GUI-Programmierung mit Java

PROG 2: Einführung in die Programmierung für Wirtschaftsinformatiker

---

Steffen Helke
Technische Universität Berlin
Fachgebiet Softwaretechnik

15. April 2013

Übersicht

- GUI-Bibliotheken in Java
- GUI-Fenster
- Layout-Manager
- GUI-Komponenten

Teil I der Vorlesung PROG 2

Entwicklung grafischer Schnittstellen

Graphical User Interface

– GUI-Bibliotheken in Java –

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

Software-Bibliotheken zur GUI-Entwicklung

Was sollte unterstützt werden?

- Grundelemente zum Zeichnen
  $\Rightarrow$ Linien, Polygone, Farben, Schriftarten, ...
- Steuerelemente als fertige GUI-Komponenten
  $\Rightarrow$ Fenster, Schaltflächen, Textfelder, ...
- Komponenten zur Behandlung von Ereignissen
  $\Rightarrow$ Eingaben mit Maus, Tastatur, ...

Angebote in Java (Auswahl)

- AWT aus JFC – Abstract Window Toolkit (1996, Sun)
- Swing aus JFC – Java Foundation Classes (1998, Sun)
- SWT – Standard Widget Toolkit aus Eclipse (2001, IBM)

Abstract Window Toolkit (AWT)

Einordnung

- seit 1996 Bestandteil der Java Foundation Classes (JFC)
- Verwendung nativer (schwergewichtiger) Komponenten des Betriebssystems (außerhalb des Java-Speicherbereichs)
- Layout-Probleme beim Wechseln der Plattform möglich

Funktionalitäten

- vergleichsweise wenig und teilweise etwas veraltet
- Primitivoperationen zum Zeichnen (Linien, Farbe, Text, ...)
- Steuerungskomponenten zur Abfrage von Eingabeereignissen
- Dialogelemente zur Kommunikation mit dem Benutzer
- Darstellung/Manipulation von Bitmaps, Ausgabe von Sound

$\Rightarrow$ böse Zungen sagen auch *Annyoing Window Toolkit*

Swing aus JFC (Java Foundation Classes)

Einordnung

- *Vorstellung* (1997, Konferenz *JavaOne*) *mit Swing-Musik*
- seit 1998 Teil der JFC, baut teilweise auf AWT auf
- Verwendung leichtgewichtiger Komponenten unabhängig vom Betriebssystems (gerendert in Java)
- nicht thread-safe

Funktionalitäten

- anpassbares Erscheinungsbild und Verhalten (look and feel)
- mehr Komponenten als AWT: Tabellen, Bäume, ...
- GUI-Steuerung mit Tastenkombinationen möglich
- automatisches Double Buffering, unterstützt MVC-Pattern
- Drag & Drop, Tool-Tips, Multiple Document Interface

Was bedeutet Threadsicherheit in einer GUI?

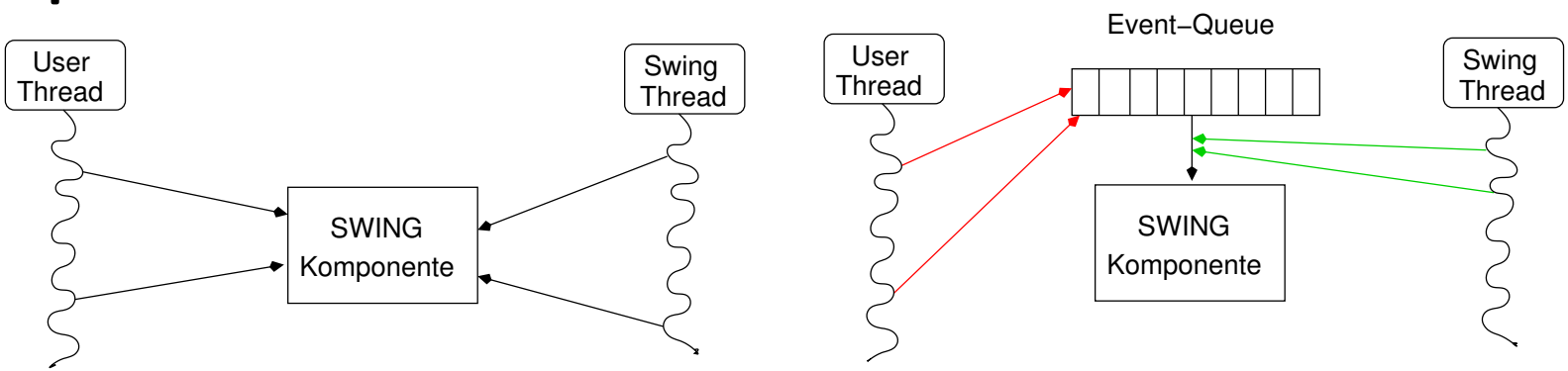
Definition

- Gleichzeitige und mehrfache Ausführung einer GUI-Komponente durch unterschiedlichen Programmbereiche führt zu keinen unerwarteten Effekten

Swing ist ...

- *nicht thread-safe* – unkontrollierte Effekte bei nebenläufiger Ausführung der GUI-Komponenten möglich
  $\Rightarrow$ besondere Vorkehrungen durch Programmierer nötig!

Beispiel



Standard Widget Toolkit (SWT)

Einordnung

- Entwicklung von IBM 2001 für Eclipse
- schwergewichtige, native Komponenten
- Effizienzprobleme auf manchen Nicht-Windows-Plattformen
- SWT-Bibliotheken oft nicht standardmäßig verfügbar
- ähnlich zu Swing nicht thread-safe!

Funktionalitäten

- Basiskomponenten ähnlich zu Swing
- spezielle Bibliothek *JFace*, um Basiskomponenten zu komplexeren GUI-Elementen zusammenzufassen
- Actions zur Entkopplung von GUI-Event und Aktion
- Komplexere GUI-Elemente: Wizards, Dialoge, ...

Was ist eigentlich ein Widget?

Kunstwort

- **Wi**(ndows) und (Ga)**dget**
- Komponente eines grafischen Fenstersystems
- kein eigenständiges Anwendungsprogramm

Widgets auf einem KDE4-Desktop

Bestandteile

1. Fenster (sichtbarer Bereich der Komponente)
2. Ereignisempfänger für Maus/Tastatur (Steuerungselement)
3. (nicht sichtbares) Objekt, zum Speichern/Verändern des Zustands der Komponente

