Grafische Benutzeroberflächen mit Swing

KW 30, Zentralübung Informatik II

2006-07-24

Klausur: Stichpunkte

1. Auswertung
   - `System.out.println("abc" instanceof String);`
     $\Rightarrow$ Ergebnis von `instanceof` ist ein `boolean`, der von `println()` problemlos ausgegeben werden kann.
   - `Object.equals` ist per == definiert.

2. Observer-Pattern: Sinn verstehen.

3. OCL: Klasseninvarianten verstehen, Kontext wurde oft vergessen.

4. UML: Aggregation (stark versus schwach).

Beobachtung: Noch mehr selbst machen. Seitdem die Übungsblätter nicht mehr Pflicht sind, hat die Qualität der Klausurabgaben stark abgenommen.

Schritte zum Erstellen eines Swing-GUIs

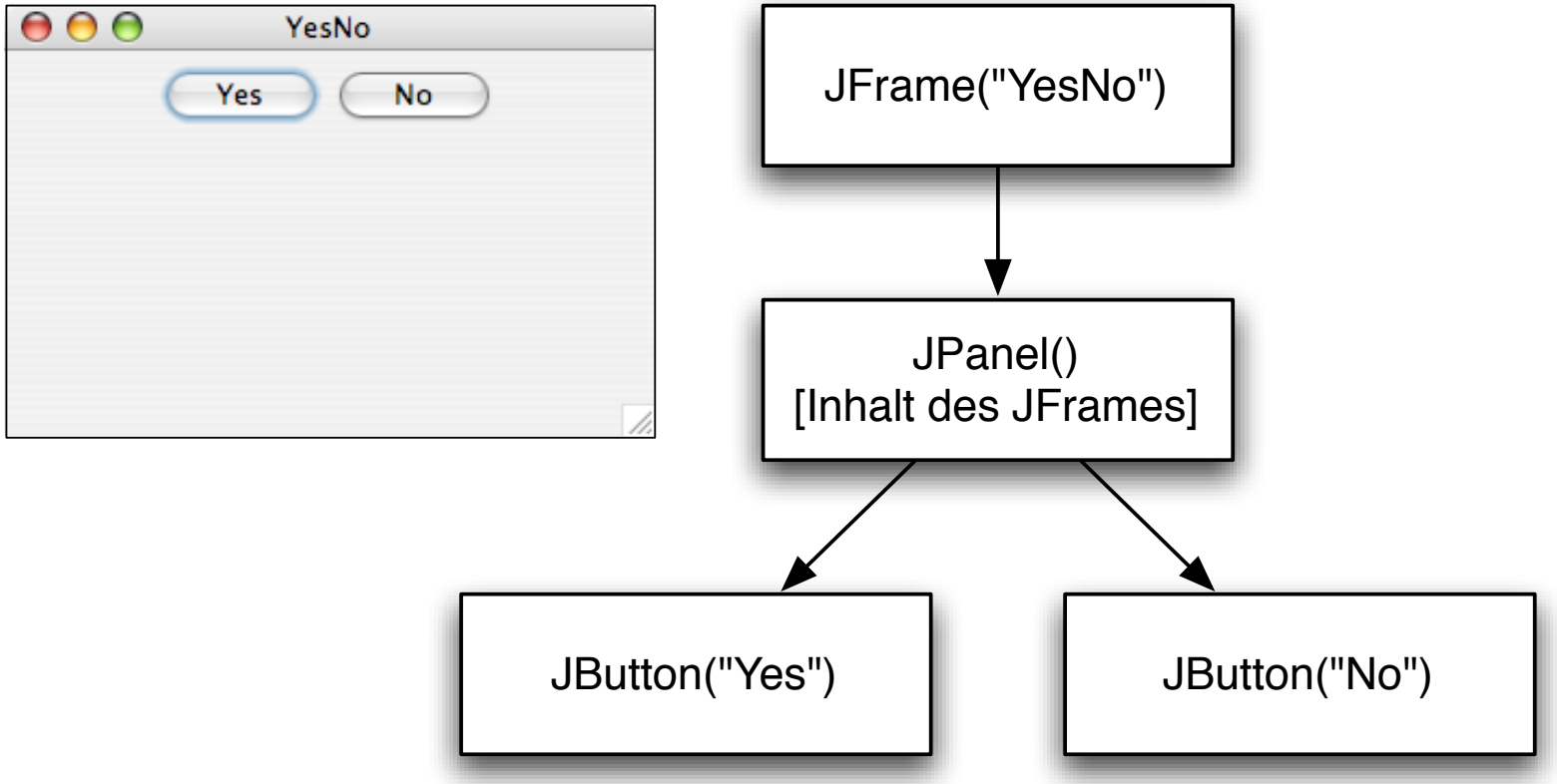
(GUI = Graphical User Interface)

- Elemente (Buttons, Bäume, Textfelder etc.) schachteln.
- Elemente präzise anordnen.
- Auf *Ereignisse* (Events) reagieren: Mausklicks, Tastendrücke etc.

