Kapitel 4: GUI-Programmierung

AWT und Swing

- Klassensammlungen zur Programmierung grafischer Benutzeroberflächen
- AWT
  - *Alternative Window Toolkit*
  - Seit JDK 1.0
  - Package `java.awt`
- Swing
  - Seit JDK 1.2
  - Package `javax.swing`
  - Mächtiger als AWT
  - Baut auf AWT auf
  - Viele Klasse beginnen zur Abgrenzung gegenüber AWT mit „J“ (z.B. `JButton`)
- In der Vorlesung wird vorwiegend Swing betrachtet.

Swing-Komponenten

- Fenster (`JFrame`, `JDialog`)
- Menüs (`JMenuBar`, `JMenu`, `JMenuItem`)
- Schaltflächen (`JButton`)
- Textfelder (`JTextField`)
- Bezeichnungsfelder (`JLabel`)
- Comboboxen (`JComboBox`)
- Checkboxen (`JCheckBox`)
- Radiobuttons (`JRadioButton`)
- Listboxen (`JList`)
- Tabellen (`JTable`)
- Messageboxen (`JOptionPane`)

javax.swing

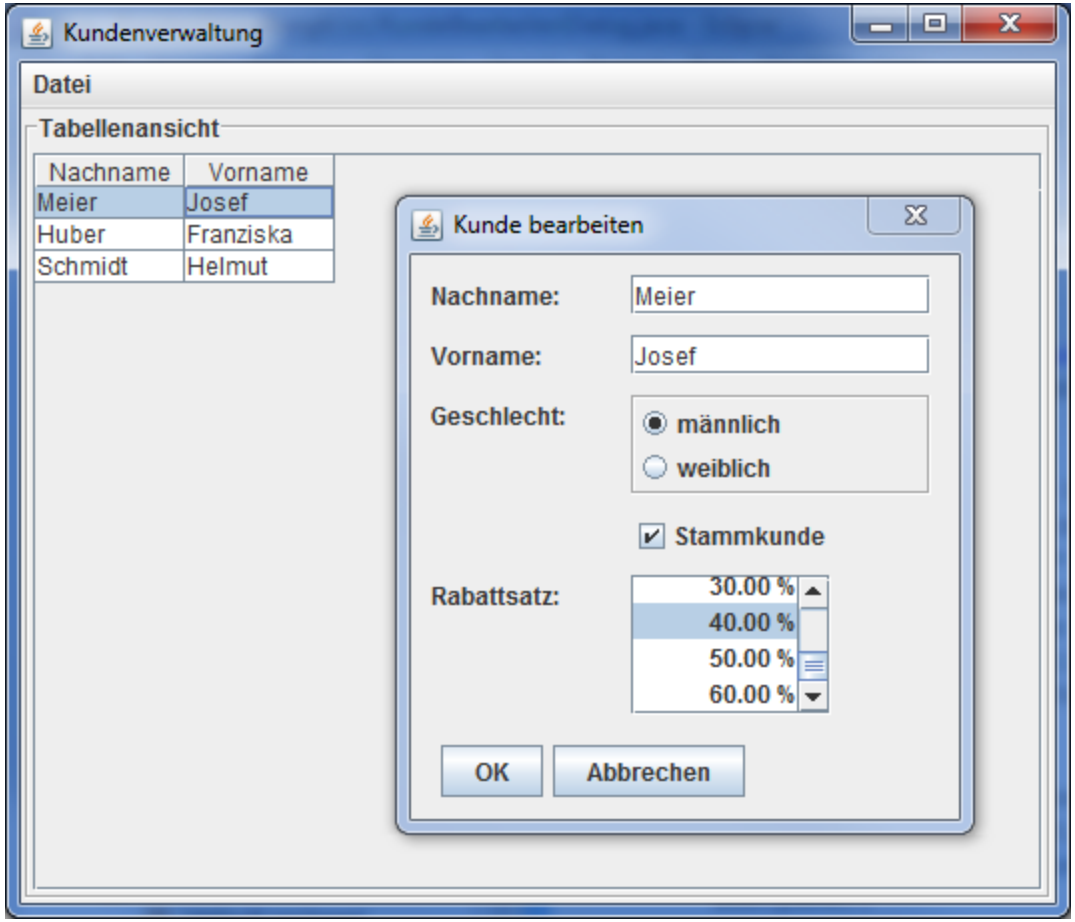
- java.lang.Object
  - java.awt.Component
    - java.awt.Container
      - java.awt.Window
        - java.awt.Frame
          - javax.swing.**JFrame**
        - java.awt.Dialog
          - javax.swing.**JDialog**
      - javax.swing.JComponent
        - javax.swing.AbstractButton
          - javax.swing.**JButton**
          - javax.swing.**JMenuItem**
            - javax.swing.**JMenu**
          - javax.swing.JToggleButton
            - javax.swing.**JCheckBox**
            - javax.swing.**JRadioButton**
        - javax.swing.**JLabel**
        - javax.swing.**JList**
        - javax.swing.**JMenuBar**
        - javax.swing.**JPanel**
        - javax.swing.**JScrollPane**
        - javax.swing.**JSeparator**
        - javax.swing.**JTable**
        - javax.swing.text.JTextComponent
          - javax.swing.**JTextField**

Beispiel: KundenVerwaltung

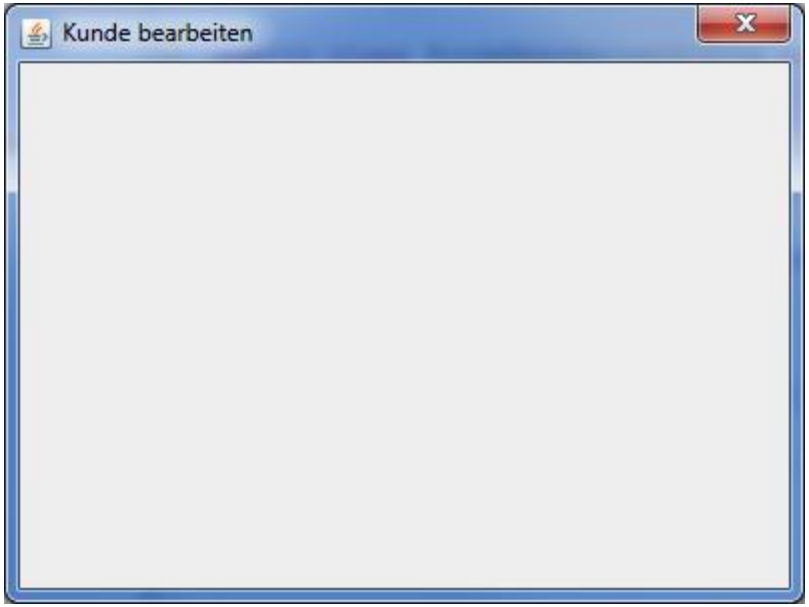
- Anwendung wird nach und nach vorgestellt.
- Anhand dieses durchgängigen Beispiels wird die GUI-Programmierung erläutert.

Ausblick

So soll die KundenVerwaltung in Version 6 einmal aussehen…



KundenVerwaltung v1 (KV v1)



KundenVerwaltung v1

Definiert Schließen-Ereignis (Standardmäßig wird nur ausgeblendet!)

Immer am Ende!

```
import javax.swing.JDialog;

public class KundeBearbeitenDialog extends JDialog {

    public KundeBearbeitenDialog() {

        setTitle("Kunde bearbeiten");
        setSize(400, 300);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args) {

        new KundeBearbeitenDialog();
    }
}
```

Fenster

- `Window` (`java.awt`): Allgemeines Fenster
- `JFrame`: Fenster mit Rahmen und Titelzeile
- `JDialog`: Fenster mit Rahmen und Titelzeile; besitzt i.d.R. ein Parent-Fenster und kann modal sein

Wichtige Methoden

`DO_NOTHING_ON_CLOSE`
`HIDE_ON_CLOSE` (default)
`DISPOSE_ON_CLOSE`
`EXIT_ON_CLOSE` (nicht `JDialog`!)

Methode	**Beschreibung**	**Implementierung**	
		JFrame	**JDialog**
`void setTitle(` `  String title)`	Setzt den Text in der Titelzeile	`Frame`	`Dialog`

`void`
`setDefaultCloseOperation(`
`  int operation)`
	Definiert das Standardverhalten beim Schließen des Fensters	`JFrame`	`JDialog`

`void setLayout(` `  LayoutManager manager)`	Definiert den Layout-Manager zur automatischen Formatierung der Komponenten	`JFrame`	`JDialog`
`void setJMenuBar(` `  JMenuBar menubar)`	Definiert die Menüleiste	`JFrame`	`JDialog`
`Component add(` `  Component component)`	Fügt eine Komponente hinzu	`Container`	

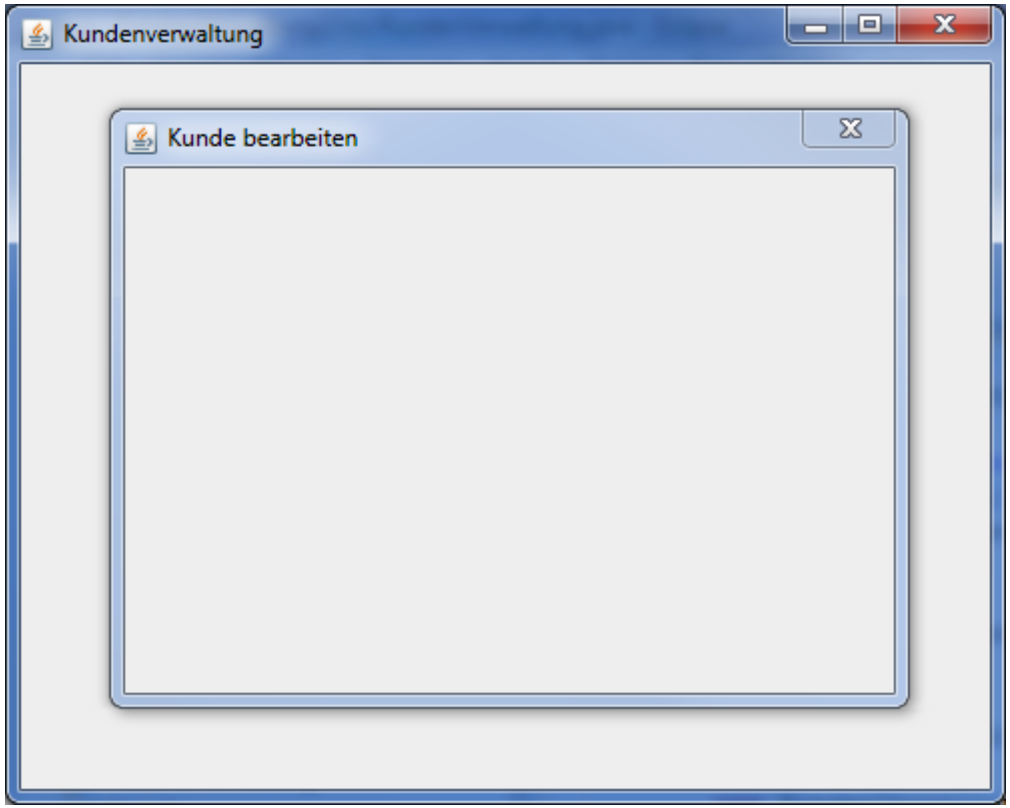
Wichtige Methoden

Methode	Beschreibung	Implementierung	
		JFrame	JDialog
`void setSize(int width, int height)`	Setzt die Größe des Fensters	Window	
`void pack()`	Setzt die Größe des Fensters anhand des Platzbedarfs seiner Komponenten (ggf. nach dem Einfügen der Komponenten setzen!)	Window	
`void setVisible( boolean visible)`	Steuert die Anzeige des Fensters (wird i.d.R. als letzte Anweisung beim Erstellen eines Fensters auf `true` gesetzt!)	Window	Dialog
`void dispose()`	Schließt das Fenster; nach dem letzten anzeigbaren Fenster wird die Anwendung beendet	Window	

`JDialog` only…

Methode	**Beschreibung**	**Implementierung**	
		`JFrame`	**`JDialog`**
`JDialog(Dialog owner)`	`owner` legt den übergeordneten Dialog fest	`---`	`JDialog`
`JDialog(Frame owner)`	`owner` legt den übergeordneten Frame fest	`---`	`JDialog`
`void setModal(` `  boolean modal)`	Definiert den Dialog als modal	`---`	`Dialog`

KundenVerwaltung v2



KundenVerwaltung v2

Definiert den Besitzer des Dialogs

Der Dialog ist modal.

```
public class KundeBearbeitenDialog extends JDialog {

   public KundeBearbeitenDialog(JFrame parent) {
      super(parent);

      setModal(true);
      setTitle("Kunde bearbeiten");
      setSize(400, 300);
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
      setVisible(true);
   }
}
```

KundenVerwaltung v2

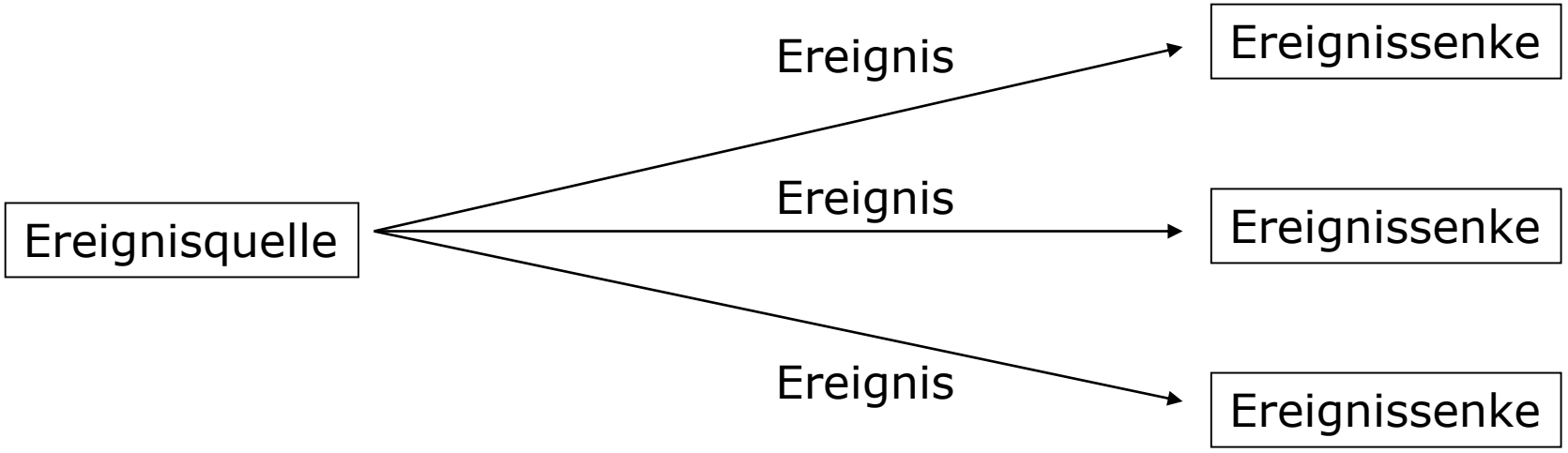
Hier ggf. weitere Komponenten des Frames hinzufügen.

```
public class KundenVerwaltung extends JFrame {

   public KundenVerwaltung() {
      setTitle("Kundenverwaltung");
      setSize(500, 500);
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);

      setVisible(true);

      new KundeBearbeitenDialog(this);
   }

   public static void main(String[] args) {

      new KundenVerwaltung();
   }
}
```

