

Graphical User Interface (GUI)

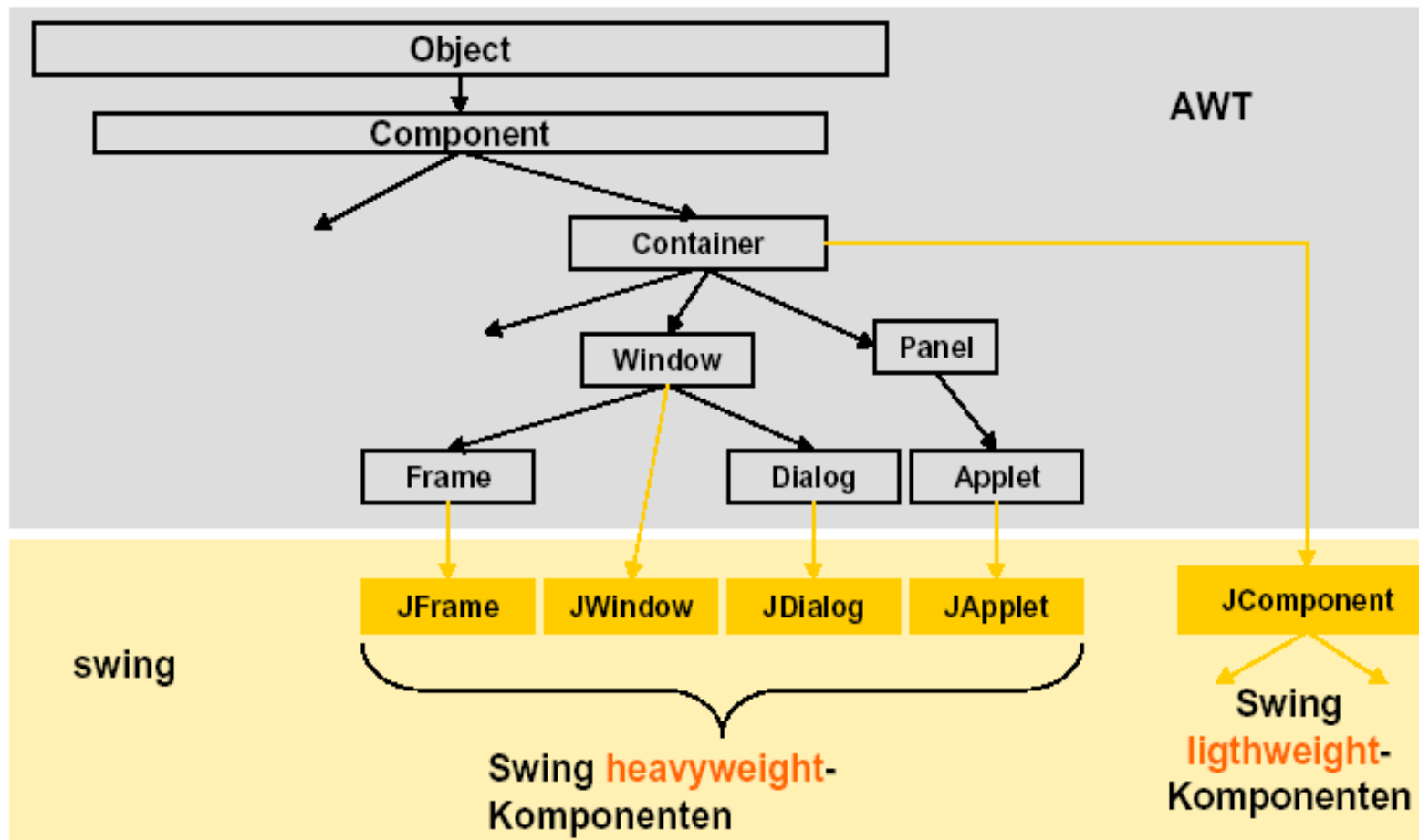
Benutzeroberflächen in Java

- JDK beinhaltet Bibliothek zur Erstellung graphischer Benutzeroberflächen:
 - AWT (Abstract Window Toolkit)
 - JFC/Swing (offiziell: Java Foundation Classes; Arbeitsname: Swing)
- neu: JavaFX (für Rich Internet Applications), Zukunft ungewiss
- ganz alt: AWT (aber Interfaces für Maus und Tastatur!)
- alt, aber noch aktuell: Swing

Warum Swing?

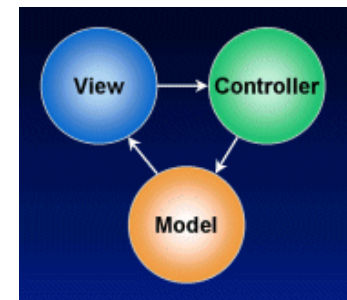
- AWT (abstract windowing toolkit)
 - war ein „Schnellschuss“
 - nicht konform zur Java Maxime „write once run multiple“
 - nur der kleinste gemeinsame Nenner für GUIs
- Probleme der AWT API führten zur Entwicklung von Swing
- Eigentlich veraltet aber fraglich, ob sich JavaFX durchsetzt

Klassenhierarchie



Aufbau von Komponenten

- Komponenten
 - Top-Level-Containers
 - Intermediate Containers
 - Atomic Components
- Zeichnung durch Java VM
 - Aufnehmen anderer Komponenten
- Model-View-Controller (MVC)
- Ressourcenhungrig - Performanceverlust?



