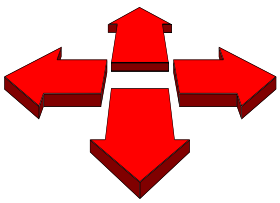
Programmierung von Graphical User Interfaces (GUIs)

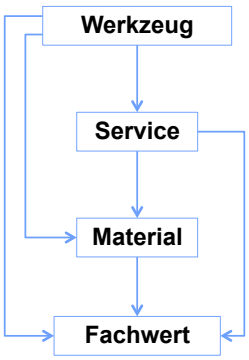
- Motivation
- GUI-Komponenten
- Reagieren auf Ereignisse
- Das Layout von GUI-Komponenten



SE2 – OOPM – Teil 2 2

Wdh.: Einfache interaktive Softwaresysteme

- Die **SE2-Entwurfsregeln** benennen vier **Elementtypen**, aus denen sich ein interaktives System zusammensetzt:
  - **Materialien** realisieren veränderliche, anwendungsfachliche Gegenstände.
  - **Fachwerte** sind anwendungsfachliche Werte; sie sind unveränderlich.
  - **Werkzeuge** bieten eine grafische Benutzungsschnittstelle und ermöglichen das interaktive Bearbeiten von Materialien.
  - **Services** bieten materialübergreifend fachliche Dienstleistungen an, die systemweit zur Verfügung stehen sollen.



Die Pfeile zeigen die **erlaubten Benutzt-Beziehungen** zwischen den Elementtypen. Jeder Elementtyp kann außerdem Elemente vom eigenen Typ benutzen (hier nicht dargestellt).

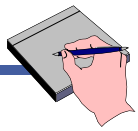
Wdh.: Werkzeugkonstruktion: Erste Schritte

- Relevante Entwurfsregeln bis zu diesem Punkt:
  - Die Werkzeug-Klasse erhält ihr **Material** als Konstruktorparameter, über Setter (wenn das Material austauschbar sein soll) oder holt es sich über Services.
  - Der Werkzeug-Klasse werden benötigte **Services als Konstruktor-parameter** übergeben.
  - Die Werkzeug-Klasse **erzeugt** ein Exemplar ihrer **UI-Klasse** im eigenen Konstruktor.
  - Die UI-Klasse eines Werkzeugs hat die Aufgaben, die **GUI-Komponenten** zu **erzeugen**, zu layouten und zu **verwalten**.
- Bevor wir diese Unterteilung weiter betrachten können, benötigen wir einiges an Grundlagenwissen über **Entwurfsmuster** und **grafische Benutzungsschnittstellen.**

Motivation – Nichts ist beständiger als der Wandel

- Technologiezyklen unterscheiden sich von fachlichen Zyklen.
- Wir müssen die fachlichen Kerne unserer Anwendungen so konstruieren, dass
  - die unausweichlichen Entwicklungen in der (GUI-)Technologie beherrschbar bleiben, und
  - fachlich motivierte Änderungen/Erweiterungen schnell, kostengünstig, und mit hoher Qualität realisiert werden können.
- Forderung deshalb: Trenne **Präsentation/Handhabung** und **Funktionalität**.
  - Es sollte aus formaler Sicht irrelevant sein, ob eine fachliche Operation über eine grafische oder eine textuelle Benutzungsschnittstelle angestoßen wird.
- Die Modell-Elemente unserer Entwurfsregeln folgen dieser Forderung:
  - **Werkzeuge** sind zuständig für die **Präsentation/Handhabung**.
  - **Services**, **Materialien** und **Fachwerte** bilden die **Funktionalität** ab.

GUI-Toolkits zur Werkzeugkonstruktion



- Betriebssysteme bieten seit einigen Jahren Unterstützung für grafische Oberflächen (z.B. Windows API, Macintosh Toolbox, Motif, ...).
- Diese APIs sind in der Regel sehr plattformabhängig und nicht objektorientiert.
- Um diese Systeme leichter zu handhaben, gibt es sogenannte **GUI-Toolkits** für Java, z.B. das **AWT** (Abstract Windowing Toolkit) und **Swing**.
- Diese
  - vereinfachen den Umgang,
  - erleichtern die Portierung,
  - stellen objektorientierte Schnittstellen zur Verfügung.
- Swing und AWT sind **objektorientierte** Toolkits zur Anbindung grafischer Benutzungsschnittstellen mit Java.
- Mit ihnen ist es möglich, den Quelltext zur Werkzeugkonstruktion (Handhabung/Präsentation) sauber von der Funktionalität zu trennen.

