Wiederholung:

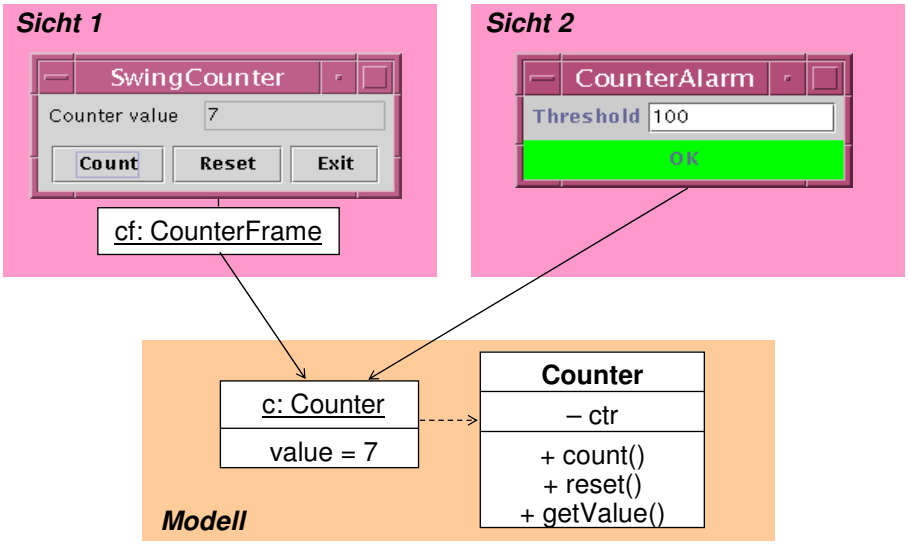
Objektorientierte Oberflächen-Programmierung mit Java und Swing

Heinrich Hußmann
Ludwig-Maximilians-Universität München
Sommersemester 2003

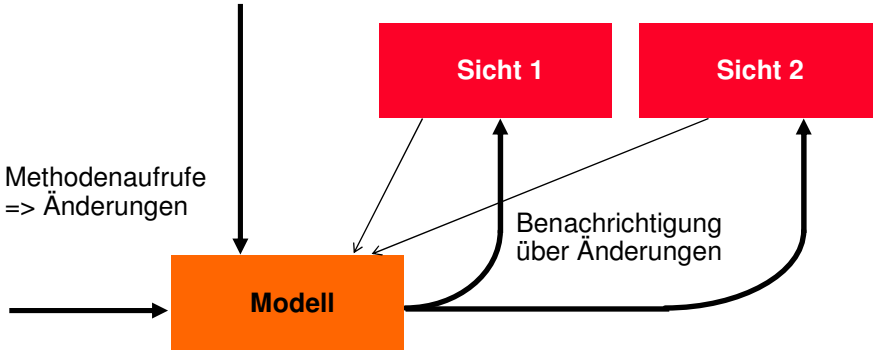
Objektorientierte Programmierung

- Geheimnisprinzip:
  - Jedes Objekt tut eine Sache und die richtig.
  - Objekte kommunizieren nur über genau definierte Schnittstellen.
- Lokalitätsprinzip, Hohe Kohäsion:
  - Jede Information wird lokal in dem Objekt gehalten, das seine Verarbeitungsmethoden bereitstellt.
- Substitutionsprinzip:
  - Klassen erben von anderen Klassen nur in der Weise, daß ein Objekt der Unterklasse überall da eingesetzt werden kann, wo ein Objekt der Oberklasse erwartet wird.
- Kollaborationsprinzip:
  - Es gibt keine langen Programmsequenzen; alle Aufgaben werden im Zusammenwirken von Objekten erledigt.
- Niedrige Kopplung:
  - Der Austausch von Programmteilen ist durch geeignete Mechanismen zu erleichtern.

