

# Objektorientierte Programmietechnik

## Kapitel 14 – Oberflächenprogrammierung

1. Grundbegriffe der Programmierung

2. Einfache Beispielprogramme

3. Datentypen und Variablen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

6. Blöcke und Methoden

7. Klassen und Objekte

8. Vererbung und Polymorphie

9. Pakete

10. Ausnahmebehandlung

11. Schnittstellen (Interfaces)

12. Ein-/Ausgabe und Streams

13. Applets

14. Oberflächenprogrammierung

Inhalte

✓ Architekturmerkmale Swing und AWT

GUI – Container

Layout Manager

Ereignisbehandlung

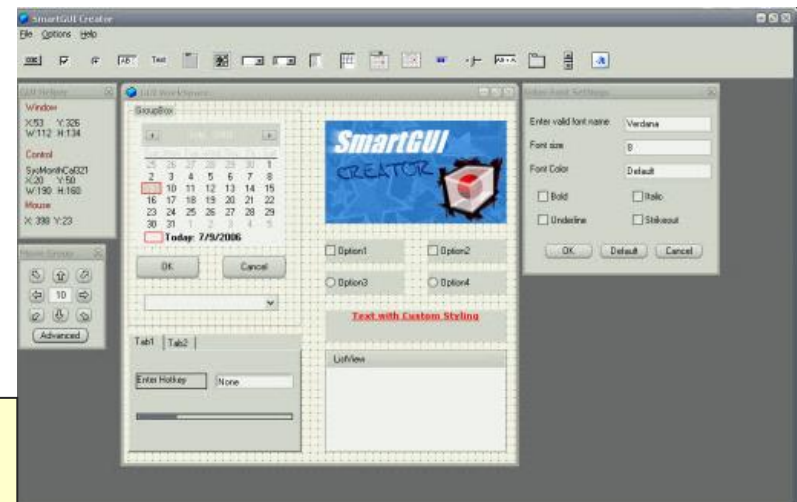
Swing Komponenten

Sonstiges

### ➤ Wichtigkeit

- Die grafische Bedienoberfläche (GUI\*) ist die zentrale Komponente der Mensch-Maschine-Interaktion.
- Das einzige, was der Anwender i.d.R. zu Gesicht bekommt, ist die GUI.
- An ihr bilden sich die Workflows ab.
- Funktion und Intuitivität bei der Bedienung sind häufig entscheidend bei der Kaufentscheidung bzw. Bewertung einer Software.
- Bei der Entwicklung von Individualsoftware sitzen Auftraggeber und –nehmer in gemeinsamen Meetings häufig vor GUI-Skizzen oder Dummies, um Wünsche und Abläufe zu diskutieren.

GUI-Dummies z.B. mit  
:: SmartGUI Creator ::



\* Graphical User Interface

### ➤ Swing und AWT

- Swing ist eine Klassenbibliothek für die Programmierung von grafischen Bedienoberflächen unter Java.
- Folgende Klassenbibliotheken werden für den Einsatz von Swing benötigt:
  - AWT (Abstract Window Toolkit) mit dem Paket `java.awt`
  - Swing mit dem Paket `javax.swing`

### ➤ Zu Beginn von Java gab es noch kein Swing, es kam nur das AWT zum Einsatz.

- AWT stellt GUI-Komponenten zur Verfügung, welche auf Bibliotheken des Betriebssystems zurückgreifen.
  - Das Zeichnen einer GUI-Komponente (z.B. eine Schaltfläche `java.awt.button`) wird durch das zugrundeliegende Betriebssystem und nicht durch die JVM durchgeführt.
  - Aussehen und Verhalten von AWT-GUI-Komponenten sind damit abhängig vom jeweiligen Betriebssystem.

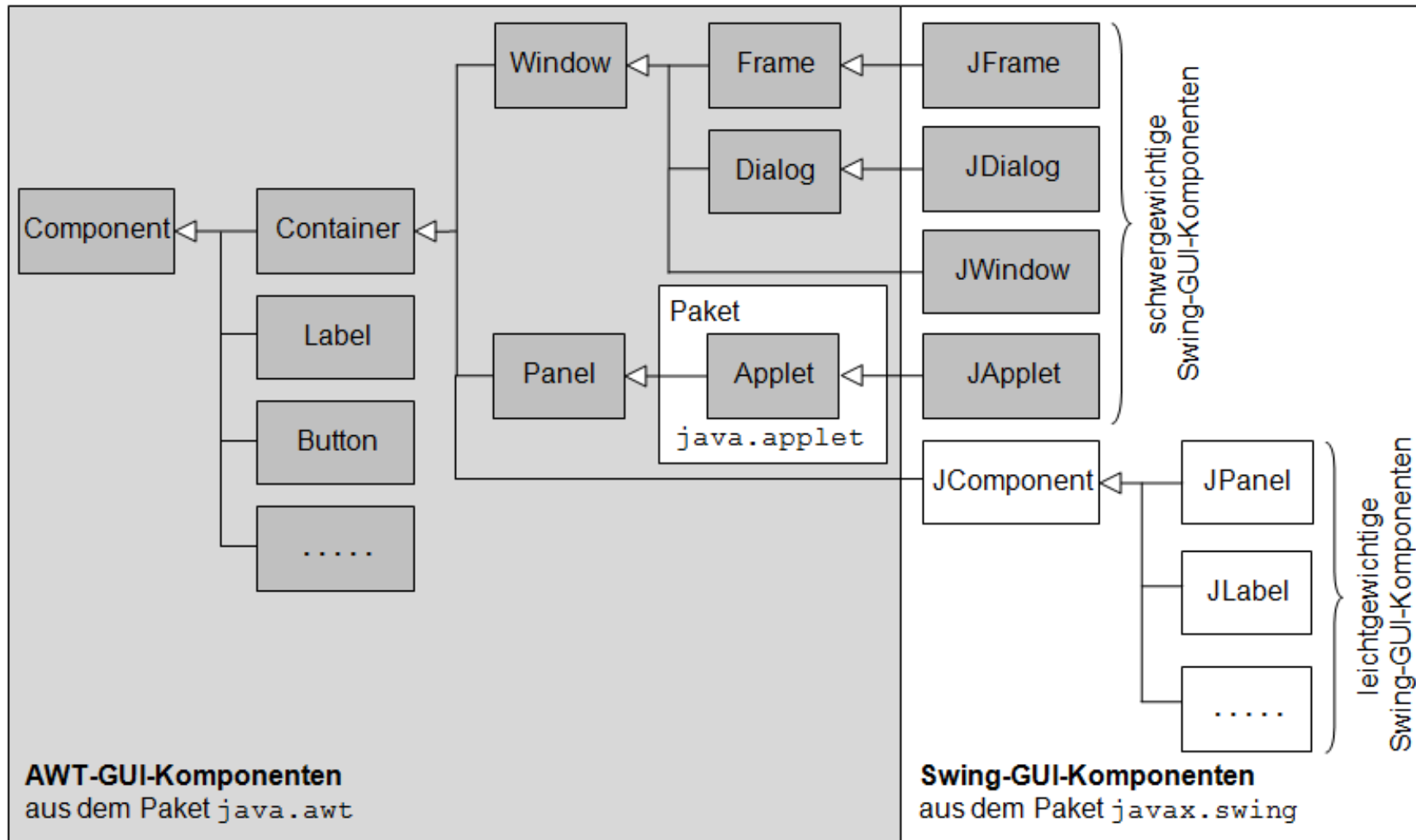
### ➤ Swing und AWT

- AWT-GUIs laufen nur problemlos auf unterschiedlichen Plattformen, wenn nur GUI-Komponenten verwendet werden, die auf allen unterstützten Betriebssystemen zur Verfügung stehen.
  - AWT-GUI-Komponenten werden aufgrund ihrer Abhängigkeit vom Betriebssystem als **schwergewichtig** bezeichnet.
- 

### ➤ Swing führte mit dem JDK 1.2 **leichtgewichtige** GUI-Komponenten ein

- Leichtgewichtige Swing-GUI-Komponenten werden mit Hilfe der Java 2D-Klassenbibliothek durch die JVM selbst auf dem Bildschirm gezeichnet.
- Das Aussehen und Verhalten ist somit unabhängig vom Betriebssystem.
- Ausnahme:  
Die Klassen `JFrame`, `JDialog`, `JWindow` und `JApplet` sind schwergewichtige Swing-GUI-Komponenten (also abhängig vom Betriebssystem)

## ➤ Swing und AWT



### Aufgabe 14.01 Die erste GUI

Implementieren Sie die folgende Klasse in Eclipse und führen Sie die Anwendung aus.

