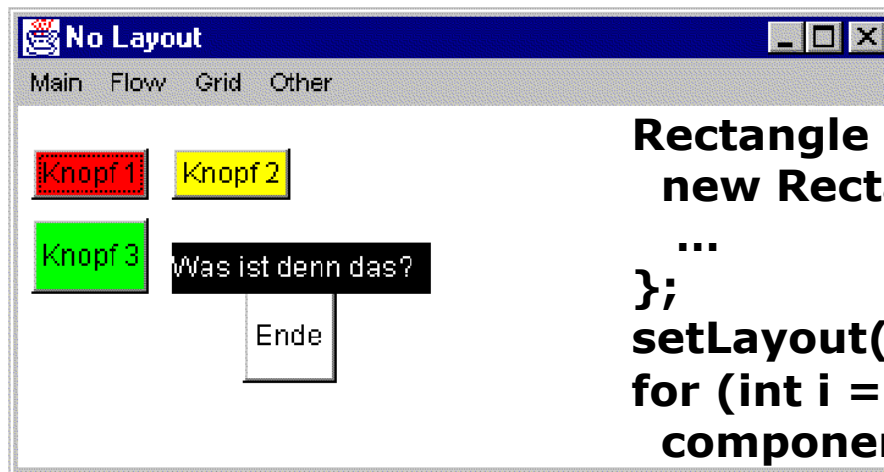


SWING

Layout-Manager

NullLayout

**direkte Platzierung der Komponenten,
durch Angabe der genauen Position,
sowie der Breite und der Höhe**




```
Rectangle bound[] = {  
    new Rectangle( 10, 60, 50, 22),  
    ...  
};  
setLayout(null);  
for (int i = 0; i < component.length; ++i) {  
    component[i].setBounds(bound[i]);  
    add(component[i]);  
}
```

Layout-Manager

- zur Anordnung von Komponenten
- **FlowLayout**: von links nach rechts zentriert in einer Reihe (Standard für JPanel)
- **BorderLayout**: unterteilt in 5 Bereiche: Norden, Süden, Osten, Westen und Mitte (Standard für JFrame)
- **CardLayout**: legt Komponenten wie beim Kartenstapel übereinander; nur die oberste ist sichtbar
- **OverlayLayout**: (Swing) Komponenten übereinander (überlappend)
- **GridLayout**: tabellenartige Anordnung in gleichgroßen Zeilen und Spalten
- **GridBagLayout**: wie GridLayout, aber die einzelnen Zellen (Komponenten) können unterschiedlich groß sein
- **BoxLayout**: (Swing) entweder horizontale oder vertikale Anordnung
- **GroupLayout**: (Swing) Anordnung in horizontalen und/oder vertikalen Gruppen von Komponenten

Zuordnen eines Layoutmanagers



```
public class Dialog1 extends JFrame{

    public Dialog1()
    {
        super("Layoutmanager");           // unser Fenster
        setLayout(new GridLayout(3,2,15,10)); //z, s, hgap, vgap);
        add(new JButton("1"));    add(new JButton("2"));
        add(new JButton("3"));    add(new JButton("4"));
        add(new JButton("5"));    add(new JButton("6"));
        //pack();                  // kleinstmögliches Fenster
        setSize(100,80);          // Größe selbst gesetzt
    }

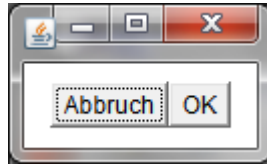
    public static void main(String[] args) {
        Dialog1 dialog = new Dialog1();
        dialog.setVisible(true);
    }
}
```

Layoutmanager

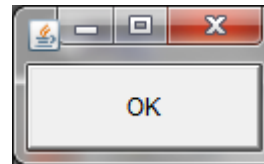
java.awt.Container (setLayout(LayoutManager mgr))

- <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

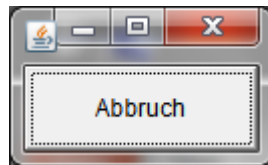
- GridBagLayout()



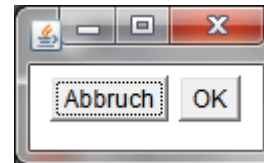
- BorderLayout()



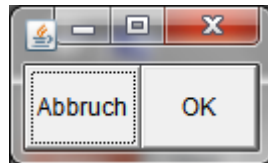
- CardLayout()



- FlowLayout()

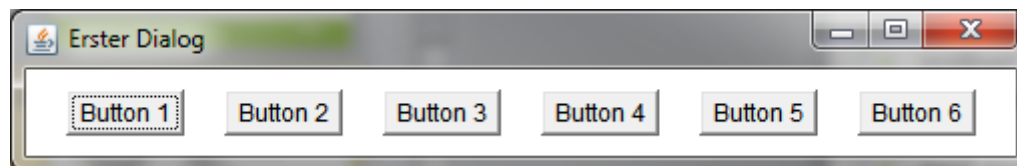


- GridLayout()

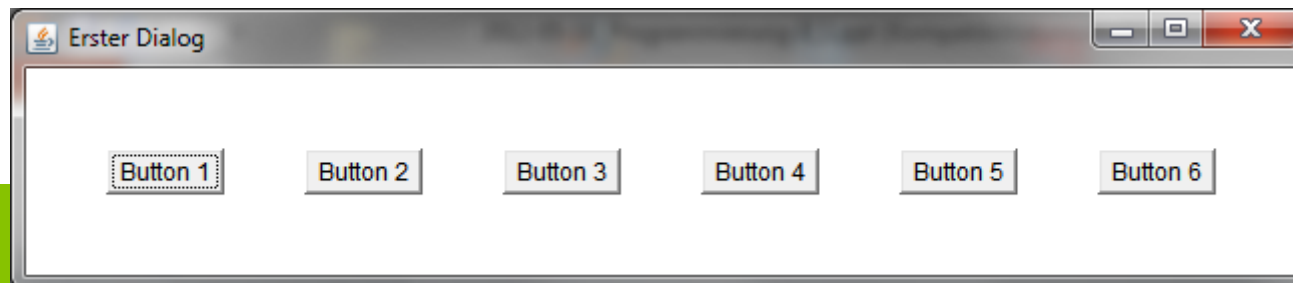


FlowLayout

- ordnet Dialogelemente nebeneinander in einer Zeile an (wenn sie in eine Zeile passen – ansonsten weitere Zeile)
- `setLayout (new FlowLayout())` // parameterloser Konstruktor
- `FlowLayout(int align)` // Bündigkeit
 - `FlowLayout.CENTER` // default
 - `FlowLayout.LEFT` // linksbündig
 - `FlowLayout.RIGHT` // rechtsbündig
- `FlowLayout(int align, int hgap, int vgap)`
 - `hgap` // horizontaler Abstand zwischen den Buttons
 - `vgap` // vertikaler Abstand zwischen den Buttons



