DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

Vorlesung Programmieren
Thema 15:
GUI-Programmierung mit Swing (I):
Aufbau, Container, Komponenten

Olaf Herden
Fakultät Technik
Studiengang Informatik

Stand: 06/2024

Gliederung

- Grundlagen
- Aufbau von Swing-Applikationen
- Container
- Komponenten, Klassen, Pakete
- Wichtige Komponenten
- Beispiel

GUIs in Java

- GUI: Graphical User Interface
- Java:
  - AWT (Abstract Windowing Toolkit) ab Version 1.0:
    - Plattformabhängige Darstellung der Elemente (schwergewichtig)
    - Plattformabhängige Behandlung von Benutzereingaben
    - Zeichenfläche stets rechteckig und ohne Überlappungen
    - Umfasst Schnittmenge aller Plattformen
  - Swing: ab Version 1.2, im Folgenden ausführlich
  - JavaFX:
    - Weiterentwicklung plattformübergreifend
    - War von Java 7 bis 10 Bestandteil von Java SE
    - Siehe z.B.
      - https://javabeginners.de/Frameworks/JavaFX/index.php
      - Diverse Tutorials auf youtube

Swing

- Swing-Klassen in Java geschrieben
- Ermöglicht Realisierung plattformunabhängiger grafischer Benutzeroberflächen (leichtgewichtig)
- Komponenten können beliebige Form annehmen
- Swing-Komponenten Obermenge aller Plattformen
- Einheitliches, dynamisches und standardisiertes Aussehen (Pluggable Look and Feel)
- Erheblich größerer Umfang als AWT, z.B. viele sog. Widgets (z.B. Standarddialoge)
- Einsatz Entwurfsmuster MVC (Model – View – Controller)

JFC

- JFC (Java Foundation Classes) bestehen aus:
  - Swing-Komponentenbibliothek (oftmals ungenauerweise synonym zu JFC verwendet)
  - Unterstützung frei wählbares Look and Feel
  - API für Dialoghilfen
  - API zum Einbinden von 2D-Grafik, Text und Bildern
  - Unterstützung Drag&Drop

Übersicht

- Grundlagen
- Aufbau von Swing-Applikationen
- Container
- Komponenten, Klassen, Pakete
- Wichtige Komponenten
- Beispiel

Aufbau Swing-Applikation

- Importieren benötigter Swing-Pakete (und AWT-Pakete)
- Konstruktor: Subklasse von `JFrame`
- Einzelne Komponenten (Container, Textfelder, Schaltflächen, ...) erzeugen und konfigurieren
- Komponenten in Container einfügen
- Komponenten innerhalb des Containers anordnen
- Ereignisbehandlung:
  - Welche Komponente löst welches Ereignis aus?
  - Listener (Komponenten zur Ereignisbehandlung) realisieren

Beispiel (I)

```
( 1) import javax.swing.*;
( 2) import java.awt.*;
( 3) import java.awt.event.*;
( 4)
( 5) // Klasse FensterSwing
( 6) // Legt ein leeres Fenster-Objekt an
( 7) // Objekt dient als Gerüst für jede Swing-Applikation
( 8)
( 9) public class FensterSwing extends JFrame{
(10)
(11) // Konstruktor legt Fenster an
(12) // Titelzeile ist uebergebene Zeichenkette
(13)
(14)   public FensterSwing(String titel){
(15)     super(titel);
(16)   }
(17) }
```

Beispiel (II)

```
( 1)   // Klasse Anwendung
( 2)   // Demonstriert Funktionsweise der Klasse FensterSwing
( 3)
( 4)   public class Anwendung{
( 5)
( 6)     public static void main(String[] args){
( 7)          JFrame meinFenster = new FensterSwing("Erste Swing-Anwendung");
( 8)     }
( 9)   }
```

- Demo: Beispiel 1
- Fenster angelegt, Programm kompilierbar, beim Ausführen passiert nichts
- Lösung: Methode `setVisible()`

Erweiterung um `setVisible`

```
( 1)   // Klasse Anwendung
( 2)   // Demonstriert Funktionsweise der Klasse FensterSwing
( 3)
( 4)   public class Anwendung{
( 5)
( 6)     public static void main(String[] args){
( 7)       JFrame meinFenster = new FensterSwing("Erste Swing-Anwendung");
( 8)       meinFenster.setVisible(true);
( 9)     }
(10)   }
```


- Demo: Beispiel 2
- Resultat: 
- Möglich: Größe verändern (klein oder Vollbild), Kontextmenü mit rechter Maustaste und Schließen (aber nicht Beenden)
- Wünschenswert: Weitere Eigenschaften, z.B. Zwischengrößen, „richtiges“ Beenden

Erweiterung um `setSize`

```
( 1)   // Klasse Anwendung
( 2)   // Demonstriert Funktionsweise der Klasse FensterSwing
( 3)
( 4)   public class Anwendung{
( 5)
( 6)     public static void main(String[] args){
( 7)       JFrame meinFenster = new FensterSwing("Erste Swing-Anwendung");
( 8)       meinFenster.setVisible(true);
( 9)       meinFenster.setSize(100,200);
( 9)     }
(10)   }
```

- Demo: Beispiel 3
- Resultat:
  - Fenster öffnet in angegebener Größe
  - Horizontales und vertikales Vergrößern/-kleinern

Erweiterung um Listener

```
( 1) public FensterSwing(String titel){
( 2)   ...
( 3)   // Listener erstellen ...
( 4)   WindowListener meinListener = new WindowAdapter(){
( 5)         public void windowClosing(WindowEvent ereignis){
( 6)           System.exit(0);
( 7)         }
( 8)   };
( 9)   // ... und hinzufuegen
(10)   this.addWindowListener(meinListener);
(11) }
  }
```

- Demo: Beispiel 4
- Resultat: Optisch gleich, aber nach Schließen Konsole wieder frei

Übersicht

- Grundlagen
- Aufbau von Swing-Applikationen
- Container
- Komponenten, Klassen, Pakete
- Wichtige Komponenten
- Beispiel

Container

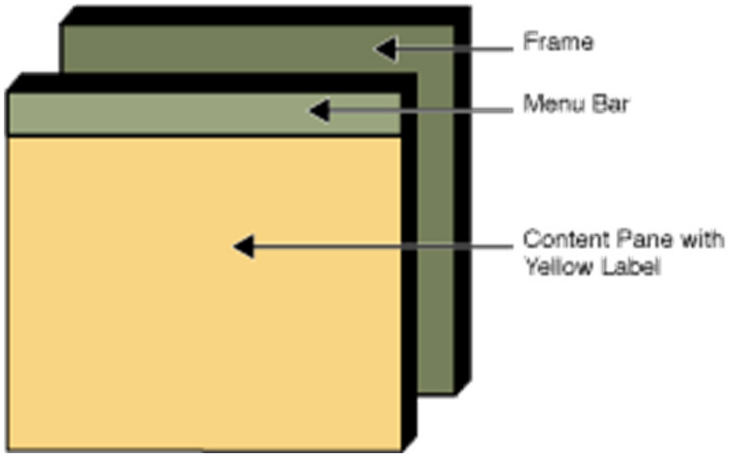
- Behälter für Komponenten
- Funktion: Komponenten nehmen, gruppieren, anordnen und als Einheit behandeln
- Container/Komponenten bilden Hierarchie, d.h. sie können andere Container/Komponenten aufnehmen
- Vorteile:
  - Übersichtlichkeit
  - Leichte Änderbarkeit
  - Wiederverwendbarkeit
- Zwei Arten von Containern:
  - Top Level Container
  - Sonstige (General Purpose und Special Purpose Container)
- Gemeinsame Oberklasse aller Container ist `java.awt.Container`:
  - Hinzufügen und Entfernen von Komponenten
  - Anordnen der Komponenten auf der Oberfläche (Layout)

Top Level Container (I)

