return new Insets(20,0,25,0);
    }
}
```


15.5.1.2 Frames-Application-Beispiel

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

public class WindowbyApplication extends Frame {
  private Frame popUpWindow;
  private Button open, close, exit;

    WindowbyApplication (String titel) {

      super(titel);
      setLayout(new FlowLayout());

      open = new Button("Open Window");
      add(open);

      close = new Button("Close Window");
      add(close);

      exit = new Button("Terminate Application");
      add(exit);

      popUpWindow = new BaseFrame1("A Popup Window");
      popUpWindow.setSize(300,300);
      popUpWindow.show();
```

15.5.1.2 Frames-Application-Beispiel

```
        PopupActions handlebutton = new PopupActions(
                                this,popUpWindow, open, close, exit);

        open.addActionListener(handlebutton);
        close.addActionListener(handlebutton);
        exit.addActionListener(handlebutton);
      }

      void terminate(){
        this.hide();
        System.exit(0);
      }

      public static void main (String args[]) {

        WindowbyApplication wapp = new  WindowbyApplication
                                            ("Application-Window");
        wapp.setSize(600,300);
        wapp.show();
      }
}
```


15.5.1.2 Frames-Application-Beispiel

```
/* actions for popup window */

import java.awt.*;
import java.awt.event.*;

public class PopupActions implements ActionListener{

  private  WindowbyApplication theApp;
  private Frame popUpWindow;
  private Button open, close, exit;

  PopupActions(WindowbyApplication win, Frame puw,
                                Button o, Button c, Button e) {

    theApp = win;
    popUpWindow = puw;
    open = o;
    close = c;
    exit = e;
  }
```

15.5.1.2 Frames-Application-Beispiel

```
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof Button) {

    if (e.getSource() == open) {
      if (!popUpWindow.isShowing())
        popUpWindow.show();
      }
      else if (e.getSource() == close) {
        if (popUpWindow.isShowing())
          popUpWindow.hide();
        }

        else if (e.getSource() == exit) {
          theApp.terminate();
        }
      }
    }
}

class BaseFrame1 extends Frame {....
// keine Aenderung gegenueber Applet-Version
```


2 Dialog Boxes

- *Dialog Boxes* sind einfache *Windows,* mit denen üblicherweise Nachrichten, bzw. Warnungen oder Fehlermeldungen ausgegeben werden.
- Sie *“popen”* im Falle einer Meldung auf und erwarten in der Regel eine Reaktion des Benutzers.
- Mögliche Reaktionsalternativen werden z. B. durch verschiede Buttons dargestellt.
- Man hat die Möglichkeit *Dialog Boxes*
  - modal - es werden nur noch Eingaben in die *Dialog Box* ausgewertet

  oder
  - nonmodal - Eingaben in andere Fenster bleibt möglich

  zu kreieren.

2 Dialog Boxes

- Konstruktoren:
  - *Dialog(Frame)* - kreiert eine (nicht sichtbare) *DialogBox* dem *“current” Frame* zugeordnet.
  - *Dialog(Frame, String)* - kreiert eine (nicht sichtbare) *Dialog Box* dem *“current” Frame* zugeordnet mit *String* als Titel.
  - *Dialog(Frame, String, boolean)* - kreiert eine (nicht sichtbare) *Dialog Box* dem “current” Frame zugeordnet mit String als Titel als modale Box, wenn der 3. Parameter den Wert *“true”* hat.

- Mit den Methoden *show()* und *hide()* werden die Boxes sichtbar bzw. nicht sichtbar gemacht.