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.FlowLayout;
public class Aufg_15_01{
    public static void main (String[] args){
        // Hauptfenster erzeugen
        JFrame frame = new JFrame ("Hauptfenster");
        // Label erzeugen
        JLabel label1 = new JLabel ("Hallo Welt!");
        // Layout-Manager setzen
        frame.setLayout (new FlowLayout());
        // Label dem Hauptfenster hinzufuegen.
        frame.add (label1);
        // Größe des Fensters
        frame.setSize (400, 100);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        // sichtbar machen
        frame.setVisible (true);
    }
}
```

1. Grundbegriffe der Programmierung

2. Einfache Beispielprogramme

3. Datentypen und Variablen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

6. Blöcke und Methoden

7. Klassen und Objekte

8. Vererbung und Polymorphie

9. Pakete

10. Ausnahmebehandlung

11. Schnittstellen (Interfaces)

12. Ein-/Ausgabe und Streams

13. Applets

14. Oberflächenprogrammierung

Inhalte

✓ Architekturmerkmale Swing und AWT

✓ GUI – Container

Layout Manager

Ereignisbehandlung

Swing Komponenten

Sonstiges

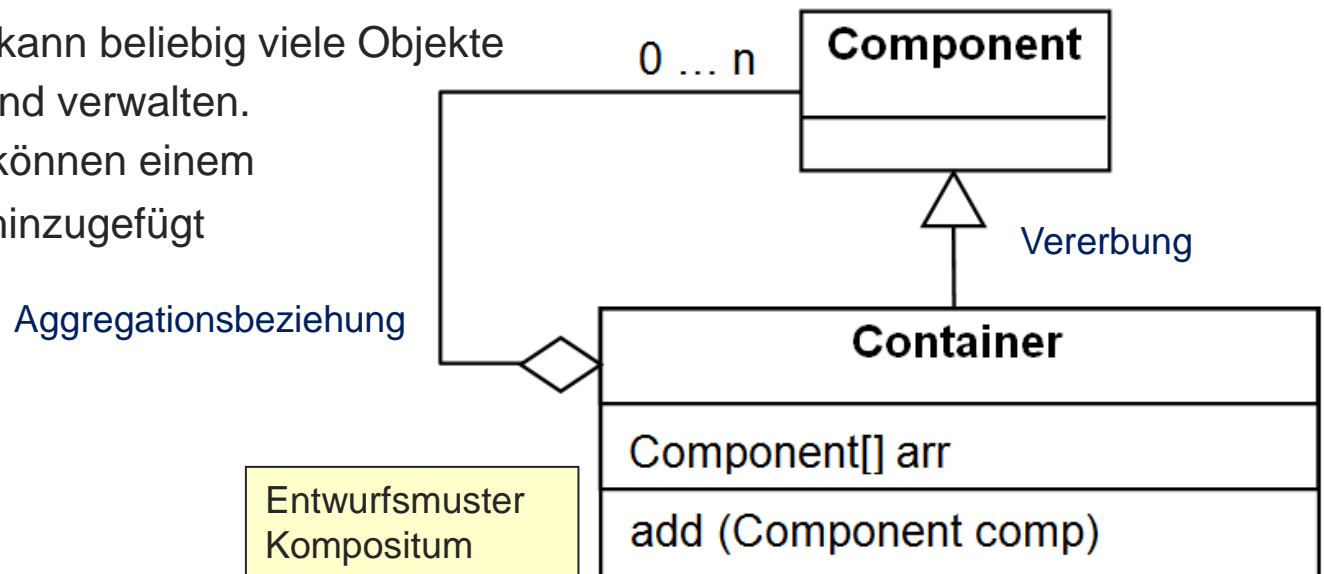


## ➤ GUI - Container

- Ein GUI-Container kann als ein Zeichenbereich auf dem Bildschirm verstanden werden.
- Ein GUI-Container ist eine GUI-Komponente, der andere GUI-Komponenten hinzugefügt werden können
- GUI-Container können dabei auch ineinander verschachtelt werden.

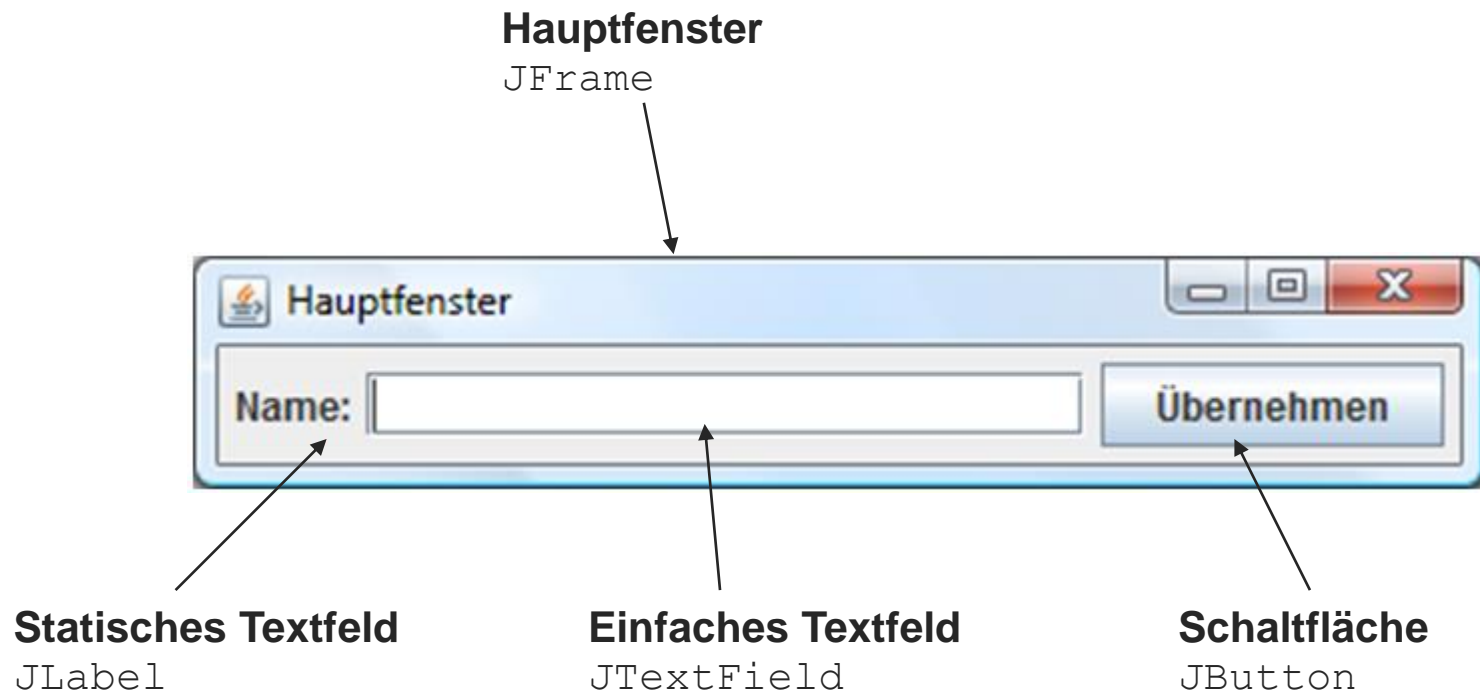
➤ Eigenschaften der Klasse `Container`

- Alle Swing-GUI-Komponenten leiten von der Klasse `Container` ab.
- Ein Objekt der Klasse `Container` kann beliebig viele Objekte vom Typ `Component` aufnehmen und verwalten.
- Durch Aufruf der Methode `add()` können einem GUI-Container GUI-Komponenten hinzugefügt werden.



➤ Beispiel für das Entwurfsmuster Kompositum

- Einem Hauptfenster werden drei weitere Swing-GUI-Komponenten hinzugefügt:



➤ Beispiel für das Entwurfsmuster Kompositum

- Einem Hauptfenster werden drei weitere Swing-GUI-Komponenten hinzugefügt:

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

public class KompositumTest{
    public static void main (String[] args){
        // Für JFrame wird die Container-Funktionalität genutzt.
        JFrame frame = new JFrame ("Hauptfenster");

        JLabel label = new JLabel ("Name:");
        JTextField textfield = new JTextField (20);
        JButton button = new JButton ("uebernehmen");

        frame.setLayout (new FlowLayout());

        // Dem Hauptfenster werden durch Aufruf der Methode add() die
        // anderen GUI-Komponenten hinzugefügt.
        frame.add (label);
        frame.add (textfield);
        frame.add (button);

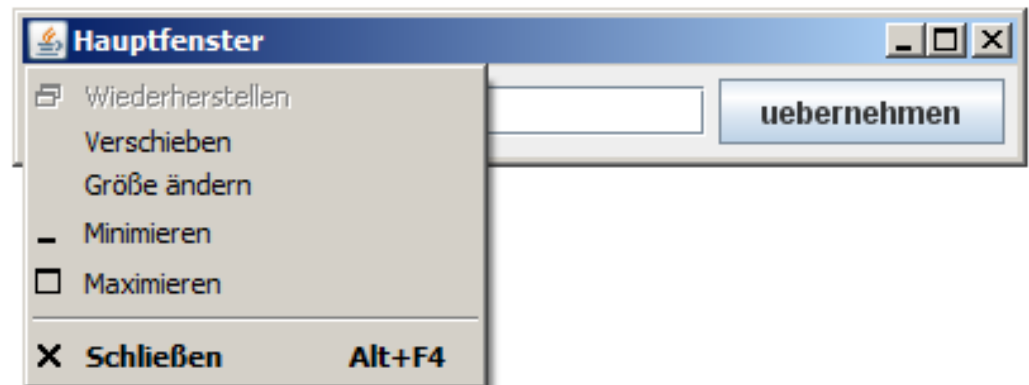
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible (true);
    }
}
```

➤ Die Klassen `JFrame`, `JDialog`, `JWindow` (und `JApplet`) sind die schwergewichtigen Swing-GUI-Container.

- Sie werden auch als Hauptfenster oder Top-Level-Container bezeichnet.
- Sie umhüllen die grafische Bedienoberfläche der Anwendung.
- Ein Hauptfenster kann als schwergewichtiger Top-Level-Container kein anderes Hauptfenster in sich aufnehmen

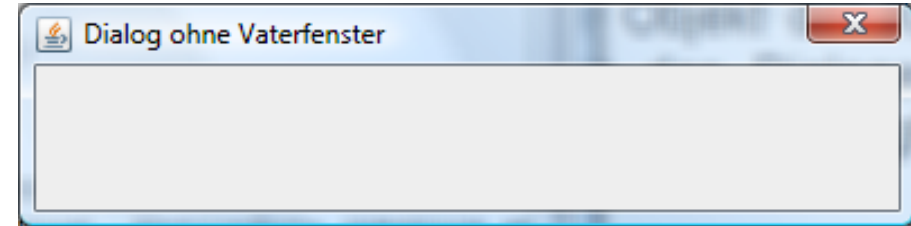
➤ Hauptfenster

- `JFrame`
  - besitzen einen Rahmen und eine Titelleiste
  - optionale Schaltflächen zum Schließen, Minimieren und Maximieren des Fensters
  - Systemmenü



➤ Hauptfenster

➤ JDialog



- besitzen einen Rahmen und eine Titelleiste
- im Gegensatz zu JFrame besitzt ein JDialog keine Schaltflächen zum Minimieren und Maximieren des Fensters

```
import javax.swing.*;

public class DialogTest
{
    public static void main (String[] args)
    {
        JDialog dialog = new JDialog();
        dialog.setTitle ("Dialog ohne Vaterfenster");
        dialog.setDefaultCloseOperation (JFrame.DISPOSE_ON_CLOSE);
        dialog.setSize (400, 100);
        dialog.setVisible (true);
    }
}
```

### ➤ Hauptfenster

### ➤ JWindow

- besitzt keinen Rahmen und somit auch keine Titelleiste, kein Systemmenü und keine Schaltflächen zum Minimieren, Maximieren und Schließen des Fensters
- Mit einem `JWindow`-Hauptfenster kann ein Anwender nicht interagieren (weder Verschieben noch Größe ändern)
- z.B. eingesetzt als Startfenster (Splashscreen)

```
import javax.swing.*;

public class JWindowTest
{
    public static void main (String[] args)
    {
        JWindow window = new JWindow();
        window.setSize (410, 150);
        window.setVisible (true);
    }
}
```

### ➤ Leichtgewichtige GUI-Container

- Ein leichtgewichtiger GUI-Container kann keinen schwergewichtigen in sich aufnehmen.
- Ein leichtgewichtiger GUI-Container kann beliebige leichtgewichtige Swing-GUI-Komponenten in sich aufnehmen.
  - beliebig tiefe Verschachtelung ist möglich

### ➤ JPanel

- `JPanel` ist unsichtbar, für ihn wird nichts auf dem Bildschirm gezeichnet
- Es werden nur die dem `JPanel` hinzugefügten GUI-Komponenten gezeichnet
- Im Prinzip ist ein `JPanel` ein unsichtbares Gruppierungsobjekt `JWindow`

## Leichtgewichtige GUI – Container JPanel

### ➤ JPanel

```
import javax.swing.*;
public class PanelTest
{
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Hauptfenster");
        // 1. leichtgewichtigen GUI-Container erzeugen und befüllen.
        JPanel panel = new JPanel();
        panel.add (new JLabel ("Name:"));
        panel.add (new JTextField (30));

        // 2. leichtgewichtigen GUI-Container erzeugen und befüllen.
        //...

        // GUI-Container und -Komponenten in Hauptfenster legen.
        frame.add (panel);

        frame.pack();
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setVisible (true);
    }
}
```



## Leichtgewichtige GUI – Container und Hilfsklassen I

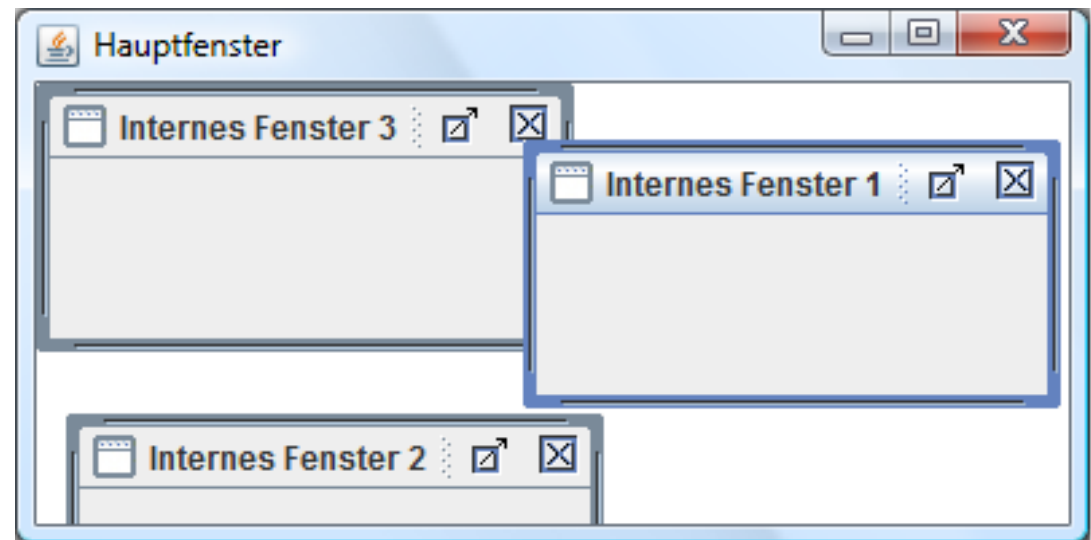
### ➤ Leichtgewichtige GUI-Container und Hilfsklassen

- Im Gegensatz zu `JPanel` kann ein Objekt vom Typ `JInternalFrame` als ein sichtbares Gruppierungsobjekt für GUI-Komponenten bezeichnet werden.

### ➤ JInternalFrame

- bedient sich der Hilfsklasse `JDesktopPane` (abgeleitet von `JLayeredPane`)
- `JDesktopPane` stellt eine virtuelle Desktop-Fläche zur Verfügung.
- Eine virtuelle Desktop-Fläche kann eine beliebige Anzahl von internen Fenstern des Typs `JInternalFrame` verwalten.

Demo in Eclipse:  
`InternalFrameText.java`



## Leichtgewichtige GUI – Container und Hilfsklassen II

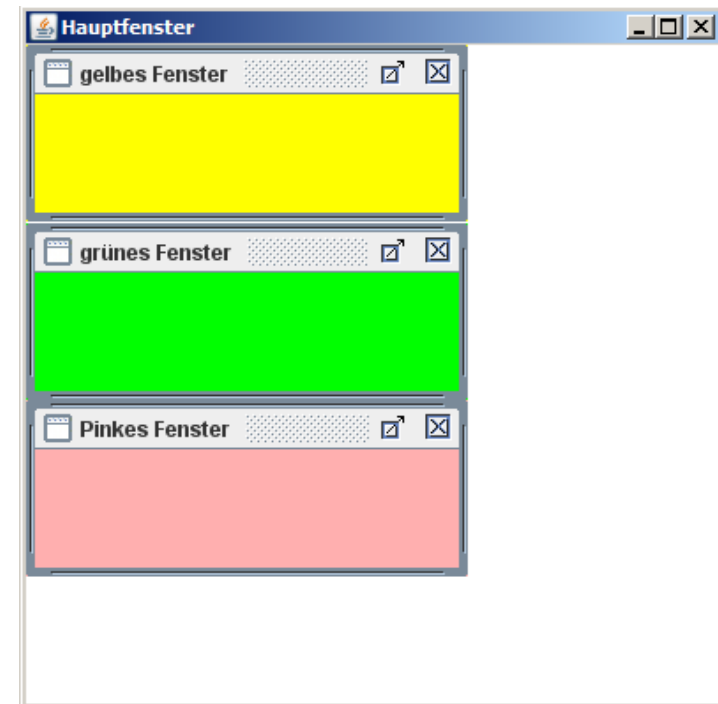
### ➤ InternalFrameTest.java

```
import javax.swing.*;
public class InternalFrameTest{
    public static void main (String[] args){
        JFrame frame = new JFrame ("Hauptfenster");
        // Virtuelle Desktop-Flaeche erzeugen.
        JDesktopPane desktop = new JDesktopPane();

        // Drei interne Fenster erzeugen und der virtuellen Desktopflaeche hinzufuegen.
        for (int i = 1; i <= 3; i++)
        {
            // Die uebergabeparameter vom Typ boolean geben an, dass das
            // interne Fenster resized, geschlossen und maximiert werden kann.
            JInternalFrame internal = new JInternalFrame
                ("Internes Fenster " + i, true, true, true);
            internal.setSize (200, 100);
            internal.setVisible (true);
            desktop.add (internal);
        }
        // Dem Hauptfenster die virtuelle Desktop-Flaeche hinzufuegen.
        frame.add (desktop);
        frame.setSize (400, 200);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setVisible (true);
    }
}
```

### Aufgabe 14.02 Die ersten Fenster

- ✓ Entwickeln Sie eine Klasse für ein Hauptfenster (`JFrame`) mit einer virtuellen Desktopfläche (`JDesktopPane`). Auf der virtuellen Desktop-Fläche sollen drei interne Fenster (`JInternalFrame`) angezeigt werden. Diese internen Fenster besitzen verschiedene Hintergrundfarben und werden initial nicht überlappend auf dem Desktop angezeigt. Recherchieren Sie die erforderlichen Methoden zum Setzen der Hintergrundfarbe (`backgroundcolor`) und zum Setzen der Position (`location`) in der API.
- ✓ Das Hauptfenster soll zentral auf dem Bildschirm dargestellt werden. Recherchieren Sie hierfür nach einer entsprechenden Funktion in der API.