- Stehen Komponenten-Hierarchie ganz oben
- Anschaulich: Äußerer Rahmen eines Programms
- Es gilt:
  - Jedes Programm hat mindestens einen Top Level Container
  - Alle anderen Komponenten befinden sich innerhalb des Top Level Containers
  - Top Level Container können keine anderen Top Level Container enthalten
- Folgende Klassen zählen zu den Top Level Containern:
  - (Applets (Klasse `JApplet`))
  - Dialoge (Klasse `JDialog`): Fenster, das zu einem „Oberfenster“ gehört
  - Rahmen (Klassen `JWindow` und `JFrame`): Fenster mit Rahmen, Titel und Icons zum Schließen
  - Alle vier Klassen implementieren Interface `RootPaneContainer`

Top Level Container (II)

- Aufnehmen anderer Komponenten nicht direkt, sondern über die Zwischenkomponente des Content Pane

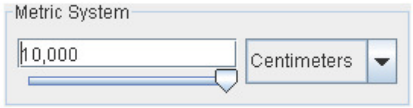


- Syntax: `fenster.getContentPane().add(<Neue Komponente>);`

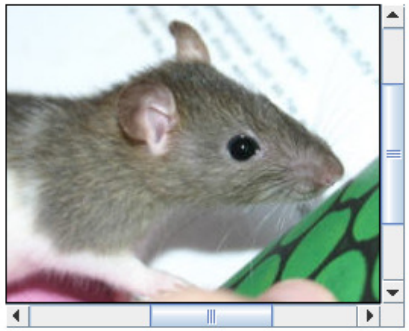
[https://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html]

General Purpose Container

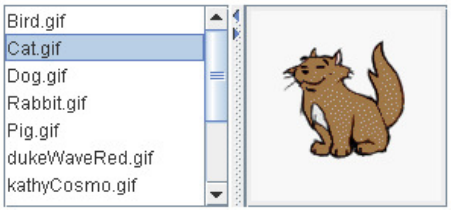
- Behälter für allgemeine bzw. universelle Anwendungen



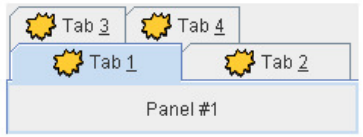
JPanel



JScrollPane



JSplitPane



JTabbedPane

JToolBar

[https://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html]

Special Purpose Container (I)

- Behälter für spezielle Einsatzgebiete
- Bsp.: Internal Frame, Layered Pane, Root Pane

- Internal Frame:
  - Viele Anwendungen kommen mit einem Fenster aus
  - Werden als SDI-Anwendungen bezeichnet (Single Document Interface)
  - Technisch: Benötigen nur ein einziges `JFrame`-Objekt
  - Häufig werden aber mehrere Fenster benötigt, z.B. in Textverarbeitung gleichzeitig mehrere Dokumente bearbeiten
  - Solche Anwendungen werden als MDI-Anwendungen bezeichnet (Multiple Document Interface)
  - Realisierung in Swing: interne Rahmen mit gleichem Verhalten wie `JFrame`-Objekt, Bedienung aber unabhängig voneinander
  - Vorteil: Wechsel zwischen internen Rahmen performanter als zwischen zwei Hauptrahmen

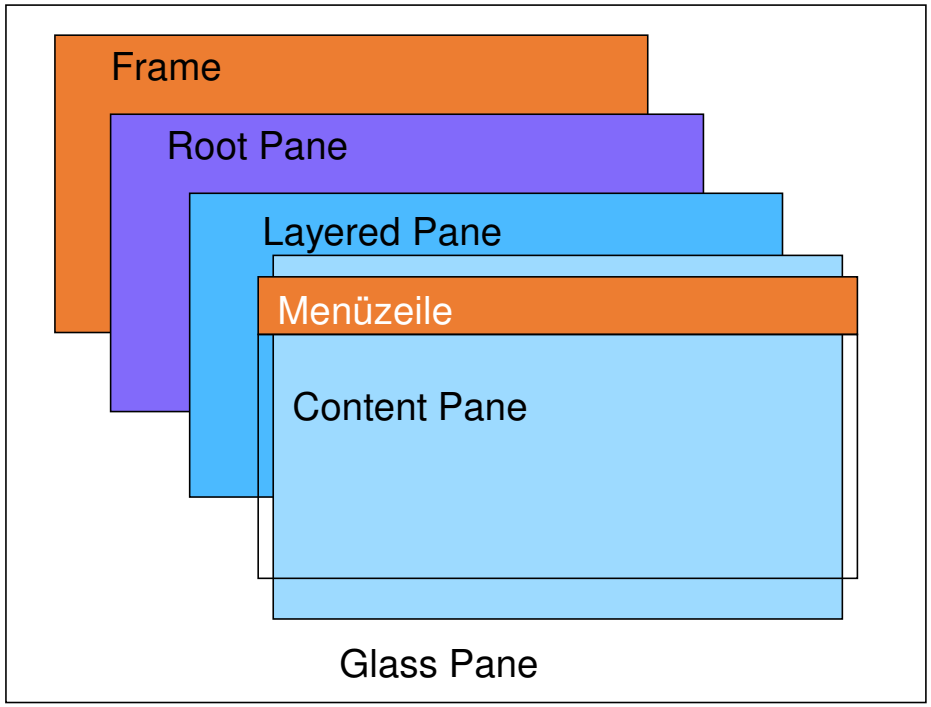
Special Purpose Container (II)

- Layered Pane:
  - Dienen der Darstellung überlappender Komponenten
  - Dadurch können z.B. auch 3D-Effekte erzielt werden
  - Werden relativ selten verwendet

- Root Pane:
  - Werden von den Top Level Containern und internen Rahmen automatisch erzeugt (technisch: Interface `RootPaneContainer` wird implementiert)
  - Bestehen aus folgenden Teilen:
    - Root Pane: Enthält Glass Pane, Layered Pane und einen Layout Manager (dieser kontrolliert andere Panes und Menüzeile)
    - Layered Pane: Enthält Menüleiste (optional) und Content Pane
    - Content Pane: Enthält Komponenten
    - Glass Pane: Standardmäßig transparentes Pane der obersten Ebene (Zweck: Ereignisse abfangen)

Aufbau Root Pane

- .



Übersicht

- Grundlagen
- Aufbau von Swing-Applikationen
- Container
- Komponenten, Klassen, Pakete
- Wichtige Komponenten
- Beispiel

Komponenten (I)

- Einfache Komponenten (Basic Controls):
  - Schaltflächen
  - Radio Buttons
  - Checkboxen
  - Listenfelder
  - Comboboxen
  - Menüs
  - Schieberegler
  - Textfelder
- Nicht-editierbare Komponenten:
  - Dienen der Anzeige von Informationen
  - Tool-Tipps
  - Fortschrittsbalken