Teil I der Vorlesung PROG 2

Entwicklung grafischer Schnittstellen

Graphical User Interface

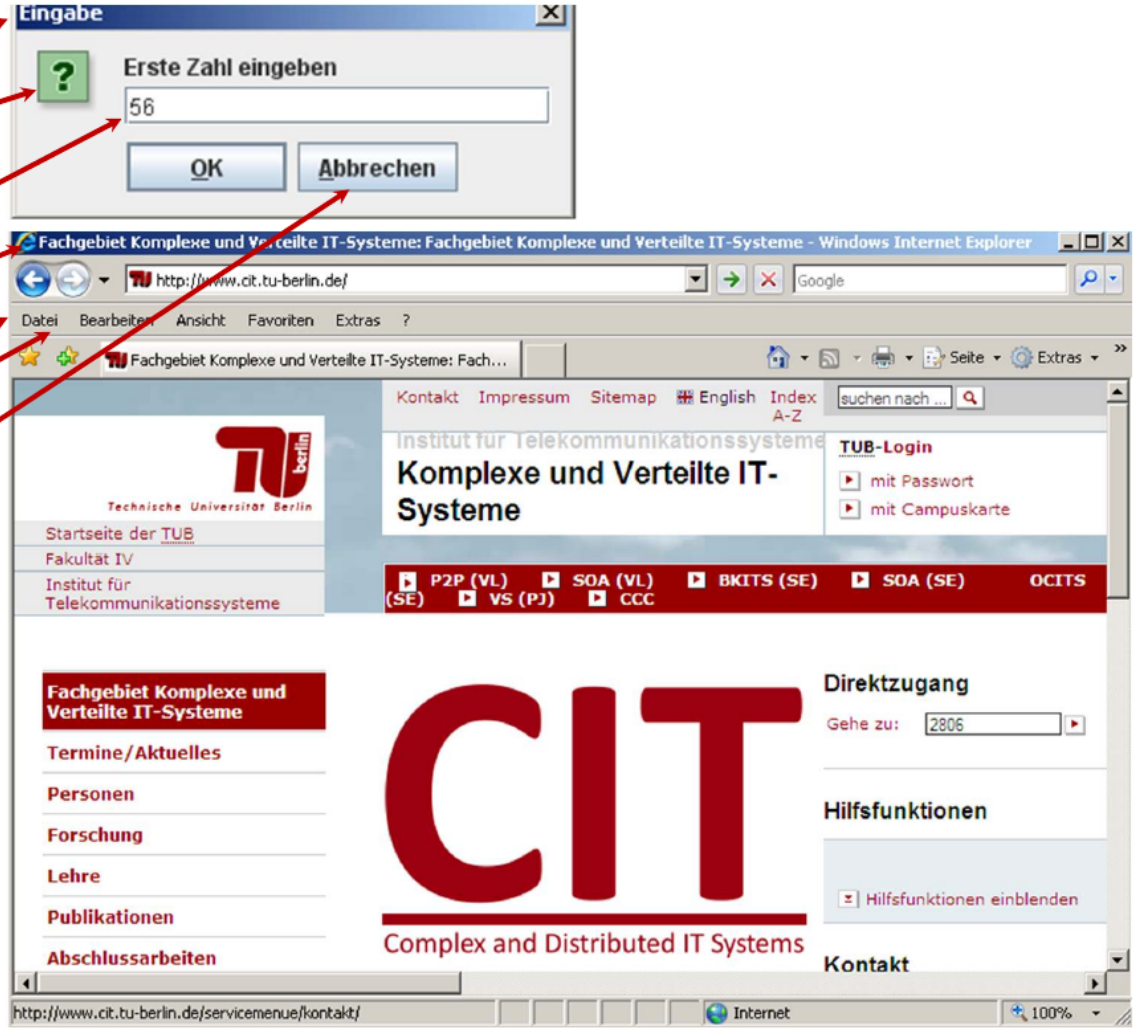
– GUI-Fenster in Java/Swing –

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

Steuerelemente einer grafischen Oberfläche

Oft verwendet

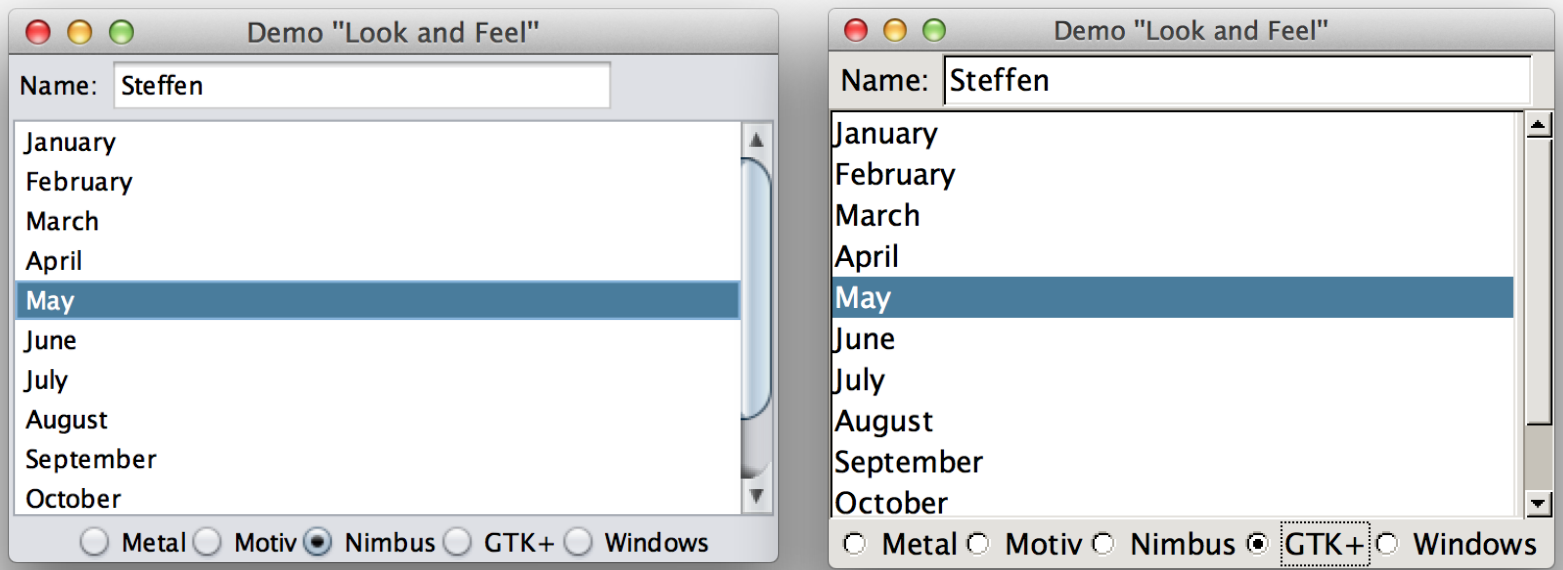
- Labels
- Icons
- Text fields
- Title bar
- Menu bar
- Menus
- Buttons
- Check box
- Lists
- Combo box
- Panel



Quelle: Abbildung aus *Vorlesungsfolien Odej Kao, TU Berlin, MPGI 4, WS 2010/11*

Beispielprogramm: Look and Feel

- Fenster mit Titel, Maximieren, Minimieren, Schließen
- Feld zur Texteingabe
- Auswahlliste mit Scrollbalken und Hilfetext
- Buttons zum Umschalten von *Look and Feel*



GUI-Elemente der Swing-Bibliothek (Auswahl)

- **JLabel:** Darstellung von Text oder Icon, unveränderbar
- **JTextField:** Benutzereingabe von der Tastatur

- **JButton:** Mausklick aktiviert Ereignis
- **JCheckBox:** Auswahl einer oder mehrerer Optionen

- **JList:** Liste mit Auswahlmöglichkeiten
- **JComboBox:** Kombination von Liste & Texteingabe

- **JPanel:** Bereich zur Positionierung/Organisation verschiedener Komponenten, auch Malbereich für Grafiken
- **JFrame:** Grundattribute und Verhalten eines Fensters, d.h. Titelschrift, Buttons für Maximieren/Schließen, Vergrößern/Verkleinern, ...