2 Dialog Boxes

- Ein Beispiel;

```
import java.awt.*;

class BaseFrame2 extends Frame {
  String message = "This is a Window";
  TextDialog dl;
  Label l;

  BaseFrame2(String title) {
    super(title);
    setLayout(new BorderLayout());

    l = new Label(message, Label.CENTER);
    l.setFont(new Font("Helvetica", Font.PLAIN, 12));
    add("Center", l);

// make a dialog for this window
    dl = new TextDialog(this, "Enter Text", true);
    dl.setSize(250,100);
    Button b = new Button("Set Text");
    BaseFrameActions handlebutton = new BaseFrameActions(this);;
    b.addActionListener(handlebutton);
    add("South", b);
  }
}
```

2 Dialog Boxes

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

class TextDialog extends Dialog implements ActionListener {
    TextField tf;
    BaseFrame2 theFrame;

    TextDialog(Frame parent, String title, boolean modal) {
        super(parent, title, modal);
        theFrame = (BaseFrame2)parent;
        setLayout(new BorderLayout(10,10));
        setBackground(Color.white);
        tf = new TextField(theFrame.message,20);
        add("Center", tf);

        Button b = new Button("OK");
        b.addActionListener(this);
        add("South", b);
    }
```


2 Dialog Boxes

```
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() instanceof Button) {
            String label = ((Button)e.getSource()).getLabel();

            if (label == "OK") {
                hide();
                theFrame.l.setText(tf.getText());
            }
        }
    }
}
```

3 Window Events

- *Windows* und *Dialogs* sind beim *Eventhandling* im Verhalten identisch.

Event Name	Listener	Method
Window opened	WindowListener	public void windowOpened(WindowEvent e)
Window moved	ComponentListener	public void componentMoved (ComponentEvent e)
Window activated	WindowListener	public void windowActivated (WindowEvent e)
Window deactivated	WindowListener	public void windowDeactivated (WindowEvent e)
Window iconified	WindowListener	public void windowIconified (WindowEvent e)
Window deiconified	WindowListener	public void windowDeiconified (WindowEvent e)
Window closing	WindowListener	public void windowClosing(WindowEvent e)
Window closed	WindowListener	public void windowClosed(WindowEvent e)

  - W*indow opened:*
    Dieser *Event* wird bei ersten Öffnen eines *Windows* ausgelöst.
  - *Window activated:*
    Dieser *Event* wird ausgelöst, wenn ein *Window* in den Vordergrund oder *Inputmodus* kommt

3 Window Events

- *Windows* und *Dialogs* sind beim *Eventhandling* im Verhalten identisch(cont):
  - *Window deactivated:*
    Dieser *Event* wird ausgelöst, wenn ein anderes *Window* in den Vordergrund oder *Inputmodus* kommt.
  - *Window iconified:*
    Dieser *Event* wird ausgelöst, wenn ein *Window* ein Ikon wird.
  - *Window deiconified:*
    Dieser *Event* wird ausgelöst, wenn aus einem Ikon ein *Window* wird.
  - *Window closing:*
    Dieser *Event* wird ausgelöst, wenn bei einem *Window* der *Close- oder Quit- Button* betätigt wird.
    (Die entsprechenden Maßnahmen muss das Programm einleiten)
  - *Window closed:*
    Dieser *Event* wird ausgelöst, wenn ein *Window* geschlossen worden ist.

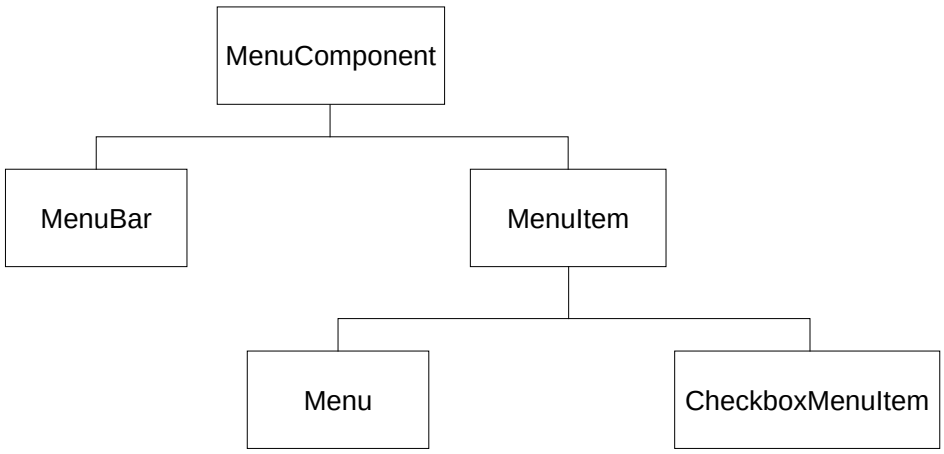
3 Window Events

- Wenn man in einer Applikation einen *window closing event* behandelt, das Fenster schließt und zerstört und damit auch die Applikation beenden will, sollte man dies dem System durch Aufruf der Methode *System.exit(0)* kundtun:

```
public void windowClosing(WindowEvent e) {
    win.hide();
    win.destroy();
    System.exit(0);
}
```


15.6 Menus

- Klassenhierachie:



- In Java werden zwei Menuarten unterschieden
  - *Pop-Up-Menus*, die in der Kopfleiste eines Windows angeordnet sind
  - und *Pop-Up-Menus*, die irgendwo angeordnet werden können.

1 Menu Bars

- Kreiert wird ein Objekt der *MenuBar class* wie üblich:

```
MenuBar mbar = new MenuBar;
```

- Um *MenuBar* einem *Frame* zuzuordnen stellt die Klasse *Frame* die *set.MenuBar() method* zur Verfügung.

```
window1.setMenuBar(mbar);
```

- Einzelne *Menus* werden dem *MenuBar* durch Instantiierung von *Menu-Objects* wie folgt hinzugefügt:

```
Menu myMenu = new Menu(“Edit”);
mbar.add(myMenu);
```


1 Menu Bars

- Mit der *setHelpMenu() method* kann man in einigen Systemen ein spezielles *Help-Menu* kreieren, dass in der Regel einem speziellen Platz zugeordnet wird:

```
Menu helpmenu = new Menu(“Help”);
mbar.add(helpmenu);
mbar.setHelpMenu(helpmenu);
```

- Mit den Methoden *disable() und enable()* kann man den Zugriff auf *Menus* verhindern bzw. wieder ermöglichen, um z. B. Falschbedienungen zu verhindern.

```
mymenu.disable();
...
mymenu.enable();
```

2 Menu Items

- Insgesamt kann man vier Arten von Menu Items Menus hinzufügen:
  - Instanzen der MenuItem class, als “Normale” Menu Items
  - Instanzen der CheckBoxMenuItem class
  - Sub-Menus, mit eigenen Menu Items
  - Separatoren, als Linien zur Strukturierung von Menugruppen.
- Seit Java 1.1 ist auch die Einrichtung von Shortcuts möglich, als Ersatz für das Anklicken von Items.
- Menu Items werden nach üblichen Schema wie folgt angelegt:

```
Menu myMenu = new Menu(“Tools”);
myMenu.add(new MenuItem(“Info”));
myMenu.add(new MenuItem(“Colors”));
```

- Submenus werden wie folgt kreiert:

```
Menu subMenu = new Menu(“Sizes”);
myMenu.add(new subMenu);
subMenu.add(new MenuItem(“Small”));
subMenu.add(new MenuItem(“Large”));
```


2 Menu Items

- Hinzufügen eines CheckBoxMenuItems

```
CheckBoxMenuItem coords =
    CheckBoxMenuItem(“Show Koordinates”);
new Menu subMenu = new Menu(“Sizes”);
myMenu.add(coords);
```

- Hinzufügen eines Separators:

```
MenuItem sepline = new MenuItem(“-”);
myMenu.add(sepline);
```

- Wie das gesamte Menu können auch auf Items die Zugriffe verhindert bzw. wieder ermöglicht werden, um z. B. Falschbedienungen zu verhindern.