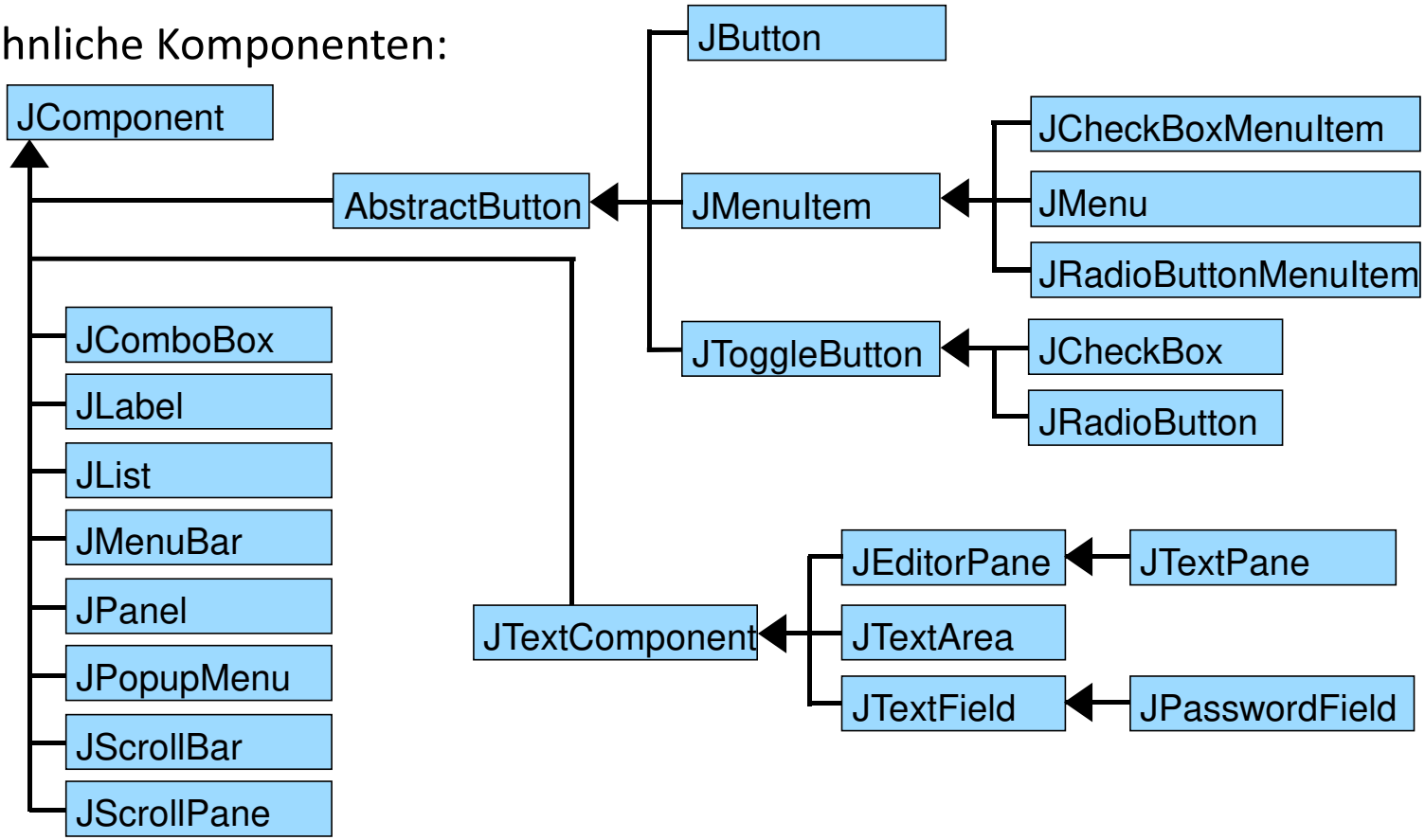
Komponenten (II)

- Editierbare Komponenten:
  - Dienen Interaktion mit Benutzer*innen
  - Dialog zur Auswahl einer Farbe
  - Dialog zum Auswählen und Öffnen einer Datei
  - Ebenso Tabellen, Textfelder und Trees
  - Vorteil: Sehr komplexe Funktionalität wird zusammengefasst

- Alle Komponenten erben von gemeinsamer Superklasse `JComponent`:
  - Behandlung von Tastatureingaben
  - Rahmen
  - Zugriff, Locale-Eigenschaften (z.B. Darstellung Datum und Währung), Tool-Tipps
- Siehe auch z.B. https://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html

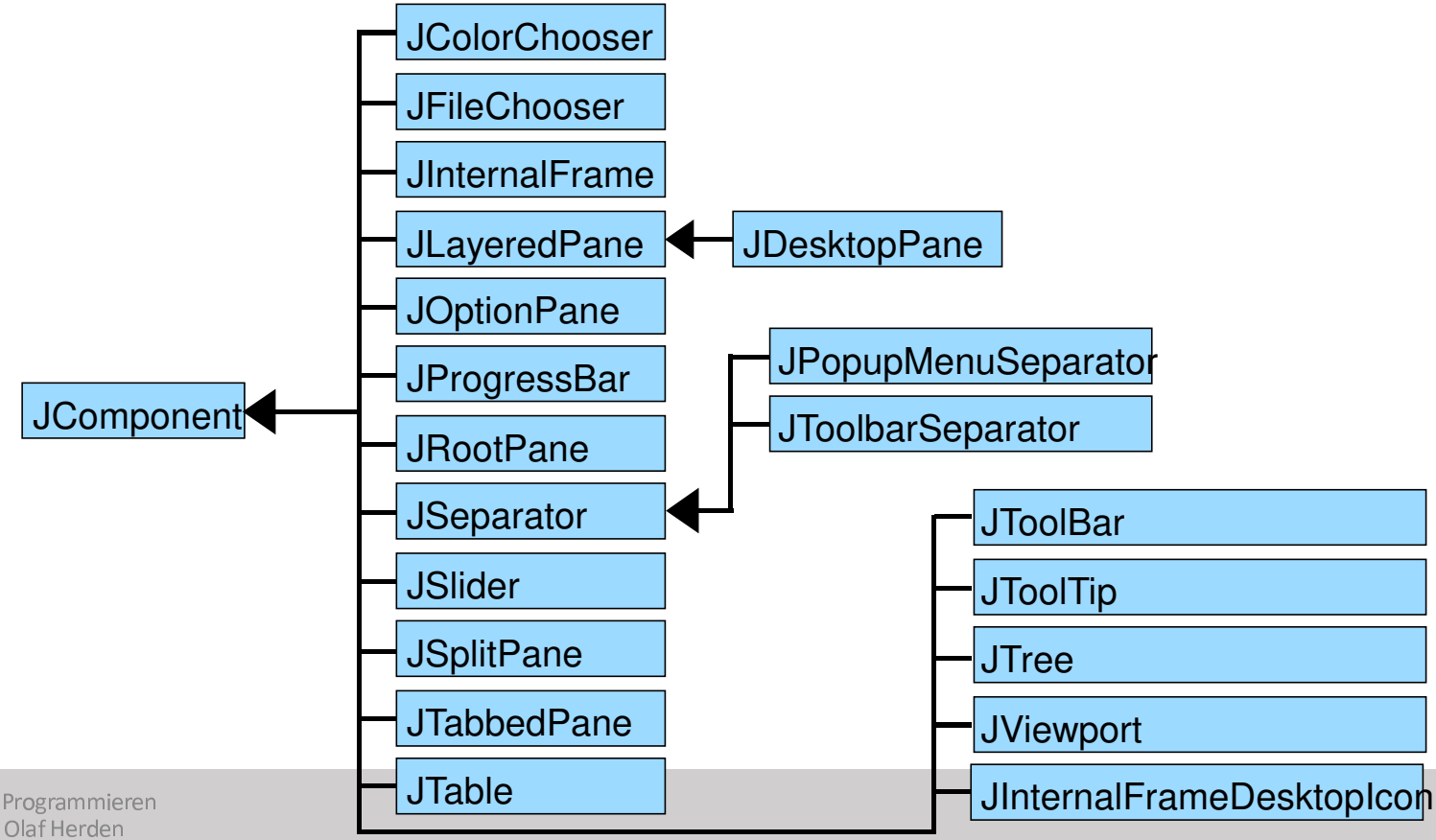
Swing-Klassenhierarchie (I)

- AWT-ähnliche Komponenten:



Swing-Klassenhierarchie (II)

- Erweiterte und neue Komponenten in Swing:



Wichtige Swing-Pakete

- `javax.swing`: Wichtigste Swing-Komponenten (Schaltflächen, Menüs, ...) und Grundfunktionalitäten (für Applets)
- `javax.swing.event`: Ereigniserkennung und –verarbeitung
  - Swing-Programme arbeiten ereignisorientiert, d.h. Anklicken einer Schaltfläche oder eines Menüeintrags löst ein Ereignis aus
  - Dieses muss vom Programm erkannt und in geeigneter Weise darauf reagiert werden
- `javax.swing.plaf`: Plaf (Pluggable Look and Feel), Paket bietet Wechselmöglichkeit zwischen verschiedenen Oberflächenerscheinungsbildern (z.B. Windows, MOTIF)
- `javax.swing.table`: Tabellenkomponente unterstützende Klassen
- `javax.swing.text`: Textfelder verschiedener Ausprägung (Normale, gesperrte, Passwortfelder)
- `javax.swing.text.rtf`: Verwaltung von Texten im RTF-Format (Rich Text Format), ermöglicht z.B. Fett- oder Kursivschrift

