10

Grafische Oberflächen

1. Aufbau Grafischer Oberflächen
2. Komponenten
3. Ereignisverarbeitung

10.1 Aufbau grafischer Oberflächen

- Ziel: Programme die nicht nur im Konsolenfenster laufen, sondern mit grafischer Benutzungsschnittstelle (Graphical User Interface, GUI) ausgestattet sind
- Varianten:
  - ***Applikationen*** (eigenständige Anwendungsprogramme mit grafischen Benutzeroberflächen) → in dieser Vorlesung
  - ***Applets*** (Programme, die in Webseiten integriert werden und innerhalb eines Browsers in einer abgeschlossenen Umgebung ausgeführt werden) → heutzutage selten geworden
  - ***Java FX*** (eigenständiges Framework, Anwendungen können auch im Browser ausgeführt werden) → soll(te) Swing ersetzen
- Grundprinzip beim Aufbau der grafischen Oberfläche für Varianten 1+2 gleich:
  - Grafische Oberflächen werden nach dem Baukastenprinzip aus ***Komponenten*** zusammengesetzt.
  - Spezielle Komponenten (so genannte ***Container***) können selbst wieder Komponenten enthalten.
  - Ausgangspunkt für den Aufbau der Komponenten ist ein ***Basis-Container.***
  - Komponenten können mit einer ***Ereignisverarbeitung*** ausgestattet werden, um auf Aktionen des Benutzers bzw. der Benutzerin zu reagieren.

Aufbau grafischer Oberflächen

- In der Java-Klassenbibliothek finden sich mit den ***Java Foundation Classes*** plattformunabhängige Bausteine, um portable Programme mit grafischen Benutzungsoberflächen zu entwickeln:
  - ***AWT*** (Abstract Window Toolkit) Paket **`java.awt`**
  - ***Swing*** Paket **`javax.swing`**
- Grobe Einteilung der verschiedenen Klassen in vier Gruppen:
  - ***Grundkomponenten*****:** einfache Oberflächen-Elemente wie zum Beispiel Beschriftungen (Labels), Knöpfe (Buttons), Auswahlfelder oder Klapptafeln
  - ***Container*****:** spezielle Komponenten, die selbst wieder Komponenten enthalten können
  - ***Layout-Manager***, ***Farben*** und ***Fonts*****:** Klassen für die Anordnung und die Gestaltung der einzelnen Komponenten
  - ***Ereignisse*** und ***Listener*****:** Klassen für die Erzeugung und Verarbeitung von Ereignissen, also für die Interaktion der Komponenten mit den Anwendern

10.1.2 Einige Grundlagen anhand einfacher Beispiele

- Mit Swing:

```
import javax.swing.*;
public class FrameOhneInhaltSwing {
  public static void main(String[] args) {
    JFrame fenster = new JFrame();
    fenster.setTitle("Mein erstes Swing-Fenster");
    fenster.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fenster.setSize(300,150);
    fenster.setVisible(true);
  }
}
```

Swing-Klassen importieren

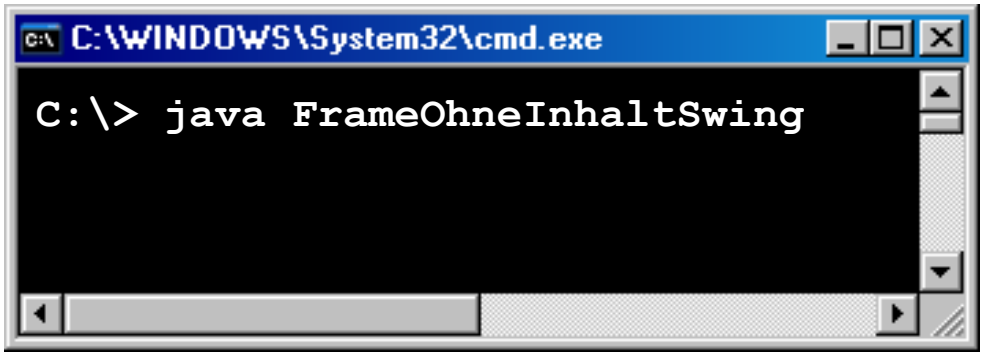
JFrame als Basis-Container

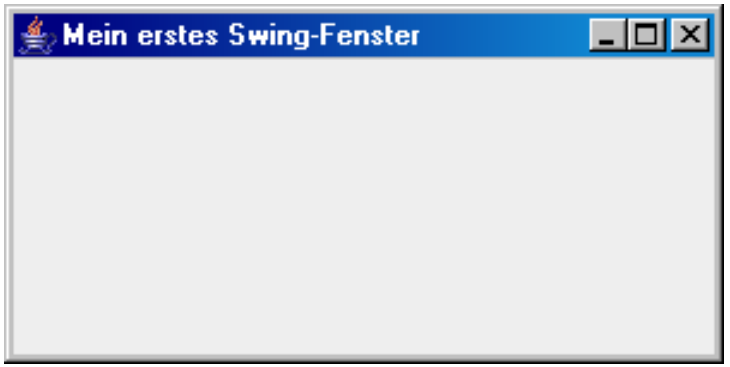
Fenster-Beschriftung setzen

Standard-Close-Operation festlegen

Fenster-Größe in Pixel festlegen

Fenster sichtbar machen





Einige Grundlagen anhand einfacher Beispiele

- Mit Swing und als eigene Frame-Klasse, die von **`JFrame`** erbt:

```
import javax.swing.*;
public class FrameOhneInhalt extends JFrame {
  public FrameOhneInhalt () { // Konstruktor
    // Hier werden spaeter die Komponenten hinzugefuegt
  }
}
```

- Frame-Erzeugung in der **`main`**-Methode der gleichen oder einer anderen Klasse:

```
  public static void main(String[] args) {
    FrameOhneInhalt fenster = new FrameOhneInhalt();
    fenster.setTitle("Frame ohne Inhalt");
    fenster.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fenster.setSize(300,150);
    fenster.setVisible(true);
}
```

Einige Grundlagen anhand einfacher Beispiele

- Erweiterung: Frame, der auch eine Komponente enthält:

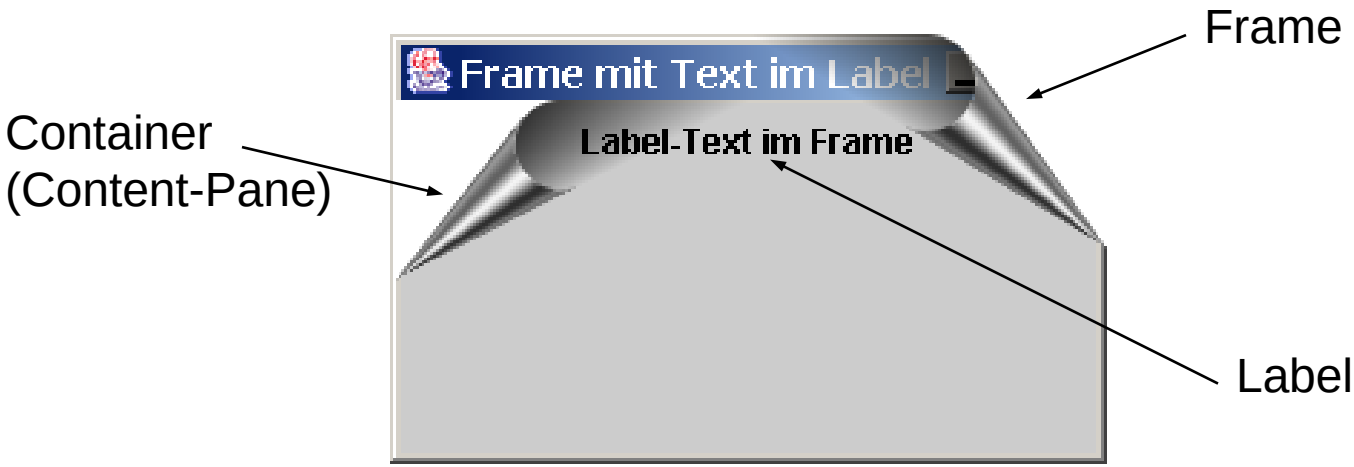
```
import java.awt.*;
import javax.swing.*;
public class FrameMitText extends JFrame {
  Container c;            // Container dieses Frames
  JLabel beschriftung;  // Label, das im Frame erscheinen soll
  public FrameMitText() { // Konstruktor
    c = getContentPane();   // Bestimme Referenz auf Container
    c.setLayout(new FlowLayout());          // Setze das Layout
    beschriftung = new JLabel("Label-Text im Frame");
    c.add(beschriftung);      // Erzeuge das Label und fuege es
                              // dem Container hinzu
  }
}
```

Label als Komponente eingefügt

FlowLayout: Zeile für Zeile wird von links nach rechts mit Komponenten aufgefüllt, die zeilenweise zentriert werden

Einige Grundlagen anhand einfacher Beispiele

- Die Komponenten werden nicht direkt in den Frame sondern in dessen ***Container*** bzw. ***Content-Pane*** eingefügt.



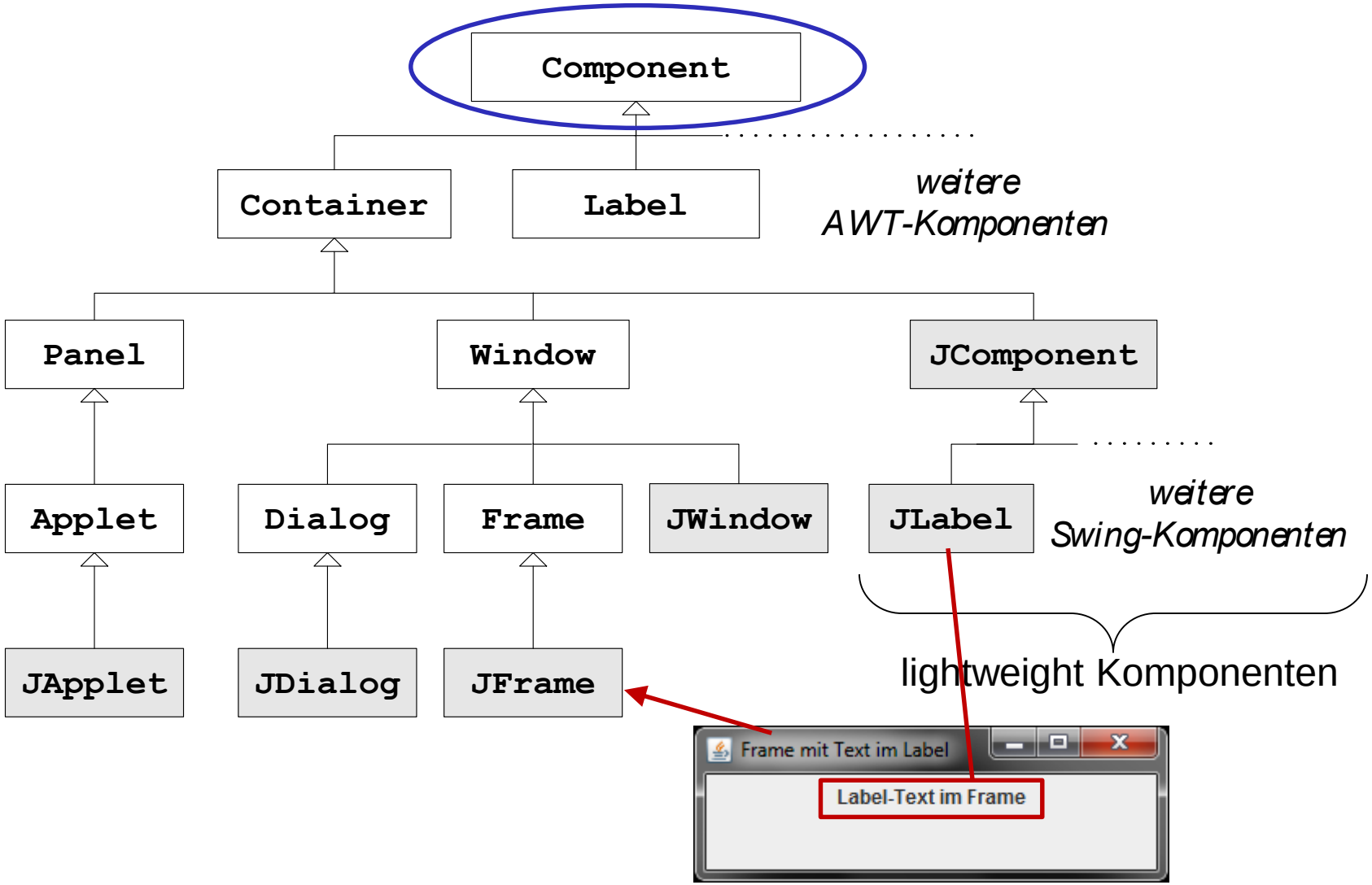
- Für den Zugriff auf die Content-Pane besitzt jedes **`JFrame`**-Objekt eine Instanzmethode **`getContentPane`**, die eine Referenz vom Typ **`Container`** zurückliefert.
- Für das Einfügen wird die Instanzmethode **`add`** des Containers verwendet.
- **Achtung**: Seit Java 5 ist der Aufruf **`add(…)`** auch direkt für das **`JFrame`**-Objekt möglich. Er wird automatisch umgesetzt zu **`getContentPane().add(…)`**.
- Analoges gilt für das Setzen des Layouts!