```
coords.disable();
...
coords.enable();
```

- Menu Events
  - Actionlistener

3 Popup Menu

- Popup-Menus unterscheiden sich von den MenuBars nur hinsichtlich der zusätzlichen Möglichkeit, sie beliebig zu plazieren und darin, dass eine Zuordnung zu einem Window nicht zwingend ist.

- Kreieren eines Popup-Menus

```
PopupMenu pm = new PopupMenu(“Edit”)
```

- Das Hinzufügen von Items ist identisch mit dem bei normalen Menus.

- Da nicht plattformeinheitlich geregelt ist, mit welcher Taste einer Maus ein Event für ein Popup-Menu generiert wird, empfielt es sich alle möglichen MouseEvents zu berücksichtigen.

3 Popup Menu

- Beispiel:

```
import java.awt.*;

class BaseFrame3 extends Frame {
  String message = "This is a Window";
  TextDialog2 dl;
  Label l;

  BaseFrame3(String title) {
    super(title);
    setLayout(new BorderLayout());

    l = new Label(message, Label.CENTER);
    l.setFont(new Font("Helvetica", Font.PLAIN, 12));
    add("Center", l);

// make a dialog for this window
    dl = new TextDialog2(this, "Enter Text", true);
    dl.setSize(350,150);

    Button b = new Button("Set Text");
    BaseFrameActions2 handleact = new BaseFrameActions2(this);;
    b.addActionListener(handleact);
    add("South", b);
```

3 Popup Menu

```
    MenuBar mb = new MenuBar();
    Menu m = new Menu("Colors");

    MenuItem redmi = new MenuItem("Red");
    redmi.addActionListener(handleact);
    m.add(redmi);
    MenuItem bluemi = new MenuItem("Blue");
    bluemi.addActionListener(handleact);
    m.add(bluemi);
    MenuItem greenmi = new MenuItem("Green");
    greenmi.addActionListener(handleact);
    m.add(greenmi);
    m.add(new MenuItem("-"));
    CheckboxMenuItem boldmi = new CheckboxMenuItem("Bold Text");
    boldmi.addActionListener(handleact);
    m.add(boldmi);
    mb.add(m);
    setMenuBar(mb);
  }

  public Insets getInsets() {
    return new Insets(20,0,25,0);
  }
}
```


3 Popup Menu

```
/* actions for frame window */

import java.awt.*;
import java.awt.event.*;

public class BaseFrameActions2 implements ActionListener {

  BaseFrame3 theApp;

  BaseFrameActions2(BaseFrame3 win) {
    theApp = win;
  }
```

3 Popup Menu

```
public void actionPerformed(ActionEvent e) {

    if (e.getSource() instanceof Button)
      theApp.dl.show();
    else if (e.getSource() instanceof MenuItem) {
      String label = ((MenuItem)e.getSource()).getLabel();
      if (label.equals("Red"))
        theApp.l.setBackground(Color.red);
      else if (label.equals("Blue"))
        theApp.l.setBackground(Color.blue);
      else if (label.equals("Green"))
        theApp.l.setBackground(Color.green);
      else if (label.equals("Bold Text")) {
        if (theApp.l.getFont().isPlain()) {
          theApp.l.setFont(new Font("Helvetica", Font.BOLD, 12));
        }
        else theApp.l.setFont(new Font("Helvetica", Font.PLAIN, 12));
      }
      theApp.repaint();
    }
  }
}
```


3 Popup Menu

```
import java.awt.*;

public class PopupWindowMenuTest extends Frame {
  Frame window;
  Button open, close;

  PopupWindowMenuTest(String titel) {
    super(titel);
    setLayout(new FlowLayout());
    PopupActions3 handlebutton = new PopupActions3(this);
    open = new Button("Open Window");
    open.addActionListener(handlebutton);
    add(open);
    close = new Button("Close Window");
    close.addActionListener(handlebutton);
    add(close);
    window = new BaseFrame3("A Popup Window");
    window.setSize(350,250);
    window.show();
  }
  public static void main (String args[]) {
   PopupWindowMenuTest wapp = new PopupWindowMenuTest ("Application-Frame");
    wapp.setSize(600,300);
    wapp.show();
  }
}
```