Sichten: Motivierendes Beispiel (2)



Modell und Sicht



Beispiele: Verschiedene Dokumentenansichten, Statusanzeigen, Verfügbarkeit von Menüpunkten

Frage: *Wie hält man das Modell unabhängig von den einzelnen Sichten darauf ?*

Muster "Observer"

Ein Zähler (Beispiel fachliches Modell)

```
class Counter                          {
  private int ctr = 0;
  public void count () {
      ctr++;


  }
  public void reset () {
      ctr = 0;


  }
  public int getValue () {
      return ctr;
  }
}
```


Beobachtbares Modell *(Model)*

```
class Counter extends Observable {
  private int ctr = 0;
  public void count () {
      ctr++;
      setChanged();
      notifyObservers();
  }
  public void reset () {
      ctr = 0;
      setChanged();
      notifyObservers();
  }
  public int getValue () {
      return ctr;
  }
}
```

- Das fachliche Modell enthält keinerlei Bezug auf die Benutzungsoberfläche !

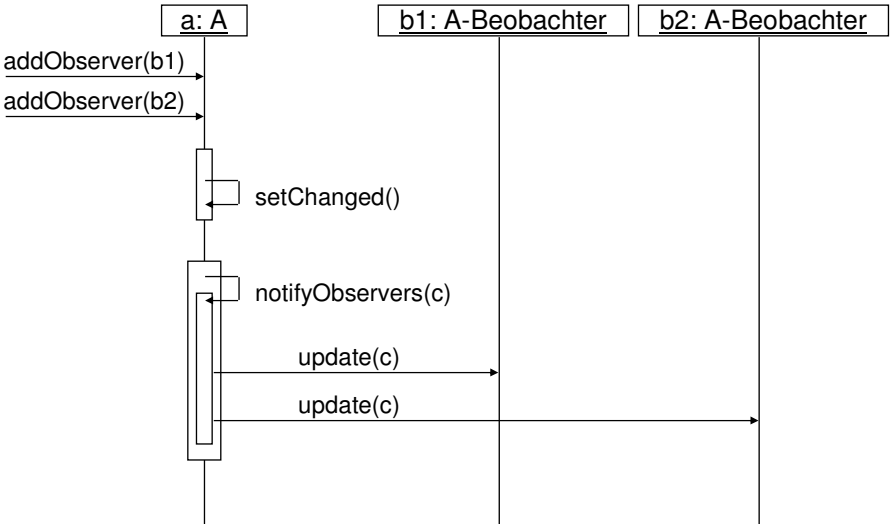
java.util.Observable, java.util.Observer

```
public class Observable {
  public void addObserver (Observer o);
  public void deleteObserver (Observer o);

  protected void setChanged();
  public void notifyObservers ();
  public void notifyObservers (Object arg);
}

public interface Observer {
  public void update (Observable o, Object arg);
}
```

Argumente für notifyObservers():

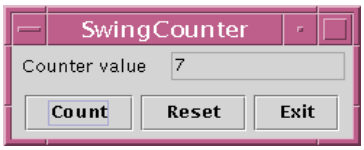
- meist nur Art der Änderung, nicht gesamte Zustandsinformation
- Beobachter können normale Methodenaufrufe nutzen, um sich näher zu informieren.

Beispielablauf



Graphische Benutzungsoberflächen

- 1980: Smalltalk-80-Oberfläche (Xerox)
- 1983/84: Lisa/Macintosh-Oberfläche (Apple)
- 1988: NextStep (Next)
- 1989: OpenLook (Sun)
- 1989: Motif (Open Software Foundation)
- 1987/91: OS/2 Presentation Manager (IBM)
- 1990: Windows 3.0 (Microsoft)
- 1995-2001: Windows 95/NT/98/2000/ME/XP (Microsoft)
- 1995: Java AWT (SunSoft)
- 1997: Swing Components for Java (SunSoft)