Ereignisbehandlung

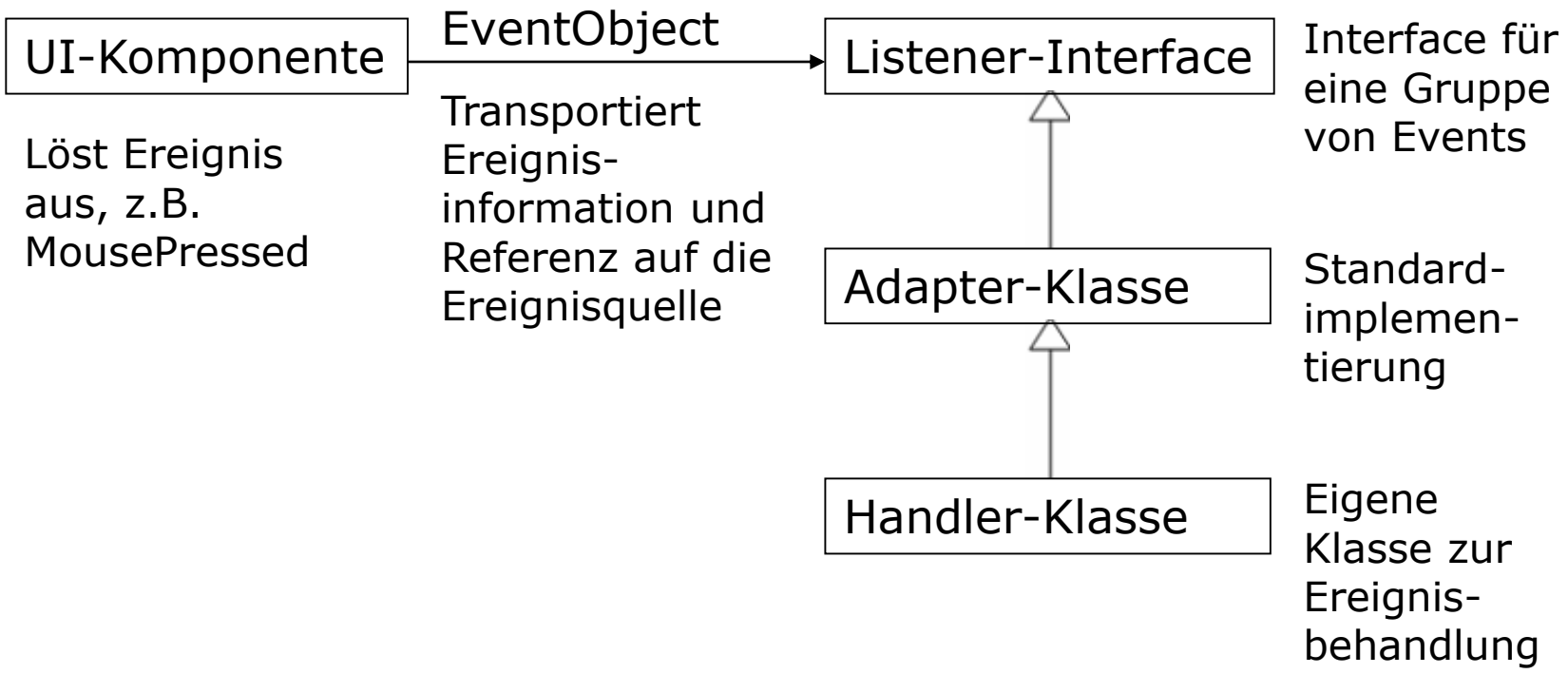
- Generell kommunizieren Komponenten in GUI-Frameworks oftmals über Ereignisse.
- Die Unterstützung von Ereignissen bedeutet, dass Komponenten beliebigen Interessenten Mitteilungen über Zustandsänderungen senden können, ohne an diese eng gekoppelt zu sein.

Ereignisbehandlung



- Löst Ereignis aus
- Sendet Ereignis-Objekte an alle Interessenten bzw. Ereignissenken
- Transportieren Informationen über das ausgelöste Ereignis
- Verarbeiten das Ereignis

Ereignisbehandlung in Swing



Ereignisbehandlung in Swing

- Sämtliche Listener implementieren das Interface `EventListener` (hat keine Methoden deklariert)
- Events – von der Klasse `EventObject` abgeleitet – beinhalten Informationen über das ausgelöste Ereignis und identifizieren die Ereignisquelle (`getSource()`).
- Adapter-Klassen implementieren die Methoden des Listeners standardmäßig. Somit müssen nur relevante Methoden überschrieben werden. Adapter-Klassen existieren (selbstverständlich) nur für Listener mit mehr als einer Methode.
- Details siehe „Lesson: Writing Event Listeners“: https://docs.oracle.com/javase/tutorial/uiswing/events/index.html

Ausgewählte Listener

Listener	Adapter	Event	Methoden
`ActionListener`		`ActionEvent`	`actionPerformed()`
`ItemListener`		`ItemEvent`	`itemStateChanged()`

`MouseListener`	`MouseAdapter`	`MouseEvent`	`mouseClicked()`
			`mouseEntered()`
			`mouseExited()`
			`mousePressed()`
			`mouseReleased()`

`MouseMotionListener`	`MouseMotionAdapter`	`MouseEvent`	`mouseDragged()` `mouseMoved()`

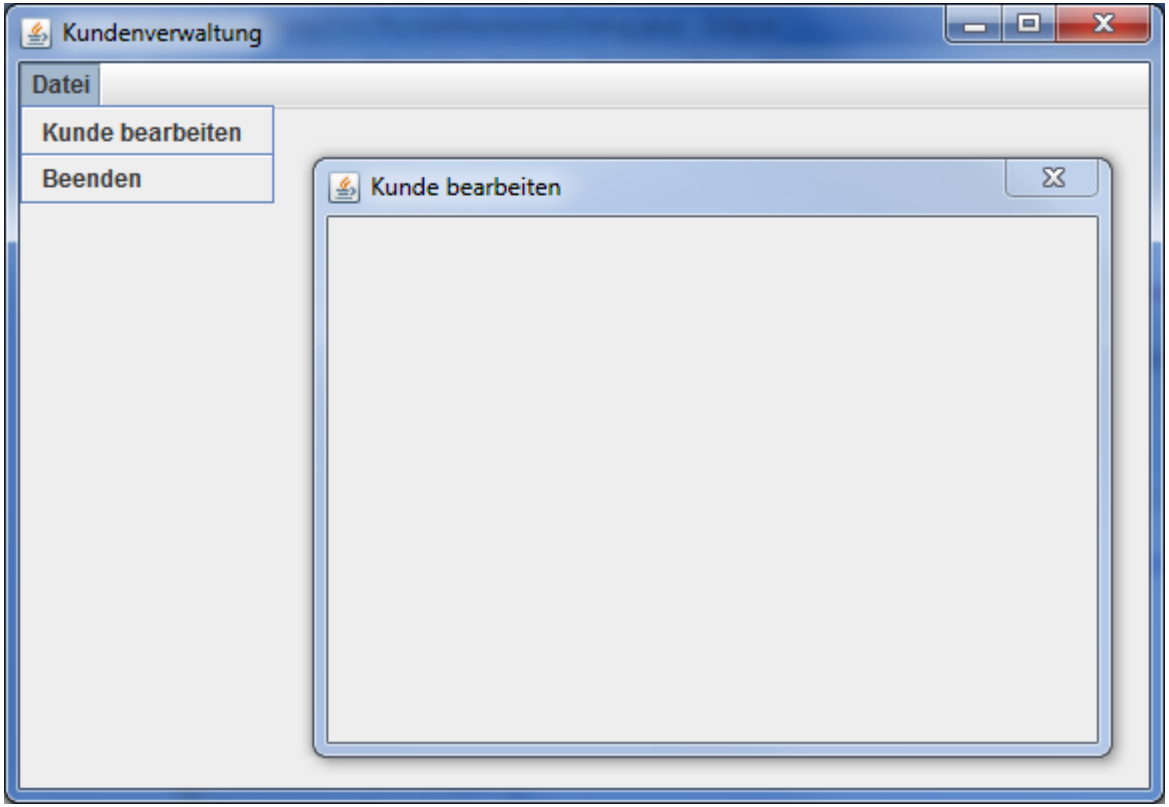
`WindowListener`	`WindowAdapter`	`WindowEvent`	`windowActivated()`
			`windowClosed()`
			`windowClosing()`
			`windowDeactivated()`
			`windowDeiconified()`
			`windowIconified()`
			`windowOpened()`

Anwendung der Listener

- `MouseListener` und `MouseMotionListener` können auf alle Swing-Komponenten angewendet werden.
- Die weiteren Listener sind auf die Anwendung bestimmter Komponenten beschränkt:

Komponente	**ActionListener**	**ItemListener**	**WindowListener**
JButton	x		
JCheckBox	x	x	
JComboBox	X	x	
JDialog			X
JFrame			X
JMenuItem	x	x	
JRadioButton	x	x	
JTextField	x		

KundenVerwaltung v3



KundenVerwaltung v3

```
JMenuBar menubar = new JMenuBar();
setJMenuBar(menubar);

JMenu menu;
JMenuItem mi;

menubar.add(menu = new JMenu("Datei"));

menu.add(mi = new JMenuItem("Kunde bearbeiten"));
mi.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      new KundeBearbeitenDialog(KundenVerwaltung.this);
   }
});

menu.add(new JSeparator());

menu.add(mi = new JMenuItem("Beenden"));
mi.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      dispose();
   }
});
```

Der Frame ist die „äußere“ Klasse…

Menüs

- `JMenu`-Objekte lassen sich beliebig verschachtelt.
- Auf unterster Ebene befinden sich i.d.R. ausschließlich `JMenuItem`-Objekte
- Dadurch entsteht eine **Baumstruktur**, konkreter ein Menübaum, mit `JMenuItem`-Objekten i.d.R. als Blätter (Leafs).
- Im Übrigen bilden auch viele andere Komponenten in GUIs eine Baumstruktur (z.B. Fenster/Tabelle/Textfeld)

Wichtige Methoden zu Menüs

- `JMenuBar`

`JMenu addJMenu(JMenu m)`	Anhängen eines Menüs an der `JMenuBar`

- `JMenu`

`JMenu(String text)`	K. mit dem Beschriftungstext als Parameter
`JMenuItem add(JMenuItem mi)`	Anhängen eines Menüeintrages
`void addSeparator()`	Anhängen eines Trennstriches

- `JMenuItem`

`JMenuItem(String text)`	K. mit dem Beschriftungstext als Parameter
`void addActionListener( ActionListener l)`	Anhängen eines `ActionListener`s (wird bei der Auswahl des Menüeintrages aufgerufen)

[bs] Design-Patterns

- Ein Pattern (Muster) ist eine bewährte Lösung eines Problems in einem bestimmten Kontext.
- Patterns wurden erstmals von Christopher Alexander in den 1970er Jahren für das Architekturumfeld definiert.
- Entwurfsmuster in der Softwaretechnik wurden erstmals 1995 im Buch *Design Patterns* von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides, auch als *Gang-of-Four* bzw. *GoF* bekannt, beschrieben.
- Das Wissen über Patterns erleichtert den Softwareentwurf und die Kommunikation dessen. Es erhöht das Abstraktionsniveau in der Programmierung.