AWT versus Swing

- AWT: Abstract Window Toolkit
  - Verwendet die nativen GUI-Elemente des unterliegen Betriebssystems.
- Swing
  - Setzt auf AWT auf, zeichnet aber alle Elemente selbst.
  - Die Klassennamen beginnen mit einem „J“ (JComponent, JButton, JFrame etc.).

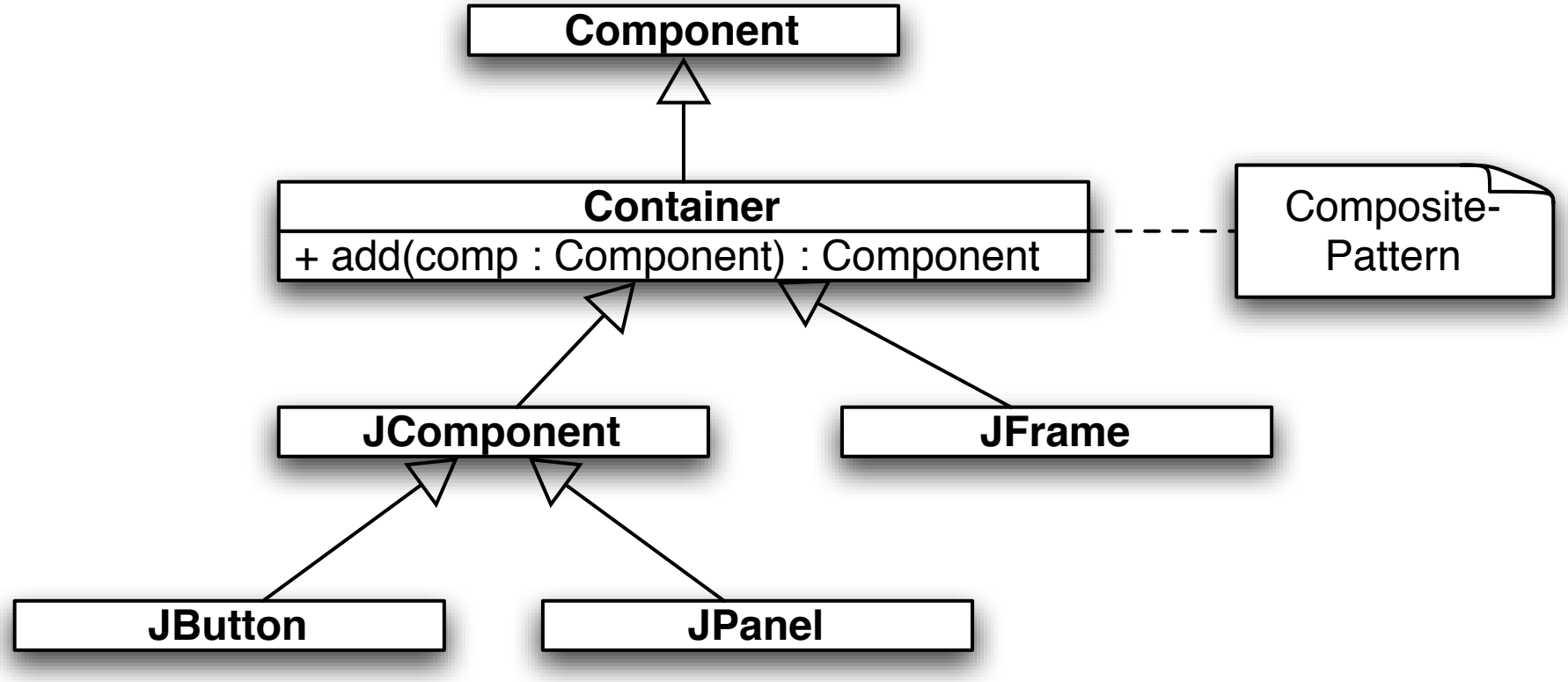
Components: Geschachtelte GUI-Elemente



Components: Geschachtelte GUI-Elemente

- GUI-Elemente heissen in Swing *Components* (auch: Widgets).
- GUI-Elemente sind geschachtelt
- Es ergibt sich eine *Containment Hierarchy* (Baum, Relation *contains*).
- Welches Entwurfsmuster? Composite-Pattern!