### ➤ Java-Projekte exportieren

- Die zu einem Projekt gehörenden Klassen werden in ein Java-Archiv (JAR) exportiert (vergleichbar mit ZIP-Datei).
- JAR-Dateien können einfach verteilt, weitergegeben oder selbst in anderen Projekten als Klassenbibliothek eingebunden werden.
- JAR-Dateien können als ausführbar deklariert werden.
- Befehl zum Ausführen einer Jar-Datei (main-Klasse wird aufgerufen)

```
java -jar test.jar
```

Demo in Eclipse

#### **Aufgabe:**

Exportieren Sie Ihr Projekt aus Aufgabe 14.02 als ausführbare JAR-Datei.  
Starten Sie Ihr Programm per Doppelklick.

1. Grundbegriffe der Programmierung

2. Einfache Beispielprogramme

3. Datentypen und Variablen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

6. Blöcke und Methoden

7. Klassen und Objekte

8. Vererbung und Polymorphie

9. Pakete

10. Ausnahmebehandlung

11. Schnittstellen (Interfaces)

12. Ein-/Ausgabe und Streams

13. Applets

14. Oberflächenprogrammierung

Inhalte

✓ Architekturmerkmale Swing und AWT

✓ GUI – Container

✓ Layout Manager

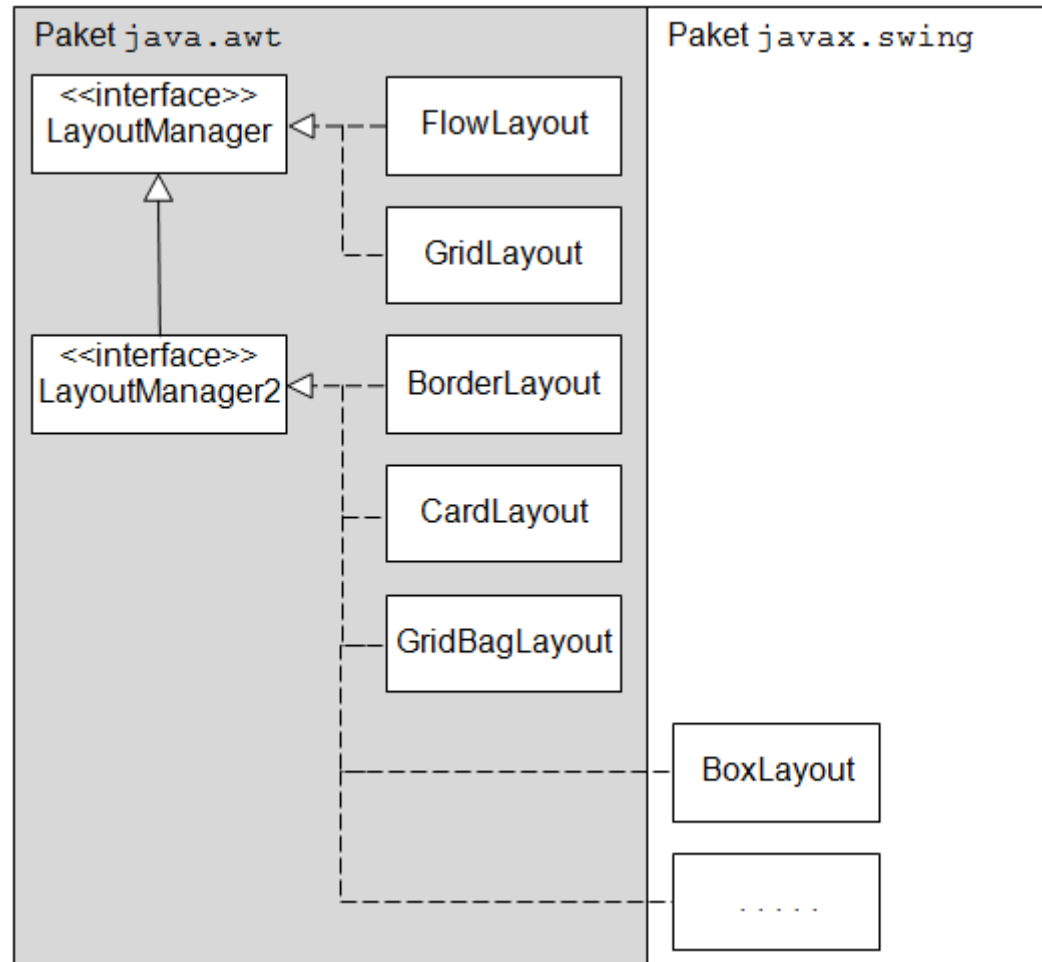
Ereignisbehandlung

Swing Komponenten

Sonstiges

- Layout-Manager
- Ein Layout-Manager legt die Anordnung von hinzugefügten GUI-Komponenten fest.

- Er definiert das Verhalten von GUI-Komponenten in Positionierung und Größe (z.B. bei Vergrößerung oder Verkleinerung des umschließenden Fensters).
- Bei der Oberflächenprogrammierung mit Swing-GUI-Komponenten können sowohl Swing- als auch AWT-Layout-Manager eingesetzt werden.
- Mittels der `setLayout()`-Methode kann man einen GUI-Container mit einem bestimmten Layout versehen. Die Methode erwartet als Übergabeparameter-Typ `LayoutManager`.



## Layout-Manager – Flow-Layout I

### ➤ Layout-Manager – Flow-Layout

- Das Flow-Layout positioniert hinzugefügte GUI-Komponenten in einer Zeile.
- Ist eine Zeile voll, so wird die GUI-Komponente in einer neuen Zeile positioniert.

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

public class FlowLayoutTest{
    public static void main (String[] args){
        JFrame frame = new JFrame ("Layout-Manager: Flow-Layout");

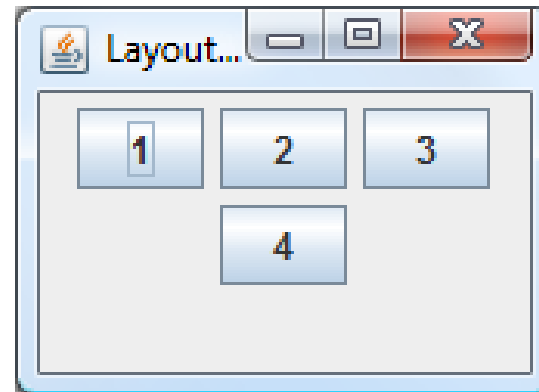
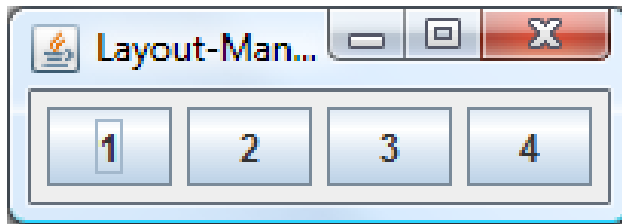
        // Layout-Manager setzen.
        frame.setLayout (new FlowLayout());

        // Die hinzugefuegten GUI-Komponenten werden nach den Regeln
        // des Layout-Managers angeordnet.
        frame.add (new JButton ("1"));
        frame.add (new JButton ("2"));
        ...

        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible (true);
    }
}
```

➤ `pack()`

- Durch den Aufruf der Methode `pack()` berechnet ein GUI – Container unter Einbindung des gesetzten Layout – Managers und der hinzugefügten GUI – Komponenten die optimale Fenstergröße
- Ausgabe des Hauptfensters mit dem Layout-Manager Flow-Layout:



Anordnung der Schaltflächen nach  
Größenänderung des Hauptfensters



## Layout-Manager – Grid-Layout I

### ➤ Layout-Manager – Grid-Layout

- Beim Grid-Layout werden neu hinzugefügte GUI-Komponenten in einem Gitter angeordnet.
- Beim Aufruf des Konstruktors können Spalten- und Zeilenanzahl übergeben werden.

```
...
public class GridLayoutTest{
    public static void main (String[] args){
        JFrame frame = new JFrame ("Layout-Manager: Grid-Layout");

        // Layout-Manager setzen. Es wird ein Grid-Layout mit 3 Zeilen
        // (1. Parameter) und 2 Spalten (2. Parameter) verwendet.
        frame.setLayout (new GridLayout (3, 2));

        // Die hinzugefuegten GUI-Komponenten werden nach den Regeln
        // des Layout-Managers angeordnet.
        frame.add (new JButton ("Zeile: 1 / Spalte: 1"));
        frame.add (new JButton ("Zeile: 1 / Spalte: 2"));
        frame.add (new JButton ("Zeile: 2 / Spalte: 1"));
        frame.add (new JButton ("Zeile: 2 / Spalte: 2"));
        frame.add (new JButton ("Zeile: 3 / Spalte: 1"));