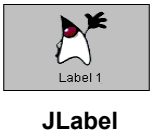
Swing baut auf dem AWT auf:
- etliche Komponenten wurden hinzugefügt;
- einige AWT-Komponenten wurden ersetzt;
- einige AWT-Komponenten werden in Swing weiterhin benutzt.

Unsere erste Swing-Applikation

- **Look:**
  - Ein Knopf mit dem Text:
  - Ein Ausgabefeld

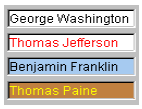
- **Feel:**
  - Wenn der Knopf gedrückt wird, dann wird ein Zähler hochgezählt.
- **Die Fragen des Tages:**
  - Welche GUI-Komponenten (der GUI-Bibliothek **Swing**) stehen uns zur Konstruktion einer graphischen Benutzungsoberfläche zur Verfügung?
  - Wie können wir diese Komponenten mit unseren fachlichen Klassen verbinden?
  - Welche Möglichkeiten gibt es für das Layout einer graphischen Benutzungsschnittstelle?

Swing-Komponenten: eine Auswahl -1-

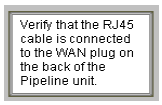


JLabel

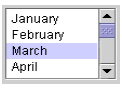
JButton



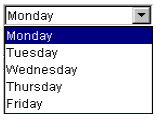
JTextField



JTextArea

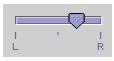


JList



JComboBox

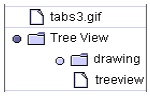
JScrollPane


JSlider

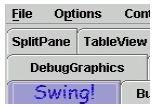
JToolTip


JToolBar


JProgressBar

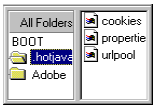


JTree

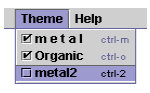


JTabbedPane

JTable

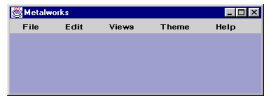


JSplitPane

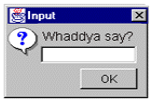


JMenu...

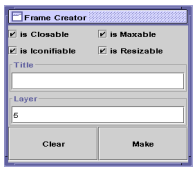
Swing-Komponenten: eine Auswahl -2-



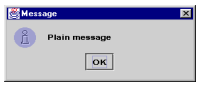
JFrame



JDialog



JInternalFrame

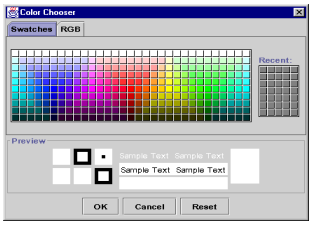


JOptionPane .showMessageDialog

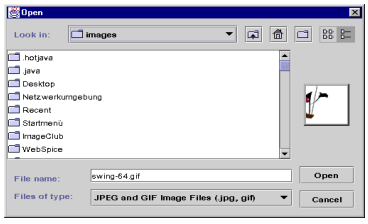
JOptionPane .showWarningDialog

JOptionPane .showInputDialog

JOptionPane .showConfirmDialog



JColorChooser



JFileChooser

SE2 – OOPM – Teil 2 9

Werkzeugkonstruktion: Nächste Schritte

- Wir zerlegen Werkzeuge immer in eine **Werkzeug-Klasse** und eine **UI-Klasse**.
- Die Werkzeug-Klasse **vermittelt** zwischen der grafischen Schnittstelle der UI-Klasse und den fachlichen Klassen.
  - Die Werkzeug-Klasse **erzeugt** ein Exemplar ihrer **UI-Klasse** im eigenen Konstruktor.
- Die UI-Klasse eines Werkzeugs hat die Aufgaben, die **GUI-Komponenten** zu **erzeugen**, zu layouten und zu **verwalten**.
  - Eine UI-Klasse **erbt nicht von UI-Framework-Klassen** wie `JFrame` oder `JPanel`, um die eigene Schnittstelle schmal zu halten. Sie definiert als oberste UI-Komponente üblicherweise einen `JFrame`.
  - Eine UI-Klasse stellt die für ihre Werkzeug-Klasse relevanten **UI-Elemente über Getter** an ihrer Schnittstelle zur Verfügung.
  - Eine UI-Klasse hat keine Abhängigkeiten zu anderen Elementtypen und verwendet nur Importe aus dem UI-Framework.
  - Eine UI-Klasse sollte als **paketinterne** Klasse deklariert werden.