Components: Klassenhierarchie (gekürzt)



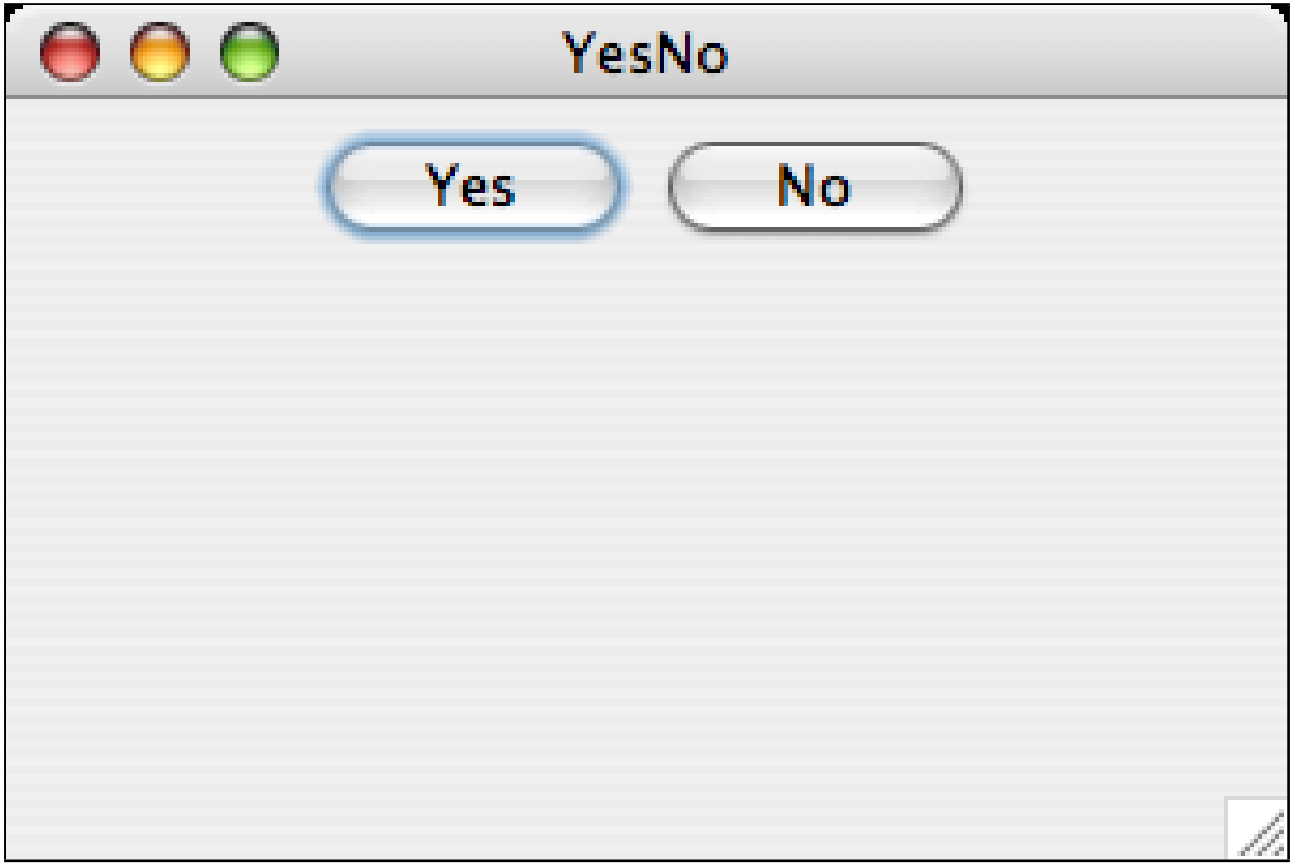
Components: Beispiel (1/2)

```
public class YesNo {
    public static JComponent createContent() {
        ...
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.add(createContent());
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
}
```