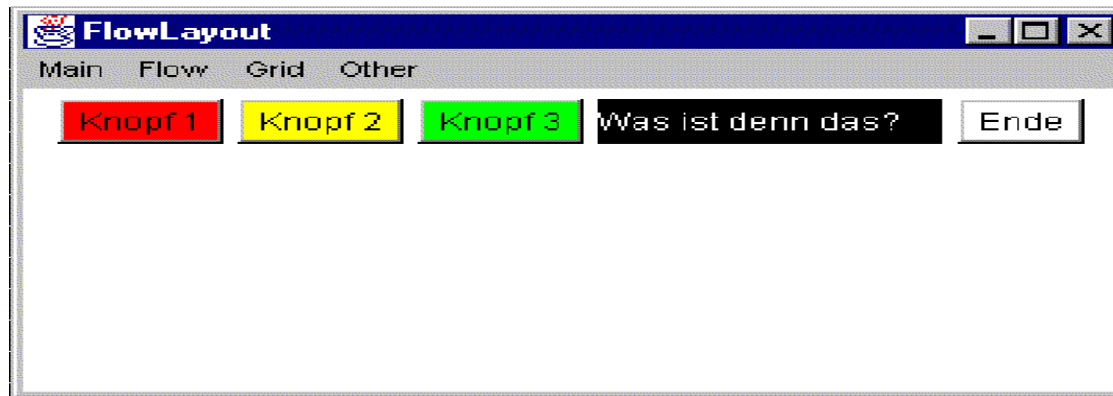
hgap: 20, vgap: 10



hgap: 40, vgap: 40

FlowLayout

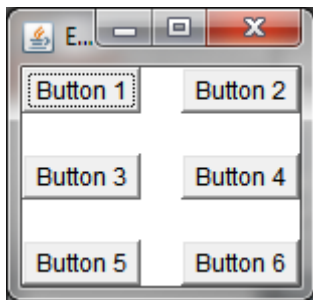
Default-Layout für Container und Komponenten



```
// int adjust = FlowLayout.LEFT;  
// int adjust = FlowLayout.CENTER;  
  
int adjust = FlowLayout.RIGHT;  
  
setLayout(new FlowLayout(adjust));  
  
for (int i = 0; i < component.length; ++i) {  
    add(component[i]);  
}
```

GridLayout

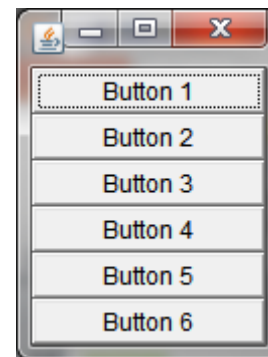
- ordnet Dialogelemente innerhalb eines rechteckigen Gitters an
- `setLayout (new GridLayout(int rows, int columns))`
 - `rows` // Anzahl der Zeilen
 - `columns` // Anzahl der Spalten
- `GridLayout(int rows, int columns, int hgap, int vgap)`



rows: 3, columns: 2 (equal),
hgap: 20, vgap: 20



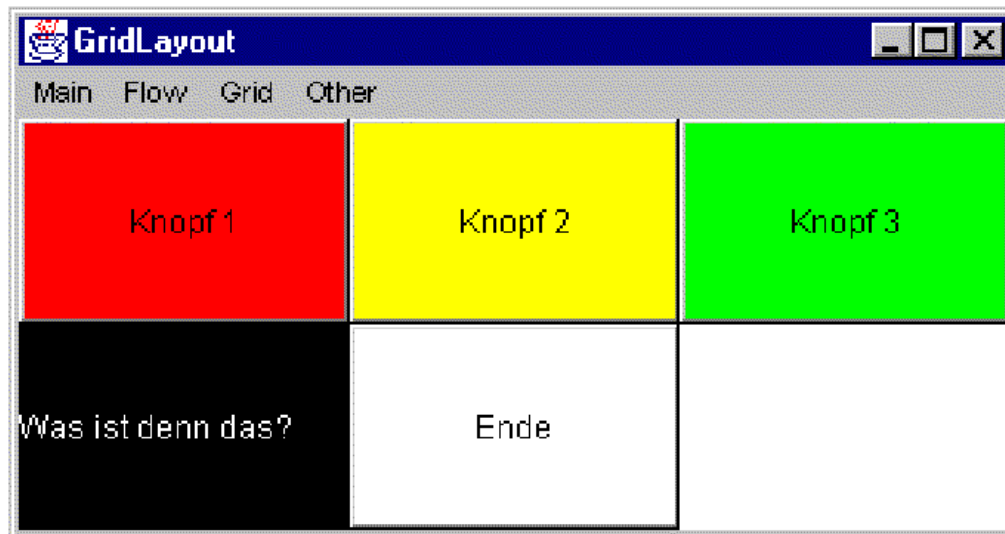
rows: 2,
columns: 3 (equal)



rows: 6,
columns: 1 (equal)

Buttons werden mit größerem Fenster (setSize) größer (im Gegensatz zu FlowLayout)

