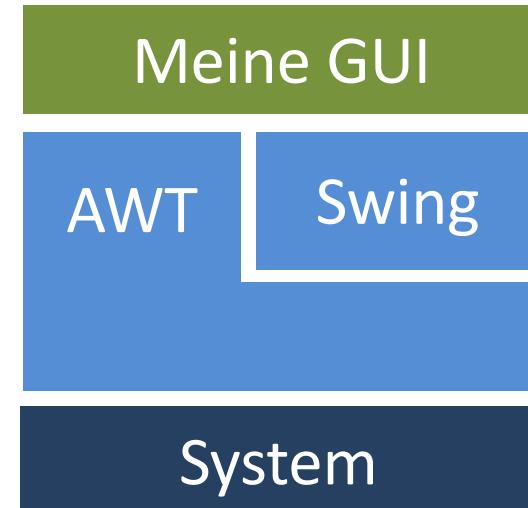


# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen

# Java AWT und Swing

- Java enthält zwei Pakete zur Programmierung grafischer Benutzeroberflächen (GUI, Graphical User Interface).
  - `java.awt`: Abstract Window Toolkit  
AWT ist das ältere, weniger komfortable Paket.
  - `javax.swing`:  
Swing ist das neuere, komfortablere Paket.
  - Swing baut auf AWT auf.
  - Swing-Komponenten sind in Java geschrieben und werden daher auch **leichtgewichtige Komponenten** genannt. Damit lassen sich Oberflächen plattformunabhängig gestalten.
  - AWT-Komponenten sind **schwergewichtige Komponenten** und haben ein plattformabhängiges Aussehen.



# Ein kleines Swing-Beispiel



```
import javax.swing.*;  
  
public class HelloWorldSwing {  
  
    public static void main(String[ ] args) {  
  
        JFrame frame = new JFrame("HelloWorldSwing");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JLabel label = new JLabel("Hello World," + " this is a small Swing Demo ");  
        frame.add(label);  
  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```

Swing-Paket importieren.

Hauptfenster `frame` mit Titelleiste erzeugen.

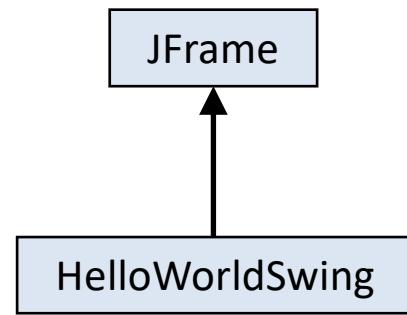
Verhalten bei Schließen des Fensters definieren.

Beschriftung erzeugen und in Fenster einfügen.

Fenstergröße passend machen und Fenster anzeigen.

# Benutzeroberfläche als eigene Frame-Klasse

- Die eigene Benutzeroberfläche kann durch Vererbung von der Hauptfensterklasse JFrame realisiert werden.



```
import javax.swing.*;  
  
public class HelloWorldSwing extends JFrame {  
  
    public HelloWorldSwing () {  
        this.setTitle("HelloWorldSwing");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        JLabel label = new JLabel("Hello World,"  
                               + " this is a small Swing Demo ");  
        this.add(label);  
        this.pack();  
        this.setVisible(true);  
    }  
  
    public static void main(String[ ] args) {  
        JFrame myApp = new HelloWorldSwing();  
    }  
}
```

# Ereignisverarbeitung (1)

- Üblicherweise interagiert der Benutzer mit der Benutzeroberfläche durch Mausbewegung, Mausklicks, Tastatureingaben, ...
- In Java gibt es das Event-Listener-Konzept.
- Ereignisquellen (z.B. Buttons) lösen **Events** (Ereignisse) aus.
- **EventListener** (Ereignis-Beobachter) verarbeiten dann die Events.



ActionEvent

```
void actionPerformed(ActionEvent e) {  
    // Verarbeitung von Event e  
    // ...  
}
```

Klicken auf Button  
löst ein **Event** aus.

**EventListener** wird aufgerufen  
und verarbeitet das Event.

# Ereignisverarbeitung (2)

- Registrierung:  
der EventListener muss bei der Ereignisquelle registriert werden:  
`button.addListener(listener)`
- Das Fenstersystem (Laufzeitumgebung) sorgt dann dafür, dass bei Auslösen des Events (Klicken auf Button) der entsprechende EventListener aufgerufen wird.



ActionEvent

```
void actionPerformed(ActionEvent e) {  
    // Verarbeitung von Event e  
    // ...  
}
```

# Swing-Beispiel mit Ereignisverarbeitung (1)



```
public class WuerfelApplikation  
extends JFrame  
implements ActionListener {  
  
    ...  
  
    public WuerfelApplikation() {  
        ...  
        JButton button = new JButton("Würfeln");  
        this.add(button);  
        button.addActionListener(this);  
        ...  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        ...  
    }  
    ...  
}
```

WuerfelApplikation implementiert das Interface ActionListener.  
Damit wird WuerfelApplikation zu einem ActionListener.

Button als Ereignisquelle

Registriere den ActionListener

Das Interface ActionListener verlangt die Implementierung dieser Methode.  
Hier wird festgelegt, was der Würfeln-Button bewirkt.

# Swing-Beispiel mit Ereignisverarbeitung (2)

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

public class WuerfelApplikation
extends JFrame
implements ActionListener {

    private JLabel wuerfel;

    public WuerfelApplikation() {
        this.setTitle("Würfel");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new FlowLayout());

        JButton button = new JButton("Würfeln");
        this.add(button);
        button.addActionListener(this);

        this.wuerfel = new JLabel("Würfelzahl: 6");
        this.add(wuerfel);

        this.pack();
        this.setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        int w = (int) (Math.random()*6 + 1);
        this.wuerfel.setText("Würfelzahl: " + w);
    }
}
```



Sobald Würfeln-Button gedrückt wird, wird eine zufällige Zahl zwischen 1 und 6 generiert und in der Beschriftung wuerfel gespeichert.

```
public void actionPerformed(ActionEvent e) {
    int w = (int) (Math.random()*6 + 1);
    this.wuerfel.setText("Würfelzahl: " + w);
}

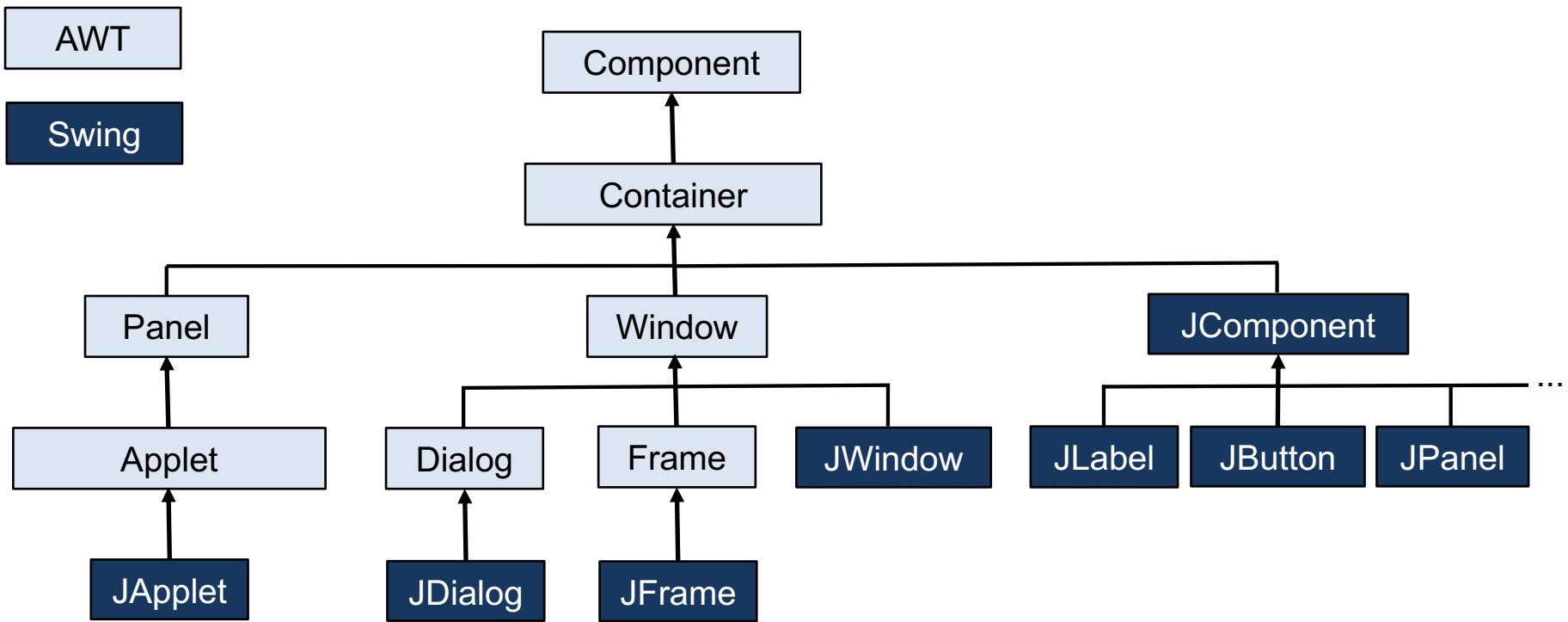
public static void main(String[] args) {
    JFrame meinApplikation
        = new WuerfelApplikation();
}
```

# Callback-Methoden und Threads

---

- In der bisherigen Programmierwelt:  
Programm besteht aus einem Hauptprogramm, das andere Funktionen aufruft, und die wiederum weitere Funktionen aufrufen.
- Bei Java-Programmen mit grafischen Benutzeroberflächen (oder ereignisorientierter Programmierung) kehrt sich die Aufrufstruktur um: es werden Listener-Methoden geschrieben, die vom System und nicht vom eigenen Programm aufgerufen werden.
- Funktionen, die nicht vom eigenen Programm, sondern vom System aufgerufen werden, werden auch **callback-Funktionen** genannt.
- Man beachte außerdem, dass in Java bei einer grafischen Benutzeroberflächen mehrere **Threads** parallel laufen:  
main thread, event-dispatching thread, ...

# Überblick über AWT und Swing



- **JFrame**, **JDialog**, **JApplet** und **JWindow** sind die sogenannten Top-Level-Container.
- Außer **JLabel** und **JButton** gibt es noch viele weitere Swing-Komponenten.
- Für Event-Handling ist das Paket [java.awt.event.\\*](#) wichtig.
- Zum Anordnen der Komponenten in einem Fenster sind das Interface [LayoutManager](#) mit seinen Klassen **FlowLayout**, **BorderLayout**, **GridLayout**, etc. wichtig.

# Hilfe zur Selbsthilfe

---

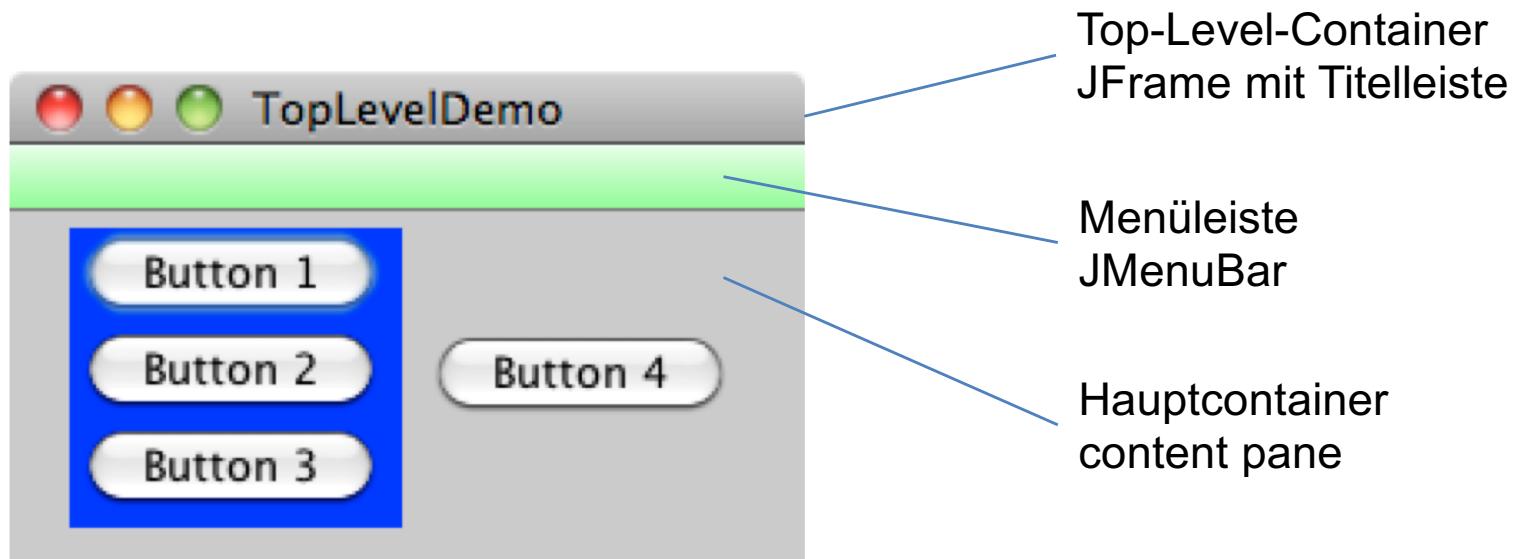
- Das Erstellen von graphischen Benutzeroberflächen erfordert sehr viel Übung und Erfahrung.
- Klein anfangen.
- Java-API ist sehr wichtig:  
<http://download.oracle.com/javase/8/docs/api/index.html>
- Sehr gut ist auch das Swing-Tutorial:  
<http://docs.oracle.com/javase/tutorial/uiswing/>
- Die Benutzung der Swing-Komponenten ist hier beschrieben:  
<http://docs.oracle.com/javase/tutorial/uiswing/components/index.html>
- Hier gibt es sehr viele Swing-Demos:  
<http://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html>  
(Java-Quellcode und JNLP-Dateien direkt zum Starten mit Java Web Start;  
viele Beispiele aus der Vorlesung sind hieraus entnommen)

# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen

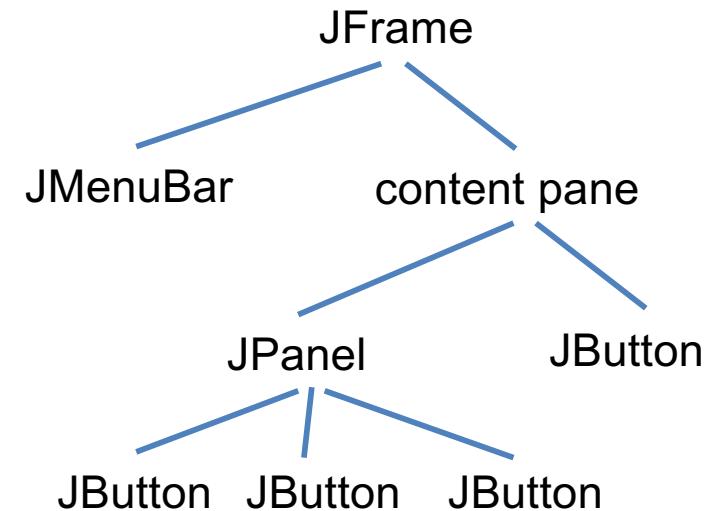
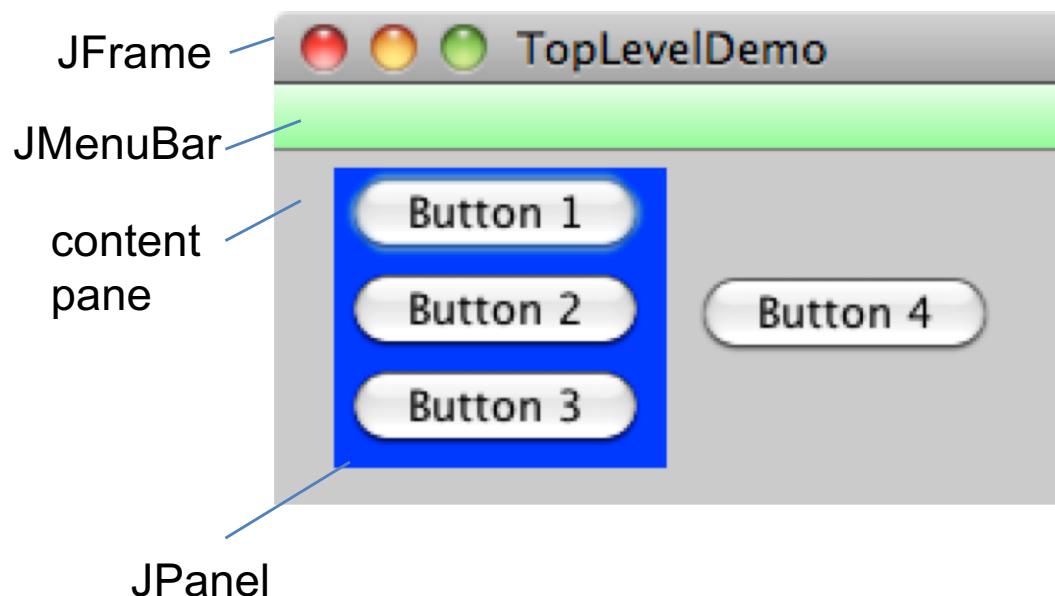
# Top-Level-Container und JFrame

- Alle Komponenten einer GUI werden in einem sogenannten Top-Level-Container eingebaut.
- **JFrame**, **JDialog**, **JApplet** und **JWindow** sind die Top-Level-Container.
- **JFrame** ist der wichtigste Top-Level-Container.
- Ein Top-Level-Container besteht aus dem **content pane** (pane = Scheibe), der die visuellen Komponenten der GUI enthält, evtl. einem Rahmen mit **Titelleiste** und einer optionalen **Menüleiste (JMenuBar)**.
- Ein Top-Level-Container hat bereits eine gewisse Grundfunktionalität wie Minimieren, Maximieren, Schließen und Verschieben.



# Container und Komponenten

- Ein Top-Level-Container kann weitere **Komponenten (Component)** enthalten, wie Schaltfächen (JButton), Beschriftungen (JLabel), Textfelder (JTextField), Panels (JPanel) etc.
- Da eine **Komponente auch ein Container** ist und daher weitere Komponenten enthalten kann, ergibt sich eine **hierarchische Komponenten-Struktur**.
- Eine Komponente wird mit **add** zu einem Container dazu gefügt:  
z.B. panel.add(button1)



# TopLevelDemo in Java (1)

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

public class ContainerDemo extends JFrame {

    public ContainerDemo() {
        this.setTitle("TopLevelDemo");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setPreferredSize(new Dimension(200, 200));
        this.setBackground(Color.lightGray);

        // Menue-Leiste:
        JMenuBar menuBar = new JMenuBar();
        menuBar.setOpaque(true);
        menuBar.setBackground(Color.green);
        menuBar.setPreferredSize(new Dimension(200, 20));
        menuBar.setToolTipText("Dies ist eine Menü-Leiste");

        // Panel (Bedienfeld):
        JPanel panel = new JPanel();
        panel.setBackground(Color.blue);
        panel.setPreferredSize(new Dimension(100, 90));

