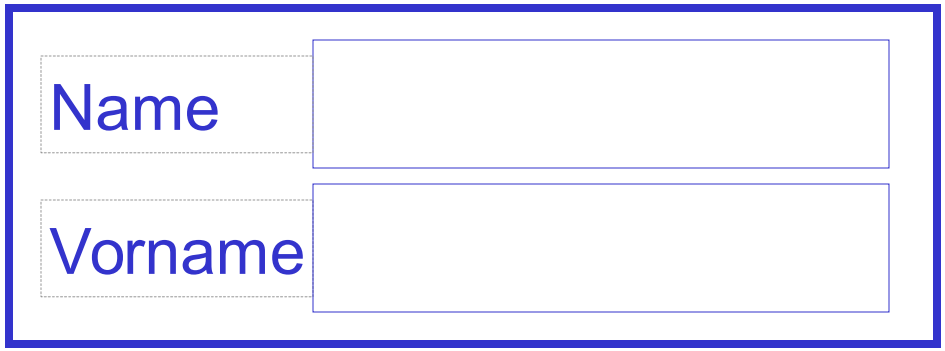
# 5. Layout-Management

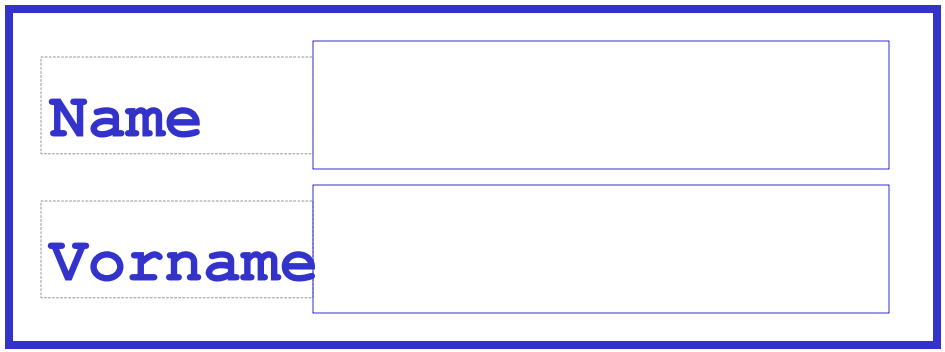
# 5.1 Grundlagen des Layout-Managements

- Explizites Setzen der Geometrie von Komponenten (x,y,width,height) ist nicht sinnvoll, weil
  - solche Anwendungen i.d.R. **nicht** auf unterschiedlichen Plattformen ablauffähig sind (Änderung der Screen Resolution, Änderung der verfügbaren Fonts)
  - es darüber hinaus umständlich ist
- Internationalisierte Anwendungen (und die sollen die Regel sein) benötigen ein automatisiertes Layout, da sich die Label-Texte ändern, wenn man eine andere Sprachvariante startet
- Größenänderung des Toplevel-Windows durch den Benutzer muß sich auch auf die genutzte Geometrie auswirken

Beispiel

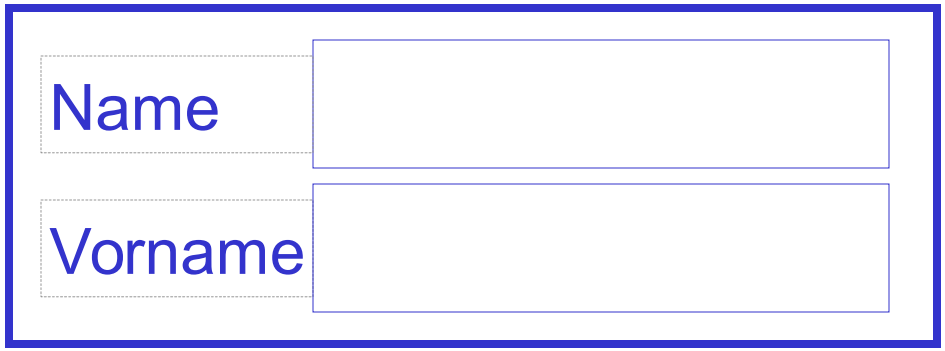
Wechsel der Plattform: ein verwendeter Font ist nicht installiert und wird durch einen Standard Font ersetzt