GridLayout



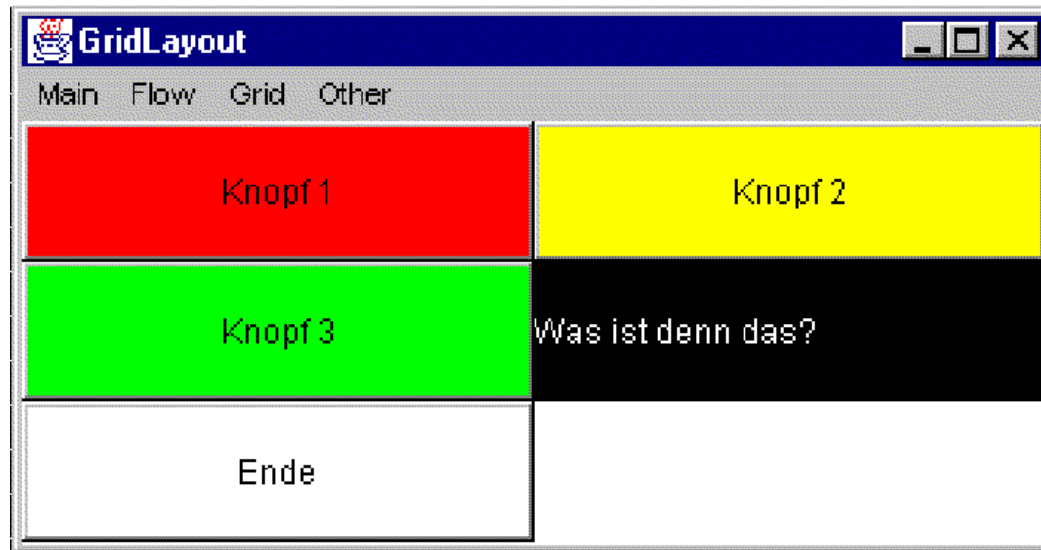
feste Zellenmatrix

2 * 3 Zellenmatrix

```
int rows = 2;  
int cols = 3;  
setLayout(new GridLayout(rows, cols));
```

```
for (int i = 0; i < component.length; ++i) {  
    add(component[i]);  
}
```

GridLayout



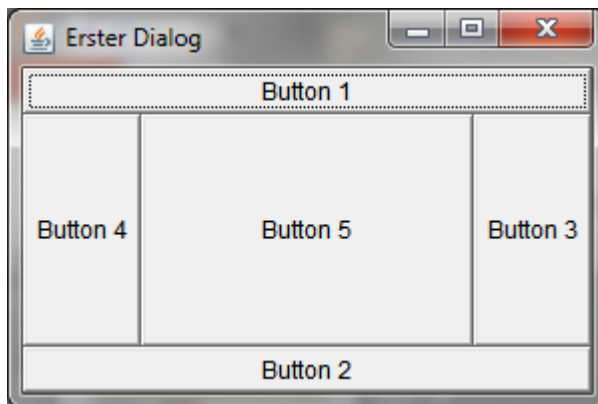
3 * 2 Zellenmatrix

```
int rows = 3;  
int cols = 2;  
setLayout(new GridLayout(rows, cols));
```

```
for (int i = 0; i < component.length; ++i) {  
    add(component[i]);  
}
```

BorderLayout

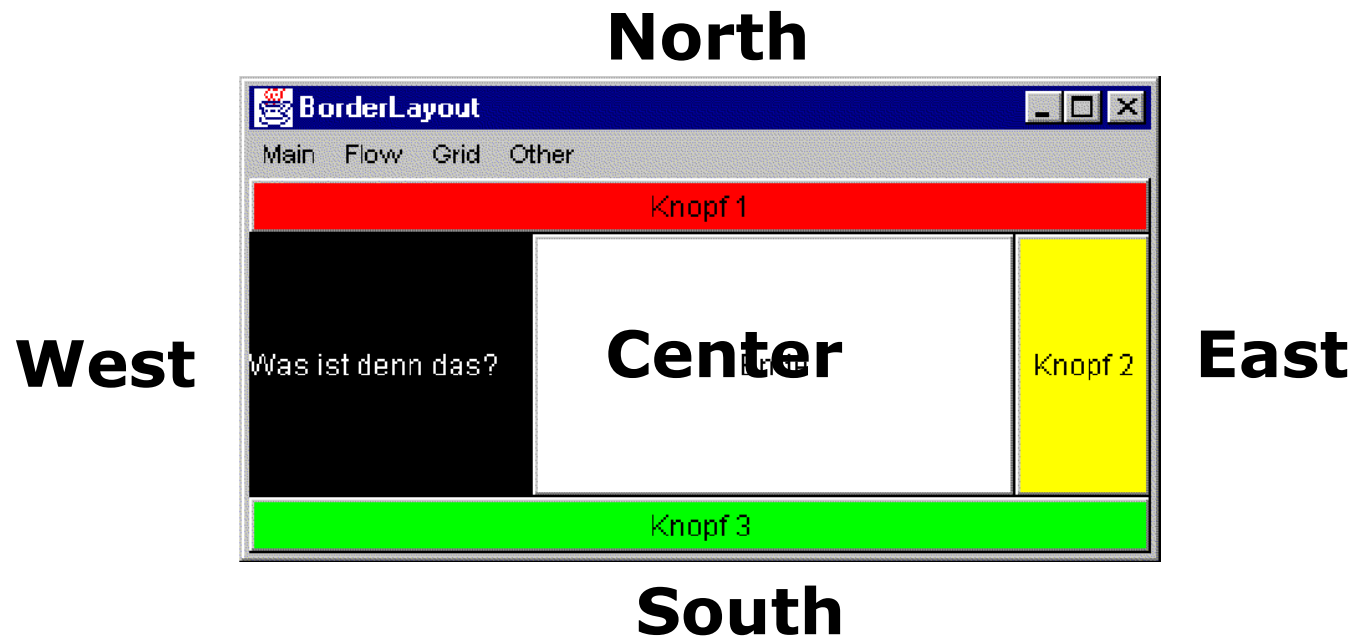
- ordnet Dialogelemente in fünf Ränder (North, South, East, West) und die Mitte (Center) an
- `setLayout (new BorderLayout())` // parameterlos
- `BorderLayout(int hgap, int vgap)` // Abstände zwischen Buttons
- `add` ändert sich: `public void add(Component comp, Object constraints)`



```
setLayout(new BorderLayout());  
add(new JButton("Button 1"), BorderLayout.NORTH);  
add(new JButton("Button 2"), BorderLayout.SOUTH);  
add(new JButton("Button 3"), BorderLayout.EAST);  
add(new JButton("Button 4"), BorderLayout.WEST);  
add(new JButton("Button 5"), BorderLayout.CENTER);  
setSize(300,200);
```