Systematischer Aufbau einer GUI

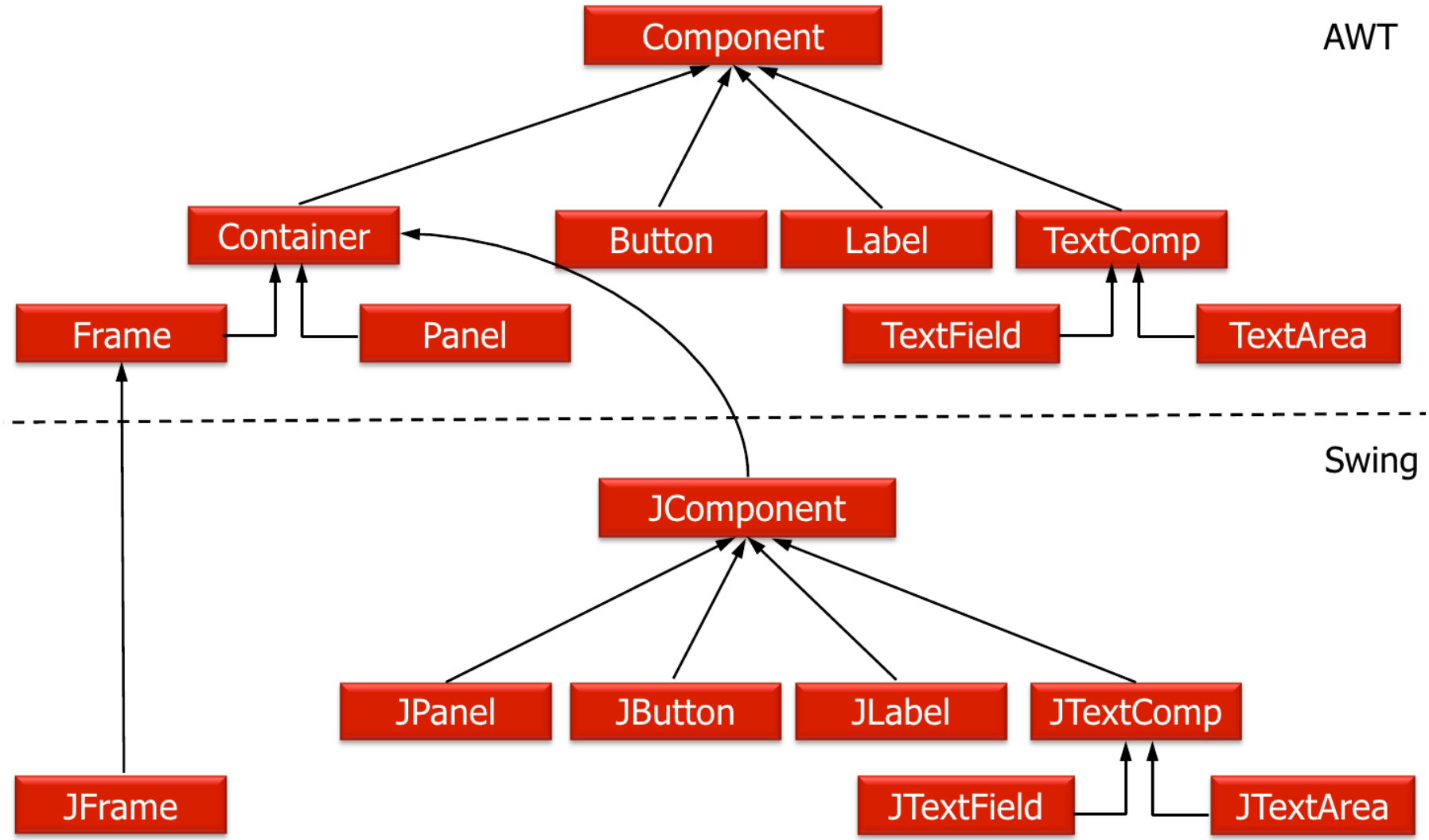
Fensteraufbau in Schichten

1. *Frame*
2. *Layout Manager* für *Frame*
3. *N Container*
4. jeweils ein *Layout Manager* pro *Container*
5. *N* GUI-Elemente pro *Container*

Klassifikation von GUI-Komponenten

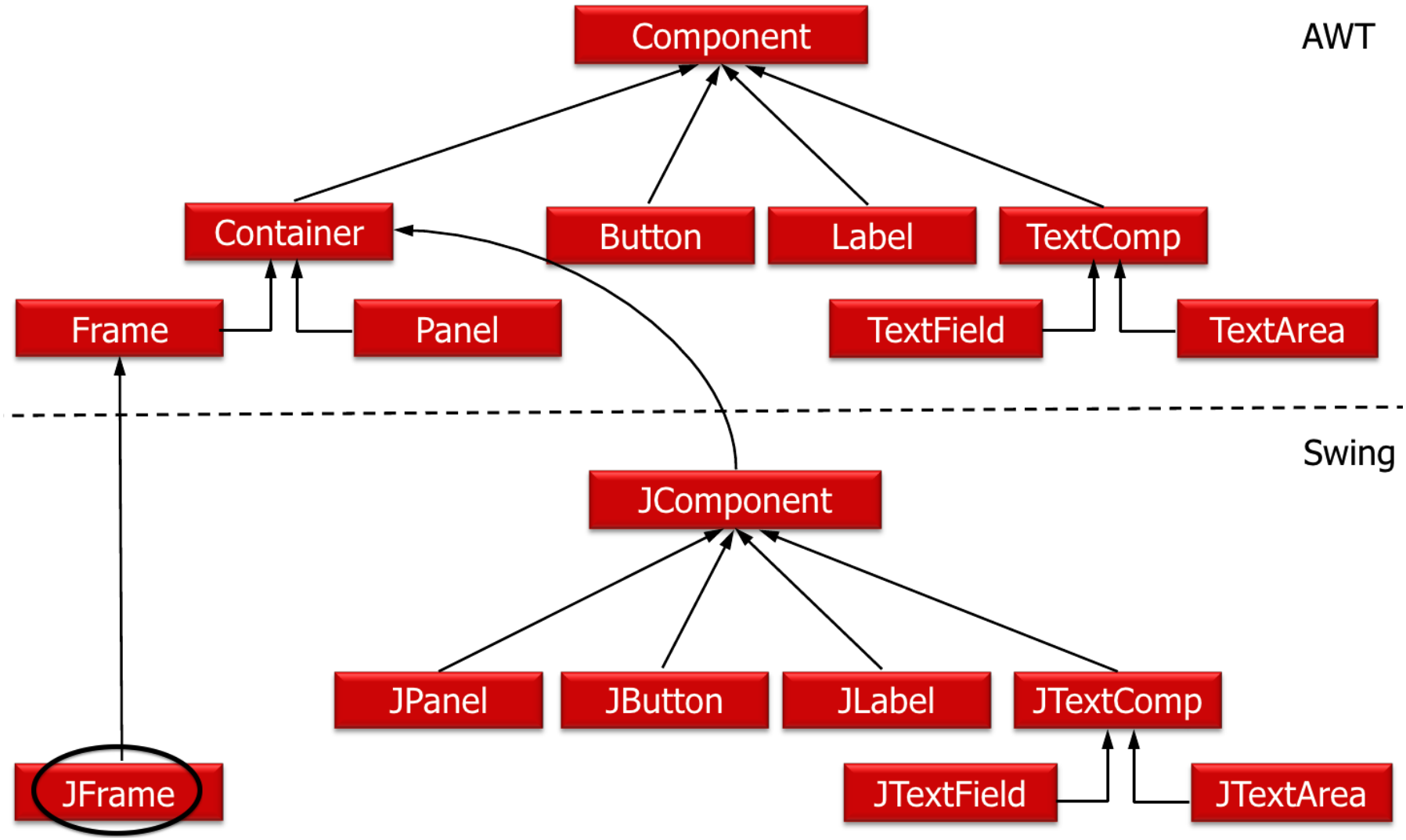
- Container als Behälter für andere Komponenten
- Spezialcontainer (*Frame* bzw. *JFrame*)
- Komponenten (Buttons, TextFields, ...)