Übersicht

- Grundlagen
- Aufbau von Swing-Applikationen
- Container
- Komponenten, Klassen, Pakete
- Wichtige Komponenten
- Beispiel

Klasse `JButton` (I)

- Klasse für Schaltflächen
- Konstruktoren:

Name	Bedeutung
`JButton()`	Erzeugt „leeren“ Button, d.h. ohne Text oder Grafik
`JButton(Action a)`	Erzeugt Button, dessen Aussehen vom Ereignis a abhängt
`JButton(Icon i)`	Erzeugt Button mit Grafik i
`JButton(String s)`	Erzeugt Button mit Zeichenkette s
`JButton(String t, Icon i)`	Erzeugt Button mit Kombination aus letzten beiden

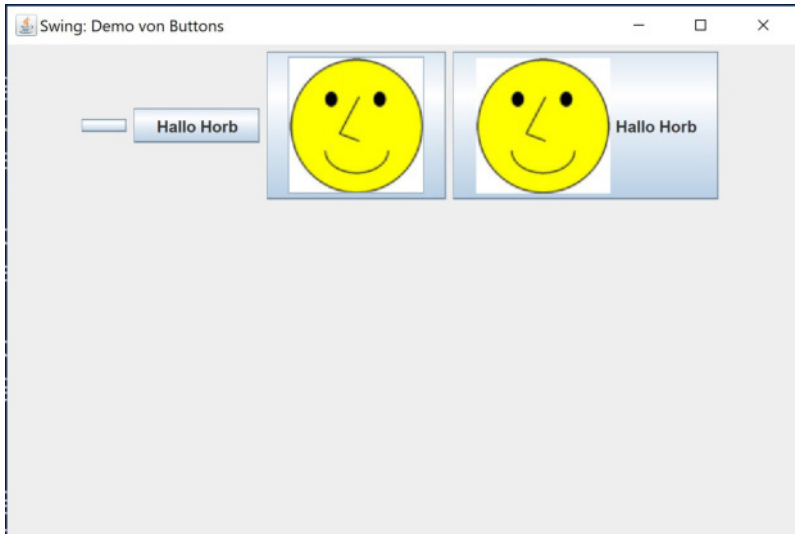
Klasse JButton (II): Beispiel (I)

```
( 1) public FensterSwing(String titel, int x, int y){
( 2)     ...
( 3)   // Anlegen der Buttons
( 4)   JButton leereSF = new JButton();
( 5)   JButton textSF = new JButton("Hallo Horb");
( 6)   Icon i = new ImageIcon("Smile_Klein.jpg");
( 7)   JButton grafikSF = new JButton(i);
( 8)   // Keine direkte Zuweisung der Komponenten
( 9)   // D.h.: this.add(leereSF); scheitert
(10)   // Stattdessen: Pane anlegen und zum Content Pane erklaeren
(11)   JPanel contentPane = new JPanel();
(12)   // ... Buttons hinzufügen ...
(13)   contentPane.add(leereSF);
(14)   contentPane.add(textSF);
(15)   contentPane.add(grafikSF);
(16)   // ... und "unseren" contentPane zum Content Pane erklären
(17)   this.setContentPane(contentPane);
(18) }
```

- Demo: Beispiel 5

Klasse `JButton` (III): Beispiel (II)

- Resultat:



- Anm.: Anordnung der Komponenten nebeneinander, obwohl keine Angaben gemacht (Später: Layout-Manager)

Beschriftungen und Textfelder (I)

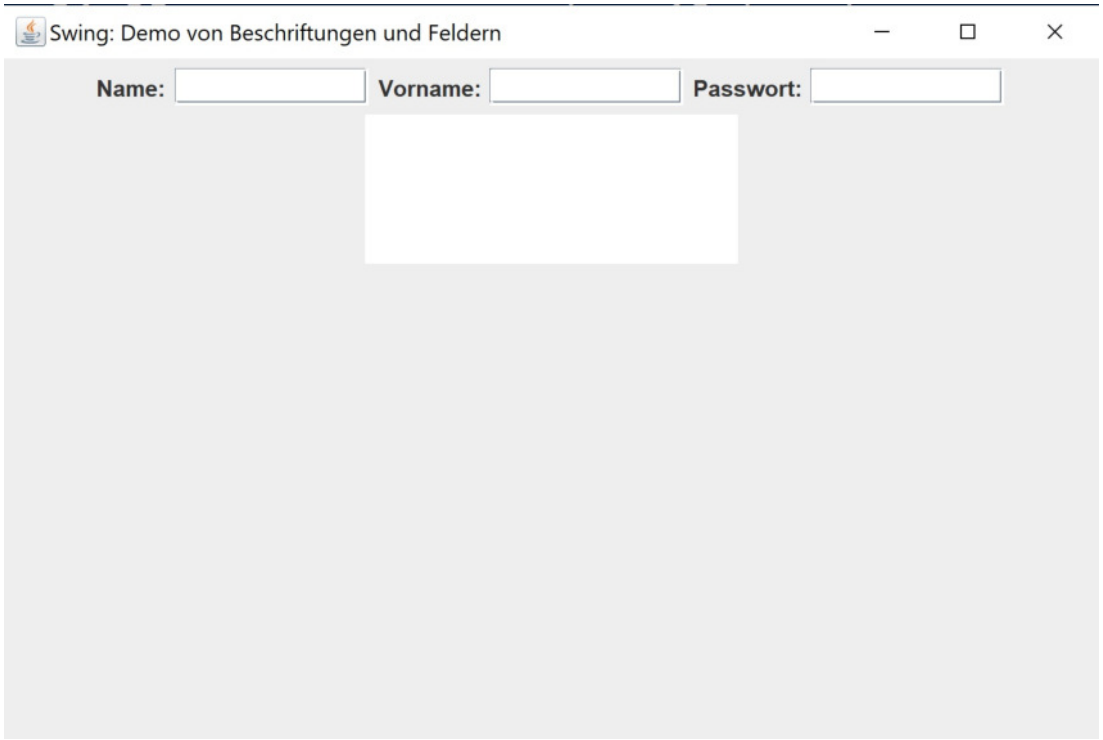
- Klasse `JLabel`:
  - Beschriftungen
  - Übergabe Zeichenkette als Argument mit Text an Konstruktor
- Klasse `JTextField`:
  - Texteingaben
  - Übergabe Zahl mit Breite in Zeichen an Konstruktor
- Klasse `JPasswordField`:
  - Geschützte Felder zum Eingeben von Passwörtern
  - Übergabe Zahl mit Breite in Zeichen an Konstruktor
- Klasse `JTextArea`:
  - Größere Textfelder

Beschriftungen und Textfelder (II): Beispiel (I)

```
( 1) // Anlegen der Beschriftungen (Labels)
( 2) JLabel labelName = new JLabel("Name:");
( 3) JLabel labelVorname = new JLabel("Vorname:");
( 4) JLabel labelPass = new JLabel("Passwort:");
( 5) // Anlegen der Textfelder
( 6) JTextField feldName = new JTextField(10);
( 7) JTextField feldVorname = new JTextField(10);
( 8) // Anlegen des Passwortfeldes
( 9) JPasswordField feldPass = new JPasswordField(10);
(10) // Anlegen der JTextArea
(11) JTextArea feldText = new JTextArea(5,20);
(12) // Alle zum Content Pane hinzufügen
(13) JPanel contentPane = new JPanel();
(14) contentPane.add(labelName);
(15) contentPane.add(feldName);
(16) contentPane.add(labelVorname);
(17) contentPane.add(feldVorname);
(18) contentPane.add(labelPass);
(19) contentPane.add(feldPass);
(20) contentPane.add(feldText);
(21) this.setContentPane(contentPane);
```

Beschriftungen und Textfelder (III): Beispiel (II)

- Demo: Beispiel 6
- Resultat:



Menüs (I)