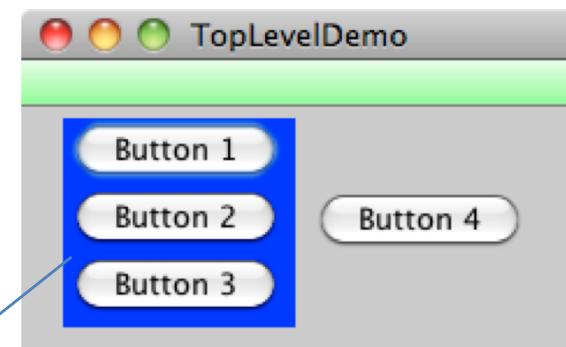
        ...
    }
}
```

Titel, Größe und Hintergrundfarbe des Hauptfensters definieren

Menüleiste mit Hinweistext (tool tip) definieren

Panel (Bedienfeld) definieren

JPanel

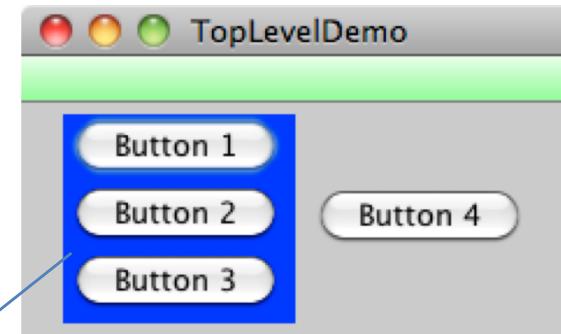


# TopLevelDemo in Java (2)

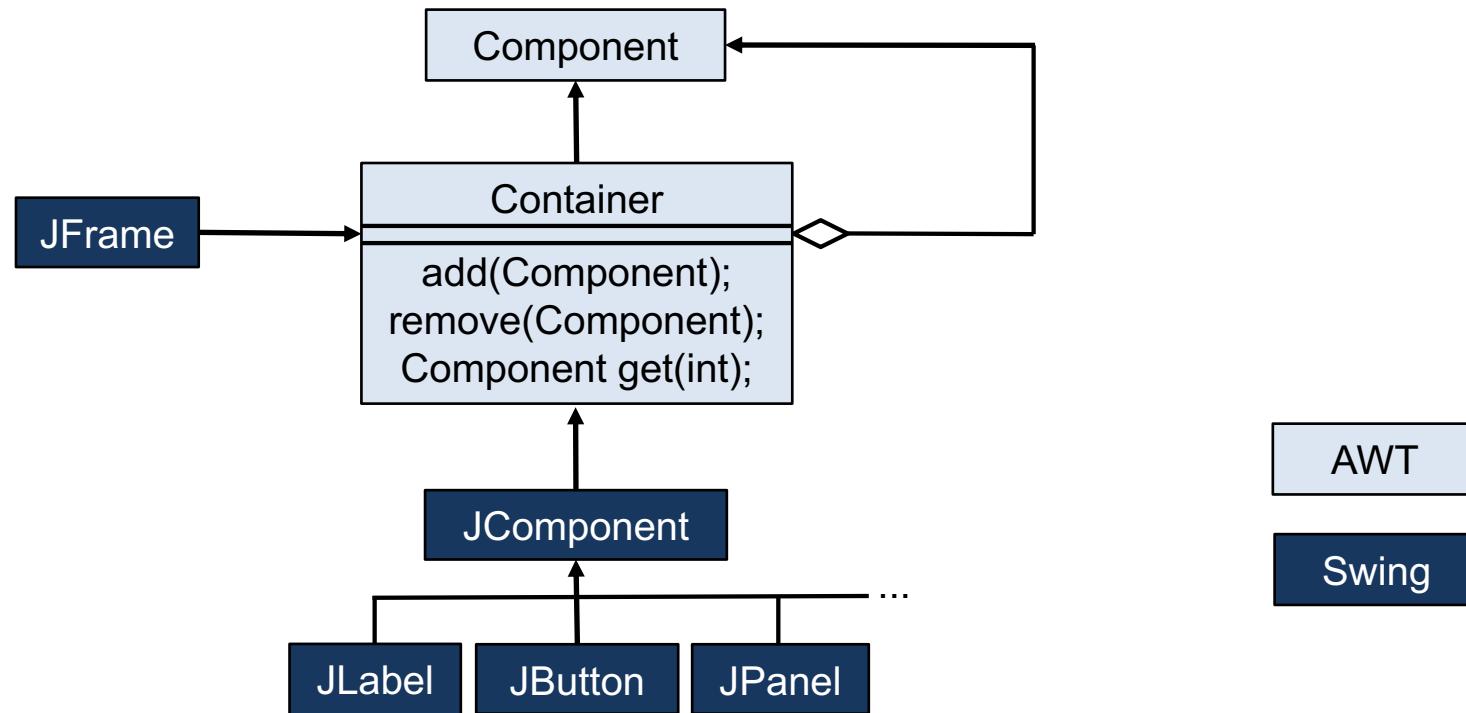
```
// Schaltflächen  
JButton button1 = new JButton("Button 1");  
JButton button2 = new JButton("Button 2");  
JButton button3 = new JButton("Button 3");  
JButton button4 = new JButton("Button 4");  
  
// Panel zusammenbauen:  
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));  
panel.add(button1);  
panel.add(button2);  
panel.add(button3);  
  
// Hauptfenster zusammenbauen:  
this.setMenuBar(menuBar);  
this.setLayout(new FlowLayout());  
this.add(panel);  
this.add(button4);  
  
// Hauptfenster ausgeben:  
this.pack();  
this.setVisible(true);  
}  
  
public static void main(String[ ] args) {  
    JFrame myApp = new ContainerDemo();  
}
```

Schaltflächen definieren.

Zusammenbau der Komponenten ergibt hierarchische Containerstruktur.  
Layouts zum Anordnen der Komponenten festlegen:  
BoxLayout für Panel und FlowLayout für Hauptfenster.



# Entwurfsmuster Kompositum



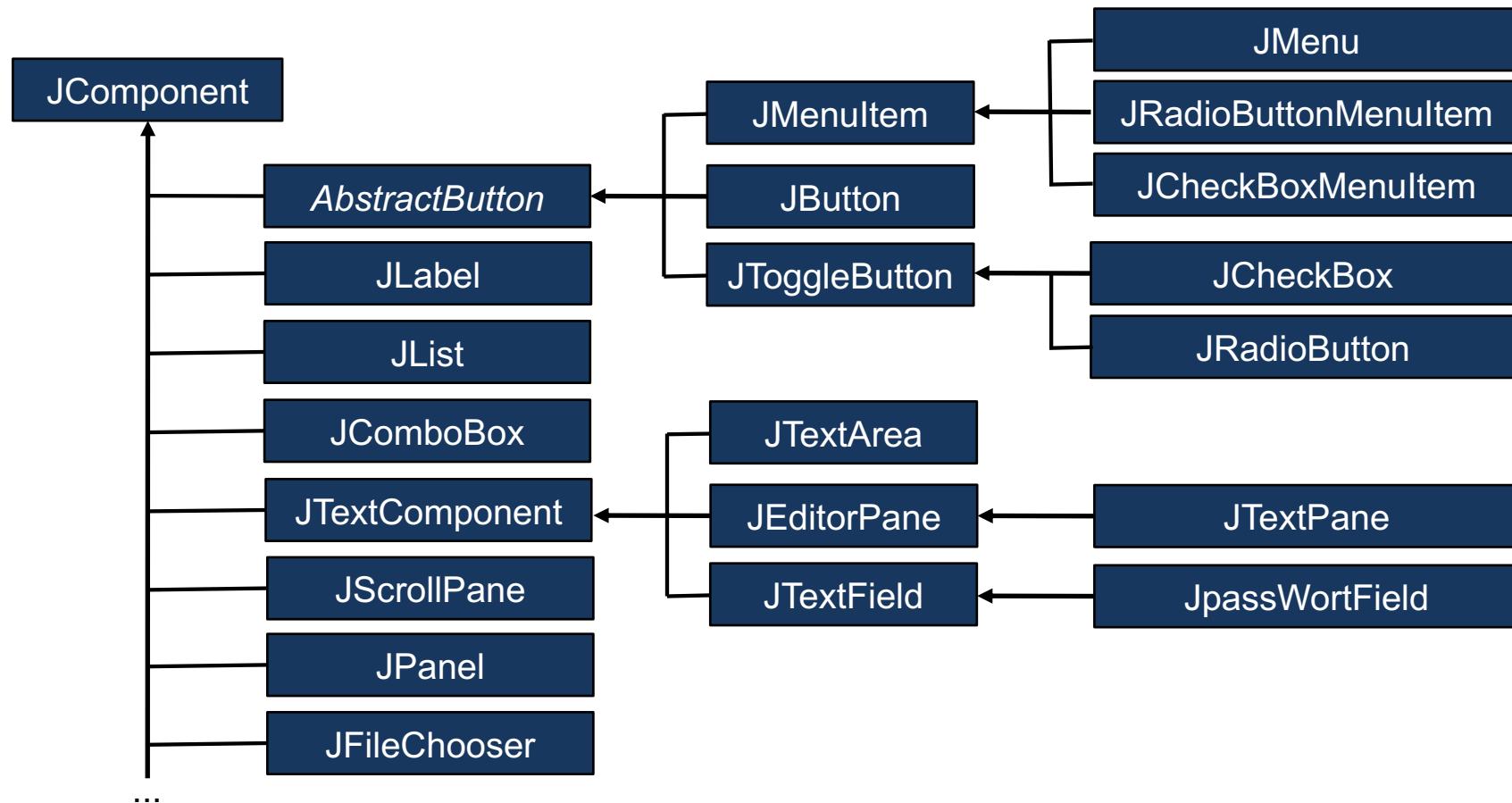
- Die Klasse `Container` folgt dem Entwurfsmuster Kompositum (siehe Kapitel 15).
- Mit dem Entwurfsmuster Kompositum lassen sich Objekte zu Baumstrukturen zusammenbauen.

# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen mit Swing

# Swing-Komponenten (JComponent)

- Die Steuer- und Bedienelemente einer grafischen Benutzeroberfläche werden als Swing-Komponenten (JComponent) realisiert.
- Folgendes Klassen-Diagramm zeigt die verschiedenen Komponentenklassen (nicht vollständig)



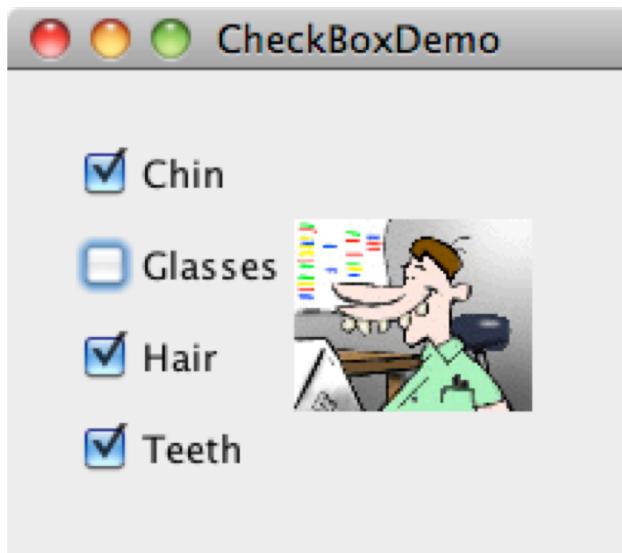
# Visuelle Übersicht über Swing-Komponenten



- Seite mit visueller Übersicht gibt es leider nicht mehr direkt bei Oracle
- Seite ist jedoch im Internet auffindbar.  
Einfach in eine Suchmaschine [A Visual Guide to Swing Components](#) eingeben.
- Die zugehörigen [Swing-Demos](#) (Java-Quellcode und JNLP-Dateien direkt zum Starten mit Java Web Start) gibt es hier:  
<http://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html>

# Auswahlknopf (JCheckBox) (1)

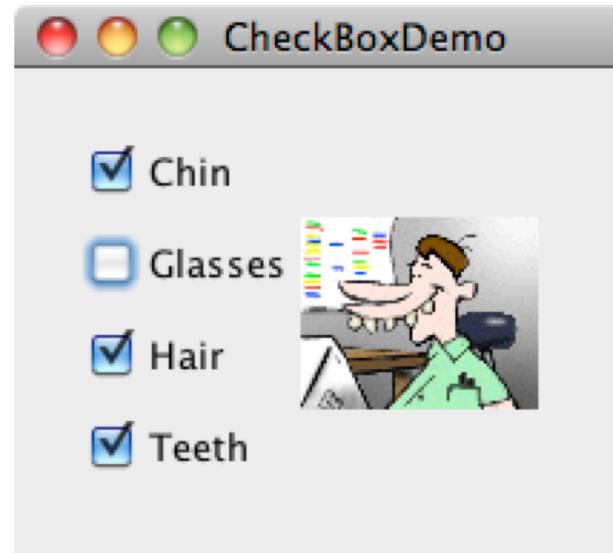
- Ein Auswahlknopf (Check Box) kann ein- oder ausgeschaltet werden.
- Oft werden Auswahlknöpfe zu einem Auswahlfeld zusammengefasst.
- Es kommt der Beobachter ItemListener zum Einsatz



```
public class CheckBoxDemo extends JFrame  
implements ItemListener {  
  
    JCheckBox chinButton;  
    JCheckBox glassesButton;  
    JCheckBox hairButton;  
    JCheckBox teethButton;  
  
    StringBuffer choices; // Zum Speichern der aktuellen Auswahl  
    JLabel pictureLabel; // Aktuelles Bild  
  
    public CheckBoxDemo() {...}  
  
    public void itemStateChanged(ItemEvent e) {...}  
  
    public void main(String[]){  
        new CheckBoxDemo();  
    }  
}
```

# Auswahlknopf (JCheckBox) (2)

```
public CheckBoxDemo() {  
    setTitle("CheckBoxDemoDemo");  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    chinButton = new JCheckBox("Chin");  
    chinButton.setSelected(true);  
  
    glassesButton = new JCheckBox("Glasses");  
    glassesButton.setSelected(false);  
  
    hairButton = new JCheckBox("Hair");  
    hairButton.setSelected(true);  
  
    teethButton = new JCheckBox("Teeth");  
    teethButton.setSelected(true);  
  
    chinButton.addItemListener(this);  
    glassesButton.addItemListener(this);  
    hairButton.addItemListener(this);  
    teethButton.addItemListener(this);  
  
    choices = "c-ht"; // chin, no glasses, hair, teeth  
  
    pictureLabel = new JLabel();  
    updatePicture();
```



```
// Put the check boxes in a column in a panel  
JPanel checkPanel  
    = new JPanel(new GridLayout(0, 1));  
checkPanel.add(chinButton);  
checkPanel.add(glassesButton);  
checkPanel.add(hairButton);  
checkPanel.add(teethButton);  
add(checkPanel, BorderLayout.LINE_START);  
add(pictureLabel, BorderLayout.CENTER);  
  
pack();  
setVisible(true);  
}
```

# Auswahlknopf (JCheckBox) (3)

```
private void updatePicture() {
    ImageIcon icon = new ImageIcon( "images/geek-" + choices.toString() + ".gif");
    pictureLabel.setIcon(icon);
}
```

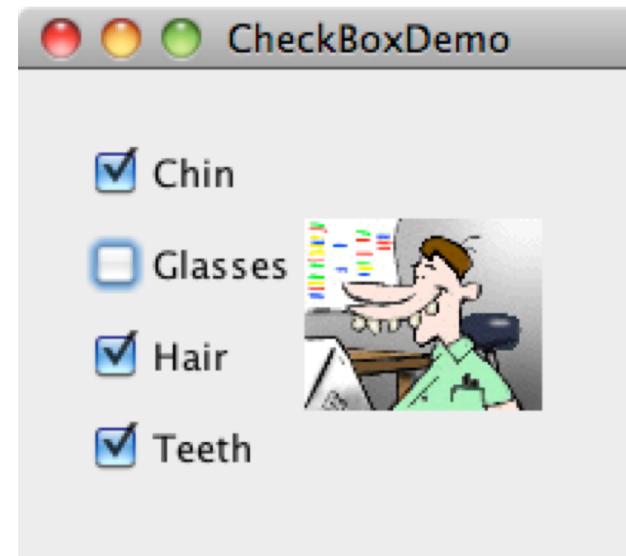
```
public void itemStateChanged(ItemEvent e) {
    int index;
    char c;
    Object source = e.getItemSelectable();

    if (source == chinButton) {
        index = 0; c = 'c';
    } else if (source == glassesButton) {
        index = 1; c = 'g';
    } else if (source == hairButton) {
        index = 2; c = 'h';
    } else if (source == teethButton) {
        index = 3; c = 't';

        if (e.getStateChange() == ItemEvent.DESELECTED)
            c = '-';
    }

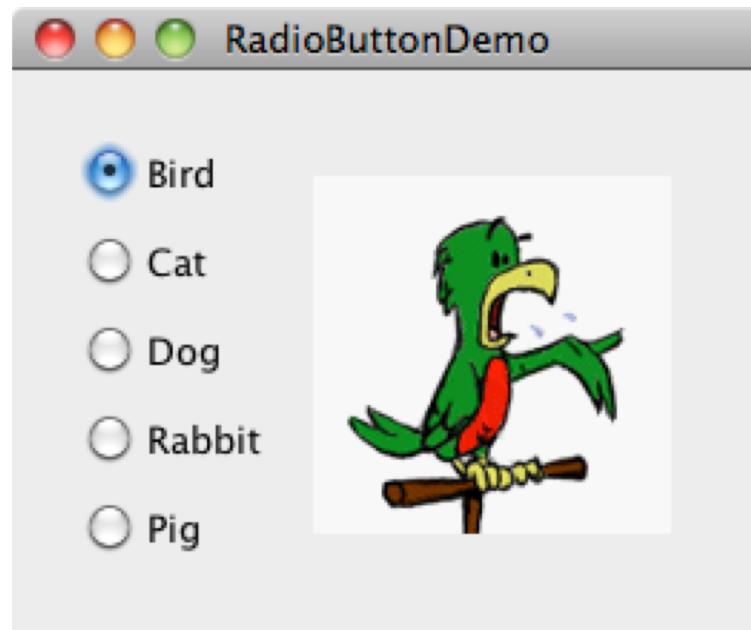
    choices.setCharAt(index, c);
    updatePicture();
}
```

Es gibt 16 unterschiedliche Bilder:  
geek-cght.gif,  
geek-cgh-.gif,  
geek-cg-t.gif,  
...



# Tastknopf (JRadioButton) (1)

- Ein Tastknopf (Radio Button) kann ebenfalls ein- oder ausgeschaltet werden.
- Mehrere Tastknöpfe werden zu einer Gruppe (ButtonGroup) zusammengefasst, wobei dann immer nur genau ein Knopf gedrückt ist (wie bei Senderwahltaste am Radio)
- Hier kommt der Beobachter ActionListener zum Einsatz



# Tastknopf (JRadioButton) (2)

```
JRadioButton birdButton;  
JRadioButton catButton ;  
...  
JLabel picture;  
  
public RadioButtonDemo() {  
    ...  
    birdButton = new JRadioButton("Bird");  
    birdButton.setSelected(true);  
    catButton = new JRadioButton("Cat");  
    ...  
    ButtonGroup group = new ButtonGroup();  
    group.add(birdButton);  
    group.add(catButton);  
    ...  
    birdButton.addActionListener(this);  
    catButton.addActionListener(this);  
    ...  
    picture = new JLabel(new ImageIcon("bird.gif"));  
  
    JPanel radioPanel = new JPanel(new GridLayout(0, 1));  
    radioPanel.add(birdButton);  
    radioPanel.add(catButton);  
    ...  
    add(radioPanel, BorderLayout.LINE_START);  
    add(picture, BorderLayout.CENTER);  
    ...  
}
```

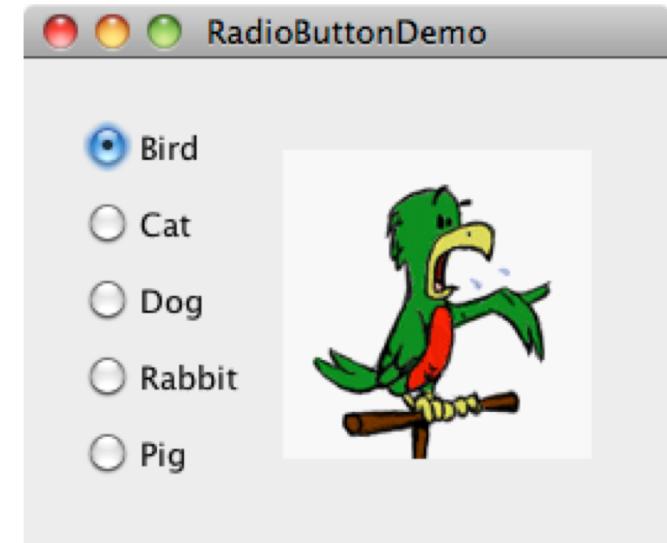
Buttons definieren

Buttons gruppieren

ActionListener registrieren

Buttons zu einem Panel zusammenfassen

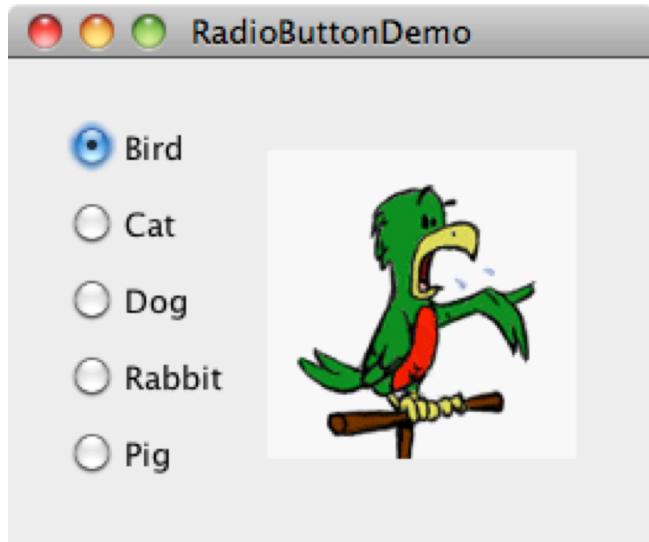
Alles zusammenbauen



# Tastknopf (JRadioButton) (3)

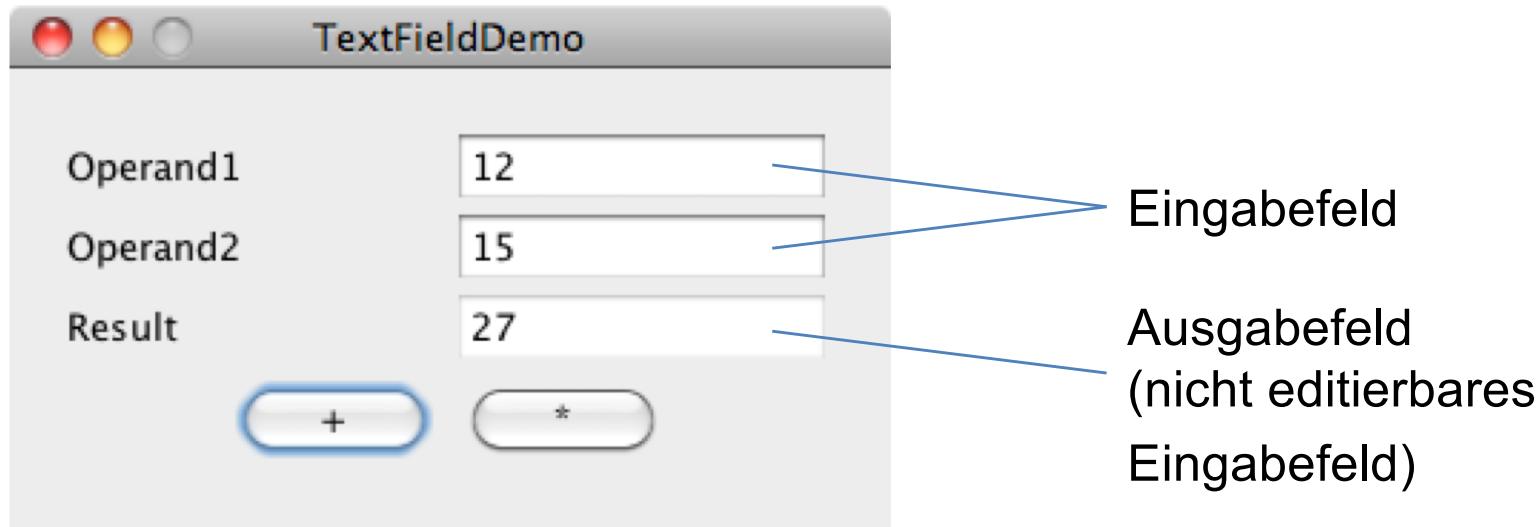
```
public void actionPerformed(ActionEvent e) {  
    Object source = e.getSource();  
    picture.setIcon(new ImageIcon((JButton)source.getText() + ".gif"));  
}
```

Action-Listener



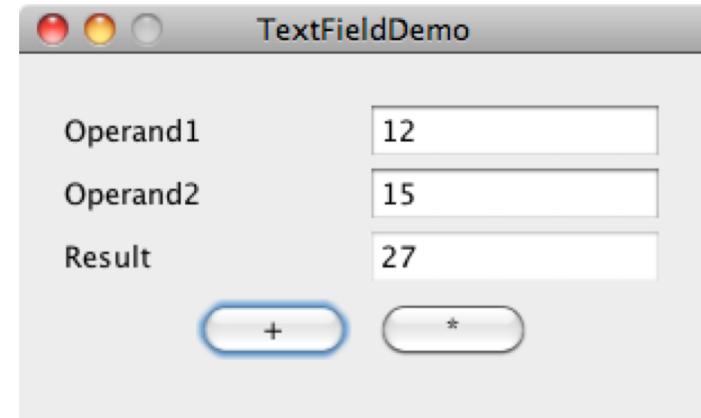
# Texteingabefeld (JTextField) (1)

- Texteingabefeld dient zur Eingabe von einzeiligen Texten.
- Beispiel: Mini-Taschenrechner



# Texteingabefeld (JTextField) (2)