10.1.3 Die AWT- und Swing-Klassenbibliothek

- Abstract Window Toolkit
  - Auch Programme mit grafischer Oberfläche sollten portierbar bleiben.
  - Alle Fenster- und Dialog-Elemente werden daher vom darunter liegenden Betriebssystem zur Verfügung gestellt. (***Peer***-Ansatz: Alle Komponenten reichen die auszuführenden Aktionen an plattformspezifische GUI-Objekte, die Peers, weiter).
  - Komponenten, die Peer-Objekte benötigen (***heavyweight*** Komponenten), sehen auf unterschiedlichen Plattformen unterschiedlich aus.
  - GUI-Funktionalitäten sind dadurch auf die unterstützten Funktionalitäten beschränkt.
- Swing
  - Nur noch wenige Komponenten benutzen plattformspezifische GUI-Objekte.
  - Fast alle Komponenten sind in Java geschrieben und daher ***lightweight***.
  - Oberfläche kann plattformunabhängig vollständig selbst gestaltet werden und noch zur Laufzeit im ***Look and feel*** verändert werden.
  - Wesentlich mehr Möglichkeiten zur Oberflächengestaltung stehen zur Verfügung.
  - Swing ist eine Erweiterung von AWT, kein Ersatz!

Die AWT- und Swing-Klassenbibliothek

- Hierarchie der AWT- und Swing-Klassen (**Komponenten**)

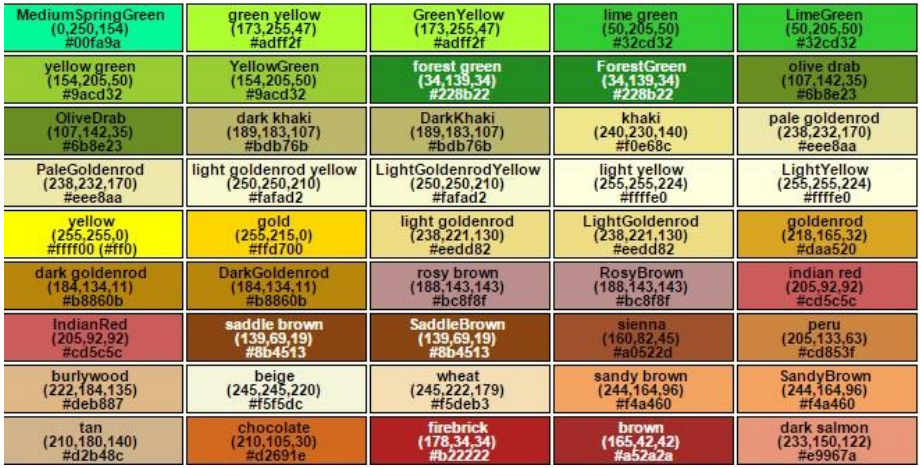


10.1.4 Grundlegende Methoden von Komponenten

- Einige *öffentliche* Instanzmethoden der *abstrakten* Klasse **`Component`**:
  - **`Color getBackground()`** liefert die Hintergrundfarbe der Komponente
  - **`Color getForeground()`** liefert die Vordergrundfarbe der Komponente
  - **`Font getFont()`** liefert die Schriftart der Komponente
  - **`int getHeight()`** liefert die Höhe der Komponente (in Pixel)
  - **`int getWidth()`** liefert die Breite der Komponente (in Pixel)
  - **`void setBackground(Color c)`** setzt die Hintergrundfarbe der Komp.
  - **`void setForeground(Color c)`** setzt die Vordergrundfarbe der Komp.
  - **`void setFont(Font f)`** setzt die Schriftart der Komponente
  - **`void setSize(int x, int y)`** setzt die Größe der Komp. (in Pixel)
- Die Klasse **`Color`** ermöglicht die Festlegung eigener Farben über RGB-Werte und stellt Klassen-Konstanten mit vordefinierten Farbwerten zur Verfügung
  - z. B. **`new Color(100,30,45)`** oder **`Color.RED`** bzw. **`Color.red`**
- Die Klasse **`Font`** ermöglich die Festlegung von Schriftarten
  - z. B. **`new Font("SansSerif", Font.BOLD+Font.ITALIC, 12)`**

10.1.4.1 Die Klasse java.awt.Color

- Klasse zur Repräsentation von Farben
- Verschiedene Farbmodelle, vor allem **R**(ot), **G**(rün), **B**(lau): additives Modell, nach dem jede Farbe aus einer Mischung dreier Grundfarben definiert wird. Die Werte für R, G und B liegen zwischen 0 und 255 (http://de.wikipedia.org/wiki/RGB-Farbraum)
  - **`public Color(int r, int g, int b)`** (Konstruktor)
  - vordefinierte Farben: **`Color.BLACK`**, **`Color.WHITE`**, **`Color.BLUE`**, **`Color.YELLOW`**, …
- X Colors z.B. auf den Seiten: http://en.wikipedia.org/wiki/X11_color_names