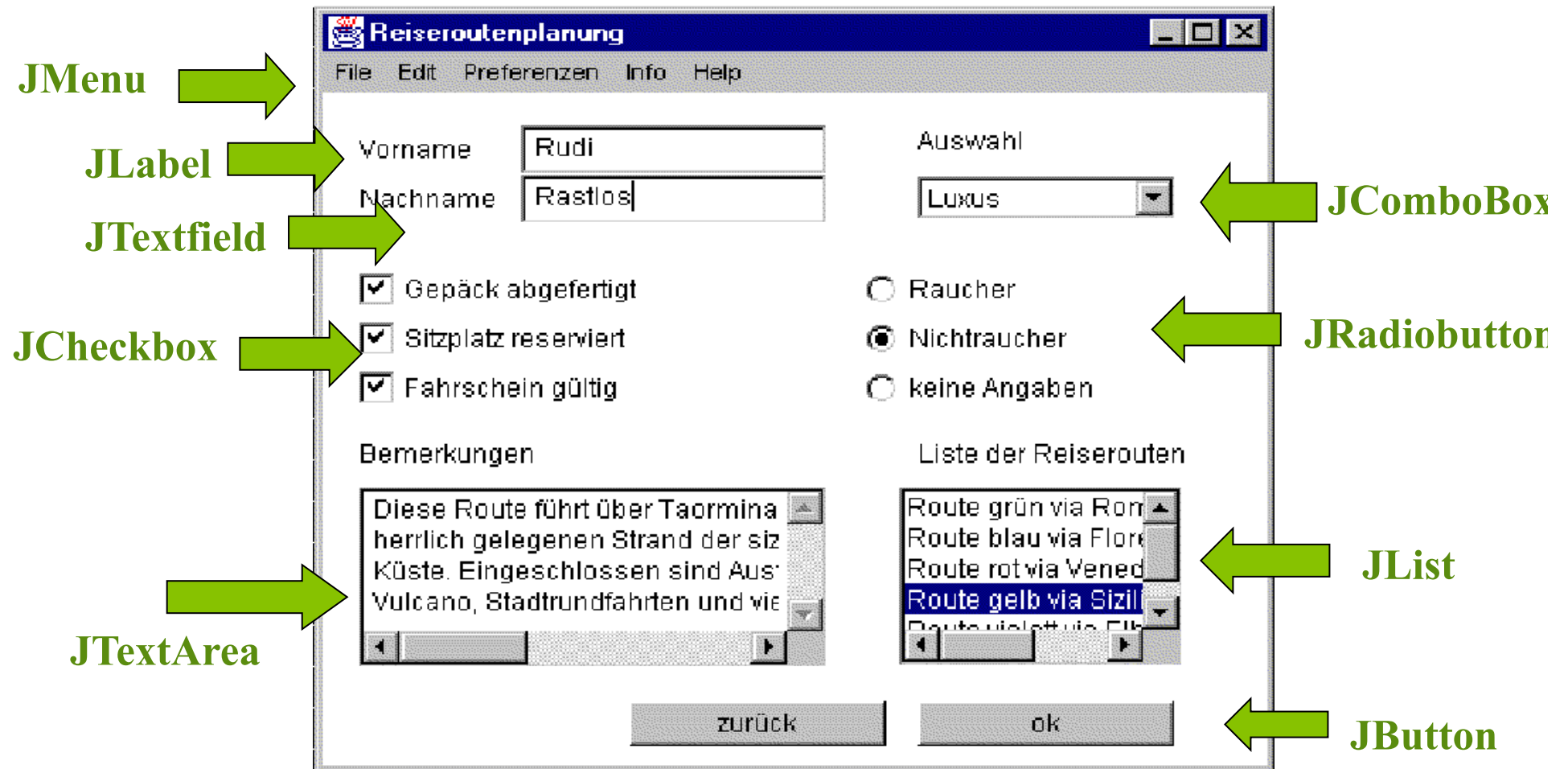
Bausteine einer GUI

- **Fenster:** äußerer Rahmen jeder grafischen Anwendung
- **Komponenten:** alle Oberflächenelemente
 - einfache Komponenten (z.B. Listenelemente, Buttons)
 - Container-Komponenten (werden mithilfe von Layout-Managern gestaltet und betten andere Komponenten ein)
- **Menüs:** einblendbare Befehlslisten
- **Layout-Manager:** besondere Klassen zur Positionierung von Komponenten
- **Events:** sind Nachrichten, die gesendet werden, wenn sich Tastatur- oder Mausereignisse ereignet haben
- **Zeichenoperationen:** für das Zeichnen von Punkten, Linien, Text usw. in Fenstern und Komponenten

Vorgehen zum Erstellen einer GUI

1. Erzeugen eines Hauptfensters (von `JFrame` abgeleitet)
2. Anhängen von Komponenten an das Fenster
3. evtl. Anhängen von Komponenten an Komponenten

GUI (Grafical User Interface)



Basisklasse: JComponent

Basisklasse für alle Swingkomponenten ist JComponent.