[bs] Design-Patterns

Fett markierte Patterns werden noch näher betrachtet…

- Übersicht der 23 „klassischen“ Design-Patterns der GoF:

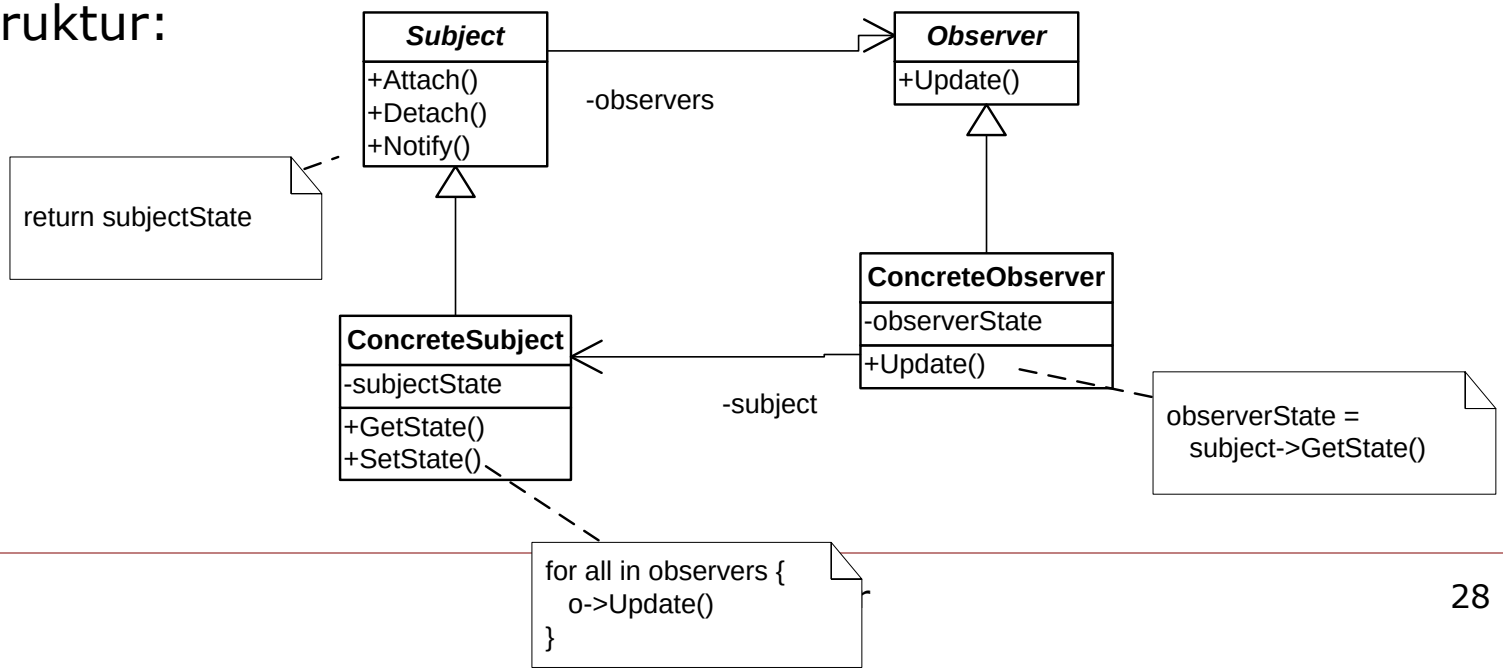
- Creational-Patterns
  - Abstract-Factory
  - Builder
  - Factory-Method
  - Prototype
  - Singleton
- Structural-Patterns
  - Adapter
  - Bridge
  - **Composite**
  - **Decorator**
  - Facade
  - Flyweight
  - Proxy
- Behavioral-Patterns
  - Chain-of-Responsibility
  - Command
  - Interpreter
  - **Iterator**
  - Mediator
  - Memento
  - **Observer**
  - State
  - **Strategy**
  - Template-Method
  - Visitor

[bs] Observer-Pattern

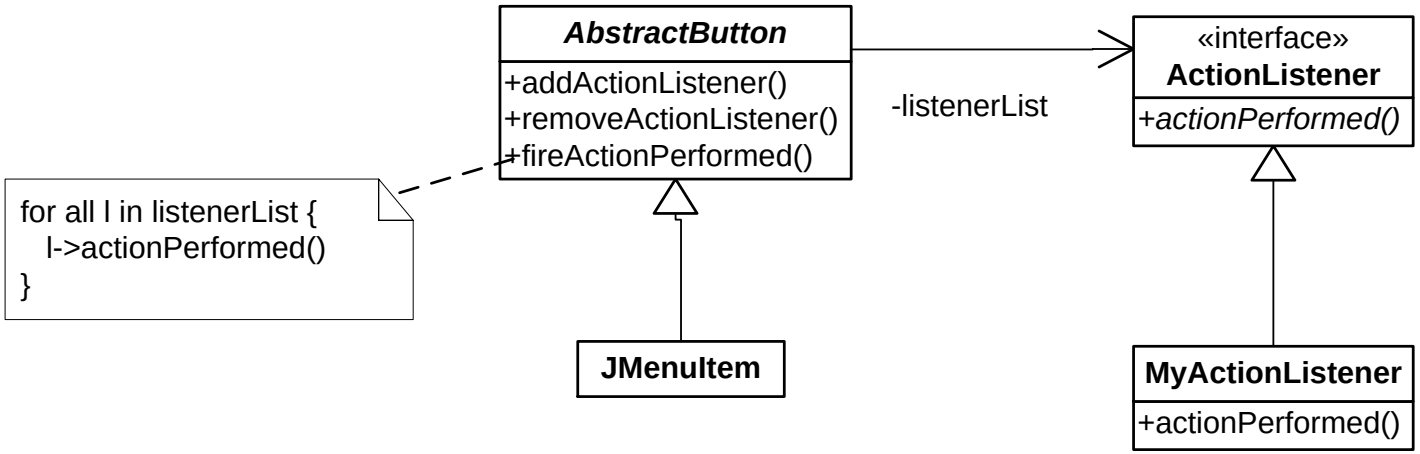
- Zweck:

  Definiert eine Eins-zu-viele-Abhängigkeit zwischen Objekten in der Art, dass alle abhängigen Objekte benachrichtigt werden, wenn sich der Zustand des einen Objekts ändert.

- Struktur:

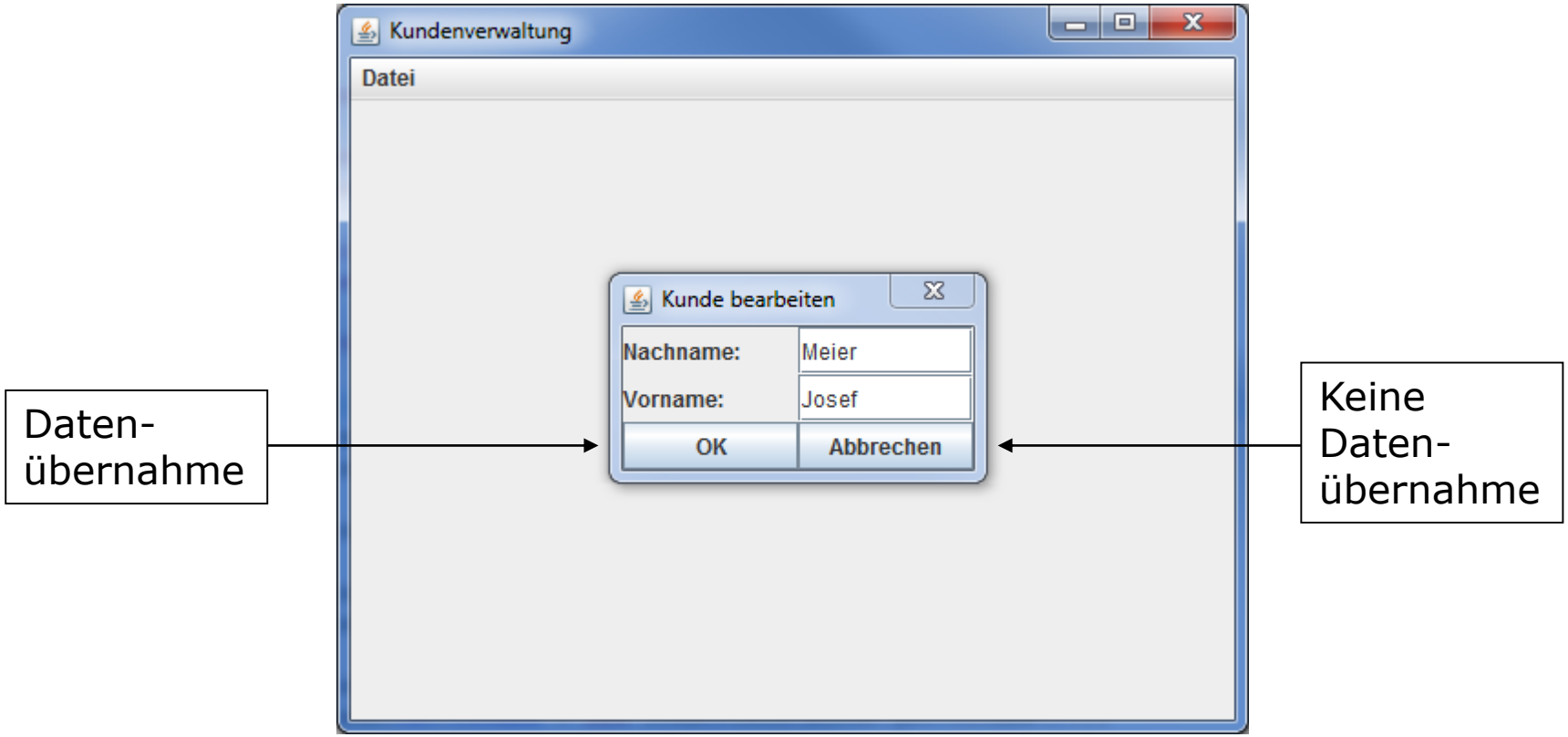


[bs] Beispiel `JMenuItem` - `ActionListener`



- `MyActionListener` benötigt von Vorneherein keine Referenz auf das `JMenuItem`. Er erhält diese als Argument von `actionPerformed()`. Der Parameter vom Typ `ActionEvent` erlaubt den Zugriff per `getSource()` (I.d.R. wird dieser Zugriff bei ActionListenern aber ohnehin nicht gebraucht.)
- Statt `MyActionListener` ist der *ConcreteObserver* eigentlich die anonyme innere Klasse in `KundenVerwaltung`

KundenVerwaltung v4



KundenVerwaltung v4

```
public class Kunde {
    private String nachname;
    private String vorname;

    public Kunde() {
    }
    public Kunde(String nachname, String vorname) {
        this.nachname = nachname;
        this.vorname = vorname;
    }
    public String getNachname() {
        return nachname;
    }
    public void setNachname(String nachname) {
        this.nachname = nachname;
    }
    public String getVorname() {
        return vorname;
    }
    public void setVorname(String vorname) {
        this.vorname = vorname;
    }
}
```

KundenVerwaltung v4

```
public class KundenVerwaltung extends JFrame {

   private Kunde kunde;

   public KundenVerwaltung(Kunde kunde) {
      this.kunde = kunde;
      …
      menu.add(mi = new JMenuItem("Kunde bearbeiten"));
      mi.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            new KundeBearbeitenDialog(KundenVerwaltung.this,
                                      KundenVerwaltung.this.kunde);
         }
      });
      …
   }
   public static void main(String[] args) {
      Kunde kunde = new Kunde("Meier", "Josef");
      new KundenVerwaltung(kunde);
   }
}
```

KV v4

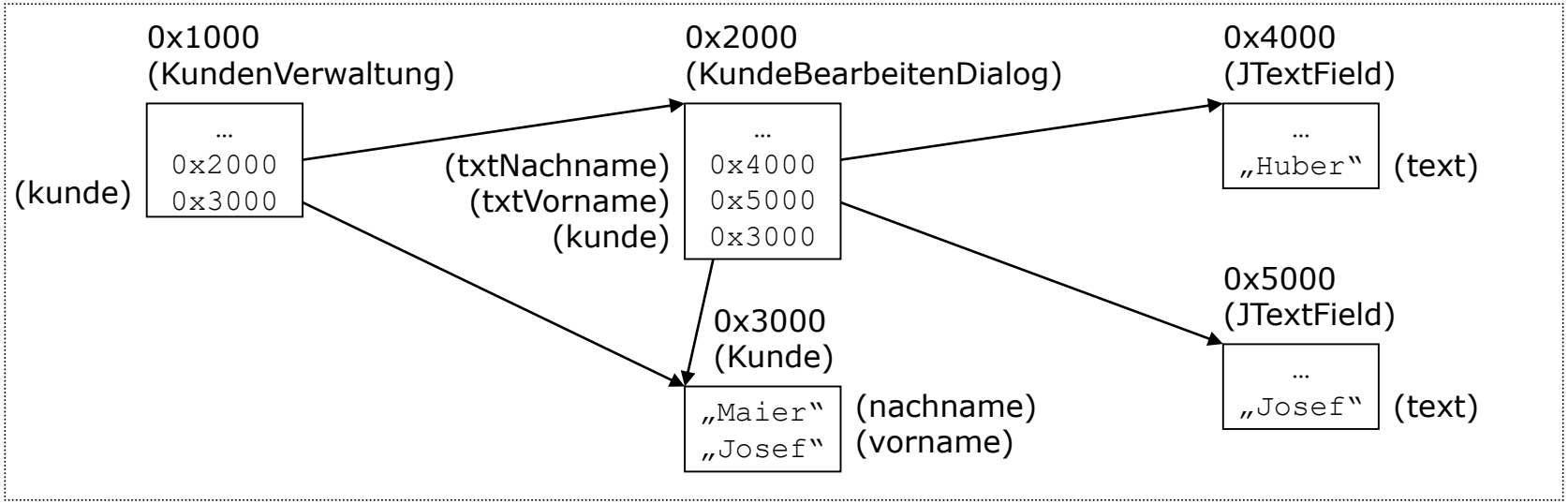
Referenz(!) auf den zu bearbeitenden Kunden

```
public class KundeBearbeitenDialog extends JDialog {
    …
    private Kunde kunde;

    private JTextField txtNachname;
    private JTextField txtVorname;

    public KundeBearbeitenDialog(JFrame parent, Kunde kunde) {
        …
        this.kunde = kunde;
        …
        setLayout(new GridLayout(3, 2));

        add(new JLabel("Nachname:"));
        add(txtNachname = new JTextField(kunde.getNachname()));
        add(new JLabel("Vorname:"));
        add(txtVorname = new JTextField(kunde.getVorname()));

        JButton cmdok = new JButton("OK");
        add(cmdok);
        JButton cmdcancel = new JButton("Abbrechen");
        add(cmdcancel);

        cmdok.addActionListener(new MyOkHandler());
        cmdcancel.addActionListener(new MyCancelHandler());

        pack();
        setVisible(true);
    }
}
```

KundenVerwaltung v4

```
public class KundeBearbeitenDialog extends JDialog {

   class MyOkHandler implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         kunde.setNachname(txtNachname.getText());
         kunde.setVorname(txtVorname.getText());
         dispose();
      }
   }

   class MyCancelHandler implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         dispose();
      }
   }
   …
}
```

KundenVerwaltung v4



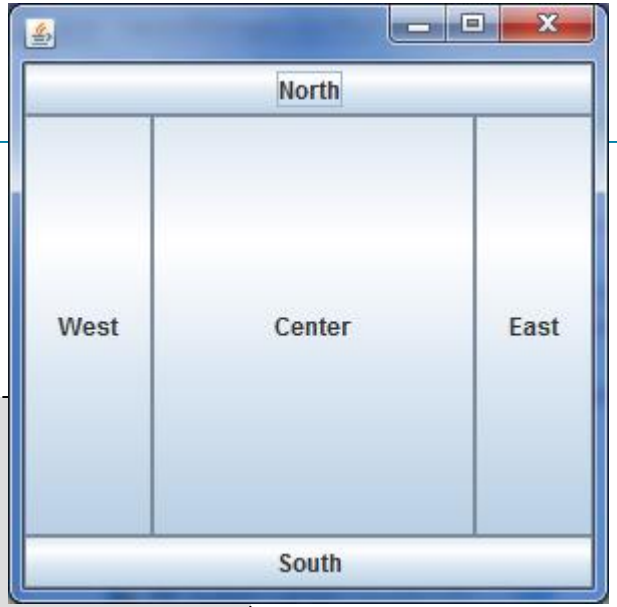
Der Ablauf im Wesentlichen:

- `KundenVerwaltung` erzeugt eine Instanz von `Kunde` und eine Instanz von `KundenBearbeitenDialog`.
- `KundeBearbeitenDialog` erhält diese `Kunde`-Instanz und erzeugt zwei Instanzen von `JTextField` (`txtNachname` und `txtVorname`).
- Die `Kunde`-Instanz wird beim OK-Klicken verändert („Maier" würde durch „Huber" ersetzt werden), beim Abbrechen nicht.