        ...
    }
}
```

Demo in Eclipse:  
GridLayoutTest.java

➤ Hauptfenster mit Layout-Manager Grid-Layout



➤ Regeln

- Die erste hinzugefügte GUI-Komponente wird in die linke obere Ecke angeordnet.
  - Zeile 1, Spalte 1
- Weitere GUI-Komponenten füllen Spalte um Spalte die aktuelle Zeile von links nach rechts.
- Ist das Ende der Zeile erreicht (alle Spalten sind also gesetzt), wird die nächste Zeile befüllt (beginnend bei Spalte 1)
- Nicht benötigte Zeilen werden nicht angezeigt
- Zellen eines Grid-Layouts besitzen alle die gleiche Größe. Bei Vergrößerung des Hauptfensters, passen sich die GUI-Komponenten der neuen Größe proportional an.

## Aufgabe 14.03

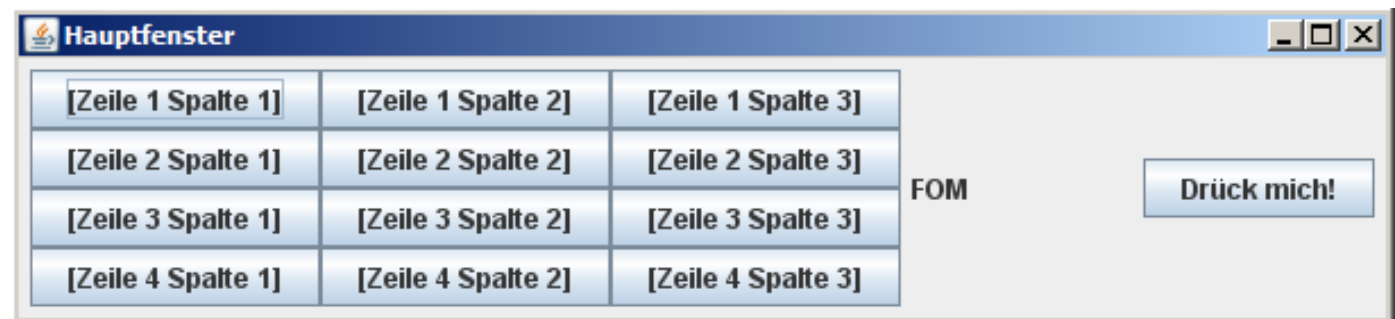
### Aufgabe 14.03 Layout – Manager

Implementieren Sie eine Klasse `Aufg_14_03` mit folgenden Anforderungen:

Erzeugen Sie ein Hauptfenster mit Flow-Layout. Legen Sie zwei `JPanels` an, beide verwenden `Grid-Layout`:

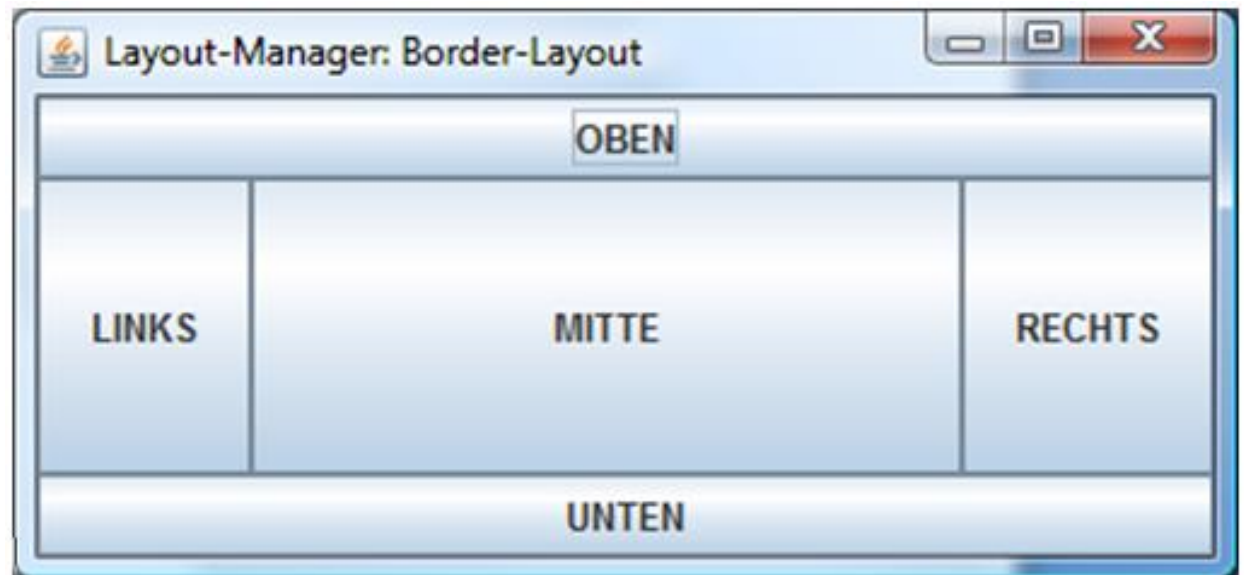
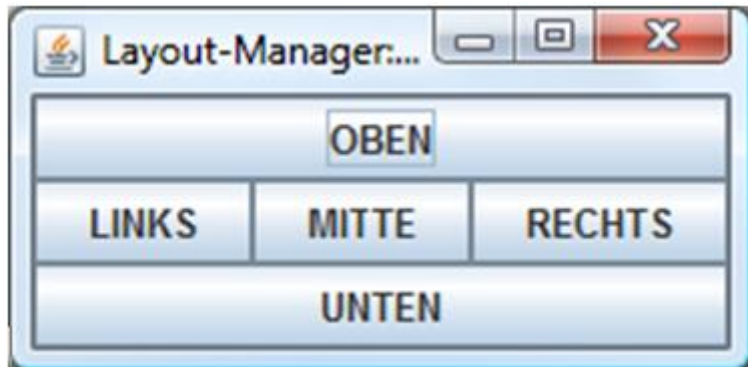
- `JPanel 1` hat 4 Zeilen und 3 Spalten
- Legen Sie mit Hilfe von Schleifen `JButton`-Schaltflächen auf das erste `JPanel`
- Die Beschriftung der `JButtons` lautet dabei: [Zeile 1 Spalte 1] etc.
- `JPanel 2` hat 1 Zeile und 2 Spalten
- Die linke Komponente ist ein `JLabel` mit dem Text „FOM“.
- Die rechte Komponente ist ein `JButton` mit dem Text „Drück mich!“.

Fügen Sie beide `JPanels` dem Hauptfenster zu.



### ➤ Layout-Manager – Border-Layout

- Der BorderLayout-Manager verteilt die GUI-Komponenten auf die vier Randbereiche (oben, unten, links und rechts) sowie den Mittelbereich.
- Die Randbereiche werden in Anlehnung an die Himmelsrichtungen beschrieben:
  - BorderLayout.NORTH
  - BorderLayout.SOUTH
  - BorderLayout.WEST
  - BorderLayout.EAST
- Der Mittelbereich heißt:
  - BorderLayout.CENTER



## Layout-Manager – Border-Layout II

### ➤ Layout-Manager – Border-Layout

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

public class BorderLayoutTest{
    public static void main (String[] args){
        JFrame frame = new JFrame ("Layout-Manager: Border-Layout");

        // Fuer ein Hauptfenster vom Typ JFrame ist der Layout-Manager
        // Border-Layout bereits voreingestellt.
        // Beim Aufruf der Methode add() wird als zweiter Parameter
        // die Randbedingung zur Positionierung uebergeben.
        frame.add (new JButton ("OBEN"), BorderLayout.NORTH);
        frame.add (new JButton ("UNTEN"), BorderLayout.SOUTH);
        frame.add (new JButton ("LINKS"), BorderLayout.WEST);
        frame.add (new JButton ("RECHTS"), BorderLayout.EAST);
        frame.add (new JButton ("MITTE"), BorderLayout.CENTER);

        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

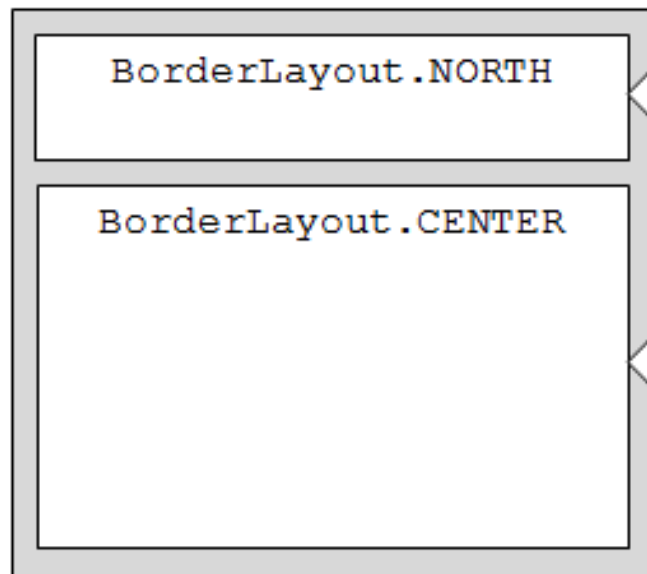
        // Optimale Fenstergroesse ermitteln und setzen.
        frame.pack();
        frame.setVisible (true);
    }
}
```

Demo in Eclipse:  
BorderLayoutTest.java

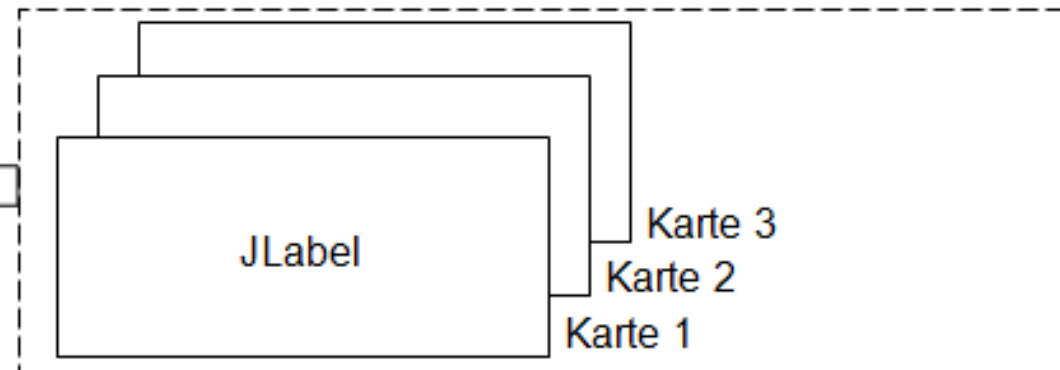
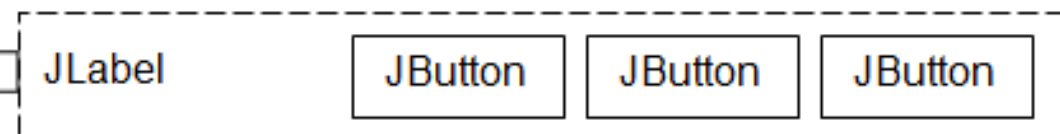
### ➤ Layout-Manager – Card-Layout

- Der Card-Layout-Manager verteilt die GUI-Komponenten vergleichbar mit einem Kartenstapel
- Es ist immer nur die oberste Karte sichtbar:

GUI-Container vom Typ `JFrame`  
mit `Border-Layout`



GUI-Container vom Typ `JPanel` mit `Flow-Layout`



GUI-Container vom Typ `JPanel` mit `Card-Layout`

## ➤ Layout-Manager – Card-Layout II

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class CardLayoutTest
{
    public static void main (String[] args)
    {
        // Hauptfenster und zwei JPanel-Container anlegen.
        JFrame frame = new JFrame ("Kartenstapel-Beispiel");
        JPanel panel = new JPanel();
        JPanel pane2 = new JPanel();
        ButtonController controller = new ButtonController (pane2);
        // Ersten JPanel-Container befüllen.
        panel.add (new JLabel ("Welche Karte soll angezeigt werden?"));
        for (int i = 1; i <= 3; i++)
        {
            JButton ref = new JButton (new Integer(i).toString());
            ref.addActionListener (controller);
            panel.add (ref);
        }
        // JPanel-Container dem Hauptfenster hinzufuegen.
        frame.add (panel, BorderLayout.NORTH);
        // Zweiten JPanel-Container konfigurieren und befüllen.
        pane2.setBackground (Color.YELLOW);
        pane2.setLayout (new CardLayout (5, 5));
        ..siehe nächste Folie !
    }
}
```

**Demo in Eclipse:**  
CardLayoutTest.java

## Layout-Manager – Border-Layout II

### ➤ Layout-Manager – Card-Layout III

```
...
    // Der zweite Parameter der Methode add() stellt wiederum
    // eine Randbedingung dar. Mit Hilfe des zweiten Parameters
    // kann spaeter die anzuzeigende Karte bestimmt werden.
    pane2.add (new JLabel ("Karte 1"), "1");
    pane2.add (new JLabel ("Karte 2"), "2");
    pane2.add (new JLabel ("Karte 3"), "3");

    // JPanel-Container dem Hauptfenster hinzufuegen.
    frame.add (pane2, BorderLayout.CENTER);
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible (true);
}
}
class ButtonController implements ActionListener{
    JPanel ref;
    public ButtonController (JPanel ref)    {
        this.ref = ref;
    }
    public void actionPerformed (ActionEvent e)    {
        CardLayout card = (CardLayout) ref.getLayout();
        // Abhaengig davon, welche Schaltflaeche gedrueckt wurde,
        // wird die zugehoerige Karte angezeigt.
        card.show (ref, ((JButton) e.getSource()).getText());
    }
}
```

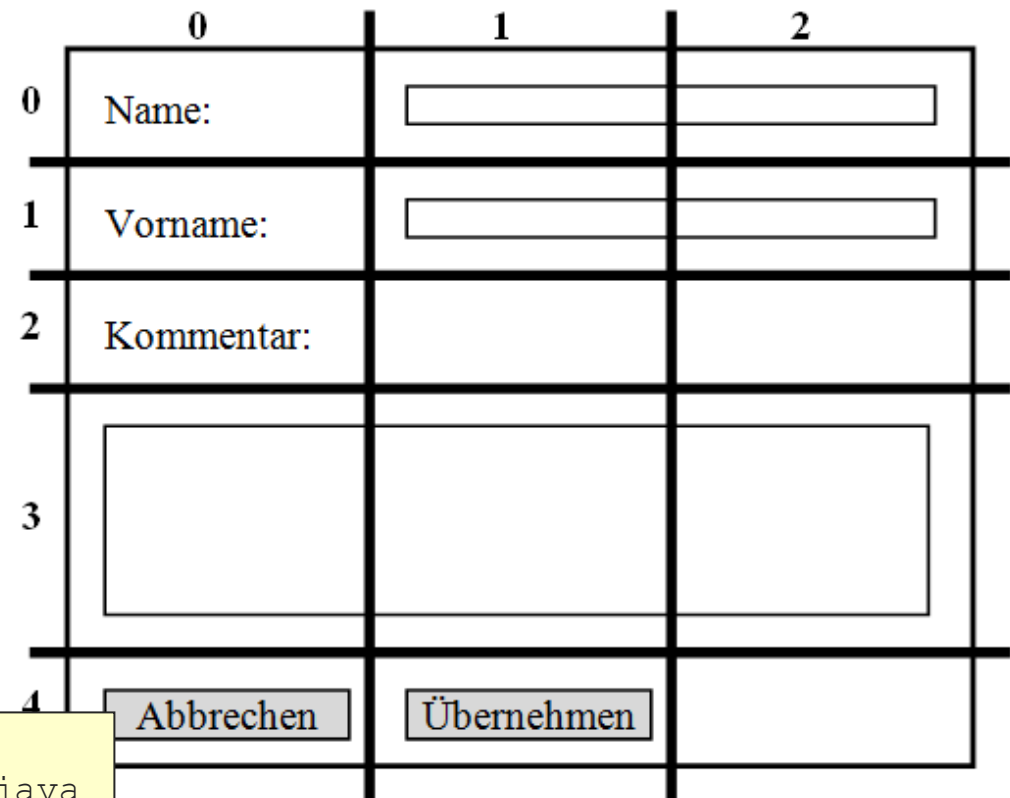
Demo in Eclipse:  
CardLayoutTest.java



## Layout-Manager – GridBag-Layout I

### ➤ Layout-Manager – GridBag-Layout

- Der GridBag-Layout-Manager legt ein Gitter Layout mit Zellen unterschiedlicher Größe fest.
- Die Positionierung einer GUI – Komponente befindet sich in einer oder mehreren Zellen. Diese werden durch Eigenschaften der Klasse `GridBagConstraints` festgelegt:
- Datenfelder die zur Verfügung stehen:
  - `gridx` und `gridy`
  - `gridheight` und `gridwidth`
  - `weightx` und `weighty`
  - `ipadx` und `ipady`
  - `fill`
  - `insets`
  - `anchor`



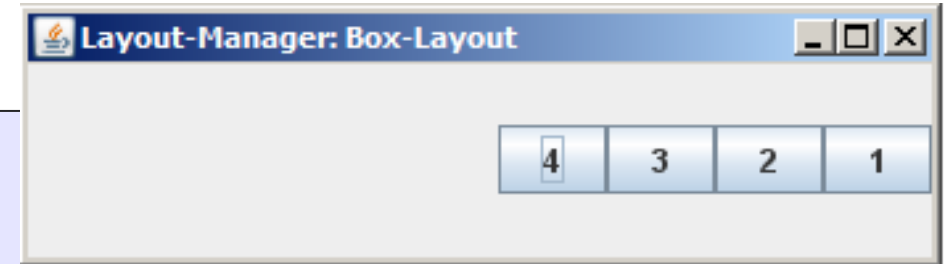
Demo in Eclipse:  
`GridBagLayoutTest.java`

## Layout-Manager – Box-Layout I

---

- Layout-Manager – Box-Layout, wird durch die Klasse `BoxLayout` implementiert.
- Ordnet hinzugefügte GUI-Komponenten in einer Reihe an:
  - entweder horizontal oder vertikal
  - die Richtung der Reihe wird über Konstanten angegeben:
    - `X_AXIS`: waagerecht von links nach rechts
    - `Y_AXIS`: senkrecht von oben nach unten
    - `LINE_AXIS`: waagerecht, Orientierung des Verlaufs richtet sich nach der Orientierung des GUI-Containers
    - `PAGE_AXIS`: senkrecht, Orientierung des Verlaufs richtet sich nach der Orientierung des GUI-Containers
- Die Orientierung eines GUI-Containers kann mit Hilfe der Methode `setComponentOrientation()` gesetzt werden.
  - wird auf die `ContentPane` angewendet, nicht auf das Hauptfenster direkt!

## ➤ Layout-Manager – Box-Layout II