Sie vererbt folgende Eigenschaften zur Laufzeit

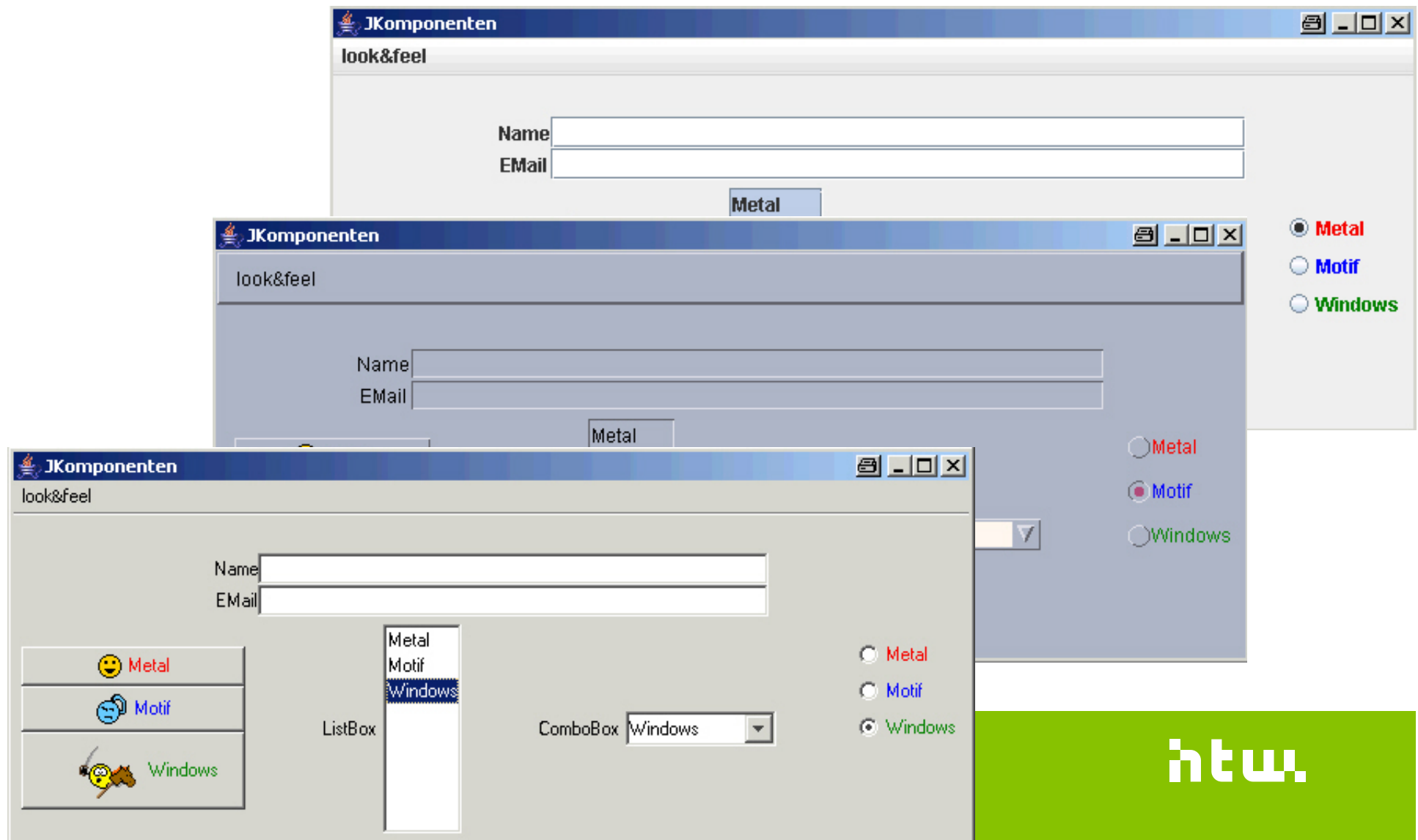
- das **Look & Feel** kann geändert werden
- Definition von **Tastenkombinationen** zur Aktivierung von Komponenten (Mnemonics)
- minimale und maximale **Größe** setzen
- **ToolTip** setzen

Look & Feel

Plugable Look & Feel (Motif , Window und Metal)

- Look & Feel einer Komponente kann vollständig und einfach ersetzt werden
- jede Komponente kann erweitert oder modifiziert werden
- Das Look & Feel lässt sich explizit überschreiben, entweder für die gesamte Applikation oder für einzelne Komponenten

