JAVA AWT

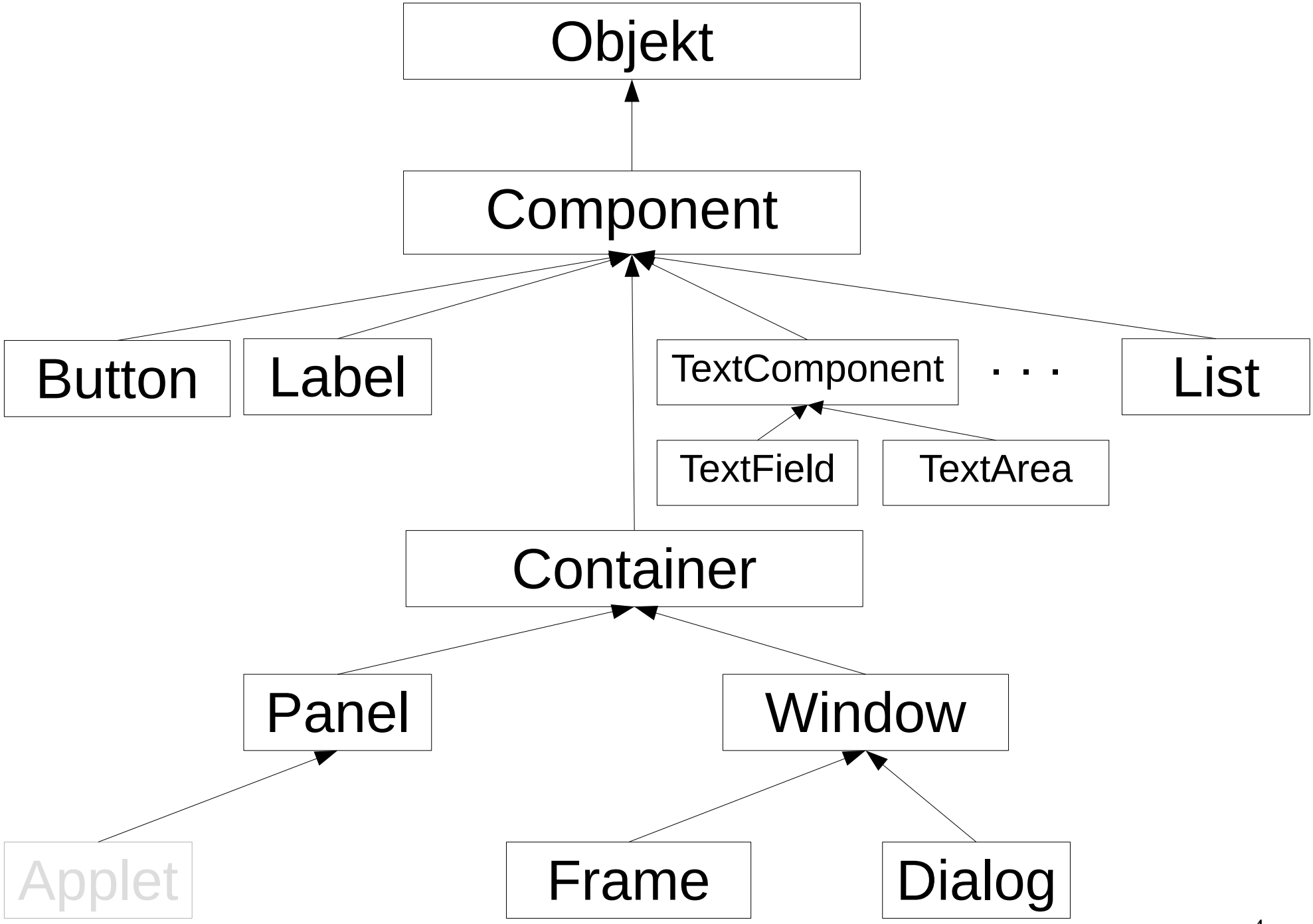
- Wir verlassen die Welt der KonsolAnwendungen und lernen, wie man Anwendungen mit GraphicUserInterface schreibt.
- Macht deutlich mehr Spass!
- Es gibt verschiedene JavaLibraries für die GUI-Programmierung:
  - AbstractWindowingToolkit (AWT)
  - Swing
  - SWT (mit swt ist eclipse programmiert)
  - JavaFX (deutlich anders)