Components: Beispiel (2/2)

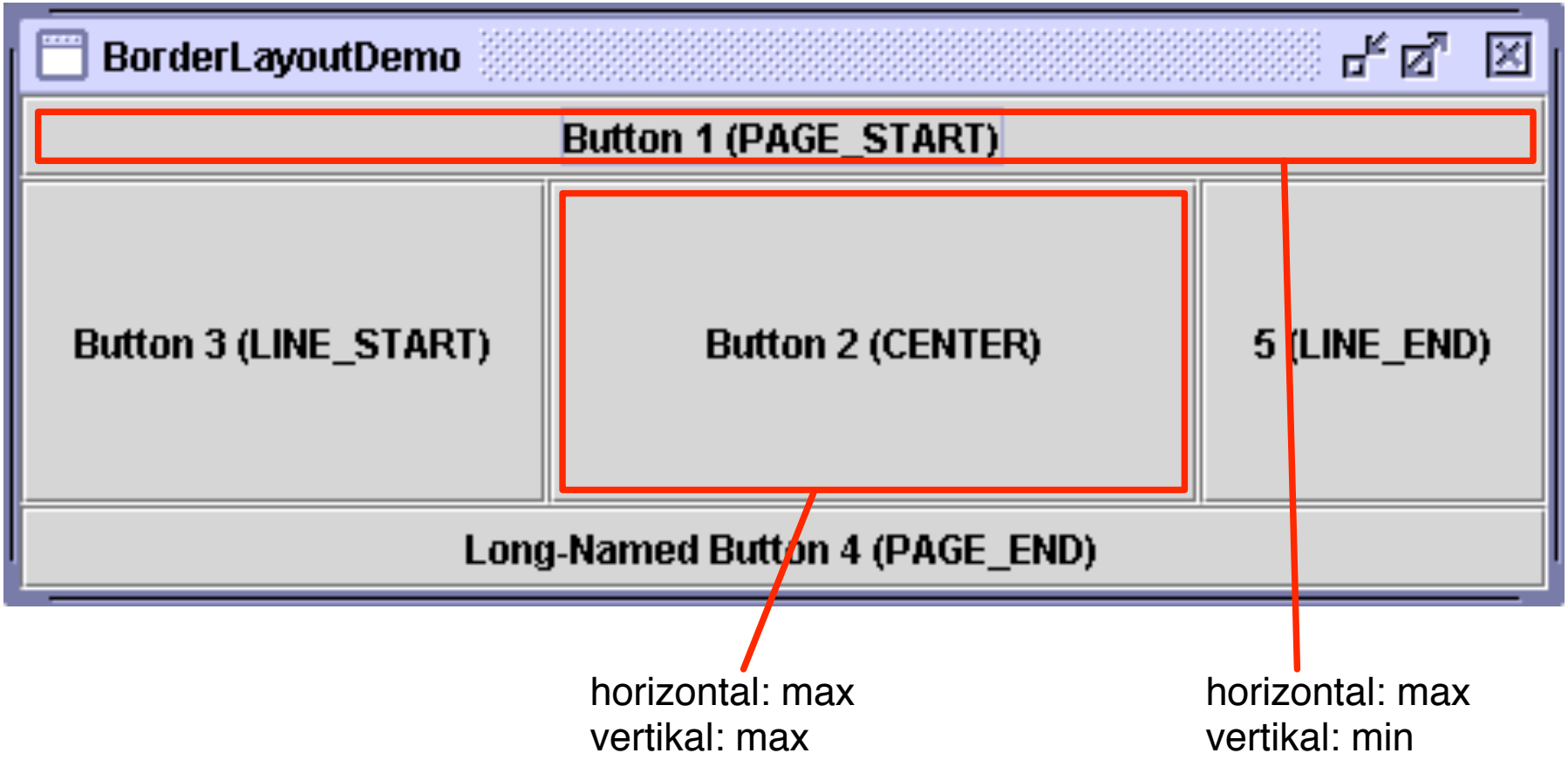
```
public static JComponent createContent() {
    JPanel contentPanel = new JPanel();
    contentPanel.add(new JButton("Yes"));
    contentPanel.add(new JButton("No"));
    return contentPanel;
}
```

Component-Beispiel: So sieht es aus



Layout: Components anordnen

- Wie macht man das plattform- und auflösungsunabhängig?
- Lösungs-Idee:
  - Komponenten haben eine minimale Größe.
  - Man gibt, als Gitter, an, wie und ob Komponenten vergrößert werden sollen, wenn zusätzlicher Platz ist.