SE2 – OOPM – Teil 2 10

Wdh.: Werkzeugkonstruktion: Erste Schritte

- Relevante Entwurfsregeln bis zu diesem Punkt:
  - Die Werkzeug-Klasse erhält ihr **Material** als Konstruktorparameter, über Setter (wenn das Material austauschbar sein soll) oder holt es sich über Services.
  - Der Werkzeug-Klasse werden benötigte **Services als Konstruktor-parameter** übergeben.
  - Die Werkzeug-Klasse **erzeugt** ein Exemplar ihrer **UI-Klasse** im eigenen Konstruktor.
  - Die UI-Klasse eines Werkzeugs hat die Aufgaben, die **GUI-Komponenten** zu **erzeugen**, zu layouten und zu **verwalten**.

Kontrollfluss in Anwendungen – Traditionelles Prinzip: Eingabe, Verarbeitung, Ausgabe

```
OPERATO 005                    HAUPTMENUE

   1  Umsatzverarbeitung
   2  Beratungsunterstuetzung/Abfragen
   3  Kontoauszug
   4  Bestandspflege
   5  3270
   6  Daten- und Sachgebietserfassung
   7  SB-Verwaltung
   8  Abstimmung Arbeitsplatz
   9  Systemdaten

   •  Auswahl

ENTER F10=EXPERTE F13=ABMELDUNG
```

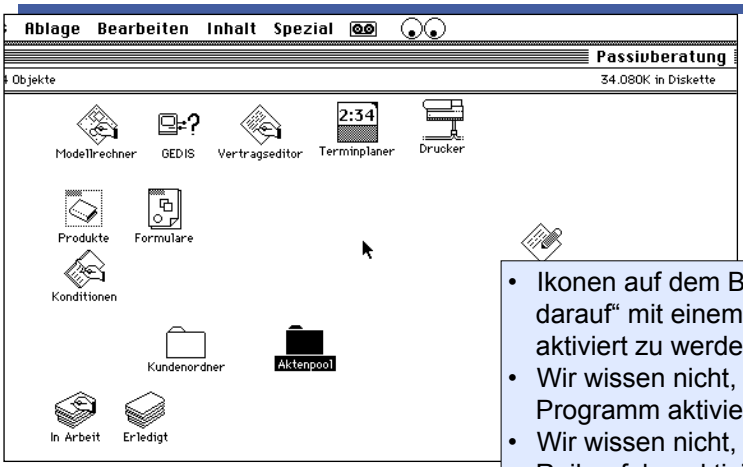
Eingabe, Verarbeitung, Ausgabe – Merkmale

- **Prinzipielle Struktur des Programmtextes:**

```
While not ende do
     input, process, output.
process:
     if lastInput = X
      then doX
      else if lastInput = Y
            then doY
             else error
```

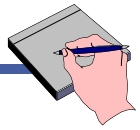
- **Vorteile**
  - Vollständige Kontrolle des Programms ist beim Anwendungsentwickler.
  - Angemessen für 24x80 Applikationen
- **Nachteile**
  - Keine Anleitung für die Trennung von fachlichem und GUI-spezifischem Code.
  - Austausch der Oberfläche (des GUIs) tendenziell schwierig.

Welche Art von Systemen wollen wir bauen (können)?



- Ikonen auf dem Bildschirm „warten darauf“ mit einem Doppelklick aktiviert zu werden. Sie sind **reaktiv**.
- Wir wissen nicht, wann welches Programm aktiviert wird.
- Wir wissen nicht, in welcher Reihenfolge aktivierte Programme vom Benutzer verwendet werden.