```
public class TextFieldDemo extends JFrame  
implements ActionListener {  
  
    JTextField op1TextField;  
    JTextField op2TextField;  
    JTextField resTextField;  
    JButton sumButton;  
    JButton multButton;  
  
    public TextFieldDemo() {  
        setTitle("TextFieldDemo");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JLabel op1Label = new JLabel("Operand1");  
        JLabel op2Label = new JLabel("Operand2");  
        JLabel resLabel = new JLabel("Result");  
        op1TextField = new JTextField("0",10);  
        op2TextField = new JTextField("0",10);  
        resTextField = new JTextField("0",10);  
        resTextField.setEditable(false);  
        sumButton = new JButton("+");  
        multButton = new JButton("*");  
  
        sumButton.addActionListener(this);  
        multButton.addActionListener(this);
```



Labels, Textfields und Buttons definieren

Bei Buttons ActionListener registrieren

# Texteingabefeld (JTextField) (3)

```
JPanel panel1 = new JPanel();
panel1.setLayout(new GridLayout(3,2));
panel1.add(op1Label);
panel1.add(op1TextField);
panel1.add(op2Label);
panel1.add(op2TextField);
panel1.add(resLabel);
panel1.add(resTextField);
```

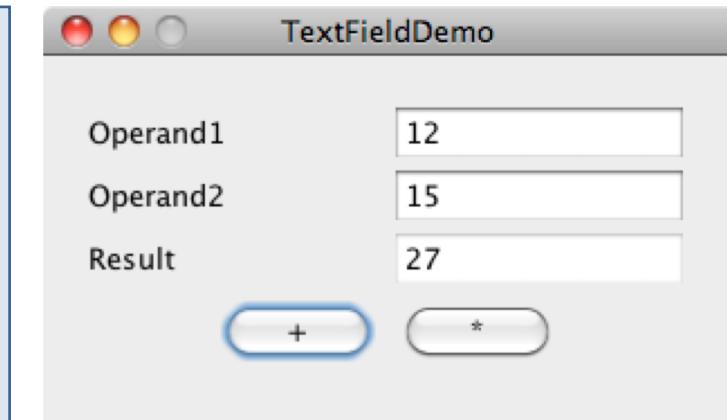
Textfields und Labels zu Panel zusammenfassen

```
JPanel panel2 = new JPanel();
panel2.add(sumButton);
panel2.add(multButton);
```

Buttons zu Panel zusammenfassen

```
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel,BoxLayout.Y_AXIS));
panel.add(panel1);
panel.add(panel2);
panel.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
setContentPane(panel);
```

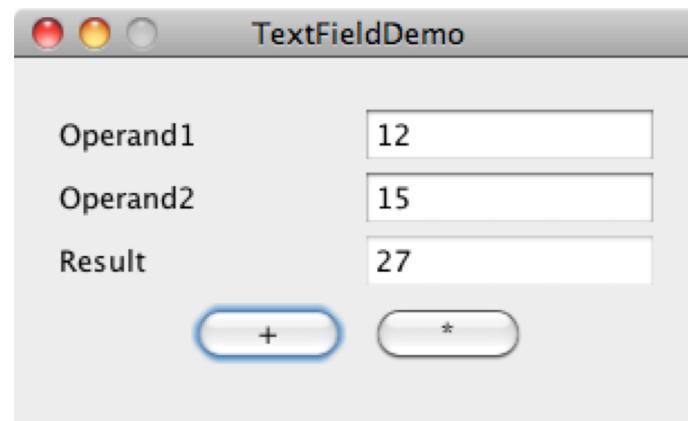
```
pack();
setResizable(false);
setVisible(true);
}
```



Die beiden Panels zu einem Panel zusammenfassen und mit leerer Umrandung versehen und in Hauptfenster einbauen.

# Texteingabefeld (JTextField) (4)

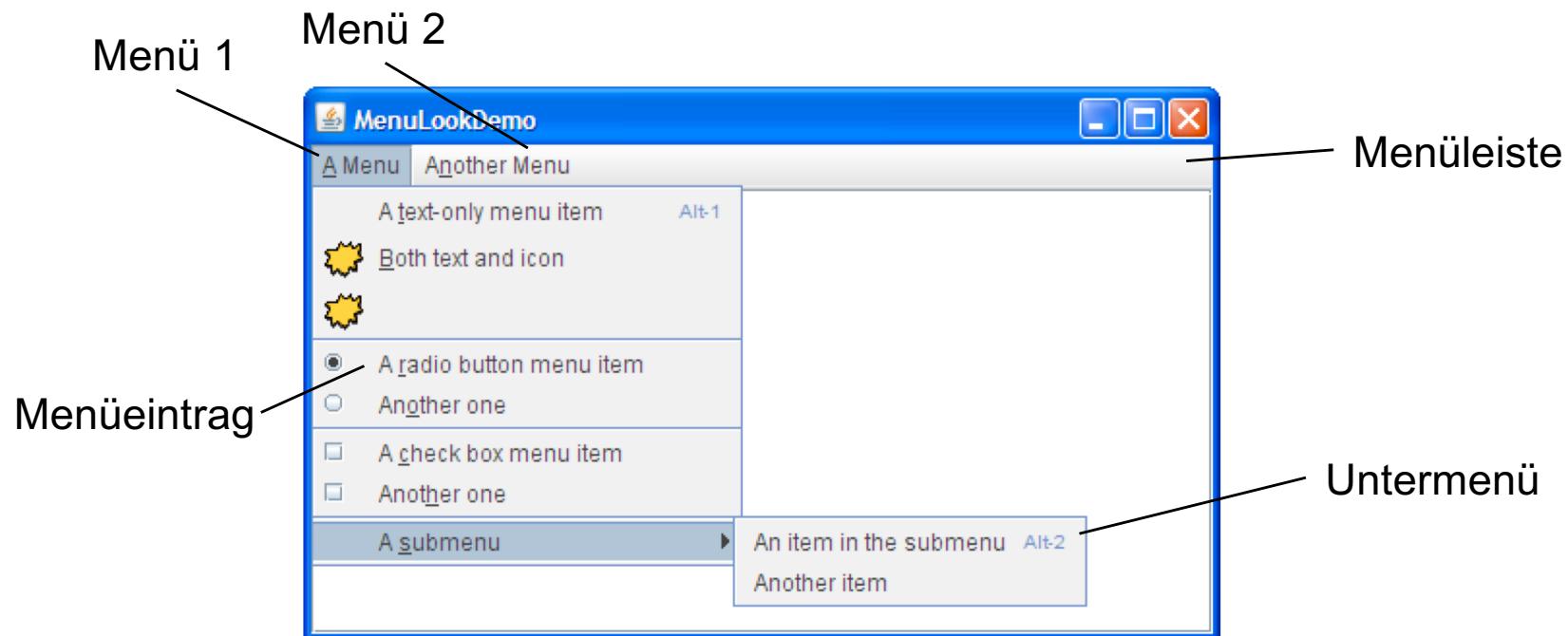
```
public void actionPerformed(ActionEvent e) {  
    Object source = e.getSource();  
    String s1 = op1TextField.getText();  
    String s2 = op2TextField.getText();  
    int o1 = Integer.parseInt(s1);  
    int o2 = Integer.parseInt(s2);  
    if (source == sumButton) {  
        resTextField.setText("'" + (o1+o2));  
    }  
    else if (source == multButton) {  
        resTextField.setText("'" + (o1*o2));  
    }  
  
    public static void main(String[] args) {  
        JFrame myApplication = new TextFieldDemo();  
    }  
}
```



ActionListener führt die gewünschte Berechnung durch (addiere oder multipliziere) und schreibt das Ergebnis in das Resultatsfeld

# Menüs (JMenu) (1)

- Menüs (JMenu) werden der Menüleiste (JMenuBar) als sogenannte Drop-Down Menüs hinzugefügt.
- Ein Menü besteht aus einer Folge von Menüeinträgen ( JMenuItem ) oder weiteren Untermenüs



- Außerdem gibt es **Popup-Menüs (JPopupMenu)**, die üblicherweise durch Klicken der rechten Maustaste erscheinen.

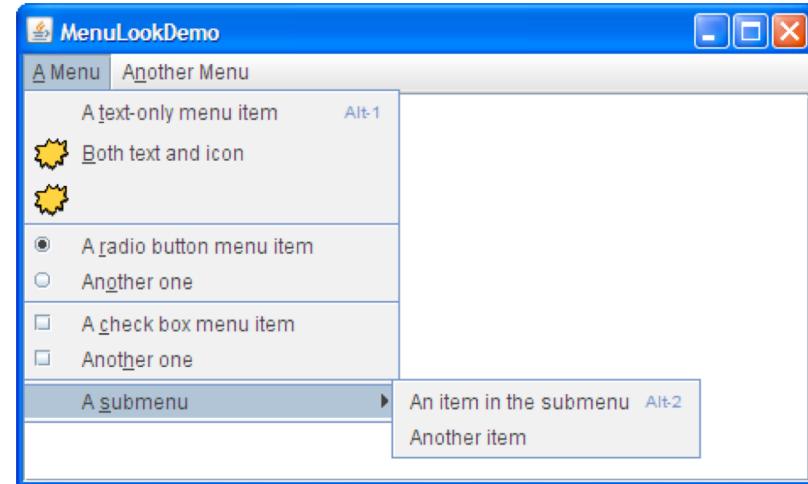
# Menüs (JMenu) (2)

```
// Menubar:  
JMenuBar menuBar = new JMenuBar();  
myFrame.setJMenuBar(menuBar);
```

```
// Menu 1:  
JMenu menu1 = new JMenu("A Menu");  
menu1.setMnemonic(KeyEvent.VK_A);  
menuBar.add(menu1);
```

```
JMenuItem menuItem1  
    = new JMenuItem("A text-only menu item");  
menuItem1.setMnemonic(KeyEvent.VK_T);  
menuItem1.addActionListener(this);  
menu1.add(menuItem1);
```

```
ImageIcon icon = createlmagelIcon("images/middle.gif");  
JMenuItem menuItem2 =  
    = new JMenuItem("Both text and icon", icon);  
menuItem2.setMnemonic(KeyEvent.VK_B);  
menuItem2.addActionListener(this);  
menu1.add(menuItem2);  
  
...
```



```
// Submenu:  
menu1.addSeparator();  
JMenu submenu = new JMenu("A submenu");  
submenu.setMnemonic(KeyEvent.VK_S);  
...  
menu1.add(submenu);
```

```
// Menu 2:  
JMenu menu2 = new JMenu("Another Menu");  
menu2.setMnemonic(KeyEvent.VK_N);  
menuBar.add(menu2);  
...
```

# Aufklappbare Auswahlliste (JComboBox)

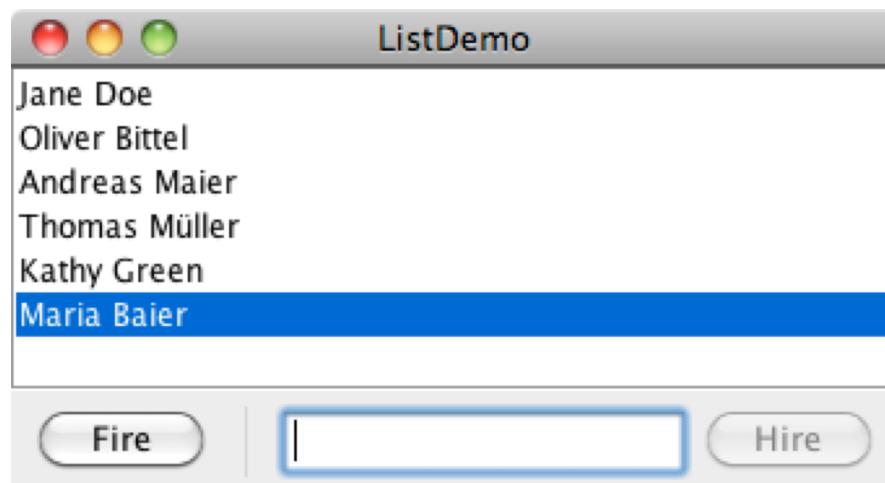
- Eine ComboBox ist eine Schaltfläche, die bei Drücken eine Auswahlliste aufklappt.
- Es kann nur ein Element aus der Auswahlliste ausgewählt werden.
- Es gibt auch ComboBoxen, die mit einem Eingabefeld statt einer Schaltfläche versehen sind. Der Benutzer kann dann entweder freien Text eingeben oder die Auswahlliste aufklappen.



```
String[ ] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };
JComboBox petList = new JComboBox(petStrings);
petList.setSelectedIndex(4);
petList.addActionListener(myFrame);
```

# Auswahlliste (JList) (1)

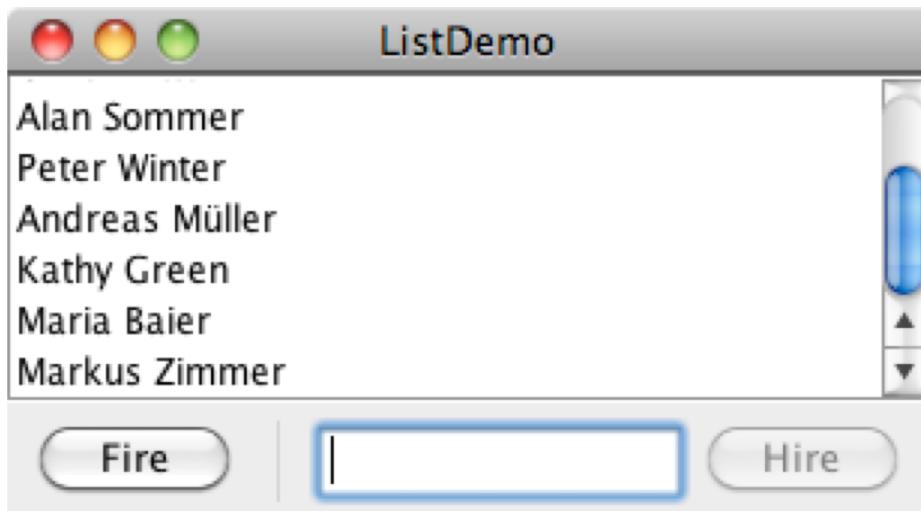
- Eine **JList** ist eine Auswahlliste, die bereits aufgeklappt ist.
- Es können beliebig viele Einträge der Liste ausgewählt werden (sofern der Selektionsmodus entsprechend gesetzt ist).
- Eine JList bietet über ein Listen-Modell (Interface **ListModel**) die Möglichkeit, Elemente einzufügen oder zu löschen.



```
DefaultListModel listModel  
= new DefaultListModel();  
listModel.addElement("Jane Doe");  
listModel.addElement("Oliver Bittel");  
listModel.addElement("Andreas Maier");  
...  
JList list = new JList(listModel);  
list.setSelectionMode(SINGLE_SELECTION);  
list.setSelectedIndex(5);  
...
```

# Auswahlliste (JList) (2)

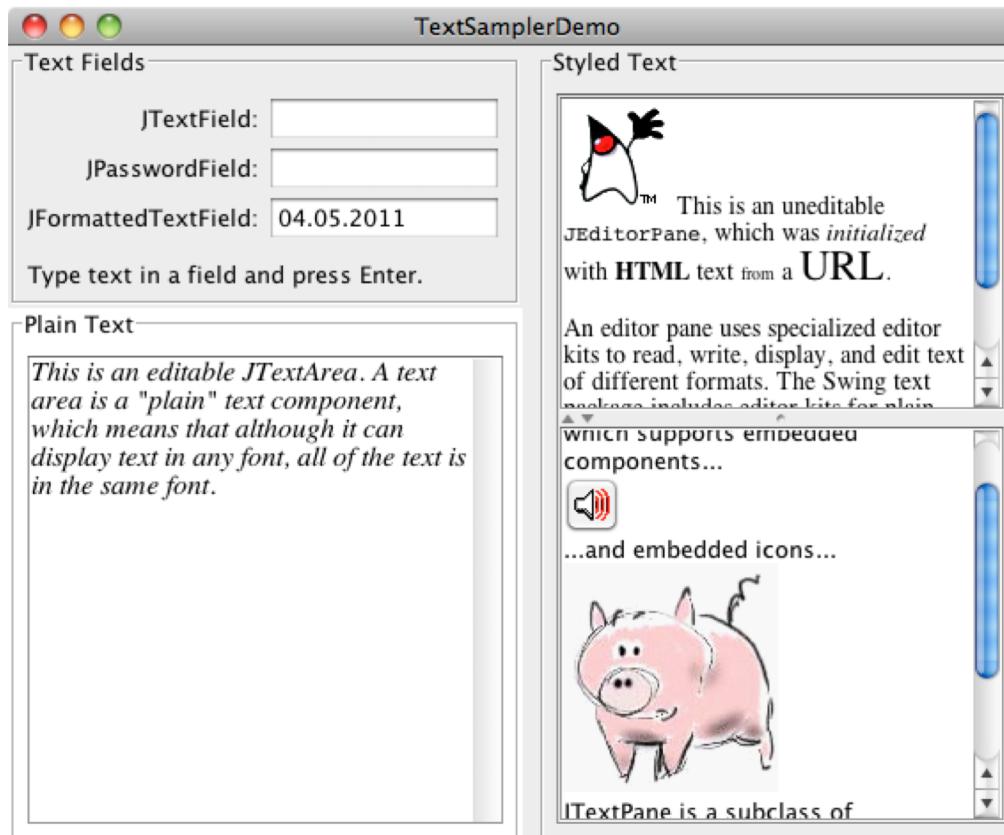
- Eine JList wird üblicherweise in einen scrollbaren Behälter (JSScrollPane) gepackt.