BorderLayoutDemo
Button 1 (PAGE_START)
Button 3 (LINE_START)
Button 2 (CENTER)
5 (LINE_END)
Long-Named Button 4 (PAGE_END)
horizontal: max
vertikal: max
horizontal: max
vertikal: min

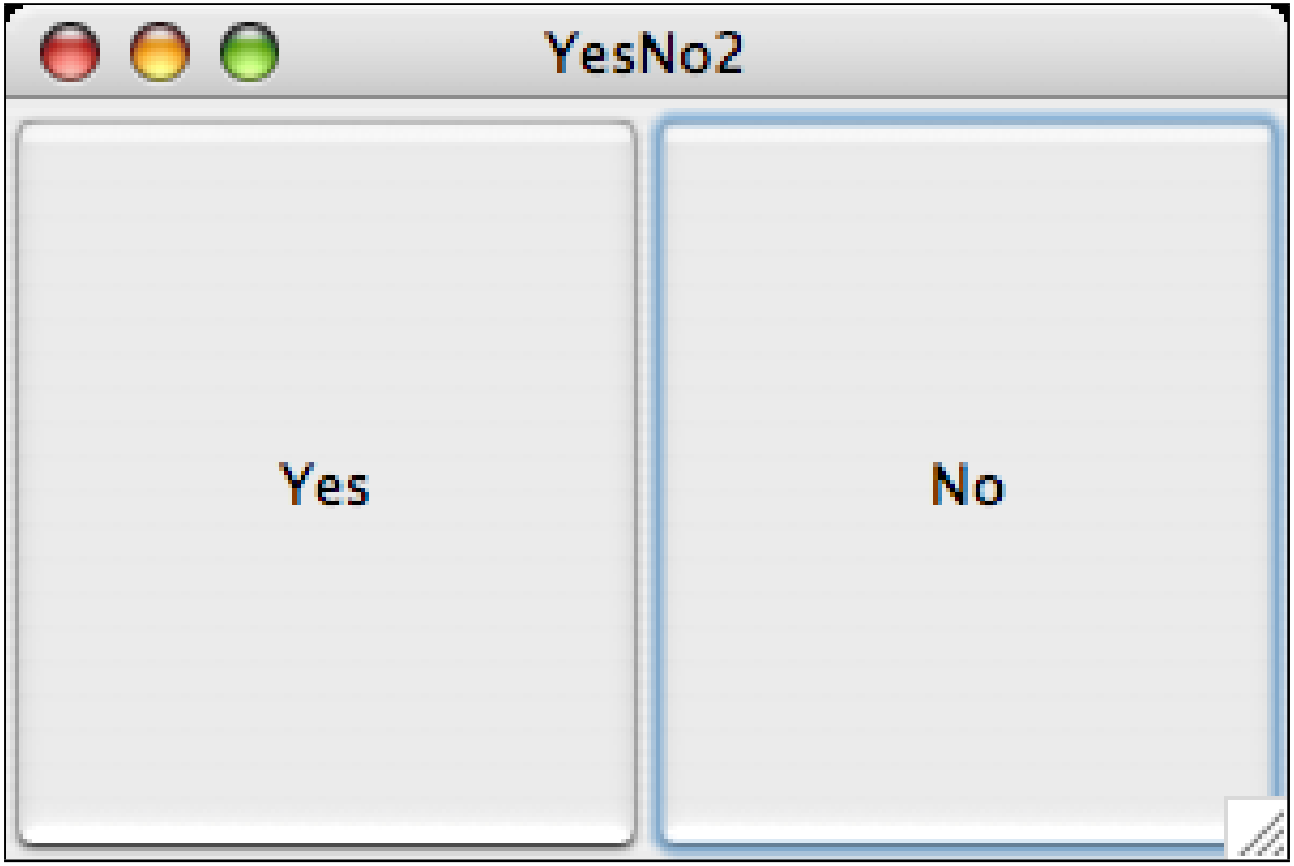
Komponenten werden in minimaler Größe hintereinander aufgereiht.

Gitter mit vorgegebener Anzahl von Zeilen und Spalten.

