

Programmieren II

Grafische Benutzeroberflächen mit Swing



Heusch 2 (2. Bd)
Ratz 13, 14

Institut für Angewandte Informatik

```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

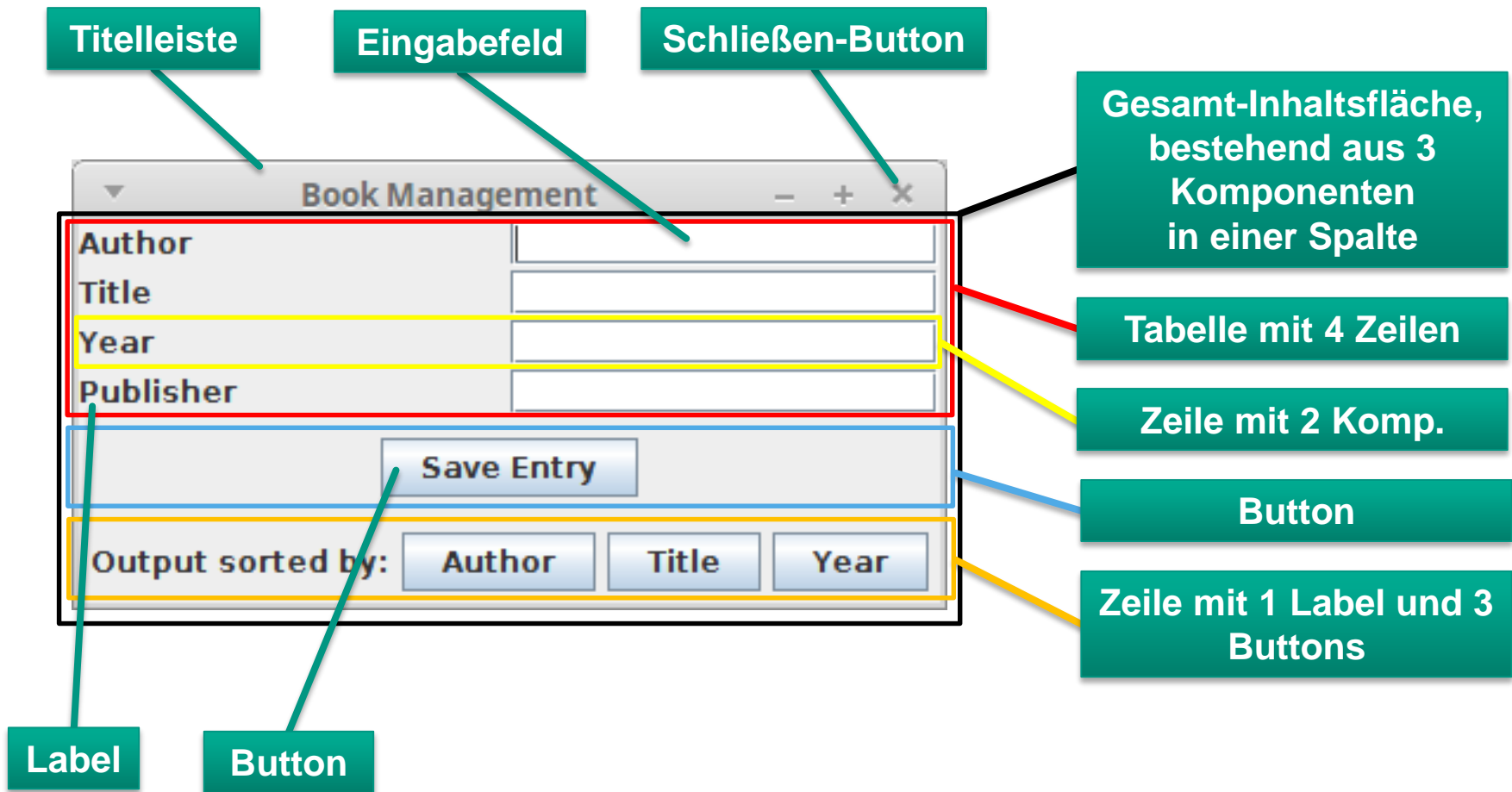
Grundlegendes zum Aufbau grafischer Benutzeroberflächen in Frames (1)

- Der Aufbau grafischer Benutzeroberflächen (Graphical User Interface, GUI) für Applikationen erfolgt nach einem hierarchischen Baukastenprinzip. Bestandteile:
 - **(GUI-)Grundkomponenten:** Einfache Oberflächenelemente wie z.B. Beschriftungen (Labels), Knöpfe (Buttons), Auswahlmenüs (Choice box) und Schiebebalken (Scroll bar).
 - **Container:** Behälter, die Komponenten enthalten. Komponenten können Grundkomponenten sowie selbst wieder Container sein.
- **Top-Level-Container:** Der Basis-Container eines Fensters. Z.B. ein Frame, das aus Titelleiste, Inhaltsfläche (Content Pane) und Fensterrand besteht
- Für die Anordnung und Gestaltung der einzelnen Komponenten sind **Layout-Manager**, **Farben** und **Fonts** zuständig.

Grundlegendes zum Aufbau grafischer Benutzeroberflächen in Fenstern (2)

- Der Ablauf von Programmen mit grafischer Benutzeroberfläche wird über Ereignisse und Empfänger gesteuert :
 - **Ereignisse** (Events) sind Aktionen, die der Benutzer beim Arbeiten mit der grafischen Oberfläche auslösen kann, indem er z.B. eine Schaltfläche drückt.
 - **Empfänger (Listener)** dienen als Empfänger von Ereignissen. Sie regeln, wie das Programm auf ein Ereignis reagieren soll, und müssen vom Programmierer erstellt werden.

Beispiel für Aufbau eines Fensters



Programm siehe Folie „Beispiel für geschachteltes Layout“

Graphische Benutzeroberflächen mit Java

- Zur Entwicklung graphischer Benutzeroberflächen bietet Java von Haus aus entsprechende Bibliotheken an:
 - Abstract Windowing Toolkit (AWT) – Teile veraltet –
 - **Swing (Java Foundation Classes JFC)**
 - Standard Widget Toolkit (SWT) – nicht im Java-Standard –
 - JavaFX – im Java-Standard seit Java 8

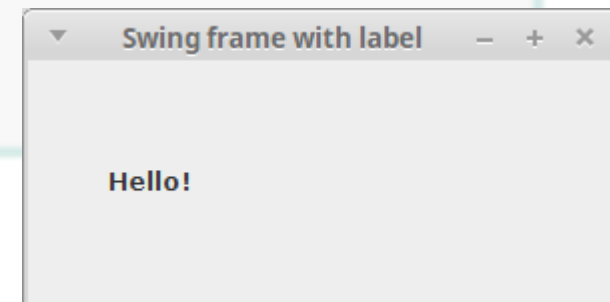
- Diese sind grundsätzlich plattformunabhängig, unterscheiden sich jedoch u. a. in
 - ihrer Mächtigkeit und
 - Abstraktion vom zugrunde liegenden (Betriebs-)System.

Einführendes Beispiel für Swing

- Erzeugen eines Fensters mit Label (Text)
- Geeignete Klasse in Swing: `JFrame`

```
import javax.swing.*;

public class LabelFrame1 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Swing frame with label");
        frame.add(new JLabel("Hello!"));
        frame.setSize(300, 150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



Anmerkung: Das Programm kann über die Schaltfläche X beendet werden.

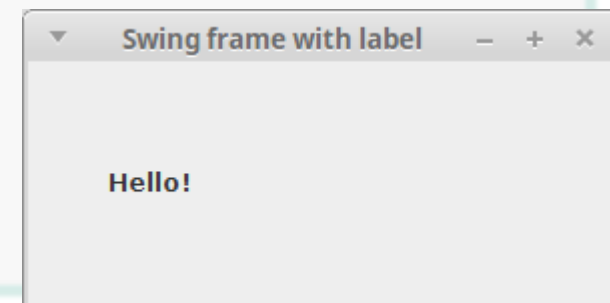
Alternative Implementierungen (1)

```
import javax.swing.*;

public class LabelFrame2 {
    JFrame frame;

    public LabelFrame2() {
        this.frame = new JFrame();
        this.frame.setTitle("Swing frame with label");
        this.frame.add(new JLabel("Hello!"));
        this.frame.setSize(300, 150);
        this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.frame.setVisible(true);
    }

    public static void main(String[] args) {
        new LabelFrame2();
    }
}
```



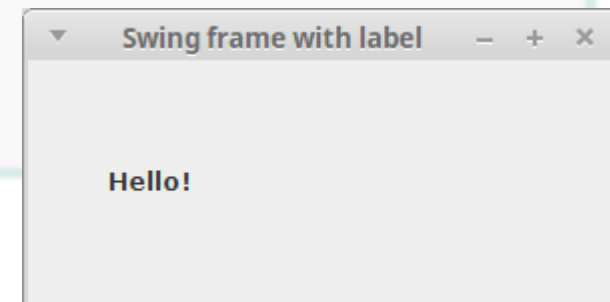
Alternative Implementierungen (2)

```
import javax.swing.*;

public class LabelFrame3 extends JFrame {

    public LabelFrame3() {
        this.setTitle("Swing frame with label");
        this.add(new JLabel("Hello!"));
        this.setSize(300, 150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        new LabelFrame3();
    }
}
```



Weiteres Vorgehen

- Im Folgenden werden nacheinander behandelt:
 - Grundlegendes zu AWT, Swing, SWT und JavaFX sowie Vergleich
 - Detaillierungen zu Swing (und AWT):
 - Übersicht über Komponenten und Klassenhierarchie
 - Container
 - Grundkomponenten
 - Layoutmanager
 - Standard-Dialoge

- In einem folgenden Kapitel werden dann Ereignisse (Events) und deren Verarbeitung (Listener) behandelt.