Abstract Windowing Toolkit

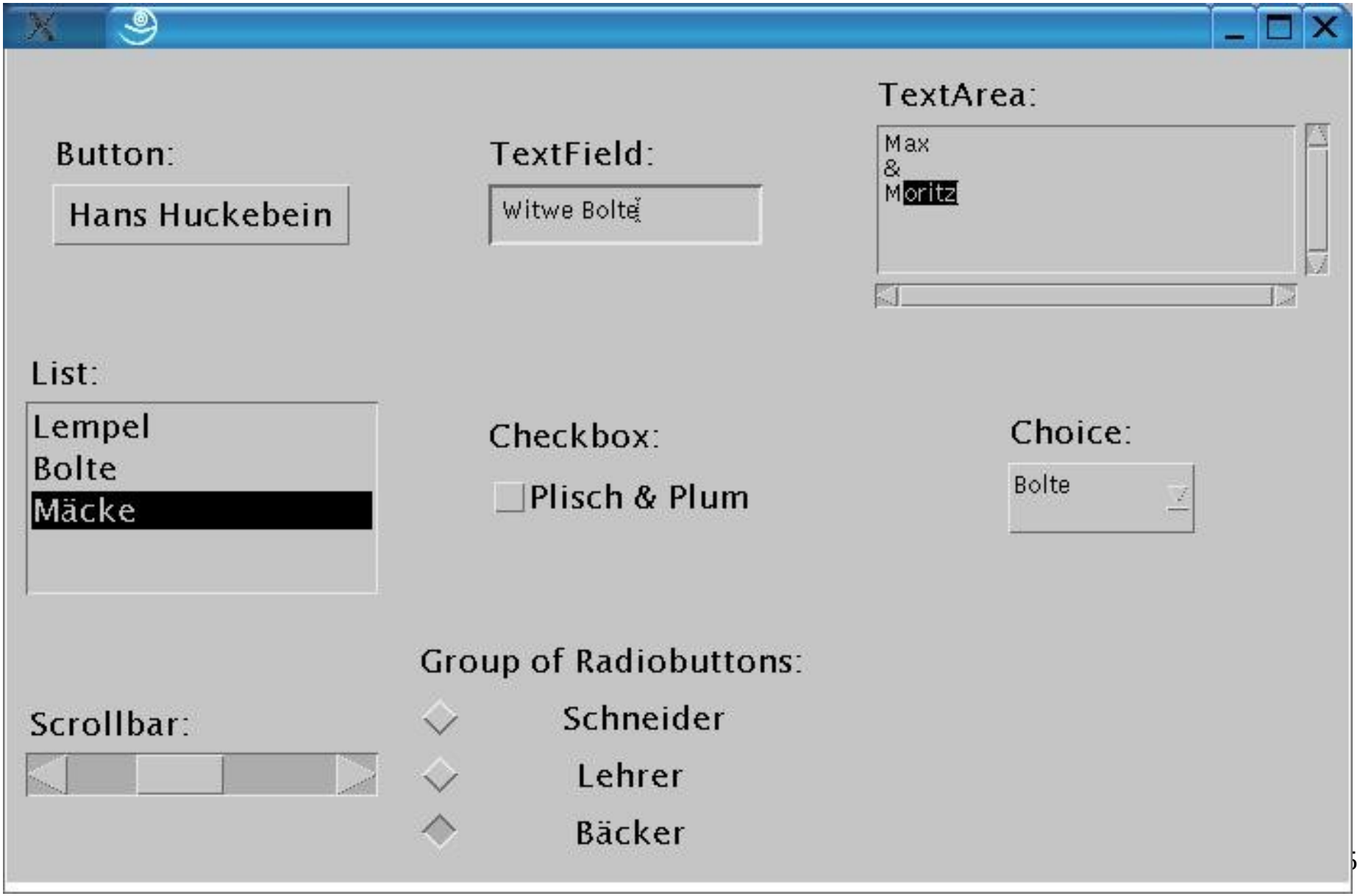
- import java.awt.*
- Package aus der java Klassenbibliothek, stellt Klassen für graphische Benutzeroberflächen bereit
- Die Komponenten (Button, TextField usw.) basieren auf den Implementationen der zugrunde liegenden Plattform
- look & feel ist somit plattformbestimmt