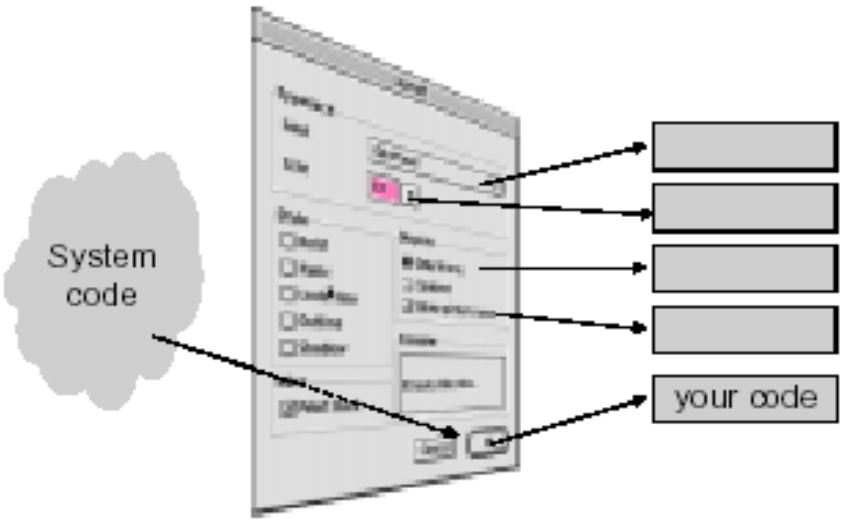
Merkmale reaktiver Software



- **Grundsätzlich:**
  - Steuerung des Kontrollflusses liegt außerhalb des Quelltextes des Anwendungsentwicklers.
- **Vorteile:**
  - Kenntnisse über die Spezifika des **Eventing** sind nicht notwendig.
  - Trennung von GUI- und Applikationscode wird erleichtert. Daraus folgen bessere Möglichkeiten der Änderbarkeit.
- **Nachteile:**
  - (Aufwendige) Einarbeitung in/Verständnis für die zugrundeliegenden GUI-Bibliotheken notwendig.

SE2 – OOPM – Teil 2 15

Reaktive Programmierung – die grundlegende Idee



© Michael Kölling, Monash University, Australia

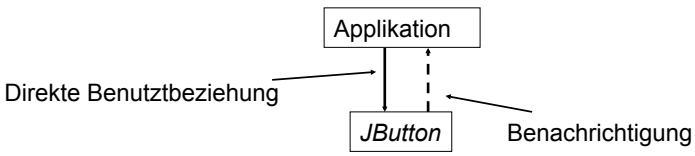
SE2 – OOPM – Teil 2 16

Ereignisverarbeitung mit Ereignissen (Events)

- Jede Mausbewegung, jeder Mausklick und jeder Tastendruck wird vom Systemcode einer GUI registriert.
- Für jede dieser Aktionen wird ein **Ereignis/Event** (vom engl. event) erzeugt.
- Events können sehr elementare Aktionen sein (Mausbewegung) oder sich aus mehreren Aktionen zusammensetzen (ein „Mausklick“ besteht beispielsweise aus den Aktionen „Mausknopf gedrückt“ und „Mausknopf wieder losgelassen“).
- Für viele Komponenten gibt es **High-Level-Events**, die die typischen Aktionen auf einer Komponenten modellieren (Bsp.: „button pressed“ auf einem Button).
- Ein solches Event wird an alle Teile des Anwendungssystems verschickt, die sich für diese Aktion bei einer GUI-Komponente **angemeldet** haben.

Anbindung der Applikation an die Oberfläche – das Prinzip

- Der Anwender löst an der Oberfläche **Ereignisse** aus.
- Die Applikation kann auf solche Ereignisse reagieren, indem sie **Listener** implementiert.
- Durch diesen Benachrichtigungsmechanismus kann die Oberflächenkomponente von der Applikation verwendet und die Applikation von der Komponente benachrichtigt werden, ohne dass
  - ➡ eine direkte zyklische Benutzt-Beziehung entsteht und
  - ➡ die Komponente alle ihre Listener explizit kennen muss.

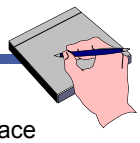


Das Konzept der Listener

- Damit Anwendungscode vom Systemcode aufgerufen werden kann, muss der Anwendungscode eine Schnittstelle haben, die dem System bekannt ist.
- Java stellt zu diesem Zweck die **Listener-Interfaces** zur Verfügung.
- Beispiel `ActionListener`:

```
public interface ActionListener
{
     void actionPerformed(ActionEvent event);
}
```

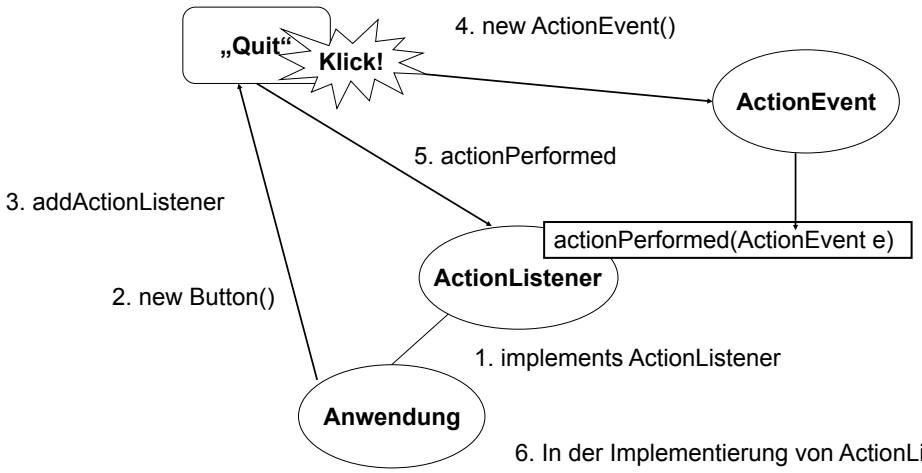
Das Konzept der Listener (II)