Layout: Beispiel

```
public static JComponent createContent() {
    JPanel contentPanel = new JPanel();
    contentPanel.setLayout(new GridLayout(1, 2)); // rows, cols
    contentPanel.add(new JButton("Yes"));
    contentPanel.add(new JButton("No"));
    return contentPanel;
}
```

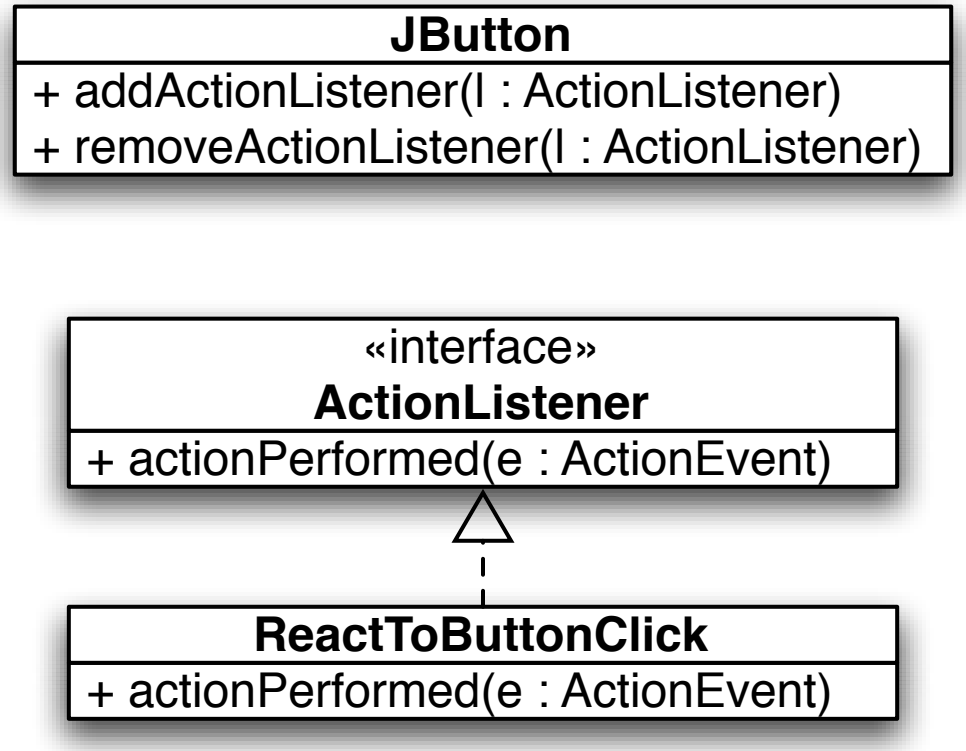
Layout-Beispiel: So sieht es aus



Events

- Idee: Beobachte und reagiere auf Veränderungen in dem GUI: Maus-Klicks, Tastendrücke.
- Welches Entwurfsmuster? Observer-Pattern
- Andere Namensgebung:
  - Listener: Observer
  - addListener(): attach()
  - removeListener(): detach()
  - fireEvent(): notify()
- Zusätzlich zum Observer-Pattern: Event-Objekte mit Informationen über das Ereignis.