Bestandteile

- Klassen zum Bau der Oberfläche
  - Components
  - Container
  - LayoutManager
- Eventhandling
  - Listener
  - Adapterclasses
  - Events


Objekt
Component
Button
Label
TextComponent
. . .
List
TextField
TextArea
Container
Panel
Window
Applet
Frame
Dialog

AWT-Components



Einbau eines Controls

- Referenzvariable erzeugen
  - Button okButton;
- Objekt erzeugen
  - okButton=new Button(“OK“);
- Control (Button) in Container einsetzen
  - myPanel.add(okButton);
  - add(okButton); // entspr. this.add(okButton);
- Ggf. mit EventListener verbinden
  - okButton.addActionListener(myActionListener);

Die Container

- Frame
  - Das Hauptfenster einer Applikation
- Dialog
  - Dialogfenster (modal/nicht modal)
- Applet
  - Hauptfenster für ein Applet
- Panel
  - Unsichtbarer Container, verhält sich nach außen, wie eine Component, kann je nach Layoutmanager unterschiedlich viele Komponenten aufnehmen

Frame

- Nur für Applikationen auf dem Desktop
- Eine Applikation hat genau ein Objekt der Klasse Frame
- Default LayoutManager ist BorderLayout (wird später betrachtet)
- Die Größe des Frame wird bestimmt durch die Funktionen
  - public void setSize(int width, int height);
  - public void pack();

setSize vers. pack

setSize:
Die Größe des Frame (der Appliaktion) wird durch die angegebenen Paremeter bestimmt. Der Inhalt muss sich (irgendwie einfügen). Da Javaprogramme unter ganz verschiedenen Plattformen (und Auflösungen) laufen, kann das problematisch sein.

pack:
Es wir von jeder, in der Oberfgläche vorkommenden Komponente die bevorzugte Größe ermittelt, mit diesen Angaben dann die Größe der umgebenden Container und auf diese Weise bestimmt, wie groß die Applikation sein uss, um optimal angezeigt zu werden.
Diese methode ist in der Regel zu bevorzugen!

Dialog

- Muss immer mit einem Frame-Objekt vebunden sein, Applets können keine Dialoge haben.
- Default LayoutManager ist BorderLayout.
- Dient der Eingabe von Werten oder Einstellungen.
- Es gibt vordefinierte Dialoge, wie FileOpenDialog oder PrintDialog.
- Modal / nicht modal
  - Modal: Die Anwendung wird eingefroren, solange der Dialog geöffnet ist (z.Bsp.: FileOpenDialog).
  - Nicht Modal: es kann beliebig zwischen Applikations- und Dialogfenster gerwechselt werden (z.Bsp.: FindDialog).

Applet (nicht mehr relevant)

- Hat keinen Constructor sondern eine parameterlose init-Funktion.
- Ist immer in eine html-Seite eingebunden.
- Hat keinen Zugang zu Ressourcen des Rechners, auf dem es ausgeführt wird (Sandbox).
- Kann keine Dialoge öffnen und keine Menübars enthalten.
- Kann Netzverbindungen immer nur zu dem Host aufbauen, von dem es geladen worden ist.
- Für signierte Applets können Einschränkungen aufgehoben werden.

Panel

- Unsichtbarer Container.
- Verhält sich nach außen, wie eine Komponente.
- Eine Javaanwendung (Applet oder Applikation) kann beliebig viele Panels verwenden.
- Default LayoutManager ist FlowLayout.
- Kann auch als Grundlage zum Bau neuer Komponenten verwendet werden.
- Unscheinbar, weil nicht sichtbar, aber am meisten verwendeter Container.

Die LayoutManager

FlowLayout

Alle Komponenten werden von links nach rechts nebeneinander angeordnet.

GridLayout

Die Fläche wird in gleich große Zellen, angeordnet in Zeilen und Spalten eingeteilt.