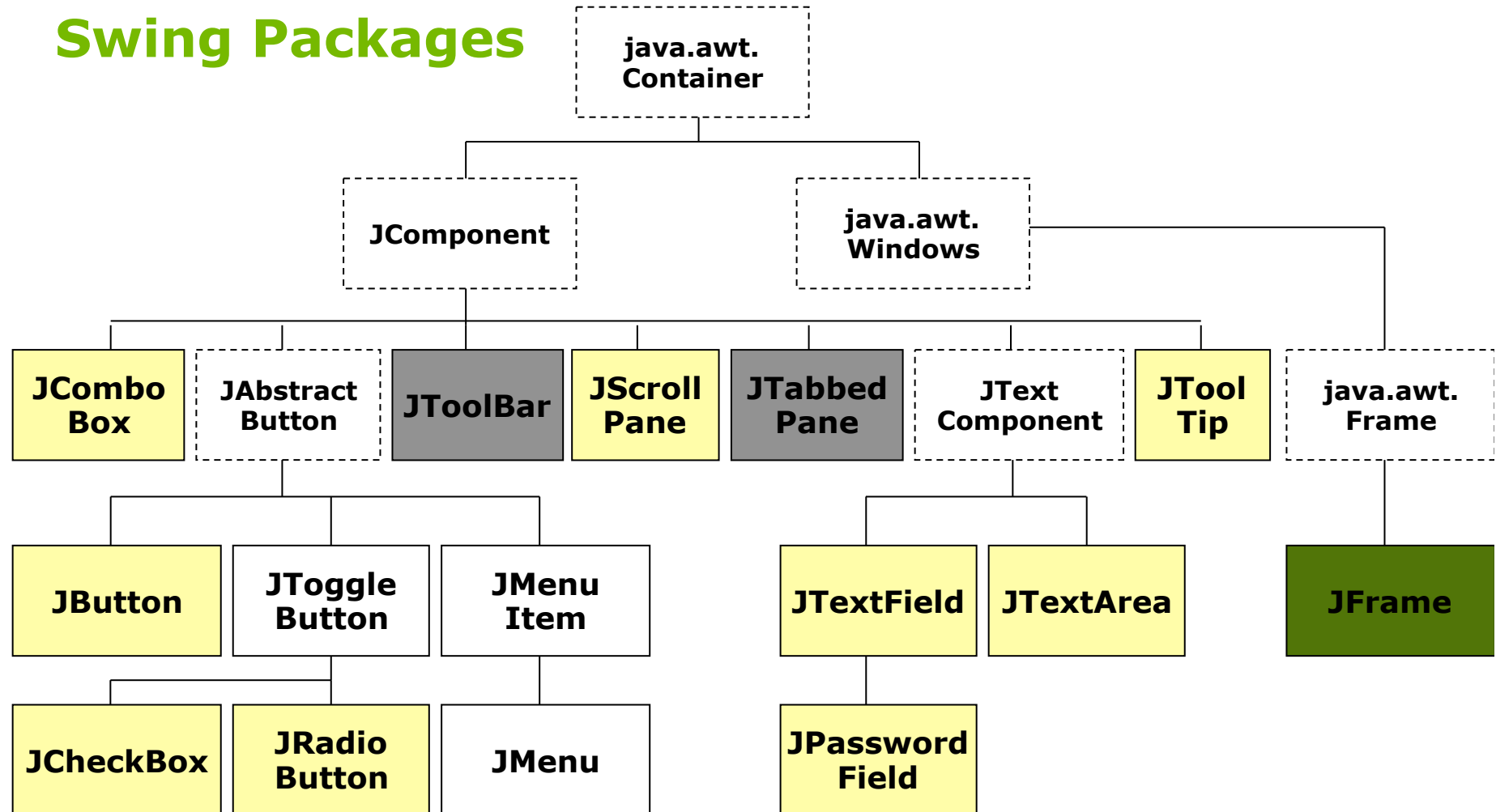
Look & Feel



Aufbau von Komponenten

- Top-Level-Containers
 - JFrame, JWindow, JDialog, JApplet
- Intermediate Containers
 - JPanel, JTabbedPane, JInternalFrame, JToolBar,...
- Atomic Components
 - JLabel, JButton, JScrollBar, JCheckBox,...

Swing Packages



Atomic Komponenten

Intermediate Container

Top-Level Container

Abstrakte Klassen



Wichtige Packages

`java.awt.*`

auf Grund Vererbung AWT

`java.awt.event.*`

zur Ereignisbehandlung

`javax.swing.*`

Standardkomponenten

`javax.swing.text.*`

Umgang mit editierb. Text

`javax.swing.border.*`

Grenzen von Komponenten

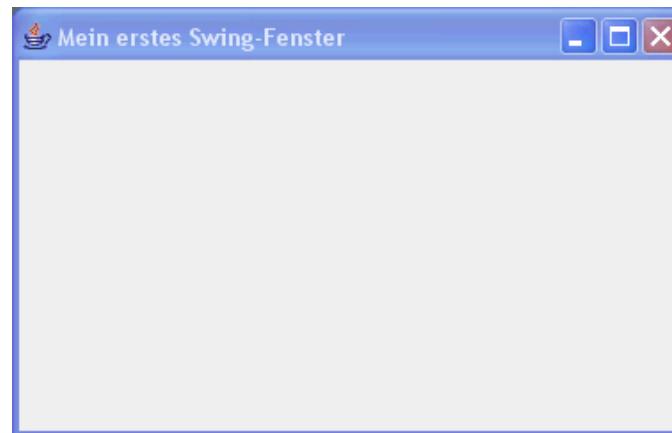
`javax.swing.plaf.*`

plugable look and feel

...

JFrame

- wichtigste Hauptfensterklasse in Swing
- bildet eigenständiges Fenster
- Hauptfenster mit Rahmen, Systemmenü und Standardschaltflächen



JFrame

Ableitung einer Fensterklasse von JFrame

class MeinFenster extends JFrame

Konstruktoren

- *JFrame();*
Erzeugt leeres Fenster ohne Titel
- *JFrame(String title);*
Erzeugt leeres Fenster mit Titel

JFrame

Wichtige Methoden

- *setLayout(Layoutmanager manager);*
Layout setzen
- *setSize(int, int);*
Grösse des Frame festlegen
- *setVisible(boolean);*
Sichtbarkeit festlegen
- *setJMenuBar(JMenuBar);*
platziert Menü auf Fenster

Grundgerüst für GUI mit Swing über JFrame

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

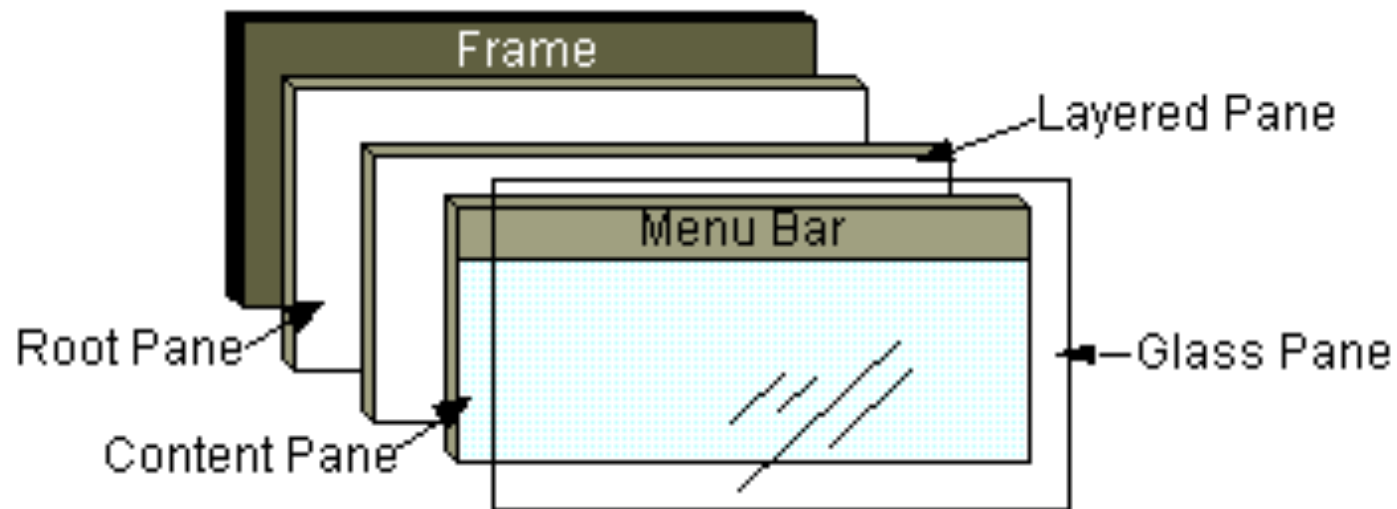
public class GrundgeruestSwing extends JFrame {