- *Nord-, Süd-Buttons behalten Höhe, skalieren auf Fensterbreite*
- *Ost-, West-Buttons behalten Breite, skalieren auf Fensterhöhe (minus N, S)*
- *Center-Button wird eingepasst (kann ganz verschwinden)*

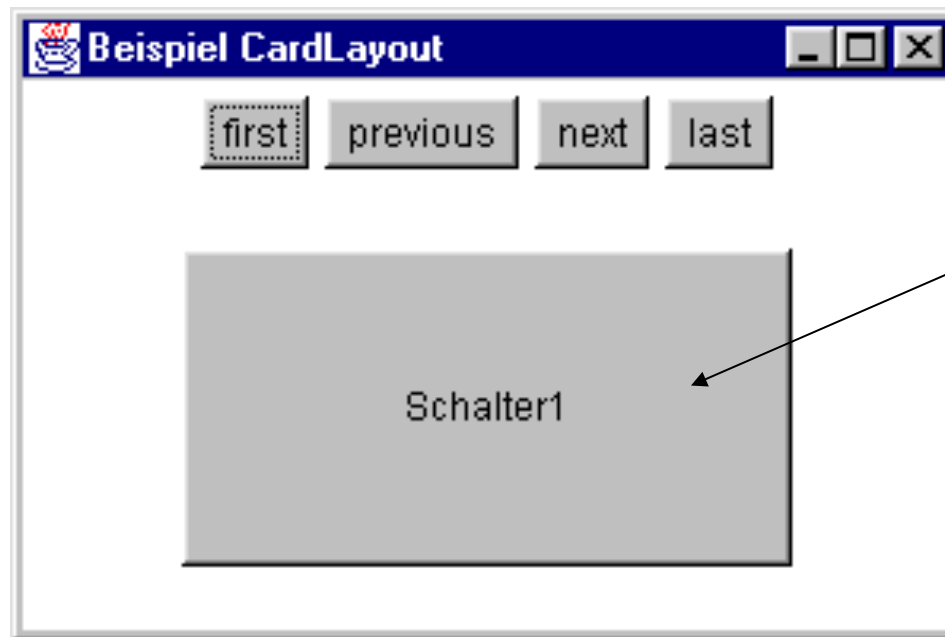
BorderLayout



```
String[] adjust = { BorderLayout.NORTH,  
                    BorderLayout.EAST,  
                    BorderLayout.SOUTH,  
                    BorderLayout.WEST,  
                    BorderLayout.CENTER};  
for (int i = 0; i < component.length; ++i) {  
    add(adjust[i % adjust.length], component[i]);  
}
```

CardLayout

Kartenstapel übereinandergelegt



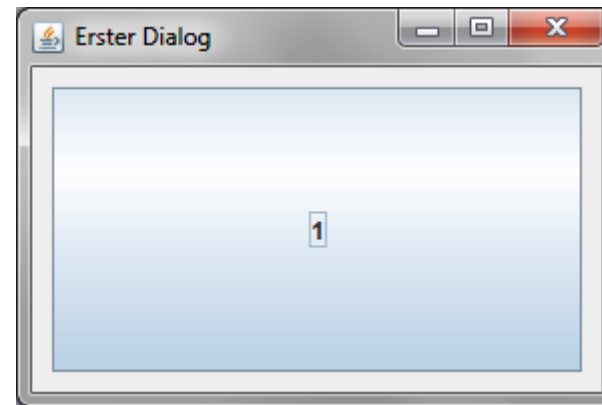
*Je nach Auswahl
des Buttons
schaltet ein
anderes Panel
auf.*

Wurde in Swing ersetzt durch ***JTabbedPane***
(Kartenreiter)

CardLayout Beispiel

```
panel=new JPanel();
panel.setLayout(new
CardLayout(10,10));//hgap, vgap
JButton b1 = new JButton("1");
panel.add(b1);
b1.addActionListener(this);
JButton b2 = new JButton("2");
panel.add(b2);
b2.addActionListener(this);
JButton b3 = new JButton("3");
panel.add(b3);
b3.addActionListener(this);
JButton b4 = new JButton("4");
panel.add(b4);
b4.addActionListener(this);
JButton b5 = new JButton("5");
panel.add(b5);
b5.addActionListener(this);
JButton b6 = new JButton("6");
panel.add(b6);
b6.addActionListener(this);
add(panel);
```

```
@Override
public void actionPerformed(ActionEvent arg0)
{
    CardLayout cl = (CardLayout)
(panel.getLayout());
    cl.next(panel);
}
```



GridBagLayout

Beispiel GridBagLayout

Pizza Service

<i>Adresse</i>	<i>Pizza</i>
Name:	Sorte:
<input type="text"/>	<input type="checkbox"/> Margherita
Vorname:	<input type="checkbox"/> Mais
<input type="text"/>	<input type="checkbox"/> Schinken
Straße:	<input type="checkbox"/> Pfeffersalami
<input type="text"/>	<input type="checkbox"/> Funghi
Hausnummer:	<input type="checkbox"/> Hawaii
<input type="text"/>	<input type="checkbox"/> Mozzarella
	<input type="checkbox"/> Calzone

Anmerkungen

☐ Abholer

Ein komplizierter
Alleskönner

GridBagLayout

Pizza Service		
Adresse	Pizza	
Name:	Sorte:	Extras:
<input type="text"/>	<input type="checkbox"/> Margherita	<input type="checkbox"/> Mais
Vorname:	<input type="checkbox"/> Schinken	<input type="checkbox"/> Pfeffersalami
<input type="text"/>	<input type="checkbox"/> Funghi	
Straße:	<input type="checkbox"/> Hawaii	
<input type="text"/>	<input type="checkbox"/> Mozzarella	
Hausnummer:	<input type="checkbox"/> Calzone	
<input type="text"/>		
Anmerkungen		
<input type="text"/>		
<input type="checkbox"/> Abholer		
Abbruch	Löschen	OK

Wieviele Zellen zählen Sie?

