Layout-Manager

Bei grafischen Oberflächen ist die Platzierung der Komponenten eine wichtige Aufgabe. Layout-Manager bieten die Möglichkeit Komponenten automatisch von Java platzieren zu lassen. Dabei wird die Platzierung auch bei einer Änderung der Fenstergröße an die Komponenten weitergegeben und entsprechend angepasst.

Ein solcher Layout-Manager wurde schon im ersten GUI-Beispiel verwendet:

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

public class MeinPanel extends JPanel {
    public MeinPanel() {
        BorderLayout basis = new BorderLayout();
        this.setLayout(basis);

        JLabel yellowLabel = new JLabel();
        yellowLabel.setOpaque(true);
        yellowLabel.setBackground(new Color(248, 213, 131));
        yellowLabel.setPreferredSize(new Dimension(200, 180));

        this.add(yellowLabel);
    }
}
```

Ein Layout-Manager-Objekt der Klasse BorderLayout erstellen.

Das Layout-Objekt wird mit der Methode setLayout als geltendes Layout festgelegt.

Hinzufügen des Labels zum Panel. Das Label wird so platziert, wie es der Layout-Manager (BorderLayout) standardmäßig vorsieht.

Methode add

Ausgangspunkt für die Anzeige ist das Hinzufügen einer GUI-Komponente mit einer add-Methode. Bei dieser Methode gibt es auch überladene Varianten. Diesen kann man neben der GUI-Komponente als ersten Parameter auch einen Index und/oder ein Objekt für zusätzliche Bedingungen übergeben. Es kann aber sein, dass diese je nach Layout-Manager anders interpretiert oder sogar ganz ignoriert werden.

`Component`	`add(Component comp)` Appends the specified component to the end of this container.
`Component`	`add(Component comp, int index)` Adds the specified component to this container at the given position.
`void`	`add(Component comp, Object constraints)` Adds the specified component to the end of this container.
`void`	`add(Component comp, Object constraints, int index)` Adds the specified component to this container with the specified constraints at the specified index.

Die beiden ersten add-Methoden geben die übergebene GUI-Komponente wieder zurück, damit sie gleich weiterverarbeitet werden kann. In den meisten Fällen kann – wie im obigen Beispiel - diese Rückgabe ignoriert werden und die Methode ohne Auffangvariable verwendet werden.

Setzen des Layout-Managers

Wenn die add-Methode aufgerufen wurde, entscheidet der festgelegte Layout-Manager, wo die Komponente platziert wird. Dieser kann einem Container (z.B. einem JFrame-Objekt oder einem JPanel-Objekt) mit Hilfe der Methode setLayout zugeordnet werden.

void	**setLayout**(**LayoutManager** mgr) Sets the layout manager for this container.

Als Parameter verlangt diese Methode ein Layout-Manager-Objekt, das aus einer konkreten Layout-Manager-Klasse erzeugt wird. Es gibt viele Layout-Manager sowohl im Package `java.awt` als auch im Package `javax.swing`, die bei swing-GUIs eingesetzt werden können.

Layout-Manager im Package `java.awt`:

- `BorderLayout` (das Standard-Layout für `JFrame`): unterteilt ein Fenster in vier Randbereiche (oben: `PAGE_START`, unten: `PAGE_END`, links: `LINE_START`, rechts: `LINE_END`) und einen zentralen Bereich (`CENTER`)