- Im Anwendungscode wird eine Klasse geschrieben, die dieses Interface implementiert. In der Implementierung von `actionPerformed` steht dann der Quelltext, der auf das Event reagiert.
- Damit dieser Code wirklich aufgerufen werden kann, meldet der Anwendungscode die implementierende Klasse bei der GUI-Komponente als Listener an.
- Wenn ein Ereignis eintritt (etwa ein Buttonklick), erzeugt die Komponente (der Button) ein Event-Objekt und übergibt dieses als Parameter nacheinander **allen Listenern**, die sich bei der Komponente angemeldet haben, durch Aufruf ihrer Operation `actionPerformed`.
- Das Event-Objekt enthält dann Informationen über das Ereignis (auslösende Komponente etc.).
- Beispiel „Knopf“ (Achtung: Kein vollständiger Java-Code):

```
public class Knopf implements ActionListener {
public void actionPerformed(ActionEvent e)
     { _zaehler++;  zeigeZaehler(); }
}
```

Das Konzept der Listener im schematischen Ablauf



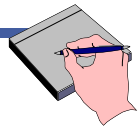
Java Spezial: Listener mit anonymen inneren Klassen

- Ein Sprachmechanismus von Java wurde speziell für eine vereinfachte Implementierung von Listener-Interfaces entworfen: die **anonymen inneren Klassen**.
- Mit diesem Mechanismus kann an einer Stelle, an der ein Exemplar einer ein Interface implementierenden Klasse übergeben werden soll, direkt ein **spezieller Ausdruck** stehen (hier rot hervorgehoben):

```
myButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            // geeignete Reaktion; Zugriff auf private Felder
            // des „umgebenden“ Exemplars möglich!
        }
    });
```

- Dieser Ausdruck **erzeugt** nicht nur an der Aufrufstelle **ein Exemplar** einer Klasse, sondern **definiert auch** gleich **die erzeugende Klasse selbst** – namenlos, wie hier zu sehen, deshalb der Begriff anonyme innere Klasse.

Anonyme innere Klassen



- Wir erinnern uns: Java erlaubt die Schachtelung von Klassen durch **geschachtelte Klassen**. Neben den statischen geschachtelten Klassen gibt es auch drei Arten von geschachtelten Klassen, die **innere Klassen** genannt werden.
- Ein Exemplar einer inneren Klasse benötigt immer ein Exemplar der Klasse, in die die innere Klasse hineingeschachtelt ist; das umgebende Objekt kann man als das **Wirtsobjekt** bezeichnen, das innere Objekt als **Parasit-Objekt**.
- In Java ist das Verhältnis zwischen geschachtelten und umgebenden Klassen **sehr eng**: Zwei Exemplare beider Klassen können wechselseitig auf **alle Exemplarvariablen** (auch auf private) des jeweils anderen **zugreifen**.
- Anonyme innere Klassen können sogar auf die lokalen Variablen der Methode zugreifen, in der sie definiert wurden; allerdings nur, wenn diese als `final` deklariert sind.

SE2 – OOPM – Teil 2 23

Komponenten und Listener: Beispiele

- Wichtige Listener:
  - `JRadioButton`: `ActionListener`, `ItemListener`
  - `JList`: `ActionListener`, `ListSelectionListener`
  - `JComboBox`: `ActionListener`, `ItemListener`
  - `JTextField`: `ActionListener`

SE2 – OOPM – Teil 2 24

Zur Vervollständigung: Events, Listener

- Es gibt in AWT und Swing verschiedene Typen von Ereignissen. Zu jedem Typ existiert eine **Event-Klasse**.
- Die Event-Objekte **tragen** alle nötigen **Informationen** über das aktuelle Ereignis mit sich. Zum Beispiel verfügen alle Java-Events über die Operationen `getSource` (Event-Quellkomponente) und `getID` (Event-Typ als Konstante). Viele Events verfügen auch über `consume`, um das Event zu „verbrauchen".
- Event-Objekte werden von angemeldeten **Listenern** verarbeitet. Ein **Listener-Interface** definiert alle notwendigen Antwortmethoden, mit denen auf bestimmte Ereignisse reagiert werden kann.