Abstract Windowing Toolkit (AWT) (1)

- Das Abstract Windowing Toolkit (AWT) verwendet nur graphische Interaktions-Komponenten (Widgets), die auf „allen“ Plattformen unterstützt werden.
 - Komponenten wie Bäume oder Tabellen sind im AWT nicht vorhanden.
- Das AWT ist im Grunde eine *einheitliche objektorientierte Schicht* über dem jeweiligen nativen Fenstersystem der Plattform und stellt diese Schicht über "Peers" innerhalb der Plattform bereit.
 - Peer-Ansatz: Jede AWT-Klasse hat eine „Partnerklasse“ auf der darunter liegenden Betriebssystemseite

Abstract Windowing Toolkit (AWT) (2)

- Über einen LayoutManager ist es möglich, die einzelnen GUI-Komponenten relativ zu platzieren.
 - Da diese Komponenten aber je nach Betriebssystem ein unterschiedliches Aussehen besitzen, kann die Oberfläche beim *einem* Betriebssystem optimal aussehen, bei einem *anderen* aber inakzeptabel sein.
- AWT-Komponenten unterliegen wie andere Objekte der Garbage Collection.
 - Dadurch werden die vom Programm nicht mehr benötigten Ressourcen von der JVM freigegeben, ohne dass der Programmierer dies steuern muss.

Zur Historie:

- **AWT** war, mit 6 Wochen Entwicklungszeit bis zur ersten Release, eine Schnellgeburt und hatte dementsprechend einige Schwächen.
- Diese wurden durch die **Internet Foundation Classes (IFC)** von Netscape zunächst bereinigt.
- Um eine Spaltung der Java-Gemeinde zu vermeiden, schloss sich Sun mit Netscape und anderen Unternehmungen zu einem Implementierungskonsortium zusammen.
- Ergebnis waren die **Java Foundation Classes (JFC)**, die das Swing-Framework enthalten, und die inzwischen in den Java-Standard aufgegangen sind.

- Das Framework **Swing** bzw. die **JFC** stellt dem GUI-Entwickler das Beste von AWT und IFC in einer umfassenden Werkzeugsammlung zur Verfügung.
- In Swing werden die graphischen Komponenten in Heavyweight- und Lightweight-Komponenten unterschieden.
 - Unter **Heavyweight-Komponenten** versteht man Anwendungsfenster und Dialogfenster, die wie in AWT einen Peer als Implementationshilfe besitzen.
 - **Lightweight-Komponenten** hingegen (`JButton`, `JTextField` usw.) wurden vollständig in Java implementiert und müssen von Heavyweight-Komponenten zu ihrer Darstellung aufgenommen werden.

Java Foundation Classes (1)

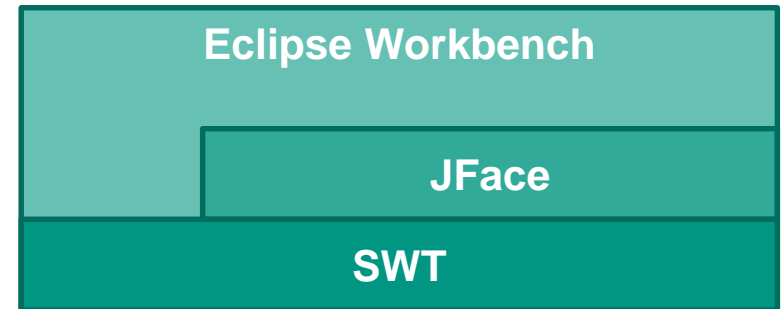
- **Swing-GUI-Komponenten.**
- **Pluggable Look&Feel:** Möglichkeit, das Aussehen der Komponenten zur Laufzeit zu ändern, ohne das Programm neu zu starten.
Alle Komponenten des Swing-Sets haben diese Fähigkeit automatisch.
- **Accessibility:** Unterstützung für Menschen mit Behinderungen.
Diese API erlaubt mit neuen Interaktionstechniken Zugriff auf die JFC- und AWT-Komponenten. Zu diesen Techniken zählen unter anderem Lesegeräte für Blinde, eine Lupe für den Bildschirm und auch Spracherkennung.

Java Foundation Classes (2)

- **Java 2D-API:** Die 2D-Funktionsbibliothek ist eine neue Technik, die über eine Objektbeschreibung – ähnlich wie Postscript – Objekte bildet und diese auf dem Bildschirm darstellt.
Zu den Fähigkeiten der Bibliothek gehört es, komplexe Objekte durch Pfade bilden zu können und darauf Bewegungs- und Verschiebeoperationen anzuwenden.
- **Drag&Drop:** Daten können mittels Drag&Drop leicht von einer Applikation zur anderen übertragen werden.
Dabei profitieren Java-Programme auch davon, Daten zu nutzen, die nicht aus Java-Programmen stammen.

Standard Widget Toolkit (SWT) und JFace

- Bei der Entwicklung der (ersten) Eclipse-Plattform entschieden sich das Team von OTI (Object Technology International) und IBM gegen die JFC.



Stattdessen entschlossen sie sich, zum Erstellen der GUI eine eigene OpenSource-API zu entwickeln.

- Diese neue API trägt den Namen **Standard Widget Toolkit** (SWT) und stellt GUI-Komponenten (Widgets), wie Buttons, Bäume oder Tabellen über native Komponenten des Betriebssystems zur Verfügung.
- **JFace** ist eine Sammlung von Klassen, um SWT zu nutzen.
- SWT wird im Rahmen dieser Vorlesung nicht behandelt.

JavaFX (1)

- JavaFX (Entwickler: Oracle) ist ein Framework für „reichhaltige“ (rich) Internet-Anwendungen. Diese können aber auch als Desktop-Anwendungen eingesetzt werden.
- Das Aussehen der Elemente des Benutzer-Interfaces kann mit Hilfe von Style Sheets (JavaFX CSS) festgelegt werden.
- Mit einer XML-basierten Sprache (FXML) kann der Aufbau des Benutzer-Interfaces separat von der Applikationslogik des Codes beschrieben werden (MVC-Konzept).
- Mit Java 8 wurde das bereits in Java 7 / Update 6 eingeführte JavaFX offiziell in die Java SE integriert.

JavaFX (2)

- Von Oracle ist angekündigt, dass JavaFX Swing ablösen soll.
- Es soll die Standardtechnologie für die Erstellung von grafischen Oberflächen werden, nicht nur für Web-Anwendungen sondern auch im Desktop-Bereich wie auch bei mobilen Java-Applikationen.
- Die aktuelle Version 8 bietet aber noch nicht alle Swing-Funktionen.
- Inwieweit JavaFX sich wirklich durchsetzen wird, muss sich erst noch erweisen.
- Im Rahmen dieser Vorlesung wird JavaFX ebenfalls nicht behandelt.

Swing-Komponenten – Allgemeines

- Auf den nächsten Folien werden ganz überwiegend Swing-Komponenten (**Java Foundation Classes JFC**) behandelt.
- Diese verwenden keine Peers vom Betriebssystem, sondern sind komplett in Java definiert.
- Man kann das Aussehen und die Funktionalität (Look&Feel) für diese Komponenten im Java-Programm genau festlegen:
 - entweder über vorgefertigten Designs,
 - oder indem man alle Details selbst festlegt, und damit ein produktspezifisches Look&Feel erzeugt.

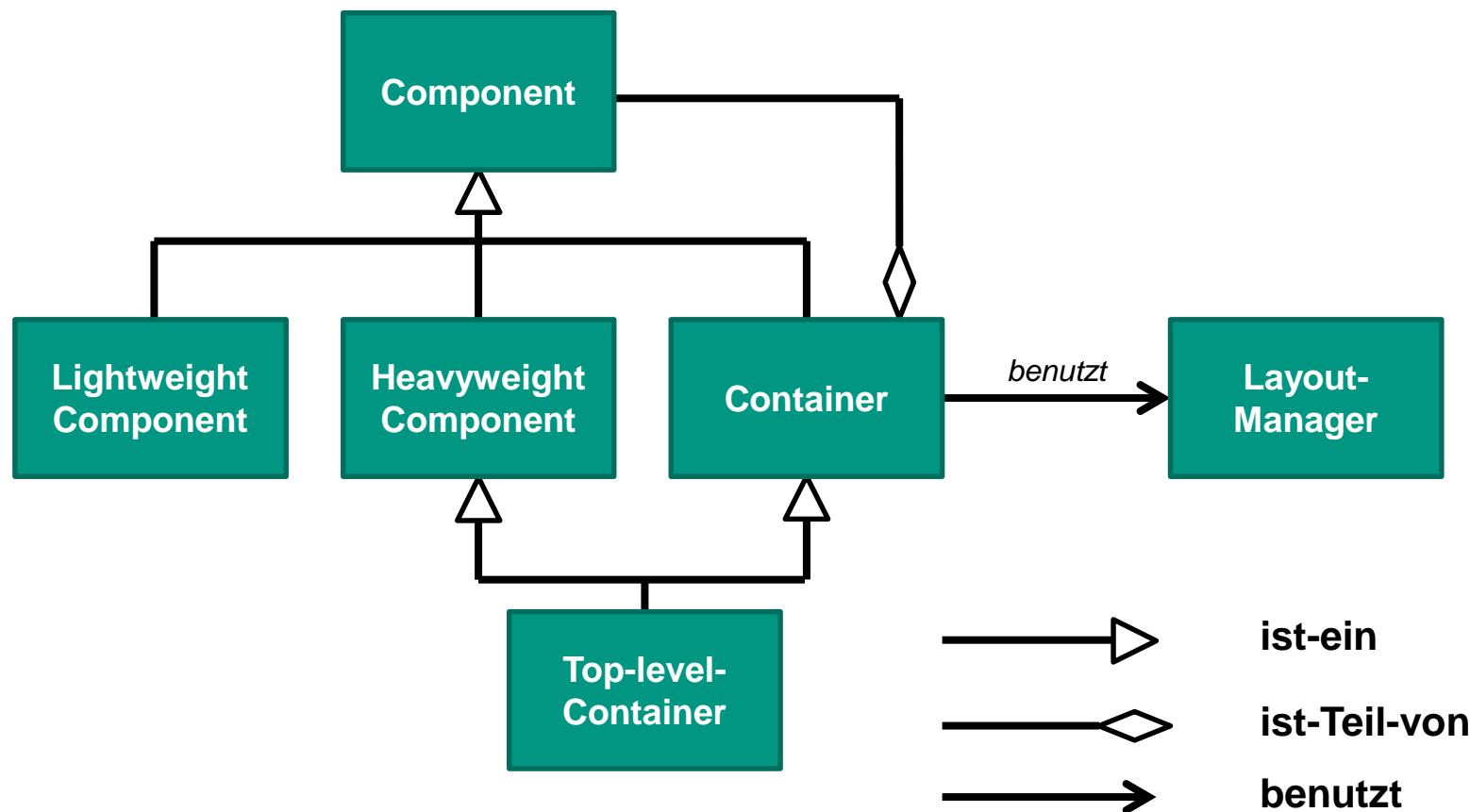
Swing-Komponenten, Übersicht und Zusammenhang (1)