- `CardLayout`: zeigt immer nur ein Objekt im Container an (ähnlich wie nur das oberste Objekt in einem Kartenstapel sichtbar ist)
- `FlowLayout` (das Standard-Layout für ein `JPanel`): ordnet die Komponenten fließend in einer Richtung an, ähnlich wie die Worte innerhalb eines Absatzes (inkl. Wechsel in eine neue Zeile bei Bedarf)
- `GridBagLayout`: ermöglicht eine flexible Anordnung der GUI-Komponenten anhand der angegeben `GridBagConstraints`
- `GridLayout`: ordnet die GUI-Komponenten der Reihe nach in einem Gitter an, dessen Zeilen und Spaltenanzahl festgelegt werden kann (nicht mit `Grid`**`Bag`**`Layout` verwechseln).

Layout-Manager im Package `javax.swing`:

- `BoxLayout`: ordnet die GUI-Komponenten entweder entlang einer horizontalen oder vertikalen Achse an, wobei Komponenten, die keinen Platz mehr haben, einfach abgeschnitten oder gar nicht mehr angezeigt werden.
- `GroupLayout`: ermöglicht ein sehr flexibles Layout, bei dem GUI-Komponenten jeweils an den anderen GUI-Komponenten in horizontaler und vertikaler Richtung durch Bilden von Gruppen ausgerichtet werden.
- `OverlayLayout`: ordnet die GUI-Komponenten übereinander an (was meist nur dann Sinn macht, wenn kleinere Komponenten über größeren Komponenten platziert werden)
- `ScrollPaneLayout`: wird von der GUI-Komponente `JScrollPane` benutzt, um einen kleineren Bereich aus einem größeren Teil darzustellen.
- `SpringLayout`: Ein sehr flexibler Layout-Manager, der damit arbeitet Abstände und Bedingungen zwischen den Rändern von GUI-Komponenten festzulegen (z.B. "Komponente 2 ist 5 Pixel vom rechten Rand der Komponente 1 auf gleicher Höhe").
- `ViewportLayout`: Ein Layout, das von der GUI-Komponente `JViewport` verwendet wird.

Gerade die besonders flexiblen Layout-Manager sind oft nur in Verbindung mit einem GUI-Builder (einem Programm, bei dem man die GUI mittels Drag-And-Drop zusammenstellen kann) sinnvoll. In diesem Kapitel liegt der Fokus jedoch auf der grundsätzlichen Funktionsweise von Layout-Managern und die Umsetzung im Programmcode. Deshalb werden hier auch nur die grundlegenden Layout-Manager vorgestellt.

JFrame für die Beispiele

Die vorgestellten Layout-Manager werden jeweils in einem eigenen Panel dargestellt, das in ein `JFrame` als Top-Level-Container eingebunden ist. Dieses ist vom Prinzip her für jedes Beispiel gleich und ist nur hier angegeben. Auch wenn es im Beispiel nicht direkt ersichtlich ist, nutzt `JFrame` auch einen Layout-Managers um das hinzugefügte Panel zentriert über die ganze Fläche darzustellen, den `BorderLayout`-Manager (siehe auch Kapitel). `BorderLayout` ist der Standard-Layout-Manager der JFrame-Klasse und muss daher nicht extra gesetzt werden.

Wenn eine GUI-Komponente (in diesem Fall das Layout-Panel) bei einem `BorderLayout` ohne zweiten Parameter mit `add` hinzugefügt wird, dann wird dieser Component im zentralen Bereich angeordnet und zwar so groß es nur geht. Falls in den anderen Bereichen des `BorderLayout`-Managers keine GUI-Komponenten vorhanden sind, bedeutet das, dass die Komponente im Zentrum den gesamten Platz verbraucht.