Klassenbibliothek: GUI-Elemente (Ausschnitt)



Quelle: Abbildung aus *Vorlesungsfolien Odej Kao, TU Berlin, MPGI 4, WS 2010/11*

Klassenbibliothek: GUI-Elemente (Ausschnitt)



Quelle: Abbildung aus *Vorlesungsfolien Odej Kao, TU Berlin, MPGI 4, WS 2010/11*

Klasse JFrame (javax.swing.JFrame)

Beschreibung

- Fenster mit Rahmen, Systemmenü und Schaltflächen

Konstruktoren

- **public** JFrame()
- **public** JFrame(String title)

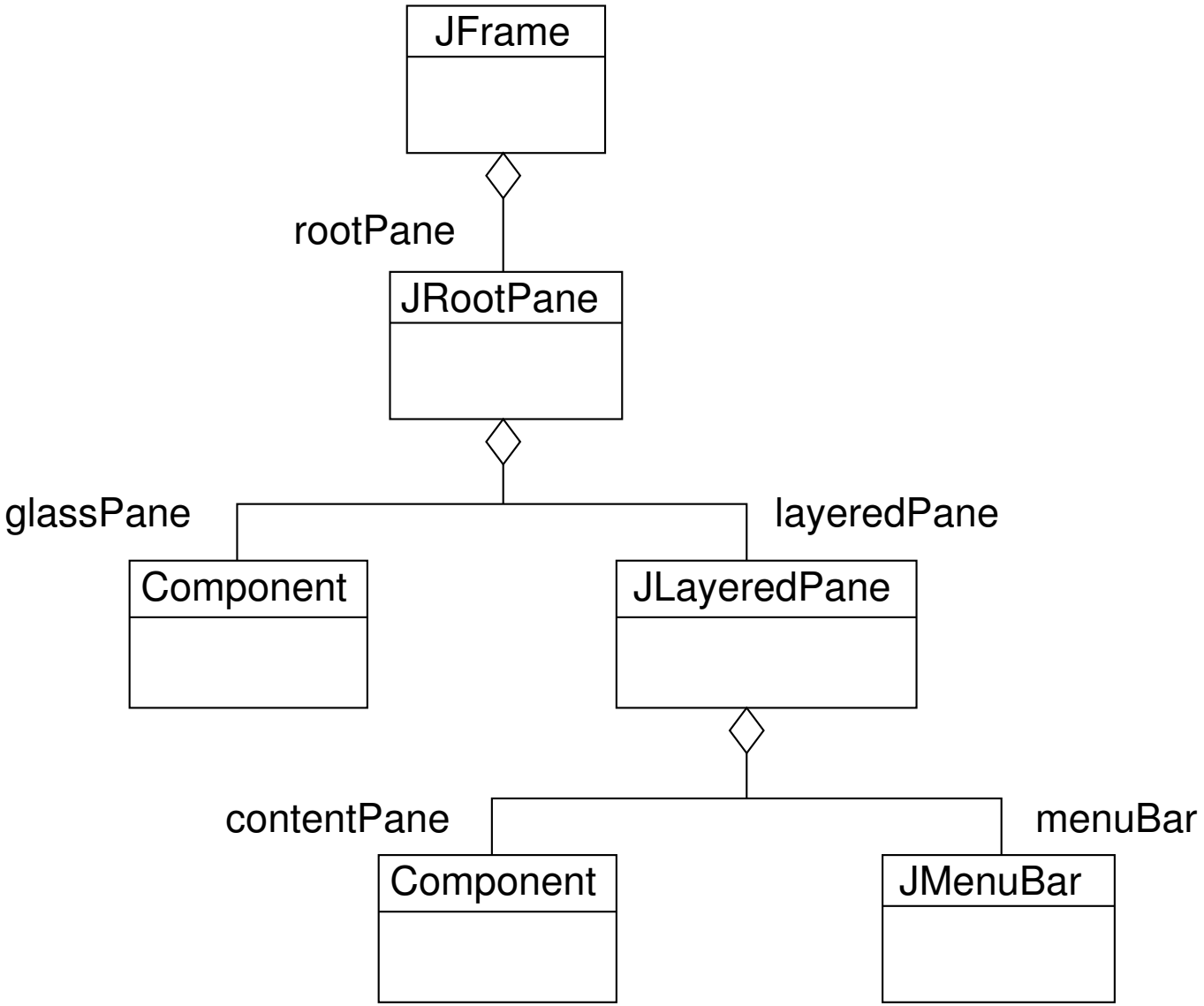
Aufbau eines Fensters auf dem Top-Level

- enthält nur eine Komponente *JRootPane*
- dient als Verwaltungsinstanz (Fenstermanager)

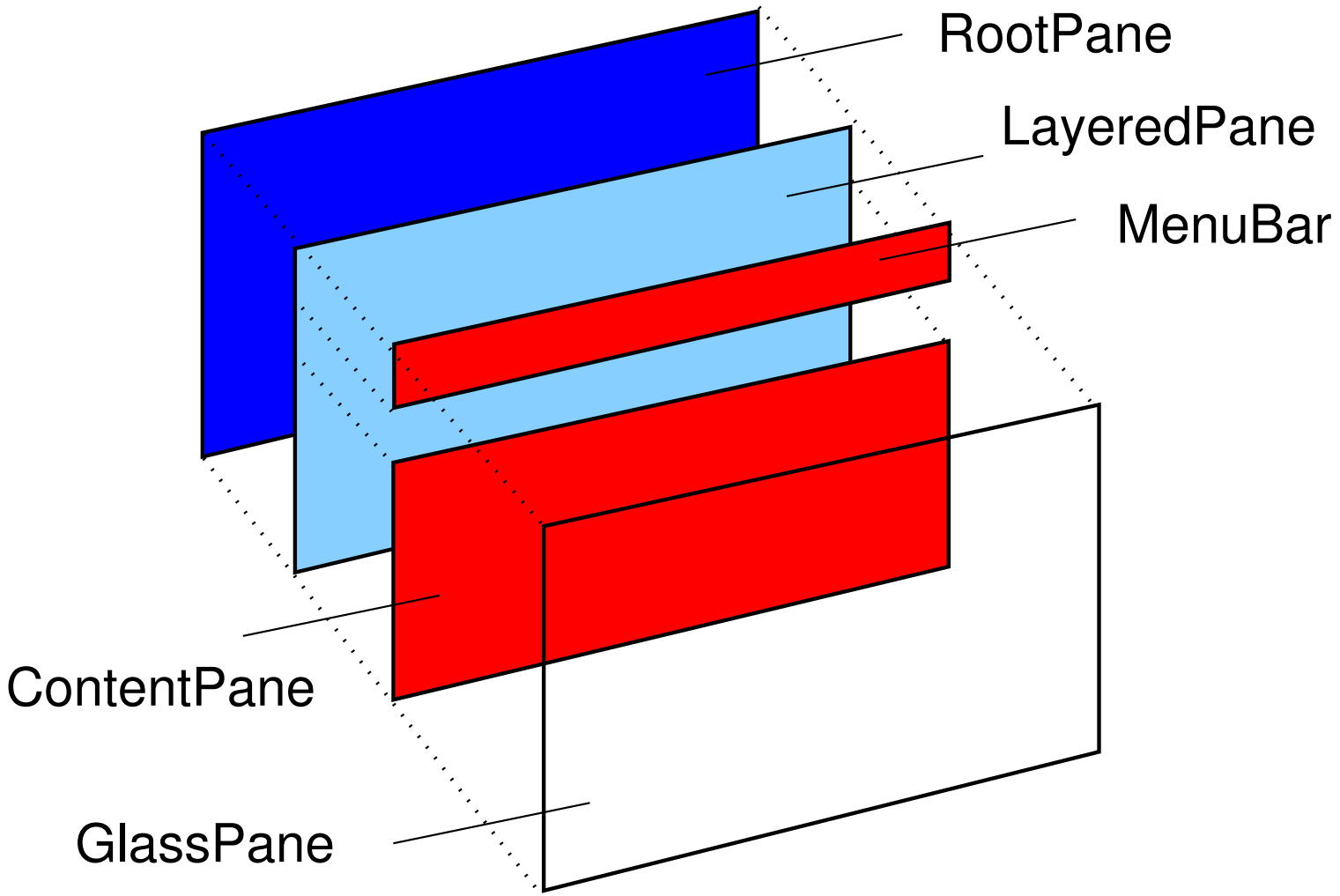
Bestandteile von JRootPane

- *LayeredPane:* bestehend aus Menü (*MenuBar*) und Container für Dialogelemente (*ContentPane*)
- *GlassPane:* über *LayeredPane* platziert und durchsichtig, dient z.B. zum Abfangen des Mauszeigers