- Aufbau:
  - Menüleiste (Objekt der Klasse `JMenuBar`)
  - Enthält Reihe Menüs (Objekte der Klasse `JMenu`)
  - Enthält Reihe von Einträgen (Objekte der Klasse `JMenuItem`)
- Objekte nacheinander erzeugen und ineinander eingefügen
- Wichtiger Unterschied zu Schaltflächen (und anderen Komponenten):
  - Menüs werden nicht dem Content Pane zugeordnet
  - Siehe auch letzter Abschnitt: Panes
  - Einträge können Menüs und Menüs der Menüleiste unmittelbar mit `add`-Methode zugeordnet werden
  - Menüleiste wird mit `set`-Methode aktiviert

Menüs (II): Beispiel (I)

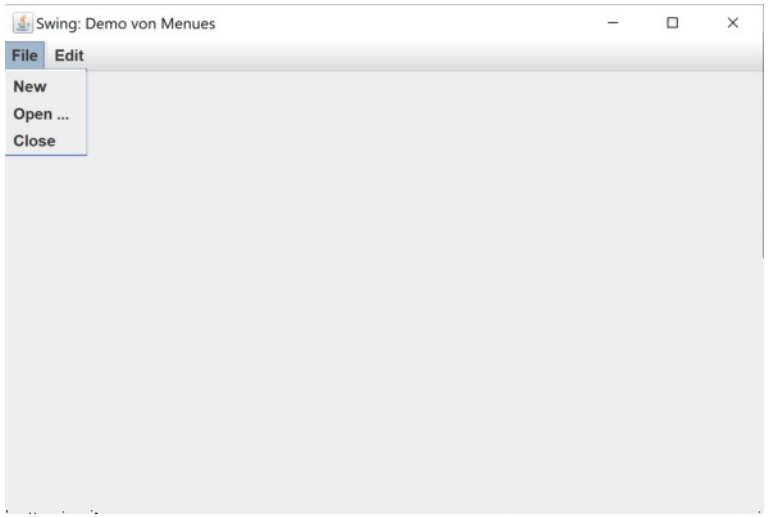
```
( 1) // Anlegen der Menüleiste
( 2) JMenuBar menueLeiste = new JMenuBar();
( 3) // Anlegen von Menüs
( 4) JMenu menueDatei = new JMenu("Datei");
( 5) JMenu menueBearbeiten = new JMenu("Bearbeiten");
( 6) // Zuweisen der Menüs zur Menüleiste
( 7) menueLeiste.add(menueDatei);
( 8) menueLeiste.add(menueBearbeiten);
( 9) // Anlegen der Menüeinträge
(10) JMenuItem eintragMenueDateiNeu = new JMenuItem("Neu");
(11) JMenuItem eintragMenueDateiOeffnen = new JMenuItem("Öffnen ...");
(12) JMenuItem eintragMenueDateiSchliessen = new JMenuItem("Schließen");
(13) JMenuItem eintragMenueBearbeitenMarkieren = new MenuItem("Markieren");
(14) JMenuItem eintragMenueBearbeitenKopieren = new JMenuItem("Kopieren");
(15) JMenuItem eintragMenueBearbeitenEinfuegen = new JMenuItem("Einfügen");
...
```

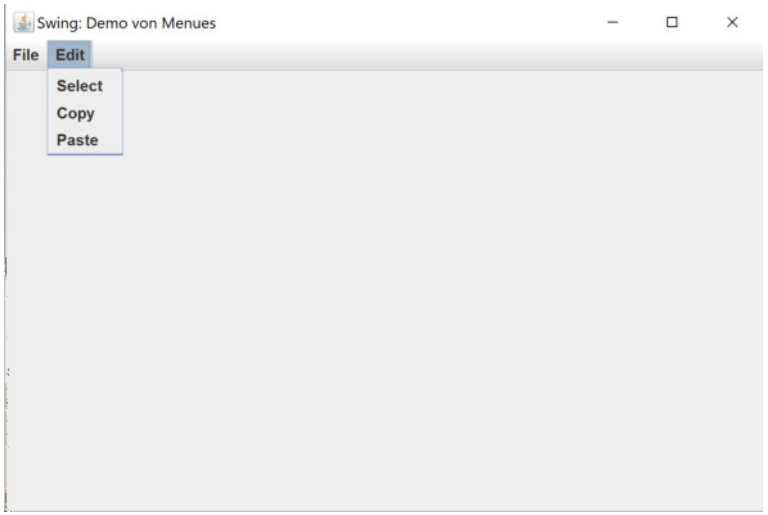
Menüs (III): Beispiel (II)

```
...
(16) // Zuweisen der Menüeinträge zu ihren Menüs
(17) menueDatei.add(eintragMenueDateiNeu);
(18) menueDatei.add(eintragMenueDateiOeffnen);
(19) menueDatei.add(eintragMenueDateiSchliessen);
(20) menueBearbeiten.add(eintragMenueBearbeitenMarkieren);
(21) menueBearbeiten.add(eintragMenueBearbeitenKopieren);
(22) menueBearbeiten.add(eintragMenueBearbeitenEinfuegen);
(23) // Menüleiste aktivieren
(24) this.setJMenuBar(menueLeiste);
```

Menüs (IV): Beispiel (III)

- Demo: Beispiel 7
- Resultat:





Menüs (V): Anmerkungen

- Reihenfolge Erzeugung von Menüleisten und Komponenten innerhalb des Code beliebig
- Menüs bieten viele weitere Optionen:
  - Untermenüs
  - Einträge mit Radio Buttons, Checkboxen oder Grafiken versehen
  - Bereiche durch Separatoren abtrennen
  - Shortcuts

Dialoge (I)

- Standarddialoge: Realisieren Interaktionen mit Benutzer*innen
- Gegenteil: Benutzerdefinierte Dialoge
- Standarddialoge arbeiten modal, d.h. volle Übernahme Programmkontrolle, nach Abarbeitung des Dialogs weitere Ausführung des Programms
- Klasse `JOptionPane`: Bietet Standarddialoge an:
  - Informationsdialoge (Message Dialog): Darstellung Informationen über Zustand des Programms, meistens Warnungen oder Fehlermeldungen
  - Eingabedialoge (Information Dialog): verlangen Eingabe bestimmter Informationen (z.B. Kennung und Passwort)
  - Auswahldialoge (Confirm Dialog): Auswahl aus vorgegeben Anzahl von Alternativen, auch zur Bestätigung von Informationen verwendet
  - Optionendialoge (Option Dialog): Kombination der drei anderen

Informationsdialoge (I)

- Darstellung von Informationen
- Methode `showMessageDialog()` der Klasse `JOptionPane`:
  - `showMessageDialog(<Container>,<Text>,<Titel>,<Typ>)`
  - Folgende Konstanten für Typ:
    - `ERROR_MESSAGE`: Fehlermeldung
    - `INFORMATION_MESSAGE`: Informationsmeldung
    - `WARNING_MESSAGE`: Warnung
    - `QUESTION_MESSAGE`: Frage
    - `PLAIN_MESSAGE`: Leerer Dialog

Informationsdialoge (II): Beispiel (I)

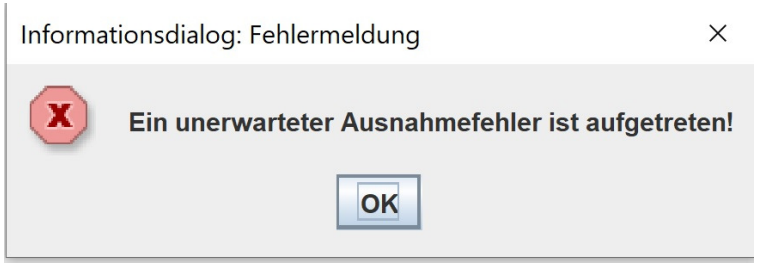
```
( 1) // Content Pane ermitteln
( 2) Container contentPane = getContentPane();
( 3) // Informationsdialoge
( 4) // 1) Fehlermeldung
( 5) JOptionPane.showMessageDialog(
( 6)    contentPane,
( 7)    "Ein unerwarteter Ausnahmefehler ist aufgetreten!",
( 8)    "Informationsdialog: Fehlermeldung",
( 9)    JOptionPane.ERROR_MESSAGE);
(10) // 2) Informationsmeldung
(11) JOptionPane.showMessageDialog(
(12)    contentPane,
(13)    "Dies ist ein allgemeiner Hinweis!",
(14)    "Informationsdialog: Informationsmeldung",
(15)    JOptionPane.INFORMATION_MESSAGE);
...
```