LayoutManager

- LayoutManager bestimmen Größe und Lage von Komponenten relativ zu deren Container-Komponente.
- Die Komponenten werden diesbezüglich dynamisch angepasst.
- Pro Container kann ein LayoutManager frei zugeordnet werden.
- Mit `setLayout()` kann man hierbei einen LayoutManager zuweisen.

BorderLayout

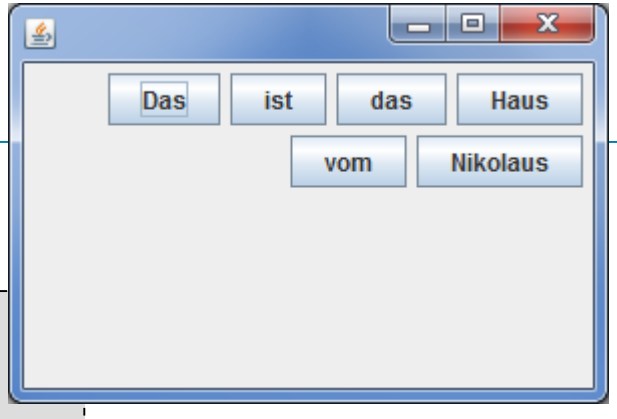



```
public class MyFrame extends JFrame {
    public MyFrame() {
        setSize(300, 300);
        setLayout(new BorderLayout());

        add(BorderLayout.EAST, new JButton("East"));
        add(BorderLayout.SOUTH, new JButton("South"));
        add(BorderLayout.WEST, new JButton("West"));
        add(BorderLayout.NORTH, new JButton("North"));
        add(BorderLayout.CENTER, new JButton("Center"));

        setVisible(true);
    }
}
```

BorderLayout

- Bezeichnung der Bereiche nach Himmelsrichtungen und „Center“
- Nicht verwendete Sektionen werden von den anderen i.d.R. ausgefüllt (außer falls Center nicht genutzt wird)
- Standard-LayoutManager für u.a. `JFrame` und `JDialog`.

FlowLayout




```
public class MyFrame extends JFrame {
    public MyFrame() {
        setSize(350, 200);
        setLayout(new FlowLayout(FlowLayout.RIGHT));

        add(new JButton("Das"));
        add(new JButton("ist"));
        add(new JButton("das"));
        add(new JButton("Haus"));
        add(new JButton("vom"));
        add(new JButton("Nikolaus"));
        setVisible(true);
    }
}
```

LEFT	Linksbündig
CENTER	Zentriert (Default)
RIGHT	Rechtsbündig
LEADING	Sprachabhängig; Bei Links-Rechts-Ausrichtung wie LEFT
TRAILING	Sprachabhängig; Bei Links-Rechts-Ausrichtung wie RIGHT

`FlowLayout`

- Komponenten werden der Reihe nach angeordnet.
- Typischerweise zur Positionierung von Buttons in einem Panel
- Standard-LayoutManager für `JPanel`

GridLayout


```
public class MyFrame extends JFrame {
   public MyFrame() {
      setSize(300, 150);
      setLayout(new GridLayout(2,3));

      add(new JButton("Eins"));
      add(new JButton("Zwei"));
      add(new JButton("Drei"));
      add(new JButton("Vier"));
      add(new JButton("Fünf"));
      add(new JButton("Sechs"));

      setVisible(true);
   }
}
```

Wenn der letzte Button fehlt…

GridLayout

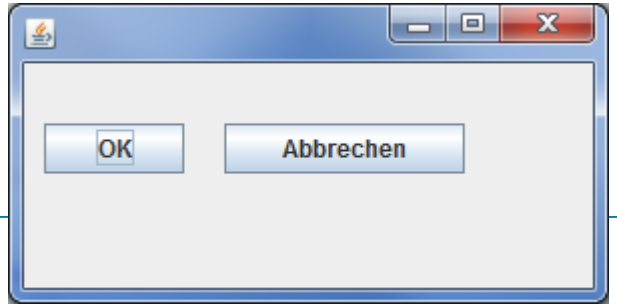
- Matrixartige Anordnung der Komponenten
- Somit besitzen die Komponenten allesamt dieselbe Größe.

Weitere LayoutManager

- `BoxLayout`
- `CardLayout`
- `GridBagLayout`
- `GroupLayout`
- `SpringLayout`

- Siehe „A Visual Guid to Layout Managers“: https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Ohne...




```
public class MyFrame extends JFrame {
    public MyFrame() {
        setSize(300, 150);
        setLayout(null);

        JButton cmdOK = new JButton("OK");
        cmdOK.setBounds(10, 30, 70, 25);
        add(cmdOK);
        JButton cmdCancel = new JButton("Abbrechen");
        cmdCancel.setBounds(100, 30, 120, 25);
        add(cmdCancel);

        setVisible(true);
    }
}
```

Explizit auf `null` setzen, sonst wird der Default-`LayoutManger` angewendet

synonym

```
cmdOK.setLocation(10, 30);
cmdOK.setSize(70, 25);
```

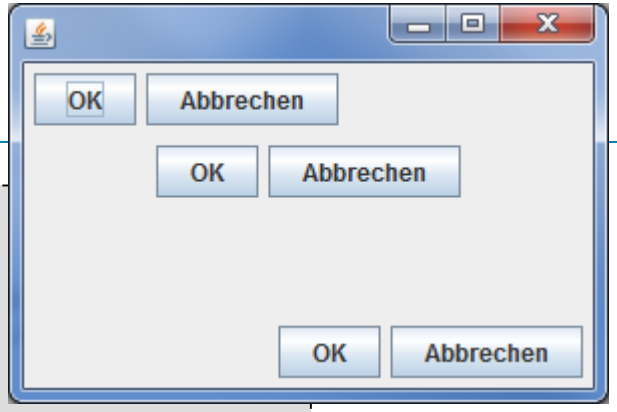
x

y

width

height

Verschachtelt…



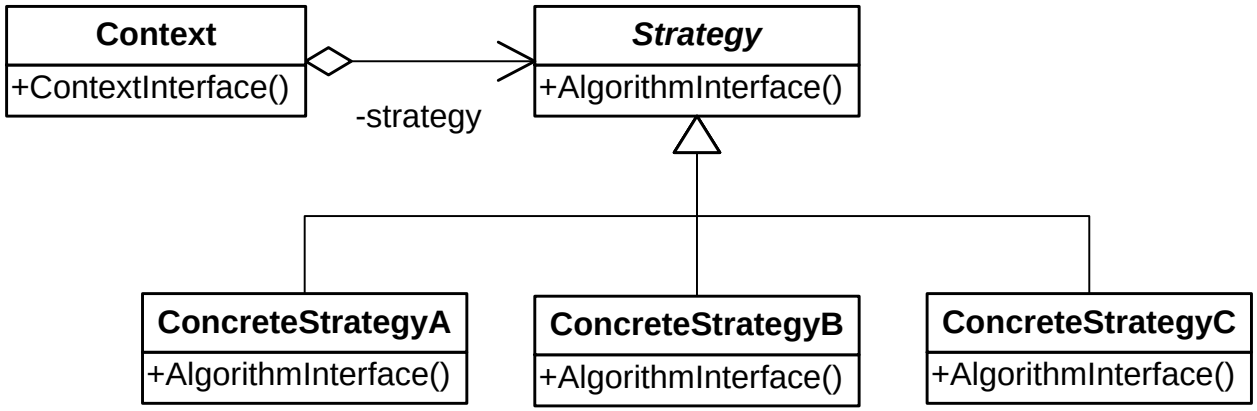

```
public class MyFrame extends JFrame {
   public MyFrame() {
      setSize(300,200);
      JPanel panel;

      add(BorderLayout.NORTH, panel = new JPanel());
      panel.setLayout(new FlowLayout(FlowLayout.LEFT));
      panel.add(new JButton("OK"));
      panel.add(new JButton("Abbrechen"));

      add(BorderLayout.CENTER, panel = new JPanel());
      panel.setLayout(new FlowLayout(FlowLayout.CENTER));
      panel.add(new JButton("OK"));
      panel.add(new JButton("Abbrechen"));

      add(BorderLayout.SOUTH, panel = new JPanel());
      panel.setLayout(new FlowLayout(FlowLayout.RIGHT));
      panel.add(new JButton("OK"));
      panel.add(new JButton("Abbrechen"));

      setVisible(true);
   }
}
```

[bs] Strategy-Pattern

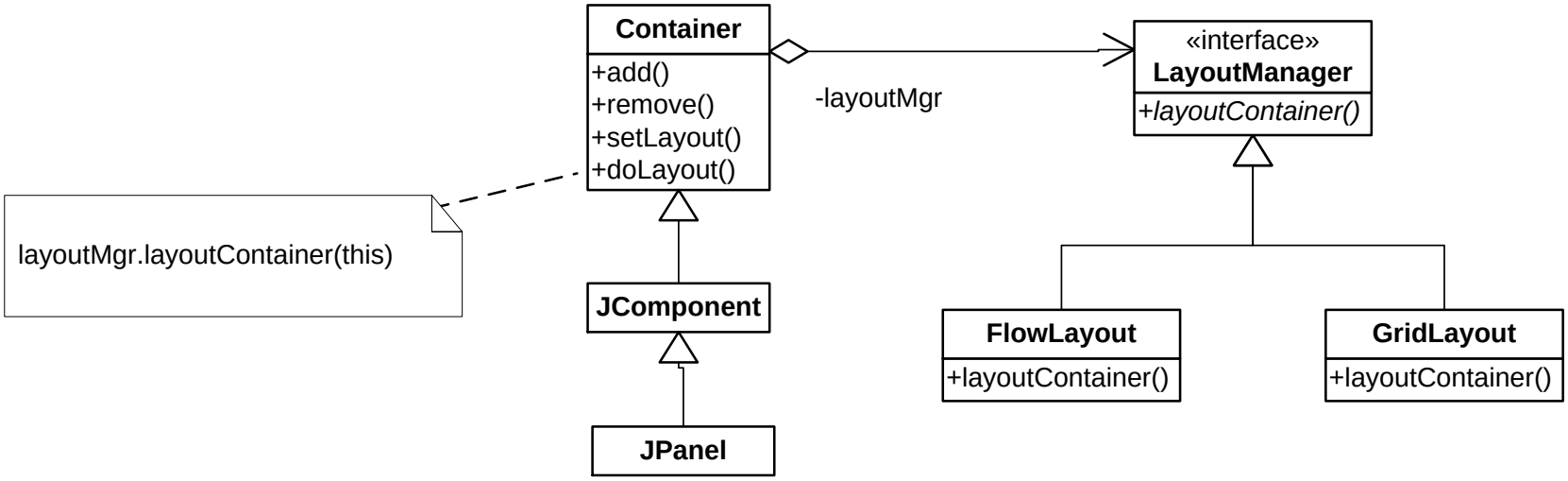
- Zweck:

> Definiert eine Familie von Algorithmen, kapselt sie einzeln und macht sie austauschbar. Strategy ermöglicht es, den Algorithmus unabhängig von den Clients die ihn einsetzen, variieren zu lassen.

- Struktur:



[bs] Beispiel `LayoutManager`



- `LayoutManager` sind *Strategien* für beliebige `Container` wie z.B. `JPanel` zur Bestimmung ihres Layouts (Position, Größe etc.) und deren Komponenten.
- `LayoutManager` bieten zum Festlegen des Layouts die Methode `layoutContainer()` an.

JLabel (Bezeichnungsfeld)

- Wichtige Methoden:

`JLabel(String text)`
`JLabel(String text, int horizontalAlignment)`
`JLabel(String text, Icon icon, int horizontalAlignment)`
`String getText()`
`void setText(String text)`
`Icon getIcon()`
`void setIcon(Icon icon)`
`int getHorizontalAlignment()`
`void setHorizontalAlginment(int alignment)`
`int getVerticalAlignment()`
`void setVerticalAlginment(int alignment)`

- Horizontale Ausrichtung:

`JLabel.LEFT`
`JLabel.CENTER`
`JLabel.RIGHT`
`JLabel.LEADING`
`JLabel.TRAILING`

- Vertikale Ausrichtung:

`JLabel.TOP`
`JLabel.CENTER`
`JLabel.BOTTOM`

`JTextField` (Textfeld)

- Wichtige Methoden:

`JTextField(String text)`	`text`: Initialer Inhalt
`JTextField(int columns)`	`columns`: Anzahl von Spalten zur Berechnung der bevorzugten Breite (wird i.d.R. von Layout-Managern verwendet)
`JTextField(String text, int columns)`	
`String getText()`	Gibt den Inhalt des Feldes zurück
`void setText(String text)`	Setzt den Inhalt des Feldes
`String getSelectedText()`	Gibt den markierten Text des Feldes zurück
`void setEditable(boolean editable)`	Aktiviert oder deaktiviert, dass der Inhalt des Textfeldes bearbeitet werden kann

JButton (Schaltfläche)

- Wichtige Methoden:

`JButton(String text)`	`text`: Beschriftung
`void addActionListener(ActionListener l)`	Fügt einen Handler hinzu, der auf einen Button-Klick regiert.

- Anwendung:

```
JButton b = new JButton("Abbrechen");
add(b);

b.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      dispose();
   }
});
```

JPanel

- `JPanel` stellt einen allgemeinen Container für GUI-Komponenten dar.
- `JPanel` kann genutzt werden, um den Komponenten ein gemeinesames Layout per `LayoutManager` zu geben.
- Die Position der Komponenten ist relativ zur Position des Panels.