```
import javax.swing.*;
import java.awt.ComponentOrientation;
public class BoxLayoutTest{
    public static void main (String[] args){
        JFrame frame = new JFrame ("Layout-Manager: Box-Layout");

        frame.getContentPane().setComponentOrientation(
            ComponentOrientation.RIGHT_TO_LEFT);

        // Layout-Manager setzen.
        frame.setLayout (new BoxLayout (frame.getContentPane(),
                                         BoxLayout.LINE_AXIS));

        // Hinzugefuegte GUI-Komponenten werden nach den Regeln des
        // Layout-Managers und der Container-Orientierung angeordnet.
        frame.add (new JButton ("1"));
        frame.add (new JButton ("2"));
        frame.add (new JButton ("3"));
        frame.add (new JButton ("4"));

        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.setSize (350,100);
        frame.setVisible (true);
    }
}
```

Demo in Eclipse:  
BoxLayoutTest.java

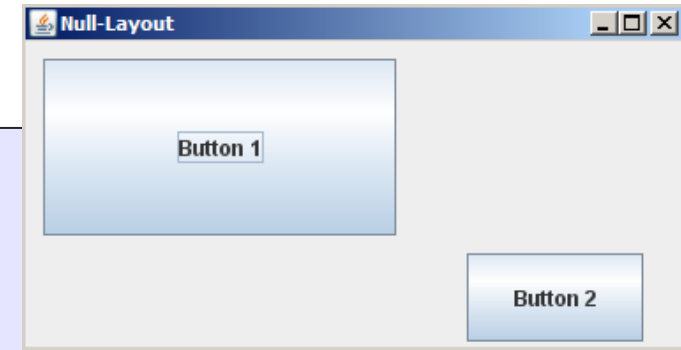
## Layout-Manager – Null-Layout I

---

- Layout-Manager – Null-Layout,
- GUI-Komponenten können durch Vorgabe von Koordinaten und Größe im GUI-Container angeordnet werden (`absolute Positionierung`).
- Wird die Größe des GUI-Containers verändert, erfolgt keine Verschiebung der GUI-Komponenten.
- Verwendung des Null-Layouts erfolgt durch:
  - `setLayout(null);`
- Für jede einzelne GUI-Komponente kann die Größe und die Position mit den folgenden Methoden festgelegt werden:
  - `setBounds(x, y, width, height)` **oder**
  - `setLocation(x, y)` **und** `setSize(width, height)`

## ➤ Layout-Manager – Null-Layout II

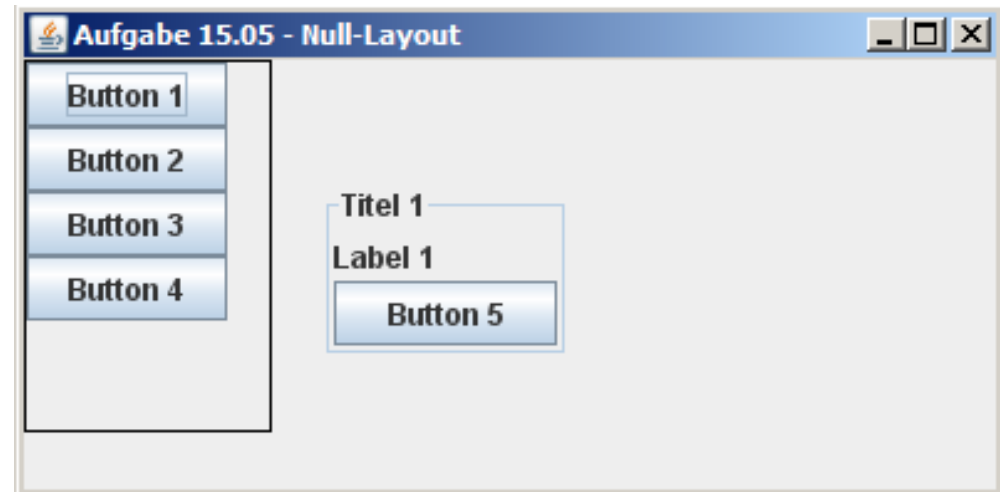
```
import javax.swing.JButton;  
import javax.swing.JFrame;  
  
public class NullLayoutTest  
{  
    public static void main (String[] args)  
    {  
        JFrame frame = new JFrame ("Null-Layout");  
  
        frame.setLayout(null);  
  
        JButton b1 = new JButton("Button 1");  
        b1.setBounds(10, 10, 200, 100);  
        frame.add(b1);  
  
        JButton b2 = new JButton("Button 2");  
        b2.setLocation(250, 120);  
        b2.setSize(100, 50);  
        frame.add(b2);  
  
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);  
        frame.setSize(380,200);  
        frame.setVisible (true);  
    }  
}
```



Demo in Eclipse:  
NullLayoutTest.java

## Aufgabe 14.05 Fenster Layout

- ✓ Erstellen Sie ein Hauptfenster im Null-Layout, welches zwei `JPanel`s anzeigen soll.
- ✓ Das erste `JPanel` soll seinerseits das `BoxLayout` in senkrechter Verlaufsrichtung entlang der Y-Achse verwenden und vier `JButtons` aufnehmen.
- ✓ Das zweite `JPanel` soll das `BorderLayout` verwenden. Fügen Sie ein `JLabel` und einen `JButton` hinzu (NORTH und SOUTH).
- ✓ Beide `JPanel`s sollen einen Rahmen erhalten (über `setBorder(Border border)`). `JPanel1` erhält einen einfachen schwarzen Rahmen, `JPanel2` einen Rahmen mit einem Titel. Recherchieren Sie hierzu `BorderFactory` in der API.



1. Grundbegriffe der Programmierung

2. Einfache Beispielprogramme

3. Datentypen und Variablen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

6. Blöcke und Methoden

7. Klassen und Objekte

8. Vererbung und Polymorphie

9. Pakete

10. Ausnahmebehandlung

11. Schnittstellen (Interfaces)

12. Ein-/Ausgabe und Streams

13. Applets

14. Oberflächenprogrammierung

Inhalte

✓ Architekturmerkmale Swing und AWT

✓ GUI – Container

✓ Layout Manager

✓ Ereignisbehandlung

Swing Komponenten

Sonstiges

- Bei der Interaktion des Benutzers mit der grafischen Bedienoberfläche werden Ereignisse ausgelöst.
  - z.B. beim Klicken auf einen `JButton` wird das Ereignis „Schaltfläche gedrückt“ ausgelöst.
  - Das Ereignis wird dann an einen sogenannten Controller weitergereicht.
  - Der Controller reagiert dann entsprechend darauf.
  - Wie sage ich z.B. dem `JButton`, dass er auf einen Klick reagieren soll?
  - Bislang passiert ja noch nichts, wenn man darauf klickt:





## ➤ Ereignisbehandlung

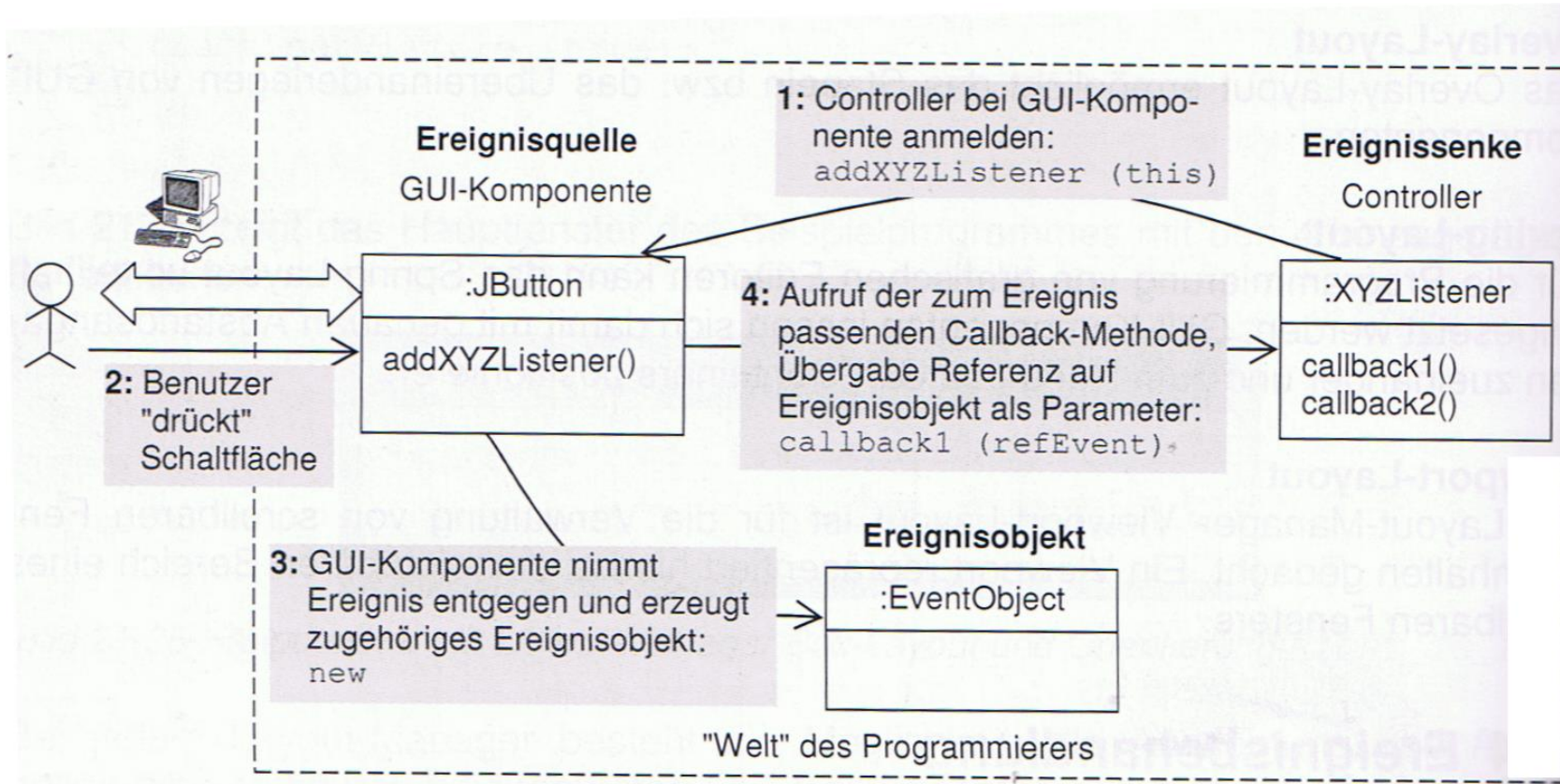


Bild 21-36 Ereignissenke und Ereignisquelle in der "Welt" des Programmiers

- Einer GUI-Komponente sind Callback-Schnittstellen (mit Namen XYZ-Listener) zugeordnet, die der Programmierer in einem Controller implementieren muss, falls er Ereignisse für die GUI-Komponente abfangen und verarbeiten möchte.
  
- Beispiel Schaltfläche `Jbutton`
  - Callback-Schnittstelle: `ActionListener`
  - Callback-Methode: `actionPerformed()`
  
- Damit die GUI-Komponente weiß, an welche Controller eintretende Ereignisse weitergeleitet werden müssen, muss der Programmierer seinen Controller bei der GUI-Komponente anmelden:
  - `addXYZListener()`

## Ereignisbehandlung Beispiel I

### ➤ JButton