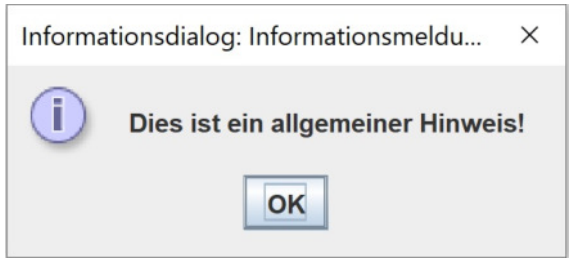
Informationsdialoge (III): Beispiel (II)

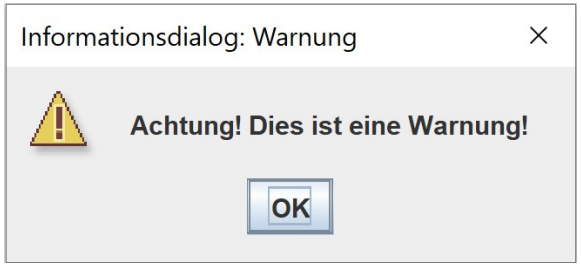
```
...
(16) // 3) Warnung
(17) JOptionPane.showMessageDialog(
(18)    contentPane,
(19)    "Achtung! Dies ist eine Warnung!",
(20)    "Informationsdialog: Warnung",
(21)    JOptionPane.WARNING_MESSAGE);
(22) // 4) Frage
(23) JOptionPane.showMessageDialog(
(24)    contentPane,
(25)    "Frage: Wollen Sie weitermachen?",
(26)    "Informationsdialog: Frage",
(27)    JOptionPane.QUESTION_MESSAGE);
(28) // 5) Leerer Dialog
(29) JOptionPane.showMessageDialog(
(30)    contentPane,
(31)    "Leerer Informationsdialog.",
(32)    "Informationsdialog: Leer",
(33)    JOptionPane.PLAIN_MESSAGE);
```

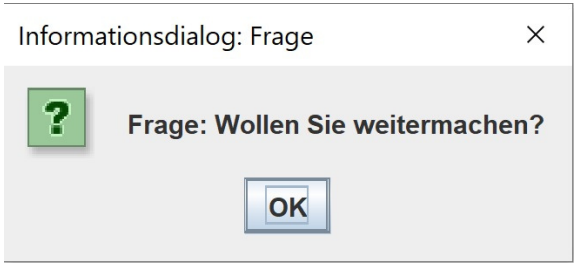
Informationsdialoge (IV): Beispiel (III)

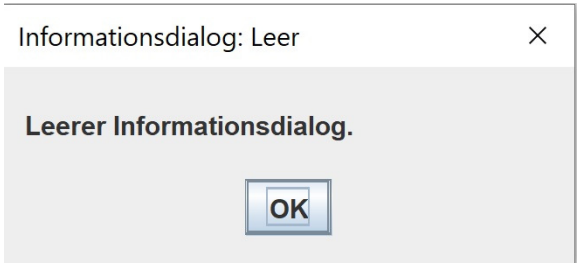
- Demo: Beispiel 8
- Resultat:











Eingabedialoge (I)

- Eingabe aus Menge vorgegebener Werte
- Methode `showInputDialog()` der Klasse `JOptionPane`:
  - `showInputDialog(<Parameter>)` mit Parametern:

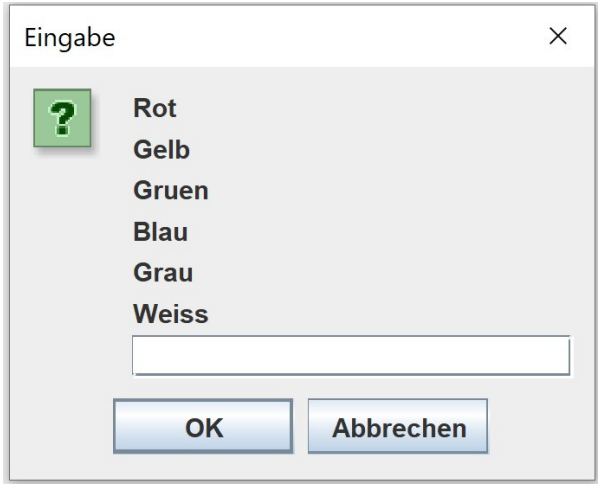
Parameter	Beispiel	Beschreibung
`Component parentComponent`	cp	Gibt den Content Pane an.
`Object message`	„Bitte auswählen:“	Text zur Eingabeaufforderung
`String title`	„Eingabedialog“	Titelzeile des Dialogs
`int messageType`	`JOptionPane.INFORMATION_MESSAGE`	Typ des Dialogs
`Icon icon`		Anzuzeigendes Icon
`Object selectionValues[]`	Wert	Zulässige Eingabewerte
`Object initial_SelectionValue`	Ein spezieller Wert	Standardeingabewert

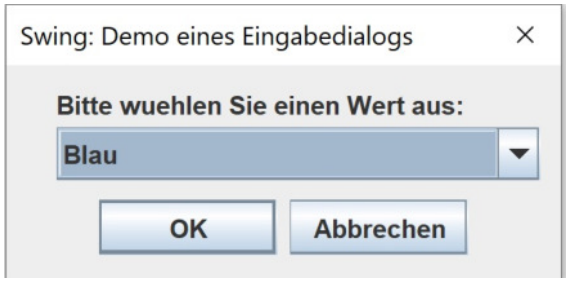
Eingabedialoge (II): Beispiel (I)

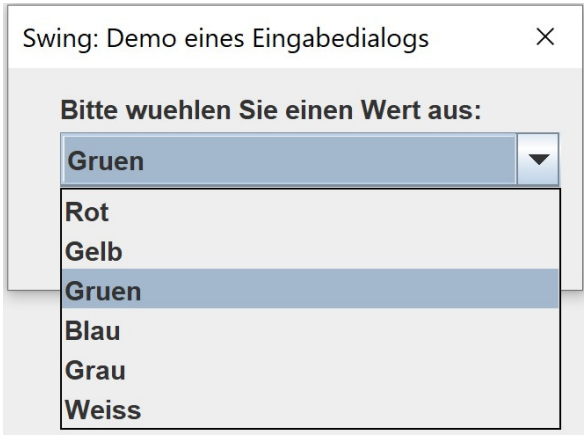
```
( 1) // Zulässige Werte
( 2) String[] werte = {"Rot","Gelb","Grün","Blau","Grau","Weiß"};
( 3) // Eingabedialog in einfachster Form
( 4) String s =
( 5)   (String) JOptionPane.showInputDialog(contentPane,werte);
                                          Erscheinungsort   Liste zulässiger Werte
( 6) // Und etwas komplizierter, aber komfortabler als Liste
( 7) String t = (String) JOptionPane.showInputDialog(
( 8)          contentPane,                                  ← Erscheinungsort
( 9)          "Bitte wählen Sie einen Wert aus:",           ← Eingabeaufforderung
(10)          "Swing: Demo eines Eingabedialogs",           ← Titelzeile
(11)          JOptionPane.INFORMATION_MESSAGE,              ← Dialogtyp
(12)          new ImageIcon("Bunt.jpg"),                    ← Verwendetes Icon
(13)          werte,                                        ← Liste zulässiger Werte
(14)          "Blau");                                      ← Standardwert
```

Eingabedialoge (III): Beispiel (II)

- Demo: Beispiel 9
- Resultat:







Auswahldialoge (I)