- Wichtige Methoden:

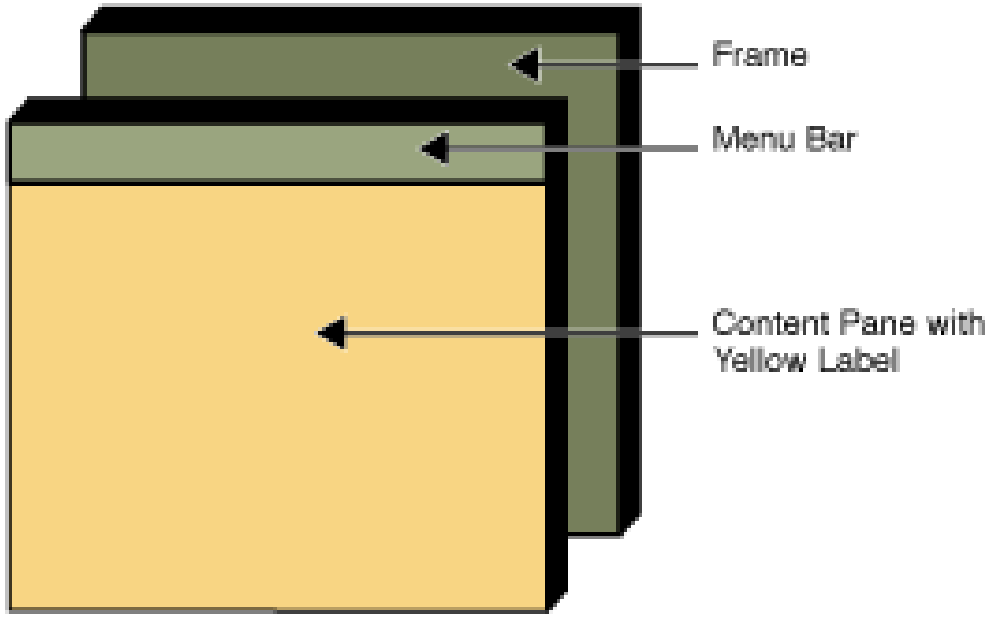
`JPanel()`	Erstellt ein `JPanel` mit `FlowLayout`
`JPanel(LayoutManager layout)`	Erstellt ein `JPanel`; erwartet einen `LayoutManager` als Argument
`void setLayout(LayoutManager layout)`	Setzt einen `LayoutManager`
`Component add(Component component)`	Fügt eine Komponente hinzu

ContentPane und RootPane

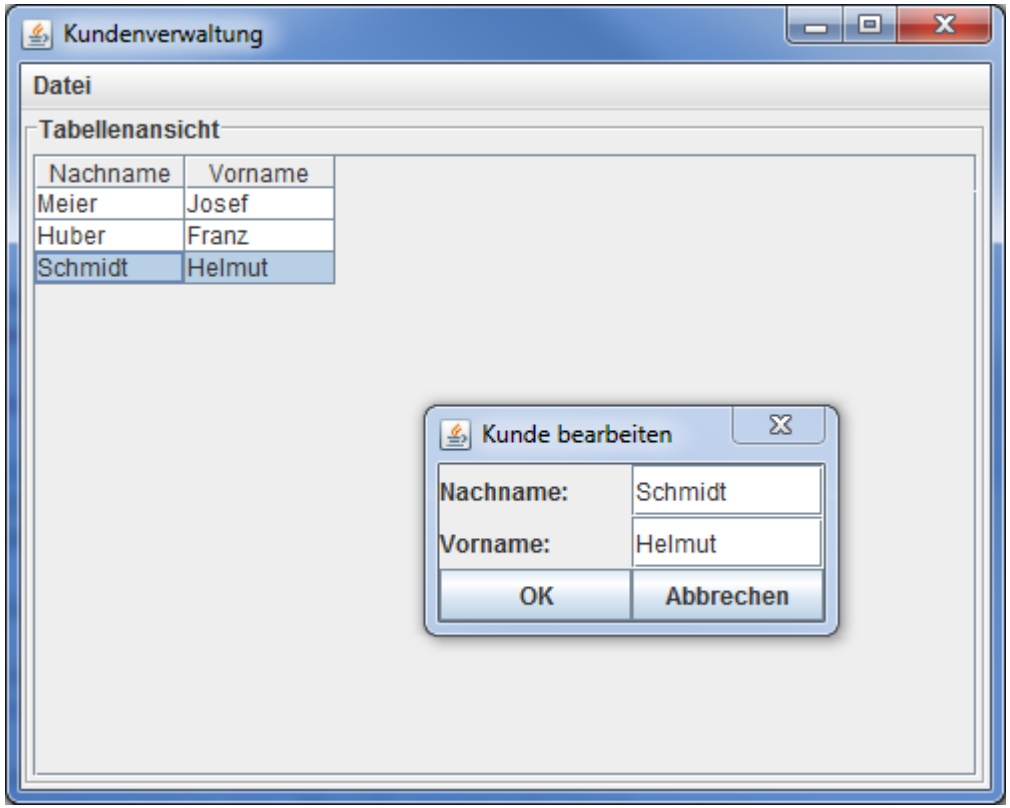
- `JFrame` und `JDialog` besitzen als Wurzelkomponente (Top-Level-Container) ein sog. *RootPane* vom Typ `JRootPane`
- Das RootPane hält ein *ContentPane* und optional eine MenuBar
- Das ContentPane beinhaltet i.d.R. alle UI-Komponenten des Fensters außer der MenuBar
- ContentPane ist standardmäßig für `JFrame`/`JDialog` ein `JPanel`
- Zur einfacheren Handhabung bieten `JFrame`/`JDialog` Methoden wie `add()`, `remove()`, `setLayout()`, die an die entsprechenden Methoden des ContentPane delegieren.

Basisstruktur von Fenstern

- Quelle: „Using Top-Level Containers“: https://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html



KundenVerwaltung v5

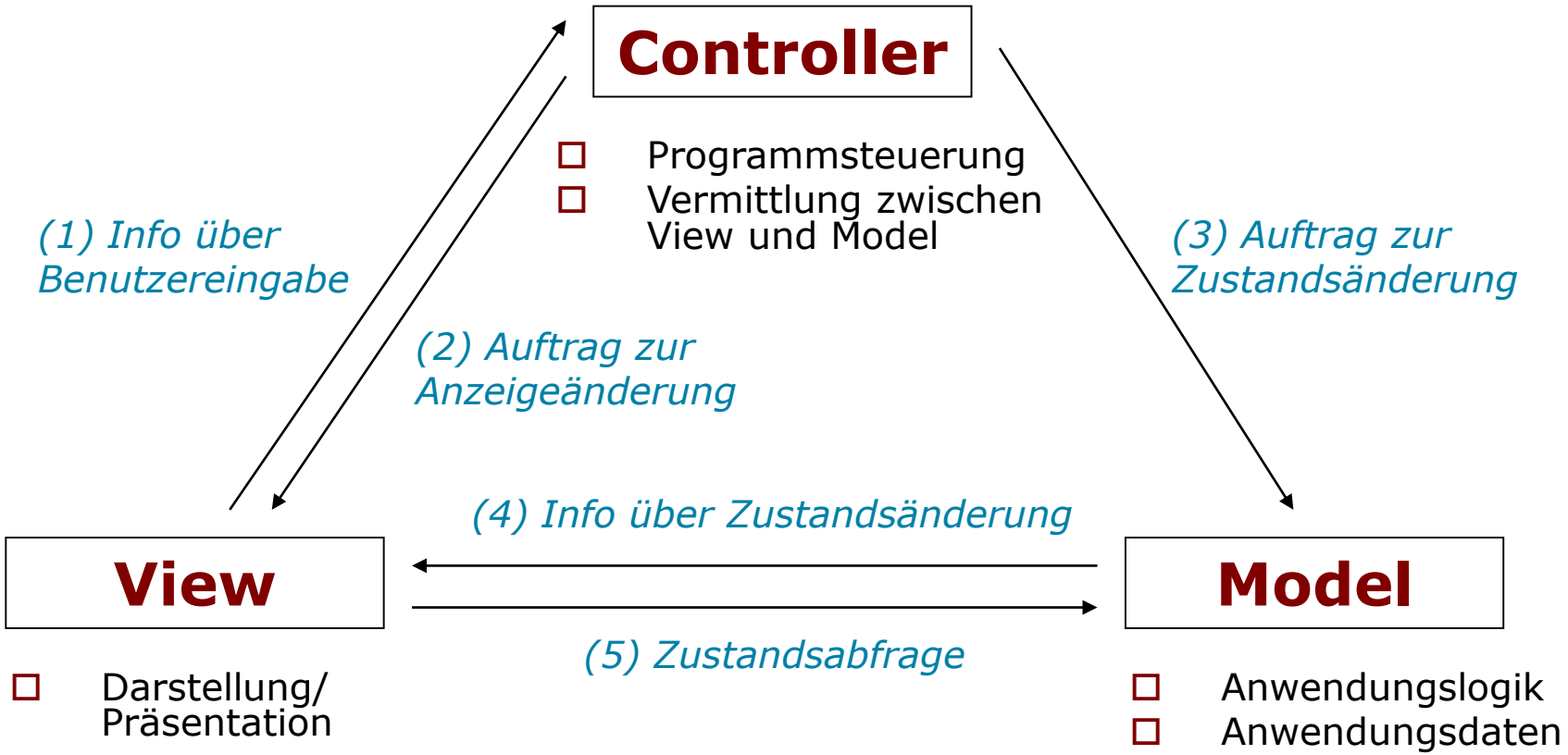


Dialog öffnet sich durch Doppelklick in der Liste oder per Menü

[bs] Model-View-Controller (MVC)

- Trennung von Zuständigkeiten speziell bei Anwendungsdialogen
- MVC forciert damit **Separation-of-Concerns** (SoC; siehe insb. Edsgar W. Dijkstra, 1976, A Discipline of Programming).
- Erstmals wurde MVC 1979 für die Benutzeroberflächen in Smalltalk beschrieben.
- Trennung erfolgt in **M**odel (Anwendungslogik), **V**iew (Präsentation) und **C**ontroller (Programmsteuerung)
- Die Komponenten sind untereinander lose gekoppelt (im Kern mittels Dependency-Injection).

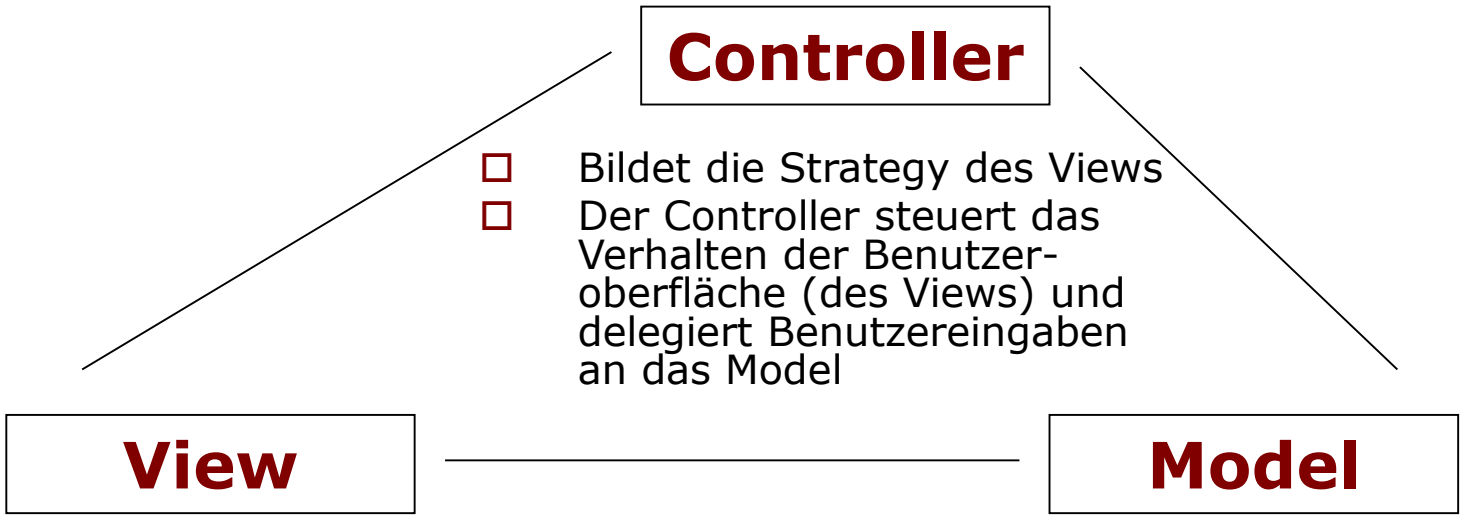
[bs] Kommunikation bei MVC



[bs] MVC als Pattern

- MVC ist ein **Architektur-Pattern**.
- Es ist im Wesentlichen aus den Design-Patterns **Composite** (View), **Strategy** (Controller) und **Observer** (Model) zusammengesetzt.