Events und Listener: Weitere Beispiele

Event-Typ und wichtige Methoden	Listener	Methoden
FocusEvent isTemporary()	**FocusListener**	**focusGained()** **focusLost()**
ItemEvent getItem() getItemSelectable() getStateChange()	**ItemListener**	**itemStateChanged()**
TextEvent	**TextListener**	**textValueChanged()**
ComponentEvent getComponent()	**ComponentListener**	**componentHidden()** **componentMoved()** **componentResized()** **componentShown()**

Es gibt noch mehr Events...

Event / EventListener zusammengefasst

Der Mechanismus zur Behandlung von Oberflächen-Ereignissen in Java ist der sog. **Event/EventListener-Mechanismus**.

Dieser ist allgemein so aufgebaut:

1. Jede GUI-Komponente implementiert für eine bestimmte Art von Ereignissen eine **add*<EventListener>*()** – Methode.
2. Über diese Methode kann an einer GUI-Komponente ein Objekt „angemeldet“ werden, welches das Interface ***<EventListener >*** implementiert.
3. Wird ein entsprechendes Ereignis durch den Benutzer ausgelöst, so werden alle angemeldeten Listener-Objekte benachrichtigt.
4. Informationen über den Ereignistyp sowie weitere evtl. notwendige Informationen werden über ein Event-Objekt übermittelt.

Werkzeugkonstruktion: Ereignisverarbeitung

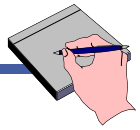
- Wir haben nun das technische Rüstzeug, um die Entwurfsregeln zur Ereignisverarbeitung nachvollziehen zu können:
  - Eine UI-Klasse **stellt** die für ihre Werkzeug-Klasse relevanten **UI-Elemente** über Getter **in ihrer Schnittstelle zur Verfügung**.
  - Die Werkzeug-Klasse **erzeugt** für diese UI-Widgets **Listener**, die passende Aktionen ausführen, und **registriert** diese an den Widgets.

Die Werkzeug-Klasse reagiert auf Ereignisse!

  - Die Werkzeug-Klasse gibt die anzuzeigenden Materialien oder Fachwerte in die UI, in folgender Weise:
    - Die Werkzeug-Klasse **holt** sich von der UI-Klasse ein **Widget** und **setzt** bei diesem die anzuzeigenden **Informationen** des Materials oder des Fachwerts.
    - Sofern nötig, wird das darzustellende Element über einen **Formatierer** für die jeweilige Darstellung angepasst.

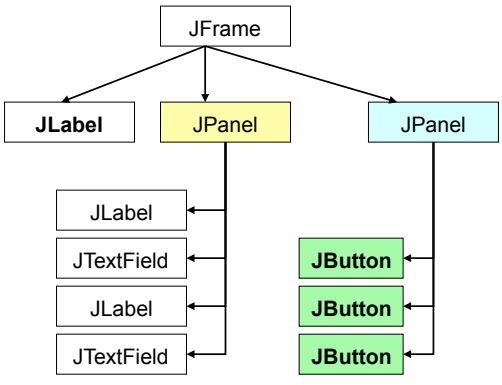
Die Werkzeug-Klasse bestimmt, wie ein Material oder ein Fachwert dargestellt wird!

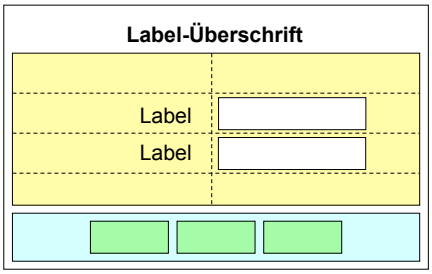
Auf dem Weg zu strukturierten Swing-Oberflächen: Komponenten erzeugen und schachteln



- Die einzelnen Komponenten einer Swing-Oberfläche werden hierarchisch angeordnet.
- Komponenten können unterschieden werden in **Containerkomponenten** und **atomare Komponenten**.
- Containerkomponenten können beliebige Komponenten (auch wieder Container) enthalten.
- Container bieten eine Schnittstelle, mit der Komponenten eingetragen werden können (Operation `add`).