```
import javax.swing.JFrame;

public class FrameFlowLayout extends JFrame  {

    public FrameFlowLayout() {
        super("Meine GUI");
        PanelFlowLayout layoutPanel;
        layoutPanel = new PanelFlowLayout();
        this.add(layoutPanel);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBounds(100, 200, 300, 150);
        this.setVisible(true);
    }
      public static void main(String[] args) {
            new FrameFlowLayout();
      }
}
```

Das Layout-Panel des jeweiligen Beispiels wird erzeugt (hier z.B. FlowLayout)

Das Beispiel-Panel wird im Zentrum des BorderLayouts des Frames platziert.

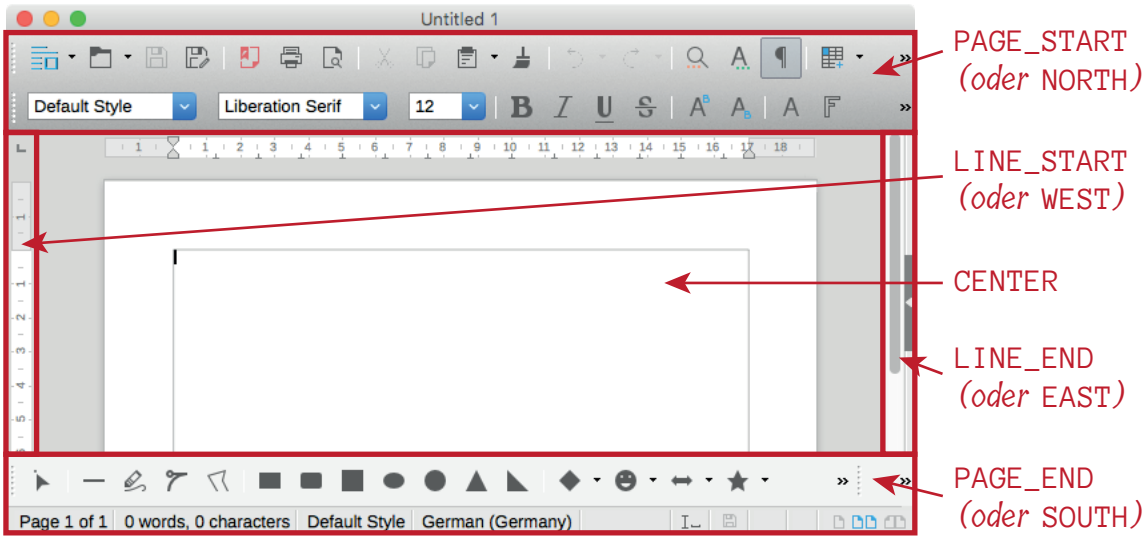
Es ist wichtig, dass die gesamte GUI erst am Schluss sichtbar gemacht wird

Die Größe, die das Frame beim Start hat, wird hier zusammen mit der Start-Postion gesetzt. Für die Screenshots werden diese Werte aber verändert.

[1]

BorderLayout

Der Layout-Manager `BorderLayout` ist das Standard-Layout für `JFrame`, da es der üblichen Anordnung der Komponenten in den fünf Bereichen innerhalb eines GUI-Fenster entspricht. Im Folgenden ist als Beispiel eine Textverarbeitungssoftware mit den Grundbereichen dieses Layout-Managers eingezeichnet:



[1] Wenn das Layout verändert wird oder die Größe einer GUI-Komponente verändert wird, wenn die GUI schon sichtbar ist, kann das Layout auf `invalid` gesetzt werden und die Änderung wird nicht angezeigt. Abhilfe schafft der Aufruf von `revalidate()` und `repaint()`.

Damit bei der Methode `add` der gewünschte Bereich als `constraints`-Parameter angegeben werden kann, bietet die Klasse `BorderLayout` entsprechende Klassenkonstanten:

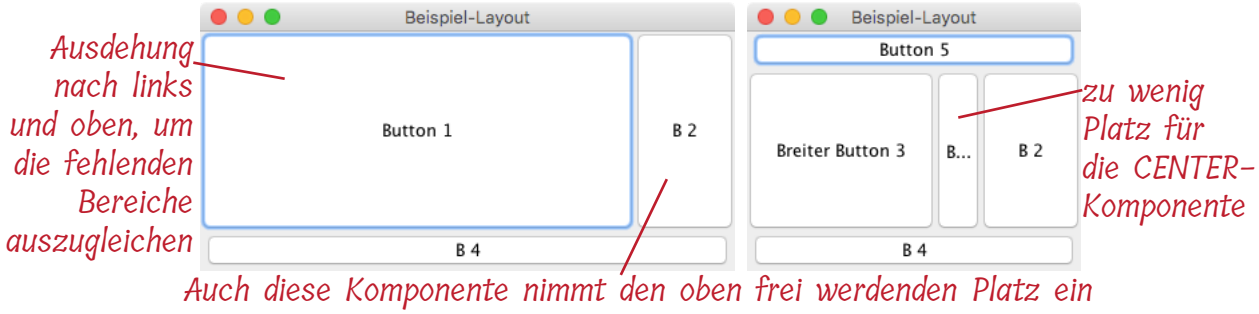
`static String`	**`CENTER`** The center layout constraint (middle of container).
`static String`	**`LINE_END`** The component goes at the end of the line direction for the layout.
`static String`	**`LINE_START`** The component goes at the beginning of the line direction for the layout.
`static String`	**`PAGE_END`** The component comes after the last line of the layout's content.
`static String`	**`PAGE_START`** The component comes before the first line of the layout's content.

Noch einmal zur Erinnerung: Klassenkonstanten werden mit *`Klassenname.Konstantenname`* angegeben (ähnlich wie Klassenmethoden).

Die Bereiche wurden in älteren Java-Versionen nach den Himmelsrichtungen benannt. Aus Kompatibilitätsgründen wurden die alten Konstanten noch beibehalten. Allerdings wird empfohlen, diese in neuen Programmen nicht mehr einzusetzen. Für die Anordnung der Bereiche gelten neben den grundsätzlichen Platzierungen noch folgende Rahmenbedingungen:

- Die PAGE-Bereiche versuchen sich in vertikaler Richtung an den Inhalt bzw. der bevorzugten Größe (`preferredSize`) der GUI-Komponenten anzupassen und so schmal wie möglich zu sein. In horizontaler Richtung den kompletten Platz zwischen dem linken und rechten Rand ausfüllen
- Die LINE-Bereiche versuchen in horizontaler Richtung so schmal wie möglich zu sein, während sie in vertikaler Richtung den Platz zwischen den beiden PAGE-Bereichen ausfüllen.
- Der CENTER-Bereich füllt den kompletten Platz zwischen allen übrigen Bereichen aus. Damit stellt er oft den größten Bereich dar.

Wenn eine oder mehrere der Bereiche weggelassen wird, füllen die anderen Bereiche den verbliebenen Platz aus. Am meisten dehnt sich dabei der CENTER-Bereich aus. Dieser Bereich wird auch als erstes reduziert, wenn zu wenig Platz da ist, um alle GUI-Komponenten vollständig anzuzeigen.



Auch hier hat Mac Os X wieder eine Besonderheit, weil dieses Look-And-Feel bei den Buttons einen sehr breiten Rand neben dem Text reserviert und die Buttons damit nicht "schmal" werden.

Konstruktoren

Die Klasse `BorderLayout` stellt zwei Konstruktoren zur Verfügung:

`BorderLayout()`
Constructs a new border layout with no gaps between components.

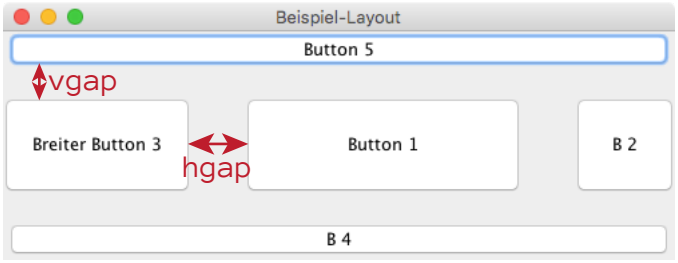
`BorderLayout(int hgap, int vgap)`
Constructs a border layout with the specified gaps between components.

hgap und vgap

Fast alle Layouts haben Konstruktoren, die die Angabe von `hgap` und `vgap` als Parameter ermöglichen. Dabei stehen diese Parameter für folgende Werte:

- `hgap`: *horizontal gap* ... Abstand zwischen den Komponenten in horizontaler Richtung
- `vgap`: *vertical gap* ... Abstand zwischen den Komponenten in vertikaler Richtung

Die genauen Auswirkungen hängen aber auch vom Look-And-Feel der swing-Oberfläche ab. So ist z.B. beim Mac-OsX-Look-And-Feel immer ein Abstand zwischen den Komponenten vorhanden, auch wenn man `hgap` und `vgap` auf 0 setzt (siehe die vorherigen Bilder zum `BorderLayout`).



Programmcode

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

public class PanelBorderLayout extends JPanel{
  private JButton b1, b2, b3, b4, b5;

  public PanelBorderLayout() {
    BorderLayout bl1 = new BorderLayout(40,20);
    this.setLayout(bl1);
    b1 = new JButton("Button 1");
    b2 = new JButton("B 2");
    b3 = new JButton("Breiter Button 3");
    b4 = new JButton("B 4");
    b5 = new JButton("Button 5");
    this.add(b1, BorderLayout.CENTER);
    this.add(b2, BorderLayout.LINE_END);
    this.add(b3, BorderLayout.LINE_START);
    this.add(b4, BorderLayout.PAGE_END);
    this.add(b5, BorderLayout.PAGE_START);
  }
}
```