Die Elemente von Swing können grob untergliedert werden in:

- **Container** sind Swing-Elemente, die dazu dienen, Komponenten aufzunehmen und zu verwalten.
- Container der obersten Ebene werden **TOP-Level-Container** genannt.
- **Komponenten** können Grundkomponenten und andere Container sein.
- Ein Container setzt die Komponenten mit Hilfe eines **Layoutmanagers** in die richtige Position.

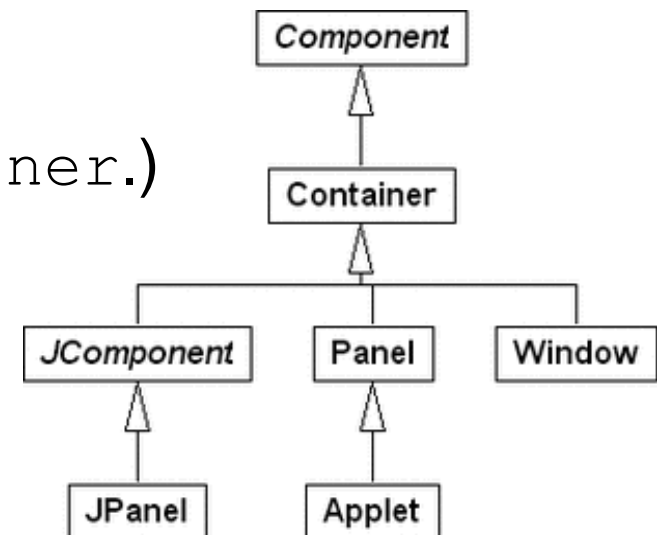
Swing-Komponenten, Übersicht und Zusammenhang (2)

- So hängen die einzelnen Swing-Komponenten zusammen:



Die JAVA-Klassen Component, JComponent und Container (1)

- Die AWT-Klasse `Component` bildet die Basisklasse der Objekte, die als grafische Komponenten auf den Schirm kommen.
- Sie wird für Swing-Komponenten zu `JComponent` erweitert. Allerdings leitet `JComponent` nicht direkt von `Component` ab, sondern von `Container`, und damit indirekt von `Component`.
(Jede `JComponent` ist damit automatisch auch ein (AWT-)Container.)



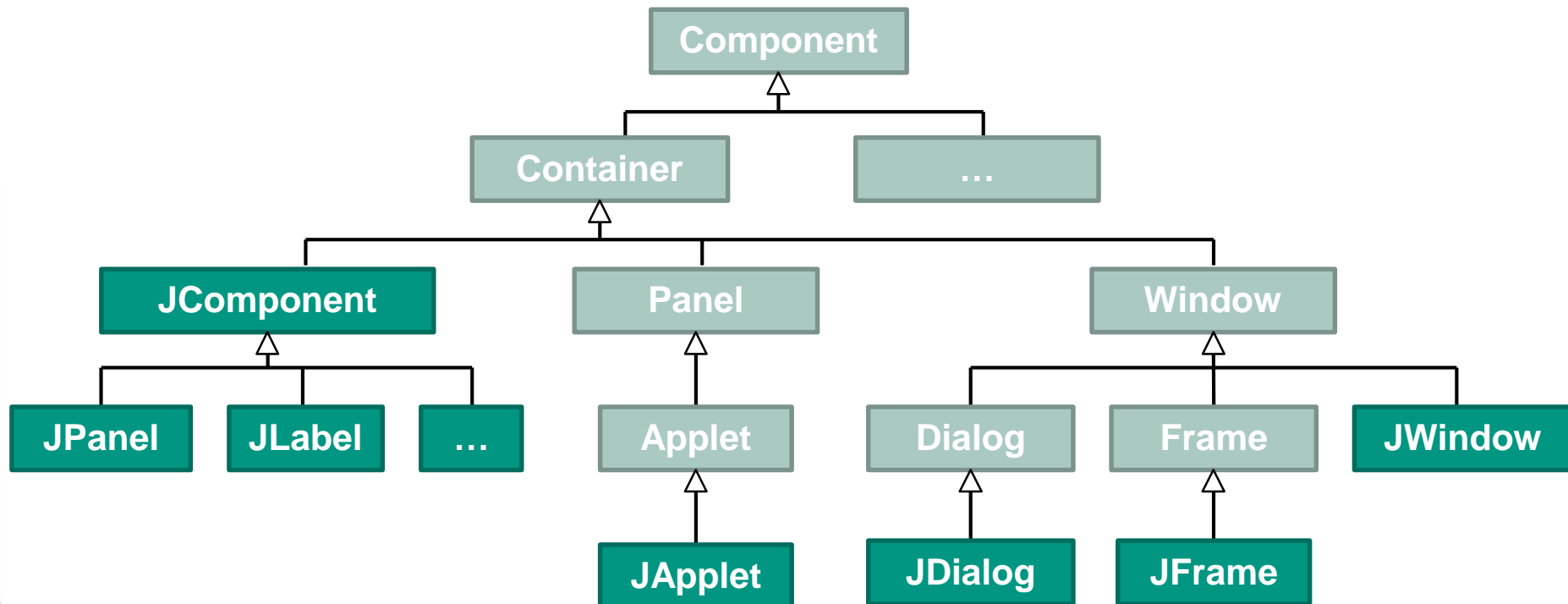
Die Klassen Component, JComponent, Container (2)

- Alle Container in Java erweitern die Klasse `Container`.
- Die Methode `add()` setzt Komponenten in den `Container`.