10.1.4.2 Die Klasse java.awt.Font

- Repräsentation einer Schriftart zur Darstellung eines Zeichensatzes
- Drei wesentliche Bestandteile: Name des Fonts (Arial, Courier,…), Stil (PLAIN, BOLD,ITALIC,….) und Größe
- verfügbare Fonts systemabhängig

```
import java.awt.GraphicsEnvironment

GraphicsEnvironment ge =
                GraphicsEnvironment.getLocalGraphicsEnvironment();

Font[] allFonts = ge.getAllFonts(); // alle verfügbaren Fonts
```

- Beispiel zum Erzeugen eines Fonts:

```
Font font = new Font("Jokerman", Font.PLAIN, 35);

Font font_2 = font.deriveFont(10.0f); // andere Größe

Font font_3 = font.deriveFont(Font.BOLD); // anderer Style

Font font_4 = new Font(null, Font.PLAIN, 18); // Default Font
                                              // Family
```

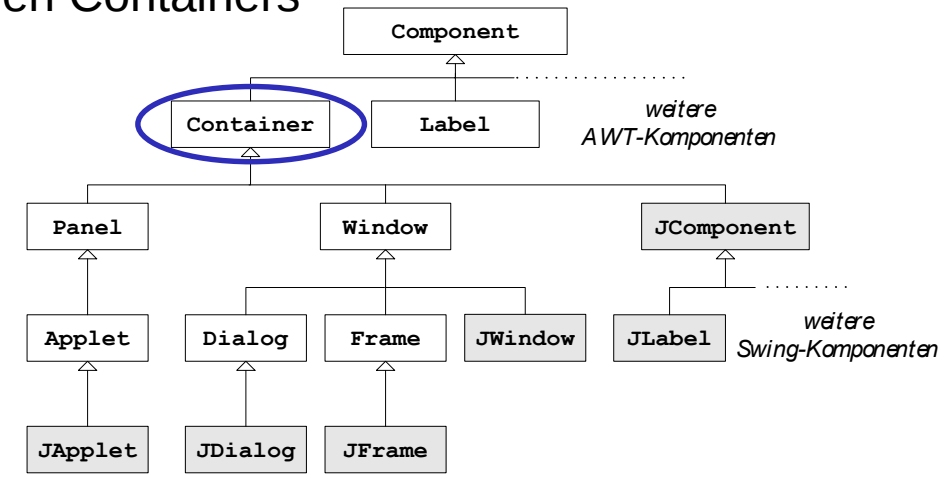
Grundlegende Methoden von Komponenten

- Beispiel

```
import java.awt.*;
import javax.swing.*;
public class FrameMitColorUndFont extends JFrame {
  Container c;
  JLabel text1, text2;
  Color hell = new Color(250,250,250);
  Font mono = new Font("Monospaced",Font.BOLD+Font.ITALIC,30);
  public FrameMitColorUndFont() {
    c = getContentPane();
    c.setLayout(new FlowLayout());
    c.setBackground(Color.BLACK);
    text1 = new JLabel("weiße Schrift");
    text1.setForeground(hell);
    text2 = new JLabel("Monospaced Text");
    text2.setForeground(hell);
    text2.setFont(mono);
    c.add(text1);
    c.add(text2);
  }
}
```

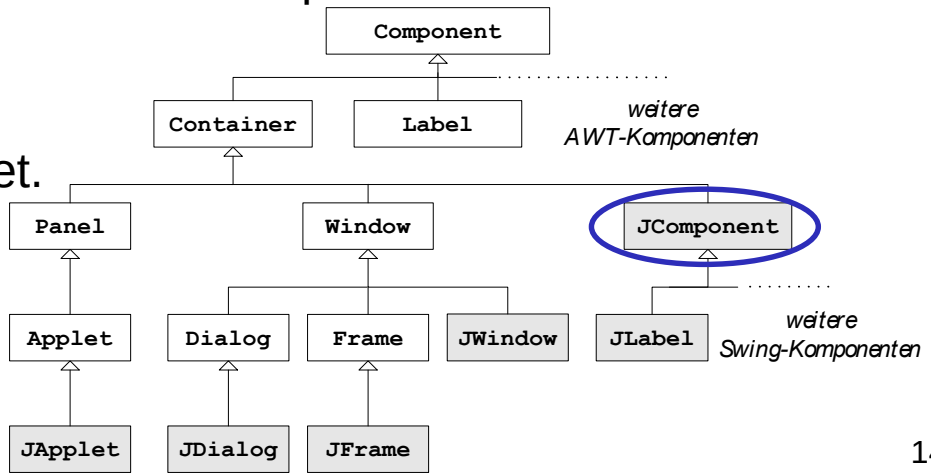
10.1.5 Grundlegende Methoden von Containern

- Einige *öffentliche* Instanzmethoden der Klasse **`Container`**:
  - **`Component add(Component comp)`**
    fügt die Komponente **`comp`** dem aufrufenden Container hinzu
  - **`void add(Component comp, Object constraints)`**
    fügt unter Beachtung der in **`constraints`** angegebenen Layout-Bedingung die Komponente **`comp`** dem aufrufenden Container hinzu
  - **`void remove(Component comp)`**
    entfernt die Komponente **`comp`** aus dem aufrufenden Container
  - **`void setLayout(LayoutManager mgr)`**
    setzt das Layout des aufrufenden Containers



10.1.6 Grundlegende Methoden von Swing-Komponenten

- Einige *öffentliche* Instanzmethoden der *abstrakten* Klasse **`JComponent`**:
  - **`boolean isOpaque()`**
    liefert **`true`**, wenn die aufrufende Komponente einen undurchsichtigen Hintergrund besitzt, andernfalls **`false`**
  - **`public void setOpaque(boolean b)`**
    schaltet den Hintergrund der aufrufenden Komponente undurchsichtig (wenn **`b`** den Wert **`true`** hat) bzw. durchsichtig (wenn **`b`** den Wert **`false`** hat)
  - **`public String getToolTipText()`**
    liefert den aktuellen Tooltip-Text der aufrufenden Komponente
  - **`public void setToolTipText(String text)`**
    legt **`text`** als Tooltip-Text für die aufrufende Komponente fest
- **Hinweis**: Einige Swing-Komponenten (z. B. **`JLabel`**) sind standardmäßig mit durchsichtigem Hintergrund ausgestattet. Veränderungen der Hintergrundfarbe haben daher erst Auswirkungen, wenn das Label den Status ***opak*** bzw. undurchsichtig erhält.