Struktur der Klasse JFrame



Komponenten eines JFrame-Objektes



Zugriff auf einzelne JFrame-Komponenten

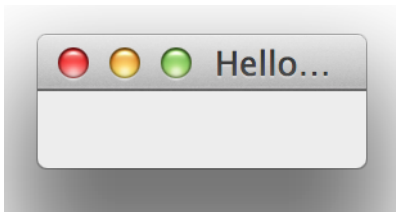
- Objekte für *RootPane*, *GlassPane*, *LayeredPane* und *ContentPane* werden beim Anlegen eines Fensters automatisch erzeugt
- *JFrame* implementiert das Interface *RootPaneContainer*
  - **public** JRootPane getRootPane()
  - **public** Container getContentPane()
  - **public** JLayeredPane getLayeredPane()
  - **public** Component getGlassPane()
- zusätzlich zu get-Methoden sind auch set-Methoden zum Ändern der Elemente verfügbar

JFrame-Fenster: Erzeugen und Einblenden

```
import javax.swing.*;

class HelloWorldSwing {
  public static void main(String[] args) {
  // Erzeuge ein neues Fenster
  JFrame window = new JFrame("Hello World (Swing)");
  // Blende das Fenster ein
  window.setVisible(true);
 }
}
```

Was fehlt noch?

- Fenster ist zu klein
- Es gibt keinen Inhalt
- Festlegen, wie das Fenster terminieren soll

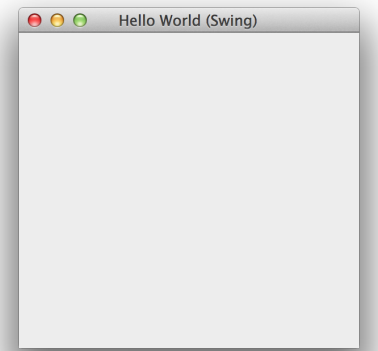


JFrame-Fenster: Setzen der Größe

```
import javax.swing.*;

class HelloWorldSwing {
  public static void main(String[] args) {
  // Erzeuge ein neues Fenster
  JFrame window = new JFrame("Hello World (Swing)");
  // Setze die Fenstergröße fest
  window.setSize(300, 300);
  // Blende das Fenster ein
  window.setVisible(true);
 }
}
```

Was fehlt noch?

- Es gibt keinen Inhalt
- Festlegen, wie das Fenster terminieren soll



JFrame-Fenster: Einfügen von Text

```
import javax.swing.*;

class HelloWorldSwing {
  public static void main(String[] args) {
  // Erzeuge ein neues Fenster
  JFrame window = new JFrame("Hello World (Swing)");
  // Füge Text ein
  JLabel text = new JLabel("Hello World");
  window.getContentPane().add(text);
  // Setze die Fenstergröße fest
  window.setSize(300, 300);
  // Blende das Fenster ein
  window.setVisible(true);
 }
}
```

Was fehlt noch?

- Festlegen, wie das Fenster terminieren soll

JFrame-Fenster: Einfügen von Text

```
import javax.swing.*;

class HelloWorldSwing {
  public static void main(String[] args) {
  // Erzeuge ein neues Fenster
  JFrame window = new JFrame("Hello World (Swing)");
  // Füge Text ein
  JLabel text = new JLabel("Hello World");
  window.add(text);
  // Setze die Fenstergröße fest
  window.setSize(300, 300);
  // Blende das Fenster ein
  window.setVisible(true);
 }
}
```

Was fehlt noch?

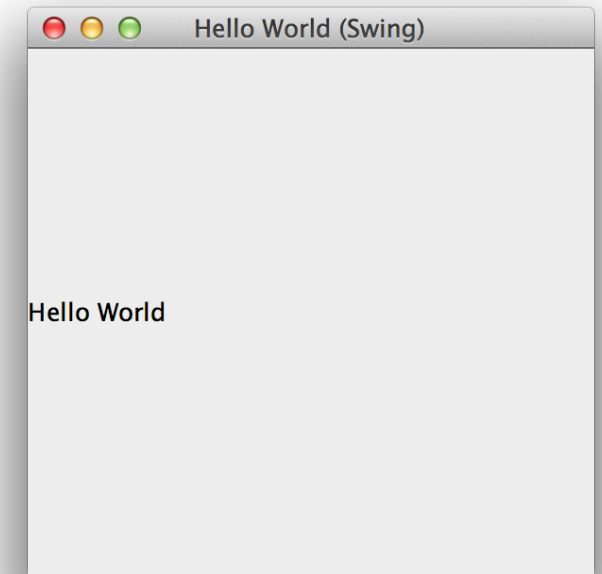
- Festlegen, wie das Fenster terminieren soll

JFrame-Fenster: Terminieren

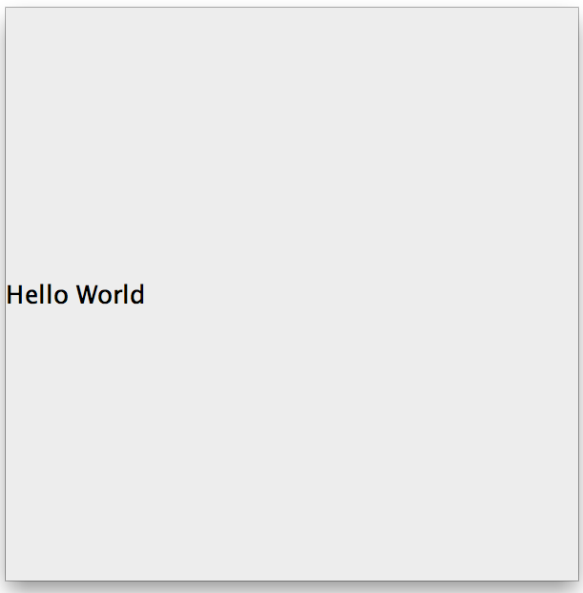
```
import javax.swing.*;

class HelloWorldSwing {
  public static void main(String[] args) {
  // Erzeuge ein neues Fenster
  JFrame window = new JFrame("Hello World (Swing)");
  // Füge Text ein
  JLabel text = new JLabel("Hello World");
  window.getContentPane().add(text);
  // Setze die Fenstergröße fest
  window.setSize(300, 300);
  // Blende das Fenster ein
  window.setVisible(true);
  // Beende, wenn Fenster geschlossen wird
  window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```

Ergebnis: Unser erstes Swing-Fenster!

JFrame-Fenster



JWindow-Fenster



- *JWindow* : Fenster ohne Rahmen, beliebige Stelle und Größe
- geeignet als Willkommensbildschirm für Anwendungen, die nach bestimmter Zeit wieder verschwinden sollen

ContentPane und MenuBar

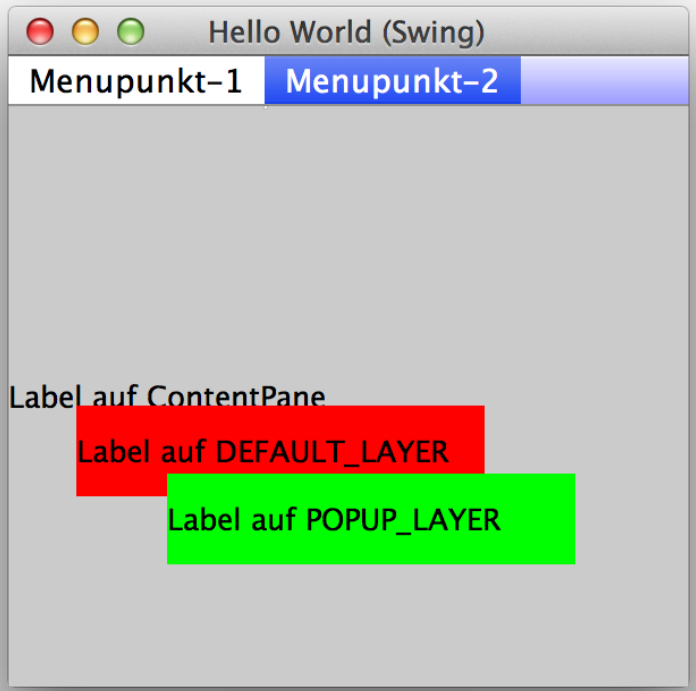
- Bestandteile des *LayeredPane*-Objekts, positioniert auf unterer Schicht
- ein *JMenuBar*-Objekt wird beim Erzeugen eines *JFrame*-Objekts nicht automatisch angelegt