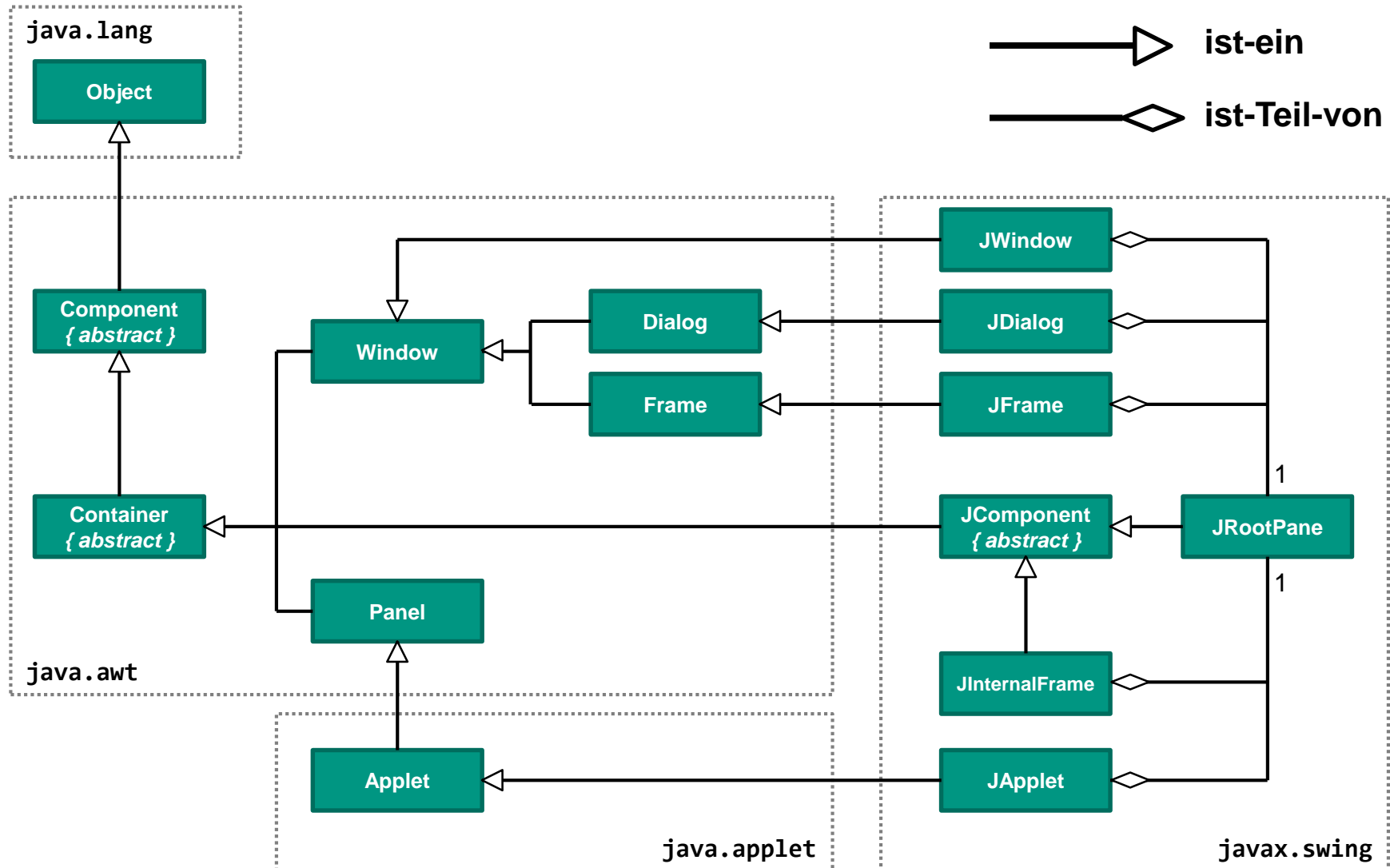
AWT

Swing

↑ Vererbung



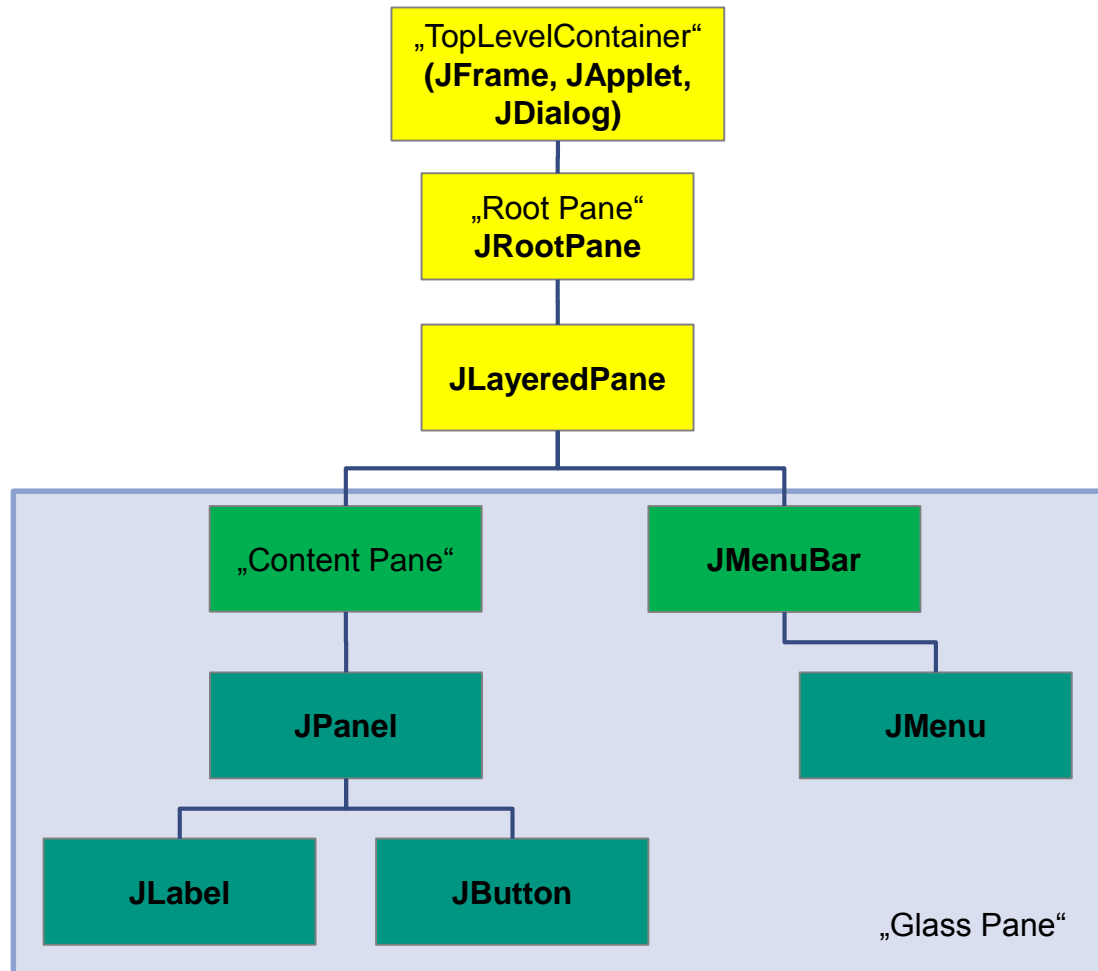
Paketzugehörigkeit



Top-Level-Container (1)

- Top-Level-Container (Container auf der obersten Ebene) können Objekte der folgenden Klassen sein:
 - `JFrame` für ein Fenster mit Rahmen
 - `JWindow` für ein Fenster ohne Rahmen
 - `JDialog` für ein Dialogfenster
 - `JApplet` für einen Applet-Anzeigebereich
- Diese Komponenten enthalten als *Wurzelkomponente* den Container `JRootPane`.
- Die `JRootPane` wiederum enthält (indirekt) ein `ContentPane`, den eigentlichen Container, in den die Komponenten eingefügt werden können.

Schichten eines Top-Level-Containers in Swing



Quelle: <http://de.wikibooks.org/wiki/Datei:TopLevelContainerHierarchy.jpg>

Methoden von Top-Level-Containern (1)

- Einige wichtige Instanzmethoden von `JFrame`, `JWindow` und `JDialog` (inkl. geerbten):
 - `void setTitle(String s)`
 - `void add(Component comp)`
Fügt Komponente zum Container hinzu.
 - `void setSize(int i1, int i2)`
 - `void pack()`
Bewirkt, dass die Fenstergröße an die bevorzugte Größe und das Layout seiner Komponenten angepasst wird.
 - `void setDefaultCloseOperation(int i)`
Setzt die Operation, die standardmäßig durchgeführt wird, wenn der Benutzer das "Schließen" des Frames veranlasst

Methoden von Top-Level-Containern (2)

- `void setVisible(boolean b)`
Macht das Fenster sichtbar oder verbirgt es, in Abhängigkeit vom Wert des Parameters b.
- `void dispose()`
Gibt alle systemeigenen Ressourcen des Fensters frei..

Hinzufügen von Komponenten zu einem JFrame

- Um ein JComponent-Objekt zu einem JFrame hinzuzufügen, muss es zu dessen Zeichenfläche, die »ContentPane« genannt wird, hinzugefügt werden.
- Eine ContentPane ist ein Container-Objekt:

```
Container con = frame.getContentPane();  
con.add(component);
```


- Dies lässt sich dann zu einer Zeile abkürzen:

```
frame.getContentPane().add(component);
```

- *Hinweis:*

Seit Java 1.5 ist dieser Umweg nicht mehr notwendig. Es geht nun auch direkt (s. auch folgende Beispiele):

```
frame.add(component);
```

 Ab Java 1.5

Schließen eines Fensters

- Wird ein Fenster geschlossen, verschwindet es in den Hintergrund, wird also geschlossen, aber die Applikation nicht beendet.
- Dieses Verhalten kann mit der Funktion `setDefaultCloseOperation(int)` geändert werden.
- Beispiele:

```
setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);  
setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);  
setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);  
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Einführendes Swing-Beispiel mit Button statt Label

```
import javax.swing.*;

public class ButtonFrame{
    JFrame frame;

    public ButtonFrame() {
        this.frame = new JFrame();
        this.frame.setTitle("Swing window with button");
        this.frame.add(new JButton("I'm a button!"));
        this.frame.setSize(300, 150);
        this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.frame.setVisible(true);
    }

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



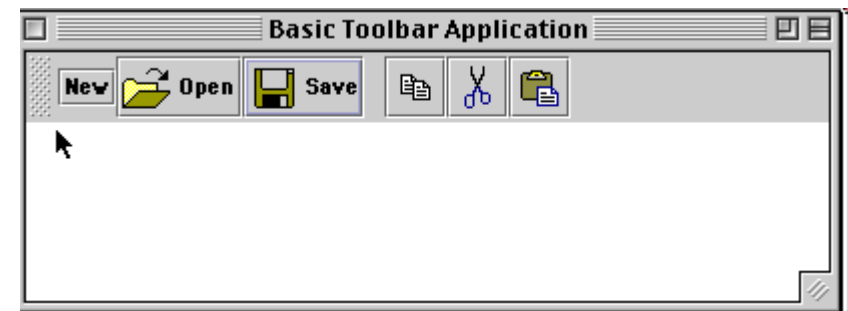
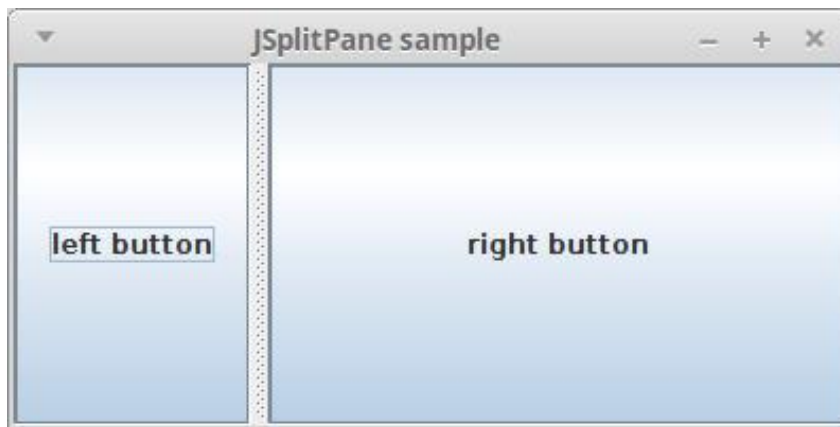
Weitere Container (1)

Zu den wichtigsten Containern in Swing zählen:

- `JPanel`: Ist im Wesentlichen eine `JComponent` mit der Möglichkeit, Kinder nach einem bestimmten Layout-Verfahren anzuordnen. („Unsichtbar“)
- `JScrollPane`: Kann Bereiche einer sehr großen Komponente mit Rollbalken anzeigen. Das ist von der Textverarbeitung bekannt, wenn der Text sehr lang ist, aber der Bildschirm viel kleiner.
- `JTabbedPane`: Zeigt Reiter in einem „Karteikasten“ an.
- `JInternalFrame`, `JDesktopPane`, `JTabbedPane`: Container für einen Bereich
- `JOptionPane`: Container für ein Dialogfenster

Weitere Container (2)

- `JSplitPane`: Ermöglicht die Darstellung zweier Komponenten über- oder nebeneinander, wobei ein so genannter Divider eine Größenveränderung erlaubt.
- Dazu kommen noch Container wie `JToolBar` und `JLayeredPane` (z.B. f. Desktops mit Fenstern).