Grundlegende Methoden von Swing-Komponenten

- Beispiel

```
import java.awt.*;
import javax.swing.*;
public class FrameMitToolTipUndOpak extends JFrame {
  Container c;
  JLabel l1, l2, l3;
  public FrameMitToolTipUndOpak() {
    c = getContentPane();
    c.setLayout(new FlowLayout());
    l1 = new JLabel("  Label-Text  ");
    c.add(l1);
    l1.setToolTipText("Des isch nur en Tescht!");
    l2 = new JLabel("  Nicht opak  ");
    l2.setBackground(Color.YELLOW);
    c.add(l2);
    l3 = new JLabel("  Opak  ");
    l3.setOpaque(true);
    l3.setBackground(Color.YELLOW);
    c.add(l3);
  }
}
```

Grundlegende Methoden von Swing-Komponenten

- Größen von Komponenten (**`JComponent`**) setzen
  - **`public void setPreferredSize(Dimension preferredSize)`** setzt die bevorzugte Größe einer Komponente
  - **`public void setMaximumSize(Dimension maximumSize)`** setzt die maximale Größe einer Komponente
  - **`public void setMinimumSize(Dimension minimumSize)`** setzt die minimale Größe einer Komponente

- Klasse **`java.awt.Dimension`** für die Größendefinition mit Konstruktor
  - **`Dimension(int width, int height)`**

10.1.7 Layout von Komponenten

- **Layout-Manager**: Kontrolle der Anordnung von Komponenten in einem Container
  - voreingestellte Manager können durch setLayout-Aufruf ersetzt werden
  - **FlowLayout**: Anordnung in einer einzigen Zeile, Umbruch hängt von Breite des Containers ab. (Default-Layout bei den meisten Komponenten)
  - **BorderLayout**: verwendet vier Regionen, denen Komponenten zugeordnet werden können (NORTH, SOUTH, EAST, WEST, CENTER)
  - **BoxLayout**: Anordnung der Komponenten in einer Zeile bzw. Spalte
  - **GridLayout**: verteilt gleich große Komponenten auf Felder eines $n \times m$ Gitters (n Zeilen mit m Spalten)
  - **GridBagLayout**: flexible Anordnung der Komponenten in einem Gitter
  - **CardLayout**: unterstützt verschiedene, austauschbare Anordnungen für mehrere Komponenten, die sich eine Ausgabeeinheit teilen
  - GroupLayout und SpringLayout : hauptsächlich von GUI Buildern verwendet

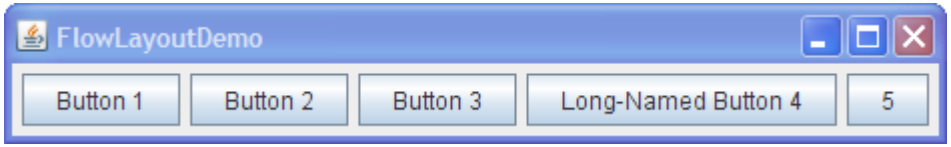
- Genaueres zur Verwendung der Layouts:

  http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

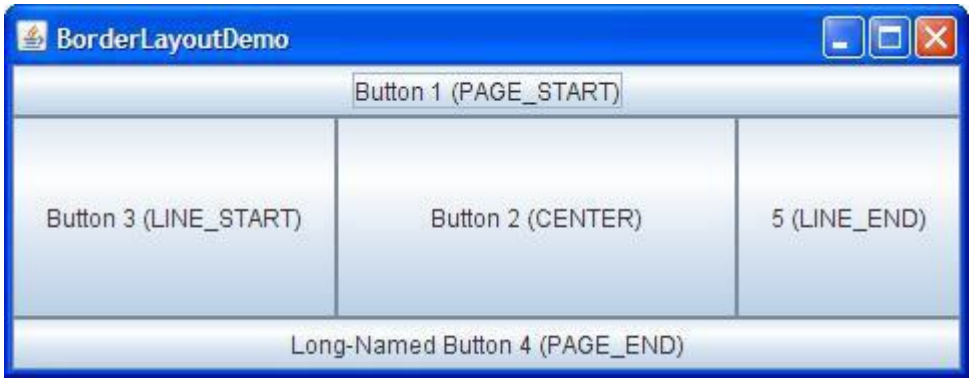
- IDEs wie z.B. NetBeans bieten Werkzeuge zur graphischen Gestaltung einer Oberfläche an. Für Eclipse gibt es diverse Plugins.