    public GrundgeruestSwing() {
        super(); // Konstruktor von JFrame
        setTitle("Titel Swing-Fenster"); // Titel des Fensters
        getContentPane().setBackground(Color.WHITE); // Hintergrundfarbe
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Klick auf x
    }

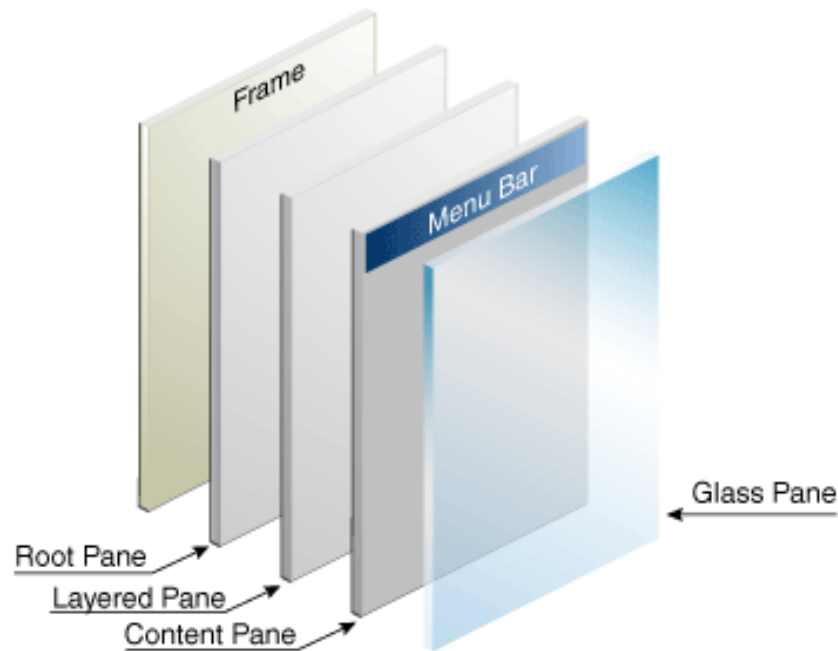
    public static void main(String args[]){
        GrundgeruestSwing hauptfenster = new GrundgeruestSwing();
        hauptfenster.setSize(400,300); // wie groß?
        hauptfenster.setLocation(200,300); // wo ?
        hauptfenster.setVisible(true); // sichtbar
    }
}
```

Aufbau eines JFrame

Ein JFrame wird durch mehrere „Layer“ dargestellt.



```
getContentPane().setBackground(Color.WHITE);
```

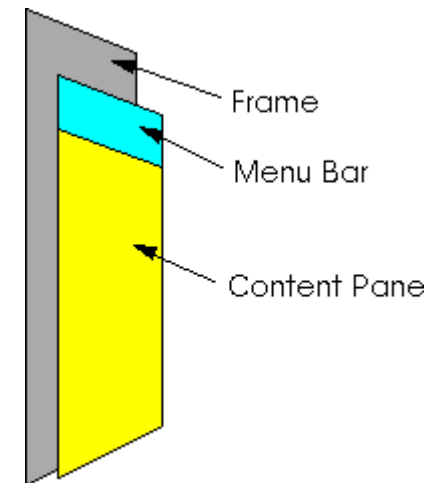


ContentPane:
repräsentiert das sichtbare
Innere eines Fensters

Komponenten auf das Fenster legen

Komponenten werden nicht direkt auf das Fenster gelegt, sondern:

- Platzierung mit ***getContentPane().add(Komponente)***
- Layoutänderung mit ***getContentPane().setLayout(...)***
- bei Nichtbeachtung Exception



Beispiel:

```
JFrame myFrame = new JFrame ();  
myFrame.getContentPane().add(new JLabel("xyz"));
```

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- DO_NOTHING_ON_CLOSE
 - macht nichts (außer, man hat die windowClosing()-Methode des Window-Listeners programmiert)
- HIDE_ON_CLOSE
 - lässt das Fenster vom Bildschirm verschwinden (man kann es aber wieder sichtbar machen)
- DISPOSE_ON_CLOSE
 - das Fenster verschwindet und alle seine Ressourcen (wie EXIT_..., wenn es nur ein Fenster gibt)
- EXIT_ON_CLOSE
 - das Programm wird beendet und somit alle zugehörigen Fenster und Ressourcen (System.exit(0))

JPanel

`javax.swing.JPanel`

- Standardcontainer
- Unterklasse von **JComponent**
- wir verwenden **JPanel**:
 - 1. dem Panel werden Komponenten (Container oder Steuerelemente) hinzugefügt (**add**)
 - 2. das Panel wird an das Fenster angefügt (**add**)
 - optional: dem Panel wird ein Layout-Manager zugewiesen (zur Anordnung der Komponenten im Panel)

Wir erweitern unser Grundgerüst

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

public class GrundgeruestSwing extends JFrame {

    public GrundgeruestSwing() {
        super(); // Konstruktor von JFrame
        setTitle("Titel Swing-Fenster"); // Titel des Fensters
        getContentPane().setBackground(Color.WHITE); // Hintergrundfarbe
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Klick auf x
        JPanel hauptPanel = init(); // eigene Methode init()
        this.add(hauptPanel); // Hauptpanel dem Fenster hinzufügen
    }

    private JPanel init()
    {
        JPanel panel = new JPanel();
        // hier Komponenten hinzufügen
        return panel;
    }
}
```