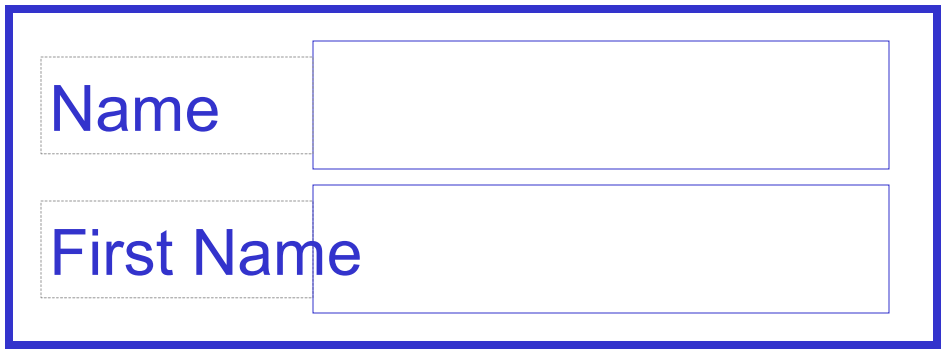


Beispiel

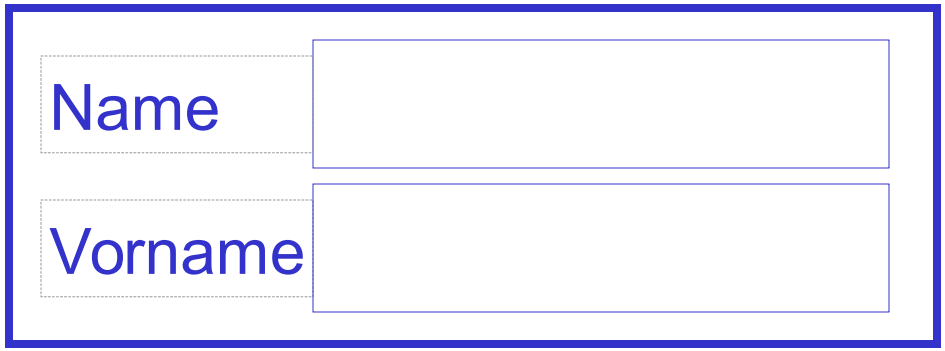
Wechsel der Sprache

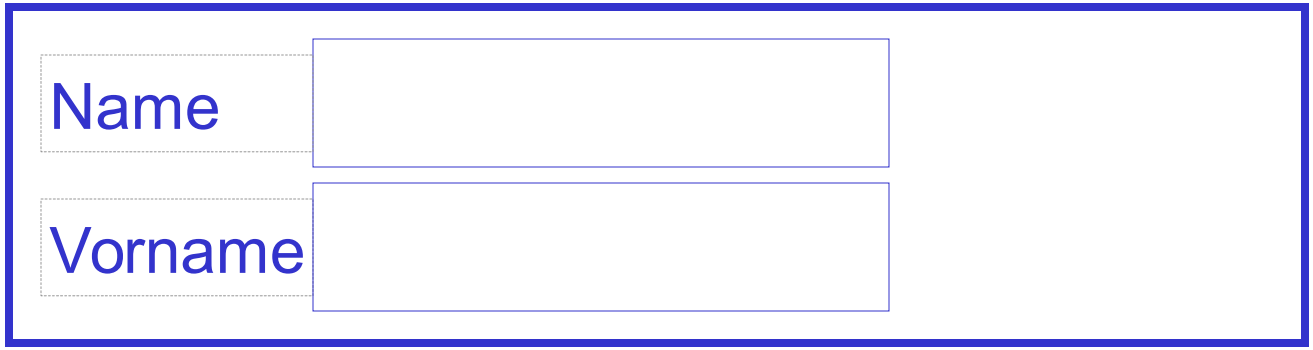


## Beispiel

Größenänderung
des
Toplevel Windows



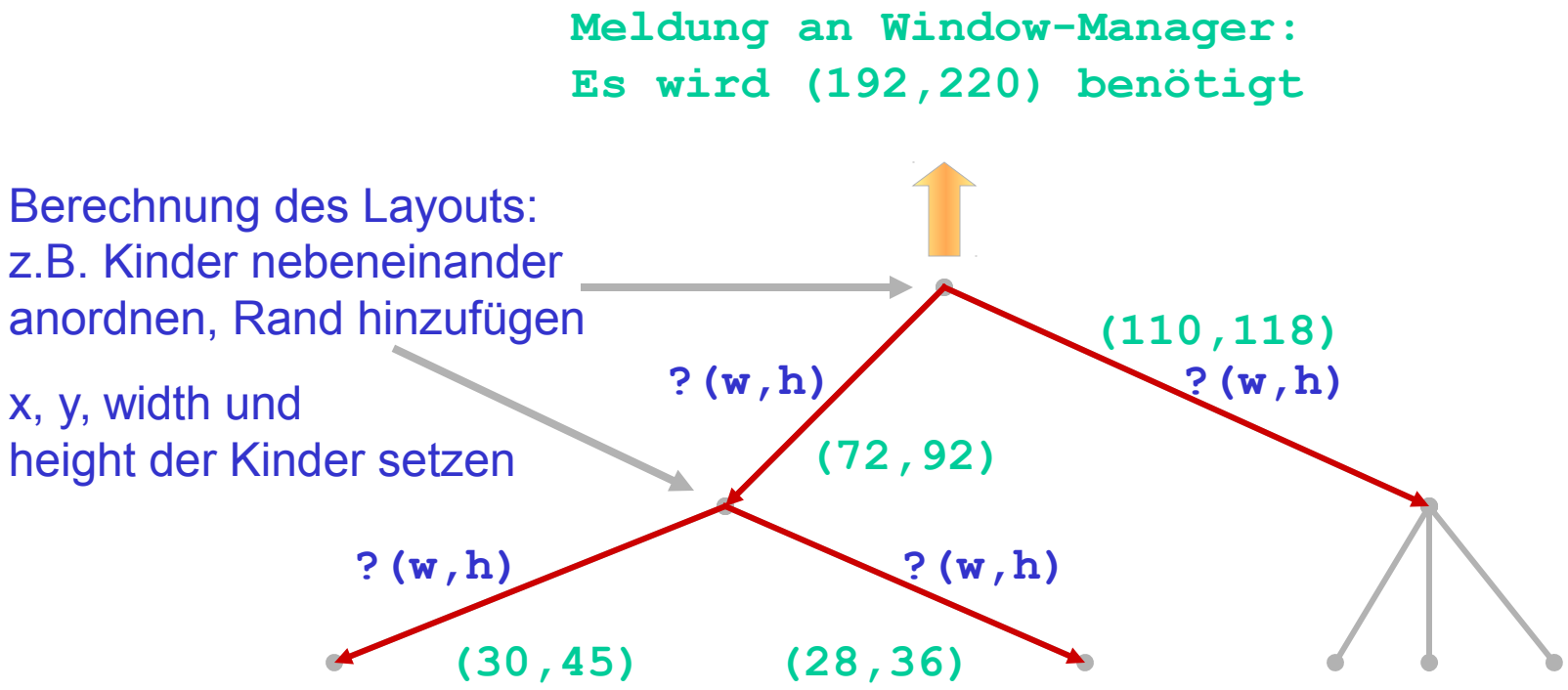
## Ziele des Layout Managements

- automatisierte Berechnung der Geometrien der Kinder eines Containers unter Berücksichtigung
  - der Platzanforderungen der Kinder (die sich aus deren Zustand und Einstellungen ergeben)
  - des insgesamt durch das Toplevel-Window bereitgestellten Platzes

## Layout-Strategie

- Jeder Layout-Manager verfolgt eine Strategie, wie er die Geometrie der Kinder berechnet; diese Strategie muß man kennen und verstanden haben, um den Layout-Manager zu verenden

Grundlage ist ein rekursiver Prozeß, der die Window-Hierarchie durchläuft und in jedem Knoten der Hierarchie für den darunter liegenden Teilbaum die Geometrie berechnet.