Borderlayout

Es stehen 5 Felder (Zellen) zur Verfügung; in jedes Feld kann genau eine Komponente aufgenommen werden; die Felder sind entspredchend ihrer Lage nach den Himmelsrichtungen benannt.

CardLayout

Wie bei einem Stapel Spielkarten wird jede Komponente auf einer neuen Karte abgebildet; es ist immer nur die oberste Karte sichtbar.

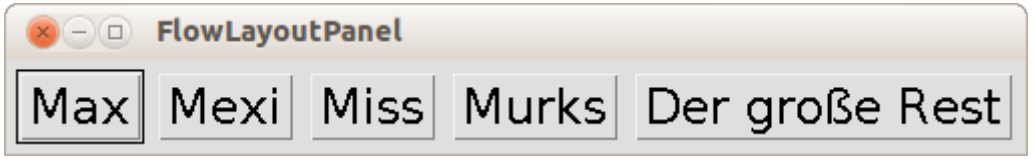
GridbagLayout

Der universellste Layoutmanager - etwas komplizierter

FlowLayoutPanel

```
import java.awt.*;
import java.awt.event.*;
public class FlowLayoutPanel extends Panel
{
  Button b1=new Button("Max");
  Button b2=new Button("Mexi");
  Button b3=new Button("Miss");
  Button b4=new Button("Murks");
  Button b5=new Button("Der große Rest");

  public FlowLayoutPanel()
  {
    setFont(new Font("System", Font.PLAIN, 24));
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(b3);
    add(b4);
    add(b5);
  }
```

```
  public static void main(String args[])
  {
    FlowLayoutPanel p=new FlowLayoutPanel();
    Frame f=new Frame("FlowLayoutPanel");
    f.add(p);

    f.addWindowListener(new WindowAdapter()
            {
              public void windowClosing(WindowEvent e)
              {
                System.exit(0);
              }
            });

    f.setVisible(true);
    f.pack();
  }
}
```



Anmerkungen dazu

- Der Aufbau dieses Programms soll die Grundlage eines jeden AWT-Javaprogrammes sein, das Sie künftig schreiben!
- Die Klasse ist von Panel abgeleitet. Somit können wir ein Objekt dieser Klasse, wie eine beliebige andere Komponente in eine andere Javaanwedung aufnehmen.
- Für die verwendeten Komponenten werden Instanzvariable angelegt.
- Die gesamte Oberfläche wird im Constructor erzeugt.

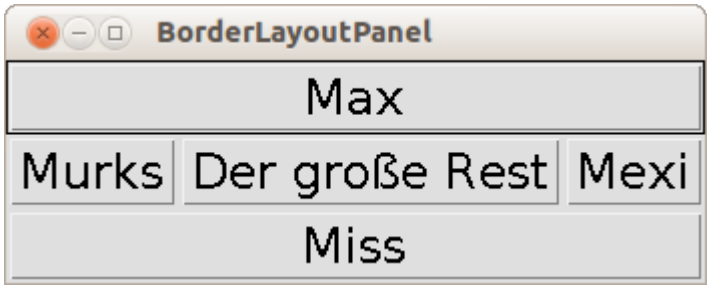
Anmerkungen dazu

- Mit Hilfe der Methode add der Klasse Container fügen wir Komponenten in Container ein. Die Anordnung der Komponenten innerhalb der Container regeln die Layoutmanager (hier FlowLayout).
- Es gibt überschriebene add-Funktionen, die je nach Layoutmanager die Positionierung/Darstellung der Komponenten im Container beeinflussen.

Anmerkungen dazu

- Die main-Funktion dient dem Test der eigenen Klasse. Sie könnte auch in einer separaten Klasse/Javadatei untergebracht sein.
- Die Main-Funktion hat nahezu immer den hier gezeigten Aufbau
  - Erzeugung eines Objektes der eigenen Klasse,
  - Erzeugung eine Frame-Objektes
  - Einsetzen des Objektes (von Panel abgeleitet) in den Frame
  - Festlegung der Größe (f.pack();)
  - Hinzufügen eines WindowListeners (wird späte betrachtet).