Wir erweitern init()

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

    JLabel label1 = new JLabel("Hallo FIW!");           // Label erzeugen
    JButton button1 = new JButton("Klick mich");         // Button erzeugen
    JButton button2 = new JButton("Ende");               // Button erzeugen

    panel.add(label1);                                   // Label ans Panel
    panel.add(button1);                                  // Button ans Panel
    panel.add(button2);                                  // Button ans Panel

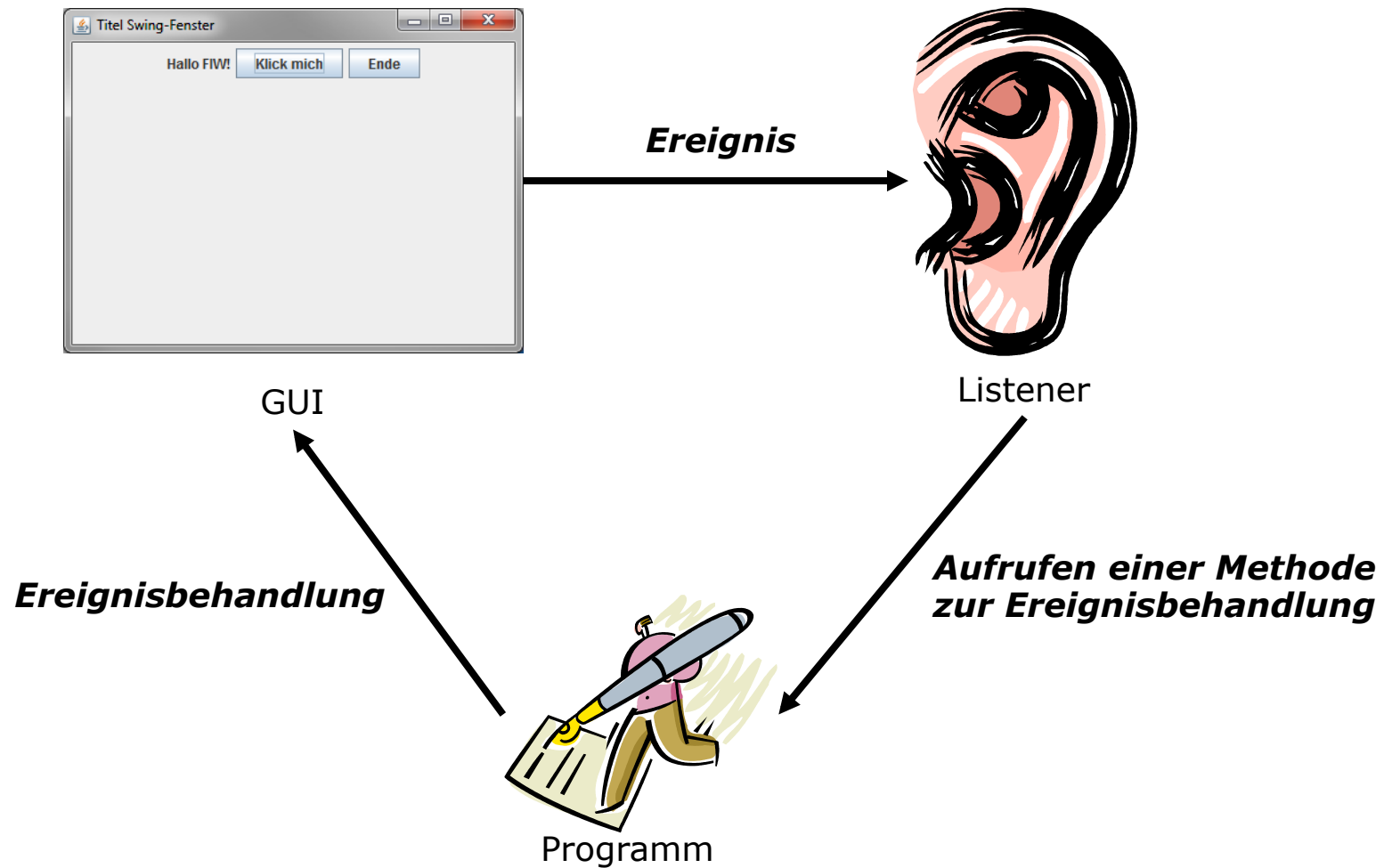
    return panel;
}
```

- Standard-Layout-Manager von **JPanel** ist **FlowLayout** -> alle Komponenten nebeneinander
- später lernen wir weitere LayoutManager kennen



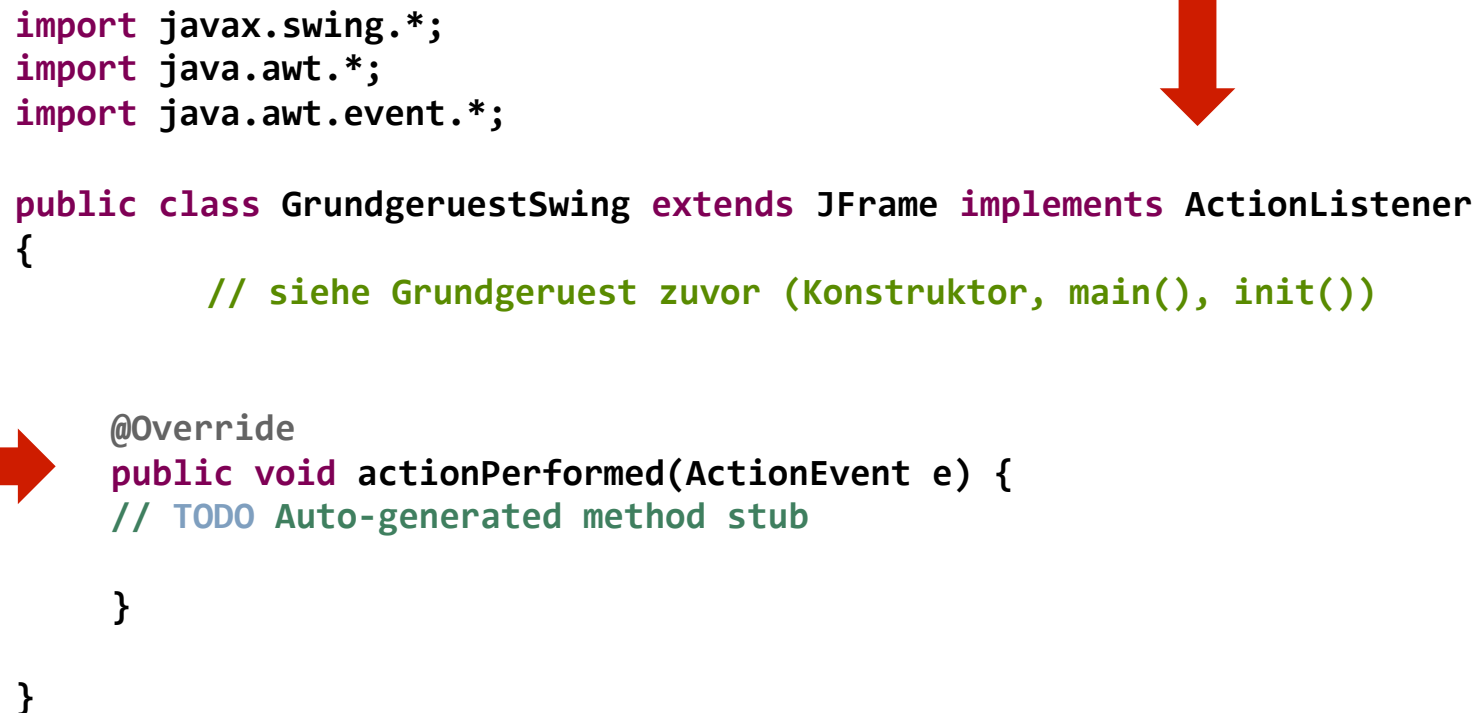
Ereignisse

Macht der Nutzer etwas?



Listener sind Interfaces