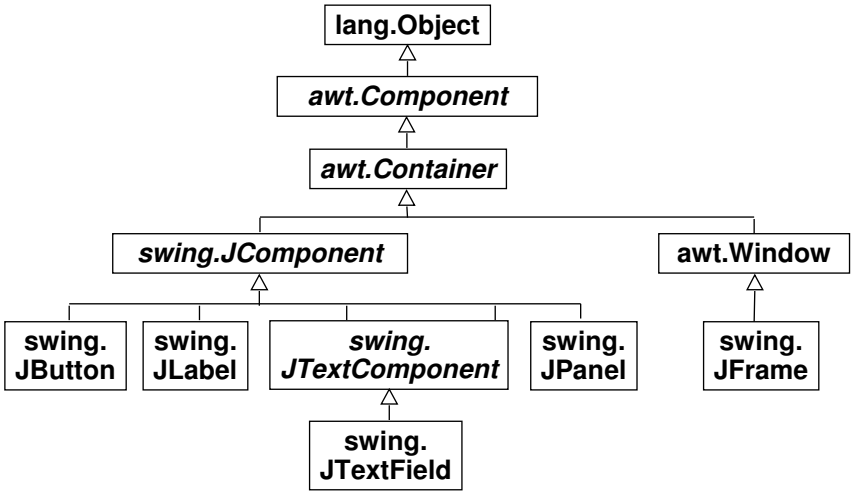
Bibliotheken von AWT und Swing

- Wichtigste AWT-Pakete:
  - **java.awt**: u.a. Grafik, Oberflächenkomponenten, Layout-Manager
  - **java.awt.event**: Ereignisbehandlung
  - Andere Pakete für weitere Spezialzwecke
- Wichtigstes Swing-Paket:
  - **javax.swing**: Oberflächenkomponenten
  - Andere Pakete für Spezialzwecke
- Viele AWT-Klassen werden auch in Swing verwendet!
- Standard-Vorspann:

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

- (Naiver) Unterschied zwischen AWT- und Swing-Komponenten:
  - AWT: Button, Frame, Menu, ...
  - Swing: JButton, JFrame, JMenu, ...

AWT/Swing-Klassenhierarchie (Ausschnitt)



- Dies ist nur ein sehr kleiner Ausschnitt!
- Präfixe "java." und "javax." hier weggelassen.

Component, Container, Window, Frame, Panel

- ***awt.Component*** (abstrakt):
  - Oberklasse aller Bestandteile der Oberfläche

```
public void setSize (int width, int height);
public void setVisible (boolean b);
```

- ***awt.Container*** (abstrakt):
  - Oberklasse aller Komponenten, die andere Komponenten enthalten

```
public void add (Component comp);
public void setLayout (LayoutManager mgr);
```

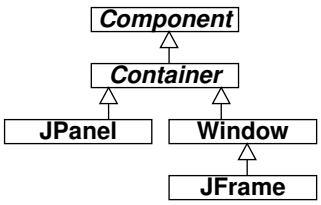
- **awt.Window**
  - Fenster ohne Rahmen oder Menüs

```
public void pack (); //Größe anpassen
```

- **swing.JFrame**
  - Größenveränderbares Fenster mit Titel

```
public void setTitle (String title);
```

- **swing.JPanel**
  - Zusammenfassung von Swing-Komponenten



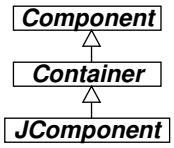
JComponent

- Oberklasse aller in der Swing-Bibliothek neu implementierten, verbesserten Oberflächenkomponenten. Eigenschaften u.a.:
  - Einstellbares "Look-and-Feel" (sh. später)
  - Komponenten kombinierbar und erweiterbar
  - Rahmen für Komponenten

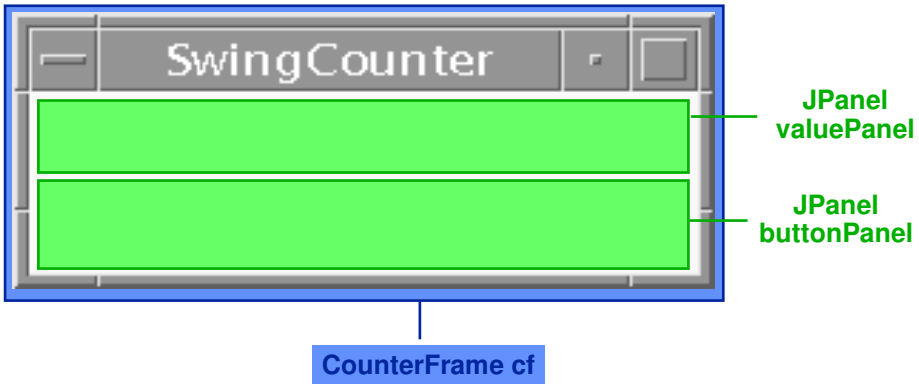
    ```
    void setBorder (Border border);
    ```

    (Border-Objekte mit `BorderFactory` erzeugbar)
  - ToolTips -- Kurzbeschreibungen, die auftauchen, wenn der Cursor über der Komponente liegt