Constructor BorderLayoutPanel




```
public BorderLayoutPanel()
{
  setFont(new Font("System", Font.PLAIN, 24));
  setLayout(new BorderLayout());
  add(b1,BorderLayout.NORTH);
  add(b2,BorderLayout.EAST);
  add(b3,BorderLayout.SOUTH);
  //add(new Button("TEST"),BorderLayout.SOUTH);
  add(b4,BorderLayout.WEST);
  add(b5,BorderLayout.CENTER);
}
```

Weiteres Vorgehen

- Speichern Sie das beispiel FlowlayoutPanel undet dem Namen BorderLayoutPanel
- Mit Suchen und Ersetzen ändern Sie alle Vorkommen von FlowLayout in BorderLayout.
- Ersetzten Sie den Conrtuctor der den für das BorderLayoutPanel (verhergehende Folie).
- Speichern und Compilieren Sie BordelayoutPanel – es sollte funktionieren .

Anmerkungen zu BorderLayout

- Die Oberfläche des Containers wird in 5 Bereiche (Zellen) eingeteilt(NORTH, SOUTH, EAST, WEST, CENTER).
- Die Zuordnung zu einer der fünf Zellen erfolgt über die Konstanten(NORTH, SOUTH, EAST, WEST, CENTER), die als zweiter Parameter in der überladenenen add-Methode angegeben werden.
- Diese Konstanten sind in der Klasse Borderlayout defniert und deshalb über beispielsweise BorderLayout.WEST zu verwenden.
- Jede Zelle des Layouts kann **nur eine Komponente** aufnehmen.

- Sollen mehrere Komponenten in einer Zelle platziert werten, dfasst man sie wiederum in einem Panel zusammen.
- Hierzu muss man nicht immer eine neue Klasse von Penel ableiten, man kann Panel auch einfach instanzieren.

  Panel p1=new Panel(new FlowLayout());

  und Komponenten mit p1.add(myComponent); einsetzen.
- Liegen alle bisher zu AWT programmierten Klassen in einem Verzeichnis, können Sie auch an Stelle

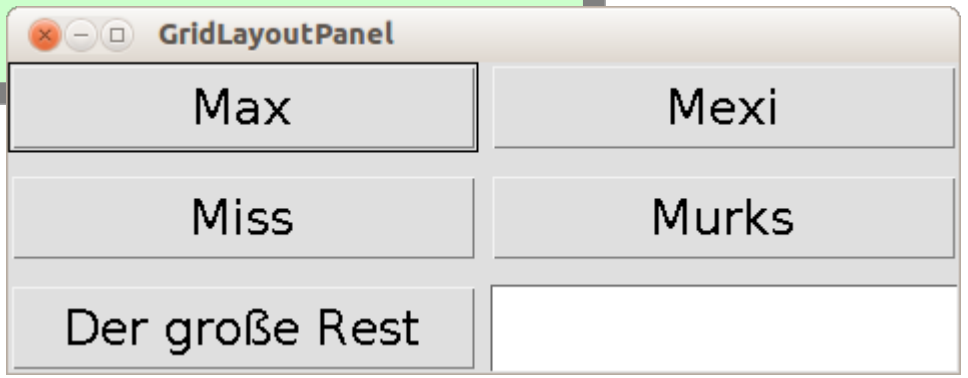
  add(b3,BorderLayout.SOUTH);

  add(new FlowLayoutPanel(),BorderLayout.SOUTH);

  programmieren, probieren Sie‘s aus!

Constructor GridLayoutPanel

```
public GridLayoutPanel()
{
  setFont(new Font("System", Font.PLAIN, 24));
  setLayout(new GridLayout(3,2,5,5));
  add(b1);
  add(b2);
  add(b3);
  add(b4);
  add(b5);
}
```



Anmerkungen zu GridLayout

- Die Oberfläche des Containers wird in eine Tabelle von **gleich großen Zellen** eingeteilt.
- Die Zuordnung der Komponenten erfolgt zeilenweise.
- Im Constructor wird die Anzahl der Zeilen/Spalten festgelegt:

  new GridLayout(3,2);

  oder

  new GridLayout(3,2,5,10);

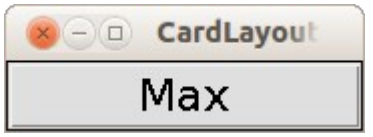
  dabei bestimmen die Werte 5 und 10 den horizontalen/vertikalen Abstand zueinander.