```
public class JButtonTest extends JFrame {
    private JButton jButton0;
    private int i = 0;

    public JButtonTest() {
        jButton0 = new JButton();
        jButton0.setText("Klick mich ["+i+"]");
        jButton0.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent event) {
                i++;
                jButton0.setText("Klick mich ["+i+"]");
                System.out.println("Event-Quelle: "+event.getSource().toString());
                System.out.println("Event-Kommando: "+event.getActionCommand());
            }
        });
        setLayout(new FlowLayout());
        add(jButton0);
    }
}
```

Demo In Eclipse:  
JButtonTest.java

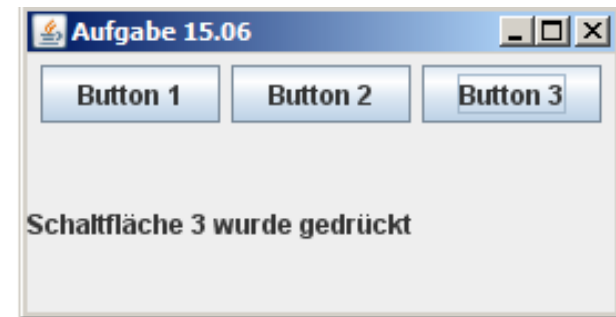
## ➤ JButton

```
public class JButtonTest2 extends JFrame implements ActionListener{  
    private JButton jButton0, jButton1;  
    public JButtonTest2() {  
        jButton0 = new JButton("OK");  
        jButton0.addActionListener(this);  
  
        jButton1 = new JButton("Abbrechen");  
        jButton1.addActionListener(this);  
  
        setLayout(new FlowLayout());  
        add(jButton0);  
        add(jButton1);  
    }  
    public void actionPerformed(ActionEvent e) {  
        if(e.getSource()==jButton0)  
            System.out.println("Schaltflaeche 1");  
        else if(e.getSource()==jButton1)  
            System.out.println("Schaltflaeche 2");  
    }  
}
```

**Demo In Eclipse:**  
JButtonTest2.java

### Aufgabe 14.06 ActionListener

Erstellen Sie ein Hauptfenster im `Border-Layout`. Im Norden fügen Sie ein `JPanel` hinzu, welches im `Flow-Layout` 3 `JButtons` aufnimmt. Im Center fügen Sie ein `JLabel` ein. Implementieren Sie `ActionListener` so, dass beim Klicken auf die `JButtons` jeweils ein anderer Text in das `JLabel` geschrieben wird (siehe Ausgabebeispiel).



1. Grundbegriffe der Programmierung

2. Einfache Beispielprogramme

3. Datentypen und Variablen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

6. Blöcke und Methoden

7. Klassen und Objekte

8. Vererbung und Polymorphie

9. Pakete

10. Ausnahmebehandlung

11. Schnittstellen (Interfaces)

12. Ein-/Ausgabe und Streams

13. Applets

14. Oberflächenprogrammierung

Inhalte

✓ Architekturmerkmale Swing und AWT

✓ GUI – Container

✓ Layout Manager

✓ Ereignisbehandlung

✓ Swing Komponenten

Sonstiges

### ➤ Text Komponenten

#### Statischer Text (JLabel)

Text

#### Einfaches Textfeld (JTextField)

Text

#### Textfeld mit verdeckter Eingabe (JPasswordField)

\*\*\*\*\*

#### Mehrzeiliges Textfeld (JTextArea)

Ein veränderbarer und mehrzeiliger Text.

#### Textfeld mit HTML-Formatierung (JEditorPane)

##### JEditorPane

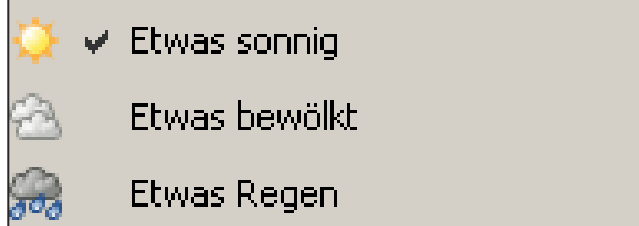
Mit der *JEditorPane* lassen sich  
in [HTML](#) formatierte Texte anzeigen.

### ➤ Schaltflächen Komponenten

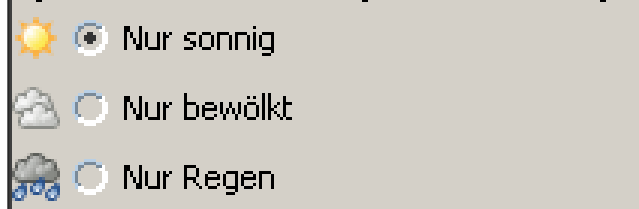
#### Einfache Schaltfläche (JButton)



#### Kontrollkästchen (JCheckBox)



#### Optionsschaltfläche (JRadioButton)





### ➤ JCheckBox

```
...
// Checkbox "sonnig" erzeugen
JCheckBox sunnyCheckbox = new JCheckBox ("Etwas sonnig", true);

// Hinzufügen zum Hauptfenster.
frame.add (sunnyCheckbox);

...

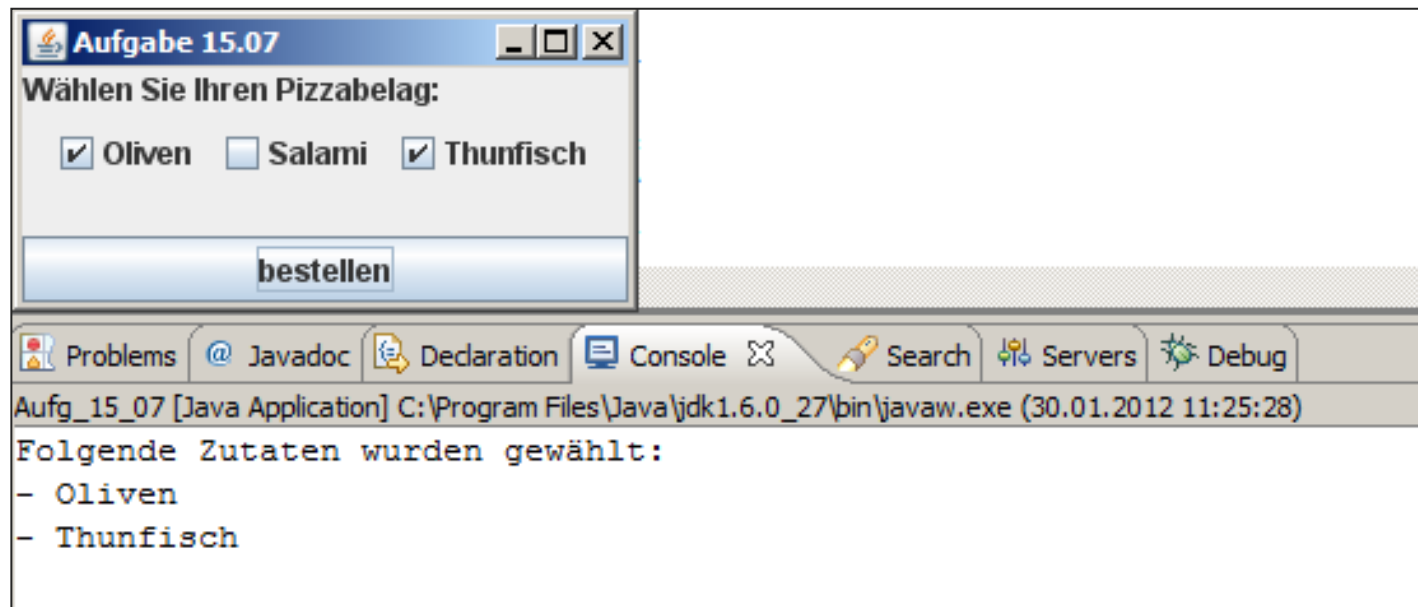
// Abfragen, ob die Checkbox ausgewählt wurde
if(sunnyCheckbox.isSelected()){
    System.out.println("Sonnig wurde ausgewählt.");
}
...
```

### Aufgabe 14.07 Swing I

Erstellen Sie ein Hauptfenster mit folgenden Komponenten (Layout-Manager Ihrer Wahl):

- 1 JLabel mit folgendem Text: „Wählen Sie Ihren Pizzabelag:“
- 3 JCheckBoxes mit folgenden Werten: Oliven, Salami, Thunfisch
- 1 JButton mit dem Text „bestellen“

Ein Anwender stellt nun über die `CheckBoxes` seine Pizza zusammen und klickt auf `bestellen`. Danach erscheinen über die Systemausgabe die ausgewählten Zutaten.



➤ Listen und Tabellen

➤ Liste (JList)



➤ Kombinationsfeld (JComboBox)



➤ Tabelle (JTable)

Matrikelnummer	Vorname	Nachname
00001	Hans	Müller
00002	Anna	Hansel
00003	Jörg	Schneider

## ➤ JComboBox

```
...
private JComboBox petList;
...
String[] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };

//Create the combo box
//Indices start at 0, so 4 specifies the pig.
petList = new JComboBox(petStrings);

petList.addActionListener(this);
...
public void actionPerformed(ActionEvent e) {

    if(e.getSource() == petList){
        System.out.println("Wert: "+combo.getSelectedItem().toString());
        System.out.println("Index: "+combo.getSelectedIndex());
    }
}
...
```

**Ausgabebeispiel:**

Wert: Bird

Index: 0

### Aufgabe 14.08 Swing II

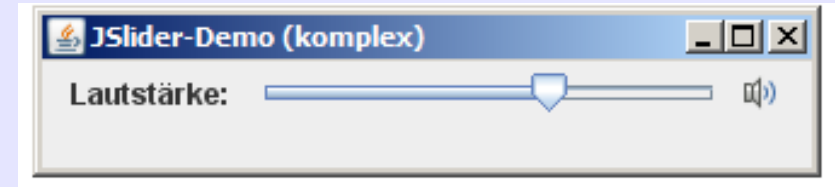
Erstellen Sie ein Hauptfenster mit folgenden Komponenten (Layout-Manager Ihrer Wahl):

- Eine `JComboBox` mit folgenden Werten: „Bitte wählen“, „Kreis zeichnen“, „Rechteck zeichnen“ und „Linie zeichnen“.
- Ein `PaintPanel`. Sie finden die Klasse `PaintPanel.java` im Online-Campus. `PaintPanel` ist eine von `JPanel` abgeleitete Klasse, welche in ihrer `paintComponent()`-Methode je nach Wert ihrer Variablen `drawItem` entweder einen Kreis, ein Rechteck oder eine Linie zeichnet.
- Fügen Sie Ihrer `JComboBox` einen `ActionListener` hinzu, sodass bei Auswahl eines Wertes die Methode `setDrawItem(int i)` der Klasse `PaintPanel` aufgerufen wird. Übergeben Sie als Parameter den Index des ausgewählten Wertes aus der `JComboBox` (z.B. „Linie zeichnen“ wurde ausgewählt, dann `setDrawItem`-Methode mit „3“ aufrufen. Danach muss das `PaintPanel` neu gezeichnet werden, rufen Sie dazu die Methode `repaint()` des `JPanels` auf.



## ➤ JSlider (Schieberegler)

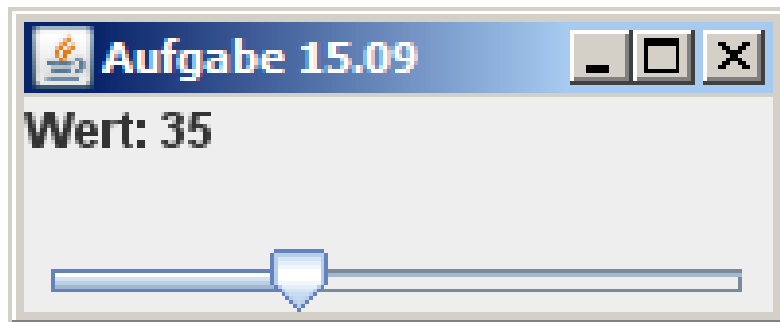
```
public class MyFrame extends JFrame implements ChangeListener{  
    private JSlider slider;  
    ...  
    // Initilaisierungsvariablen für den Schieberegler.  
    int min = 0;  
    int max = 100;  
    int start = 0;  
  
    slider = new JSlider(min,max,start);  
    add(slider);  
  
    // Event-Listener  
    slider.addChangeListener(this);  
    ...  
  
    public void stateChanged(ChangeEvent arg0) {  
        // Aktuelle Position des Schiebereglers abfragen.  
        System.out.println("Wert: "+slider.getValue());  
    }  
    ...  
}
```



### Aufgabe 14.09 JSlider

Erstellen Sie ein Hauptfenster mit folgenden Komponenten (Layout-Manager Ihrer Wahl):

- 1 `JLabel`, das den aktuellen Wert des Schiebereglers anzeigt.
- 1 `JSlider`, mit dem Startwert 0 und möglichen Werten zwischen 0 und 100.
- Versehen Sie den Schieberegler mit einem `ChangeListener`, sodass bei jeder Änderung des `JSliders` der aktuelle Wert im `JLabel` angezeigt wird (siehe Abbildung).



## ➤ Menüs I

- Die Klasse `JMenuBar` repräsentiert die Menüleiste, deren Menüeinträge durch die Basisklasse `JMenuItem` zur Verfügung gestellt werden.
- Mit der Methode `setJMenuBar()` wird die Menüleiste zum Hauptfenster hinzugefügt.