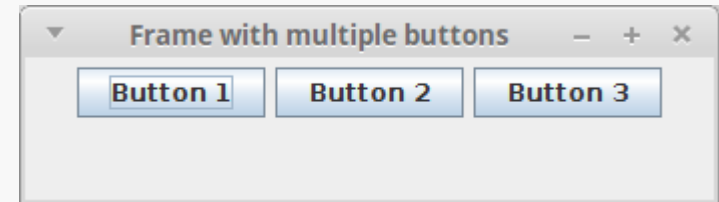
Einfaches Beispiel mit JPanel

```
import javax.swing.*;

public class MultiButton {

    public MultiButton() {
        JFrame frame = new JFrame();
        JPanel jp = new JPanel();
        for (int i = 1; i <= 3; i++){
            jp.add(new JButton("Button " + i));
        }
        frame.add(jp);
        frame.setTitle("Frame with multiple buttons");
        frame.setSize(350, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

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



Grundkomponenten (1)

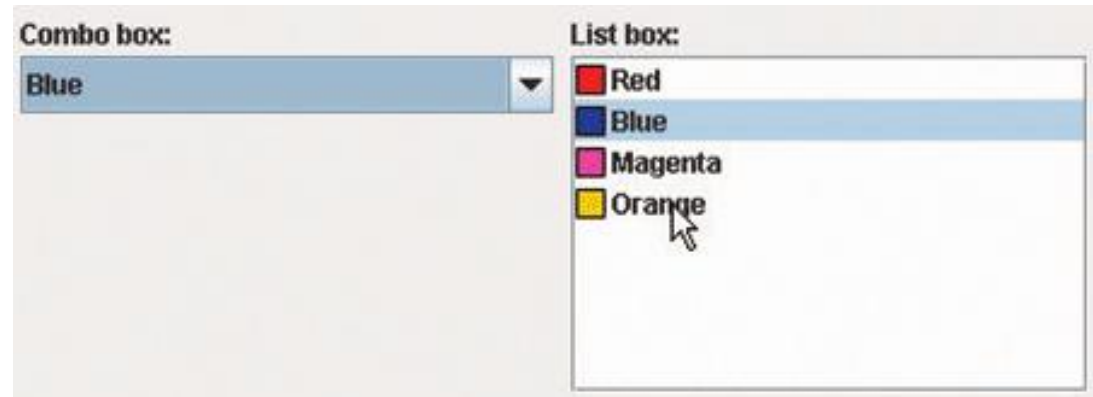
Wichtige Grundkomponenten von Swing sind:

- `JComponent` als Oberklasse für alle Swing-Komponenten (außer `Top-Level-Container`) und für eine Zeichenfläche
- `JLabel` für einen Text oder ein Icon
- `JButton` für einen Knopf mit Text oder Icon
- `JCheckBox`, `JRadioButton`, `JToggleButton` für einen Schalter
- `ButtonGroup` für die Gruppierung von `JRadioButton`- und `JComboBox`-Objekten so, dass stets *höchstens ein* Element pro Gruppe ausgewählt sein kann (d.h. für sich gegenseitig ausschließende Optionen).

Grundkomponenten (2)

- JComboBox, JList, in Verbindung mit einem ComboBoxModel bzw. ListModel und ListCellRenderer für Auswahlmenüs

Beispiel JComboBox
und JList

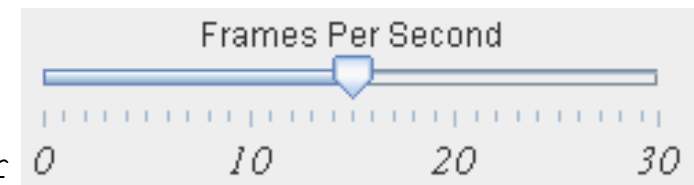


- JTextField, JPasswordField, JTextArea, in Verbindung mit String oder Document, für Text-Eingaben
- JFormattedTextField für validierende Texteingabefelder
- JFileChooser für die Auswahl einer Datei

Grundkomponenten (3)

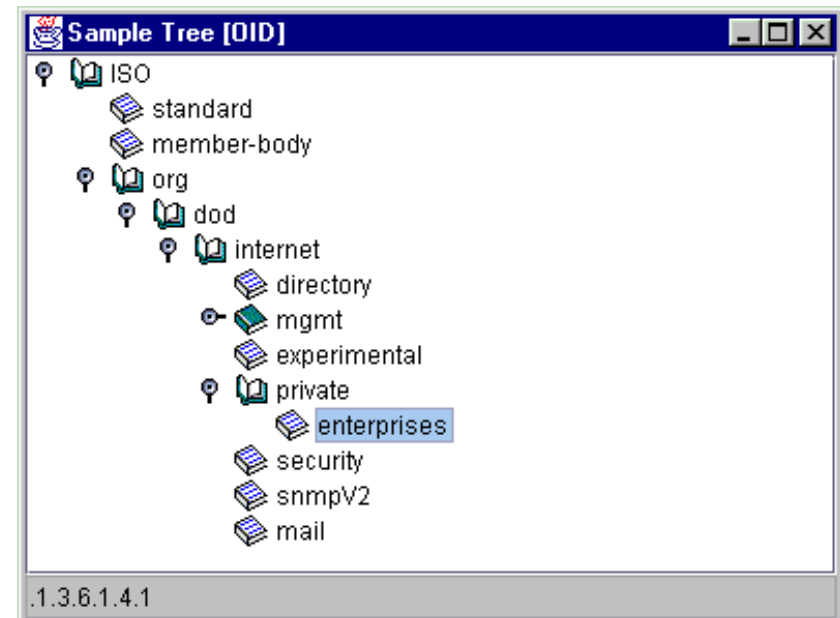
- `JMenuBar`, `JMenu`, `JMenuItem`, `JCheckboxMenuItem`, `JPopupMenu`, `JSeparator` für Menüs
- `JScrollBar`, `JScrollPane` für Scrollbars
- `ImageIcon` für ein kleines Bild (Direkte Unterklasse von `Object` – **keine** `JComponent`!)
- `JProgressBar` für die graphische Darstellung eines Zahlenwertes
- `JSlider` für die graphische Eingabe eines Zahlenwertes
- `JColorChooser` für die Auswahl einer Farbe

Beispiel `JSlider`



Grundkomponenten (4)

- `JToolTip` bzw. `setToolTipText()` für eine Zusatzinformation zu jeder Komponente
- `JTree` in Verbindung mit `TreeModel`, `TreePath`, `TreeNode` und `TreeCellRenderer` oder `TreeCellEditor` sowie `TreeSelectionModel` und `TreeSelectionListener` für die Darstellung eines hierarchischen Baums von Elementen wie z.B. ein Directory mit Subdirectories und Files



Grundkomponenten (5)

- `JTable` in Verbindung mit `TableModel` und `TableCellRenderer` für die Anordnung von Komponenten in Tabellenform
- `JEditorPane`, `JTextPane` für die formatierte Darstellung von Text.
- Hinweis: Swing-Komponenten sollten niemals mit AWT-Komponenten gemischt verwendet werden!

ÜBUNG (A1)

Beispiel für ein Menü

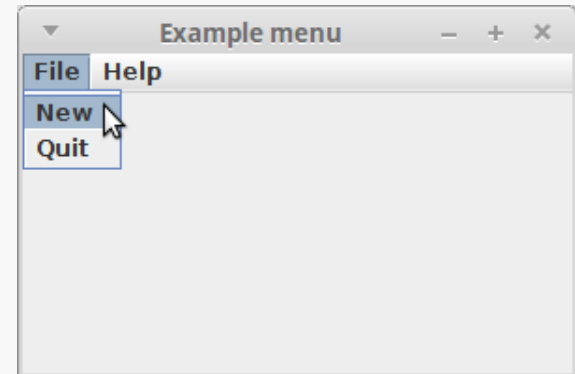
```
import javax.swing.*;

public class JMenuDemo1 extends JFrame {
    private JMenuItem newMenu, quit, info;

    public JMenuDemo1(String title) {
        super(title);
        JMenuBar menubar = new JMenuBar();
        this.setJMenuBar(menubar);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JMenu file = new JMenu("File");
        JMenu help = new JMenu("Help");
        menubar.add(file);
        menubar.add(help);
        this.newMenu = new JMenuItem("New");
        file.add(this.newMenu);
        this.quit = new JMenuItem("Quit");
        file.add(this.quit);
        this.info = new JMenuItem("Info");
        help.add(this.info);
        this.setSize(300, 200);
        this.setVisible(true);
    }