42 (14 * 3)

Gibt es Komponenten, die über mehrere Zellen gehen?

ja

Der GridBagLayout Manager

Vorteile

- hohe Flexibilität
- Universell einsetzbar

Nachteile

- sehr kompliziert
- Wechselwirkungen

Klassen

- GridBagLayout
 - Implementiert den GridBagLayout Manager
- GridBagConstraints
 - Bestimmt die Lage der Komponenten im GridBagLayout

GridBagLayout

- ist komplizierter (dafür sehr flexibel) → in Praxis aber kaum (noch) verwendet
- 1. Objekt des Typs GridBagLayout erzeugen:
 - `GridBagLayout gbl = new GridBagLayout();`
 - `setLayout(gbl);`
- 2. Objekt des Typs GridBagConstraints für jedes einzufügende Dialogelement:
 - `GridBagConstraints gbc = new GridBagConstraints();`
 -
- 3. Objektvariablen des GridBagConstraints-Elements belegen:
 - `gbc.gridx = 0;` // 1. (logische) Spalte des Grids (nicht Pixel!)
 - `gbc.gridy = 0;` // 1. (logische) Zeile des Grids (nicht Pixel!)
 - `gbc.gridwidth=1;` // 1 logische Zelle breit
 - `gbc.gridheight=1;` // 1 logische Zelle hoch
 - `gbc.weightx = 100;` // Ausprobieren!! (überschüssiger Platz)
 - `gbc.weighty = 100;` // Ausprobieren!! (überschüssiger Platz)
 - `gbc.fill = GridBagConstraints.BOTH;` // skalieren in beide Richtungen
// NONE, HORIZONTAL, VERTICAL, BOTH
 - `gbc.insets = new Insets(1,1,1,1);` // Rahmen um das Dialogelement
- Übergabe des Dialogelements zusammen mit den Eigenschaften
 - `List list = new List();` // hier: Dialogelement Liste
 - `gbl.setConstraints(list, gbc);`

Prinzipielle Vorgehensweise

```
Container container;
```

```
...
```

```
GridBagLayout gridbag = new GridBagLayout();
```

```
container.setLayout( gridbag ); // Layout setzen f. Container
```

```
GridBagConstraints constr = new GridBagConstraints();
```

```
// Für alle Komponenten
```

```
Component component;
```

```
...
```

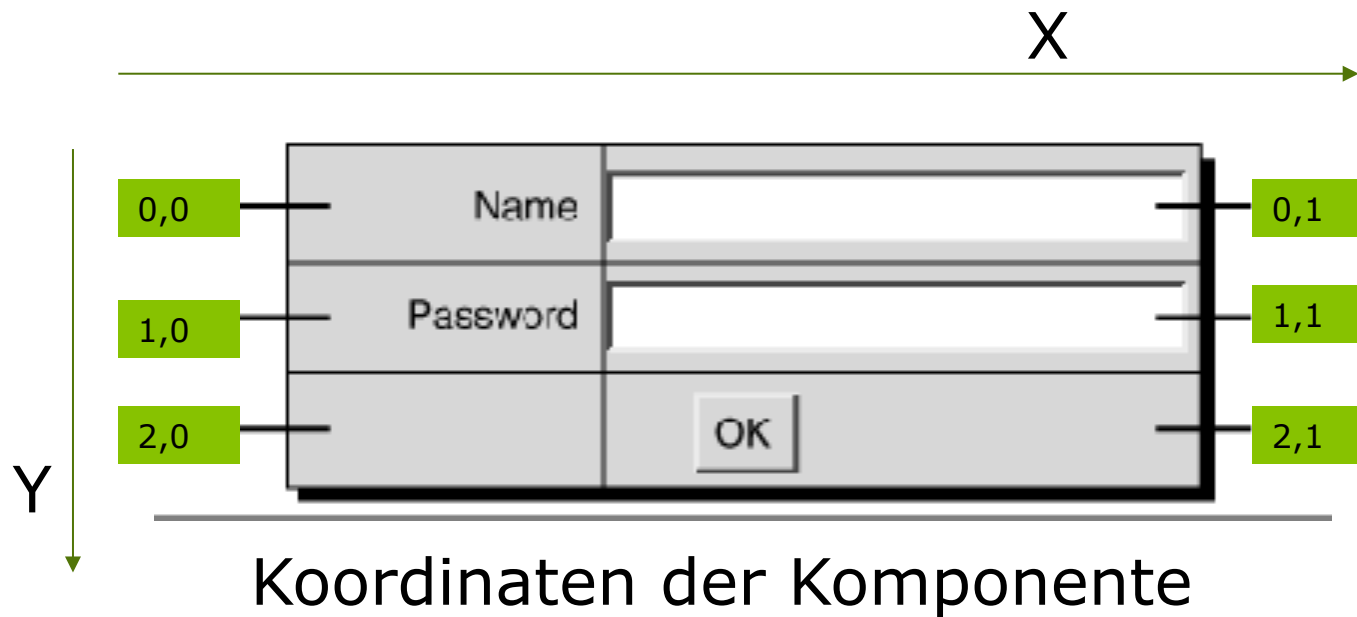
```
constr.XXX = YYY;    // Constraints setzen
```

```
...
```

```
// Komponente auf Container legen mit Constraints
```

```
container.add( component, constr );
```