CardLayout

- CardLayout ist etwas speziell. Man stelle sich ein Kartenspiel vor, ein Stapel von Spielkarten , bei dem nur die Oberste zu sehen ist. So arbeitet CardLayout.
- Im Unterschied zu anderen LayoutManagern ruft man Funktionen des LayoutManagers aus der Applikation auf.
- Im Beispiel wird dies gezeigt, mit cards.next(this), um die nächste Komponente oben, sichtbar auf den Stapel zu legen. Die vormals oberste Komponente liegt jetzt ganz unten.
- Der Aufruf im Beipiel erfolgt innerhalb eines ActionListeners, das wird später behandelt.

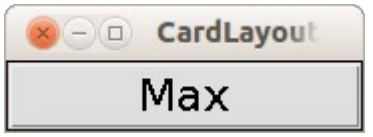
Constructor CardLayoutPanel

```
CardLayout cards=new CardLayout();

public CardPanel()
{
  setFont(new Font("System",Font.PLAIN,22));
  setLayout(cards);
  add(b1); b1.addActionListener(this);
  add(b2); b2.addActionListener(this);
  add(b3); b3.addActionListener(this);
  add(b4); b4.addActionListener(this);
  add(b5); b5.addActionListener(this);
}

public void actionPerformed(ActionEvent e)
{
  cards.next(this);
}
```



Constructor CardLayoutPanel Vers. 2

```
public CardLayoutPanel()
{
  setFont(new Font("System",Font.PLAIN,22));
  setLayout(cards);
  add("eins",b1); b1.addActionListener(this);
  add("zwei",b2); b2.addActionListener(this);
  add("drei",b3); b3.addActionListener(this);
  add("vier",b4); b4.addActionListener(this);
  add("funf",b5); b5.addActionListener(this);
  cards.show(this,"drei");
}

public void actionPerformed(ActionEvent e)
{
  cards.previous(this);
}
```