```
...
// Fenster definieren.
JFrame frame = new JFrame ("JMenuBar-Demo");

// Neues Menu erzeugen.
JMenuBar menuBar = new JMenuBar();

// Menu an Frame anhängen.
frame.setJMenuBar (menuBar);

// Menü "Datei" erzeugen.
JMenu fileMenu = new JMenu ("Datei");

// Menü an Menubar anhängen.
menuBar.add (fileMenu);

// Untermenü "Schließen" erzeugen.
JMenuItem exitItem = new JMenuItem ("Schließen");

// Untermenü an Menü "Datei" anhängen.
fileMenu.add (exitItem);
...
```



### ➤ Menüs II

- Damit beim Klicken auf einen Menüeintrag auch etwas passiert, müssen Sie einen `ActionListener` darauf anwenden.

```
public class MenuTest extends JFrame implements ActionListener{

    private JMenuItem exitItem;
    ...

    // Untermenü "Schließen" erzeugen.
    exitItem = new JMenuItem ("Schließen");
    exitItem.addActionListener(this);

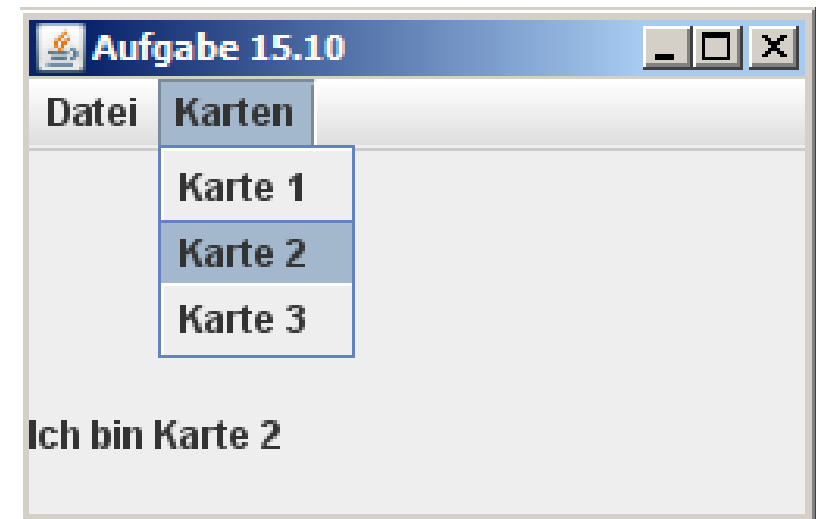
    ...

    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==exitItem)
            System.exit(0);
    }
}
```

### Aufgabe 14.10 Menü

Erstellen Sie ein Hauptfenster mit folgenden Komponenten (Layout-Manager Ihrer Wahl):

- 1 Menüleiste mit den beiden Menüs „Datei“ und „Karten“.
- Im Menü Datei befindet sich ein Menüpunkt namens „schließen“ zum Beenden des Programms.
- Im Menü Karten befinden sich 3 Menüpunkte: „Karte 1“, „Karte 2“ und „Karte 3“.
- Auf der Zeichenfläche des Hauptfensters soll sich ein `JLabel` befinden.
- Klickt man auf die Menüeinträge des Menüs Karten, soll das `JLabel` die jeweils gewählte Karte als Text darstellen. (siehe Abbildung).











## Swing Komponenten – JOptionPane I

### ➤ Vorgefertigter Optionsdialog (JOptionPane)

- Die Klasse `JOptionPane` bietet Standarddialoge um den Anwender über den Programmstatus zu informieren oder vom Anwender einfache Werte eingeben zu lassen.

#### Icons used by JOptionPane

Icon description	Java look and feel	Windows look and feel
question		
information		
warning		
error		

## ➤ Vorgefertigter Optionsdialog (JOptionPane)

### ▪ Einfache Meldung (Information)

```
//default title and icon  
JOptionPane.showMessageDialog(null, "Es ist kalt.");
```



### ▪ Warnmeldung (Warning)

```
//custom title, warning icon  
JOptionPane.showMessageDialog(null,  
    "Es ist kalt.",  
    "Kältewarnung",  
    JOptionPane.WARNING_MESSAGE);
```

Vaterkomponente oder null

Meldungstext  
Titel  
Meldungstyp

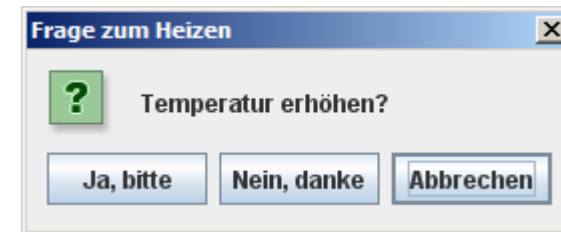


### ▪ Fehlermeldung (Error)

```
//custom title, error icon  
JOptionPane.showMessageDialog(null,  
    "Es ist zu kalt!",  
    "Kältefehler",  
    JOptionPane.ERROR_MESSAGE);
```



➤ Vorgefertigter Optionsdialog (JOptionPane)



```
//Custom button text
Object[] options = {"Ja, bitte",
                    "Nein, danke", -----Text für die Buttons
                    "Abbrechen"};

int n = JOptionPane.showOptionDialog(null,-----Vaterkomponente oder null
    "Temperatur erhöhen?",-----Meldungstext
    "Frage zum Heizen",-----Titel
    JOptionPane.YES_NO_CANCEL_OPTION,-----Optionstyp
    JOptionPane.QUESTION_MESSAGE,-----Meldungstyp
    null,-----optionales Icon
    options,-----Optionstexte
    options[2]);-----Defaultwert

System.out.println("Rückgabewert: "+n);
```

### Aufgabe 14.11 Dialog

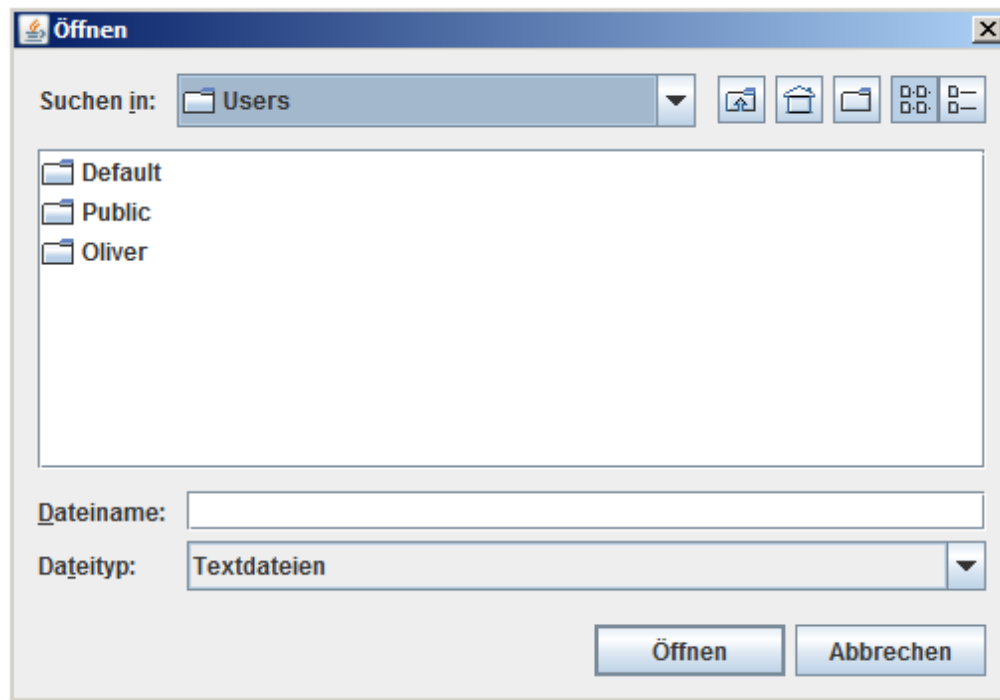
Erstellen Sie innerhalb der `main-Methode` einen Optionsdialog mit folgenden Eigenschaften:

- Meldungstext: „Wie fühlen Sie sich?“
- Titelzeile: „Eine kleine Frage“
- 3 Optionsbuttons: „gut“, „mittel“ und „schlecht“
- Verwenden Sie den Rückgabewert des Optionsdialogs um festzustellen, welcher Button gedrückt wurde.
- Wurde „gut“ gewählt, dann soll eine Warnmeldung mit dem Text „Es kommen auch wieder schlechtere Zeiten!“ erscheinen.
- Wurde „mittel“ gewählt, dann soll eine Info-Meldung mit folgendem Text erscheinen: „Da kann man ja mit leben.“
- Wurde „schlecht“ gewählt, dann soll eine Fehlermeldung mit folgendem Text erscheinen: „Kopf hoch, gleich ist Feierabend.“



### ➤ Dateiauswahldialog (JFileChooser)

- Die Klasse `JFileChooser` simuliert einen betriebssystemabhängigen Dialog zur Auswahl von Dateien und Verzeichnissen.
- Der `Selektor` ist modal und kann für das Speichern und Öffnen konfiguriert sein.
- Zudem lassen sich die Pfade und ein Filter zur Auswahl spezieller Dateien setzen.
- Nach dem Schließen und Beenden mit dem `OK-Button` stehen ausgewählte Dateien zur Verfügung.



### ➤ Dateiauswahldialog (JFileChooser)

```
import java.io.File;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class JFileChooserDemo
{
    public static void main( String[] args )
    {
        JFileChooser fc = new JFileChooser();
        fc.setFileFilter( new FileNameExtensionFilter("Textdateien", "txt", "html", "log" ) );

        int state = fc.showOpenDialog( null );
        //int state = fc.showSaveDialog( null );

        if ( state == JFileChooser.APPROVE_OPTION )
        {
            File file = fc.getSelectedFile();
            System.out.println( file.getName() );
        }
        else
            System.out.println( "Auswahl abgebrochen" );