Beispiel einer Zellen-Matrix



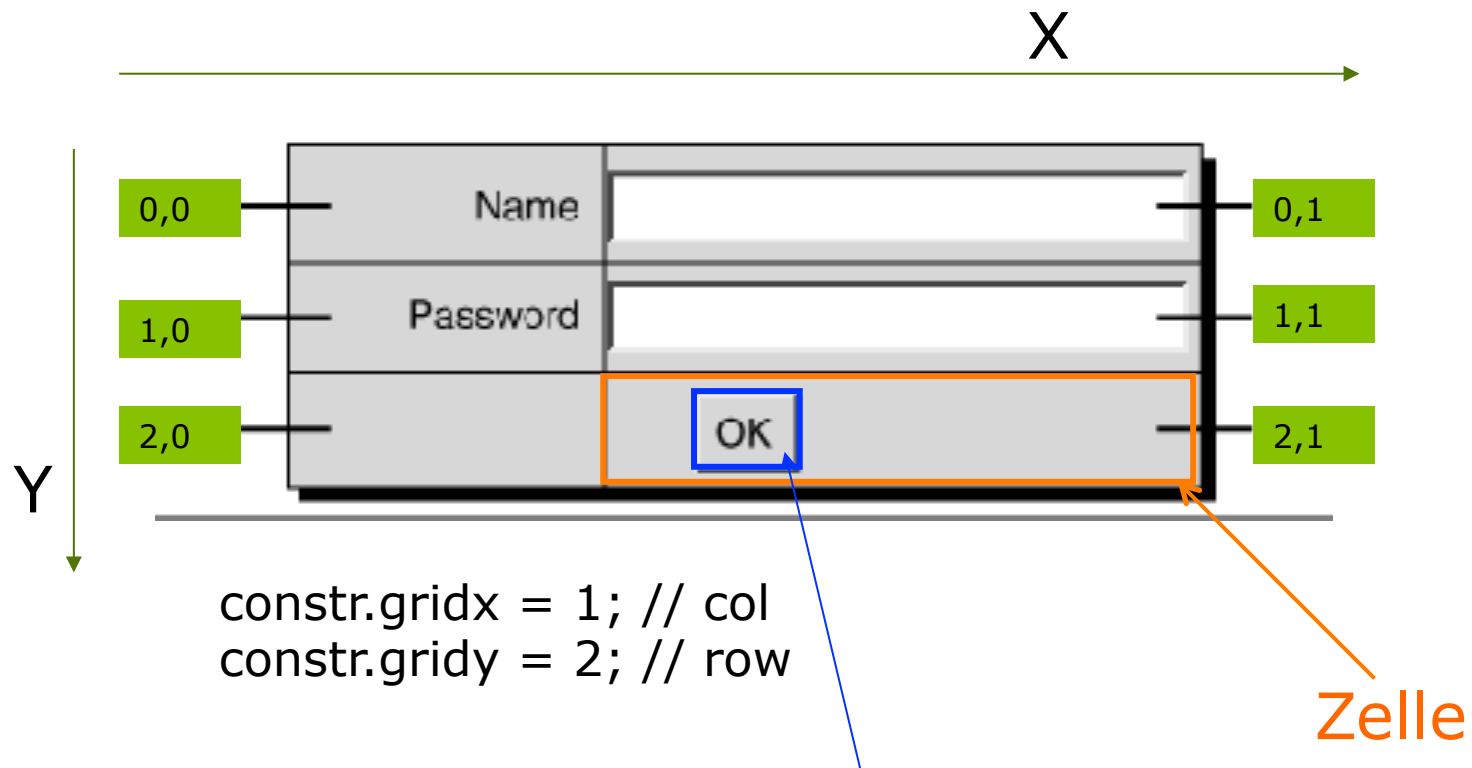
Y = Reihe

X = Spalte

Constraints

- int **gridx**, int **gridy**
 - Gitterposition (Spalte/Reihe) der Komponente in der GridBagMatrix
- Beispiel:
 - `constr.gridx = 1; // col`
 - `constr.gridy = 2; // row`

Beispiel einer Zellen-Matrix



Komponente – hier kleiner als die Zelle

Constraints

- **int `gridwidth`, int `gridheight`**
 - Bestimmt Anzahl der zu belegenden Zellen, default = 1
 - Höhe / Breite der Komponente (in Zellen gemessen)
 - Beispiel:
 - `constr.gridwidth = 2; // zwei Zellen breit`
 - `constr.gridheight = 1; // default: 1 Zelle breit`
- `constr.gridwidth = GridBagConstraints.REMAINDER;`
 - Teilt dem Layoutmanager mit, dass es sich um die letzte Komponente in der Reihe handelt.

Constraints

- **anchor**

- Ausrichtung der Komponente innerhalb der Zelle, default=center
- `constr.anchor = GridBagConstraints.NORTH;`
- `constr.anchor = GridBagConstraints.EAST;`
- `constr.anchor = GridBagConstraints.SOUTH;`
- `constr.anchor = GridBagConstraints.WEST;`
- `constr.anchor = GridBagConstraints.CENTER;`
- `constr.anchor = GridBagConstraints.NORTHEAST;`
- `constr.anchor = GridBagConstraints.NORTHWEST;`
- `constr.anchor = GridBagConstraints.SOUTHEAST;`
- `constr.anchor = GridBagConstraints.SOUTHWEST;`

Constraints

- **fill**

- Füllraum für Komponente innerhalb der Zelle, default:NONE
- `constr.fill = GridBagConstraints.BOTH;`
- `constr.fill = GridBagConstraints.VERTICAL;`
- `constr.fill = GridBagConstraints.HORIZONTAL;`
- `constr.fill = GridBagConstraints.NONE;`

Constraints

- double **weightx**, double **weighty**
 - Bestimmt die Proportionen der Komponente für zusätzlichen Raum in x,y Richtung
 - Gewichtung für zusätzlichen Raum, default: 0.0, Höchstwert: 1.0
- **insets**
 - Abstand der Komponente zum Rand default Insets(0,0,0,0)
- **ipadx, ipady**
 - vergrößern der Komponente

Beispiel

// Layout Manager

GridBagLayout **gridbag** = **new** GridBagLayout();

GridBagConstraints **constr** = **new** GridBagConstraints();

// JButton konstruieren

butOk = **new** JButton("OK");

// Constraints setzen

constr.gridx = 0; // Spalte

constr.gridy = 0; // Zeile

constr.anchor = GridBagConstraints.*EAST*;

constr.gridwidth = GridBagConstraints.*REMAINDER*;

// Button zusammen mit den Constraints an die Contentpane
übergeben

cpane.add(**butOk**, **constr**);

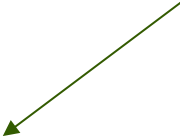
Achtung!