[bs] Design-Patterns in MVC

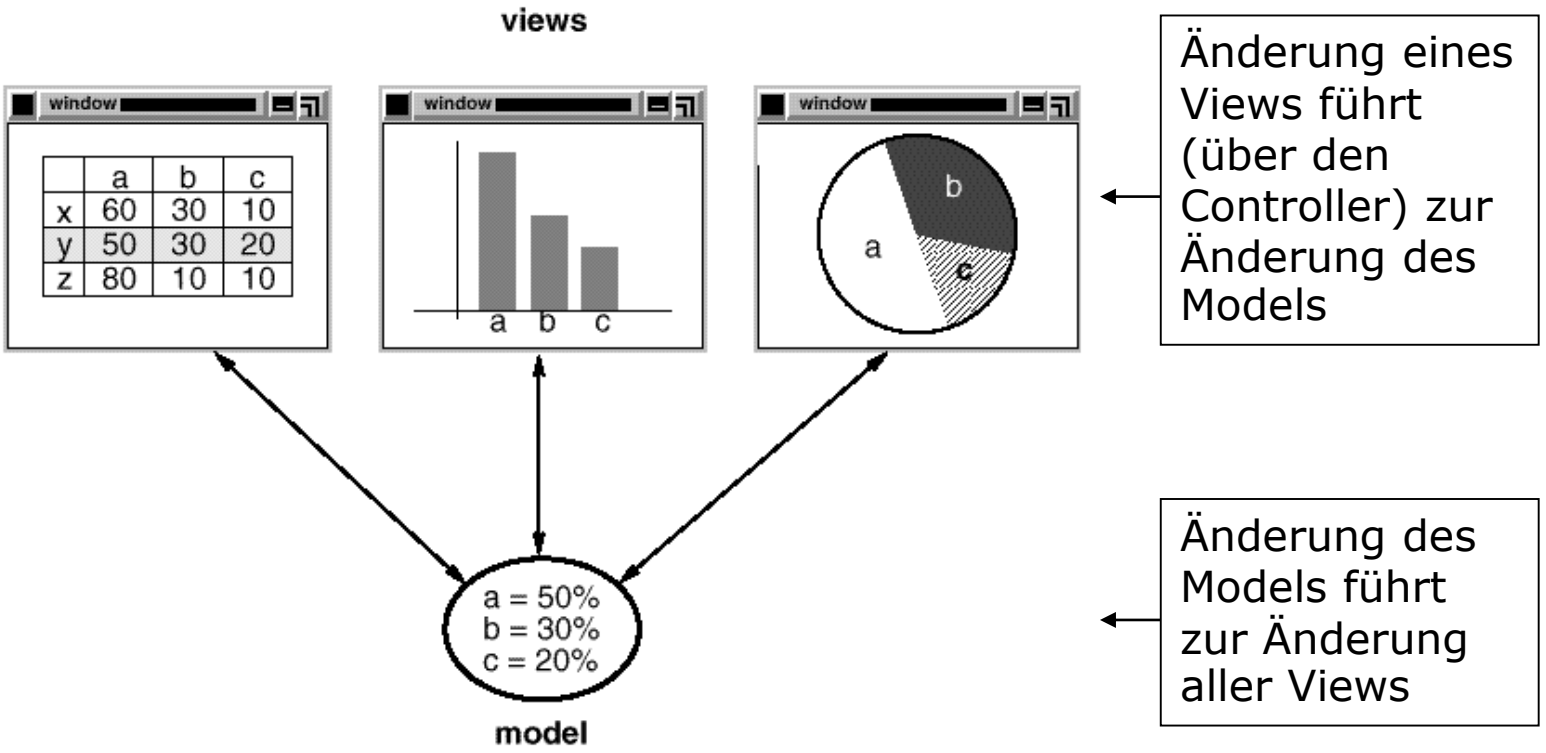


- Implementiert das Composite-Pattern
- GUI-Komponenten bilden eine Baumstruktur
- Z.B. die Aktualisierung der Anzeige löst der Controller über die oberste relevante Komponente aus, der Rest ist Aufgabe des Composite-Patterns

- Bietet einem oder sogar mehreren Views an, sich als Observer für Zustandsänderungen des Models zu registrieren

[bs] Ein Model, viele Views

- Quelle: Design Patterns, 1994, S. 5:



[bs] MVC und Swing

- MVC findet seine Anwendung auf unterschiedlichen Abstraktionsniveaus, insb. für einzelne GUI-Komponenten und konkrete Anwendungsdialoge
- Swing implementiert ein vereinfachtes Modell von MVC, indem es für seine GUI-Komponenten View und Controller zu einem sog. **UI-Delegate** zusammenfasst.
- Siehe „A Swing Architecture Overview“: http://www.oracle.com/technetwork/java/architecture-142923.html

[bs] MVC und Swing

- Das UI-Delegate ist durch die eigentliche Klasse der Swing-Komponente implementiert (z.B. `JTextField`, `JTable`)
- Das Model wird durch einen separaten Typ gekapselt (z.B. `Document`, `TableModel`)
- Das Model kann durch `setModel()` gesetzt und `getModel()` abgefragt werden.
- Alle Swing-Komponenten definieren ein Default-Model, wenn keines explizit angegeben wird.
- Für einfache Komponenten (z.B. `JTextField`) ist oft das Default-Model ausreichend. Auf entsprechende Models kann meist auch bequem über Methoden des UI-Delegate zugegriffen werden (z.B. `setText()`).
- Für komplexere Komponenten (z.B. `JTable`) ist es oft sinnvoll ein eigenes Model zu definieren. Hierfür bietet Swing abstrakte Klassen von denen abgeleitet werden kann (z.B. `AbstractTableModel`). Sie implementieren bereits die Grundfunktionen des Models (z.B. Observer-Funktionalität).

KV v5

```
public class KundeTableModel
                          extends AbstractTableModel {

   private Kunde[] kunden;

   public KundeTableModel(Kunde[] kunden) {
      this.kunden = kunden;
   }
   public String getColumnName(int col) {
      switch (col) {
         case 0: return "Nachname";
         case 1: return "Vorname";
         default: return null;
      }
   }
   public int getRowCount() { return kunden.length; }
   public int getColumnCount() { return 2; }

   public Object getValueAt(int row, int col) {
      Kunde kunde = kunden[row];
      switch (col) {
         case 0: return kunde.getNachname();
         case 1: return kunde.getVorname();
         default: return null;
      }
   }
   …
}
```

TableModel

`String getColumnName(int col)`	Gibt den Namen einer Spalte zurück; Wird als Spaltenüberschrift genutzt
`Class<?> getColumnClass(int col)`	Gibt den Typ einer Spalte zurück; Entsprechend werden Standard-Renderer und –Editoren gewählt
`int getColumnCount()`	Gibt die Spaltenzahl zurück
`int getRowCount()`	Gibt die Zeilenzahl zurück
`Object getValueAt(int row, int col)`	Gibt den Wert einer Zelle zurück
`void setValueAt(Object value, int row, int col)`	Setzt den Wert einer Zelle
`boolean isCellEditable(int row, int col)`	Informiert darüber, ob eine Zelle änderbar ist
`void addTableModelListener( TableModelListener l)`	Fügt einen Handler für Änderungen des Models hinzu
`void removeTableModelListener( TableModelListener l)`	Entfernt einen Handler für Änderungen des Models

Erweiterungen in `AbstractTableModel`

`void fireTableCellUpdated(int row, int column)`	Benachrichtigt Listener, dass sich der Inhalt einer bestimmten Zelle geändert hat
`void fireTableChanged(TableModelEvent e)`	Benachrichtigt Listener über eine Änderung der Tabelle, die durch das TableModelEvent-Argument spezifiziert ist. (Intern verwenden alle anderen fire-Methoden diese Methode)
`void fireTableDataChanged()`	Benachrichtigt Listener, dass sich alle Zellwerte der Tabelle geändert haben können
`void fireTableRowsDeleted(int firstrow, int lastrow)`	Benachrichtigt Listener, dass ein bestimmter Bereich von Zeilen gelöscht wurde
`void fireTableRowsInserted(int firstrow, int lastrow)`	Benachrichtigt Listener, dass ein bestimmter Bereich von Zeilen eingefügt wurde
`void fireTableRowsUpdated(int firstrow, int lastrow)`	Benachrichtigt Listener, dass sich der Inhalt eines bestimmten Bereiches von Zeilen geändert hat.
`void fireTableStructureChanged()`	Benachrichtigt Listener, dass sich die Tabellenstruktur geändert hat

KundenVerwaltung v5

Aspekt: Model-Delegator

```
public class KundenVerwaltung extends JFrame {

   private KundeTableModel kundeTableModel;

   public KundenVerwaltung(Kunde[] kunden) {
      …
      kundeTableModel = new KundeTableModel(kunden);
      JTable table = new JTable(kundeTableModel);
      …
   }

   public static void main(String[] args) {
      Kunde[] kunden = new Kunde[3];
      kunden[0] = new Kunde("Meier", "Josef");
      kunden[1] = new Kunde("Huber", "Franziska");
      kunden[2] = new Kunde("Schmidt", "Helmut");
      new KundenVerwaltung(kunden);
   }
}
```

KV v5

Aspekt: Event-Handling

```
public class KundenVerwaltung extends JFrame {
   private ListSelectionModel kundeSelectionModel;
   private KundeTableModel kundeTableModel;

   public KundenVerwaltung(Kunde[] kunden) {
      …
      menu.add(mi = new JMenuItem("Kunde bearbeiten"));
      mi.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            editKunde();
         }
      });
      …
      kundeTableModel = new KundeTableModel(kunden);
      JTable table = new JTable(kundeTableModel);
      kundeSelectionModel = table.getSelectionModel();
      kundeSelectionModel.setSelectionMode(
         ListSelectionModel.SINGLE_SELECTION);
      table.addMouseListener(new MouseAdapter() {
         public void mousePressed(MouseEvent event) {
            if (event.getClickCount() == 2) editKunde();
         }
      }); …
   }
   private void editKunde() {
      int row = kundeSelectionModel.getMinSelectionIndex();
      new KundeBearbeitenDialog(this, kundeTableModel.getKunde(row));
      kundeTableModel.fireTableRowsUpdated(row, row);
   }
   …
}
```

Das Model gibt dem View bescheid, dass sich die Daten geändert haben.

JTable

`JTable(TableModel model)`	Initialisiert ein `JTable` mit einem `TableModel`
`TableColumnModel getColumnModel()`	Gibt das `TableColumnModel` zurück, das alle Spalteninformationen der Tabelle beinhaltet
`TableModel getModel()`	Gibt das `TableModel` zurück, das den Tabelleninhalt verwaltet
`RowSorter <? extends TableModel> getRowSorter()`	Gibt den `RowSorter` zurück, der bestimmt, in welcher Reihenfolge die Zeilen der Tabelle angezeigt werden
`ListSelectionModel getSelectionModel()`	Gibt das `ListSelectionModel` zurück, das Informationen über selektierte Zeilen verwaltet
`void setAutoCreateRowSorter( boolean docreate)`	Bestimmt, ob ein Standard-RowSorter bereitgestellt werden soll.
`void setAutoResizeMode(int mode)`	Festlegung über die automatische Anpassung der Spaltenbreiten, u.a.: `AUTO_RESIZE_OFF`, `AUTO_RESIZE_ALL_COLUMNS`

TableColumnModel und TableColumn

- TableColumnModel

`void addColumn(TableColumn column)`	Fügt eine Spalte hinzu
`TableColumn getColumn(int index)`	Gibt eine bestimmte Spalte zurück
`void moveColumn(int columnindex, newindex)`	Verschiebt eine Spalte von `columnindex` nach `newindex`
`void removeColumn(TableColumn column)`	Entfernt eine Spalte

- TableColumn

`void setCellEditor( TableCellEditor editor)`	Definiert einen Editor zur Dateneingabe für die Zellen der Spalte
`void setCellRenderer( TableCellRenderer renderer)`	Definiert einen Renderer zur Darstellung der Zellendaten dieser Spalte
`void setPreferedWidth(int prefwidth)`	Definiert die bevorzugte Spaltenbreite

ListSelectionModel

- ListSelectionModel

`void clearSelection()`	Entfernt etwaige Selektionen
`int getMaxSelectionIndex()`	Gibt den Index der letzten selektierten Zeile zurück; `-1` wenn keine Zeile selektiert ist
`int getMinSelectionIndex()`	Gibt den Index der ersten selektierten Zeile zurück; `-1` wenn keine Zeile selektiert ist
`void setSelectionInterval(` `  int indexfrom, int indexto)`	Ändert die Selektion auf die Zeilen von `indexfrom` bis einschließlich `indexto`
`void setSelectionMode()`	Setzt den Selektionsmodus: `MULTIPLE_INTERVAL_SELECTION`, `SINGLE_INTERVAL_SELECTION`, `SINGLE_SELECTION`

RowSorter und TableCellRenderer

- RowSorter

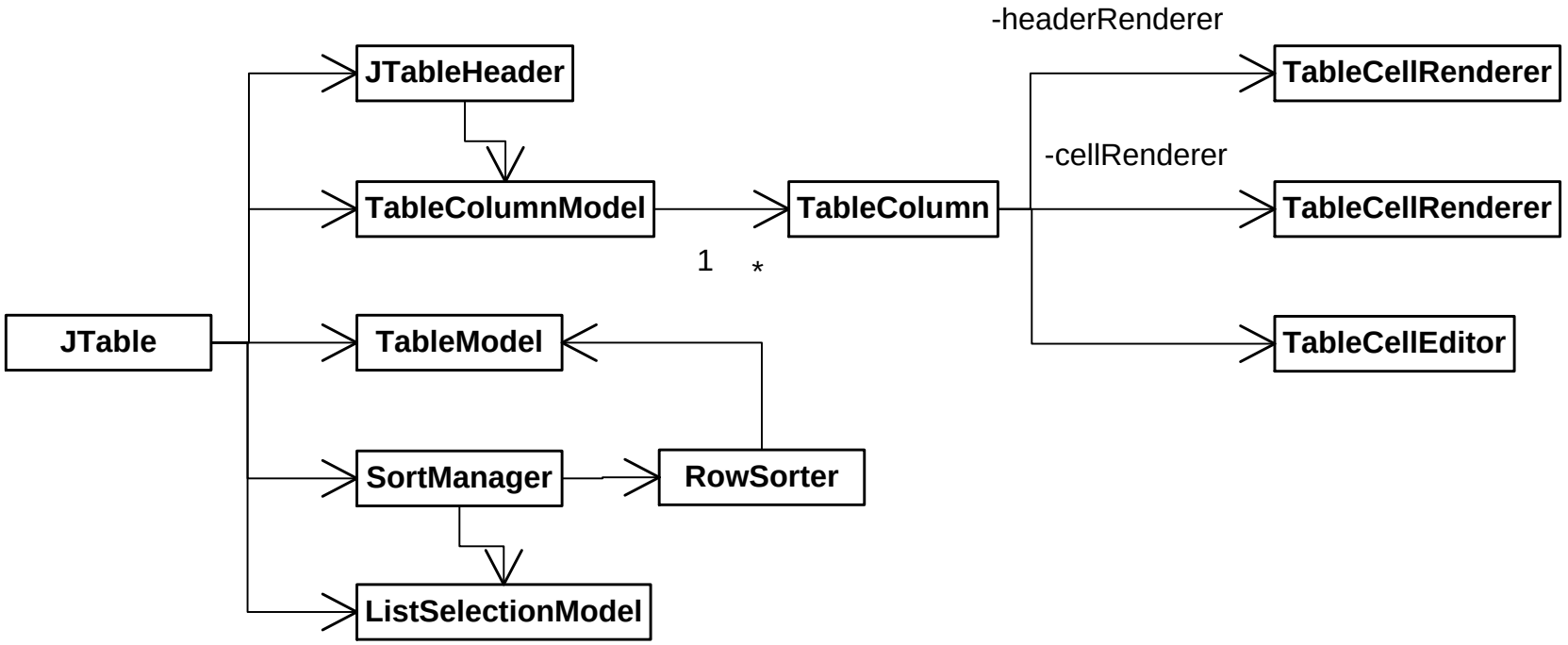
`void toggleSortOrder(int column)`	Sortiert die Tabelle nach der gegebenen Spalte; Falls nach dieser bereits sortiert ist, wird die Sortierung umgekehrt