    ```
    void setToolTipText (String text);
    ```
  - Automatisches Scrolling
- Beispiele für weitere Unterklassen von JComponent:
  - JList: Auswahlliste
  - JComboBox: "Drop-Down"-Auswahlliste mit Texteingabemöglichkeit
  - JPopUpMenu: "Pop-Up"-Menü
  - JFileChooser: Dateiauswahl



Zähler-Beispiel: Grobentwurf der Oberfläche

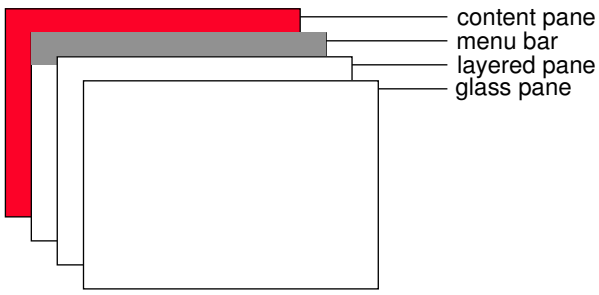


Die Sicht *(View):* Gliederung, 1. Versuch

```
class CounterFrame extends JFrame {
  JPanel valuePanel = new JPanel();

  JPanel buttonPanel = new JPanel();



  public CounterFrame (Counter c) {
    setTitle("SwingCounter");


    ... valuePanel zu this hinzufügen


    ... buttonPanel zu this hinzufügen
    pack();
    setVisible(true);
  }
}
```


Hinzufügen von Komponenten zu JFrames

- Ein JFrame ist ein "Container", d.h. dient zur Aufnahme weiterer Elemente.
- Ein JFrame ist intern in verschiedene "Scheiben" *(panes)* organisiert. Die wichtigste ist die *content pane*.

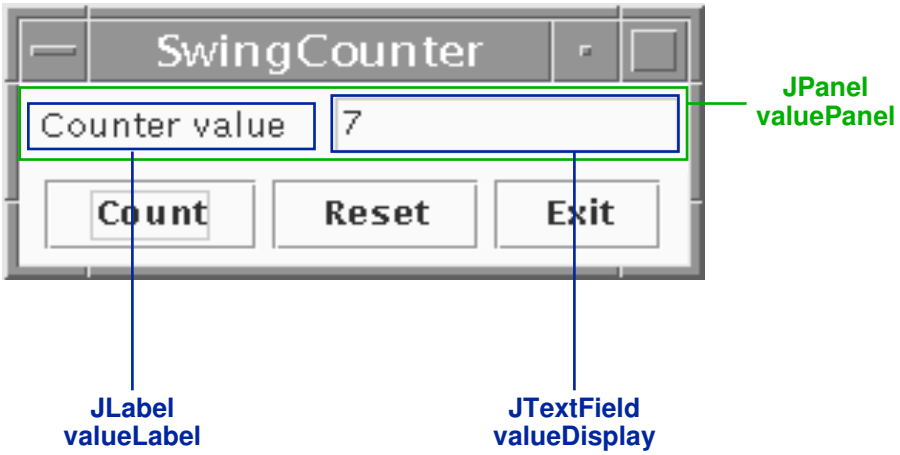


- In JFrame ist definiert:
  `Container getContentPane();`

Die Sicht *(View):* Gliederung, 2. Versuch

```
class CounterFrame extends JFrame {
  JPanel valuePanel = new JPanel();

  JPanel buttonPanel = new JPanel();



  public CounterFrame (Counter c) {
    setTitle("SwingCounter");


    getContentPane().add(valuePanel);


    getContentPane().add(buttonPanel);
    pack();
    setVisible(true);
  }
}
```


Zähler-Beispiel: Entwurf der Wertanzeige



TextComponent, TextField, Label, Button

- ***JTextComponent:***
  - Oberklasse von JTextField und JTextArea

```
public void setText (String t);
public String getText ();
public void setEditable (boolean b);
```

- **JTextField:**
  - Textfeld mit einer Zeile

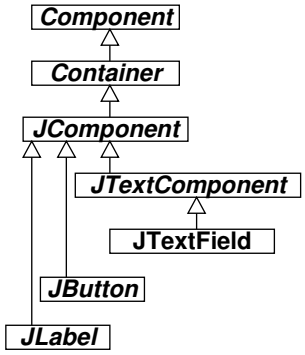
```
public JTextField (int length);
```

- **JLabel:**
  - Einzeiliger unveränderbarer Text

```
public JLabel (String text);
```