- Ähnlich Eingabedialogen, statt Auswahl Bestätigung
- Methode `showConfirmDialog()` der Klasse `JOptionPane`:
  - `showConfirmDialog(<Parameter>)` mit Parametern:

Parameter	Beispiel	Beschreibung
`Component parentComponent`	cp	Gibt den Content Pane an.
`Object message`	„Bitte bestätigen“	Text zur Bestätigung
`String title`	„Auswahldialog“	Titelzeile des Dialogs
`int buttonType`	Siehe unten	Angezeigte Schaltflächen
`int messageType`	`JOptionPane.INFORMATION_MESSAGE`	Typ des Dialogs
`Icon icon`		Anzuzeigendes Icon

Auswahldialoge (II): Beispiel (I)

- Mögliche Werte für buttonType:
  - `JOptionPane.YES_NO_CANCEL_OPTION`
  - `JOptionPane.YES_NO_OPTION`
  - `JOptionPane.YES_OPTION`

```
( 1) // Auswahldialog darstellen
( 2) JOptionPane.showConfirmDialog(
( 3)   contentPane,                                  ← Erscheinungsort
( 4)   „Hallo!",                                     ← Text zur Bestätigung
( 5)   "Swing: Demo eines Auswahldialogs",           ← Titelzeile
( 6)   JOptionPane.YES_NO_CANCEL_OPTION,             ← Schaltflächen
( 7)   JOptionPane.INFORMATION_MESSAGE,              ← Dialogtyp
( 8)  new ImageIcon("jolly.jpg"));                   ← Verwendetes Icon
```

Auswahldialoge (III): Beispiel (II)

- Demo: Beispiel 10
- Resultat:

Optionendialoge (I)

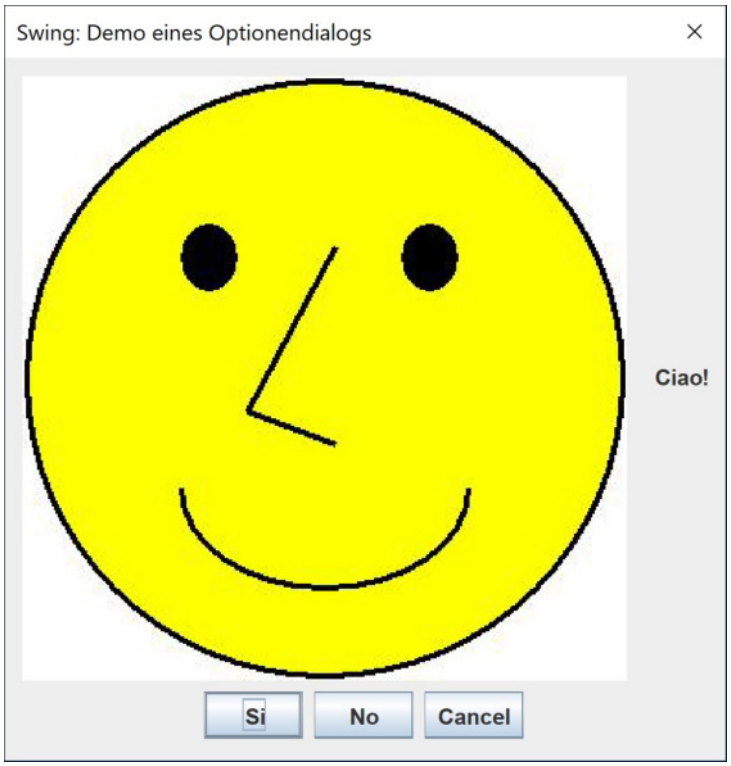
- Vielseitigste Dialoge in Swing
- Daher: längere Parameterlisten als andere Dialoge
- Bsp.weise kann man den Text für die Schaltflächen selber bestimmen

Optionendialoge (II): Beispiel (I)

```
( 1) // Schaltflächenbeschriftungen auswählen
( 2) String[] buttons = {"Si","No","Cancel"};
( 3) // Initial belegte Schaltfläche
( 4) String init = "Si";
( 5) // Auswahldialog darstellen
( 6) JOptionPane.showOptionDialog(
( 7)         contentPane,                                  ← Erscheinungsort
( 8)         „Ciao!",                                      ← Text zur Bestätigung
( 9)         "Swing: Demo eines Optionendialogs",          ← Titelzeile
(10)         JOptionPane.YES_NO_CANCEL_OPTION,             ← Schaltflächen
(11)         JOptionPane.INFORMATION_MESSAGE,              ← Dialogtyp
(12)         new ImageIcon(„smile.jpg"),                   ← Verwendetes Icon
(13)         buttons,                                      ← Schaltflächentexte
(14)         init);                                        ← Initiale Belegung
```

Optionendialoge (III): Beispiel (II)

- Demo: Beispiel 11
- Resultat:



Toggle-Buttons (I)

- Klasse `JToggleButton`
- Objekte können zwei Zustände annehmen
- Beibehalten Zustand, bis erneute Betätigung Button

Toggle-Buttons (II): Beispiel (I)

```
( 1) // Anlegen Icons
( 2) ImageIcon icon1 = new ImageIcon(„smile.jpg");
( 3) ImageIcon icon2 = new ImageIcon(„smile_Traurig.jpg");
( 4) // Anlegen ToggleButton
( 5) JToggleButton toggleSF = new JToggleButton(icon1);
```

- Umschalten durch Abfangen Ereignis:

```
( 1) public void actionPerformed(ActionEvent a){
( 2)   if (toggleSF.isSelected() == true)
( 3)     {toggleSF.setIcon(icon2);
( 4)   }
( 5)   else{
( 6)     toggleSF.setIcon(icon1);
( 7)   }
( 8) }
```

Toggle-Buttons (III): Beispiel (II)

- Demo: Beispiel 12
- Resultat:

Checkboxen (I)

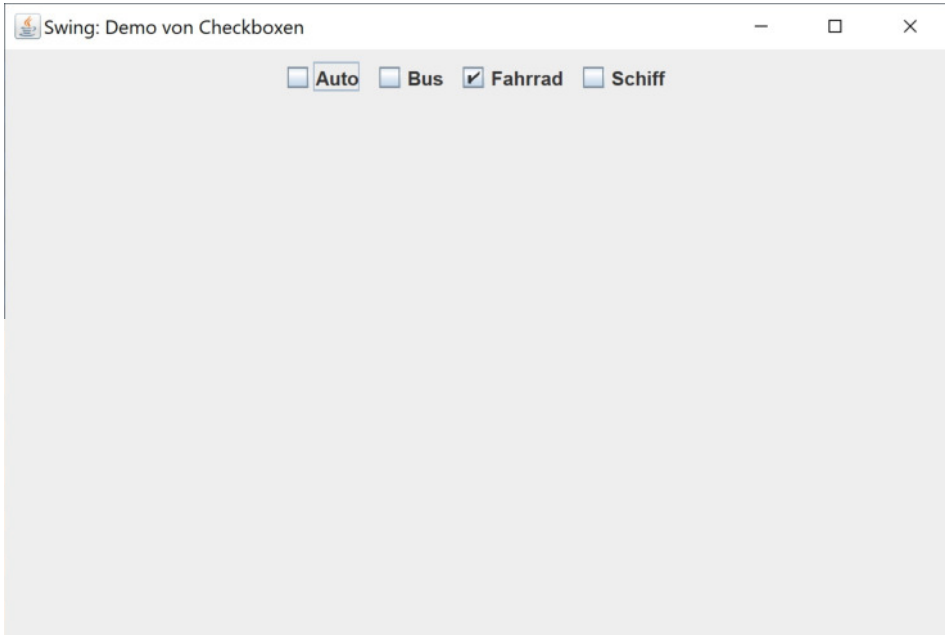
- Klasse `JCheckBox`
- Objekte können zwei Zustände annehmen
- Beibehalten Zustand, bis erneute Betätigung Button
- Unterschied zu `ToggleButton`-Objekten: Verwendung in Verbindung mit Texten
- Beispiel:

```
( 1) // Anlegen Checkboxen
( 2) JCheckBox checkBox_1 = new JCheckBox("Auto");
( 3) JCheckBox checkBox_2 = new JCheckBox("Bus");
( 4) JCheckBox checkBox_3 = new JCheckBox("Fahrrad", true);
( 5) JCheckBox checkBox_4 = new JCheckBox("Schiff", false);
```