```
JFrame window = new JFrame("...");
...
JMenuBar menu = new JMenuBar();
menu.add(new JMenu("Menupunkt-1"));
menu.add(new JMenu("Menupunkt-2"));
menu.setBackground(Color.BLUE);
window.setJMenuBar(menu);
```

LayeredPane

- ➜ geschichtete Darstellung aller im *LayeredPane*-Objekt verwalteten Komponenten
- ➜ beim Hinzufügen wird Schichttiefe festgelegt

```
JLayeredPane layer = window.getLayeredPane();
layer.add(label1,JLayeredPane.DEFAULT_LAYER);
```



Vordefinierte Schichttiefen

- DEFAULT_LAYER : 0
- MODAL_LAYER : 200
- POPUP_LAYER: 300
- Tiefe für *ContentPane* & *MenuBar*: -3000

Fensterformen als Dialoge in Swing

JDialog

➜ Sperrt aufrufendes Fenster solange, bis Benutzer im neuen Fenster JDialog eine Eingabe tätigt

```
public JDialog(Frame owner, String title,
               boolean modal)
```

- owner: Fenster, das die Eingaben benötigt
- modal: Darf owner-Fenster weiter Eingaben empfangen?

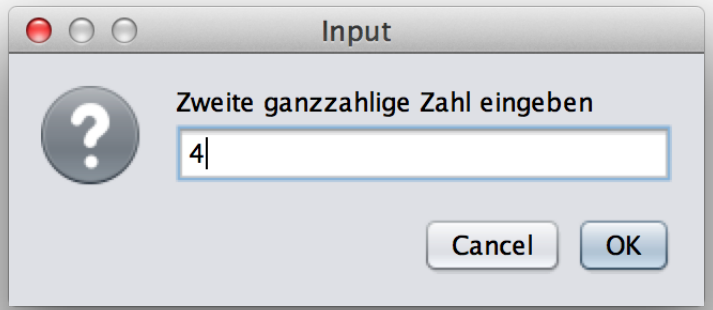
JOptionPane

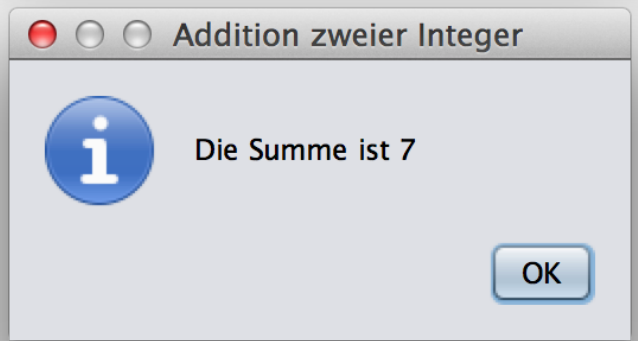
- einfache Swing-Dialoge

Dialog mit JOptionPane

Intention

- Benutzer-Interaktion über Dialog-Boxen (Ein- und Ausgaben)





Dialog mit JOptionPane

Eingabe von Strings

```
public static String showInputDialog(Object message)
```

```
String z = JOptionPane.showInputDialog("Zahl eingeben");
```

Ausgabe mit Benutzerbestätigung

```
public static void showMessageDialog(Component parent,
        Object message,String title,int messageType)
```

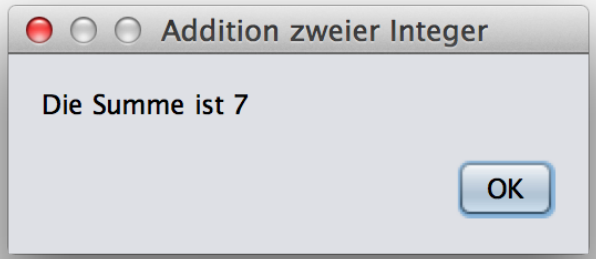
```
JOptionPane.showMessageDialog(null,"Summe ist" + sum,
   "Addition Integer",JOptionPane.INFORMATION_MESSAGE);
```

Alternativen

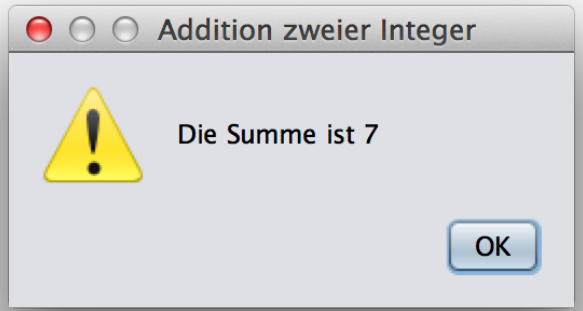
- ERROR_MESSAGE, QUESTION_MESSAGE, WARNING_MESSAGE, PLAIN_MESSAGE

Alternative Ausgaben mit JOptionPane

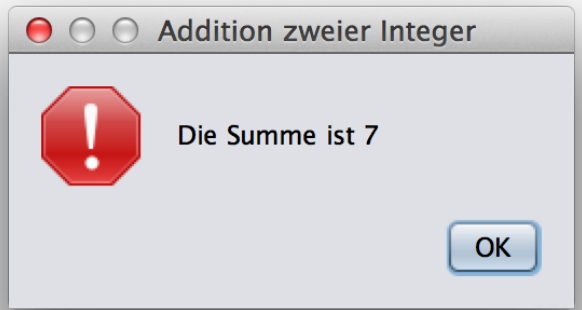
PLAIN_MESSAGE



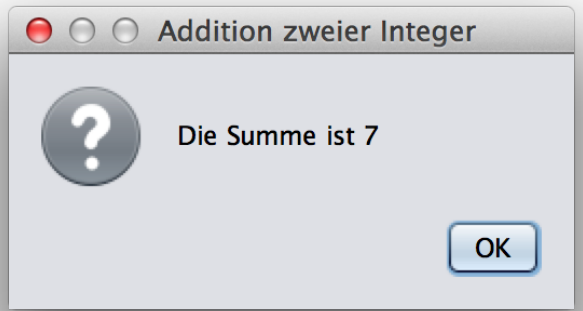
WARNING_MESSAGE



ERROR_MESSAGE



QUESTION_MESSAGE



Konfigurieren von Buttons mit JOptionPane

Erweiterung bei Ausgaben

➜ Darstellung von Buttons, wie *YES, NO, CANCEL* oder *OK*

Umsetzung

```
public static void showConfirmDialog(Component parent,
      Object message,String title,
      int optionType,
      int messageType)
```