- TableCellRenderer

`Component getTableCellRendererComponent(`
`  JTable table, Object value,`
`  boolean isselected, boolean hasfocus,`
`  int row, int column)`
	Gibt die UI-Komponente, die die Anzeige der Zelle realisiert (z.B. ein `JLabel`)

`JTable` im Überblick

- Folgende Abbildung zeigt nur die wesentlichen Komponenten von `JTable` und auch nur die wesentlichen Beziehungen untereinander:

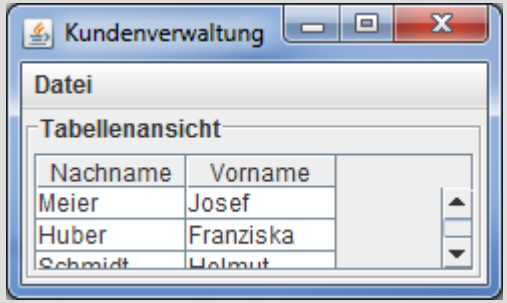


- Weitere Details zu `JTable` siehe: „How to Use Tables“: https://docs.oracle.com/javase/tutorial/uiswing/components/table.html

KundenVerwaltung v5

Aspekt:
Tabellenlayout
(bzw. Decorator)

```
public class KundenVerwaltung extends JFrame {

   private ListSelectionModel kundeSelectionModel;
   private KundeTableModel kundeTableModel;

   public KundenVerwaltung(Kunde[] kunden) {
      …
      kundeTableModel = new KundeTableModel(kunden);
      JTable table = new JTable(kundeTableModel);
      …
      table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
      JScrollPane scrollpane = new JScrollPane(table);
      JPanel titlepane = new JPanel();
      titlepane.setBorder(BorderFactory.createTitledBorder(
         BorderFactory.createEtchedBorder(), "Tabellenansicht"));
      titlepane.setLayout(new BorderLayout());
      titlepane.add(scrollpane);

      add(titlepane);
      …
   }
   …
}
```

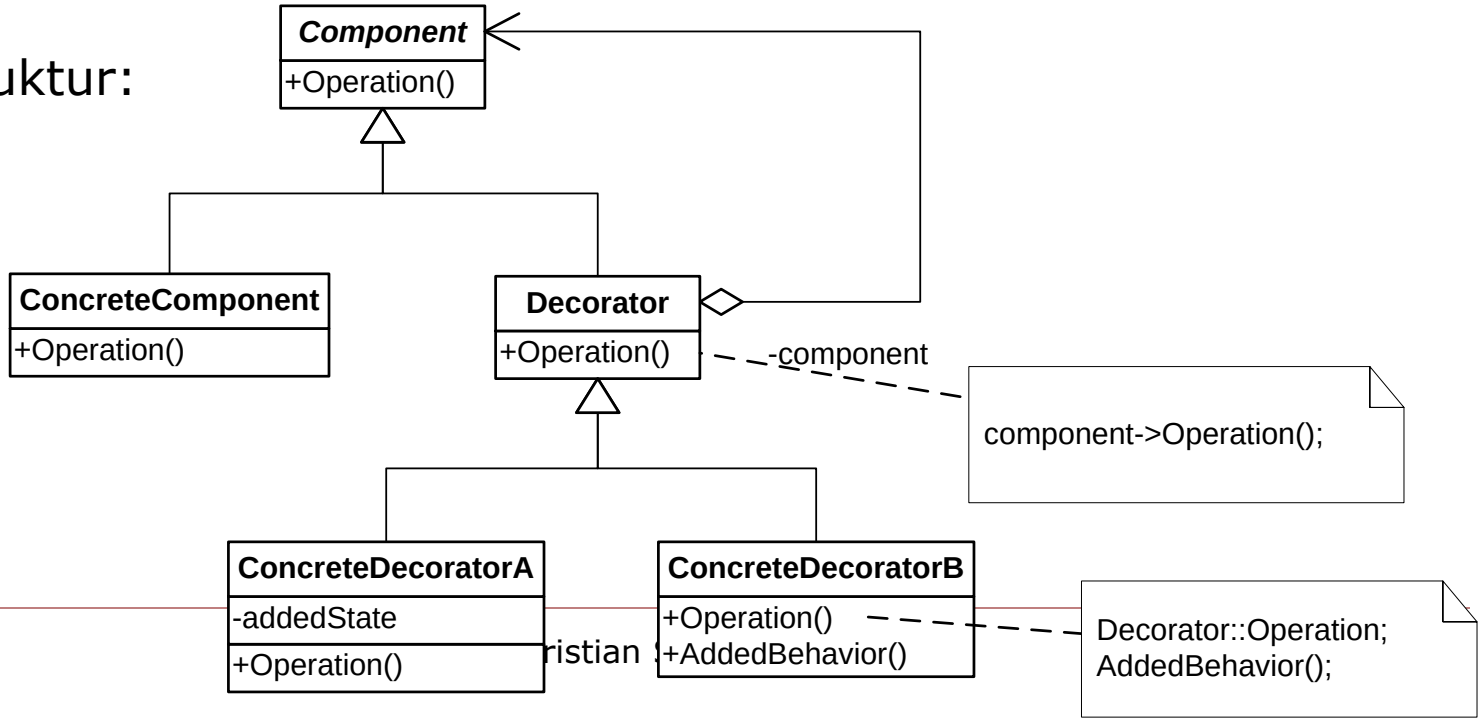


[bs] Decorator-Pattern

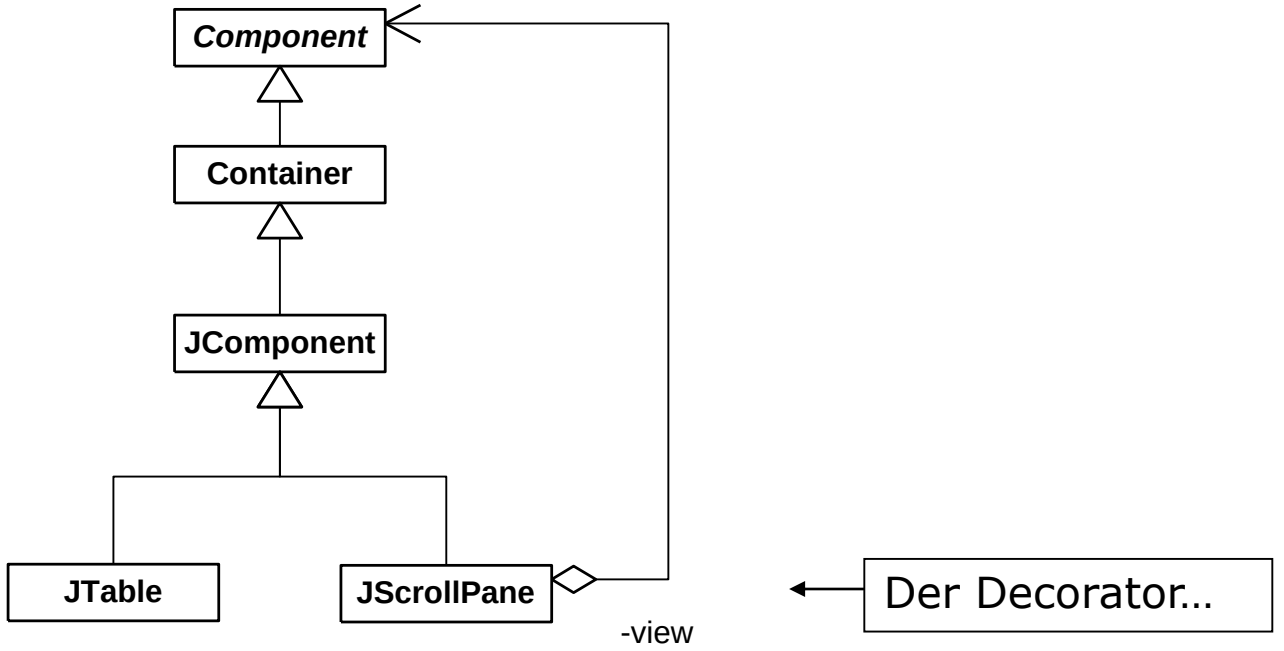
- Zweck:

> Fügt einem Objekt dynamisch zusätzliche Verantwortlichkeiten hinzu. Decorators bieten eine flexible Alternative zur Vererbung zum Zweck der Erweiterung der Funktionalität.

- Struktur:

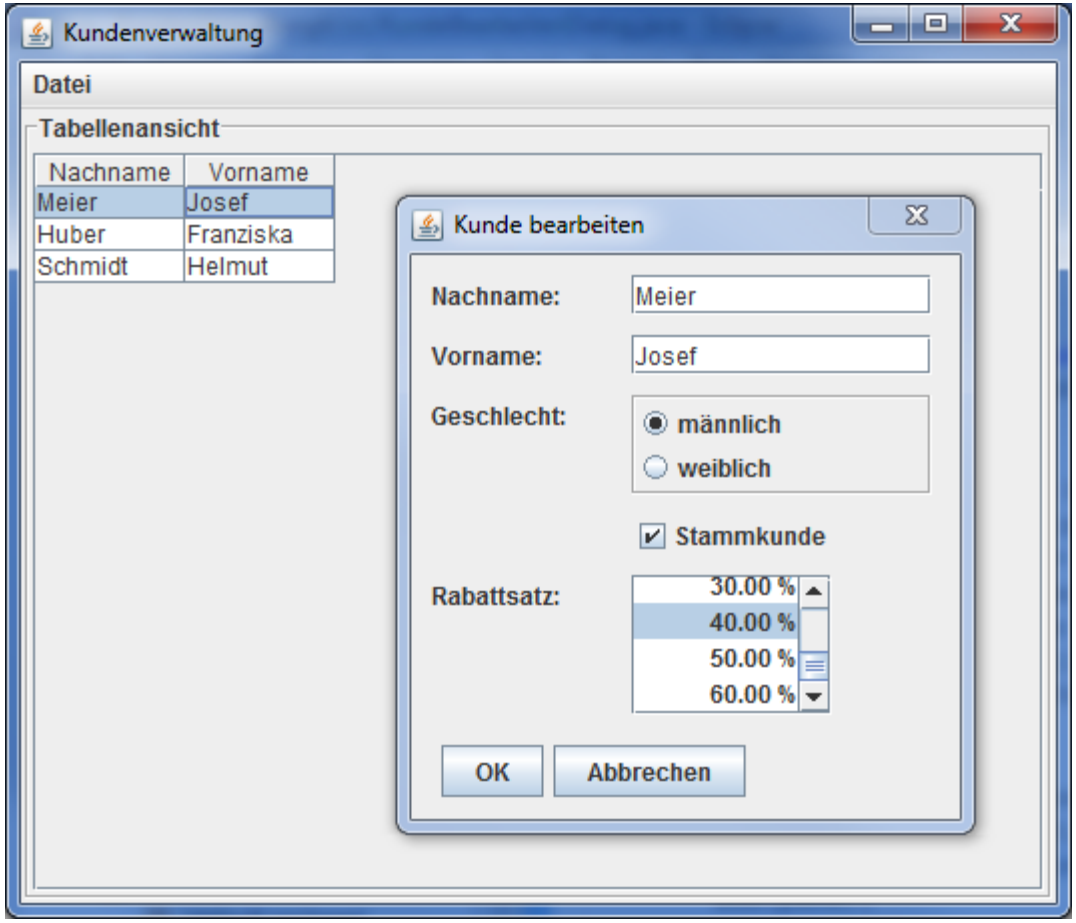


[bs] Decorator-Pattern



- Ein detaillierteres Beispiel zum Decorator-Pattern folgt beim Thema *Dateiverarbeitung* im Kapitel *Persistenz*

KundenVerwaltung v6



KundenVerwaltung v6

```
enum Geschlecht { maennlich, weiblich }

public class Kunde {
   private String nachname;
   private String vorname;
   private Geschlecht geschlecht;
   private boolean isStammkunde;
   private double rabattSatz;

   public Kunde(String nachname, String vorname, Geschlecht geschlecht,
                boolean stammkunde, double rabattsatz) { … }
   …
   public Geschlecht getGeschlecht() {
      return geschlecht;
   }

   public void setGeschlecht(Geschlecht geschlecht) {
      this.geschlecht = geschlecht;
   } …
}
```

Vorgriff Enumeration:
Definiert einen Typ für eine Menge von Konstanten.
(Details: siehe Kapitel 5)

JCheckBox (Checkbox)

- Wichtige Methoden:

`JCheckBox(String text, boolean selected)`	text: Beschriftung selected: Wert der Checkbox
`JCheckBox(String text)`	
`void setSelected (boolean selected)`	Setzt den Wert der Checkbox
`boolean isSelected()`	Gibt den Wert der Checkbox zurück
`void addItemListener(ItemListener l)`	Fügt einen Handler hinzu, um Zustandsänderungen zu überwachen.

- Anwendung:

```
chkStammkunde = new JCheckBox("Stammkunde", kunde.isStammkunde());
…
kunde.setStammkunde(chkStammkunde.isSelected());
```

`JRadioButton` (Radiobutton)

- Wichtige Methoden:

`JRadioButton(String text, boolean selected)`	`text`: Beschriftung `selected`: Wert des Radiobuttons
`JRadioButton(String text)`	
`void setModel(ButtonModel model)`	Setzt ein Model (Wenn kein `ButtonModel` explizit gesetzt wird, dann verwendet `JRadioButton` ein `JToggleButton.ToggleButtonModel`)
`void setSelected ( boolean selected)`	Setzt den Wert des Radiobuttons
`boolean isSelected()`	Gibt den Wert des Radiobuttons zurück
`void addItemListener( ItemListener l)`	Fügt einen Handler hinzu, um Zustandsänderungen zu überwachen.