## Beispiel: Blatt der Hierarchie

Strategie des Managers

Nimm ein Kind nach dem anderen, ordne sie von links nach rechts an und füge einen äußeren Rahmen der Breite dx und dy hinzu.


Label

Font: Arial 24, Höhe h=24, Breite des Textes „Vorname“ (aus der Font-Metrik) w=112

Textfeld

Font: Arial 24, Höhe h=24, 14 Zeichen Minimum Text, daher Minimum-Breite des Textes (aus der Font-Metrik) 14*10 (größtes Zeichen) w=140 , hinzu kommt ein Rahmen von 2 Pixeln rundherum, daher h=28, w=144

# 5.2 Funktionsweise des Layout-Managements

## Durchführung des Layout-Managements

- wenn eine Window-Hierarchie erstmals dargestellt wird
  - dabei stehen die Anforderungen der Window-Hierachie und eine eventuell gesetzte Größe des Frame i.d.R. im Widerspruch
- wenn der Nutzer die Größe des Toplevel-Windows ändert
- wenn der Nutzer die Größe von spezialisierten Managern ändert (z.B. JSplitPane)

## Setzen/Abfragen des Layout-Managers

- Methoden der Klasse `java.awt.Container`

```
void setLayoutManager ( LayoutManager layout );
LayoutManager getLayoutManager();
```

## Basis-Methoden in `java.awt.Component`

- werden von Layout-Managern aufgerufen

```
Dimension getMinimumSize();
Dimension getMaximumSize();
Dimension getpreferredSize();
```

## Interface `java.awt.LayoutManager`

- spezifiziert die Methoden der Layout-Manager

```
void addLayoutComponent ( String name, Component comp );
void removeLayoutComponent ( Component comp );
Dimension preferredLayoutSize ( Container parent );
Dimension minimumLayoutSize ( Container parent );
void layoutContainer ( Container parent );
```

Beispiele für Layout-Manager in AWT

- FlowLayout
- BorderLayout
- GridLayout

# 5.4.1 Class java.awt.FlowLayout

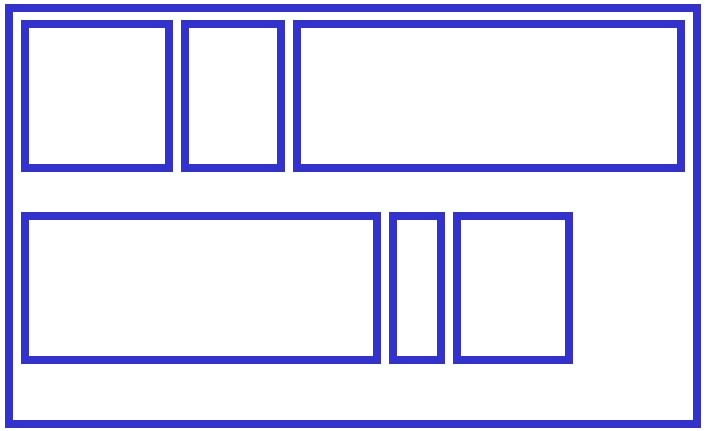
Beschreibung:

A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line.