```
...
JList list = new JList(listModel);
...
JSScrollPane listScrollPane = new JSScrollPane(list);
myFrame.add(listScrollPane);
```

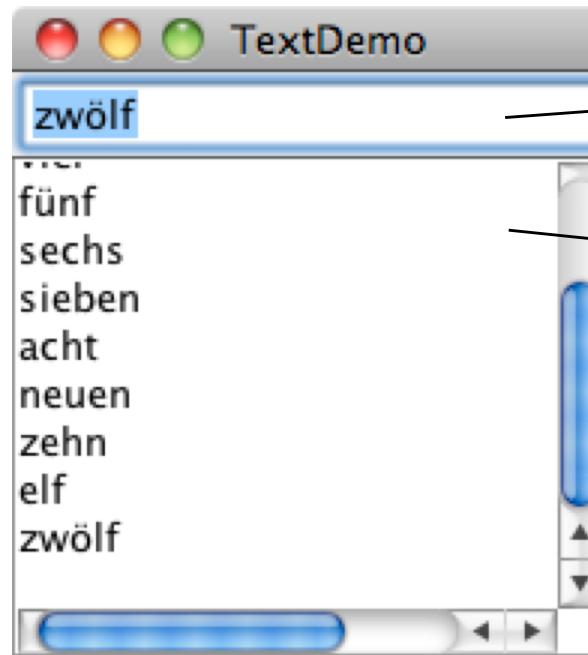
# Textkomponenten (JTextComponent)

- Textfelder ([JTextField](#)) kennen wir bereits.
- Mit [JTextArea](#) lassen sich als mehrzeilige Textfelder realisieren.
- Möchte man verschiedene Schriftarten mischen und auch Bilder dazufügen, dann gibt es noch [JEditorPane](#) und [JTextPane](#).



# Mehrzeiliger Text (JTextArea)

```
public class TextDemo extends JFrame  
implements ActionListener {  
    JTextField textField;  
    JTextArea textArea;  
  
    public TextDemo () {  
        this.setTitle("TextDemo");  
        this.setDefaultCloseOperation(...);  
        this.setLayout(...);  
  
        // JTextField:  
        textField = new JTextField(20);  
        textField.addActionListener(this);  
        this.add(textField);  
  
        // TextArea:  
        textArea = new JTextArea(5, 20);  
        textArea.setEditable(false);  
        JScrollPane scrollPane  
            = new JScrollPane(textArea);  
        this.add(scrollPane);  
  
        this.pack();  
        this.setVisible(true);  
    }  
}
```



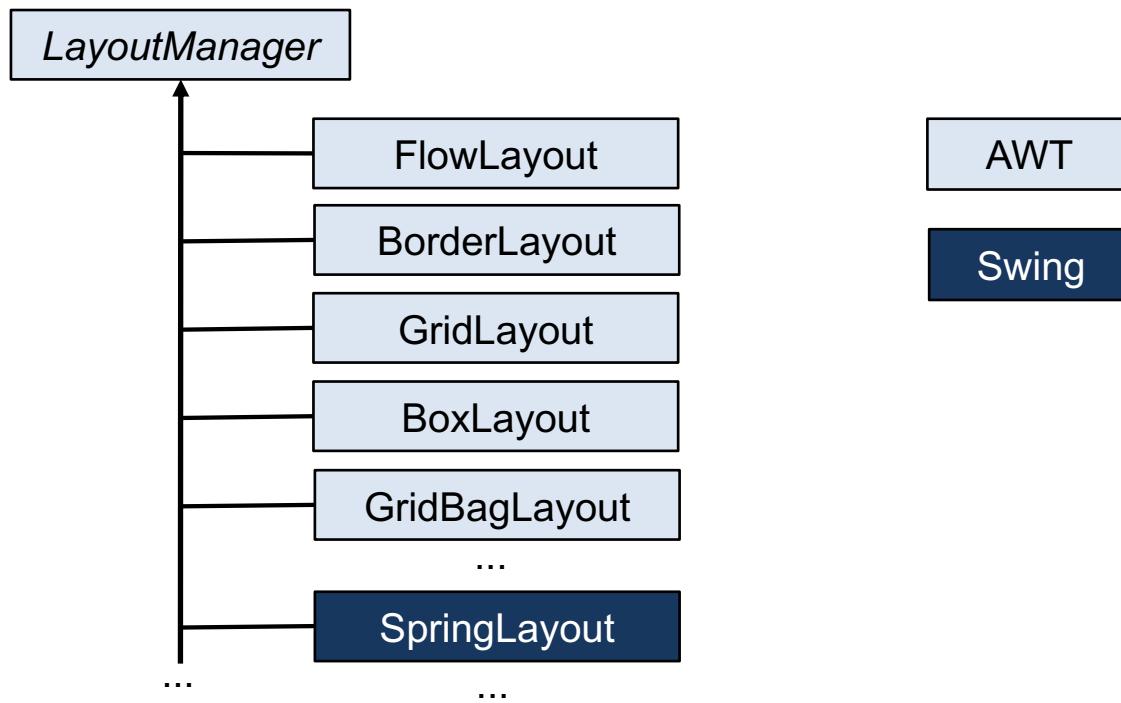
```
public void actionPerformed(ActionEvent e) {  
    String text = textField.getText();  
    textArea.append(text + "\n");  
    textField.selectAll();  
    textArea.setCaretPosition(textArea.getDocument().getLength());  
}  
  
public void main(String[ ]) {  
    JFrame demo = new TextDemo();  
}
```

# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen

# Layout-Manger

- Layout-Manager sind verantwortlich für die Anordnung und die Größen der verschiedenen Komponenten, die zu einem Container gehören.



- Festlegen des Layouts für einen Container:

```
container.setLayout(someLayoutManager);
```

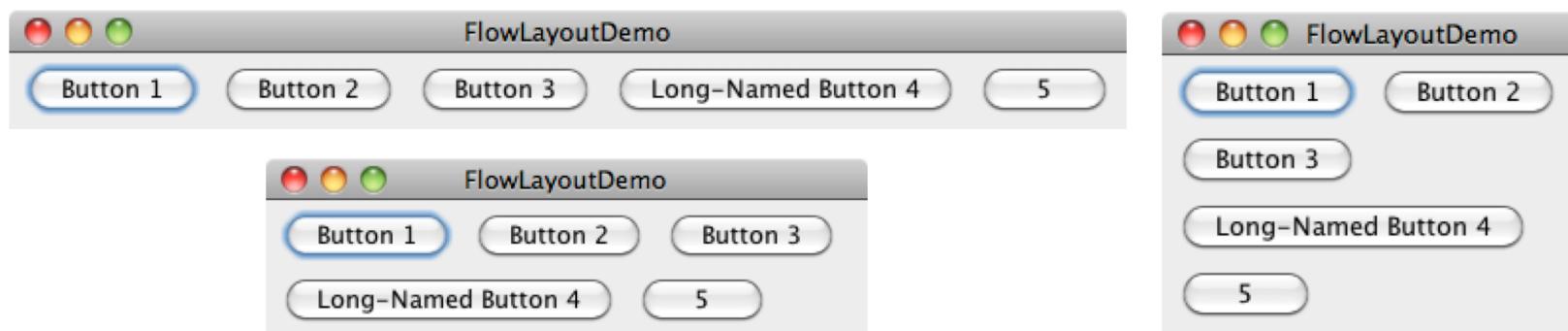
- Standard-Layout für alle Container ist `BorderLayout`.  
Ausnahme: `JPanel` hat ein `FlowLayout`.

# FlowLayout

- Die Komponenten werden zeilenweise von links nach rechts angeordnet. Falls kein Platz mehr ist, wird eine neue Zeile begonnen.

```
public class FlowLayoutDemo extends JFrame {  
    public FlowLayoutDemo() {  
        this.setTitle("FlowLayoutDemo");  
        this.setLayout(new FlowLayout(FlowLayout.LEFT));  
        this.add(new JButton("Button 1"));  
        this.add(new JButton("Button 2"));  
        this.add(new JButton("Button 3"));  
        this.add(new JButton("Long-Named Button 4"));  
        this.add(new JButton("5"));  
        ...  
    }  
    ...
```

FlowLayout mit  
linksbündiger Anordnung

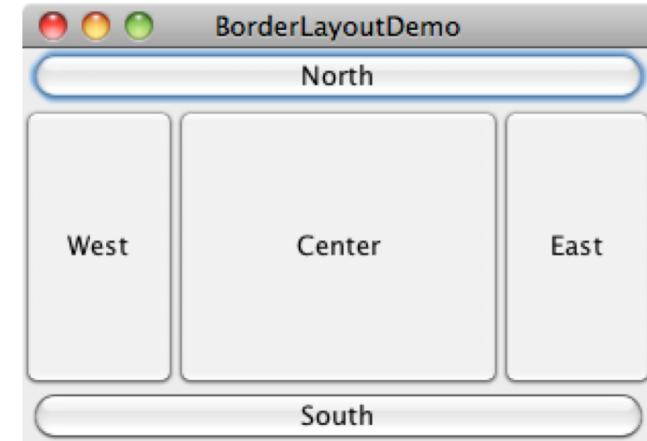
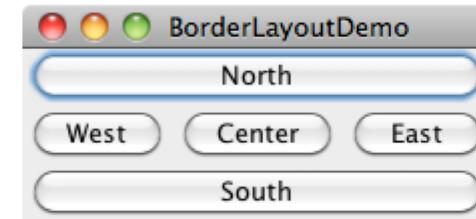


Die Anordnung der Komponenten hängt von der Fensterbreite ab.

# BorderLayout

- Die Fläche des Containers wird in fünf Gebiete eingeteilt:
  - Norden (North, Page-Start),
  - Westen (West, Line-Start),
  - Süden (South, Page-End),
  - Osten (East, Line-End)
  - Zentrum (Center)

```
public class FlowLayoutDemo extends JFrame {  
    public FlowLayoutDemo() {  
        this.setLayout(new BorderLayout());  
        this.add(new JButton("North"),BorderLayout.NORTH);  
        this.add(new JButton("West"),BorderLayout.WEST);  
        this.add(new JButton("South"),BorderLayout.SOUTH);  
        this.add(new JButton("East"),BorderLayout.EAST);  
        this.add(new JButton("Center"),BorderLayout.CENTER);  
        ...  
    }  
    ...  
}
```



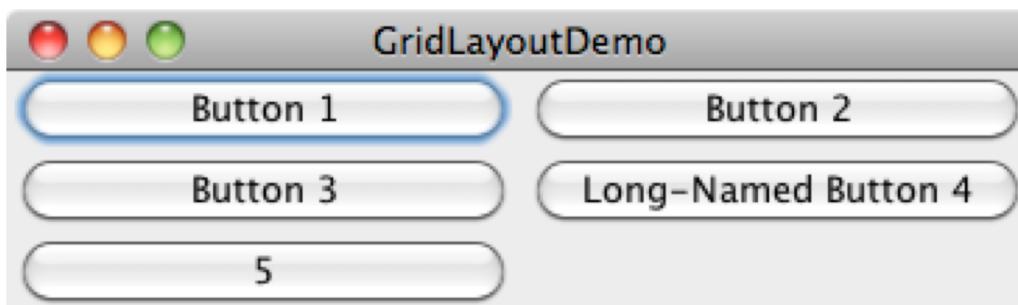
- Bei Vergrößerung des Fensters wächst auch die Breite bzw. die Höhe der Komponenten mit (beim Zentrum wächst beides mit).

# GridLayout

- Die Komponenten werden gitterförmig angeordnet.
- Dabei ist jede Komponente gleich groß und nimmt den verfügbaren Platz komplett ein.

```
public class GridLayoutDemo extends JFrame {  
    public GridLayoutDemo() {  
        this.setLayout(new GridLayout(0,2));  
        this.add(new JButton("Button 1"));  
        this.add(new JButton("Button 2"));  
        this.add(new JButton("Button 3"));  
        this.add(new JButton("Long-Named Button 4"));  
        this.add(new JButton("5"));  
        ...  
    }  
    ...
```

GridLayout mit beliebig vielen  
Zeilen und 2 Spalten  
("0" bedeutet "beliebig viele")



# BoxLayout

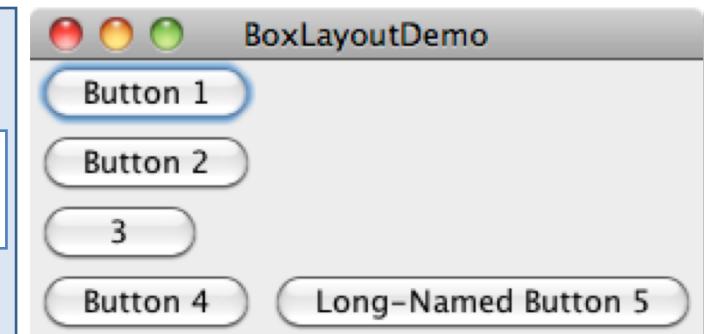
- Ähnlich wie FlowLayout nur mit mehr Möglichkeiten der horizontalen und vertikalen Anordnung.
- Anordnung und Größen der Komponenten bleibt bei Veränderung der Fenstergröße bestehen.

```
public class BoxLayoutDemo extends JFrame {  
    public BoxLayoutDemo() {  
        setTitle("BoxLayoutDemo");  
  
        JPanel panel1 = new JPanel();  
        panel1.setLayout(new BoxLayout(panel1, BoxLayout.PAGE_AXIS));  
        panel1.add(new JButton("Button 1"));  
        panel1.add(new JButton("Button 2"));  
        panel1.add(new JButton("3"));  
  
        JPanel panel2 = new JPanel();  
        panel2.setLayout(new BoxLayout(panel2, BoxLayout.LINE_AXIS));  
        panel2.add(new JButton("Button 4"));  
        panel2.add(new JButton("Long-Named Button 5"));  
  
        this.add(panel1, BorderLayout.CENTER);  
        this.add(panel2, BorderLayout.SOUTH);  
        ...  
    }  
}
```

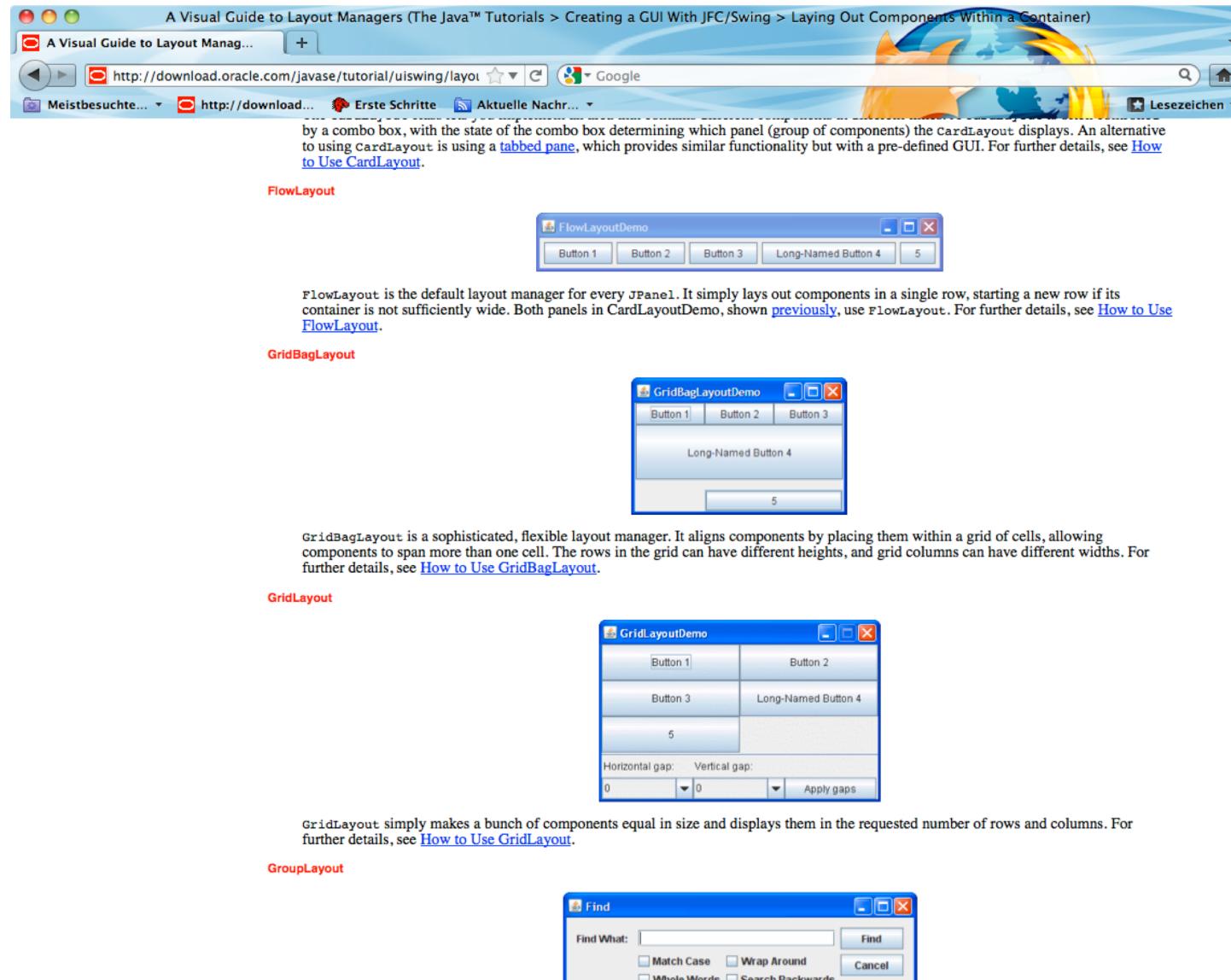
Panel 1 mit vertikaler  
Anordnung

Panel 2 mit horizontaler  
Anordnung

Panel 1 und 2  
in Hauptfenster  
einbauen.



# Weitere LayoutManager



Einen visuellen Überblick über die verschiedenen Layout-Manager findet man im Java-Swing-Tutorial:  
<http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen

# Beispiel einer einfachen Ereignis-Verarbeitung

```
public class Beeper extends JFrame implements ActionListener {  
  
    public Beeper() {  
        ...  
        JButton clickButton = new JButton("Click Me");  
        ...  
        clickButton.addActionListener(this);  
    }  
  
    ...  
  
    public void actionPerformed(ActionEvent e) {  
        // Make a beep sound:  
        Toolkit.getDefaultToolkit().beep();  
    }  
}
```

Definiere einen Button als Ereignisquelle

Registriere beim Button einen ActionListener

Definiere einen ActionListener, der ein ActionEvent verarbeitet.



Beim Clicken des Buttons wird ein ActionEvent ausgelöst und der registrierte ActionListener wird durchgeführt.

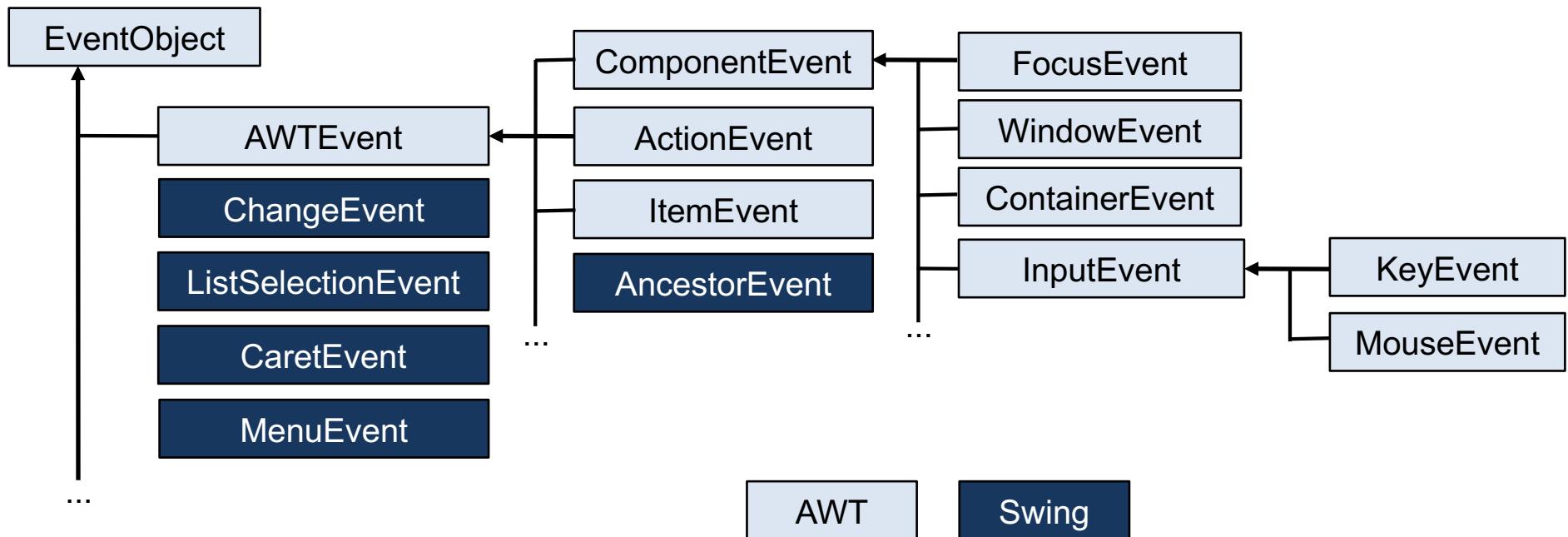
# Systematische Schreibweise

---

	Schreibweise	Beispiele
<b>Event-Klassen</b>	XxxEvent	ActionEvent MouseEvent
<b>EventListener-Interfaces</b>	XxxListener XxxYyyListener	ActionListener MouseListener MouseMotionListener
<b>Registrierung bei einer Komponente c</b>	c.addXxxListener(...) c.addXxxYyyListener(...)	c.addActionListener(...) c.addMouseListener(...) c.addMouseMotionListener(...)

# Events

- **EventObject** ist Oberklasse aller Events
- Events lassen sich in 2 Gruppen aufteilen: Low-Level- und semantische Ereignisse.
- **Low-Level-Ereignisse** werden durch das Betriebssystem oder durch Tastur und Maus (Bewegung, Klicken, ...) generiert.
- Alle ComponentEvents sind Low-Level-Ereignisse.
- Ereignisse, die in Swing einem Bedienelement zugeordnet werden können, sind sogenannte **semantische Ereignisse** (z.B. Klicken einer Schaltfläche)



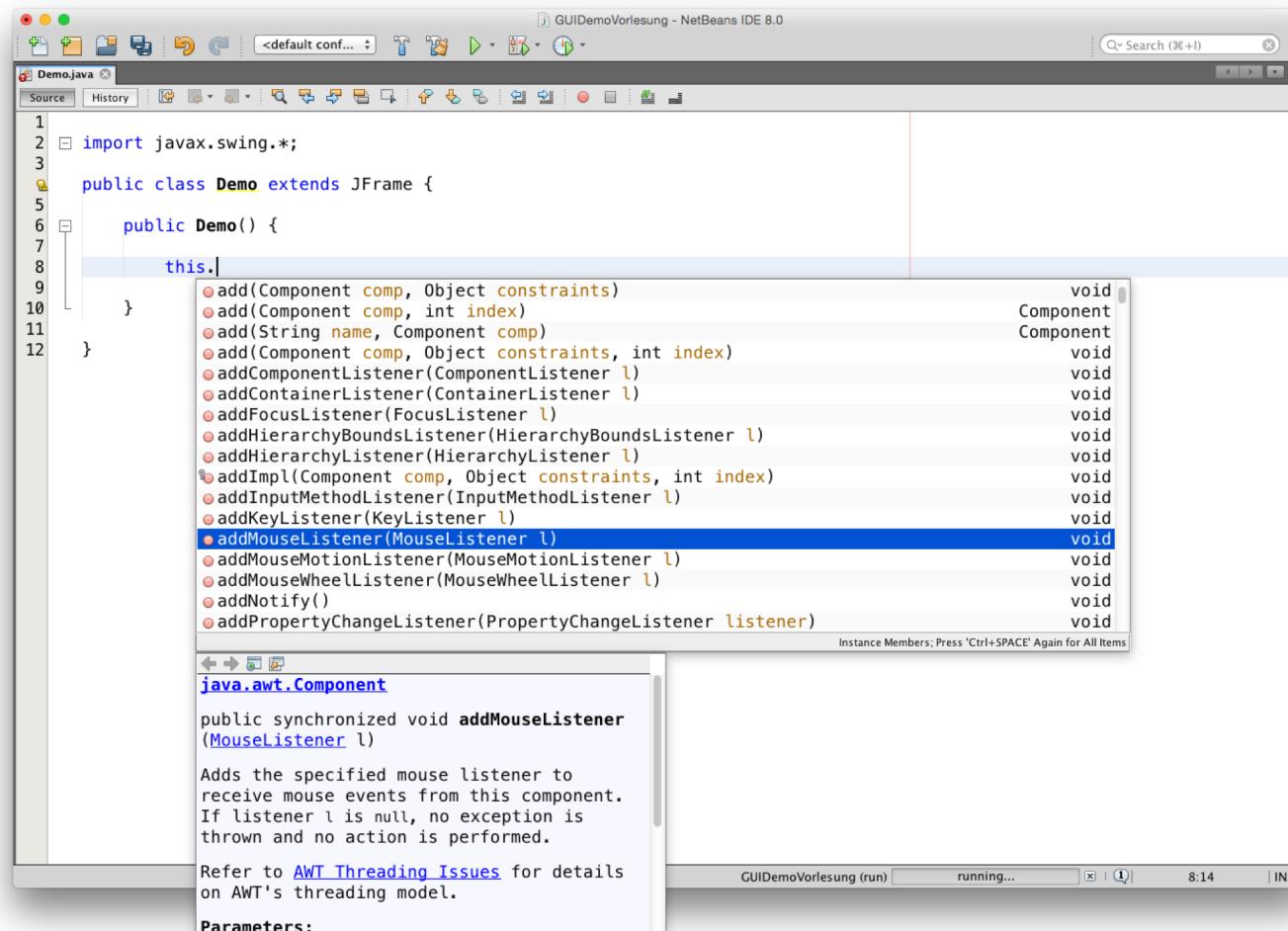
# Wichtige EventListener in der Java API

---

EventListener-Interface	EventListener-Methoden	Event-Klasse	Methoden der Eventklasse
ActionListener	actionPerformed(ActionEvent)	ActionEvent	Object getSource() ...
ItemListener	itemStateChanged(ItemEvent)	ItemEvent	ItemSelectable getItemSelectable() ...
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)	MouseEvent	int getButton() int getX() int getY() int getClickCount()
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)		
...	...	...	

# Komponenten und EventListener

- Über die Java-API lässt sich feststellen, welche EventListener für eine Komponente relevant sind.
- Empfehlung: benutze Code-Ergänzung einer IDE.



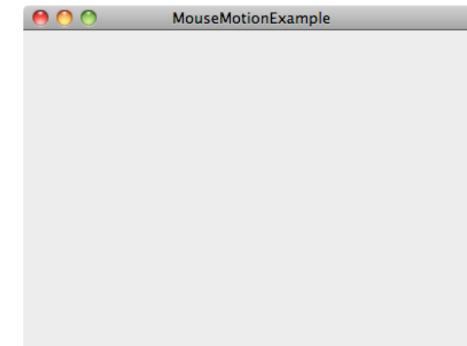
# Beispiel: MouseMotionListener

- Ein JFrame unterstützt den MouseMotionListener.