Alle Constraints, die gesetzt wurden bleiben solange erhalten, bis sie erneut gesetzt werden.

```
// Constraints setzen
constr.gridx = 0; // Spalte
constr.gridy = 0; // Zeile
constr.anchor = GridBagConstraints.EAST;
constr.gridwidth = GridBagConstraints.REMAINDER;
// 1tenButton übergeben
cpane.add( butOk, constr);

// 2tenButton übergeben
cpane.add( butRETURN, constr);
```

*Wird mit denselben
Constraints gesetzt
Fehler!*



Deshalb!

Constraints neu setzen, bevor man die Komponente übergibt.

```
// Constraints setzen
constr.gridx = 0; // Spalte
constr.gridy = 0; // Zeile
constr.anchor = GridBagConstraints.EAST;
constr.gridwidth = GridBagConstraints.REMAINDER;
// 1tenButton übergeben
cpane.add( butOk, constr);
```

```
// Constraints setzen
constr.gridx = 0; // Spalte
constr.gridy = 10; // Zeile
// 2tenButton übergeben
cpane.add( butRETURN, constr);
```

Koordinaten der Gittermatrix

constr.gridx = 0; // Spalte
constr.gridy = 10; // Zeile

0,0			0,10
10,0			10,10
20,0			20,10
30,0			

Fehlende Zeilen/Spalten werden nicht dargestellt,
wenn es keine Komponenten dafür gibt.

GroupLayout

- hat GridBagLayout ersetzt → aber nur für „einfache“ GUIs ohne GUI-Designer
- GroupLayout-Manager arbeitet immer mit 2 Gruppen:
 - eine Gruppe richtet die Komponenten horizontal aus (setHorizontalGroup())
 - eine Gruppe richtet dieselben Komponenten vertikal aus (setVerticalGroup())
- jede dieser Gruppen wird eine Gruppenklasse zugeordnet:
 - GroupLayout.SequentialGroup (ordnet die Komponenten nebeneinander (bei HorizontalGroup) oder untereinander (bei VerticalGroup) an)
 - GroupLayout.ParallelGroup (ordnet die Komponenten an einer gemeinsamen Linie aus:

Konstante aus GroupLayout.Alignment	Bei Verwendung als horizontale Gruppe	bei Verwendung als vertikale Gruppe
BASELINE	Ausrichtung an Grundlinie	-
CENTER	Zentrierung	Zentrierung
LEADING (Standard)	Ausrichtung am linken Rand	Ausrichtung am oberen Rand
TRAILING	Ausrichtung am rechten Rand	Ausrichtung am unteren Rand

GroupLayout Beispiel

```
private JPanel init()
{
    JPanel panel = new JPanel();

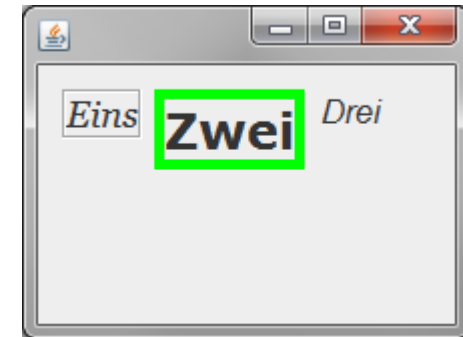
    // Layout-Manager instanziiieren und einrichten
    GroupLayout panelLayout = new GroupLayout(panel);
    panel.setLayout(panelLayout);

    //automatische Abstände einstellen
    panelLayout.setAutoCreateGaps(true);
    panelLayout.setAutoCreateContainerGaps(true);

    // Komponenten erzeugen
    JLabel label1 = new JLabel("Eins");
    label1.setBorder(BorderFactory.createEtchedBorder());
    label1.setFont(new Font("Georgia", Font.ITALIC, 18));

    JLabel label2 = new JLabel("Zwei");
    label2.setBorder(BorderFactory.createLineBorder(Color.GREEN, 5));
    label2.setFont(new Font("Verdana", Font.BOLD, 24));

    JLabel label3 = new JLabel("Drei");
    label3.setBorder(BorderFactory.createBevelBorder(4, Color.RED, Color.GRAY));
    label3.setFont(new Font("Arial", Font.ITALIC, 16));
}
```



GroupLayout Beispiel

```
//horizontale Gruppe
```

```
GroupLayout.SequentialGroup horizGroup = panelLayout.createSequentialGroup();  
horizGroup.addComponent(label1);  
horizGroup.addComponent(label2);  
horizGroup.addComponent(label3);
```

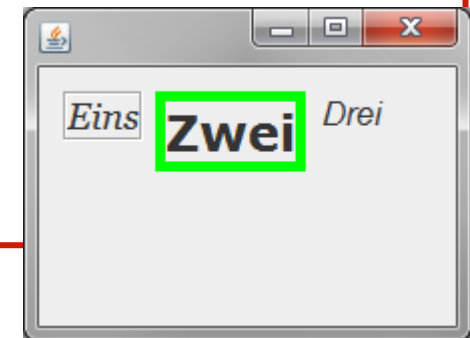
```
//vertikale Gruppe
```

```
GroupLayout.ParallelGroup vertGroup =  
panelLayout.createParallelGroup(GroupLayout.Alignment.LEADING); //oben  
vertGroup.addComponent(label1);  
vertGroup.addComponent(label2);  
vertGroup.addComponent(label3);
```

```
panelLayout.setHorizontalGroup(horizGroup);  
panelLayout.setVerticalGroup(vertGroup);
```

```
return panel;
```

```
}
```



GroupLayout Beispiel

- kleine Änderung

```
//horizontale Gruppe
GridLayout.SequentialGroup horizGroup = panelLayout.createSequentialGroup();
horizGroup.addComponent(label1);
horizGroup.addComponent(label2);
horizGroup.addComponent(label3);

//vertikale Gruppe
//GridLayout.ParallelGroup vertGroup =
// panelLayout.createParallelGroup(GroupLayout.Alignment.LEADING); //oben
GridLayout.SequentialGroup vertGroup =
panelLayout.createSequentialGroup();vertGroup.addComponent(label1);
vertGroup.addComponent(label2);
vertGroup.addComponent(label3);

panelLayout.setHorizontalGroup(horizGroup);
panelLayout.setVerticalGroup(vertGroup);

return panel;
}
```