Swing Layouts

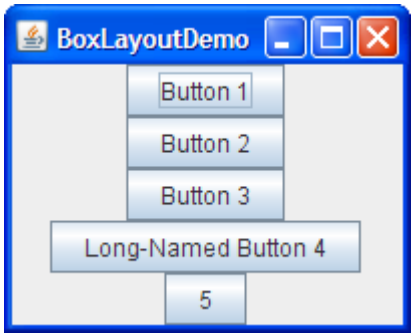
FlowLayout



BorderLayout

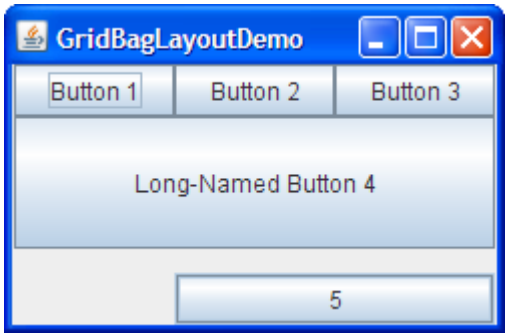


BoxLayout

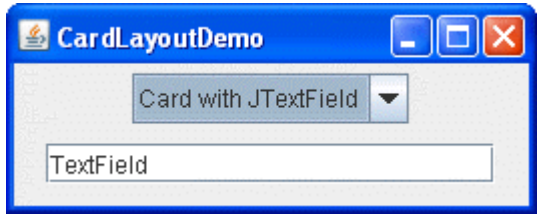


GridLayout

GridBagLayout



CardLayout



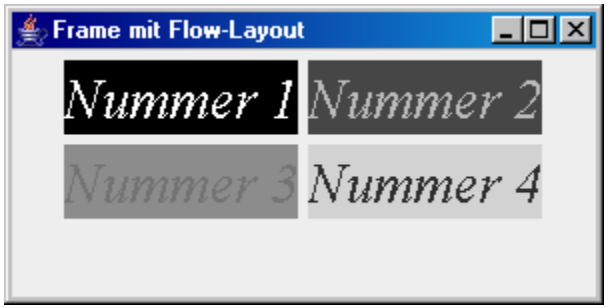
Layout-Manager

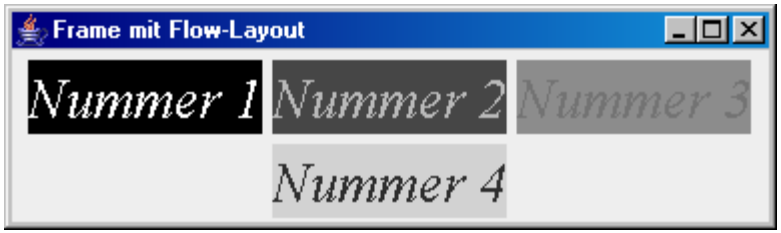
- **Flow-Layout**
  - einfachster Layout-Manager, voreingestellt für Applets
  - Anordnung von links nach rechts (Zeile für Zeile)
  - innerhalb einer Zeile standardmäßig zentriert
  - links- oder rechtsbündig kann aber auch ausgewählt werden
  - Konstruktoren
    - **`FlowLayout()`** zentrierte Anordnung, Abstand 5
    - **`FlowLayout(int align)`** Anordnung gemäß **`align`**, Abstand 5
      - Möglichkeiten für **`align`**: **`FlowLayout.CENTER`**, **`FlowLayout.LEFT`**, **`FlowLayout.RIGHT`**
    - **`FlowLayout (int align, int h, int v)`**
      Anordnung gemäß **`align`**,
      horizontaler Abstand **`h`**,
      vertikaler Abstand **`v`**

Layout-Manager

- Beispiel zum Flow-Layout

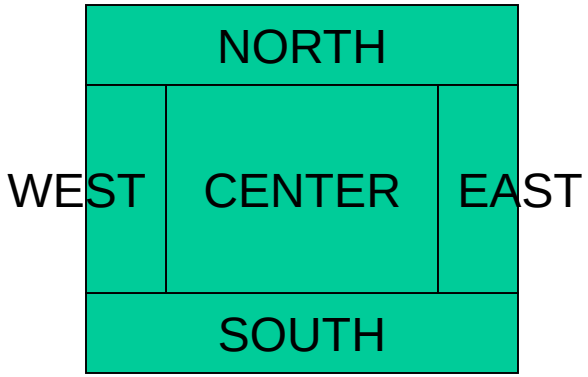
```
import java.awt.*;
import javax.swing.*;
public class FrameMitFlowLayout extends JFrame {
  Container c;
  JLabel l[] = new JLabel[4];   // Feld fuer Labels
  Font schrift = new Font("Serif",Font.ITALIC,28);
  public FrameMitFlowLayout() {
    c = getContentPane();
    c.setLayout(new FlowLayout());
    for (int i = 0; i < 4; i++) {
      int rgbFg = 255 - i*70;
      int rgbBg = i*70;
      l[i] = new JLabel("Nummer " + (i+1));
      l[i].setForeground(new Color(rgbFg,rgbFg,rgbFg));
      l[i].setBackground(new Color(rgbBg,rgbBg,rgbBg));
      l[i].setFont(schrift);
      l[i].setOpaque(true);
      c.add(l[i]);
    }
  }
}
```





Layout-Manager

- **BorderLayout**
  - Anordnung in fünf verschiedenen Gebieten



  - **`add`**-Aufruf mit zusätzlichem Parameter für Angabe des Gebiets
    - Möglichkeiten für den Parameter:

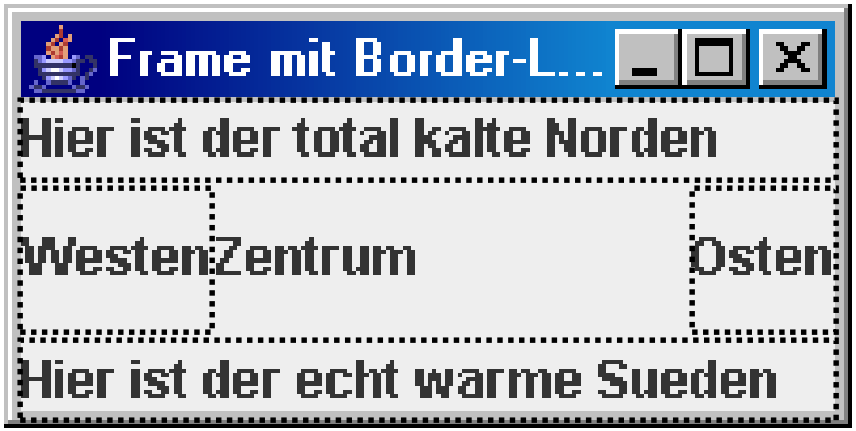
      `BorderLayout.NORTH`
      `BorderLayout.SOUTH`
      `BorderLayout.WEST`
      `BorderLayout.EAST`

      implizit vordefiniert: **`BorderLayout.CENTER`**
  - Konstruktoren
    - **`BorderLayout()`** — Abstand 5
    - **`BorderLayout (int h, int v)`** — horizontaler Abstand **`h`**, vertikaler Abstand **`v`**

Layout-Manager

- Beispiel zum Border-Layout:

```
import java.awt.*;
import javax.swing.*;
public class FrameMitBorderLayout extends JFrame {
  Container c;
  JLabel l[] = new JLabel[5];
  public FrameMitBorderLayout() {
    c = getContentPane();
    c.setLayout(new BorderLayout());
    l[0] = new JLabel("Hier ist der total kalte Norden");
    l[1] = new JLabel("Hier ist der echt warme Sueden");
    l[2] = new JLabel("Osten");
    l[3] = new JLabel("Westen");
    l[4] = new JLabel("Zentrum");
    c.add(l[0],BorderLayout.NORTH);
    c.add(l[1],BorderLayout.SOUTH);
    c.add(l[2],BorderLayout.EAST);
    c.add(l[3],BorderLayout.WEST);
    c.add(l[4],BorderLayout.CENTER);
  }
}
```