```
public class MouseMotionDemo  
extends JFrame  
implements MouseMotionListener {  
  
    public MouseMotionDemo() {  
        this.setTitle("MouseMotionExample");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setPreferredSize(new Dimension(400,300));  
        this.addMouseMotionListener(this);  
        this.pack();  
        this.setVisible(true);  
    }  
  
    public void mouseMoved(MouseEvent e) {  
        System.out.println("x = " + e.getX() + ", y = " + e.getY());  
    }  
  
    public void mouseDragged(MouseEvent e) {}  
  
    public static void main(String[] args) {  
        new MouseMotionDemo();  
    }  
}
```

MouseMotionListener wird beim Hauptfenster registriert.  
Beobachter ist die Klasse selbst.

Bei einer Mausbewegung wird die Mausposition ausgegeben.



...  
x = 309, y = 32  
x = 309, y = 28  
x = 366, y = 87  
x = 369, y = 195  
x = 393, y = 28

# Java Tutorial

---

- Das Tutorial zu EventListener enthält zahlreiche Beispiele:  
<https://docs.oracle.com/javase/tutorial/uiswing/examples/events/index.html>

Example	Zip File (contains all files necessary for the example plus NetBeans IDE project metadata)	Source Files (first file has the main method, except for examples that run only as applets)
Beeper <a href="#">[Launch]</a>	Beeper Project	Beeper.java
ComponentEventDemo <a href="#">[Launch]</a>	Component Event Demo Project	ComponentEventDemo.java
ContainerEventDemo <a href="#">[Launch]</a>	Container Event Demo Project	ContainerEventDemo.java
DocumentEventDemo <a href="#">[Launch]</a>	Document Event Demo Project	DocumentEventDemo.java
FocusEventDemo <a href="#">[Launch]</a>	Focus Event Demo Project	FocusEventDemo.java
InternalFrameEventDemo <a href="#">[Launch]</a>	Internal Frame Event Demo Project	InternalFrameEventDemo.java
KeyEventDemo <a href="#">[Launch]</a>	Key Event Demo Project	KeyEventDemo.java
ListDataEventDemo <a href="#">[Launch]</a>	List Data Event Demo Project	ListDataEventDemo.java
ListSelectionDemo <a href="#">[Launch]</a>	List Selection Demo Project	ListSelectionDemo.java
TableListSelectionDemo <a href="#">[Launch]</a>	Table List Selection Demo Project	TableListSelectionDemo.java
MouseEventDemo <a href="#">[Launch]</a>	Mouse Event Demo Project	MouseEventDemo.java BlankArea.java
MouseMotionEventDemo <a href="#">[Launch]</a>	Mouse Motion Event Demo Project	MouseMotionEventDemo.java BlankArea.java
MouseWheelEventDemo <a href="#">[Launch]</a>	Mouse Wheel Event Demo Project	MouseWheelEventDemo.java
MultiListener <a href="#">[Launch]</a>	MultiListener Project	MultiListener.java
TreeExpandEventDemo <a href="#">[Launch]</a>	Tree Expand Event Demo Project	TreeExpandEventDemo.java
TreeExpandEventDemo2 <a href="#">[Launch]</a>	Tree Expand Event 2 Demo Project	TreeExpandEventDemo2.java
WindowEventDemo <a href="#">[Launch]</a>	Window Event Demo Project	WindowEventDemo.java

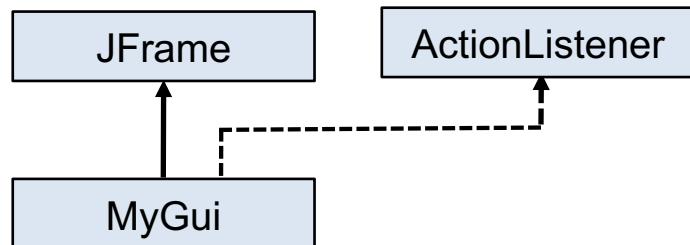
# Implementierung von EventListener

---

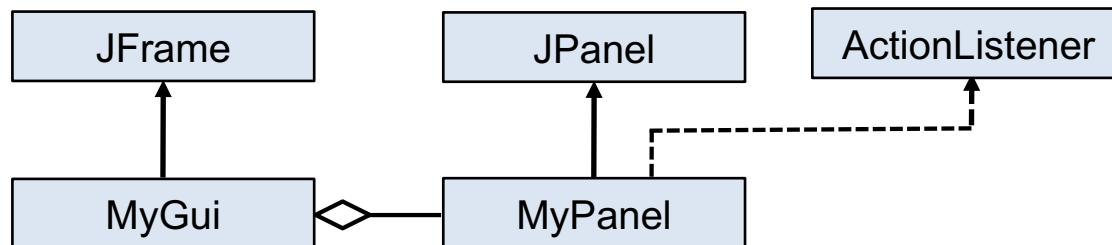
- Komponente ist selbst ein Listener
- Listener als innere Klasse
- Listener als anonyme Klasse
- Listener von einer Adapter-Klasse abgeleitet
- Separate Listener-Klasse.

# Komponente ist ein Listener

- Anwendung MyGui ist ein JFrame und ist selbst ein ActionListener (wie in den Beispielen bisher)



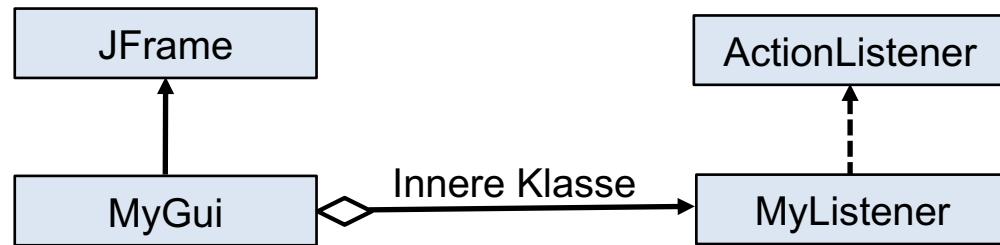
- Die Anwendung MyGui ist ein JFrame und enthält eine Komponente (z.B. JPanel), die ein ActionListener ist (siehe auch Aufgabe 10 im Praktikum)



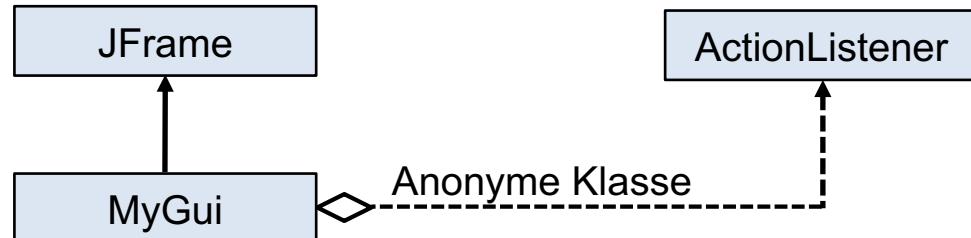
- Die Anwendung kann selbstverständlich aus mehreren Komponenten bestehen, die jeweils Listener sind.

# Listener als innere oder anonyme Klasse

- Anwendung MyGui hat eine **innere Klasse** MyListener die ein ActionListener ist. Ein ActionListener-Objekt hat damit Zugriff auf alle Daten des MyGui-Objekts.



- Benötigt der Listener nur wenig Code, dann bietet sich eine **anonyme Klasse** an.



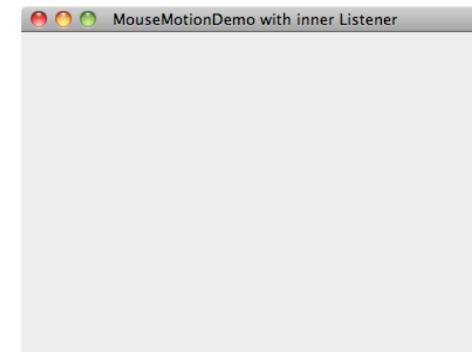
# MouseMotionDemo mit innerer Listener-Klasse

```
public class MouseMotionDemoWithInnerListener extends JFrame {  
  
    public MouseMotionDemoWithInnerListener() {  
        this.setTitle("MouseMotionDemo with inner Listener");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setPreferredSize(new Dimension(400,300));  
        this.addMouseMotionListener(new MyMouseMotionListener());  
        this.pack();  
        this.setVisible(true);  
    }  
  
    private class MyMouseMotionListener  
        implements MouseMotionListener {  
  
        public void mouseMoved(MouseEvent e) {  
            System.out.println("x = " + e.getX() + ", y = " + e.getY());  
        }  
  
        public void mouseDragged(MouseEvent e) {}  
    }  
  
    public static void main(String[] args) {  
        new MouseMotionDemoWithInnerListener();  
    }  
}
```

Registrieren eines  
MouseMotionListener

Innere MouseMotionListener-  
Klasse

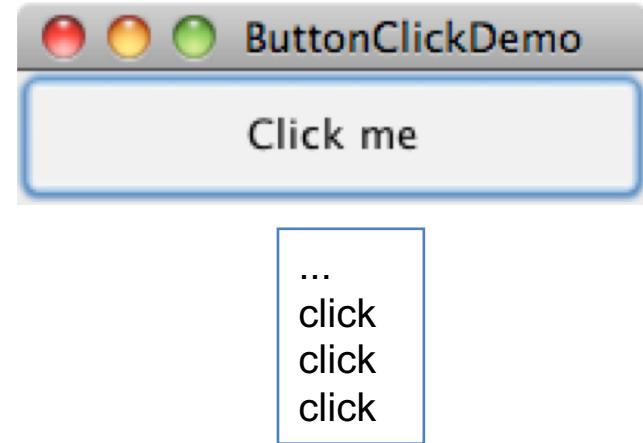
Bei einer Mausbewegung wird  
die Mausposition ausgegeben.



...  
x = 309, y = 32  
x = 309, y = 28  
x = 366, y = 87  
x = 369, y = 195  
x = 393, y = 28

# ButtonClickDemo mit anonymer Listener-Klasse

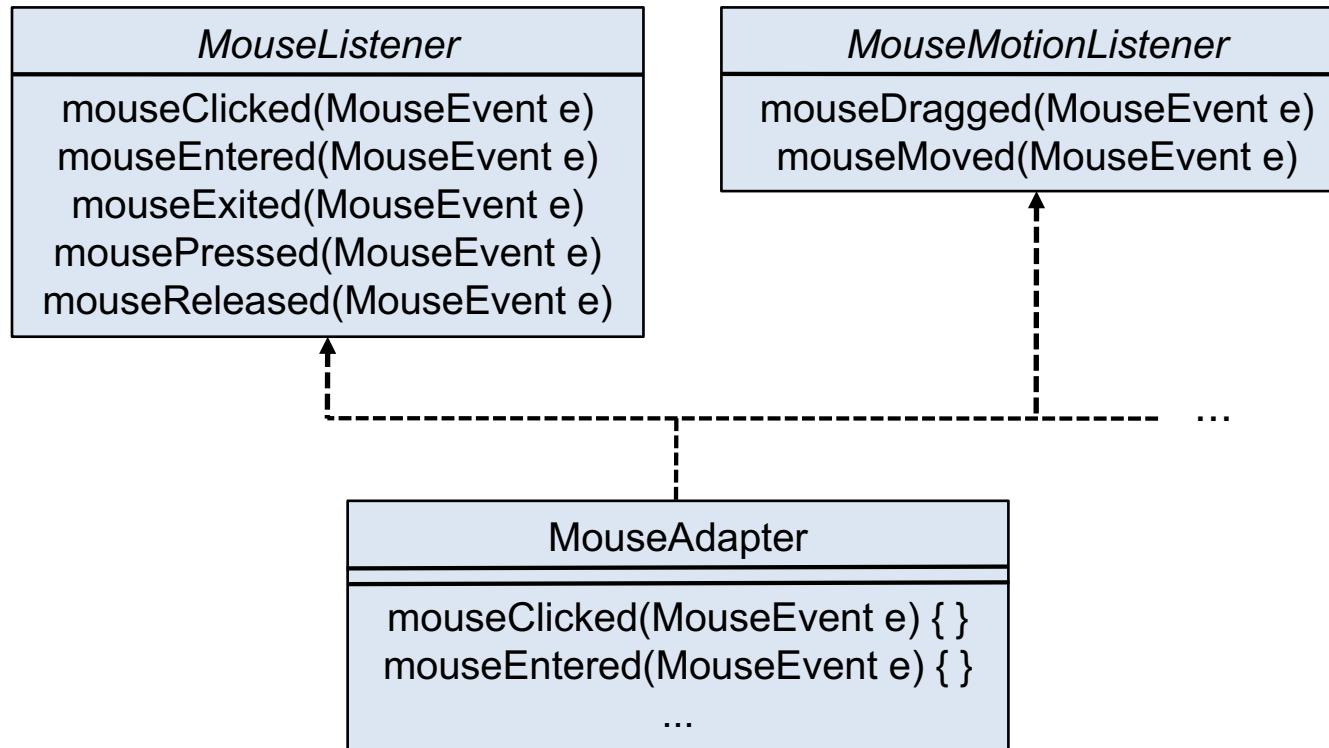
```
public class ButtonClickDemo extends JFrame {  
  
    public ButtonClickDemo() {  
        this.setTitle("ButtonClickDemo");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        JButton button = new JButton("Click me");  
  
        ActionListener buttonListener = new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.out.println("click");  
            }  
        };  
  
        button.addActionListener(buttonListener);  
  
        this.add(button);  
        this.pack();  
        this.setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new ButtonClickDemo();  
    }  
}
```



- Definition einer anonymen Klasse, die das Interface ActionListener implementiert und Erzeugen eines Objekts dieser Klasse.
- Anonyme Klasse verhält sich wie innere Klasse.

# Adapterklassen

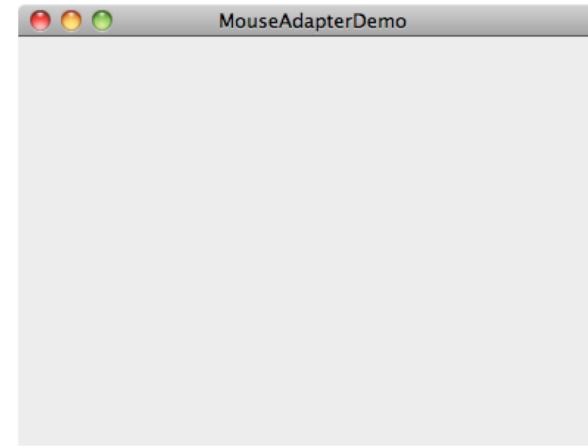
- Zu einigen EventListener-Interfaces gibt es **Adapter-Klassen**.
- Diese implementieren das Interface, enthalten aber nur leere Methoden.
- Beispiel:



- Ist man nur an einer Methode eines EventListeners interessiert, dann ist es einfacher, den entsprechenden Adapter zu erweitern und nicht das komplette Listener-Interface zu implementieren.

# MouseAdapterDemo

```
public class MouseAdapterDemo extends JFrame {  
  
    public MouseAdapterDemo() {  
        this.setTitle("MouseAdapterDemo");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setPreferredSize(new Dimension(400,300));  
        this.addMouseListener(new MyMouseListener());  
        this.pack();  
        this.setVisible(true);  
    }  
  
    private class MyMouseListener extends MouseAdapter {  
        @Override  
        public void mouseClicked(MouseEvent e) {  
            System.out.println("Mouse clicked on x = "  
                + e.getX() + ", y = " + e.getY());  
        }  
    }  
  
    public static void main(String[] args) {  
        new MouseAdapterDemo();  
    }  
}
```



```
Mouse clicked on x = 117, y = 104  
Mouse clicked on x = 280, y = 121  
Mouse clicked on x = 79, y = 240  
Mouse clicked on x = 292, y = 266  
Mouse clicked on x = 63, y = 64
```

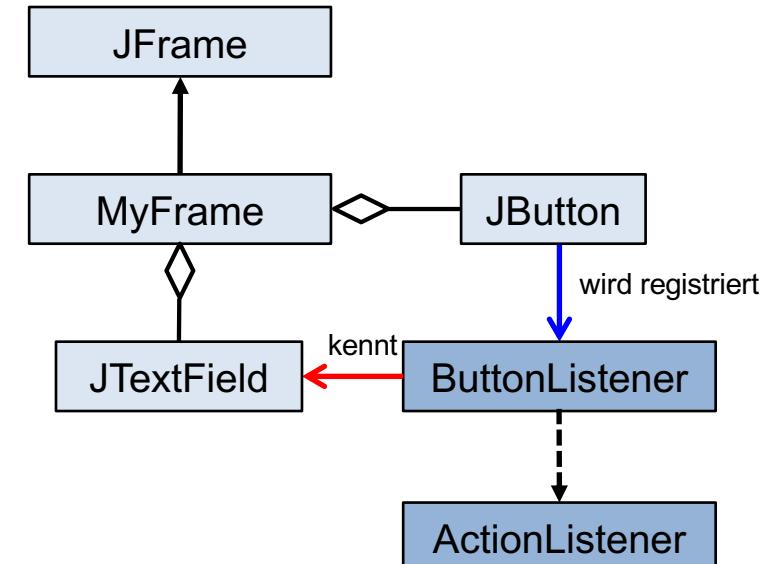
- Beachte, dass MouseAdapterDemo nicht von MouseAdapter abgeleitet werden kann (keine Mehrfachvererbung).
- Daher innere Klasse MyMouseListener

# Separate Beobachterklasse

- Listener-Klasse kann auch separat definiert werden.
- Jedoch müssen dann oft in umständlicher Weise dem Listener Komponenten des Hauptfenster übergeben werden.