BorderLayout mit einem horizontalen Abstand von 40 und einen vertikalen Abstand von 20 Pixeln erzeugen.

BorderLayout-Objekt als aktives Layout setzen

JButton-Objekte mit verschiedenen Beschriftungen erzeugen

add-Methode mit einer Klassenkonstante als `constraint`-Objekt (zweiter Parameter) aufrufen

Achtung: In jeden Bereich kann nur **eine** GUI-Komponente gesetzt werden! Wird eine zweite Komponente in den gleichen Bereich gesetzt, wird die erste Komponente aus diesem Bereich entfernt.

FlowLayout

Das FlowLayout ist ein sehr einfaches an "Fließtext" angelehntes Layout Es ist das Standard-Layout von JPanel. Das bedeutet, dass alle GUI-Komponenten der Reihe nach nebeneinander angezeigt werden. Die einzelnen Komponenten werden jedoch so klein wie möglich entsprechend ihres Inhalts oder ihrer bevorzugten Größe (preferredSize) gezeichnet. Wenn eine GUI-Komponente in einer Zeile keinen Platz mehr hat, wird sie in der darauffolgenden Zeile angezeigt. Wie bei einem Text kann man auch die Ausrichtung (linksbündig, zentriert oder rechtsbündig) angeben. Auch dafür gibt es entsprechende Klassenkonstanten in der Klasse FlowLayout.

`static int` **CENTER**
This value indicates that each row of components should be centered.

`static int` **LEFT**
This value indicates that each row of components should be left-justified.

`static int` **RIGHT**
This value indicates that each row of components should be right-justified

Der 3. Button (Button 3) hat keinen Platz mehr in der 1. Zeile, deshalb wird er in die nächste Zeile gesetzt

Kostruktoren

FlowLayout()
Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

FlowLayout(int align)
Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.

FlowLayout(int align, int hgap, int vgap)
Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

Programmcode

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

public class PanelFlowLayout extends JPanel{
  private JButton b1, b2, b3;

  public PanelFlowLayout() {
    FlowLayout lay = new FlowLayout(FlowLayout.RIGHT);
    this.setLayout(lay);
    b1 = new JButton("B 1");
    b2 = new JButton("Button 2");
    b3 = new JButton("Button 3");
    super.add(b1);
    super.add(b2);
    super.add(b3);
  }
}
```

GridLayout

Hierbei sind die Komponenten nach Zeilen und Spalten angeordnet, so dass sich ein Gittermuster ergibt.

Konstruktoren

GridLayout()
Creates a grid layout with a default of one column per component, in a single row.

GridLayout(int rows, int cols)
Creates a grid layout with the specified number of rows and columns.

GridLayout(int rows, int cols, int hgap, int vgap)
Creates a grid layout with the specified number of rows and columns.

Programmcode:

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

public class PanelGridLayout extends JPanel{
  private JButton b1, b2, b3, b4, b5;

  public PanelGridLayout() {
    GridLayout gl1 = new GridLayout(3, 2, 5, 10);
    this.setLayout(gl1);
    b1 = new JButton("Button 1");
    b2 = new JButton("Button 2");
    b3 = new JButton("Button 3");
    b4 = new JButton("Button 4");
    b5 = new JButton("Button 5");
    this.add(b1);
    this.add(b3);
    this.add(b2);
    this.add(b4);
    this.add(b5);
  }
}
```

BoxLayout

Das `BoxLayout` ist im Gegensatz zu den meisten anderen hier vorgestellten Standard-Layouts ein Teil der swing-API. Das Box-Layout kann Komponenten ähnlich wie das `FlowLayout` horizontal aber auch vertikal entlang einer imaginären Achse anordnen. Dabei bietet es die Möglichkeit die Komponente entlang dieser Achse zu verschieben. Im Gegensatz zum `FlowLayout` wird jedoch kein automatischer "Zeilenumbruch" gemacht, wenn der verfügbare Platz zu klein ist.

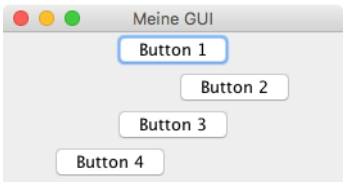
Für die Ausrichtung werden folgende Klassenkonstanten aus der Klasse `Component` aus dem Package `java.awt` verwendet:

- `BOTTOM_ALIGNMENT`: Ease-of-use constant for getAlignmentY.
- `CENTER_ALIGNMENT`: Ease-of-use constant for getAlignmentY and getAlignmentX.
- `LEFT_ALIGNMENT`: Ease-of-use constant for getAlignmentX.

- RIGHT_ALIGNMENT: Ease-of-use constant for getAlignmentX.
- TOP_ALIGNMENT: Ease-of-use constant for getAlignmentY.