Layout-Manager

- **GridLayout**
  - Anordnung in einer Matrix (zeilenweise aufgefüllt)
  - Konstruktoren
    - `GridLayout()`

      genau eine Zeile voreingestellt
    - `GridLayout (int rows, int cols)`

      `rows` Zeilen
      `cols` Spalten
    - `GridLayout (int rows, int cols, int h, int v)`

      `rows` Zeilen
      `cols` Spalten
      horizontaler Abstand `h`,
      vertikaler Abstand `v`
    - `rows` = `cols` = `0` ist nicht erlaubt, nur ein Wert kann 0 sein

      0 steht für "beliebig viele"

Layout-Manager

- Beispiel zum Grid-Layout:

```
import java.awt.*;
import javax.swing.*;
public class FrameMitGridLayout extends JFrame {
  Container c;
  JLabel l[] = new JLabel[6];
  public FrameMitGridLayout() {
    c = getContentPane();
    c.setLayout(new GridLayout(2,3,10,40));
    for (int i = 0; i < 6; i++) {
      int rgbFg = 255 - i*50, rgbBg = i*50;
      l[i] = new JLabel("Nummer " + (i+1));
      l[i].setForeground(new Color(rgbFg,rgbFg,rgbFg));
      l[i].setBackground(new Color(rgbBg,rgbBg,rgbBg));
      l[i].setFont(new Font("Serif",Font.ITALIC,10 + i*3));
      l[i].setOpaque(true);
      c.add(l[i]);
    }
  }
}
```

Layout Manager

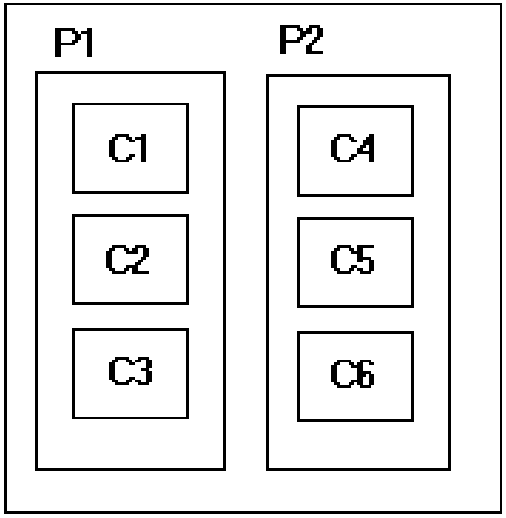
- **BoxLayout**
  - Anordnung horizontal (von links nach rechts) oder vertikal (von oben nach unten) in einer Box unter Berücksichtigung der bevorzugten Breite bzw. Höhe (Einstellung mit setPreferredSize)

```
import java.awt.*;
import javax.swing.*;
public class FrameMitBoxLayout extends JFrame {
   Container c;
   JLabel l[] = new JLabel[5];
   public FrameMitBoxLayout() {
        c = getContentPane();
        c.setLayout(new BoxLayout(c,BoxLayout.X_AXIS));
        l[0] = new JLabel(" Eins ");
        l[1] = new JLabel(" Zwei ");
        l[2] = new JLabel(" Drei ");
        l[3] = new JLabel(" Vier ");
        l[4] = new JLabel(" Fünf ");
        for (int i=0;i<5;i++){
            l[i].setFont(new Font("Arial",Font.BOLD,28));
            c.add(l[i]);
        }
   }…
}
```

BoxLayout.*Y_AXIS*

Layout durch Schachtelung von Containern mit BoxLayout

- Konstruktion von **`Box`** Instanzen, die in weiteren **`Box`** Instanzen eingebettet werden.



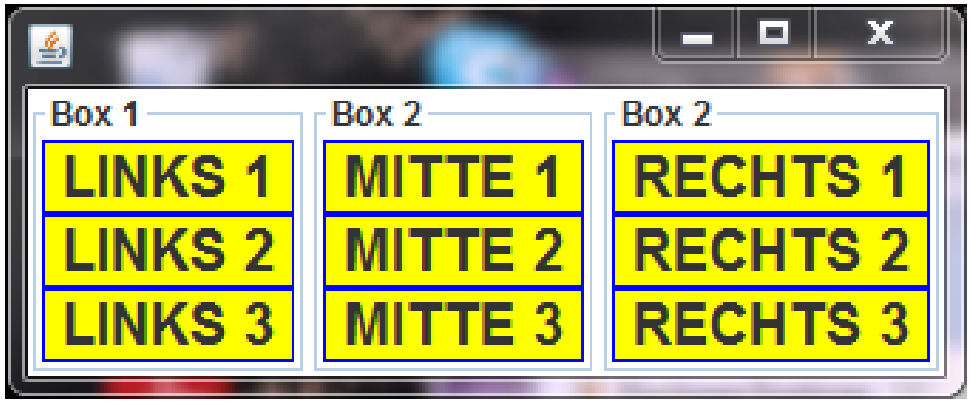
Beispiel: Schachtelung mit BoxLayouts

```
Import java.awt.*;
import javax.swing.*;
public class BoxTest extends JFrame
     Container c;
     JLabel  l[] = {new JLabel(" LINKS  1 "),new JLabel(" LINKS 2"),
                    new JLabel(" LINKS  3 "),new JLabel(" MITTE 1 "),
                    new JLabel(" MITTE  2 "),new JLabel(" MITTE 3 "),
                    new JLabel(" RECHTS 1 "), new JLabel(" RECHTS 2 "),
                    new JLabel(" RECHTS 3 ")};
      public BoxTest() {
           c = getContentPane();
           c.setLayout(new BoxLayout(c,BoxLayout.X_AXIS));
           c.setBackground(Color.white);
           Box b1 = Box.createVerticalBox();
           b1.add(l[0]);b1.add(l[1]);b1.add(l[2]);
           b1.setBorder(BorderFactory.createTitledBorder("Box 1"));
           Box b2 = Box.createVerticalBox();
           b2.add(l[3]); b2.add(l[4]);b2.add(l[5]);
           b2.setBorder(BorderFactory.createTitledBorder("Box 2"));
           Box b3 = Box.createVerticalBox();
           b3.add(l[6]);b3.add(l[7]);b3.add(l[8]);
           b3.setBorder(BorderFactory.createTitledBorder("Box 2"));

           c.add(b1);c.add(b2);c.add(b3);
```