Hierarchischer Aufbau einer Swing-Oberfläche

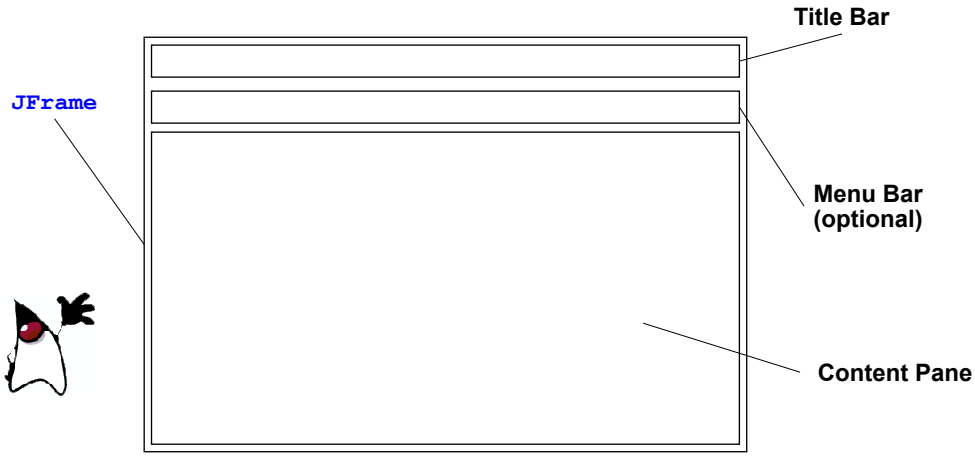




Top-Level Container

- An der Spitze der hierarchischen Struktur einer Oberfläche stehen die **Top-Level Container**. Sie korrespondieren mit den Fenstern, die vom jeweiligen Betriebssystem zur Verfügung gestellt werden.
- Top-Level Container in Swing sind bespielsweise `JFrame`, `JDialog`, `JOptionPane`, `JApplet`.
- Ein `JFrame` enthält unter anderem einen so genannten **Content Pane**. Dies ist der Container, in den die Hauptkomponenten der Oberfläche eingetragen werden.

Struktur eines `JFrame`

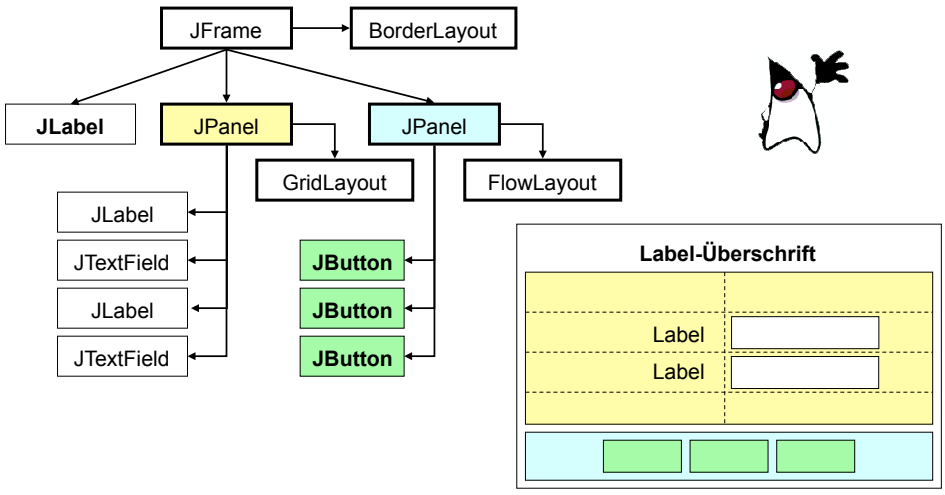


Layout festlegen

- In Java ordnen **Layout-Manager** die Komponenten in einem Container an. Oberflächen werden also nicht pixelgenau erstellt, sondern immer relativ zueinander.
- Das erleichtert beispielsweise:
  - das **Vergrößern** und **Verkleinern** von Fenstern inklusive Inhalt,
  - die **Darstellung** gleicher, aber unterschiedlich großer Widgets **auf unterschiedlichen Plattformen** (ein typischer Windows-Button kann völlig anders aussehen als ein Mac-Button).
- Jeder Container besitzt als Default einen Layout-Manager.

SE2 – OOPM – Teil 2 33

Container mit Layout



SE2 – OOPM – Teil 2 34

Layout-Manager

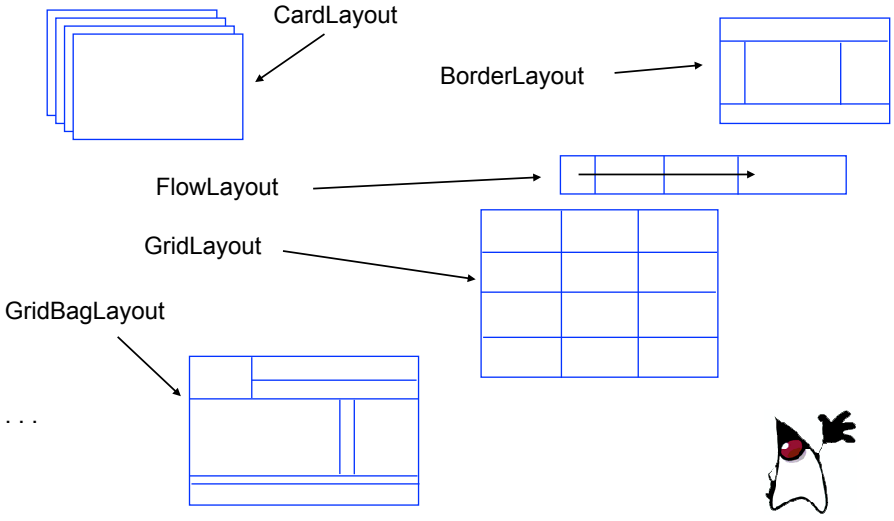
- Jeder Container hat einen Layout-Manager, der gesetzt und abgefragt werden kann:

```
void setLayout( LayoutManager );
LayoutManager getLayout();
```