    public static void main(String[] args) { new JMenuDemo1("Example menu"); }
}
```



Das Java-Look&Feel

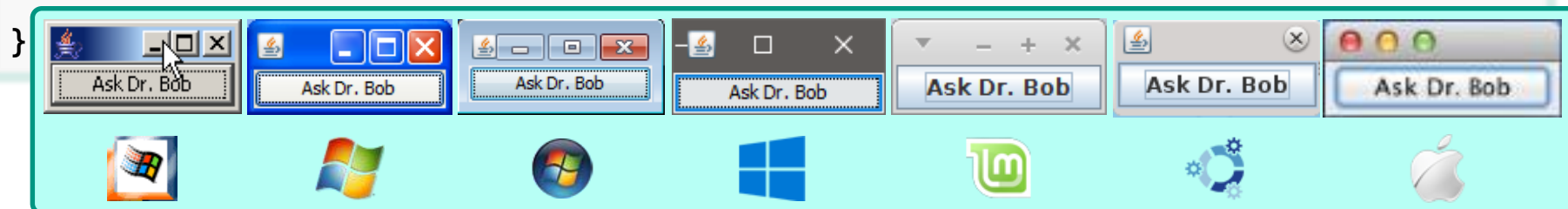
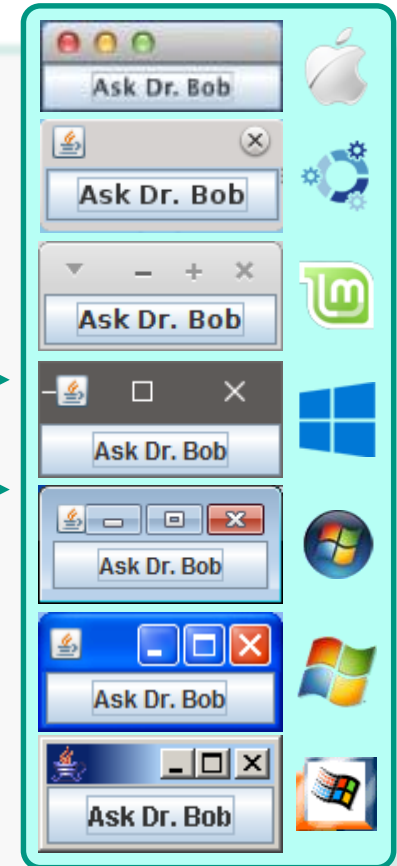
- Das Aussehen und Verhalten der GUI-Komponenten lässt sich über das flexible Look&Feel von Java einstellen.
- Das Look&Feel von Applikationen lässt sich zur Laufzeit ändern.
 - Dazu müssen wir nur eine statische Methode der Klasse `UIManagers` aufrufen, die sich um das Aussehen der Programme kümmert.
 - Hier ist es die spezielle Methode `setLookAndFeel()`, die als Parameter eine Klasse erwartet.
 - Verschiedene Methoden sind vordefiniert, mit denen wir das Java-eigene Look&Feel und das System-Look&Feel einstellen können.
 - Da Benutzer von Java-Programmen im Allgemeinen eine Oberfläche erwarten, die sie von ihrem System gewohnt sind, ist es sinnvoll, das Java-Look&Feel nach dem Erzeugen des Fensters umzuschalten.

Beispiel für Look&Feel

```
import javax.swing.*;

public class LookAndFeel {

    public static void main(String args[]) throws Exception {
        // Bei einer der folgenden Zeilenpaare die "//" entfernen!
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName() );
        // UIManager.setLookAndFeel(
        //     "javax.swing.plaf.metal.MetalLookAndFeel");
        // UIManager.setLookAndFeel(
        //     UIManager.getSystemLookAndFeelClassName());
        JFrame frame = new JFrame();
        frame.add(new JButton("Ask Dr. Bob"));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```



Layoutmanager

- Ein Layoutmanager ist dafür verantwortlich, Komponenten eines Containers nach einem bestimmten Verfahren anzuordnen, zum Beispiel zentriert oder von links nach rechts.
- Jeder Layout-Manager implementiert eine unterschiedliche Strategie zur Anordnung.
- Ein Container fragt bei einer Neudarstellung immer seinen Layoutmanager, wie er seine Komponenten anordnen soll.
- Das Layout eines Containers kann mit der Methode `void setLayout(LayoutManager mgr)` der Klasse `Container` *gesetzt* werden.

Übersicht über Layoutmanager (1)

- **FlowLayout:** Ordnet Komponenten zeilenweise von links nach rechts an.
- **BoxLayout:** Ordnet Komponenten horizontal oder vertikal an.
- **GridLayout:** Setzt Komponenten in ein Raster, wobei jedes Element die gleichen Ausmaße besitzt.
- **BorderLayout:** Setzt Komponenten in vier Himmelsrichtungen oder in der Mitte.
- **GridBagLayout:** Sehr flexibler Manager als Erweiterung von GridLayout.
- **CardLayout:** Verwaltet Komponenten wie auf einem Stapel, von dem nur einer sichtbar ist.

Übersicht über Layoutmanager (2)

- **SpringLayout:** Berücksichtigt Abhängigkeiten der Kanten von Komponenten.
- **GroupLayout:** Manche GUI-Builder verwenden dieses Layout, kommen aber häufig mit eigenen Layoutmanagern.
- **NullLayout:** Zur absoluten Positionierung
- Standard-Layout in den `Container`-Klassen ist das `BorderLayout`.
Ausnahme ist die Klasse `JPanel`, bei der das `FlowLayout` voreingestellt ist.

Beispiel FlowLayout

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

public class FlowLayoutDemo {

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setLayout(new FlowLayout());
        JComboBox choice = new JComboBox();
        choice.addItem("Mike: Mein Gott Walter");
        choice.addItem("Sweet: Co Co");
        f.add(choice);
        f.add(new JButton(">"));
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }
}
```



Beispiel BorderLayout

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

public class BorderLayoutDemo {

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(new BorderLayout(5, 5));
        f.add(new JButton("Naughty"), BorderLayout.NORTH);
        f.add(new JButton("Elephants"), BorderLayout.EAST);
        f.add(new JButton("Spray"), BorderLayout.SOUTH);
        f.add(new JButton("Water"), BorderLayout.WEST);
        f.add(new JButton("Center"));
        f.setSize(400, 150);
        f.setVisible(true);
    }
}
```

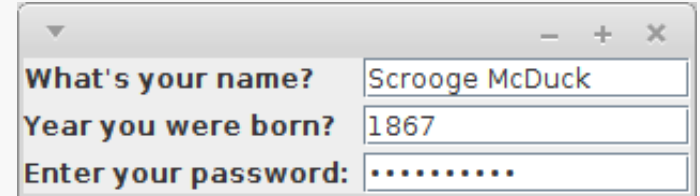


Beispiel GridLayout

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

public class GridLayoutDemo {

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(new GridLayout(3, 2, 6, 3));
        f.add(new JLabel("What's your name?"));
        f.add(new JTextField());
        f.add(new JLabel("Year you were born?"));
        f.add(new JFormattedTextField(NumberFormat.getIntegerInstance()));
        f.add(new JLabel("Enter your password:"));
        f.add(new JPasswordField());
        f.pack();
        f.setVisible(true);
    }
}
```



Parameter:

- Zeilen
- Spalten
- hor. Abstand (Pixel)
- vert. Abstand (Pixel)

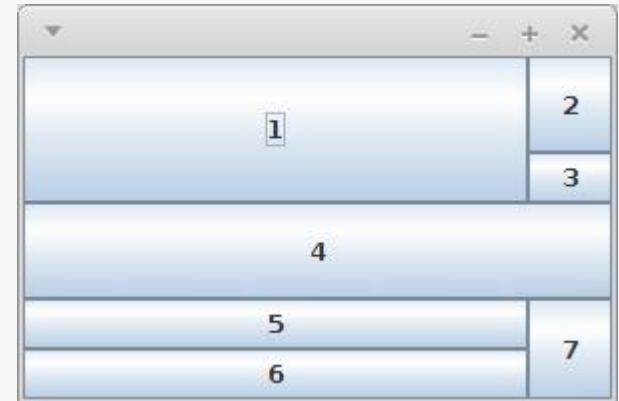
Beispiel GridBagLayout

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

public class GridBagLayoutDemo {
    private static void addComponent(Container cont, Component c,
        int x, int y, int width, int height, double weightx, double weighty) {
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.fill = GridBagConstraints.BOTH;
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = width;
        gbc.gridheight = height;
        gbc.weightx = weightx;
        gbc.weighty = weighty;
        cont.add(c, gbc);
    }

    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = f.getContentPane();
        c.setLayout( new GridBagLayout() );

        // x  y  w  h  wx  wy
        addComponent(c, new JButton("1"), 0, 0, 2, 2, 1.0, 1.0);
        addComponent(c, new JButton("2"), 2, 0, 1, 1, 0, 1.0);
        addComponent(c, new JButton("3"), 2, 1, 1, 1, 0, 0);
        addComponent(c, new JButton("4"), 0, 2, 3, 1, 0, 1.0);
        addComponent(c, new JButton("5"), 0, 3, 2, 1, 0, 0);
        addComponent(c, new JButton("6"), 0, 4, 2, 1, 0, 0);
        addComponent(c, new JButton("7"), 2, 3, 1, 2, 0, 0);
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
```



Beispiel für geschachteltes Layout (1)