Beispiel ActionListener



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

public class GrundgeruestSwing extends JFrame implements ActionListener
{
    // siehe Grundgeruest zuvor (Konstruktor, main(), init())

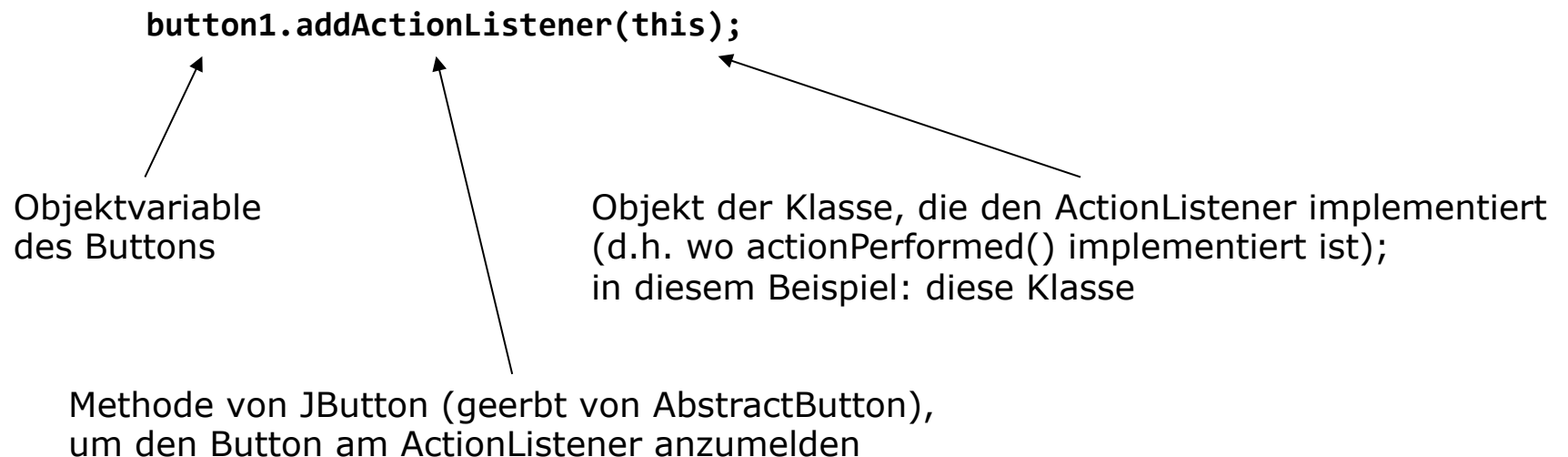
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
    }
}
```

- Eclipse:
 - schreiben Sie „implements ActionListener “ → import java.awt.event.*; ->
 - nutzen Sie „Add unimplemented methods “

Listener sind Interfaces

Beispiel ActionListener

- `actionPerformed(ActionEvent e)` ist die Methode, die aufgerufen wird, wenn eine Aktion (ein Ereignis) ausgelöst wird (z.B. Klicken eines Buttons)
- in diese Methode können wir nun schreiben, was passieren soll, wenn das Ereignis ausgelöst wird -> Ereignisbehandlung (event handler)
- vorher muss aber jeder Button an den ActionListener angemeldet werden:



Listener sind Interfaces

Beispiel ActionListener

```
public void actionPerformed(ActionEvent event) {
    Object quelle = event.getSource();

    if(quelle == klickMichbutton) {
        // change color
        farbIndex++;

        if(farbIndex == farben.length) {
            farbIndex = 0;
        }

        hauptPanel.setBackground(farben[farbIndex]);
        //label.setText(farben[farbIndex].toString());
    }
    else if(quelle == endeButton) {
        // exit program
        System.exit(0);
    }
}
```

- siehe **Ereignisbehandlung.java** in Moodle

ActionListener

- der ActionListener hat nur eine Methode: `actionPerformed(ActionEvent e)`
- tritt ein `ActionEvent` auf, so wird diese Methode automatisch aufgerufen
- `ActionEvent` ist eine Klasse aus dem Paket `java.awt.event` und erbt von `java.awt.AWTEvent`
- das `ActionEvent` selbst wird als Parameter dem Methodenaufruf übergeben
- `getSource()` ist eine Methode von `ActionEvent` (geerbt von `java.util.EventObject`) und liefert das Objekt (Typ `Object`) zurück, das das `ActionEvent` ausgelöst hat
- durch das Implementieren von `actionPerformed()` wird das durch den ActionListener ausgelöste Ereignis (`ActionEvent`) behandelt
→ dazu ist es aber notwendig, dass die Objekte, die durch den ActionListener abgehört werden sollen, beim ActionListener angemeldet werden (`addActionListener()`)
- brauchen wir auch bei Menüs, Textfeldern, Dateiauswahlfenstern, ...

Model-View-Controller (MVC)

- Swing-Komponenten basieren auf dem Entwurfsmuster Model-View-Controller:
 - *Models*: Kapselung und Verarbeitung der Daten (Modell, Daten, Berechnungen)
 - *Views*: visuelle Darstellungen der Daten (Grafik, Ansicht)
 - *Controller*: Behandlung der Ereignisse (Steuerung)
- Beispiele:
 - Model: Inhalte von Listen und Menüs, Daten für Diagramme usw.
 - View: Grafische Oberfläche, visuelles Erscheinungsbild (Look & Feel)
 - Controller: auf welche Eingaben werden reagiert; und wie wird reagiert?

... zunächst weiter mit *View*

JFrame

- **JFrame**
 - der äußere Rahmen einer Swing-Anwendung
 - siehe `javax.swing.JFrame`
 - hat
 - Titelleiste mit den drei bekannten Schaltflächen (decorated); `setUndecorated(boolean v)`
 - einen Rahmen (`javax.swing.Border`);
 - Sichtbarkeit; `setVisible(boolean v)`
 - Transparenz; `setOpacity(float f)` (0 durchsichtig; 1.0 undurchsichtig)
 - eine Größe; `setSize(int b, int h)`;
 - eine Position; `setLocation(int x, int y)`
 - eine Form; `setShape(Shape s)`
 - kann
 - im Vordergrund sein; `toFront()`
 - im Hintergrund sein; `toBack()`



JFrame – „Schema F“

```
class SwingFenster extends JFrame{
    // Instanz- oder Membervariablen

    public SwingFenster(){ // Default-Konstruktor
        this("Mein erstes Swing-Fenster"); // Aufruf des eigenen Konstruktors mit Parametern
    }

    public SwingFenster( String titel ){ // Konstruktor mit Parameter
        super(titel); // Aufruf des Konstruktors der Oberklasse

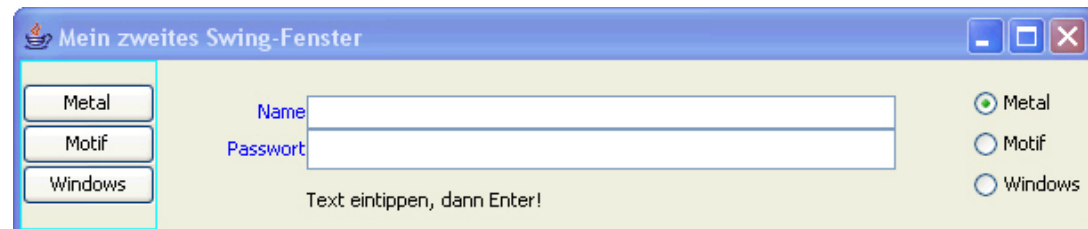
        Container cpane = this.getContentPane(); // ContentPane holen
        cpane.setLayout( new BorderLayout() );

        ...
        // Listener anbinden – nicht vergessen!
        ...
    } // end of constructor

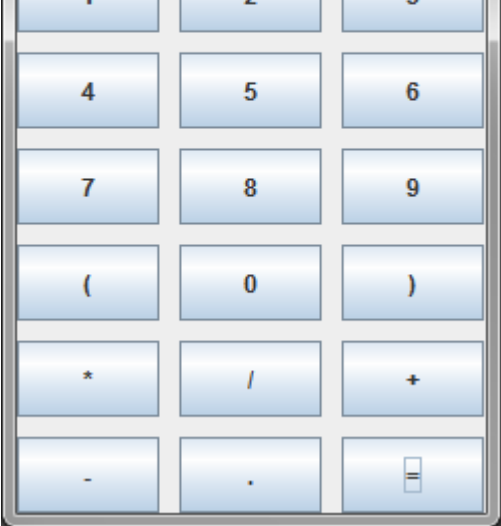
    // eigene Methoden

    // innere Klassen

} // end of class SwingFenster
```



Aufgabe

- Erstellen Sie ein Fenster mit folgender Sicht:
- 
- Melden Sie alle Buttons am ActionListener an und implementieren Sie den ActionListener (actionPerformed()) so, dass das Zeichen, das auf dem Button steht, im oberen Textfeld (JTextField) erscheint
 - Zusatzaufgabe: Führen Sie Berechnungen aus.