```
public class MyFrame extends JFrame {  
    JTextField textField;  
    JButton button;  
  
    public MyFrame() {  
        ...  
        this.add(button)  
        button.addActionListener(new ButtonListener(textField));  
    }  
    ...  
}
```

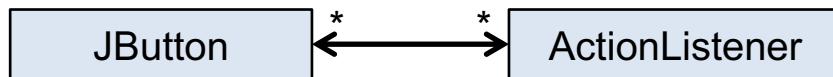
```
public class ButtonListener implements ActionListener {  
    JTextField textField;  
  
    public ButtonListener(JTextfield tf) {textField = tf;}  
  
    public void actionPerformed(ActionEvent e) {  
        // kann auf textField zugreifen: ...  
    }  
}
```



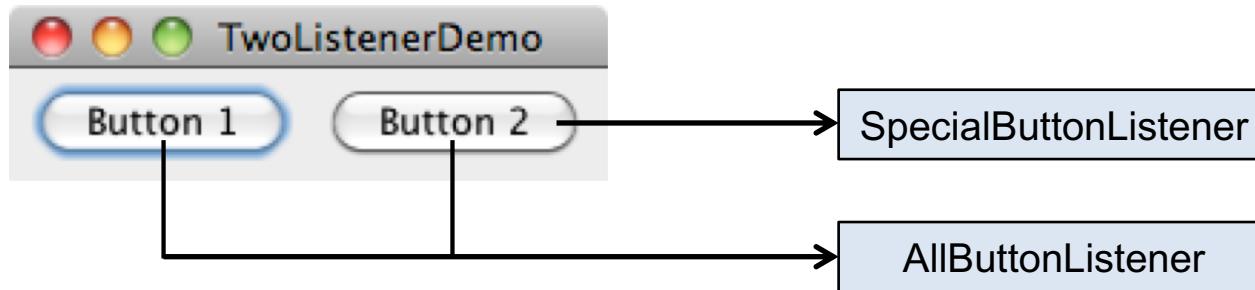
Dem ButtonListener wird  
textField übergeben, so dass  
ein Zugriff möglich wird.

# n:m-Beziehung zwischen Beobachtern und Komponenten

- Eine Komponente kann von mehreren Beobachtern beobachtet werden.
- Ein Beobachter kann unterschiedliche Komponenten beobachten.



- Beispiel:



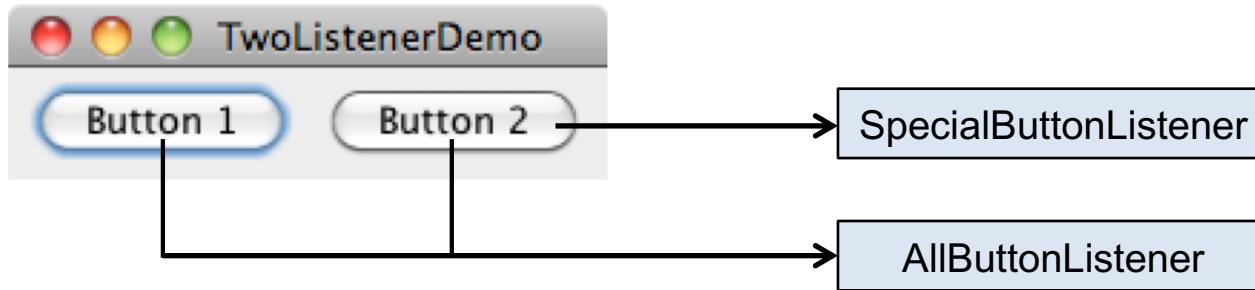
- Implementierung:  
Definiere in Hauptfenster zwei innere Beobachterklassen.

# TwoListenerDemo (1)

```
public class TwoListenerDemo extends JFrame {  
  
    JButton button1;  
    JButton button2;  
  
    public TwoListenerDemo() {  
        this.setTitle("TwoListenerDemo");  
        this.setDefaultCloseOperation(...);  
        this.setLayout(new FlowLayout());  
  
        button1 = new JButton("Button 1");  
        button2 = new JButton("Button 2");  
  
        ActionListener multiButtonListener  
            = new MultiButtonListener();  
        button1.addActionListener(multiButtonListener);  
        button2.addActionListener(multiButtonListener);  
        button2.addActionListener(  
            new SpecialButtonListener());  
  
        this.add(button1);  
        this.add(button2);  
        this.pack();  
        this.setVisible(true);  
    }  
}
```

```
private class MultiButtonListener  
implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        System.out.print("MultiButtonListerner: ");  
        Object source = e.getSource();  
        if (source == button1)  
            System.out.print(button1.getText());  
        else if (source == button2) {  
            System.out.print(button2.getText());  
            System.out.println(" wurde gedrueckt.");  
        }  
    }  
  
    private class SpecialButtonListener  
    implements ActionListener {  
        public void actionPerformed(ActionEvent e) {  
            System.out.println("SpecialButtonListener:"  
                +"Button 2 wurde gedrueckt");  
        }  
    }  
  
    public static void main(String[] args) {  
        new TwoListenerDemo();  
    }  
}
```

# TwoListenerDemo (2)



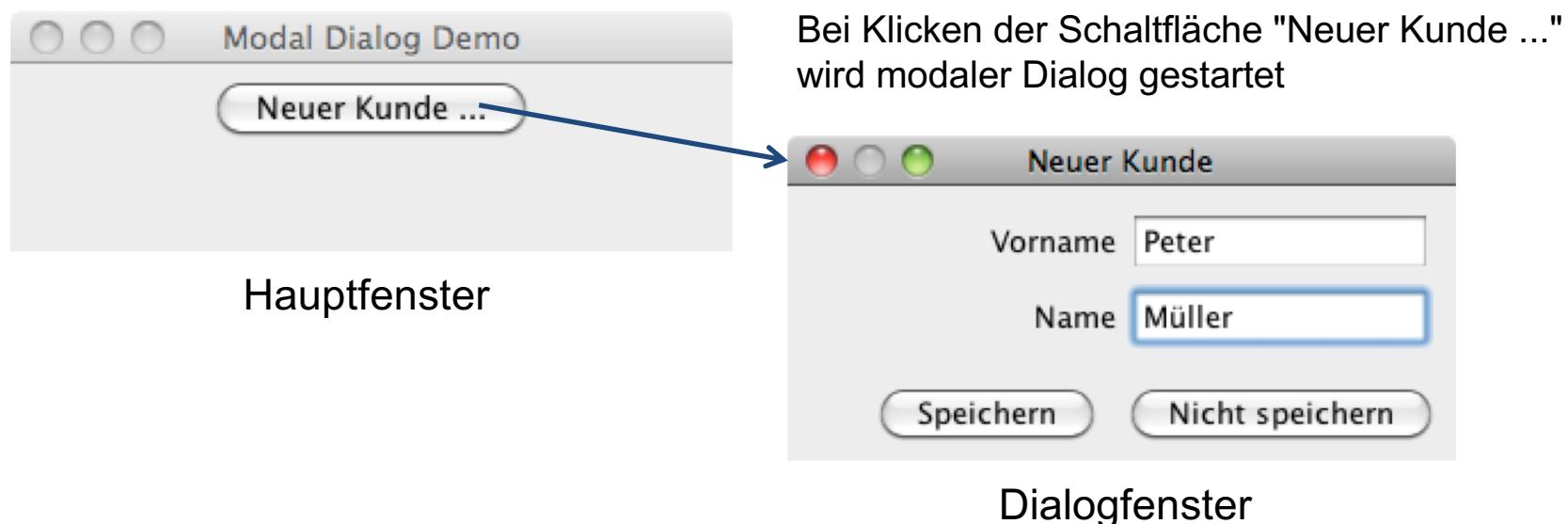
SpecialButtonListener: Button 2 wurde gedrueckt  
MultiButtonListerner: Button 2 wurde gedrueckt.  
SpecialButtonListener: Button 2 wurde gedrueckt  
MultiButtonListerner: Button 2 wurde gedrueckt.  
MultiButtonListerner: Button 1 wurde gedrueckt.

# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen

# Dialogfenster – Überblick (1)

- Dialogfenster werden üblicherweise nur temporär auf dem Bildschirm eingeblendet.
- Sie dienen dazu, bestimmte Eingaben oder Bestätigungen vom Benutzer zu erfragen.
- Dialoge sind oft formularartig aufgebaut und haben im unteren Teil eine Schaltfläche zum Speichern bzw. Abbrechen.
- Man unterscheidet **modale** und **nicht-modale Dialoge**. Bei einem modalen Dialog wird das Fenster, von dem der Dialog aufgerufen wird, für die Interaktion mit dem Benutzer gesperrt.

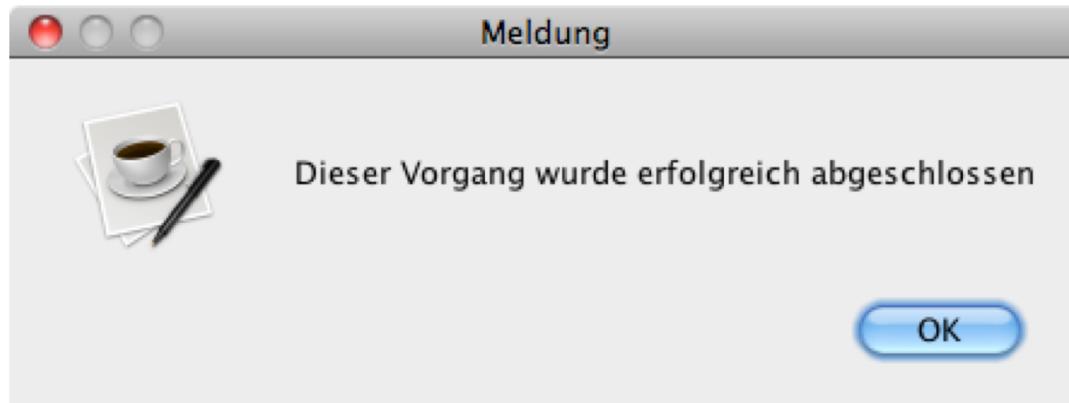


# Dialogfenster – Überblick (2)

---

- Mit Hilfe der Swing Kasse **JOptionPane** können einfache modale Dialog-Fenster erzeugt werden:
  - **Message-Dialog**: Benutzer erhält eine Nachricht, die quittiert werden muss.
  - **Confirm-Dialog**: Benutzer wird um Ja-Nein-Bestätigung gebeten.
  - **Input-Dialog**: erfragt vom Benutzer eine Eingabe
  - **Option-Dialog**: bietet dem Benutzer verschiedene Optionen an, von denen eine ausgewählt werden muss.
- Mit Hilfe der Swing-Klasse **JFileChooser** kann ein einfacher modaler Dialog zum Auswählen von Dateien bzw. Verzeichnissen realisiert werden.
- Andere Dialoge (z.B. formularartig aufgebaute Dialoge) können mit Hilfe der Swing-Klasse **JDialog** erstellt werden. Es sind sowohl **modale** als auch **nicht-modale Dialoge** möglich.

# Message-Dialog



```
JOptionPane.showMessageDialog(  
    myFrame, ——————  
    "Dieser Vorgang wurde erfolgreich abgeschlossen"  
);
```

myFrame ist das Elternfenster  
(z.B. Hauptfenster), von dem  
der Dialog gestartet wird.  
Falls myFrame == null ist,  
wird ein Standard-Fenster  
gewählt.

# Confirm-Dialog



```
int n = JOptionPane.showConfirmDialog(  
    myFrame,  
    "Soll Datei tatsächlich gelöscht werden?",  
    "Confirm Dialog",  
    JOptionPane.YES_NO_OPTION  
);  
  
if (n == JOptionPane.YES_OPTION)  
    System.out.println("Datei wird geloescht ...");  
else if (n == JOptionPane.NO_OPTION)  
    System.out.println("Datei wird nicht geloescht.");  
else  
    System.out.println("Frage wurde nicht beantworted.");
```

Die Antwort des Benutzers wird zurückgeliefert.

Es gibt unterschiedliche Dialogvarianten:  
YES\_NO\_OPTION,  
YES\_NO\_CANCEL\_OPTION,  
OK\_CANCEL\_OPTION

# Input-Dialog mit Eingabefeld



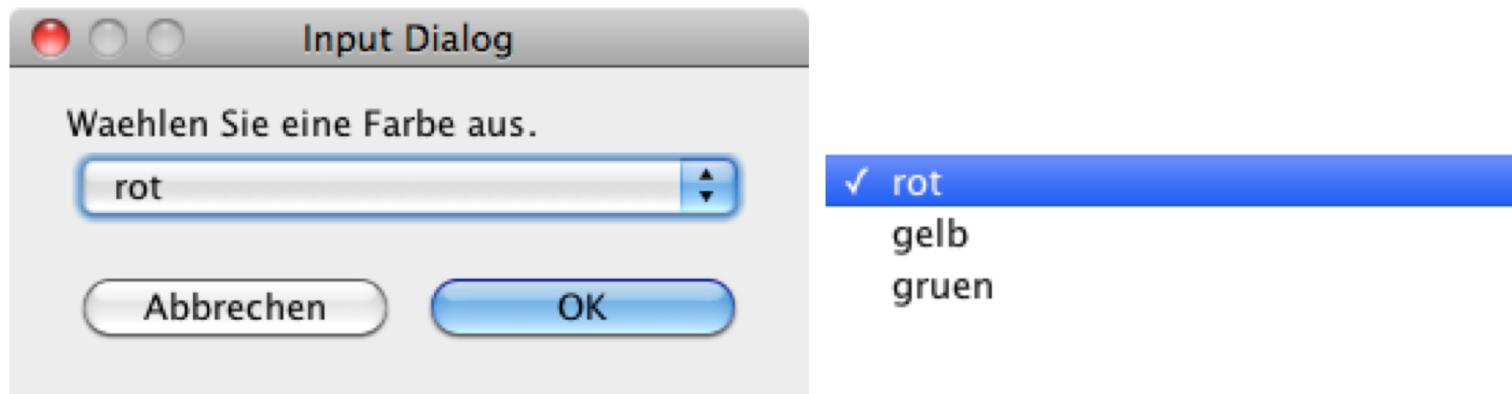
```
String s = JOptionPane.showInputDialog(  
    myFrame,  
    "Geben Sie einen Projektnamen ein:",  
    "JavaApplication"  
);  
  
if ((s != null) && (s.length() > 0))  
    System.out.println("Projekt wird neu angelegt: "+ s);
```

Die Antwort des Benutzers wird zurückgeliefert.

voreingestellter Wert

The code snippet shows the creation of an input dialog using JOptionPane.showInputDialog. The dialog is titled "JavaApplication" and has the message "Geben Sie einen Projektnamen ein:". The user has entered "JavaApplication". The code then checks if the input is not null and has a length greater than zero, printing the result to the console. A callout box points from the explanatory text "Die Antwort des Benutzers wird zurückgeliefert." to the variable "s" in the code. Another callout box points from the explanatory text "voreingestellter Wert" to the title parameter "JavaApplication" in the dialog creation line.

# Input-Dialog mit ComboBox

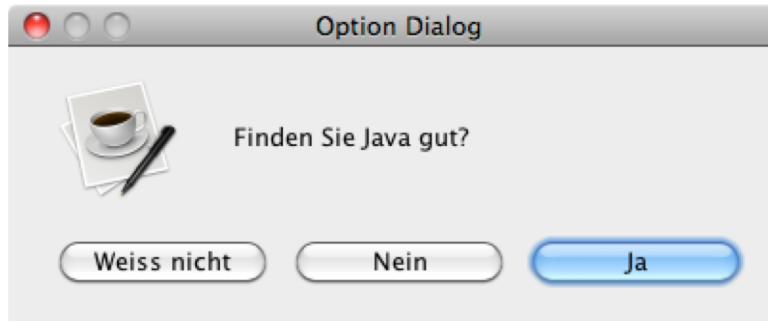


```
String options[] = {"rot", "gelb", "gruen"};  
  
String s = (String) JOptionPane.showInputDialog(  
    myFrame,  
    "Waehlen Sie eine Farbe aus.",  
    "Input Dialog",  
    JOptionPane.PLAIN_MESSAGE,  
    null,  
    options,  
    options[0]  
);  
  
if ((s != null) && (s.length() > 0))  
    System.out.println("Diese Farbe wurde ausgewählt: " + s);
```

Die Auswahlmöglichkeiten  
für die Combo-Box

voreingestellter Wert

# Option-Dialog



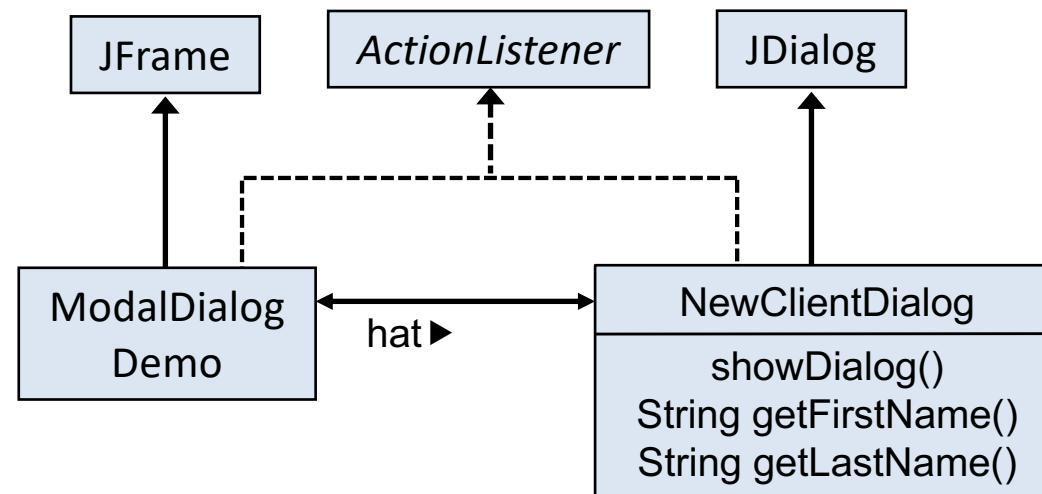
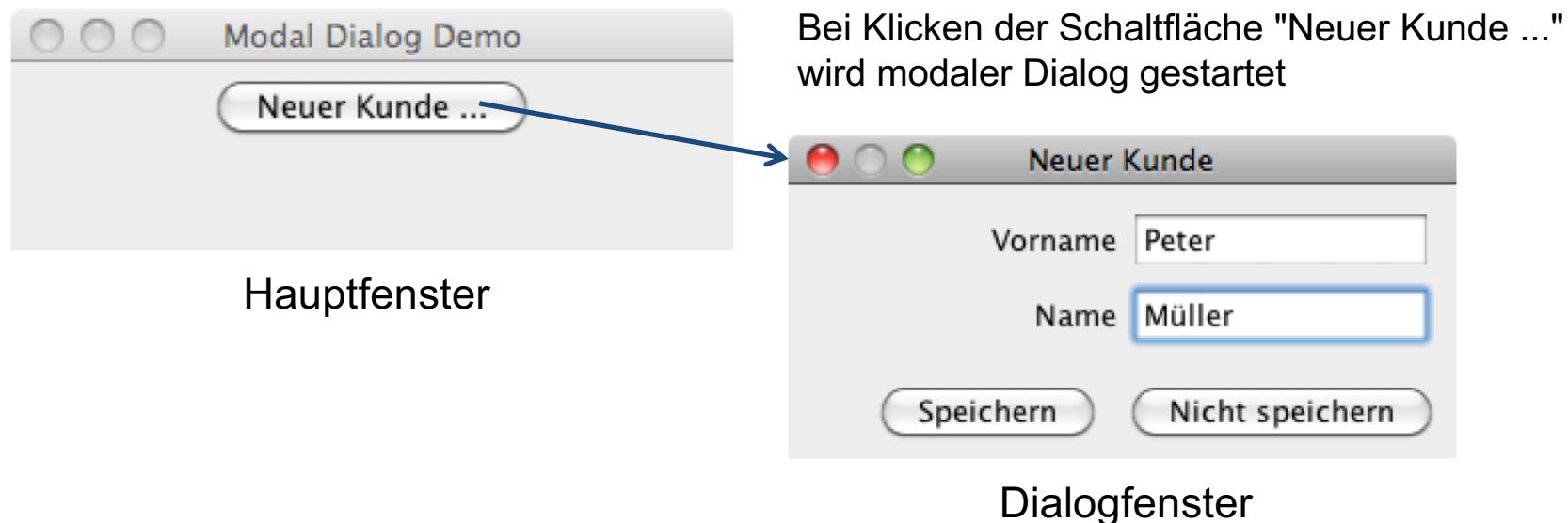
Die Antwort des Benutzers wird zurückgeliefert.