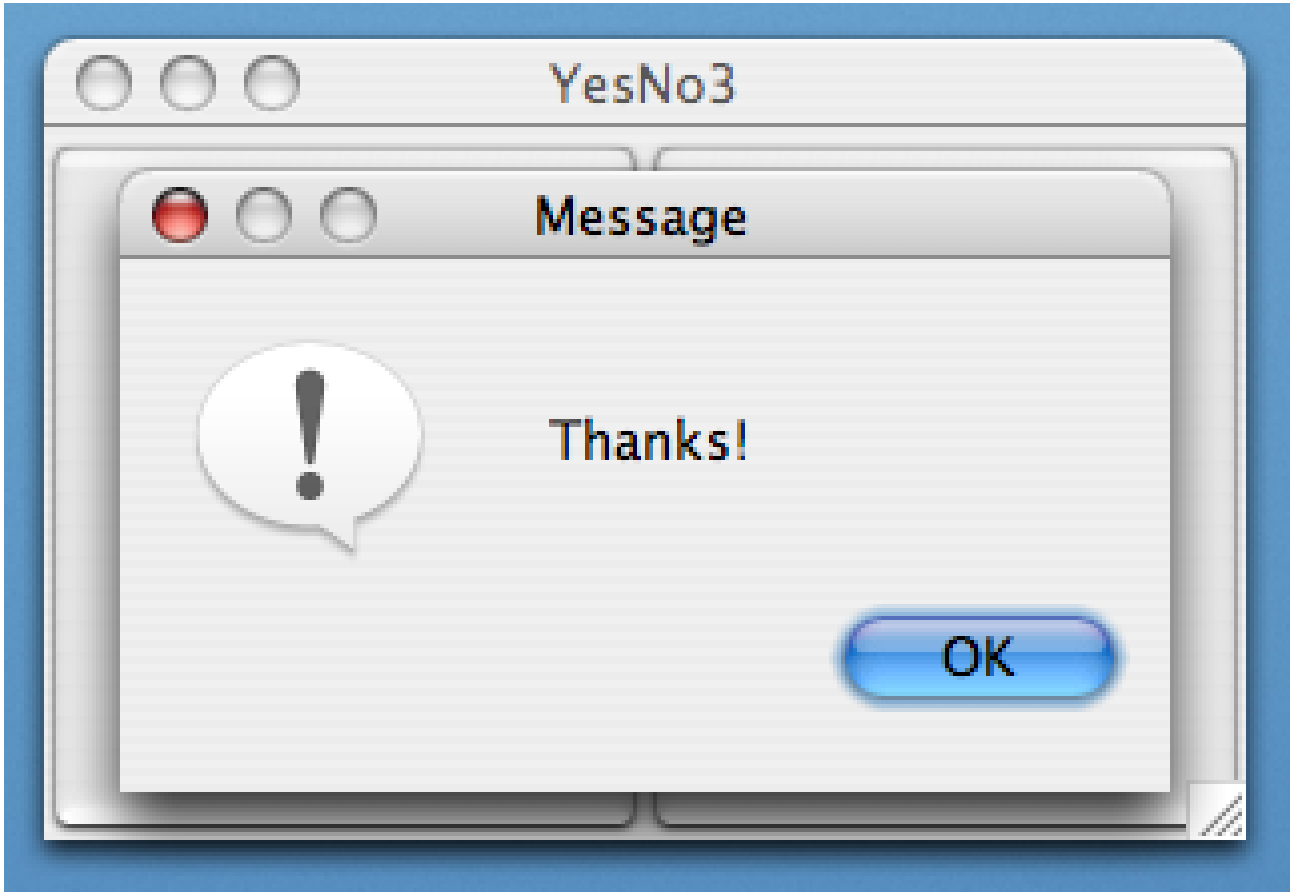
Listeners: Vereinfachtes Observer-Pattern