Layout der Benutzeroberfläche von Folie 4

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

public class BookManagement {

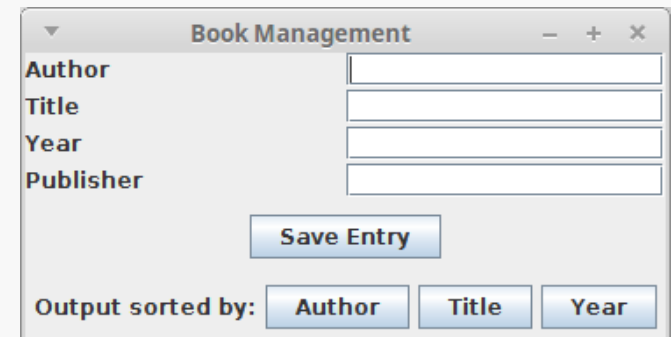
    public BookManagement() {
        JFrame jf = new JFrame("Book Management");
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setLayout(new BorderLayout(5, 5));
        JPanel top = new JPanel();
        top.setLayout(new GridLayout(4, 2, 2, 2));

        top.add(new JLabel("Author"));
        top.add(new JTextField(""));

        top.add(new JLabel("Title"));
        top.add(new JTextField(""));

        top.add(new JLabel("Year"));
        top.add(new JTextField(""));

        // weiter auf der nächsten Folie ...
    }
}
```



Beispiel für geschachteltes Layout (2)

```

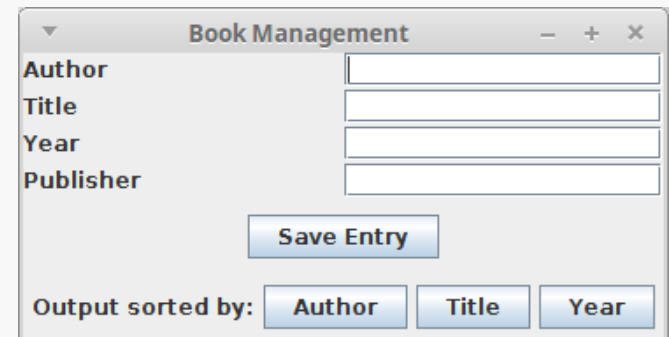
top.add(new JLabel("Publisher"));
top.add(new JTextField(""));

jf.add(top, BorderLayout.NORTH);
JPanel mid = new JPanel();
mid.add(new JButton("Save Entry"));
jf.add(mid, BorderLayout.CENTER);
JPanel bot = new JPanel();
bot.setLayout(new FlowLayout());
bot.add(new JLabel("Output sorted by:"));
bot.add(new JButton("Author"));
bot.add(new JButton("Title"));
bot.add(new JButton("Year"));
jf.add(bot, BorderLayout.SOUTH);
jf.pack();
jf.setVisible(true);
}

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

}

```



The screenshot shows a Java Swing window titled "Book Management". It contains four text input fields for "Author", "Title", "Year", and "Publisher". Below these fields is a "Save Entry" button. At the bottom, there is a label "Output sorted by:" followed by three buttons: "Author", "Title", and "Year".

ÜBUNG (A2)

Dialoge

- Dialogfenster: Spezielle Fenster für Standarddialoge, die nur *temporär* erscheinen, z.B. für Datei- oder Farbauswahl
- Erzeugt eine Java-Applikation *gleichzeitig mehrere Fenster*, unterscheidet man: **modal** („blockierend“) <-> **nicht-modal**:
 - Bildet eine Java-Applikation zwei Fenster, so kann der Anwender zwischen beiden Fenstern hin- und herschalten. Es ist nicht möglich, ein Fenster aufzubauen und dort Eingaben zu erzwingen, während das andere Fenster gesperrt ist.
 - Dafür gibt es in Java spezielle Fenster, die Dialoge, die Swing mit `javax.swing.JDialog` angeht.
 - Ist ein Dialog im Zustand **modal**, muss erst der Dialog beendet werden, damit es in einem anderen Fenster weitergehen kann – alle Benutzereingaben an andere Fenster der Java-Anwendung sind solange gesperrt.
 - Sind mehrere Fenster gleichzeitig offen und können sie Eingaben annehmen, nennt sich dieser Zustand nicht-modal.

Beispiel für modal und nicht-modal (JDialog)

```
import javax.swing.*;

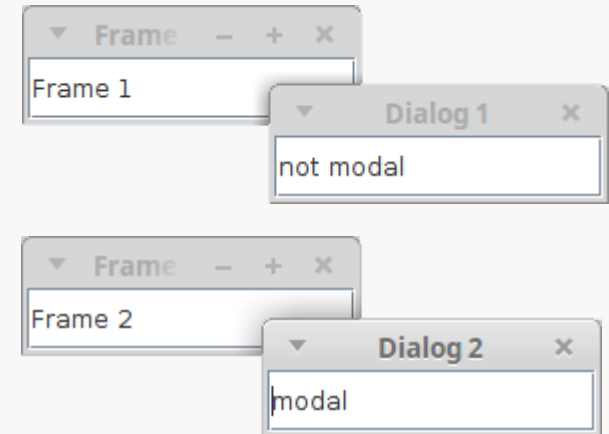
public class JDialogDemo {
    public static void main(String args[]) {
        JFrame f1 = new JFrame("Frame 1");
        f1.add(new JTextField("Frame 1"));
        f1.setSize(170, 60);
        f1.setVisible(true);

        JFrame f2 = new JFrame("Frame 2");
        f2.add(new JTextField("Frame 2"));
        f2.setSize(170, 60);
        f2.setVisible(true);

        JDialog d1 = new JDialog(f1, "Dialog 1", false); // nicht modal
        d1.add(new JTextField("not modal"));
        d1.setSize(170, 60);
        d1.setVisible(true);

        JDialog d2 = new JDialog(f2, "Dialog 2", true); // modal
        d2.add(new JTextField("modal"));
        d2.setSize(170, 60);
        d2.setVisible(true);

        f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Standarddialoge mit JOptionPane

- Für Standarddialoge bietet Java die Klasse `JOptionPane`.
- Diese Klasse erlaubt die Realisierung einfacher Dialoge mit nur *einem* statischen Methodenaufruf der Art `showXXXDialog()`:
 - Meldedialoge (`showMessageDialog()`)
 - Eingabedialoge (`showInputDialog()`)
 - Bestätigungsdialoge (`showConfirmDialog()`)
 - Optionsdialoge (`showOptionDialog()`)
- Alle diese Dialoge sind modal.

Beispiel JOptionPane

```
import javax.swing.*;

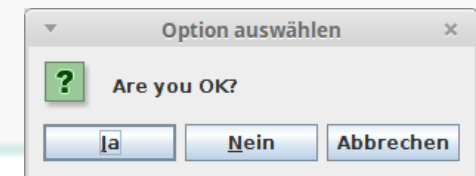
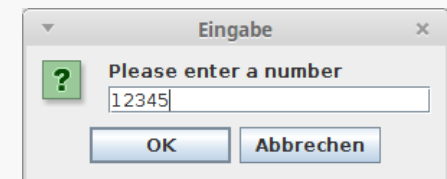
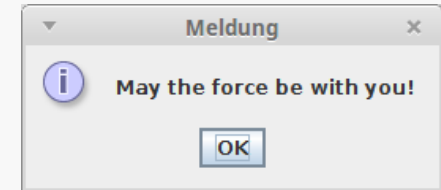
public class JOptionPaneDialog {

    public static void main(String[] args) {
        // Message dialog
        JOptionPane.showMessageDialog(null, "May the force be with you!");

        // Input dialog
        String input =
            (String)JOptionPane.showInputDialog("Please enter a number");
        System.out.println(input);

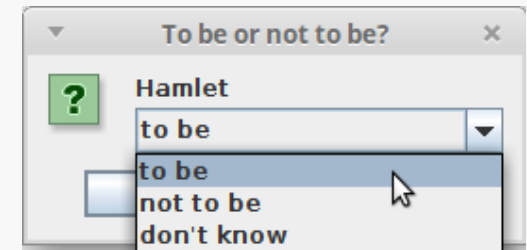
        // Confirm dialog
        JOptionPane.showConfirmDialog(null, "Are you OK?");

        // Weiter auf der nächsten Folie...
```



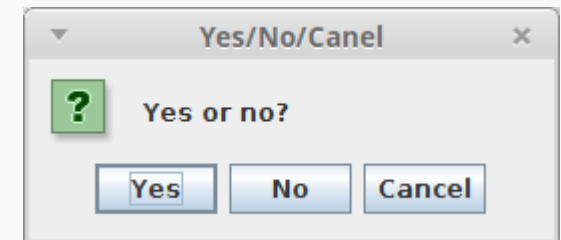
Beispiel JOptionPane (2)

```
// Select dialog
String[] options = { "to be", "not to be", "don't know" };
String selection = (String) JOptionPane.showInputDialog(null, "Hamlet",
    "To be or not to be?", JOptionPane.QUESTION_MESSAGE, null,
    options, options[1]);
System.out.println("Chosen: " + selection);
```



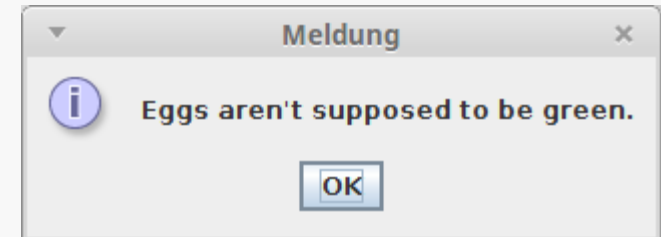
```
// Customized option dialog
String[] opts = { "Yes", "No", "Cancel" };
int n = JOptionPane.showOptionDialog(null, "Yes or no?", "Yes/No/Canel",
    JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE,
    null, opts, opts[0]);

if ( n == JOptionPane.YES_OPTION ) {
    System.out.println("Yes!");
}
}
```