```
String options[] = {"Ja", "Nein", "Weiss nicht"};  
  
int n = JOptionPane.showOptionDialog(  
    this,  
    "Finden Sie Java gut?",  
    "Option Dialog",  
    JOptionPane.DEFAULT_OPTION,  
    JOptionPane.QUESTION_MESSAGE,  
    null,  
    options,  
    options[0]);  
  
if (n == 0)  
    System.out.println("Schoen, dass Sie Java gut finden!");  
else if (n == 1)  
    System.out.println("Schade!");  
else if (n == 2)  
    System.out.println("Beantworten Sie die Frage in ein paar  
                      Monaten wieder.");  
else  
    System.out.println("Keine Antwort.");
```

Optionen

voreingestellter Wert

# Modaler Dialog mit JDialog (1)

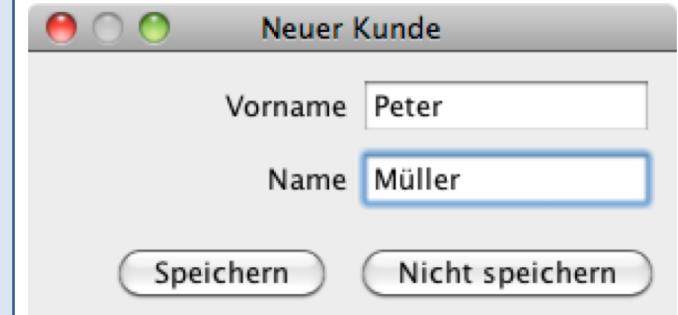


# Modaler Dialog mit JDialog (2)

```
public class ModalDialogDemo extends JFrame implements ActionListener {  
    JButton button;  
    NewClientDialog newClientDialog;  
  
    public ModalDialogDemo() {  
        setTitle("Modal Dialog Demo");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new FlowLayout()); setSize(300,100);  
  
        button = new JButton("Neuer Kunde ...");  
        button.addActionListener(this);  
        add(button);  
  
        newClientDialog = new NewClientDialog(this);  
        setVisible(true);  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        newClientDialog.showDialog();  
        System.out.println("Vorname: " + newClientDialog.getFirstName());  
        System.out.println("Name: " + newClientDialog.getLastName());  
    }  
  
    public static void main(String[] args) {new ModalDialogDemo();}  
}
```



Hauptfenster



Dialogfenster

Dialogfenster  
erzeugen

Dialogfenster  
anzeigen.

Ergebnis des Dialogs  
verarbeiten

# Modaler Dialog mit JDialog (3)

```
public class NewClientDialog extends JDialog implements ActionListener {  
  
    JTextField lastNameField = new JTextField(10);  
    JTextField firstNameField = new JTextField(10);  
    JButton saveButton = new JButton("Speichern");  
    JButton quitButton = new JButton("Nicht speichern");  
    String lastName = "";  
    String firstName = "";  
  
    public NewClientDialog(JFrame f) {  
        super(f, "Neuer Kunde", true);  
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
        saveButton.addActionListener(this);  
        quitButton.addActionListener(this);  
  
        JPanel panelN = new JPanel(new GridLayout(0,2,5,5));  
        panelN.add(new JLabel("Vorname",JLabel.RIGHT));  
        panelN.add(firstNameField);  
        panelN.add(new JLabel("Name",JLabel.RIGHT));  
        panelN.add(lastNameField);  
        panelN.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));  
        add(panelN,BorderLayout.CENTER);  
  
        JPanel panelS = new JPanel(new FlowLayout(FlowLayout.RIGHT));  
        panelS.add(saveButton);  
        panelS.add(quitButton);  
        add(panelS,BorderLayout.SOUTH);  
        pack();  
    }  
}
```

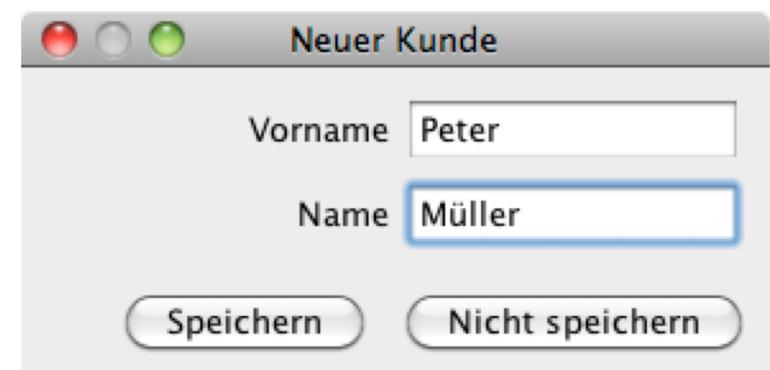
Dialogfenster wird von JDialog vererbt.

Zum Speichern des Dialogergebnis

Dialogart ist modal.

Beobachter registrieren

Dialogformular zusammenbauen



# Modaler Dialog mit JDialog (4)

```
public void showDialog() {  
    firstNameField.setText("");  
    lastNameField.setText("");  
    lastName = "";  
    firstName = "";  
    setVisible(true);  
}  
  
public void actionPerformed(ActionEvent e) {  
    Object source = e.getSource();  
    if (source == saveButton) {  
        firstName = firstNameField.getText();  
        lastName = lastNameField.getText();  
    }  
    else if (source == quitButton) {}  
    setVisible(false);  
}  
  
public String getLastName() {return lastName; }  
  
public String getFirstName() {return firstName; }  
}
```

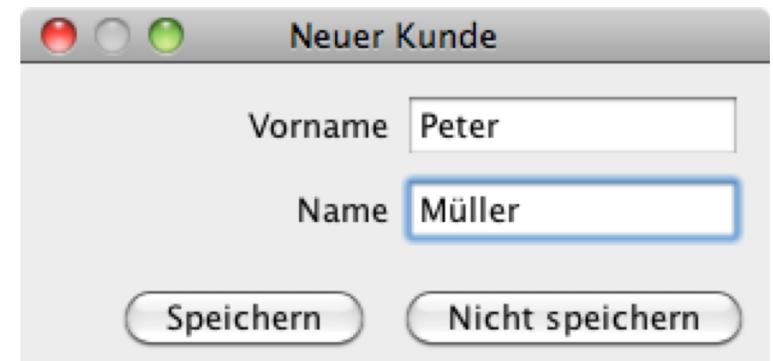
Dialogfenster anzeigen

Beobachter

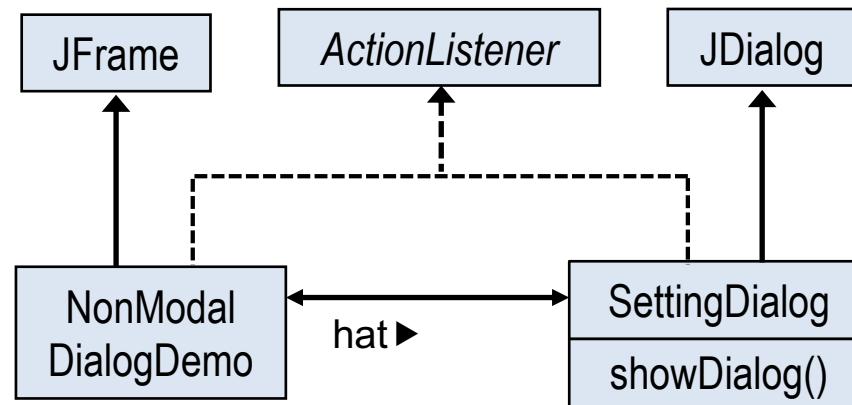
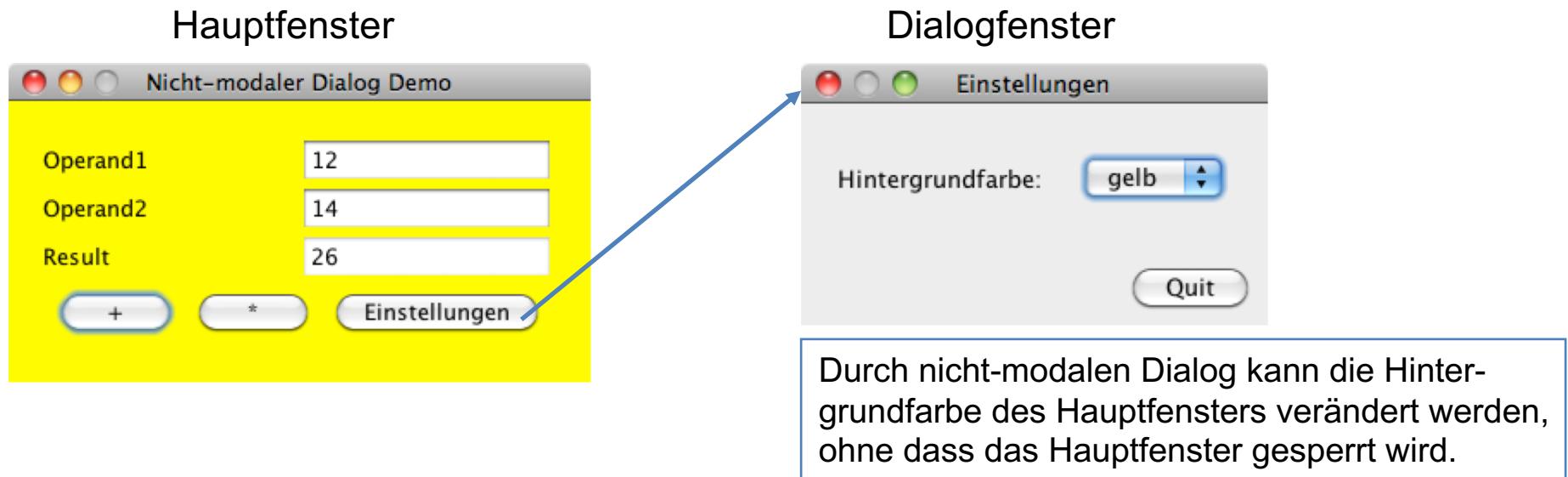
Dialogergebnis abspeichern

Dialog wird verlassen.

Methoden zum Abholen des Dialogergebnis



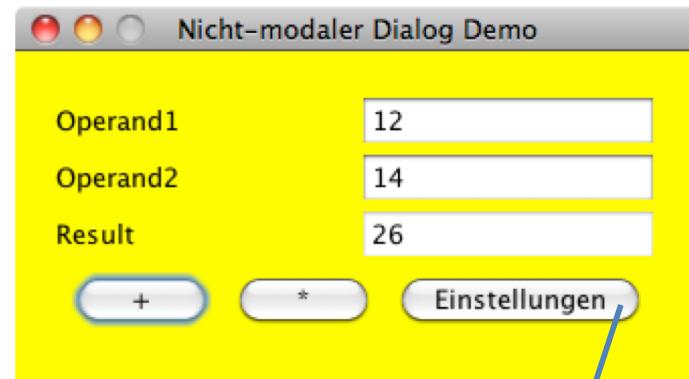
# Nicht-modaler Dialog mit JDialog (1)



# Nicht-modaler Dialog mit JDialog (2)

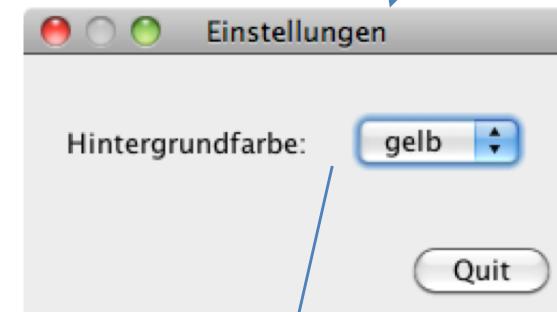
```
public class NonModalDialogDemo extends JFrame  
implements ActionListener {  
    // ...  
    JButton settingsButton = new JButton("Einstellungen");  
    SettingDialog settingDialog;  
  
    public NonModalDialogDemo() {  
        // ...  
        settingDialog = new SettingDialog(this);  
        settingsButton.addActionListener(this);  
        add(settingsButton);  
        // ...  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        Object source = e.getSource();  
        if (source == settingsButton)  
            settingDialog.showDialog();  
        else {  
            // ...  
        }  
        // ...  
    }  
}
```

Hauptfenster wie auf Seite 12-28 bis 12-29. Zusätzlich ein Dialogfenster.



Dialogfenster erzeugen

Dialogfenster



Dialogfenster anzeigen

Durch nicht-modalen Dialog kann die Hintergrundfarbe des Hauptfensters verändert werden, ohne dass das Hauptfenster gesperrt wird.

# Nicht-modaler Dialog mit JDialog (3)

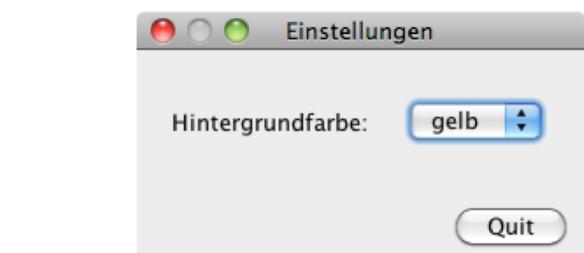
```
public class SettingDialog extends JDialog implements ActionListener {  
    JComboBox colorComboBox;  
    Color[ ] color;  
    JButton quitButton = new JButton("Quit");  
    JFrame parentFrame;  
  
    public SettingDialog(JFrame f) {  
        super(f, "Einstellungen", false);  
        parentFrame = f;  
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
  
        String [ ] colorStrings = { "rot", "grün", "gelb", "grau" };  
        color = new Color [ ] {  
            Color.RED, Color.GREEN, Color.YELLOW, Color.LIGHT_GRAY };  
        colorComboBox = new JComboBox(colorStrings);  
        colorComboBox.setSelectedIndex(3);  
  
        JPanel panelN = new JPanel(new BorderLayout());  
        panelN.add(new JLabel("Hintergrundfarbe:"), BorderLayout.LINE_START);  
        panelN.add(colorComboBox, BorderLayout.LINE_END);  
        panelN.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));  
        add(panelN,BorderLayout.CENTER);  
        JPanel panelS = new JPanel(new FlowLayout(FlowLayout.RIGHT));  
        panelS.add(quitButton);  
        add(panelS,BorderLayout.SOUTH);  
        pack();  
  
        colorComboBox.addActionListener(this);  
        quitButton.addActionListener(this);  
    }  
}
```

Dialogfenster wird von JDialog vererbt.

Dialogart ist nicht-modal.

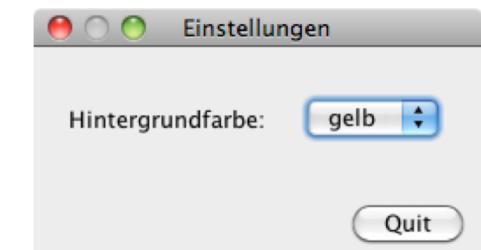
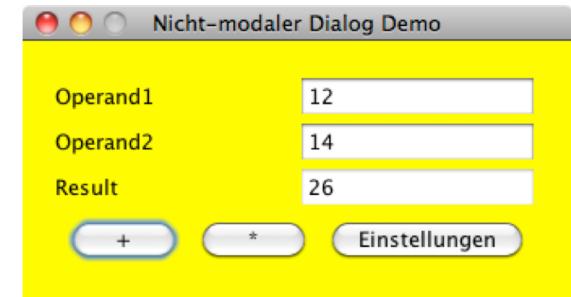
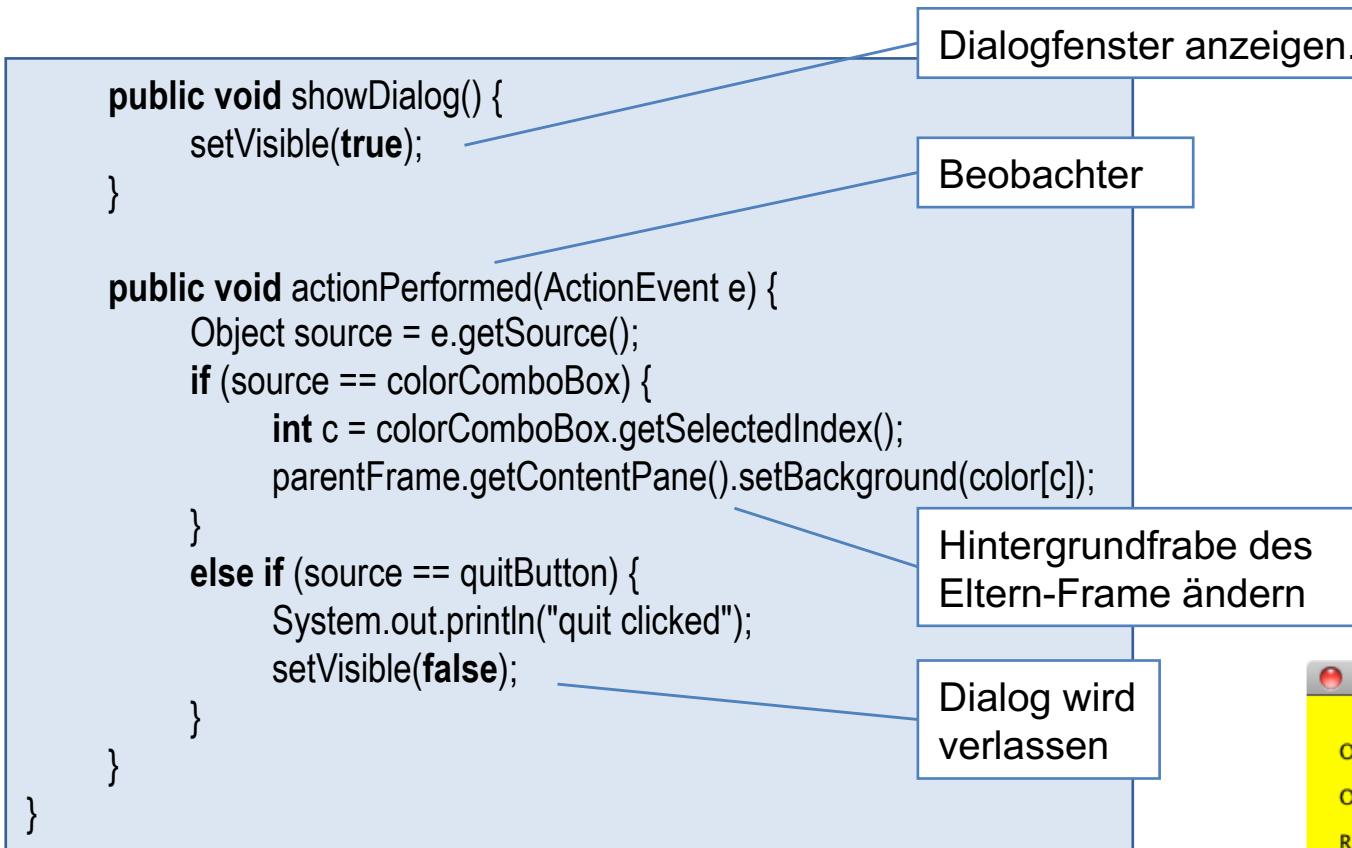
Combo-Box für Farbauswahl

Dialogfenster zusammenabauen



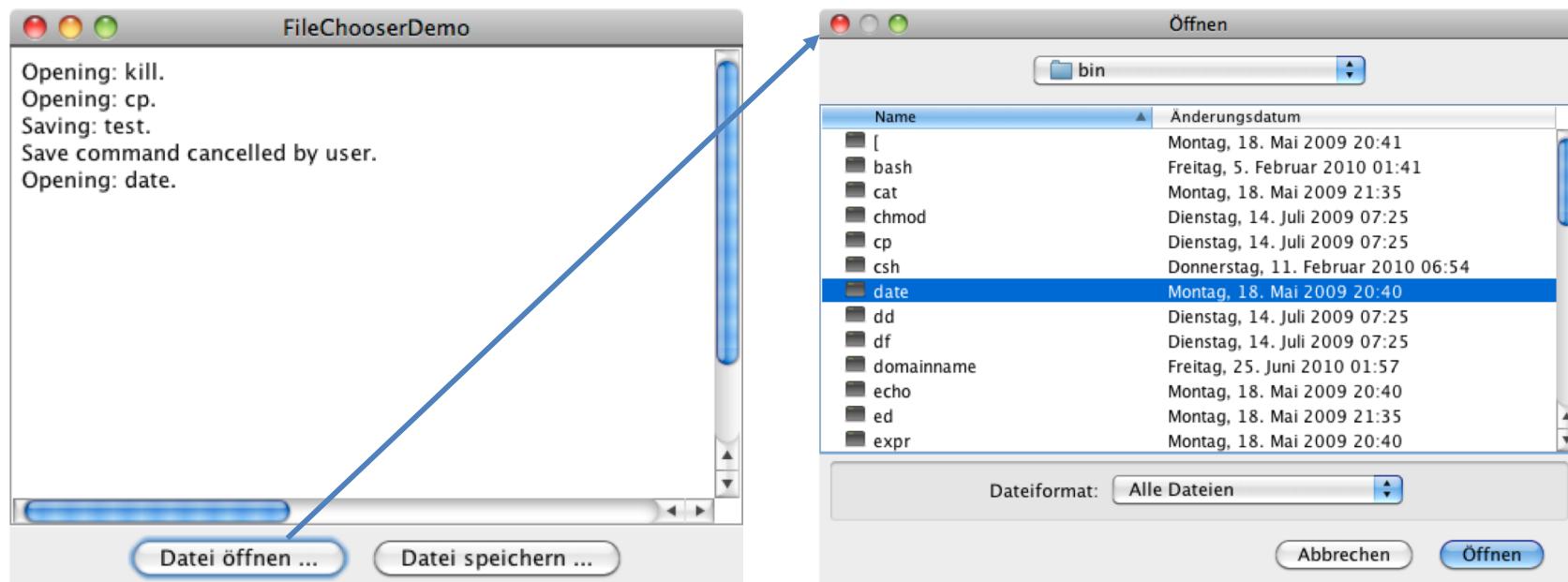
Beobachter registrieren

# Nicht-modaler Dialog mit JDialog (4)



# Dateiauswahldialog mit JFileChooser (1)

- **JFileChooser** bietet einen modalen Dialog an, um in gewohnter Weise über das Dateisystem zu navigieren und eine Datei oder Verzeichnis auszuwählen.
- Zusätzlich kann ein Filter für die Dateiauswahl definiert werden ([javax.swing.filechooser.FileFilter](#)).  
Beispiel: Öffnen von Dateien, die mit \*.jpg enden.



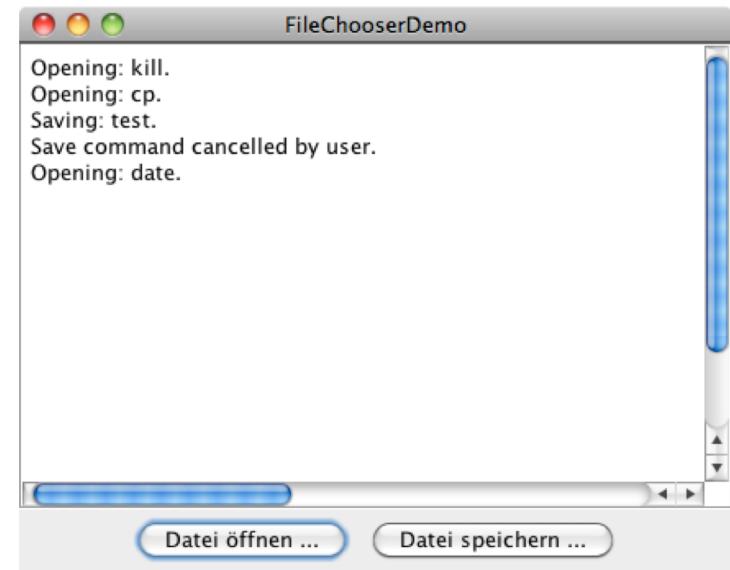
Hauptfenster mit Textfläche zum  
Mitprotokollieren der Operationen  
und der ausgewählten Dateien

Dateiauswahldialog zum  
Öffnen einer Datei.

# Dateiauswahldialog mit JFileChooser (2)