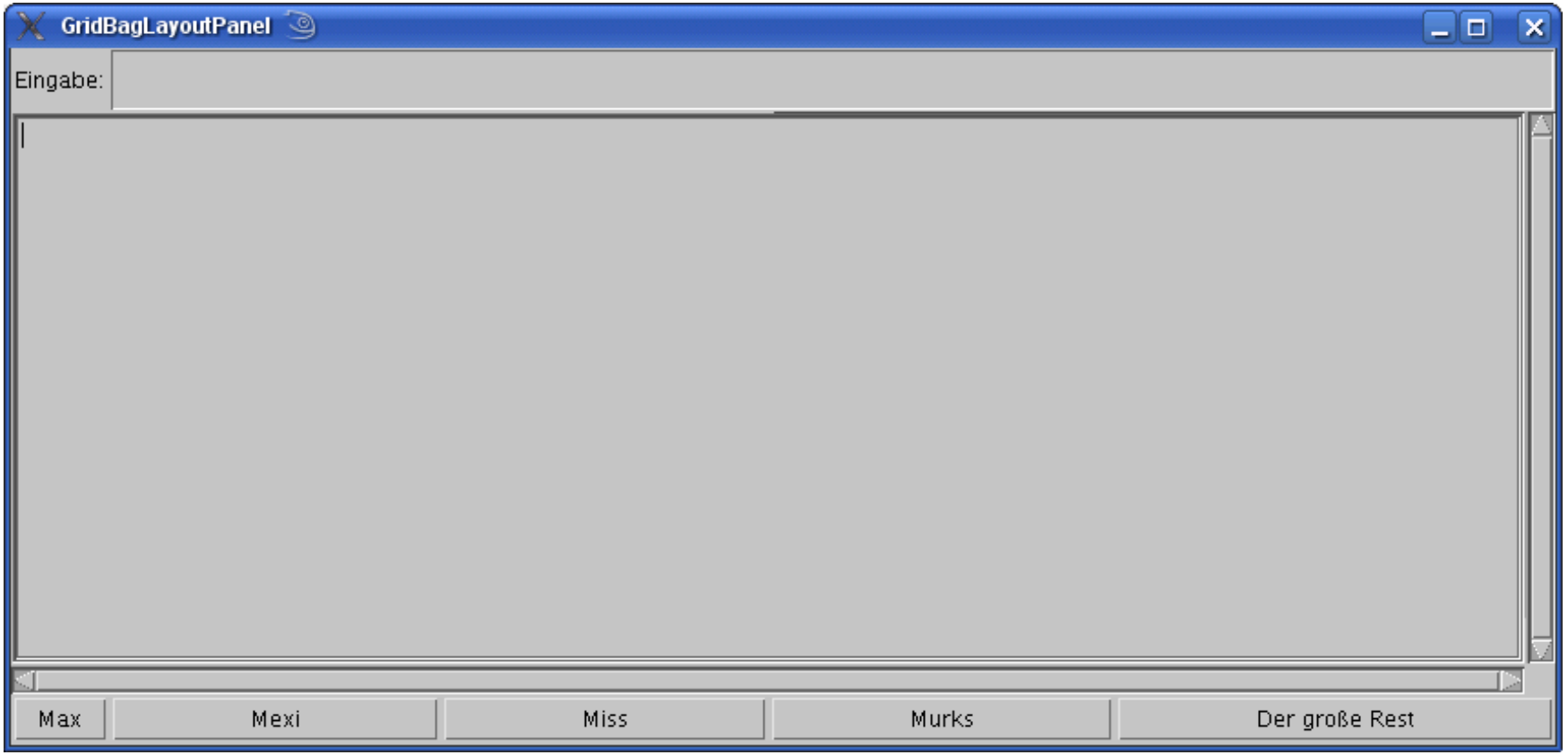
Anmerkungen zu CardLayout 2

- Im Beispiel wird von eine etwas andere Methode add verwendet. Es ist eine überladenene add-Methode von Container.
- Der angegebene String dient der Identifizierung der jeweilgen Komponente.
- Mit cards.show(this,"drei"); wird die Komponente, die mit dem Identifier “drei“ eingefügr wurde nach oben geholt.

GridBagLayout

- Fläche wird in Gitter (Tabelle) eingeteilt.
- Die Größe einer Zelle richtet sich nach der größten Komponente in jeweils Zeile und Spalte.
- Zellen können zu größeren Zellen zusammengefasst werden.
- Zellen können sequenziell oder gezielt belegt werden.
- Umfangreiche Einflussnahme auf Darstellung ist möglich.
- Einstellungen erfolgen mit Hilfe eines GridBagConstraints -objektes.

Beispiel:
Alle Komponenten sind hier in einem Panel angeordnet.
Das Layout besteht aus 3 Zeilen und 5 Spalten.
Die Zellen mit dem Index 1,2,3 und 4 der ersten Zeile wurden zu einer Zelle zusammengefasst.
Alle Zellen der 2. Zeile wurden zu einer Zelle zusammengefasst.
In der 3. Zeile belegt jeweils ein Button eine Zelle.



GridBagConstraints

- Die Steuerung der Anordnung und Darstellung einer Komponente in einer oder meheren Zellen des Layouts erfolgt über ein Objekt der Klasse GidBagConstraints.
- Es wird der add-Methode von Container als 2. Parameter übergeben.
- Das GridBagConstraints-Objekt kann für jede einzubauende Komponente verändert und wiederverwendet werden.
- Dabei können Einstellungen, die uverändert belieben, übernommen werden.
- Die Einstellungen werden direkt in die Instanzvariablen vorgenommen, es gibt keine getter und setter!