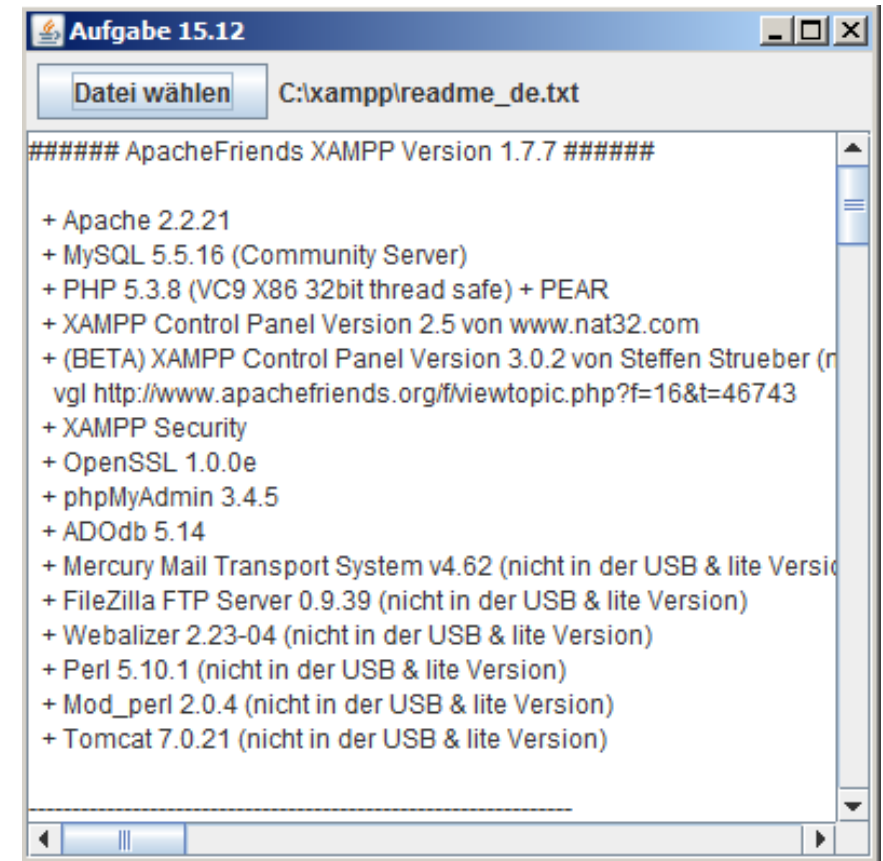
        System.exit( 0 );
    }
}
```



### Aufgabe 14.12 JFileChooser

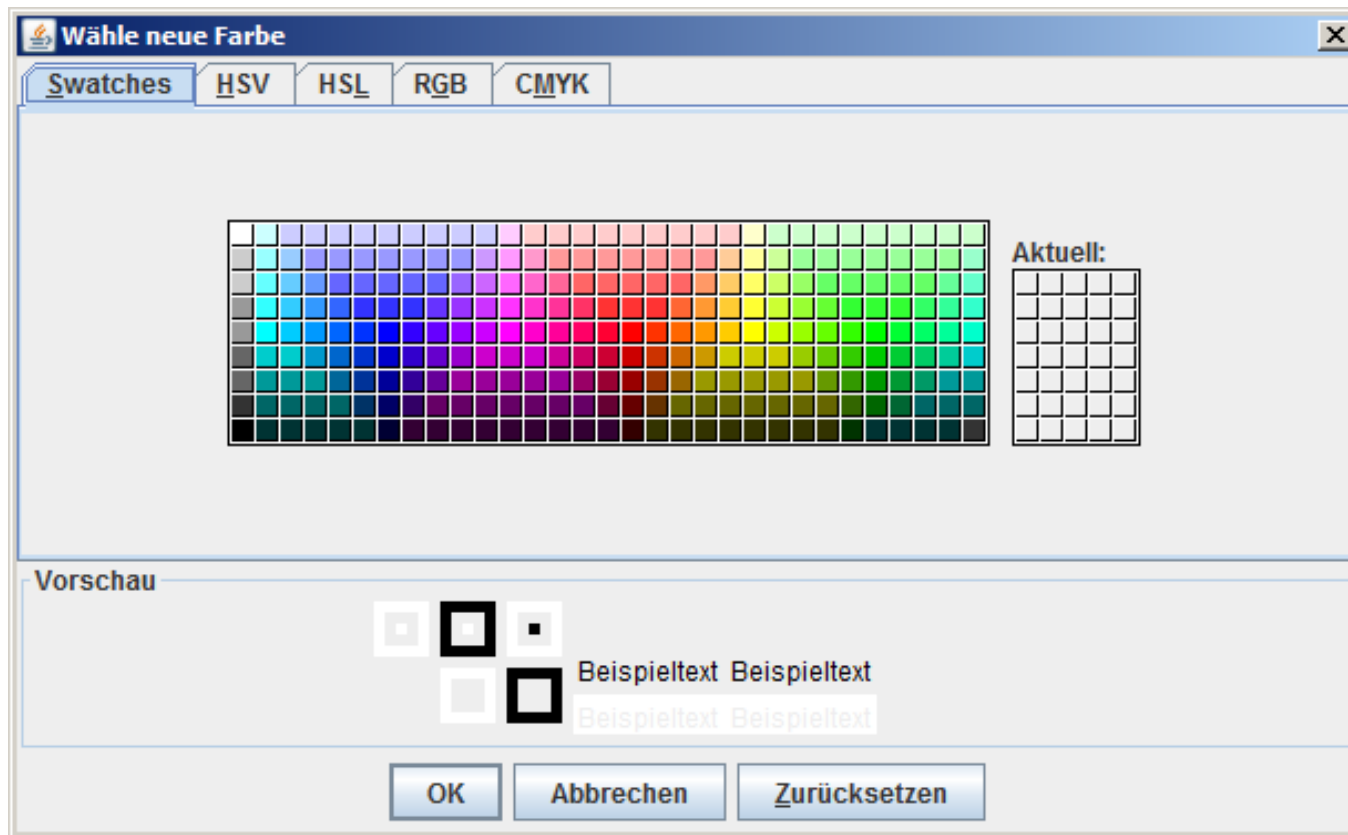
Erstellen Sie ein Hauptfenster wie in folgender Abbildung:

- Im oberen Bereich gibt es einen `JButton` „Datei wählen“. Klickt man darauf soll sich ein Dateiauswahlfenster öffnen, welches nur die Auswahl von Textdateien erlaubt.
- Wenn eine Textdatei ausgewählt wurde, soll der Name der Datei inkl. Pfad in einem `JLabel` neben dem `JButton` erscheinen.
- Außerdem soll der Dateiinhalt in einer `JTextArea` darunter angezeigt werden. Die `JTextArea` soll dabei mit Scrollbalken versehen werden. Recherchieren Sie dazu die Klasse `JScrollPane`!



### ➤ Farbauswahldialog (JColorChooser)

- Mit einem `JColorChooser` lassen sich Farben über drei unterschiedliche Reiter auswählen.
- Der Benutzer hat die Auswahl zwischen vordefinierten Farben, HSB-Werten und RGB-Werten.



## ➤ Farbauswahldialog (JColorChooser)

```
import java.awt.Color;

public class JColorChooserDemo extends JFrame implements ActionListener
{
    private JButton b;

    public JColorChooserDemo(){
        super("Farb-Button");
        b = new JButton( "Farbe ändern" );
        add( b );
        b.addActionListener(this);

        pack();
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        setVisible( true );
    }

    public static void main( String[] args )
    {
        new JColorChooserDemo();
    }

    public void actionPerformed(ActionEvent e) {

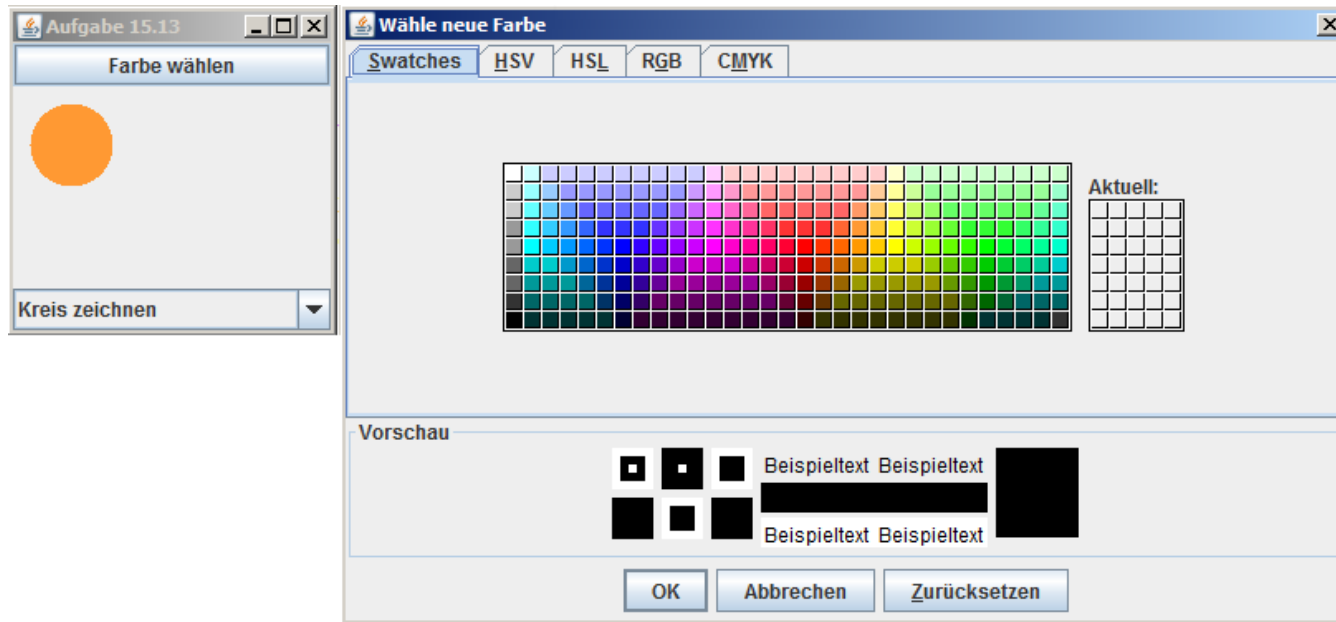
        Component comp = (Component) e.getSource();
        Color newColor = JColorChooser.showDialog( this, // Vaterkomponente
                                                    "Wähle neue Farbe", // Anzeigetext
                                                    comp.getBackground() ); // Startfarbe

        comp.setBackground( newColor );
    }
}
```

### Aufgabe 14.13 JColorChooser

Erstellen Sie ein Hauptfenster wie in folgender Abbildung:

- Erweitern Sie dazu die Aufgabe 14.08, indem Sie einen `JButton` „Farbe auswählen“ hinzufügen, der den `Farbwählerdialog` öffnet.
- Kreis, Rechteck und Linie sollen dann in der gewählten Farbe dargestellt werden. Rufen Sie dazu die Methode `setColor(Color c)` der Klasse `PaintPanel` auf.



1. Grundbegriffe der Programmierung

2. Einfache Beispielprogramme

3. Datentypen und Variablen

4. Ausdrücke und Operatoren

5. Kontrollstrukturen

6. Blöcke und Methoden

7. Klassen und Objekte

8. Vererbung und Polymorphie

9. Pakete

10. Ausnahmebehandlung

11. Schnittstellen (Interfaces)

12. Ein-/Ausgabe und Streams

13. Applets

14. Oberflächenprogrammierung

Inhalte

✓ Architekturmerkmale Swing und AWT

✓ GUI – Container

✓ Layout Manager

✓ Ereignisbehandlung

✓ Swing Komponenten

✓ Sonstiges

- Empfohlener Start einer Swing-Anwendung
- Es gibt insgesamt zwei Threads, die bei der Abarbeitung einer Swing-Anwendung beteiligt sind:
  - `Initial-Thread`  
arbeitet in der `main-Methode` und ist für die Initialisierung der grafischen Bedienoberfläche zuständig.
  - `Event-Dispatch-Thread`  
arbeitet die durch Benutzer ausgelösten Ereignisse ab.
- Problem:
  - Beide Threads greifen zeitgleich auf GUI-Komponenten zu
  - Es könnte z.B. ein Anwender ein Ereignis auslösen (z.B. auf Button klicken), bevor der Initial-Thread alle Daten für eine Tabelle aus einer Datenbank geladen hat.
  - So können inkonsistente Zustände entstehen.

➤ Empfohlener Start einer Swing-Anwendung

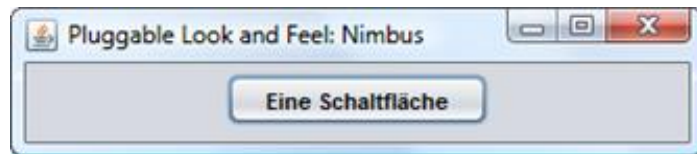
➤ Lösung:

- Aufbau und Initialisierung der grafischen Bedienoberfläche werden durch den Event-Dispatch-Thread durchgeführt.

```
...
public static void main (String[] args){
    // GUI im Event-Dispatch-Thread aufbauen und initialisieren.
    SwingUtilities.invokeLater (new Runnable(){
        public void run(){
            createGUI();
        }
    });
}

private static void createGUI(){
    JFrame frame = new JFrame ("Swing Threads");
    frame.setLayout (new FlowLayout());
    frame.add (new JButton("Drück mich!"));
    ...
}
...
```

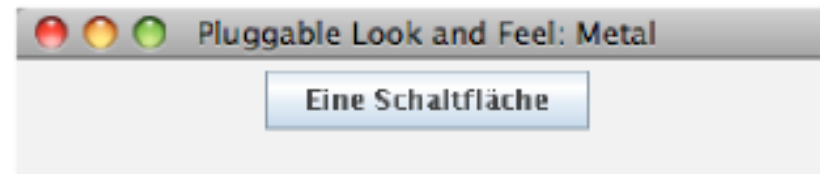
- Aussehen und Verhalten der GUI können verändert werden (Skins)
- Cross Platform-Look and Feel
  - Wird auf allen Plattformen angeboten, sieht überall nahezu gleich aus und verhält sich gleich.
  - In Java heißt der Standard-Look-and-Feel aktuell noch `Metal`. Eine neuere Variante heißt `Nimbus`.



a)



b)

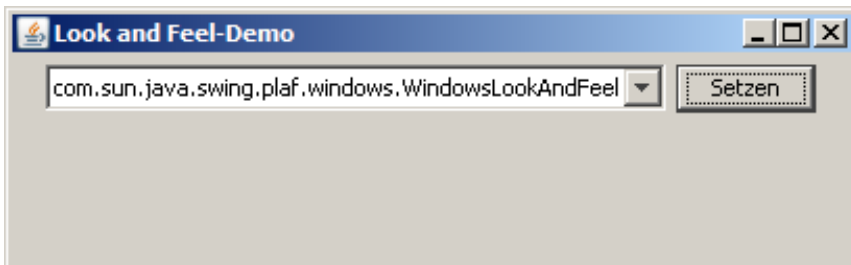




## ➤ System-Look and Feel

- Damit wird die Java-Anwendung an die native Darstellung des Betriebssystems angepasst.

System-Look-and-Feel	Plattform
CDE/Motif	Linux und Solaris, ohne GTK+ 2.2 oder höher installiert
GTK+	Linux und Solaris, mit GTK+ 2.2 oder höher installiert
Windows	Alle MS Windows-Plattformen seit XP



## Aufgabe 14.14 Look and Feel

Erstellen Sie ein Hauptfenster wie in folgender Abbildung:

- Für jedes auf Ihrem System verfügbaren Look-and-Feel soll ein eigener JButton mit dem Namen des Look-and-Feels als Text hinzugefügt werden. Tipp:

```
// alle verfügbaren Look-and-Feels auslesen
for(LookAndFeelInfo laf : UIManager.getInstalledLookAndFeels())
    System.out.println(laf.getClassName());
```

- Wenn ein Button geklickt wird, soll sich das Look-and-Feel der Anwendung sofort auf das gewählte Aussehen ändern. Tipp:

```
public void actionPerformed(ActionEvent e) {
    try{
        // Setze neues Look and Feel.
        UIManager.setLookAndFeel(e.getActionCommand());
        // Fenster neu zeichnen.
        SwingUtilities.updateComponentTreeUI (frame);
    }catch (Exception e1){...} ...
}
```