ButtonGroup

- Gruppiert eine Menge von Buttons und stellt sicher, dass maximal einer dieser Buttons selektiert wird.
- Üblicherweise gruppiert eine `ButtonGroup` eine Menge von `JRadioButton`**s**.
- Wichtige Methoden:

`void add(AbstractButton b)`	Fügt der `ButtonGroup` einen Button hinzu
`ButtonModel getSelection()`	Gibt das Model des selektierten Buttons zurück
`boolean isSelected(ButtonModel m)`	Prüft, ob ein Button eines bestimmten Models selektiert ist
`void setSelected (ButtonModel m, boolean selected)`	Setzt die Selektierung eines Buttons unter Angabe des Models

KundenVerwaltung v6

- Problem: Einbinden der Radiobuttons für das Geschlecht
- Jeder Radiobutton stellt ein Geschlecht dar, das abgefragt werden muss. Das Standardmodel speichert aber i.W. nur, ob ein Button selektiert wurde oder nicht.
- Lösung: Ein eigenes Model definieren, über das das Geschlecht abfragbar ist.

GeschlechtButtonModel

```
public class GeschlechtButtonModel
                  extends JToggleButton.ToggleButtonModel {

   private Geschlecht value;

   public GeschlechtButtonModel(Geschlecht value) {
      this.value = value;
   }

   public Geschlecht getValue() {
      return value;
   }
}
```

Defaultmodel für
JRadioButton

KundenVerwaltung v6

- Problem: Setzen und Auslesen des Geschlechts über die `ButtonGroup`
- Eine `ButtonGroup` erlaubt das Selektieren eines Buttons anhand dessen Model (`setSelected()`). In der Anwendung ist die Datenbasis aber vom Typ `Geschlecht`.
- Lösung: Ein eigenes Model definieren, über das das Geschlecht gesetzt und ausgelesen werden kann

GeschlechtButtonGroup

```
public class GeschlechtButtonGroup extends ButtonGroup {

   public Geschlecht getValue() {
      GeschlechtButtonModel model =
         (GeschlechtButtonModel)getSelection();
      return model.getValue();
   }

   public void setValue(Geschlecht g) {
      for (AbstractButton b : buttons)  {
         GeschlechtButtonModel model =
            (GeschlechtButtonModel)b.getModel();
         if (model.getValue() == g) b.setSelected(true);
      }
   }
}
```

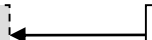
protected
Attribut;
Alternative:
getElements()

Die Lösung ist weder typsicher, noch bietet sie die beste Performance (siehe Schleife). Verbesserungsvorschläge erwünscht!

Anw. in KundeBearbeitenDialog

```
JPanel geschlechtpanel = new JPanel();
geschlechtpanel.setLayout(new BorderLayout());
geschlechtpanel.setBorder(BorderFactory.createEtchedBorder());
geschlechtpanel.setBounds(110, 70, 150, 50);
add(geschlechtpanel);

JRadioButton rbmaennlich = new JRadioButton("männlich");
rbmaennlich.setModel(new GeschlechtButtonModel(Geschlecht.maennlich));
geschlechtpanel.add(rbmaennlich, BorderLayout.NORTH);
JRadioButton rbweiblich = new JRadioButton("weiblich");
rbweiblich.setModel(new GeschlechtButtonModel(Geschlecht.weiblich));
geschlechtpanel.add(rbweiblich, BorderLayout.SOUTH);

bgGeschlecht = new GeschlechtButtonGroup();
bgGeschlecht.add(rbmaennlich);
bgGeschlecht.add(rbweiblich);
bgGeschlecht.setValue(kunde.getGeschlecht());
```

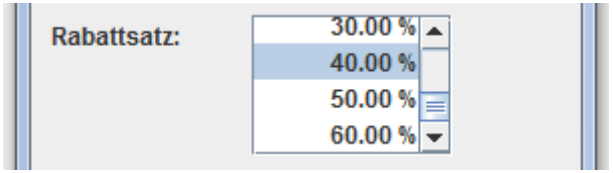
```
kunde.setGeschlecht(bgGeschlecht.getValue());
```



In MyOkHandler

`JList` (Listbox)

- So soll es aussehen:



- Wichtige Methoden:

`JList(ListModel model)`	Initialisiert ein `JList` mit einem `ListModel`
`ListModel getModel()`	Gibt das `ListModel` zurück, das den Listeninhalt verwaltet
`ListSelectionModel getSelectionModel()`	Gibt das `ListSelectionModel` zurück, das Informationen über selektierte Einträge verwaltet
`void setCellRenderer( ListCellRenderer lcr)`	Definiert einen Renderer zur Darstellung der Daten der Liste
`void setSelectedValue( Object value, boolean shouldscroll)`	Setzt einen Wert in der Liste, zu dem ggf. gescrollt werden kann
`Object getSelectedValue()`	Gibt den Wert der Liste zurück

Anw. in `KundeBearbeitenDialog`

```
ListModel model = new RabattListModel();
lstRabatt = new JList(model);

ListSelectionModel lsm = lstRabatt.getSelectionModel();
lsm.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

ListCellRenderer lcr = new RabattListCellRenderer();
lstRabatt.setCellRenderer(lcr);

JScrollPane scrollpane = new JScrollPane(lstRabatt);
scrollpane.setBounds(110, 160, 100, 70);

lstRabatt.setSelectedValue(kunde.getRabattSatz(), true);

add(scrollpane);
```

Von `Object` nach `double`; Details siehe Kapitel 5

In `MyOkHandler`

```
kunde.setRabattSatz((double)lstRabatt.getSelectedValue());
```

ListModel

- `AbstractListModel` bietet eine Standardimplementierung des Interfaces `ListModel`, bis auf die Methoden `getElementAt()` und `getSize()`

- Anwendung:

```
public class RabattListModel
                    extends AbstractListModel {
   public Object getElementAt(int index) {
      switch (index) {
         case 1: return 0.05;
         case 2: return 0.10;
         case 3: return 0.15;
         case 4: return 0.20;
         case 5: return 0.25;
         case 6: return 0.30;
         case 7: return 0.40;
         case 8: return 0.50;
         case 9: return 0.60;
         default: return 0.0;
      }
   }
   public int getSize() { return 10; }
}
```

ListCellRenderer

- Anwendung:

```
public class RabattListCellRenderer
                            extends DefaultListCellRenderer.UIResource {

   private DecimalFormat format;

   public RabattListCellRenderer() {
      format = new DecimalFormat("##0,00 %");
      setHorizontalAlignment(SwingConstants.RIGHT);
   }

   public Component getListCellRendererComponent(JList list,
      Object value, int index, boolean isSelected, boolean cellHasFocus) {

      super.getListCellRendererComponent(list, value, index,
                                         isSelected, cellHasFocus);
      double d = (double)value * 100.0;
      String s = format.format(d);
      setText(s);
      return this;
   }
}
```

Messageboxen

- Messageboxen sind Standarddialoge, um Benutzern Statusmitteilungen bereitzustellen oder einfache Abfragen zu tätigen.
- In Swing ist hierfür die Klasse `JOptionPane` hilfreich.

`JOptionPane` im Überblick

- `JOptionPane` stellt statische Methoden nach dem Muster `showXxxDialog()` zur Bereitstellung der Dialoge zur Verfügung.
- Die entsprechenden Dialoge sind modal.

`JOptionPane` im Überblick

- Dialogtypen:

`showMessageDialog()`	Gibt eine Statusmeldung aus
`showConfirmDialog()`	Erfragt eine mögliche Bestätigung
`showInputDialog()`	Erlaubt einfache Benutzereingaben
`showOptionDialog()`	Vereinigt die obigen drei Modelle (höchste Flexibilität); Wird i.d.R. für eine freie Auswahl von Buttons verwendet

- Allgemeines Layout:

<table>
<tr><td colspan="2">Titel</td></tr>
<tr><td rowspan="2">Icon</td><td>Mitteilungstext</td></tr>
<tr><td>Benutzereingabe</td></tr>
<tr><td colspan="2">Buttonleiste</td></tr>
</table>

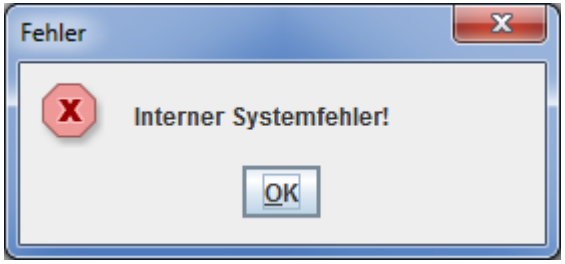
MessageDialog

- Varianten:

`showMessageDialog(Component parentComponent, Object message)`
`showMessageDialog(Component parentComponent, Object message, String title, int messageType)`
`showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)`

- Beispiel:

```
JOptionPane.showMessageDialog(
    this,
    "Interner Systemfehler!",
    "Fehler",
    JOptionPane.ERROR_MESSAGE);
```



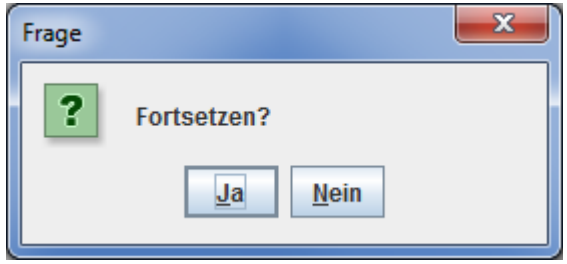
ConfirmDialog

- Varianten:

`showConfirmDialog(Component parentComponent, Object message, String title, int optionType)`
`showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType)`
`showMessageDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)`

- Beispiel:

```
int result = JOptionPane.showConfirmDialog(
   this, "Fortsetzen?", "Frage",
   JOptionPane.YES_NO_OPTION);

if (result == JOptionPane.YES_OPTION)
   System.out.println("Ja");
else
   System.out.println("Nein");
```



InputDialog

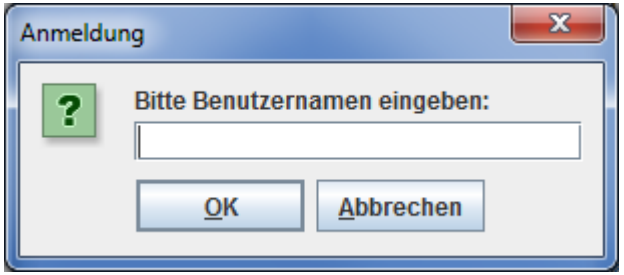
- Varianten:

```
showInputDialog(Component parentComponent, Object message,
                String title, int messageType)
```

```
showInputDialog(Component parentComponent, Object message,
                String title, int messageType, Icon icon,
                Object[] selectionValues,
                Object initialSelectionValue)
```

- Beispiel:

```
String eingabe =
   JOptionPane.showInputDialog(
      this,
      "Bitte Benutzernamen eingeben:",
      "Anmeldung",
      JOptionPane.QUESTION_MESSAGE);
System.out.println(eingabe);
```

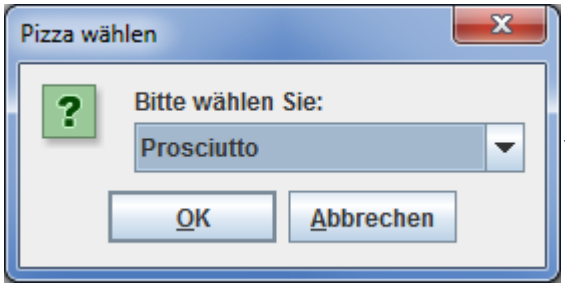


InputDialog

- Beispiel 2:

Beliebige Objekte möglich; Anzeige bestimmt `toString()`

```
Object[] pizzas = {"Margherita", "Prosciutto", "Salami" };
Object selected = JOptionPane.showInputDialog(this,
                        "Bitte wählen Sie:", "Pizza wählen",
                        JOptionPane.QUESTION_MESSAGE, null,
                        pizzas, pizzas[1]);
System.out.println(selected);
```



Vorauswahl

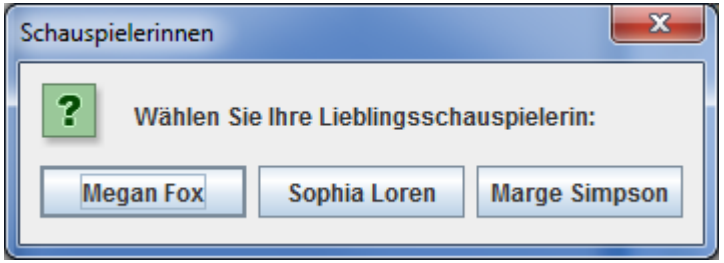
OptionDialog

- Varianten:

```
showOptionDialog(Component parentComponent, Object message,
                 String title, int optionType, int messageType,
                 Icon icon, Object[] options, Object initialValue)
```

- Beispiel:

```
Object[] options = { "Megan Fox", "Sophia Loren", "Marge Simpson" };
int result = JOptionPane.showOptionDialog(this,
   "Wählen Sie Ihre Lieblingsschauspielerin:", "Schauspielerinnen",
   JOptionPane.DEFAULT_OPTION, JOptionPane.QUESTION_MESSAGE,
   null, options, options[0]);
System.out.println(result);
```



Überblick Methodenparameter

- **`parentComponent`**
  Typischerweise der übergeordnete `JFrame`; kann auch `null` sein (Achtung: `this` in den Beispielen funktioniert nur, wenn sich der Aufruf unmittelbar in der übergeordneten Komponente befindet!)

- **`messageType`**
  Hat Einfluss auf das Layout; definiert meist ein Standardicon, Optionen:

`ERROR_MESSAGE`
`INFORMATION_MESSAGE`
`WARNING_MESSAGE`
`QUESTION_MESSAGE`
`PLAIN_MESSAGE`

Überblick Methodenparameter

- **`optionType`**
  Definiert eine Menge von Standardbuttons für den Dialog. Darauf ist man nicht beschränkt: Mit `options` können eigene Auswahloptionen definiert werden

`DEFAULT_OPTION`
`YES_NO_OPTION`
`YES_NO_CANCEL_OPTION`
`OK_CANCEL_OPTION`

- Für `showXxxDialog()`-Methoden, die einen `int` zurückgeben und eine Standardauswahl verwenden (insb. `showConfirmDialog()`) sind folgende Konstanten hilfreich:

`YES_OPTION`
`NO_OPTION`
`CANCEL_OPTION`
`OK_OPTION`
`CLOSED_OPTION`