- **JButton:**
  - Druckknopf mit Textbeschriftung

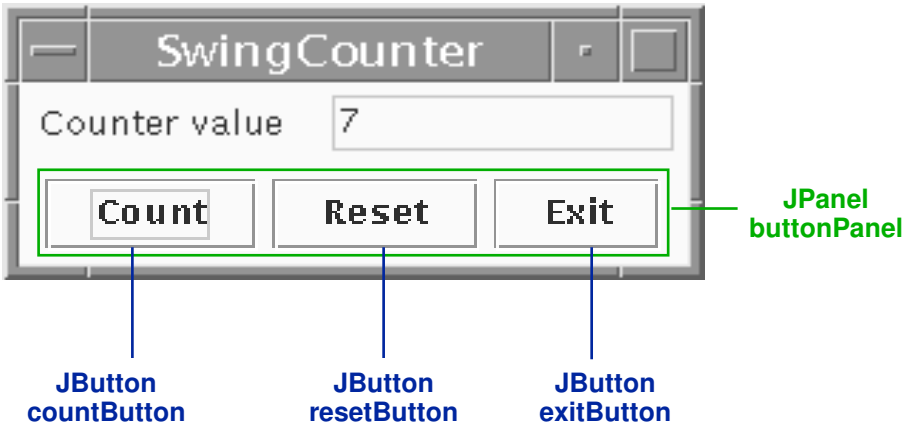
```
public JButton (String label);
```



Die Sicht *(View):* Elemente der Wertanzeige

```
class CounterFrame extends JFrame {
  JPanel valuePanel = new JPanel();
  JTextField valueDisplay = new JTextField(10);
  JPanel buttonPanel = new JPanel();



  public CounterFrame (Counter c) {
    setTitle("SwingCounter");
    valuePanel.add(new JLabel("Counter value"));
    valuePanel.add(valueDisplay);
    valueDisplay.setEditable(false);
    getContentPane().add(valuePanel);



    getContentPane().add(buttonPanel);
    pack();
    setVisible(true);
  }
}
```

Zähler-Beispiel: Entwurf der Bedienelemente



Die Sicht *(View):* Bedienelemente

```
class CounterFrame extends JFrame {
  JPanel valuePanel = new JPanel();
  JTextField valueDisplay = new JTextField(10);
  JPanel buttonPanel = new JPanel();
  JButton countButton = new JButton("Count");
  JButton resetButton = new JButton("Reset");
  JButton exitButton = new JButton("Exit");

  public CounterFrame (Counter c) {
    setTitle("SwingCounter");
    valuePanel.add(new JLabel("Counter value"));
    valuePanel.add(valueDisplay);
    valueDisplay.setEditable(false);
    getContentPane().add(valuePanel);
    buttonPanel.add(countButton);
    buttonPanel.add(resetButton);
    buttonPanel.add(exitButton);
    getContentPane().add(buttonPanel);
    pack();
    setVisible(true);
  }
}
```

Layout-Manager

- **Definition** Ein *Layout-Manager* ist ein Objekt, das Methoden bereitstellt, um die graphische Repräsentation verschiedener Objekte innerhalb eines Container-Objektes anzuordnen.
- Formal ist LayoutManager ein Interface, für das viele Implementierungen möglich sind.
- In Java definierte Layout-Manager (Auswahl):
  - FlowLayout (java.awt.FlowLayout)
  - BorderLayout (java.awt.BorderLayout)
  - GridLayout (java.awt.GridLayout)

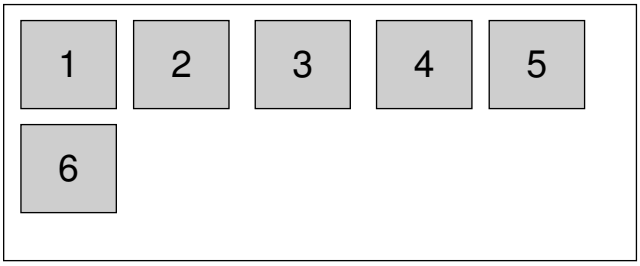
- In awt.Component:

  ```
  public void add (Component comp, Object constraints);
  ```

  erlaubt es, zusätzliche Information (z.B. Orientierung, Zeile/Spalte) an den Layout-Manager zu übergeben