- Das Standard-Layout für den Content-Pane eines `JFrame` ist das `BorderLayout`.
- Für Fortgeschrittene: Man kann eigene Layout-Klassen schreiben und benutzen.
- Es gibt in Java bereits zahlreiche Layout-Manager, von denen wir nur die wichtigsten betrachten....

SE2 – OOPM – Teil 2 35

Verschiedene Layout-Manager



SE2 – OOPM – Teil 2 36

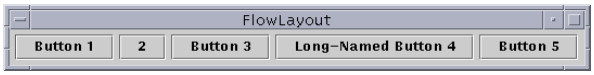
Layouts im Beispiel: `FlowLayout`

- Das **`FlowLayout`** ordnet Komponenten in einer Reihe an und bricht die Komponenten notfalls in Zeilen um.
- Die Abstände zwischen den Komponenten und die Ausrichtung zum umgebenden Frame können angegeben werden.
- **`FlowLayout`** befindet sich im Paket **`java.awt`**.

```
Container contentPane = getContentPane();
FlowLayout fl = new FlowLayout();
contentPane.setLayout( fl );

contentPane.add(new JButton("Button 1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("Button 3"));
contentPane.add(new JButton("Long-Named Button 4"));
contentPane.add(new JButton("Button 5"));

fl.setAlignment( FlowLayout.CENTER ); // LEFT, RIGHT, ...
fl.setHgap( 20 ); // Horizontales spacing
fl.setVgap( 20 ); // Vertikales spacing
```



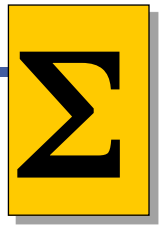
Werkzeugkonstruktion: Layout

- Relevante Entwurfsregeln zum Thema Layout:
  - Die UI-Klasse eines Werkzeugs hat die Aufgaben, die **GUI-Komponenten** zu erzeugen, zu **layouten** und zu verwalten.
- Mit anderen Worten: Die **UI-Klasse** ist ein „Sammelbehälter“ für die UI-Widgets; sie ist **vollständig für das Layout** der UI-Komponenten **zuständig**.

Zusammenfassung GUI-Programmierung

- Wir haben grundlegende Techniken zur Konstruktion **reaktiver Programme** mit einer GUI-Bibliothek kennengelernt.
- Am Beispiel der GUI-Toolkits **AWT** und **Swing** haben wir gesehen, dass
  - es sehr unterschiedliche **GUI-Komponenten** geben kann;
  - **Listener** eine Möglichkeit darstellen, um Anwendungscode durch GUI-Code über **Ereignisse** informieren zu lassen;
  - das **Layout** einer grafischen Benutzungsoberfläche sehr flexibel mit Layout-Managern gestaltet werden kann.
- Mit diesen Kenntnissen können wir nun unsere ersten **Desktop-Anwendungen** konstruieren.

Zusammenfassung SE2-Entwurfsregeln



- Die **SE2-Entwurfsregeln** für interaktive Anwendungen (insbes. Rich-Clients) geben konstruktive Hinweise für die Strukturierung von Softwaresystemen mit einer grafischen Oberfläche.
- Sie definieren vier Elementtypen:
  - **Materialien** – veränderbare fachliche Objekte, die üblicherweise Arbeitsergebnisse modellieren;
  - **Fachwerte** – unveränderliche fachliche Abstraktionen;
  - **Services**, die systemweit fachliche Dienstleistungen anbieten und häufig Materialien verwalten;
  - **Werkzeuge** zur interaktiven Bearbeitung von Materialien.
- Der Entwurf von **Materialien**, **Fachwerten** und **Services** ist **primär fachlich anspruchsvoll**, während die Konstruktion von **Werkzeugen** vor allem **technisch anspruchsvoll** ist.