- Erster Parameter: Text
- Zweiter (optionaler) Parameter: initiale Markierung

Checkboxen (II): Beispiel

- Demo: Beispiel 13
- Resultat:



Radio-Buttons (I)

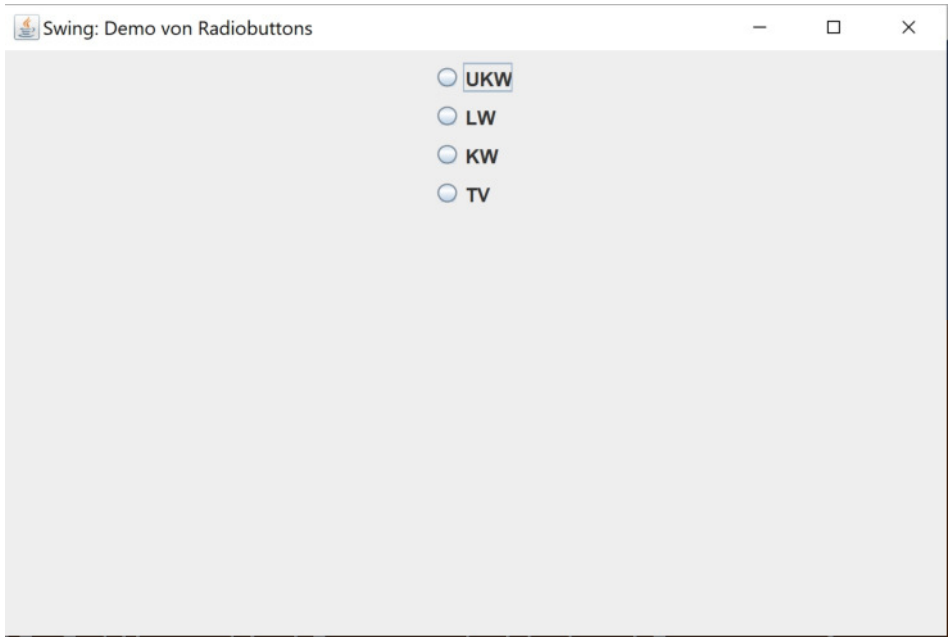
- `RadioButton`-Objekt ähnlich `CheckBox`-Objekt
- Zusammenfassung zu Gruppe möglich
- `RadioButton`-Objekt Gruppe von Radiobuttons
- Zu einem Zeitpunkt höchstens ein Button der Gruppe gesetzt
- Bei Aktivierung Button innerhalb Gruppe $\Rightarrow$ Deaktivierung des zuletzt aktivierten Button

Radio-Buttons (II): Beispiel (I)

```
( 1) // Anlegen Radiobuttons
( 2) JRadioButton radioButton_1 = new JRadioButton("UKW");
( 3) JRadioButton radioButton_2 = new JRadioButton("LW");
( 4) JRadioButton radioButton_3 = new JRadioButton("KW");
( 5) JRadioButton radioButton_4 = new JRadioButton("TV");
( 6) // Anlegen Gruppe
( 7) ButtonGroup gruppe = new ButtonGroup();
( 8) // Hinzufuegen zu Radiobuttongruppe
( 9) // Geht leider nicht so einfach: contentPane.add(gruppe);
(10) // Stattdessen:
(11) Box b = Box.createVerticalBox();
(12) contentPane.setLayout(new FlowLayout());
(13) b.add(radioButton_1);
(14) b.add(radioButton_2);
(15) b.add(radioButton_3);
(16) b.add(radioButton_4);
(17) b.add(Box.createVerticalStrut(10));
(18) contentPane.add(b);
```

Radio-Buttons (III): Beispiel (II)

- Demo: Beispiel 14
- Resultat:



Übersicht

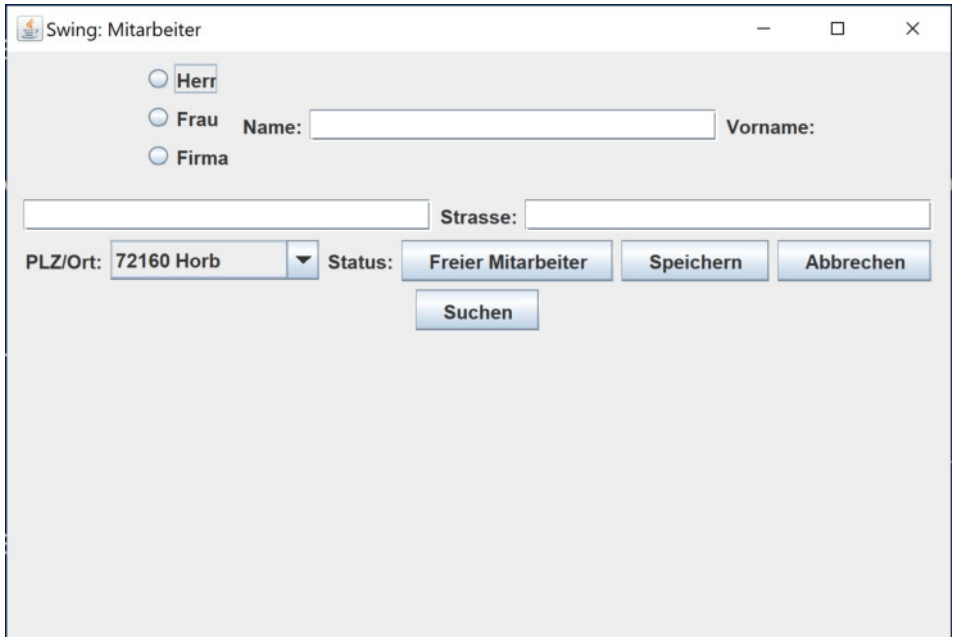
- Grundlagen
- Aufbau von Swing-Applikationen
- Container
- Komponenten, Klassen, Pakete
- Wichtige Komponenten
- Beispiel

Anforderungen

- Fenster zur Verwaltung von Mitarbeitendendaten:
  - Anrede kann „Herr“, „Frau“ oder „Firma“ sein ← RadioButton
  - Name, Vorname und Straße sollen Freitextfelder sein ← Textfelder
  - Auswahl PLZ und Ort aus Liste möglicher Kombinationen ← ComboBox
  - Statusfeld mit Werten „freie Mitarbeit“ oder „festangestellt“ ← ToggleButton
  - Drei Schaltflächen zum Speichern, Abbrechen und Suchen ← Buttons

Beispiel

- Demo: Beispiel 15
- Resultat:



Bewertung

- Darstellung aller gewünschten Informationen
- Aber noch fehlend:
  - Interaktion, z.B. nach Betätigen von „Suchen“
  - Layout, d.h. Anordnung der Komponenten
  - Aussehen

Zusammenfassung (I)

- Grundlagen:
  - AWT, Swing, JFC
- Aufbau von Swing-Applikationen:
  - Container, Komponenten, Ereignisse
- Container:
  - Top-Level-Container (Rahmen, Dialog, Applet)
  - General Purpose Container (Panel, Scroll Pane, Split Pane, Tabbed Pane, Toolbar)
  - Special Purpose Container (Internal Frame, Layered Pane, Root Pane)
- Komponenten, Klassen, Pakete:
  - AWT-ähnlich und Swing-spezifische
  - Übersicht über Vererbungshierarchie

Zusammenfassung (II)

- Wichtige Komponenten
  - Schaltflächen
  - Beschriftungen
  - Textfelder
  - Menüs
  - Dialoge (Informationsdialoge, Eingabedialoge, Auswahldialoge, Optionendialoge)
  - ToggleButtons
  - Checkboxen
  - Radiobuttons

- Beispiel

Aufgabe

- **Aufgabe**

  Bilden Sie den folgenden Screenshot (in etwa) mit Java Swing nach!