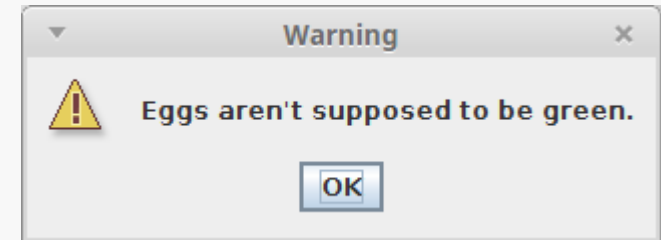


MessageDialogs mit Icons

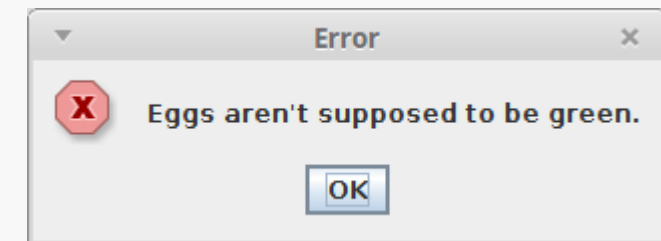
```
// default title and icon
JOptionPane.showMessageDialog(null,
"Eggs aren't supposed to be green.");
```



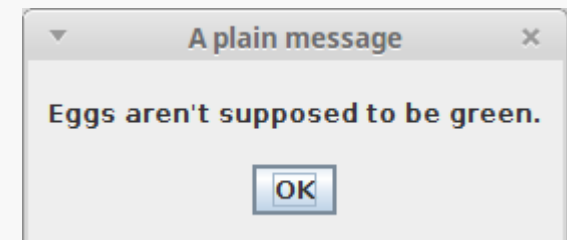
```
// custom title, warning icon
JOptionPane.showMessageDialog(null,
"Eggs aren't supposed to be green.",
"Warning",
JOptionPane.WARNING_MESSAGE);
```



```
// custom title, error icon
JOptionPane.showMessageDialog(null,
"Eggs aren't supposed to be green.",
"Error",
JOptionPane.ERROR_MESSAGE);
```



```
// custom title, no icon
JOptionPane.showMessageDialog(null,
"Eggs aren't supposed to be green.",
"A plain message",
JOptionPane.PLAIN_MESSAGE);
```

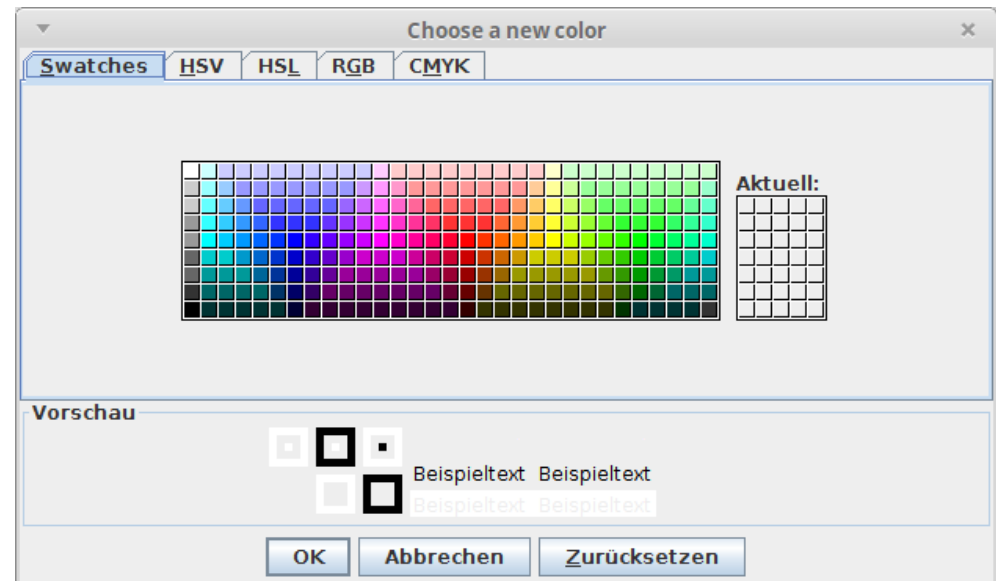


Farben und Schriftarten

- Mit Objekten der Klasse `java.awt.Color` können *Farben* festgelegt werden.
 - Es gibt vordefinierte Farben, z. B. `Color.RED`
 - Farben können auch selbst definiert werden, z. B. mit `new Color(255, 0, 0)`
 - Siehe Klassenbeschreibung in Javadoc
- Mit Objekten der Klasse `java.awt.Font` kann die *Schriftart* von Zeichen festgelegt werden. Konstruktor:
`public Font(String name, int style, int size)`
 - `name`: Name der Font-Familie, z.B. "Helvetica", "Courier", "Roman"
 - `style`: Stil, z.B. `Font.BOLD`, `Font.PLAIN`, `Font.ITALIC`
 - `size`: Größe (in Pixeln)

Farbauswahl mit JColorChooser

- Mit einem `JColorChooser` lassen sich Farben in drei unterschiedlichen Reitern auswählen. Der Benutzer hat die Auswahl zwischen vordefinierten Farben, HSB- und RGB-Werten.
- Der Dialog ist modal



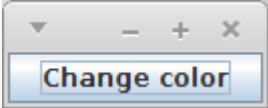
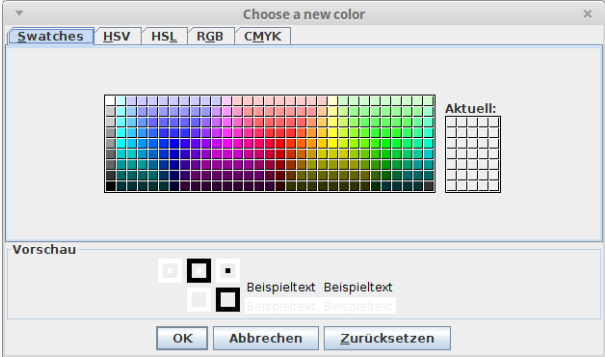
Beispiel JColorChooser

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

public class JColorChooserDemo {

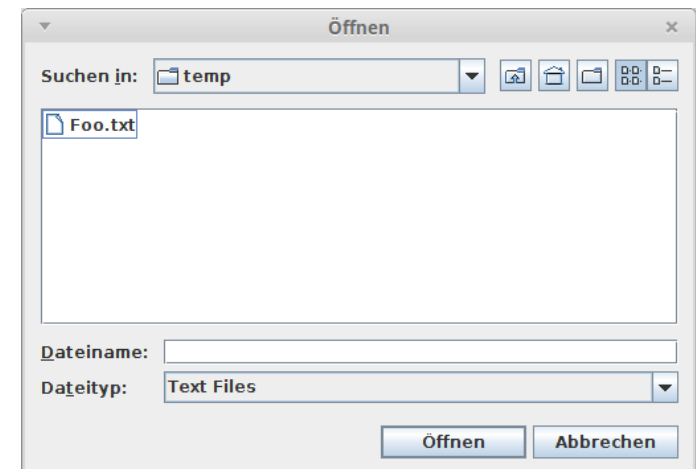
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton b = new JButton("Change color");
        f.add(b);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JComponent comp = (JComponent) e.getSource();
                Color newColor = JColorChooser.showDialog(
                    null, "Choose a new color", comp.setBackground());
                comp.setBackground(newColor);
            }
        });
        f.pack();
        f.setVisible(true);
    }
}
```

Innere Klasse.
Methode
actionPerformed
wird überschrieben

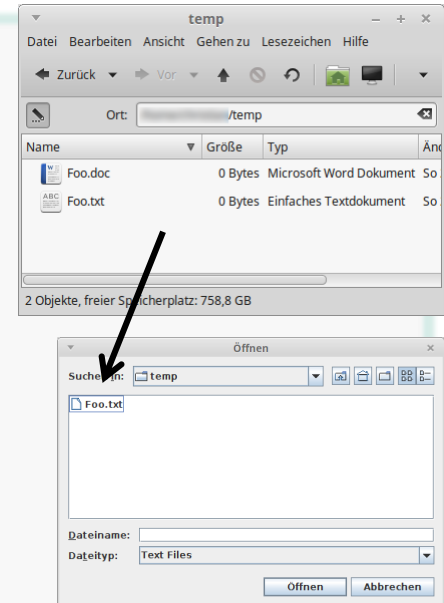
Dateiauswahl mit `JFileChooser`

- Die Klasse `JFileChooser` ermöglicht einen betriebssystemabhängigen Dateiauswahldialog zur Auswahl von Dateien und Verzeichnissen. Der Selektor ist modal und kann für das Speichern, Öffnen oder nach Angabe des Programmierers konfiguriert sein (Buttontext).
- Zudem lassen sich die Pfade und ein `javax.swing.filechooser.FileFilter` zur Auswahl spezieller Dateien setzen.
- Erst nach dem Schließen und Beenden mit dem OK-Button stehen ausgewählte Dateien zur Verfügung.



JFileChooserDialog

```
public class JFileChooserDemo {  
  
    public static void main(String[] args) {  
        JFileChooser fc = new JFileChooser();  
        fc.setFileFilter(new FileFilter() {  
            @Override  
            public boolean accept(File f) {  
                return f.isDirectory() ||  
                    f.getName().toLowerCase().endsWith(".txt");  
            }  
            @Override  
            public String getDescription() {  
                return "Text Files";  
            }  
        });  
  
        int state = fc.showOpenDialog(null); // Varianten öffnen / zeigen  
        // int state = fc.showSaveDialog(null); // Variante speichern  
        // int state = fc.showDialog(null, "Delete"); // freie Variante  
        if (state == JFileChooser.APPROVE_OPTION) {  
            System.out.println(fc.getSelectedFile().getAbsolutePath());  
        } else {  
            System.out.println("No selection");  
        }  
    }  
}
```



Online-Literatur

- „Swing-Tutorial“ von Oracle
<http://docs.oracle.com/javase/tutorial/uiswing/>
- Weiteres hilfreiches Tutorial
<http://www.java-tutorial.org>