```
JOptionPane.showConfirmDialog(null,
      "Ergebnis ok?" + sum,"Addition Integer",
      JOptionPane.YES_NO_CANCEL_OPTION,
      JOptionPane.INFORMATION_MESSAGE)
```

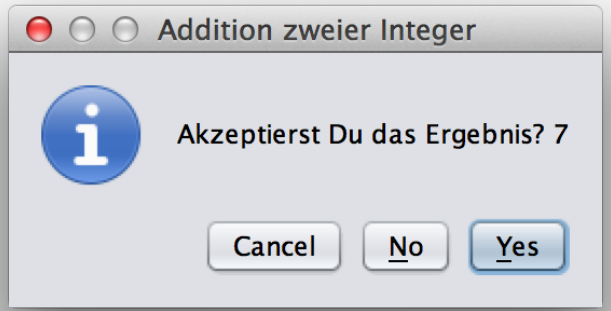
Alternativen

- YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION, CLOSED_OPTION

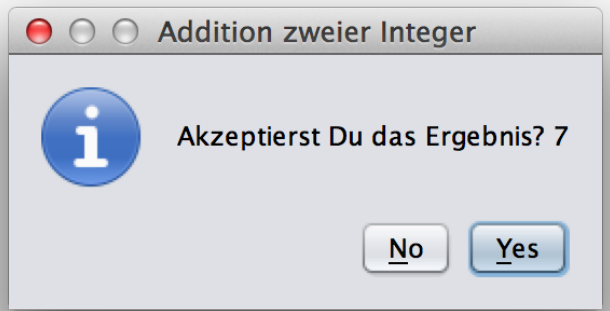
Konfigurieren von Buttons mit JOptionPane

CANCEL_OPTION

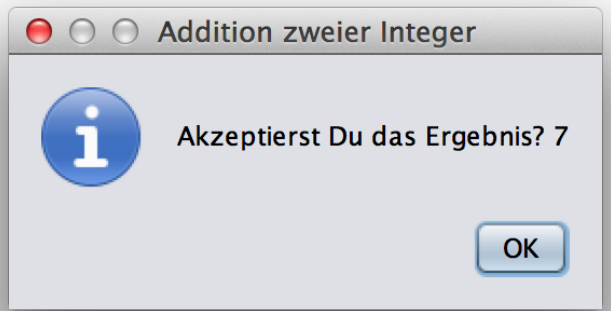
YES_NO_CANCEL_OPTION



YES_OPTION



CLOSED_OPTION



Einschub zu Code-Richtlinien

Referenz

- http://www.oracle.com/technetwork/java/codeconv-138413.html

Formatierung

- Orientierung an Standard-Formatierung von Eclipse
- neuer Block $\Rightarrow$ neue Einrückung

Bezeichner & Kommentare

- sprechende Namen (Abkürzung nur, wenn offensichtlich)
- Kommentare nur sparsam, besser verständlicher Code

Best Practices

- pro Methode nur ein *return*-Statement
- 15 Zeilen Softlimit für Methoden
- 100 Zeilen Softlimit für Klassen
- Verschachtelungstiefe $\geq$ 2-3 (*if, for, while*) vermeiden

Anwendungen mit mehreren Fenstern

Intention

- MDI-Applikationen (Multiple Document Interface)
- Hauptfenster (Desktop) mit Teilfenstern (Kindfenstern)

Desktop

- in der Regel aus *JFrame* abgeleitet
- statt *ContentPane* Verwendung von *JDesktopPane*
- *JDesktopPane* ist Desktop-Manager zum Vergrößern/Verkleinern seiner Kindfenster

Kindfenster

- aus *JInternalFrame* abgeleitet
- leichtgewichtige Komponenten (Nachbildung *echter* Fenster)
- Positionierung mit *add* und Aktivierung mit *setVisible*

Beispiel für MDI-Applikation

Attribute von *JInternalFrame*

- title
- resizable
- closeable
- maximizable
- iconifiable

```
JDesktopPane desktop = new JDesktopPane();
window.setContentPane(desktop);
JInternalFrame child = new JInternalFrame
                              ("Child",true,true,
                                       true,true);

desktop.add(child);
child.setSize(200, 200);
child.setVisible(true);
```

Teil I der Vorlesung PROG 2

Entwicklung grafischer Schnittstellen

Graphical User Interface

– Layout-Manager –

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

Möglichkeiten des Layouts

Intention

- Anordnung von GUI-Komponenten innerhalb eines Fensters
- feste Position für dynamisch veränderliche Fenster ungeeignet

Umsetzung

- automatische Anordnung mit Hilfe von Layout-Managern
- Festlegung mit *setLayout*, Elemente mit *add* hinzufügen

Beispielmanager

- *FlowLayout:* Elemente nebeneinander
- *GridLayout:* Elemente im Gitter
- *BorderLayout:* Elemente in vordefinierten Bereichen
- *CardLayout:* mehrere Unterdialoge im Fenster
- *GridBagLayout:* *GridLayout* um Bedingungsobjekte erweitert

FlowLayout

Einordnung

- einfachster Layout-Manager von AWT
- Anordnung von links nach rechts, von oben nach unten
- Reihenfolge beim Hinzufügen entscheidend
- Verändern der Fenstergröße $\Rightarrow$ neue Anordnung

Umsetzung

```
FlowLayout(int align,int hgap,int vgap)
```

- align: zentriert, links- oder rechtsbündig

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

- hgap: horizontaler Abstand zwischen Komponenten
- vgap: vertikaler Abstand zwischen Komponenten

GridLayout

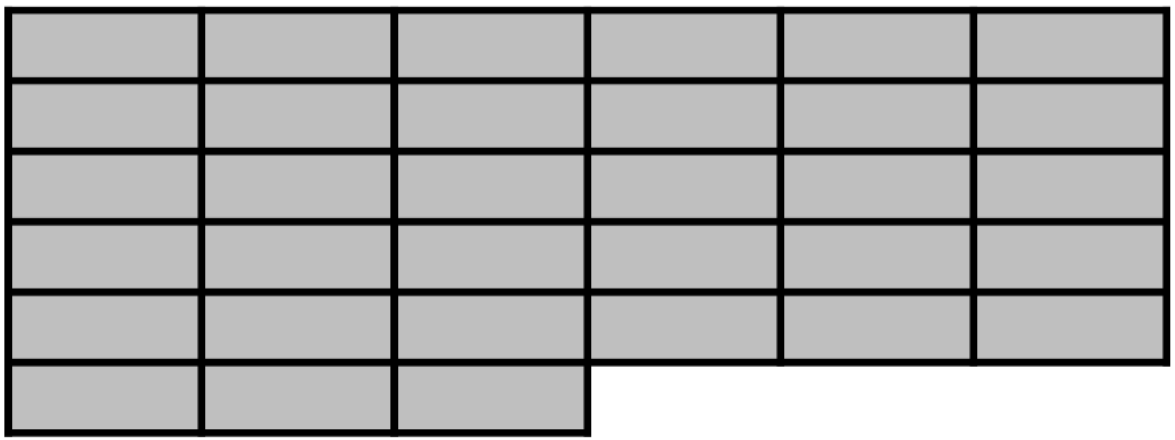
Matrixartige Anordnung

- Konstruktor

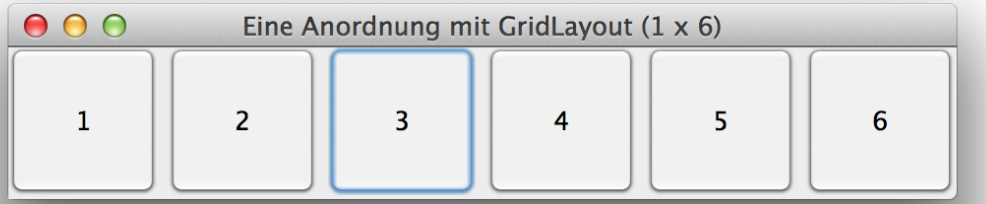
```
new GridLayout(rows, cols)
```

- Elemente zeilenweise hinzufügen
- bei zu wenig Elementen, bleiben Felder leer
- bei zu viel Elementen, wird Spaltenzahl erhöht

  $\Rightarrow$ flexible Handhabung bei $\text{rows} = 0$ oder $\text{cols} = 0$



Beispiel: 6 Buttons im GridLayout