Klassenkonstante

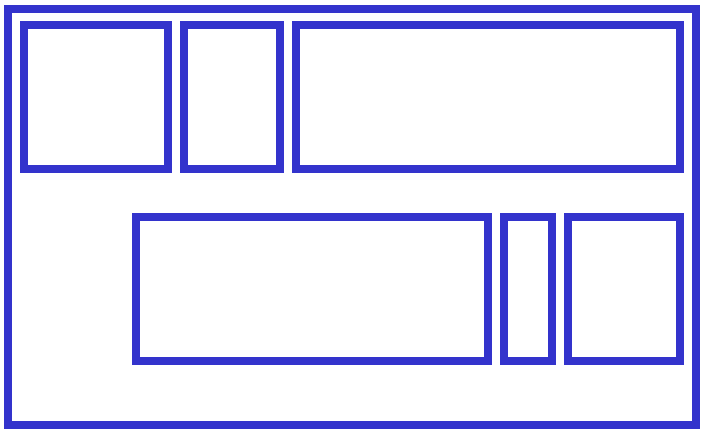
```
FlowLayout.CENTER
FlowLayout.LEFT
FlowLayout.RIGHT
```

LEFT

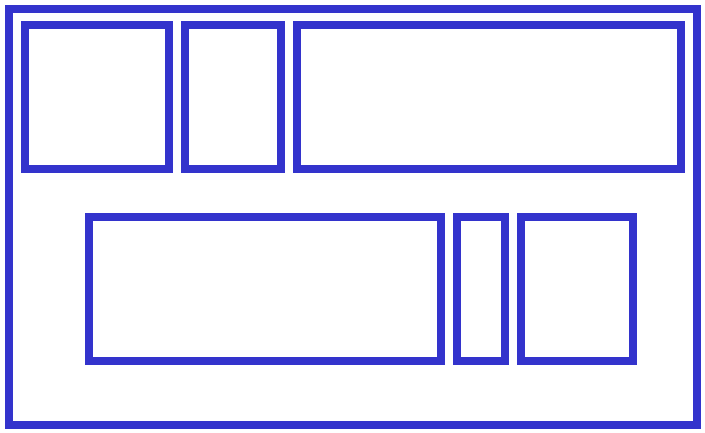


# 5.4.1 Class java.awt.FlowLayout

RIGHT

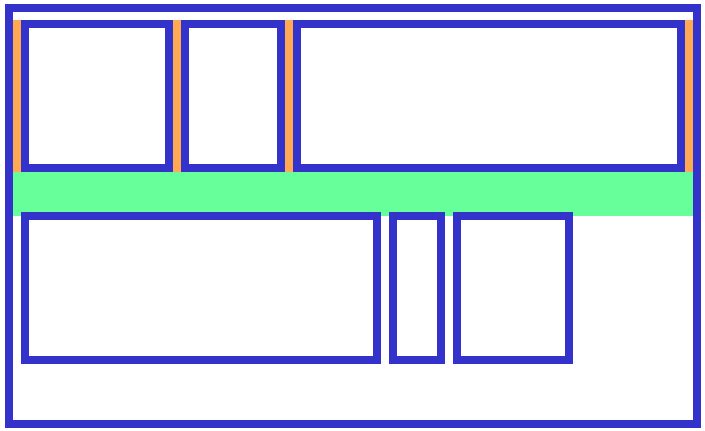


CENTER



Gaps:

- Abstände zwischen den Komponenten
- Hgap: horizontal
- Vgap: vertical



# 5.4.1 Class java.awt.FlowLayout

## Konstruktoren:

```
FlowLayout()                    Alignment=CENTER, hgap=5, vgap=5
FlowLayout(int align)                             hgap=5, vgap=5
FlowLayout(int align, int hgap, int vgap)
```