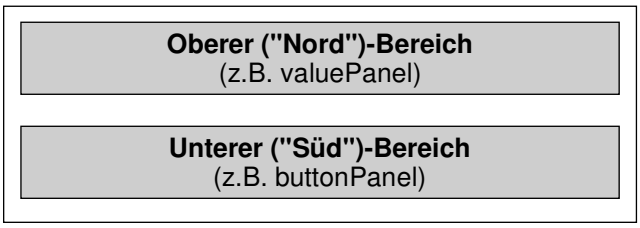
Flow-Layout

- Grundprinzip:
  - Anordnung analog Textfluß:
    von links nach rechts und von oben nach unten
- Default für Panels
  - z.B. in valuePanel und buttonPanel
    für Hinzufügen von Labels, Buttons etc.
- Parameter bei Konstruktor: Orientierung auf Zeile, Abstände
- Constraints bei **`add`**: keine



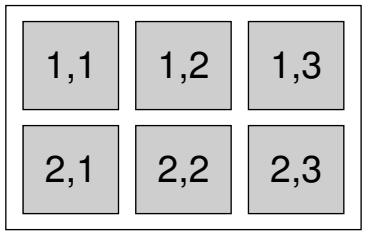
Border-Layout

- Grundprinzip:
  - Orientierung nach den Seiten (N, S, W, O) bzw. Mitte (center)
- Default für Window, Frame
  - z.B. in CounterFrame für Hinzufügen von valuePanel, buttonPanel
- Parameter bei Konstruktor: Keine
- Constraints bei **`add:`**
  - **`BorderLayout.NORTH, SOUTH, WEST, EAST, CENTER`**



Grid-Layout

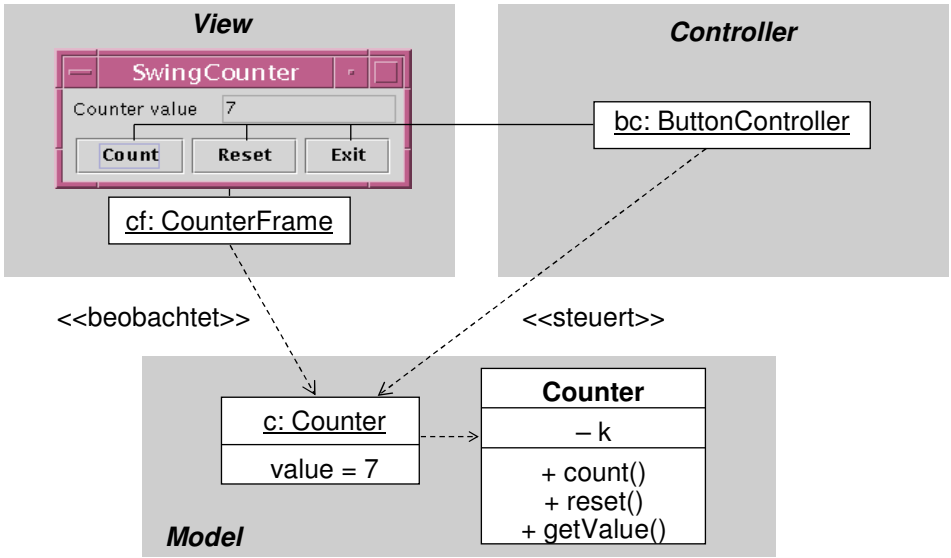
- Grundprinzip:
  - Anordnung nach Zeilen und Spalten
- Parameter bei Konstruktor:
  - Abstände, Anzahl Zeilen, Anzahl Spalten
- Constraints bei **`add:`**
  - Zeilen- und Spaltenindex als int-Zahlen