```
public class GridLayoutBsp extends JFrame {
  private JButton buttons[];
  private final String names[] = {"1","2","3","4","5","6" };
  private GridLayout gridLayout;

  public GridLayoutBsp() {
    super( "Eine Anordnung mit GridLayout (1 x 6)" );
    gridLayout = new GridLayout( 1, 6, 5, 5 );
    setLayout( gridLayout );
    buttons = new JButton[ names.length ];
    for (int count = 0;count < names.length; count++) {
            buttons[count] = new JButton(names[count]);
            add(buttons[count]);
    }
}}
```

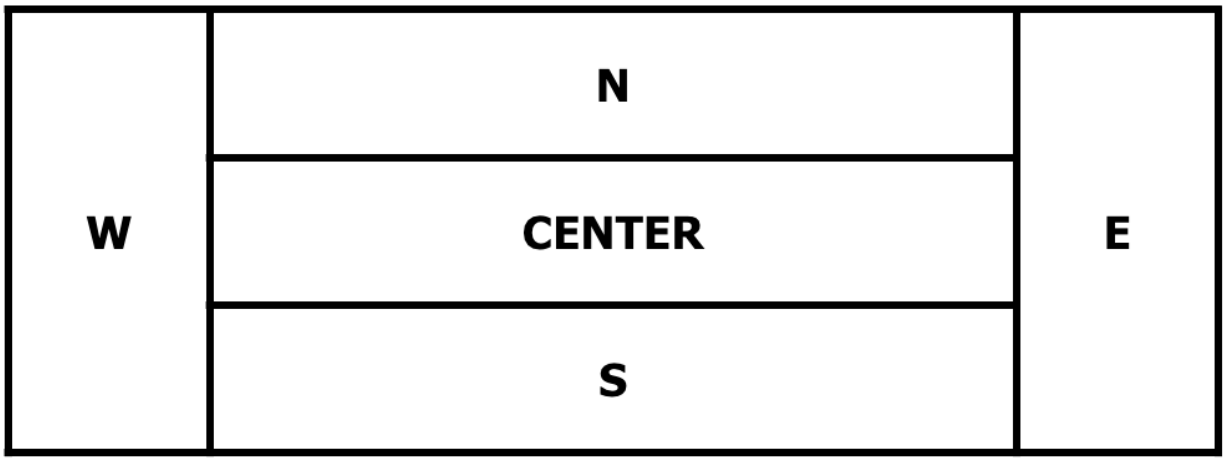
BorderLayout

Anordnung mit 5 definierbaren Bereichen

- *NORTH, WEST, SOUTH, EAST, CENTER*
- Hinzufügen zu einem Bereich mit

```
container.add(element, BorderLayout.WEST)
```

- für leere Bereiche gilt: Höhe oder Breite ist 0
- automatische Breitenberechnung für Randbereiche, der Rest wird dem Zentrum zugeordnet



GritBagLayout

Einordnung

- sehr flexibel, aber auch kompliziert
- Umgang mit Elementen unterschiedlicher Größe
- Reihenfolge des Hinzufügens ist nicht für Layout relevant

Koordinaten zur Anordnung

- rechteckiges Gitter von Zellen
- Zellen können unterschiedlich groß sein
- flexible Platzierung, z.B. können GUI-Elemente über mehrere Zellen gehen

Syntax

```
layout = new GridBagLayout();
```

```
constraints = new GridBagConstraints();
```

Constraints im GritBagLayout

- In welcher Spalte liegt der linke Rand eines GUI-Elementes?

```
int gridx/gridy
```

- Über wie viele Zeilen erstreckt sich das Element?

```
int gridwidth/gridheight
```

- An welcher Kante der Zelle wird das Dialogelement befestigt?

```
int anchor
```

- Was passiert bei Größenveränderung?

```
int fill
```

- Wie ist die Mindestgröße definiert?

```
int ipadx/ipady
```

- Layout-Schachtelung: z.B. nur eine Zelle mit GridLayout

$\Rightarrow$ **Umsetzung beliebig komplexer Layouts möglich!**

Weitere Layoutmanager

AWT

- *CardLayout:* Kartenstapel, bei dem nur das oberste Element sichtbar ist

Swing

- *BoxLayout:* Boxen werden hintereinander dargestellt und auf gleicher Höhe ausgerichtet
- *OverlayLayout:* Elemente werden übereinander positioniert
- *JTabbedPane:* Elemente sind durch Reiter identifizierbar, Element mit aktiven Reiter wird dargestellt
- *GroupLayout:* Vertikales und horizontales Layout werden unabhängig behandelt

Teil I der Vorlesung PROG 2

Entwicklung grafischer Schnittstellen

Graphical User Interface

– Komponenten in einer GUI –

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

Ausgabe von Text mit Labeln in einer GUI

Features

- Setzen von Text, Position, Verbindung mit Icons
- Tooltips (Hilfeanzeige, wenn Mauszeiger überm Label ist)

Beispiel

```
lab = new JLabel(); // Label erzeugen
lab.setText("Darzustellender Text");
lab.setIcon(BspIcon);
lab.setHorizontalTextPosition(SwingConstants.CENTER);
lab.setVerticalTextPosition(SwingConstants.BOTTOM);
lab.setToolTipText("Label bei Mousezeiger");
```

Ein- und Ausgabe von Text mit Textfeldern

Features

- realisiert durch *JTextField*
- kann editierbar oder schreibgeschützt gesetzt werden
- Betätigen der *ENTER*-Taste löst Ereignis aus
- *JPasswordField* ist Erweiterung für Passwörter (Zeichen werden versteckt)

Beispiel

```
t = new JTextField("Unveränderlicher Text",21);
// false = unveränderbar
t.setEditable(false);
```

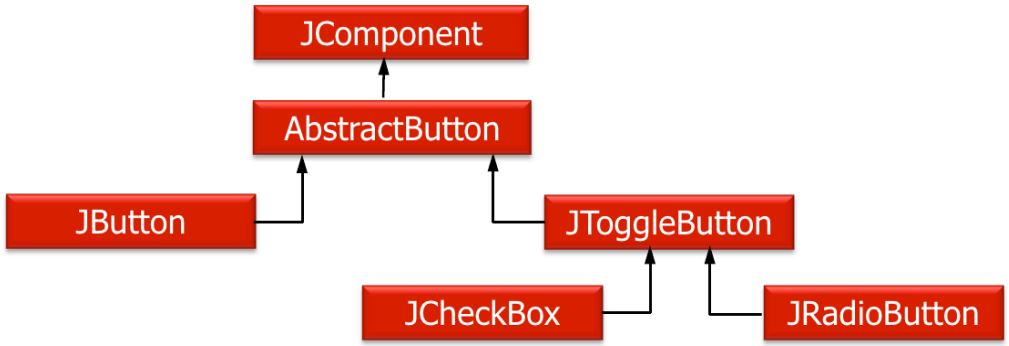
Buttons in einer GUI

Klassischer Button

- mit *JButton*

Wechselnde Button

- Zustand bleibt (*CheckBox*)
- Deaktiviert aktuellen Zustand (*RadioButton*)



Beispiel

```
private JButton button1, button2
button1 = new JButton("Nur Text als Label");
button2 = new JButton("Beispiel mit Icon",bspIcon);
```

CheckButtons und RadioButtons

CheckButton

➜ Erzeugung mit *JCheckBox* analog zu *JButton*

RadioButton

➜ zweistufige Erzeugung

1. Erstellung einzelner Elemente

```
private JRadioButton button1, button2;
button1 = new JRadioButton("Bold",false);
button2 = new JRadioButton("BoldItalic",true);
```

2. Zusammenfassung zu logischer Gruppe

```
private ButtonGroup radioGroup;
radioGroup = new ButtonGroup();
radioGroup.add(button1);
radioGroup.add(button2);
```