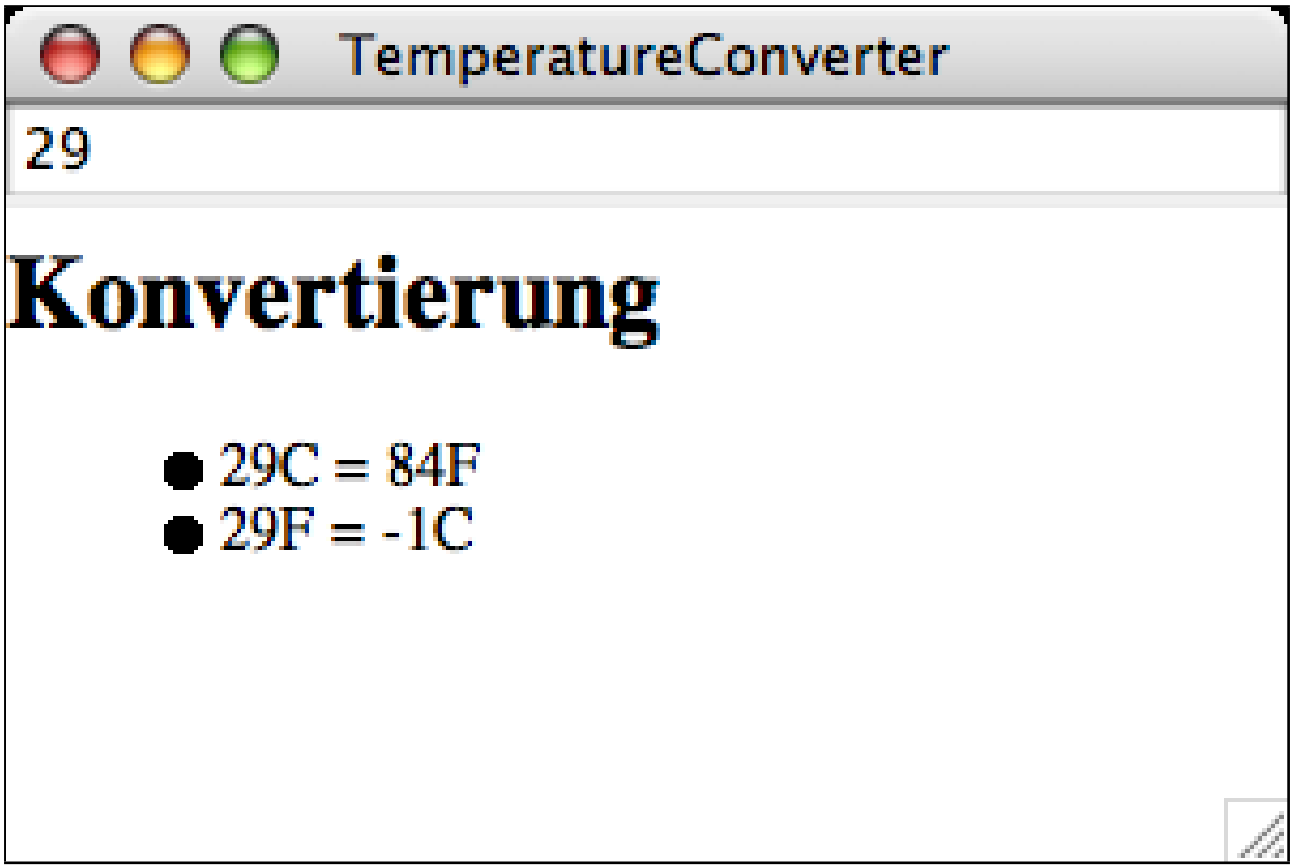
Events: Beispiel

```
public static JComponent createContent() {
    final JPanel contentPanel = new JPanel();
    contentPanel.setLayout(new GridLayout(1, 2)); // rows, cols
    JButton yesButton = new JButton("Yes");
    yesButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(contentPanel, "Thanks!");
        }
    });
    contentPanel.add(yesButton);
    contentPanel.add(new JButton("No"));
    return contentPanel;
}
```

Event-Beispiel: So sieht es aus



TemperatureConverter: So sieht es aus



TemperatureConverter-Highlight: BorderLayout

```
public Component createContent() {
    JPanel contentPanel = new JPanel();
    contentPanel.setLayout(new BorderLayout());

    contentPanel.add(createOutputPane(), BorderLayout.CENTER);
    contentPanel.add(createInputField(), BorderLayout.PAGE_START);

    return contentPanel;
}
```

TemperatureConverter-Highlight: HTML anzeigen

```
private Component createOutputPane() {
    this.outputPane = new JTextPane();
    this.outputPane.setContentType("text/html");
    this.outputPane.setEditable(false);
    return this.outputPane;
}
private void setOutput(int inputInt) {
    StringBuilder sb = new StringBuilder();
    sb.append("<h1>Konvertierung</h1>");
    sb.append("<ul>\n");
    sb.append("<li> "+inputInt+"C = "+celsiusToFahrenheit(inputInt)+"F\n")
    sb.append("<li> "+inputInt+"F = "+fahrenheitToCelsius(inputInt)+"C\n")
    sb.append("</ul>\n");
    this.outputPane.setText(sb.toString());
}
```

Zum Weiterbilden

- Java Tutorial, Swing Trail: http://java.sun.com/docs/books/tutorial/uiswing/
  Das Java-Tutorial von Sun ist generell gut gemacht und ziemlich umfangreich.

- Matisse-GUI-Builder: http://form.netbeans.org/
  Standardmäßig bei NetBeans mit dabei.

Für Fortgeschrittene

- Databinding: http://www.clientjava.com/blog/2005/12/26/1135631690 8.html
  Vereinfacht die Verbindung zwischen Modellschicht und Präsentationsschicht.

- JGoodies Forms Layout Manager: http://www.jgoodies.com/freeware/forms/
  Durchdachter LayoutManager, der viel Nerv aus dem Layouting nimmt.

- Eclipse Standard Widget Toolkit (SWT): http://www.eclipse.org/swt/
  (Nur) für Leute, die mit Swing nicht zufrieden sind: SWT hat „native“ Widgets.