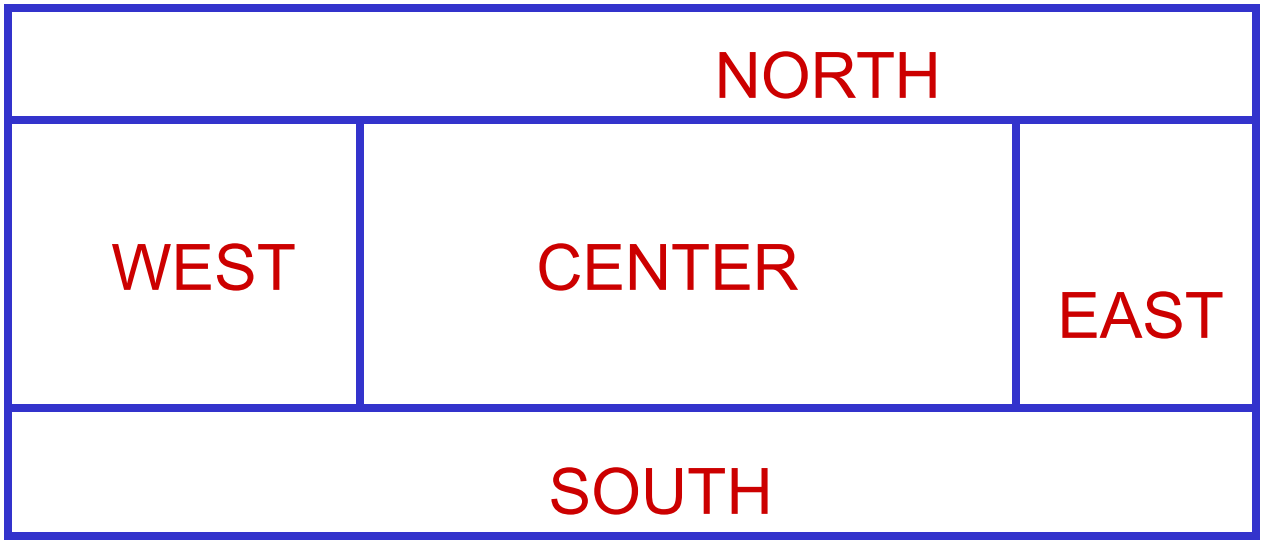
## Methoden:

```
int getAlignment();
int setAlignment();
int getHgap();
int setHgap();
int getVgap();
int setVgap();
```

# 5.4.2 java.awt.BorderLayout

Beschreibung:

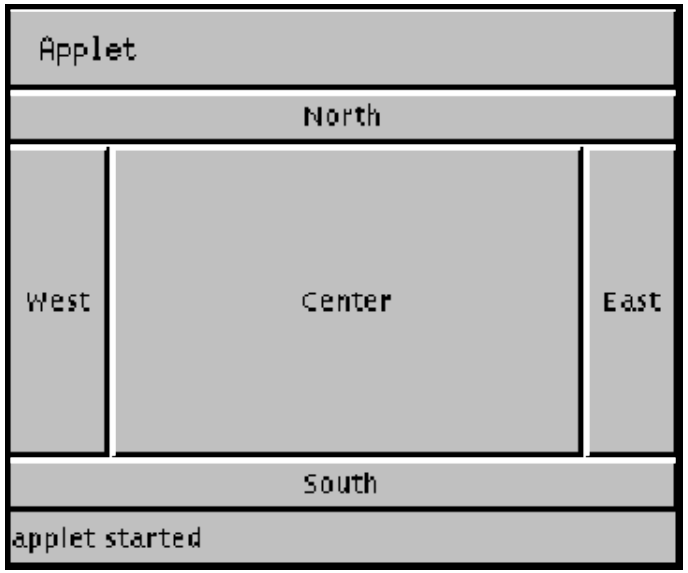
A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.



# 5.4.2 java.awt.BorderLayout

## Klassenkonstante:

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




## Konstruktoren:

```
BorderLayout()                    hgap=5, vgap=5
BorderLayout(int hgap, int vgap)
```

## Methoden:

```
int getHgap();          int setHgap();
int getVgap();          int setVgap();
```

# 5.4.3 java.awt.GridLayout

Beschreibung:

The GridLayout class is a layout manager that lays out a container's components in a rectangular grid.
The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

Konstruktoren:

```
GridLayout()                                    rows=1, hgap=5, vgap=5
GridLayout(int rows, int cols)                        hgap=5, vgap=5
GridLayout(int rows, int cols, int hgap, int vgap)
```

# 5.4.3 java.awt.GridLayout

## Methoden:

```
int setRows();
int setRows();
int setColumns();
int setColumns();
int getHgap();
int setHgap();
int getVgap();
int setVgap();
```