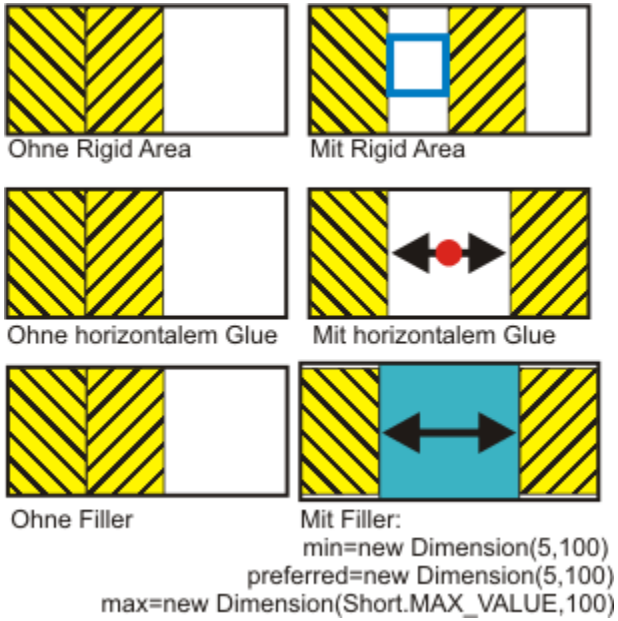
Beispiel: Schachtelung mit BoxLayouts

```
    for (int i=0;i<9;i++){
        l[i].setBackground(Color.yellow);
        l[i].setOpaque(true);
        l[i].setFont(new Font("Arial",Font.BOLD,20));
        l[i].setBorder(BorderFactory.createLineBorder(Color.BLUE));
    }
}
```



Anpassung des Layouts mit Klasse javax.swing.Box

- Klasse Box liefert Komponente mit BoxLayout
  - **`Box createVerticalBox()`** und **`Box createHorizontalBox()`**
- Klasse Box liefert unsichtbare Komponenten für Container-Instanzen, die das Layout beeinflussen
  - **`Box createRigidArea(Dimension d)`**
  - **`Component createVerticalStrut(int height)`**
  - **`Component createVerticalGlue()`**



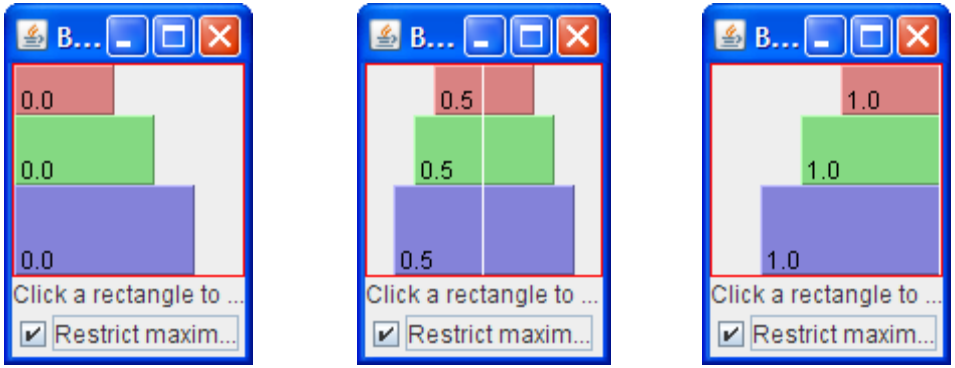
Beispiel:

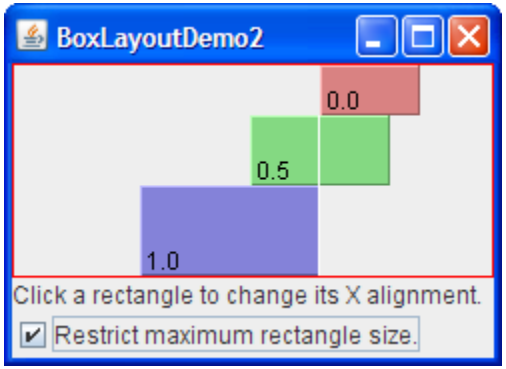
```
import java.awt.*;
import javax.swing.*;
public class FrameMitBoxLayout2 extends JFrame {
        Container c;
         JLabel l[] = new JLabel[3];
         public FrameMitBoxLayout2() {
                  c = getContentPane();
                  c.setLayout(new BoxLayout(c,BoxLayout.X_AXIS));

                  l[0] = new JLabel(" Eins ");
                  l[1] = new JLabel(" Zwei ");
                  l[2] = new JLabel(" Drei ");
                  for (int i=0;i<3;i++){
                        l[i].setFont(new Font("Arial",Font.BOLD,20));
                        l[i].setBorder(
                              BorderFactory.createLineBorder(Color.blue));
                  }
                  c.add(l[0]);
                  c.add(Box.createRigidArea(new Dimension(25,0)));
                  c.add(l[1]);
                  c.add(Box.createHorizontalGlue());
                  c.add(l[2]);
         }
}
```

Anpassung des Layouts mit Alignments

- Für **`BoxLayout`**: https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html
- **`setAlignmentX(float f)`** und **`setAlignmentY(float f)`**





Manuelles Layout ohne LayoutManager

- Layout Manager einer Komponente c kann auch auf **`null`** gesetzt werden (**`c.setLayout(null)`**).
- Für alle Kinder der Container Komponente muss **`setBounds`** Methode aufgerufen werden
- **`repaint`** Methode der Komponente aufrufen
- Probleme beim Verändern der Größe

```
Container c = getContentPane();
Insets insets = c.getInsets();
JButton b1 = new JButton("1");
Dimension dim = b1.getPreferredSize();
b1.setBounds(insets.left+100,insets.top+100,
                dim.width,dim.height);
```