GridBagConstraints

gridx	Horizontale Position im Gitter	0,1,… , RELATIVE
gridy	Vertikale Position im Gitter	
gridwidth	Anzahl der Zellen, die horizontal zusammengefasst werden	1,2,… , RELATIVE; REMAINDER
gridheight	Anzahl der Zellen, die vertikal zusammengefasst werden	
fill	Größe der Komponente im Verhältnis zur Größe der Zelle	NONE, HORIZONTAL, VERTICAL, BOTH
ipadx	Pixel die der Größe der Komponente zugefügt werden	
ipady		
insets	Leerer Rand um die Komponente in der Zelle	
anchor	Position der Komponente in der Zelle	CENTER (the default), NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST
weightx	Gewicht bei Resize (horizontal)	0.0 … 1.0
weighty	Gewicht bei Resize (vertikal)	

GridbagConstraints

aus der API-Dokumentation

gridx, gridy
Specifies the cell at the upper left of the component's display area, where the upper-left-most cell has address gridx=0, gridy=0. Use GridBagConstraints.RELATIVE (the default value) to specify that the component be just placed just to the right of (for gridx) or just below (for gridy) the component that was added to the container just before this component was added.

gridwidth, gridheight
Specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1. Use GridBagConstraints.REMAINDER to specify that the component be the last one in its row (for gridwidth) or column (for gridheight). Use GridBagConstraints.RELATIVE to specify that the component be the next to last one in its row (for gridwidth) or column (for gridheight).

fill
Used when the component's display area is larger than the component's requested size to determine whether (and how) to resize the component. Valid values are GridBagConstraints.NONE (the default), GridBagConstraints.HORIZONTAL (make the component wide enough to fill its display area horizontally, but don't change its height), GridBagConstraints.VERTICAL (make the component tall enough to fill its display area vertically, but don't change its width), and GridBagConstraints.BOTH (make the component fill its display area entirely).

ipadx, ipady
Specifies the internal padding: how much to add to the minimum size of the component. The width of the component will be at least its minimum width plus ipadx*2 pixels (since the padding applies to both sides of the component). Similarly, the height of the component will be at least the minimum height plus ipady*2 pixels.

insets
Specifies the external padding of the component -- the minimum amount of space between the component and the edges of its display area.

anchor
Used when the component is smaller than its display area to determine where (within the area) to place the component. Valid values are GridBagConstraints.CENTER (the default), GridBagConstraints.NORTH, GridBagConstraints.NORTHEAST, GridBagConstraints.EAST, GridBagConstraints.SOUTHEAST, GridBagConstraints.SOUTH, GridBagConstraints.SOUTHWEST, GridBagConstraints.WEST, and GridBagConstraints.NORTHWEST.

weightx, weighty
Used to determine how to distribute space; this is important for specifying resizing behavior. Unless you specify a weight for at least one component in a row (weightx) and column (weighty), all the components clump together in the center of their container. This is because when the weight is zero (the default), the GridBagLayout puts any extra space between its grid of cells and the edges of the container.

Beispiel

```
import java.awt.*;
import java.awt.event.*;
public class GridBagLayoutPanel extends Panel
{
  GridBagConstraints C=new  GridBagConstraints();
  Button b1=new Button("Max");
  Button b2=new Button("Mexi");
  Button b3=new Button("Miss");
  Button b4=new Button("Murks");
  Button b5=new Button("Der große Rest");

  TextArea  ta=new TextArea(20,100);
  Label      l=new Label ("Eingabe:");
  TextField tf=new TextField(100);
```

```
public GridBagLayoutPanel()
{
  setLayout((new GridBagLayout()));
  C.weightx=1.0;C.weighty=1.0;
  C.fill=GridBagConstraints.BOTH;

  C.gridx=0; C.gridy=0;
  add(l,C);

  C.gridx=1;
  C.gridwidth=GridBagConstraints.REMAINDER;
  add(tf,C);

  C.gridwidth=GridBagConstraints.REMAINDER;
  C.gridx=0; C.gridy=1;
  add(ta,C);

  C.gridwidth=1;
  C.gridy=2;
  add(b1,C);

  C.gridx=GridBagConstraints.RELATIVE;
  add(b2,C);
  add(b3,C);
  add(b4,C);
  add(b5,C);
}
```

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

    GridBagLayoutPanel p=new GridBagLayoutPanel();
    Frame f=new Frame("GridBagLayoutPanel");
    f.setLayout(new BorderLayout());
    f.add(p, BorderLayout.CENTER);

    f.addWindowListener(new WindowAdapter()
          {
           public void windowClosing(WindowEvent e)
           { System.exit(0); }
          });
    f.pack();
    f.setVisible(true);
  }
}
```