Programmcode:

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

public class PanelBoxLayout extends JPanel {

  private JButton b1, b2, b3, b4;

  public PanelBoxLayout() {
    BoxLayout bl = new BoxLayout(this, BoxLayout.LINE_AXIS);
    this.setLayout(bl);
    b1 = new JButton("Button 1");
    b1.setAlignmentX(Component.CENTER_ALIGNMENT);
    b2 = new JButton("Button 2");
    b2.setAlignmentY(Component.BOTTOM_ALIGNMENT);
    b3 = new JButton("Button 3");
    b3.setAlignmentX(Component.CENTER_ALIGNMENT);
    b4 = new JButton("Button 4");
    b4.setAlignmentY(Component.TOP_ALIGNMENT);
    this.add(b1);
    this.add(b2);
    this.add(b3);
    this.add(b4);
  }
}
```



Für eine vertikale Ausrichtung sieht der Programmcode entsprechend geändert aus:

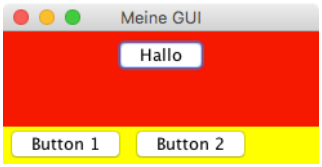
```
import javax.swing.*;
import java.awt.*;

public class PanelBoxLayout2 extends JPanel {
  private JButton b1, b2, b3, b4;

  public PanelBoxLayout2() {
    BoxLayout bl = new BoxLayout(this, BoxLayout.PAGE_AXIS);
    this.setLayout(bl);
    b1 = new JButton("Button 1");
    b1.setAlignmentX(Component.CENTER_ALIGNMENT);
    b2 = new JButton("Button 2");
    b2.setAlignmentX(Component.LEFT_ALIGNMENT);
    b3 = new JButton("Button 3");
    b3.setAlignmentX(Component.CENTER_ALIGNMENT);
    b4 = new JButton("Button 4");
    b4.setAlignmentX(Component.RIGHT_ALIGNMENT);
    this.add(b1);
    this.add(b2);
    this.add(b3);
    this.add(b4);
  }
}
```

Die Ausrichtung orientiert sich bei unterschiedlichen Ausrichtungen an der Mitte der ersten Komponente, die entsprechend der Achsenausrichtung zentriert wird. Wenn alle Komponenten die gleiche Ausrichtung haben, dann orientiert sich die Ausrichtung am Rand des Containers.

```
BoxLayout bl = new BoxLayout(this, BoxLayout.PAGE_AXIS);
this.setLayout(bl);
b1 = new JButton("Button 1");
b1.setAlignmentX(Component.RIGHT_ALIGNMENT);
b2 = new JButton("Button 2");
b2.setAlignmentX(Component.RIGHT_ALIGNMENT);
b3 = new JButton("Button 3");
b3.setAlignmentX(Component.RIGHT_ALIGNMENT);
this.add(b1);
this.add(b2);
this.add(b3);
```

Schachteln von Layout-Managern

Selten wird in einer Oberfläche nur ein Layout verwendet. Für viele GUIs ist das `BorderLayout` das Basis-Layout. Allerdings kann man in einen Bereich des `BorderLayouts` nur eine einzige GUI-Komponente geben. Die meisten Layouts benötigen aber mehrere Komponenten in einem Bereich, z.B. viele Buttons im oberen (`PAGE_START`) Bereich. Um das zu erreichen, kann man allerdings einen Container, z.B. ein `JPanel`-Objekt in den oberen Bereich geben. Dieses `JPanel`-Objekt kann wiederrum ein eigenes Layout haben und dadurch mehrere GUI-Komponenten anordnen. Da diese `JPanel`-Objekte rein der Gruppierung dienen, werden sie nicht als eigene `JPanel`-Klassen umgesetzt. Stattdessen werden direkt aus der Klasse `JPanel` Objekte erzeugt.




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

public class PanelGeschachtelt extends JPanel {
  private JButton b, b1, b2;

  public PanelGeschachtelt() {
    BorderLayout layout = new BorderLayout();
    this.setLayout(layout);
    // zentraler Bereich
    b = new JButton("Hallo");
    JPanel mitte = new JPanel(new FlowLayout()); // "Gruppierungspanel"
    mitte.add(b);
    mitte.setBackground(Color.RED);
    this.add(mitte, BorderLayout.CENTER);

    // unterer Bereich
    b1 = new JButton("Button 1");
    b2 = new JButton("Button 2");
    JPanel panelUnten = new JPanel(); // "Gruppierungspanel"
    panelUnten.setLayout(new BoxLayout(panelUnten,
                                       BoxLayout.LINE_AXIS));
    panelUnten.add(b1); // Buttons zum "Gruppierungspanel"
    panelUnten.add(b2); // hinzufügen und nicht zu this
    panelUnten.setBackground(Color.YELLOW);
    this.add(panelUnten, BorderLayout.PAGE_END);
  }
}
```