GroupLayout – das Beispiel

horizontale Gruppe (Sequentielle Gruppe)

Parallele Gruppe

label1
label2

Parallele Gruppe

textFeld
passwortFeld

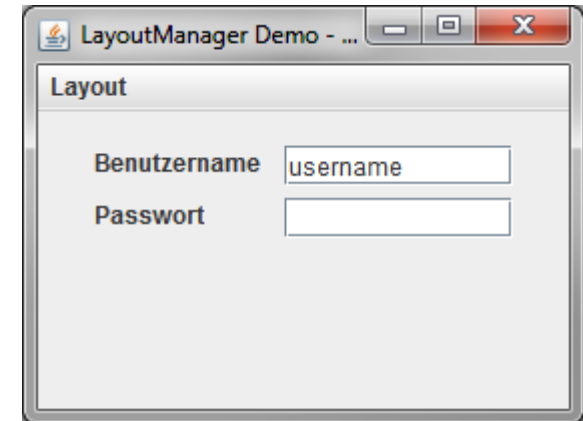
vertikale Gruppe (Sequentielle Gruppe)

Parallele Gruppe

label1 textFeld

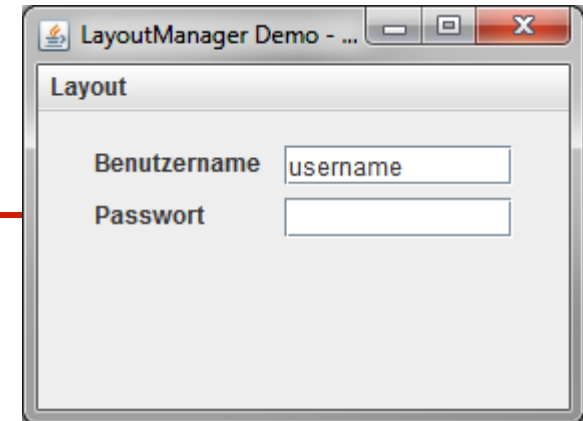
Parallele Gruppe

label2 passwortFeld

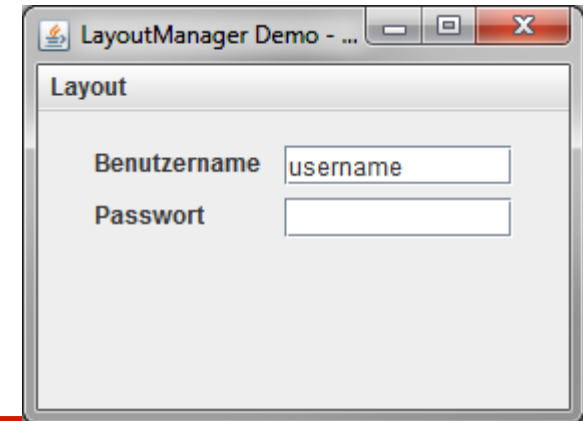


GroupLayout – ein Beispiel

```
private JPanel init() {  
    JPanel panel = new JPanel();  
    GroupLayout groupLayout = new GroupLayout(panel);  
    panel.setLayout(groupLayout);  
  
    JLabel label1      = new JLabel("Benutzername");  
    JTextField textFeld = new JTextField("username");  
    JLabel label2      = new JLabel("Passwort");  
    JPasswordField passwortFeld = new JPasswordField(10);  
  
    // automatische Abstände  
    groupLayout.setAutoCreateGaps(true);  
    groupLayout.setAutoCreateContainerGaps(true);  
  
    GroupLayout.SequentialGroup horizGruppe = groupLayout.createSequentialGroup();  
    GroupLayout.ParallelGroup hsub1 = groupLayout.createParallelGroup();  
    hsub1.addComponent(label1);  
    hsub1.addComponent(label2);  
    GroupLayout.ParallelGroup hsub2 = groupLayout.createParallelGroup();  
    hsub2.addComponent(textFeld);  
    hsub2.addComponent(passwortFeld);  
    horizGruppe.addGroup(hsub1);  
    horizGruppe.addGroup(hsub2);  
}
```



GroupLayout – ein Beispiel



```
GroupLayout.SequentialGroup vertGruppe = GroupLayout.createSequentialGroup();
GroupLayout.ParallelGroup vsub1 = GroupLayout.createParallelGroup();
vsub1.addComponent(label1);
vsub1.addComponent(textFeld);
GroupLayout.ParallelGroup vsub2 = GroupLayout.createParallelGroup();
vsub2.addComponent(label2);
vsub2.addComponent(passwordFeld);

vertGruppe.addGroup(vsub1);
vertGruppe.addGroup(vsub2);

groupLayout.setHorizontalGroup(horizGruppe);
groupLayout.setVerticalGroup(vertGruppe);

JPanel gesamt = new JPanel();
gesamt.add(panel);
return gesamt;
}
```

Schachteln von Layoutmanagern

Panel einfügen

```
//Panel 1
```

```
JPanel panel1 = new JPanel();  
panel1.setLayout(new GridLayout(3,2));  
panel1.add(new JButton("Button 1"));  
panel1.add(new JButton("Button 2"));  
panel1.add(new JButton("Button 3"));  
panel1.add(new JButton("Button 4"));  
panel1.add(new JButton("Button 5"));  
panel1.add(new JButton("Button 6"));
```

```
//Panel 2
```

```
JPanel panel2 = new JPanel();  
panel2.setLayout(new FlowLayout(FlowLayout.RIGHT));  
panel2.add(new JButton("Abbruch"));  
panel2.add(new JButton("OK"));
```

```
//Hauptfenster
```

```
setLayout(new BorderLayout());  
add(panel1, BorderLayout.CENTER);  
add(panel2, BorderLayout.SOUTH);  
pack();
```