```
public class FileChooserDemo extends JFrame  
implements ActionListener {  
  
    JButton openButton = new JButton("Datei öffnen ...");  
    JButton saveButton = new JButton("Datei speichern ...");  
    JTextArea text;  
    JFileChooser fc;  
  
    FileChooserDemo() {  
        setTitle("FileChooserDemo");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        fc = new JFileChooser();  
        fc.setFileSelectionMode(JFileChooser.FILES_ONLY);  
  
        text = new JTextArea(20,80);  
        text.setMargin(new Insets(5,5,5,5));  
        text.setEditable(true);  
        JScrollPane logScrollPane = new JScrollPane(text);  
  
        JPanel buttonPanel = new JPanel();  
        buttonPanel.add(openButton);  
        buttonPanel.add(saveButton);  
        openButton.addActionListener(this);  
        saveButton.addActionListener(this);  
        add(buttonPanel, BorderLayout.SOUTH);  
        add(logScrollPane, BorderLayout.CENTER);  
        pack();  
        setVisible(true);  
    }  
}
```

Hauptfenster

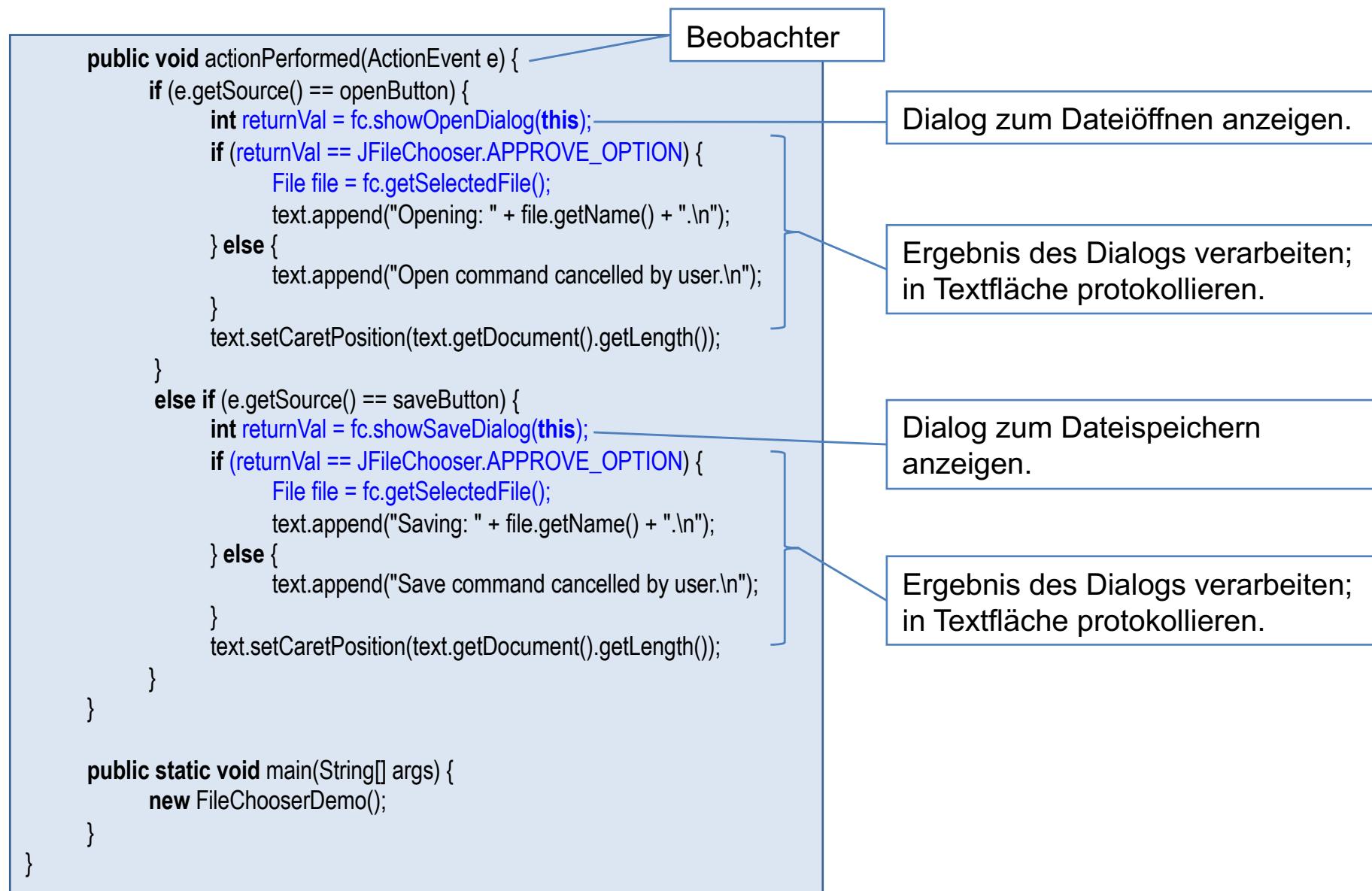


Dateiauswahldialog definieren.

Scrollbare Textfläche zum Protokollieren.

Hauptfenster zusammenauen und Beobachter registrieren.

# Dateiauswahldialog mit JFileChooser (3)



# Kapitel 12: Grafische Benutzeroberflächen mit Swing

- Einleitung
- Hauptfenster und Container
- Swing-Komponenten
- Layout-Manager
- Ereignisverarbeitung
- Dialogfenster
- Zeichnen

# paint-Methode

---

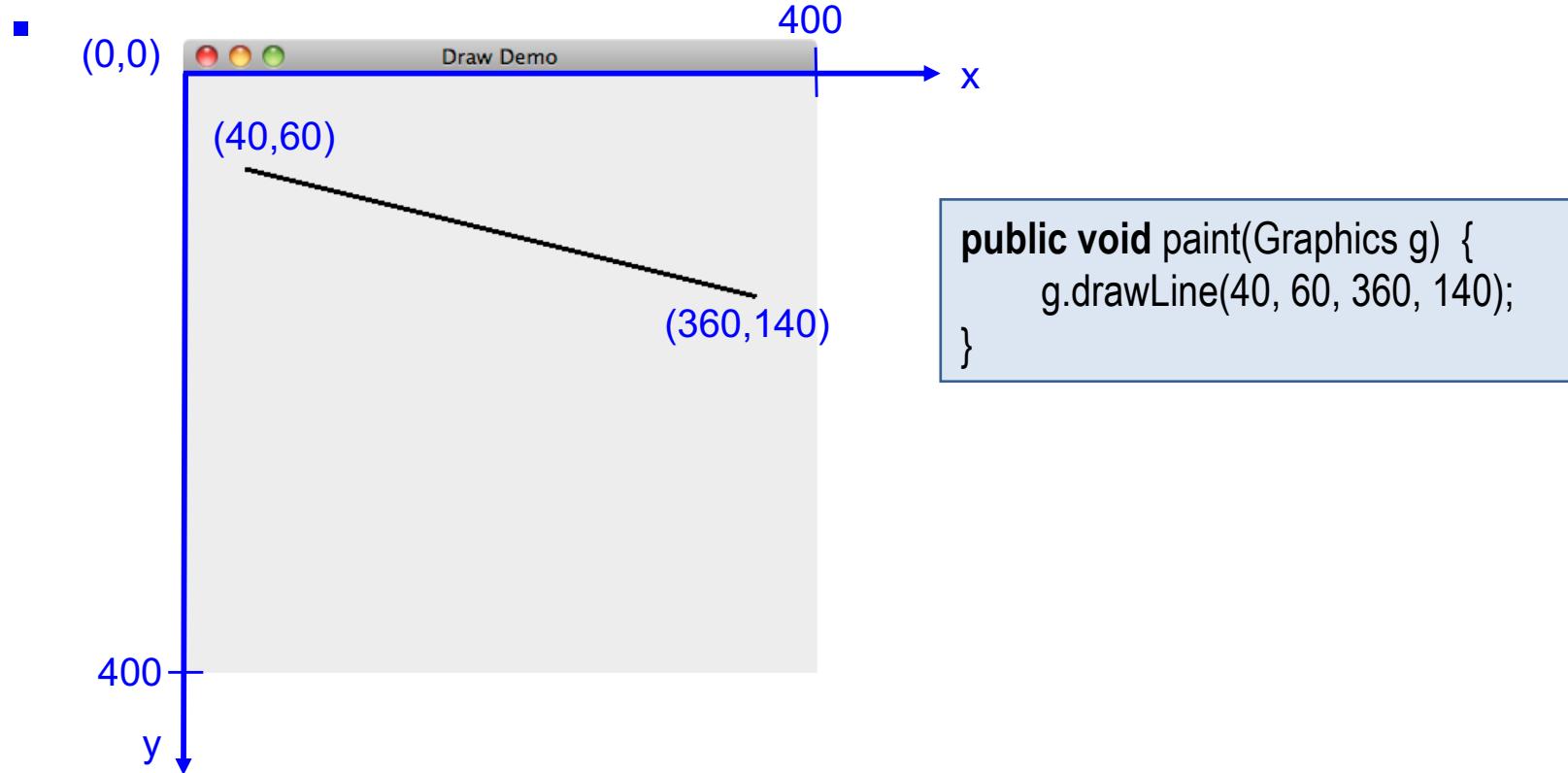
- In Java kann in jede Komponente (z.B. JPanel) gezeichnet werden.
- Dazu muss für die Komponente die **paint-Methode** überschrieben werden.

```
@Override  
public void paint(Graphics g) {  
    g.drawLine(10, 10, 50, 20);  
    // ...  
}
```

- Die paint-Methode wird vom Fenster-System aufgerufen, sobald die betreffende Komponente neu gezeichnet werden muss (z.B. bei Öffnen eines Fensters)
- Durch Aufruf von **repaint()** kann das Neuzeichnen erzwungen werden.

# Klasse Graphics

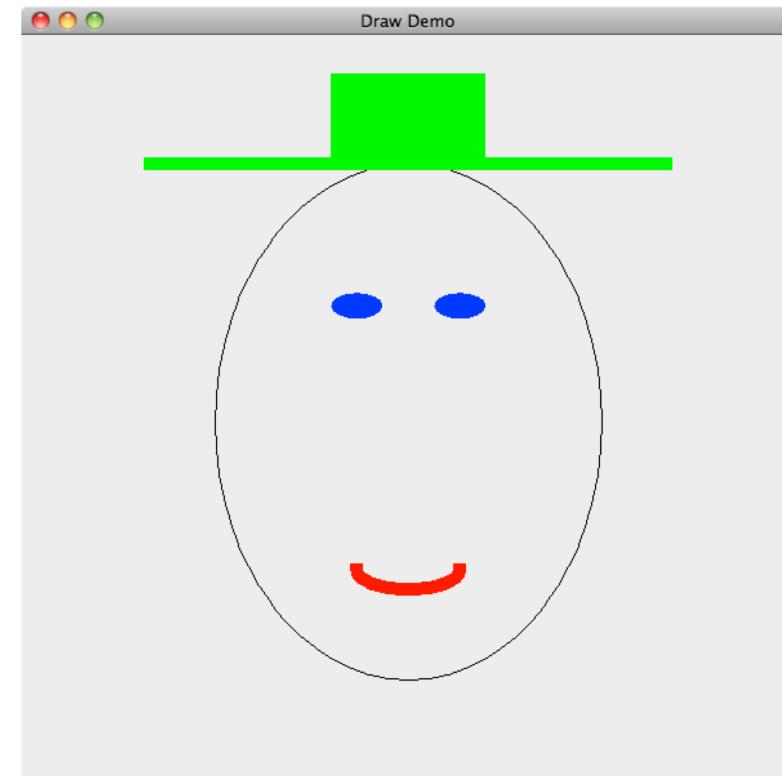
- **Graphics** stellt einen sogenannten Grafik-Kontext zur Verfügung, der zum Zeichnen benötigt wird.
- Das Koordinatensystem der Zeichenfläche hat seinen Ursprung (0,0) in der linken oberen Ecke.



# Beispiel mit Graphics2D

```
public class DrawDemo extends JFrame {  
    public DrawDemo() {  
        setTitle("Draw Demo");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setSize(600,600);  
        this.add(new DrawPanel());  
        this.setVisible(true);  
    }  
    public static void main(String[] args) {  
        new DrawDemo();  
    }  
}
```

```
public class DrawPanel extends JPanel {  
    public void paint(Graphics g) {  
        Graphics2D g2 = (Graphics2D) g;  
        g2.drawOval(150, 100, 300, 400);  
        g2.setColor(Color.red);  
        g2.setStroke(new BasicStroke(10));  
        g2.drawArc(280, 400, 50, 20, 0, -180);  
        g2.setColor(Color.blue);  
        g2.fillOval(240, 200, 40, 20);  
        g2.fillOval(320, 200, 40, 20);  
        g2.setColor(Color.green);  
        g2.drawLine(100, 100, 500, 100);  
        g2.fillRect(240, 30, 120, 70);  
    }  
}
```



Um mehr Möglichkeiten (z.B. Stiftbreite setzen) zu erhalten, sollte der Grafikkontext in Graphics2D konvertiert werden.

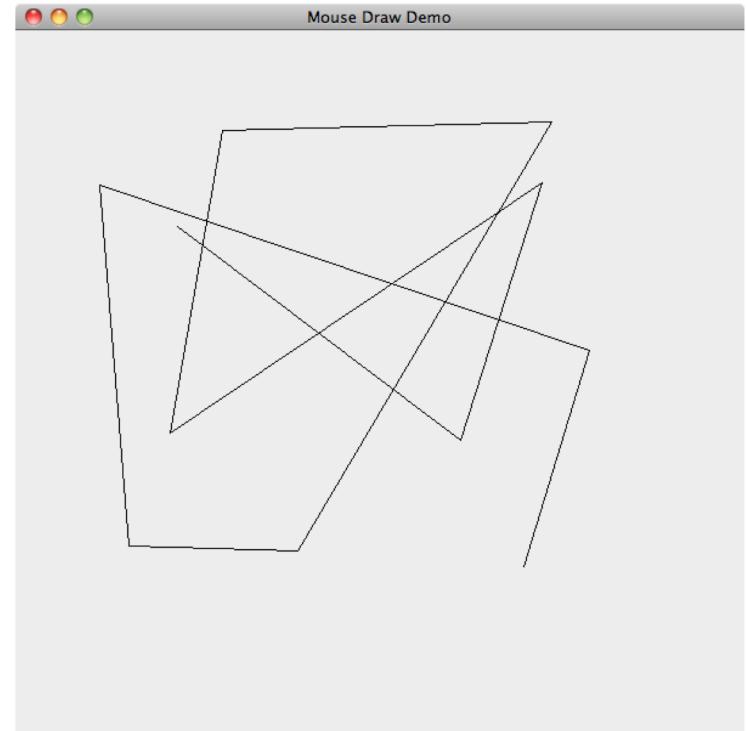
Stiftbreite setzen

# Beispiel mit MouseAdapter und repaint

```
public class MouseDrawDemo extends JFrame {  
    public MouseDrawDemo() {  
        setTitle("Mouse Draw Demo");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setSize(600,600);  
        add(new MouseDrawPanel());  
        setVisible(true);  
    }  
    public static void main(String[] args) {  
        new MouseDrawDemo();  
    }  
}
```

```
public class MouseDrawPanel extends JPanel {  
    private int[] x = new int[100];  
    private int[] y = new int[100];  
    int n = 0;  
  
    public MouseDrawPanel() { addMouseListener(new MyMouseListener()); }  
  
    public void paint(Graphics g) { g.drawPolyline(x, y, n); }  
  
    private class MyMouseListener extends MouseAdapter {  
        public void mouseClicked(MouseEvent e) {  
            if (n >= 100) return;  
            x[n] = e.getX(); y[n] = e.getY();  
            n++;  
            repaint();  
        }  
    }  
}
```

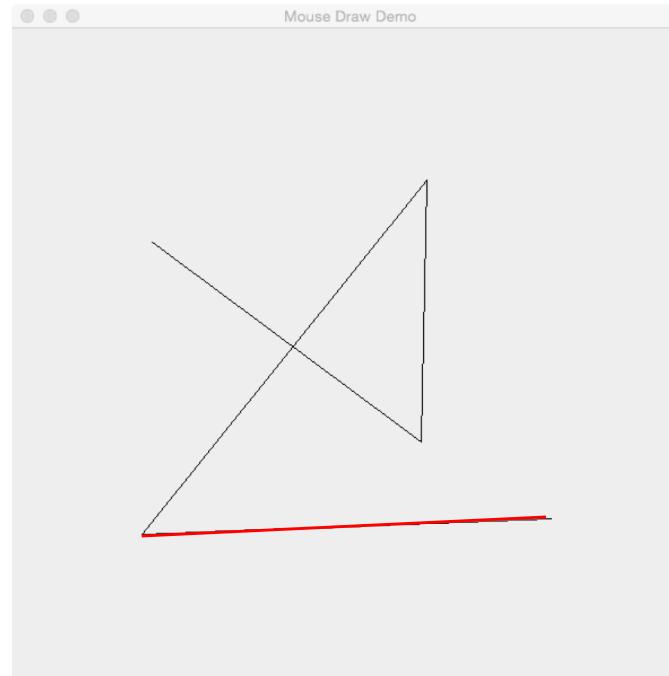
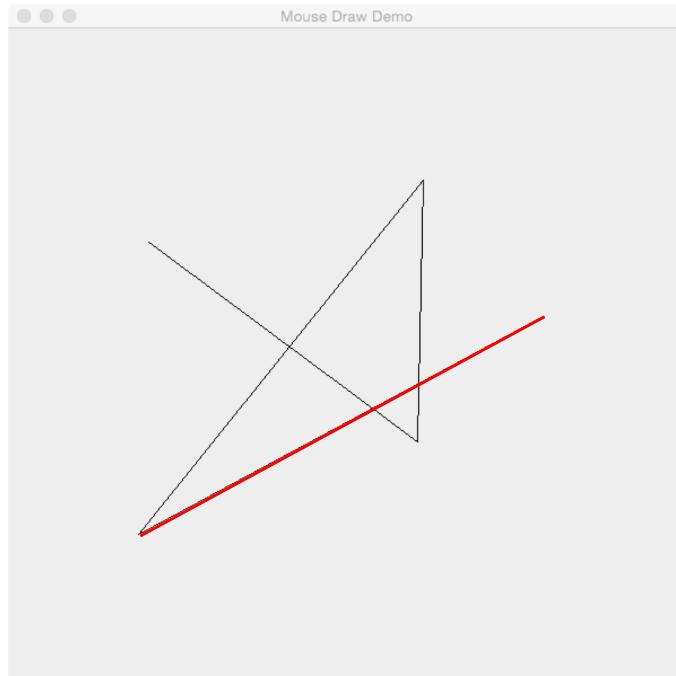
- Einfaches Zeichnen eines Polygonzugs



# Beispiel erweitert (1)

---

- Die **neue Kante** soll durch Mausbewegungen animiert werden, bevor sie durch einen Maus-Click endgültig fixiert wird.
- Durch Doppel-Click soll die Polygon-Zeichnung beendet werden.



# Beispiel erweitert (2)

```
class MouseDrawPanel extends JPanel {  
    private int[ ] x = new int[100];  
    private int[ ] y = new int[100];  
    int n = 0;  
  
    public MouseDrawPanel() {  
        MouseAdapter myMouseListener = new MyMouseListener();  
        addMouseListener(myMouseListener);  
        addMouseMotionListener(myMouseListener);  
    }  
  
    public void paint(Graphics g) { g.drawPolyline(x, y, n+1); }  
  
    private class MyMouseListener extends MouseAdapter {  
        @Override  
        public void mouseClicked(MouseEvent e) {  
            n++;  
            if (e.getClickCount() == 2)  
                System.exit(0);  
        }  
  
        @Override  
        public void mouseMoved(MouseEvent e) {  
            x[n] = e.getX();  
            y[n] = e.getY();  
            repaint();  
        }  
    }  
}
```

- Sowohl Maus-Clicks als auch Maus-Bewegungen müssen abgefangen werden.
- Es wird daher ein **MouseListener** und ein **MouseMotionListener** benötigt.
- Beides leistet der **MouseAdapter** (siehe auch Seite 12-58), wobei die Methoden **mouseClicked** und **mouseMoved** geeignet überschrieben werden müssen.

(x[0],y[0]), ..., (x[n-1],y[n-1]) sind der bereits fixierte Teil des Polygonzugs.  
Die letzte Ecke (x[n],y[n]) ist noch nicht fixiert.