Die Sicht *(View):* Alle sichtbaren Elemente

```
class CounterFrame extends JFrame {
  JPanel valuePanel = new JPanel();
  JTextField valueDisplay = new JTextField(10);
  JPanel buttonPanel = new JPanel();
  JButton countButton = new JButton("Count");
  JButton resetButton = new JButton("Reset");
  JButton exitButton = new JButton("Exit");

  public CounterFrame (Counter c) {
    setTitle("SwingCounter");
    valuePanel.add(new JLabel("Counter value"));
    valuePanel.add(valueDisplay);
    valueDisplay.setEditable(false);
    getContentPane().add(valuePanel, BorderLayout.NORTH);
    buttonPanel.add(countButton);
    buttonPanel.add(resetButton);
    buttonPanel.add(exitButton);
    getContentPane().add(buttonPanel, BorderLayout.SOUTH);
    pack();
    setVisible(true);
  }
}
```


Model-View-Controller-Architektur



Zähler-Beispiel: Anbindung Model/View

```
class CounterFrame extends JFrame
   implements Observer {
  ...
  JTextField valueDisplay = new JTextField(10);
  ...

  public CounterFrame (Counter c) {
    ...
    valuePanel.add(valueDisplay);
    valueDisplay.setEditable(false);
    valueDisplay.setText(String.valueOf(c.getValue()));
    ...
    c.addObserver(this);
    pack();
    setVisible(true);
  }

  public void update (Observable o, Object arg) {
    Counter c = (Counter) o;
    valueDisplay.setText(String.valueOf(c.getValue()));
  }
}
```


java.awt.event.ActionEvent, ActionListener

```
public class ActionEvent extends AWTEvent {
  ...
  // Konstruktor wird vom System aufgerufen

  public Object getSource ()
  public String getActionCommand()
  ...
}

public interface ActionListener
    extends EventListener {
  public void actionPerformed (ActionEvent ev);
}
```

Die Steuerung *(Controller)*

```
class ButtonController implements ActionListener {

  Counter myCounter;

  public void actionPerformed (ActionEvent event) {
    String cmd = event.getActionCommand();
    if (cmd.equals("Count"))
      myCounter.count();
    if (cmd.equals("Reset"))
      myCounter.reset();
    if (cmd.equals("Exit"))
      System.exit(0);
  }

  public ButtonController (Counter c) {
    myCounter = c;
  }
}
```


Zähler-Beispiel: Anbindung des Controllers

```
class CounterFrame extends JFrame {
  ...
  JPanel buttonPanel = new JPanel();
  JButton countButton = new JButton("Count");
  JButton resetButton = new JButton("Reset");
  JButton exitButton = new JButton("Exit");

  public CounterFrame (Counter c) {
    ...
    ButtonController bc = new ButtonController(c);
    countButton.addActionListener(bc);
    buttonPanel.add(countButton);
    resetButton.addActionListener(bc);
    buttonPanel.add(resetButton);
    exitButton.addActionListener(bc);
    buttonPanel.add(exitButton);
    ...
  }
}
```

Alles zusammen: CounterFrame (1)

```
class CounterFrame extends JFrame implements Observer {

  JPanel valuePanel = new JPanel();
  JTextField valueDisplay = new JTextField(10);
  JPanel buttonPanel = new JPanel();
  JButton countButton = new JButton("Count");
  JButton resetButton = new JButton("Reset");
  JButton exitButton = new JButton("Exit");

  public CounterFrame (Counter c) {
      setTitle("SwingCounter");
      valuePanel.add(new JLabel("Counter value"));
      valuePanel.add(valueDisplay);
      valueDisplay.setEditable(false);
      valueDisplay.setText(String.valueOf(c.getValue()));
      getContentPane().add(valuePanel,BorderLayout.NORTH);
      ButtonController bc = new ButtonController(c);
      countButton.addActionListener(bc);
      buttonPanel.add(countButton);
      resetButton.addActionListener(bc);
      buttonPanel.add(resetButton);
      exitButton.addActionListener(bc);
      buttonPanel.add(exitButton);
      getContentPane().add(buttonPanel,BorderLayout.SOUTH);
```


Alles zusammen: CounterFrame (2)

```
      addWindowListener(new WindowCloser());
      c.addObserver(this);
      pack();
      setVisible(true);
   }

   public void update (Observable o, Object arg) {
      Counter c = (Counter) o;
      valueDisplay.setText(String.valueOf(c.getValue()));
   }
}

class ButtonController implements ActionListener {
  ... (wie oben) ...
}

class WindowCloser implements WindowListener {
  ... (wie in Kapitel 7.2) ...
}
```